Final Project

December 12, 2022

Team: Roger Qiu, Verity Pierson, & Shailja Somani
Dataset Used: Details of house sales in King County, Washington between May 2014 and May 2015.
Project Github: https://github.com/shailja-somani-0/ADS500B-rog-ver-sha

1 Import and Clean Dataset

[63]:
```
# Import packages necessary
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import os
from IPython.display import display
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error as MSE
import xgboost as xg
from sklearn.model_selection import GridSearchCV
```

[2]:
```
# Check working directory if necessary - commented out as not necessary
#os.getcwd()
```

[3]:
```
#importing the csv file
df = pd.read_csv('house_sales.csv', header=0, sep=',')
df[:20]
```

[3]:
```
           id             date      price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000   221900.0       3.0       1.00       1180.0
1  6414100192  20141209T000000   538000.0       3.0       2.25       2570.0
2  5631500400  20150225T000000   180000.0       2.0       1.00        770.0
3  2487200875  20141209T000000   604000.0       4.0       3.00       1960.0
4  1954400510  20150218T000000   510000.0       3.0       2.00       1680.0
5  7237550310  20140512T000000  1225000.0       4.0       4.50       5420.0
6  1321400060  20140627T000000   257500.0       3.0       2.25       1715.0
```

```
7   2008000270  20150115T000000   291850.0       3.0       1.50       1060.0
8   2414600126  20150415T000000   229500.0       3.0       1.00       1780.0
9   3793500160  20150312T000000   323000.0       3.0       2.50       1890.0
10  1736800520  20150403T000000   662500.0       3.0       2.50          NaN
11  9212900260  20140527T000000   468000.0       2.0       1.00       1160.0
12   114101516  20140528T000000   310000.0       3.0       1.00          NaN
13  6054650070  20141007T000000   400000.0       3.0       1.75       1370.0
14  1175000570  20150312T000000   530000.0       5.0       2.00       1810.0
15  9297300055  20150124T000000   650000.0       4.0       3.00       2950.0
16  1875500060  20140731T000000   395000.0       3.0       2.00       1890.0
17  6865200140  20140529T000000   485000.0       4.0       1.00       1600.0
18    16000397  20141205T000000   189000.0       NaN       1.00       1200.0
19  7983200060  20150424T000000   230000.0       3.0       1.00       1250.0

    sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0     5650.0     1.0           0     0  …      7        1180              0
1     7242.0     2.0           0     0  …      7        2170            400
2    10000.0     1.0           0     0  …      6         770              0
3     5000.0     1.0           0     0  …      7        1050            910
4     8080.0     1.0           0     0  …      8        1680              0
5   101930.0     1.0           0     0  …     11        3890           1530
6     6819.0     2.0           0     0  …      7        1715              0
7     9711.0     1.0           0     0  …      7        1060              0
8     7470.0     1.0           0     0  …      7        1050            730
9     6560.0     2.0           0     0  …      7        1890              0
10    9796.0     1.0           0     0  …      8        1860           1700
11    6000.0     1.0           0     0  …      7         860            300
12   19901.0     1.5           0     0  …      7        1430              0
13    9680.0     1.0           0     0  …      7        1370              0
14    4850.0     1.5           0     0  …      7        1810              0
15    5000.0     2.0           0     3  …      9        1980            970
16   14040.0     2.0           0     0  …      7        1890              0
17    4300.0     1.5           0     0  …      7        1600              0
18    9850.0     1.0           0     0  …      7        1200              0
19    9774.0     1.0           0     0  …      7        1250              0

    yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0       1955             0    98178  47.5112 -122.257           1340
1       1951          1991    98125  47.7210 -122.319           1690
2       1933             0    98028  47.7379 -122.233           2720
3       1965             0    98136  47.5208 -122.393           1360
4       1987             0    98074  47.6168 -122.045           1800
5       2001             0    98053  47.6561 -122.005           4760
6       1995             0    98003  47.3097 -122.327           2238
7       1963             0    98198  47.4095 -122.315           1650
8       1960             0    98146  47.5123 -122.337           1780
9       2003             0    98038  47.3684 -122.031           2390
```

```
10     1965          0  98007  47.6007 -122.145           2210
11     1942          0  98115  47.6900 -122.292           1330
12     1927          0  98028  47.7558 -122.229           1780
13     1977          0  98074  47.6127 -122.045           1370
14     1900          0  98107  47.6700 -122.394           1360
15     1979          0  98126  47.5714 -122.375           2140
16     1994          0  98019  47.7277 -121.962           1890
17     1916          0  98103  47.6648 -122.343           1610
18     1921          0  98002  47.3089 -122.210           1060
19     1969          0  98003  47.3343 -122.306           1280

    sqft_lot15
0         5650
1         7639
2         8062
3         5000
4         7503
5       101930
6         6819
7         9711
8         8113
9         7570
10        8925
11        6000
12       12697
13       10208
14        4850
15        4000
16       14018
17        4300
18        5095
19        8850

[20 rows x 21 columns]
```

[4]:
```
# Checking to see if there are major outliers
df.describe()
```

[4]:
```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.161300e+04  2.161300e+04  20479.000000  20545.000000  20503.000000
mean   4.580302e+09  5.400881e+05      3.372821      2.113507   2081.073697
std    2.876566e+09  3.671272e+05      0.930711      0.768913    915.043176
min    1.000102e+06  7.500000e+04      0.000000      0.000000    290.000000
25%    2.123049e+09  3.219500e+05      3.000000      1.500000   1430.000000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1920.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000  12050.000000
```

```
           sqft_lot        floors    waterfront          view     condition  \
count  2.056900e+04  21613.000000  21613.000000  21613.000000  21613.000000
mean   1.517982e+04      1.494309      0.007542      0.234303      3.409430
std    4.148617e+04      0.539989      0.086517      0.766318      0.650743
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000
50%    7.620000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.070800e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000  21613.000000
mean       7.656873   1788.390691     291.509045   1971.005136     84.402258
std        1.175459    828.090978     442.575043     29.373411    401.679240
min        1.000000    290.000000       0.000000   1900.000000      0.000000
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000

            zipcode           lat          long  sqft_living15     sqft_lot15
count  21613.000000  21613.000000  21613.000000   21613.000000   21613.000000
mean   98077.939805     47.560053   -122.213896    1986.552492   12768.455652
std       53.505026      0.138564      0.140828     685.391304   27304.179631
min    98001.000000     47.155900   -122.519000     399.000000     651.000000
25%    98033.000000     47.471000   -122.328000    1490.000000    5100.000000
50%    98065.000000     47.571800   -122.230000    1840.000000    7620.000000
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000
```

1.1 Fill in Null Values

[5]:
```python
# looking for the number of nulls in the data
df.isnull().sum()
```

[5]:
```
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
```

[6]:
```
# Use (sqft_above + sqft_basement) to fill in nulls for sqft_living
df['sqft_living'].fillna((df['sqft_above'] + df['sqft_basement']), inplace =␣
↪True)
```

[7]:
```
# Use sqft_lot15 to fill in nulls for sqft_lot
df['sqft_lot'].fillna(df['sqft_lot15'], inplace = True)
```

[8]:
```
# Use the median of bedrooms for various buckets of sqft_living to fill in␣
↪bedroom nulls
# Set boundaries for bins & assign each bin a label to turn sqft_living into a␣
↪categorical variable
bins = [0,1000,1500,2000,2500,3000,15000]
labels=[1,2,3,4,5,6]
# Column "a" becomes that categorical variable
df['a'] = pd.cut(df['sqft_living'], bins=bins, labels=labels,␣
↪include_lowest=True)
# Loop through all values of column a, get the median bedrooms for each, then␣
↪fill in
df['bedrooms'] = df.groupby('a')['bedrooms'].apply(lambda x: x.fillna(x.
↪median()))
```

[9]:
```
# Use the median bathrooms for each value of bedrooms to fill in bathroom nulls
# Set boundaries for bins & assign each bin a label to turn sqft_living into a␣
↪categorical variable
bins = [0,1,2,3,4,5,6,7,8,9,10,35]
labels=[1,2,3,4,5,6,7,8,9,10,11]
# Column "a" becomes that categorical variable
df['a'] = pd.cut(df['bedrooms'], bins=bins, labels=labels, include_lowest=True)
# Loop through all values of column a, get the median bathrooms for each, then␣
↪fill in
df['bathrooms'] = df.groupby('a')['bathrooms'].apply(lambda x: x.fillna(x.
↪median()))
```

[10]:
```
# Checking to make sure filling in the nulls worked
df.isnull().sum()
```

[10]:
```
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
a                0
dtype: int64
```

[11]:
```
# Drop categorical column used to make bins to fill nulls
df = df.drop(columns=["a"])
```

1.2 Clean Up Date Field

[12]:
```
# Get first 8 characters of date field - remove T000000
df['date'] = pd.to_numeric(df['date'].str[:8])
```

[13]:
```
# Checking the object types
df.dtypes
```

[13]:
```
id                 int64
date               int64
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
```

[14]:
```python
# Checking to see if the update made a change in the rows
df.describe()
```

[14]:
```
                 id          date         price      bedrooms     bathrooms  \
count  2.161300e+04  2.161300e+04  2.161300e+04  21613.000000  21613.000000
mean   4.580302e+09  2.014390e+07  5.400881e+05      3.373803      2.113335
std    2.876566e+09  4.436582e+03  3.671272e+05      0.916691      0.758138
min    1.000102e+06  2.014050e+07  7.500000e+04      0.000000      0.000000
25%    2.123049e+09  2.014072e+07  3.219500e+05      3.000000      1.750000
50%    3.904930e+09  2.014102e+07  4.500000e+05      3.000000      2.250000
75%    7.308900e+09  2.015022e+07  6.450000e+05      4.000000      2.500000
max    9.900000e+09  2.015053e+07  7.700000e+06     33.000000      8.000000

        sqft_living      sqft_lot        floors    waterfront          view  \
count  21613.000000  2.161300e+04  21613.000000  21613.000000  21613.000000
mean    2079.899736  1.499496e+04      1.494309      0.007542      0.234303
std      918.440897  4.075517e+04      0.539989      0.086517      0.766318
min      290.000000  5.200000e+02      1.000000      0.000000      0.000000
25%     1427.000000  5.040000e+03      1.000000      0.000000      0.000000
50%     1910.000000  7.616000e+03      1.500000      0.000000      0.000000
75%     2550.000000  1.062500e+04      2.000000      0.000000      0.000000
max    13540.000000  1.651359e+06      3.500000      1.000000      4.000000

       …         grade    sqft_above  sqft_basement      yr_built  \
count  …  21613.000000  21613.000000   21613.000000  21613.000000
mean   …      7.656873   1788.390691     291.509045   1971.005136
std    …      1.175459    828.090978     442.575043     29.373411
min    …      1.000000    290.000000       0.000000   1900.000000
25%    …      7.000000   1190.000000       0.000000   1951.000000
50%    …      7.000000   1560.000000       0.000000   1975.000000
75%    …      8.000000   2210.000000     560.000000   1997.000000
max    …     13.000000   9410.000000    4820.000000   2015.000000
```

```
       yr_renovated       zipcode           lat          long  sqft_living15  \
count  21613.000000  21613.000000  21613.000000  21613.000000   21613.000000
mean      84.402258  98077.939805     47.560053   -122.213896    1986.552492
std      401.679240     53.505026      0.138564      0.140828     685.391304
min        0.000000  98001.000000     47.155900   -122.519000     399.000000
25%        0.000000  98033.000000     47.471000   -122.328000    1490.000000
50%        0.000000  98065.000000     47.571800   -122.230000    1840.000000
75%        0.000000  98118.000000     47.678000   -122.125000    2360.000000
max     2015.000000  98199.000000     47.777600   -121.315000    6210.000000

          sqft_lot15
count   21613.000000
mean    12768.455652
std     27304.179631
min       651.000000
25%      5100.000000
50%      7620.000000
75%     10083.000000
max    871200.000000

[8 rows x 21 columns]
```

[15]:
```
# Take a peek at the cleaned df
df.head()
```

[15]:
```
           id      date     price  bedrooms  bathrooms  sqft_living  sqft_lot  \
0  7129300520  20141013  221900.0       3.0       1.00       1180.0    5650.0
1  6414100192  20141209  538000.0       3.0       2.25       2570.0    7242.0
2  5631500400  20150225  180000.0       2.0       1.00        770.0   10000.0
3  2487200875  20141209  604000.0       4.0       3.00       1960.0    5000.0
4  1954400510  20150218  510000.0       3.0       2.00       1680.0    8080.0

   floors  waterfront  view  …  grade  sqft_above  sqft_basement  yr_built  \
0     1.0           0     0  …      7        1180              0      1955
1     2.0           0     0  …      7        2170            400      1951
2     1.0           0     0  …      6         770              0      1933
3     1.0           0     0  …      7        1050            910      1965
4     1.0           0     0  …      8        1680              0      1987

   yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15
0             0    98178  47.5112 -122.257           1340        5650
1          1991    98125  47.7210 -122.319           1690        7639
2             0    98028  47.7379 -122.233           2720        8062
3             0    98136  47.5208 -122.393           1360        5000
4             0    98074  47.6168 -122.045           1800        7503

[5 rows x 21 columns]
```

2 Data Analysis and Visualization

Identify categorical, ordinal, and numerical variables within the data
Provide measures of centrality and distribution with visualizations
Diagnose for correlations between variables and determine independent and dependent variables
Perform exploratory analysis in combination with visualization techniques to discover patterns and features of interest

[71]:
```
# reading the first 5 rows of all the fields to check their data type
display(df.head())

print("All fields seem to be numerical.\nDiscrete fields are: id, price,␣
↪bedrooms, bathrooms, floors, waterfront, view, condition, grade, yr_built,␣
↪yr_renovated, and zipcode. \nContinuous fields are: sqft_living, sqft_lot,␣
↪sqft_above, sqft_basement, lat, long, sqft living, sqft_lot. \nThere are no␣
↪fields with categorical or ordinal data types.")
```

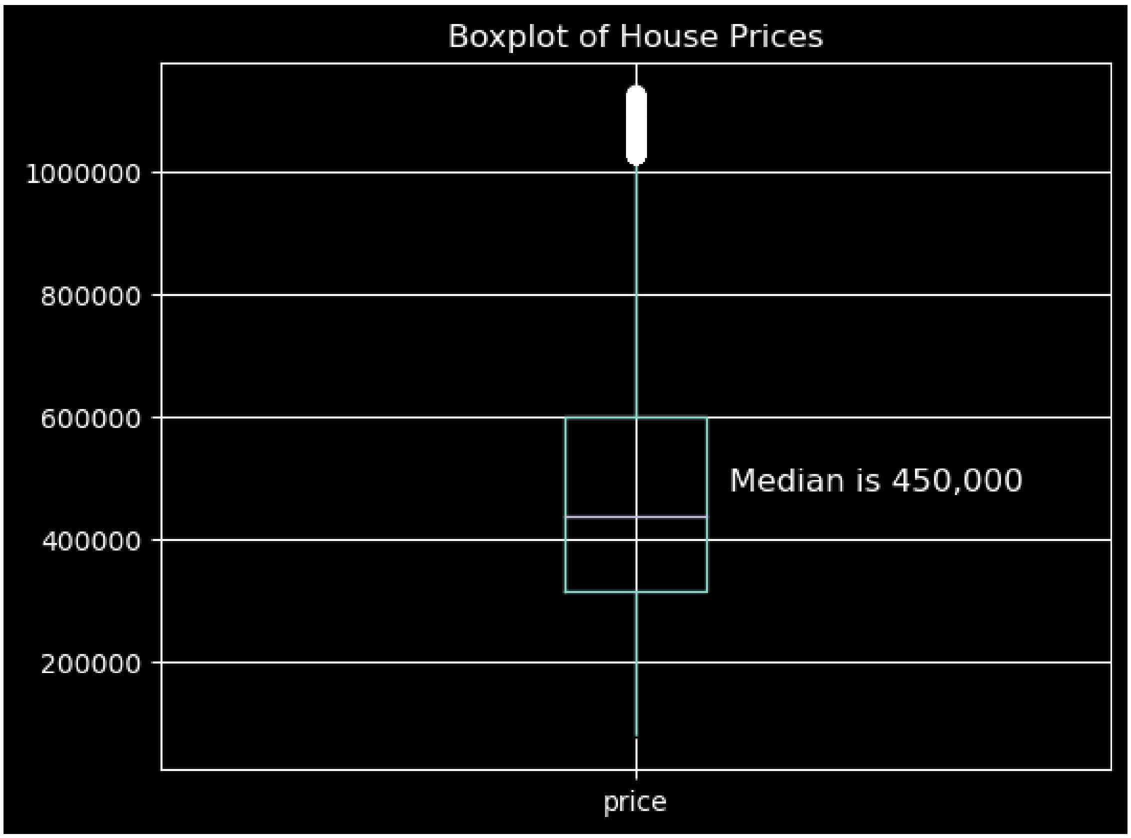
```
           id      date     price  bedrooms  bathrooms  sqft_living  sqft_lot  \
0  7129300520  20141013  221900.0       3.0       1.00       1180.0    5650.0
1  6414100192  20141209  538000.0       3.0       2.25       2570.0    7242.0
2  5631500400  20150225  180000.0       2.0       1.00        770.0   10000.0
3  2487200875  20141209  604000.0       4.0       3.00       1960.0    5000.0
4  1954400510  20150218  510000.0       3.0       2.00       1680.0    8080.0

   floors  waterfront  view  …  grade  sqft_above  sqft_basement  yr_built  \
0     1.0           0     0  …      7        1180              0      1955
1     2.0           0     0  …      7        2170            400      1951
2     1.0           0     0  …      6         770              0      1933
3     1.0           0     0  …      7        1050            910      1965
4     1.0           0     0  …      8        1680              0      1987

   yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15
0             0    98178  47.5112 -122.257           1340        5650
1          1991    98125  47.7210 -122.319           1690        7639
2             0    98028  47.7379 -122.233           2720        8062
3             0    98136  47.5208 -122.393           1360        5000
4             0    98074  47.6168 -122.045           1800        7503

[5 rows x 21 columns]
```

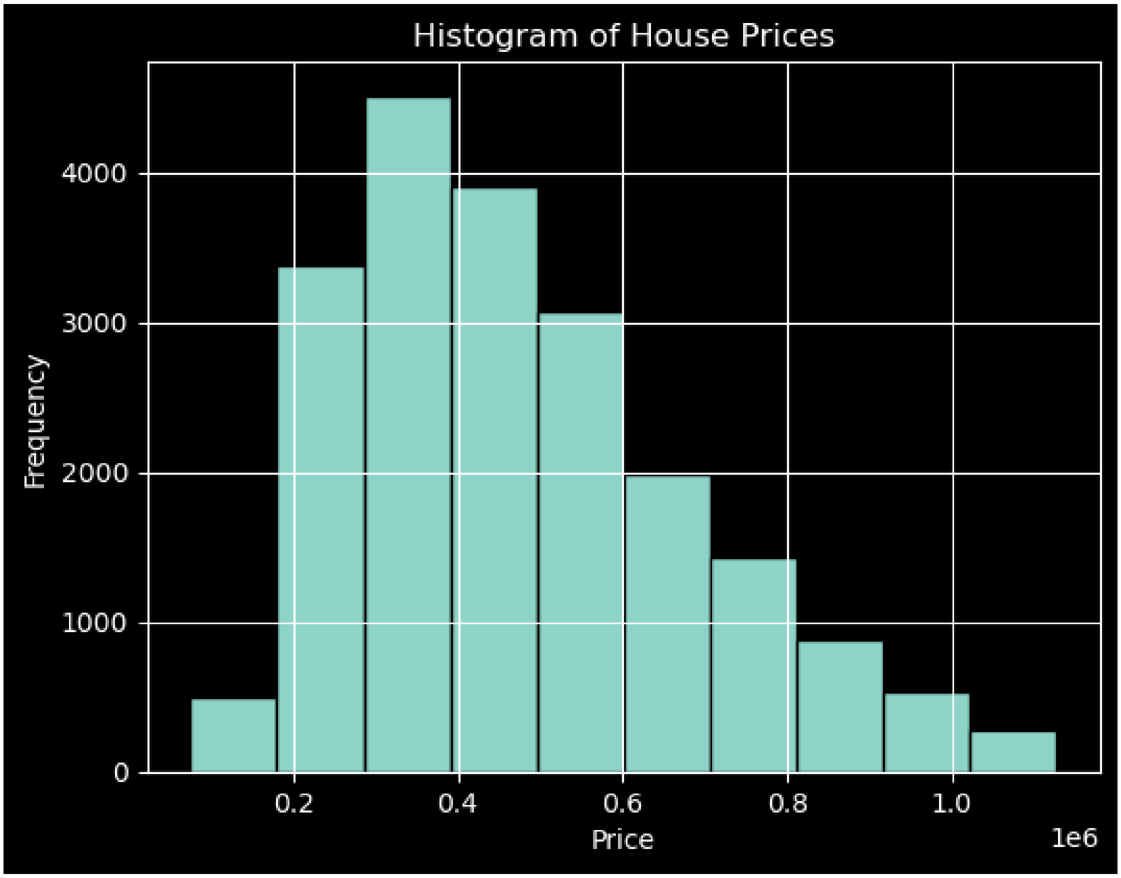
```
All fields seem to be numerical.
Discrete fields are: id, price, bedrooms, bathrooms, floors, waterfront, view,
condition, grade, yr_built, yr_renovated, and zipcode.
Continuous fields are: sqft_living, sqft_lot, sqft_above, sqft_basement, lat,
long, sqft living, sqft_lot.
There are no fields with categorical or ordinal data types.
```

[16]:

```
print("Now lets look at the centrality and distribution with boxplot and␣
  ↪histogram visualizations of a few select fields: price, sqft_living, and␣
  ↪yr_built.")
print("Let's start with house prices")

# find the median and 1st and 3rd quartiles
print("The quartiles and the median are:\n" + str((df.price.quantile([0.25,0.
  ↪5,0.75]))))

# lets remove the outliers now for our plot

# find IQR
# find third and 1st quartile of price
# then subtract to find the difference
q3, q1 = np.percentile(df.price, [75, 25])
iqr = q3 - q1
print("The IQR is " + str(iqr.round(3)))

# create the lower and upper outlier points
lower_outliers = q1 - 1.5*iqr
upper_outliers = q3 + 1.5*iqr

# create the new subset based on all rows that are greater than the lower␣
  ↪outliers and less than the upper outlier.
updated_price = df.loc[(df['price'] > lower_outliers) & (df['price'] <␣
  ↪upper_outliers)]

# do not display y axis in scientific notation
plt.ticklabel_format(style='plain')

# create boxplot
updated_price.boxplot(column=["price"])
plt.title('Boxplot of House Prices')
plt.text(1.1, 480000, 'Median is 450,000', fontsize=12)
plt.show()
plt.close()

# create histogram
updated_price.hist(column='price', ec='black')
plt.title('Histogram of House Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
plt.close()

print("We can see that house prices are right skewed and most frequent around␣
  ↪the 400k area.")
```

```
Now lets look at the centrality and distribution with boxplot and histogram
visualizations of a few select fields: price, sqft_living, and yr_built.
Let's start with house prices
The quartiles and the median are:
0.25    321950.0
0.50    450000.0
0.75    645000.0
Name: price, dtype: float64
The IQR is 323050.0
```



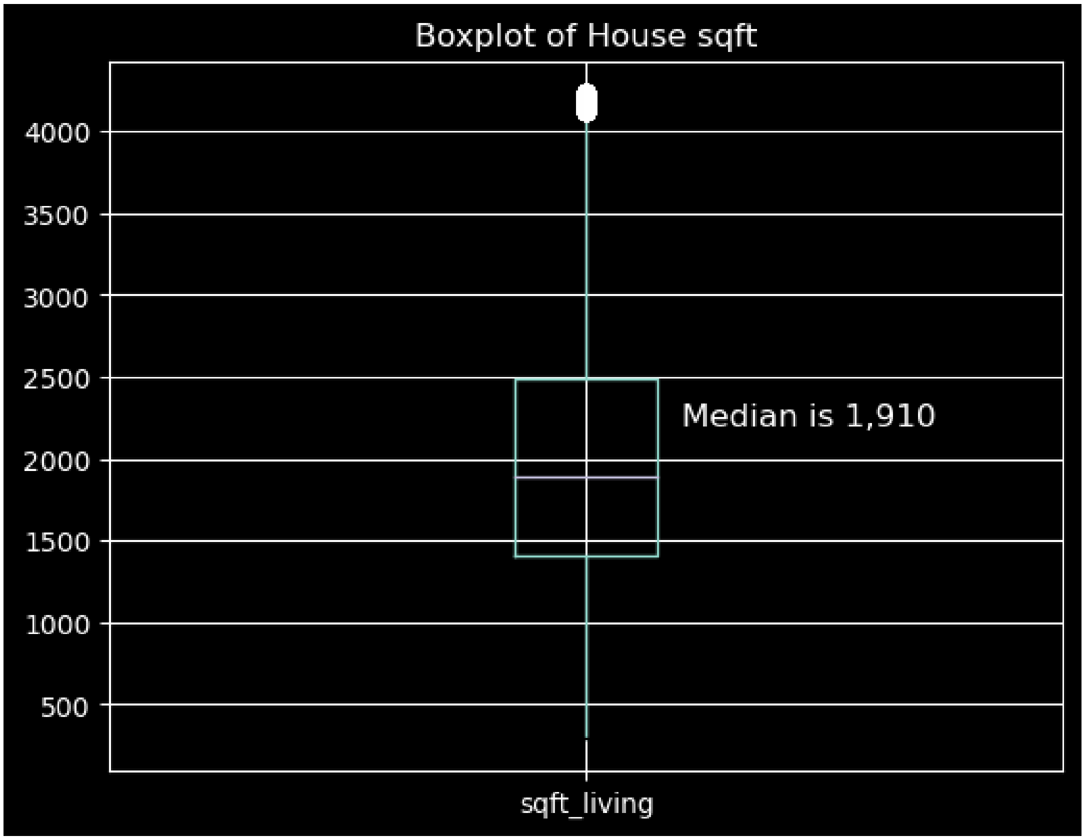


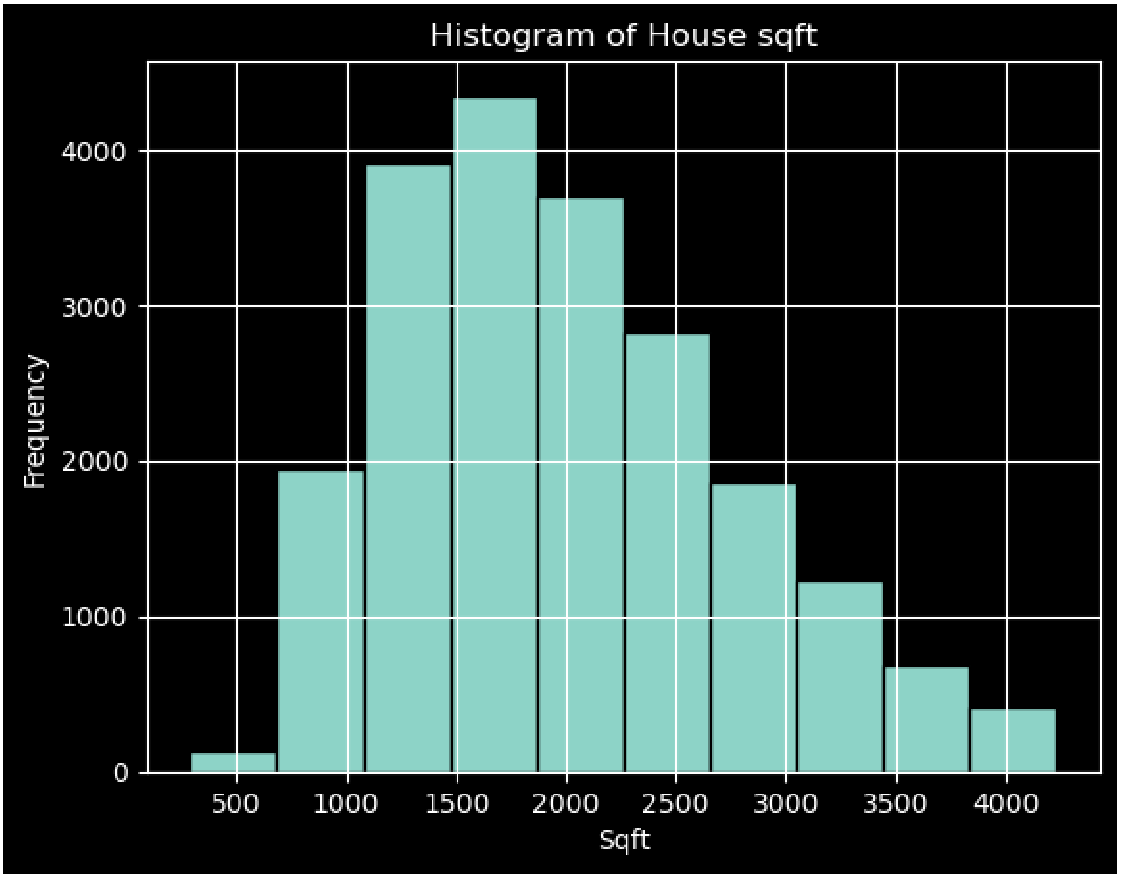
We can see that house prices are right skewed and most frequent around the 400k area.

[21]:
```
print("Now lets look at the square footage of living space")

# find the median and 1st and 3rd quartiles
print("The quartiles and the median are:\n" + str((df.sqft_living.quantile([0.
↪25,0.5,0.75]))))

# lets remove the outliers now for our plot

# find IQR
# find third and 1st quartile of price
# then subtract to find the difference
q3, q1 = np.percentile(df.sqft_living, [75, 25])
iqr = q3 - q1
print("The IQR is " + str(iqr.round(3)))

# create the lower and upper outlier points
lower_outliers = q1 - 1.5*iqr
```

```python
upper_outliers = q3 + 1.5*iqr

# create the new subset based on all rows that are greater than the lower␣
↪outliers and less than the upper outlier.
updated_sqft = df.loc[(df['sqft_living'] > lower_outliers) & (df['sqft_living']␣
↪< upper_outliers)]

# do not display y axis in scientific notation
plt.ticklabel_format(style='plain')

# create boxplot
updated_sqft.boxplot(column=["sqft_living"])
plt.title('Boxplot of House sqft')
plt.text(1.1, 2200, 'Median is 1,910', fontsize=12)
plt.show()
plt.close()

# create histogram
updated_sqft.hist(column='sqft_living', ec='black')
plt.title('Histogram of House sqft')
plt.xlabel('Sqft')
plt.ylabel('Frequency')
plt.show()
plt.close()

print("We can see that house prices are right skewed and most frequent around␣
↪the 2000 area.")
```

```
Now lets look at the square footage of living space
The quartiles and the median are:
0.25    1427.0
0.50    1910.0
0.75    2550.0
Name: sqft_living, dtype: float64
The IQR is 1123.0
```


Boxplot of House sqft
4000
3500
3000
2500
2000
1500
1000
500
Median is 1,910
sqft_living

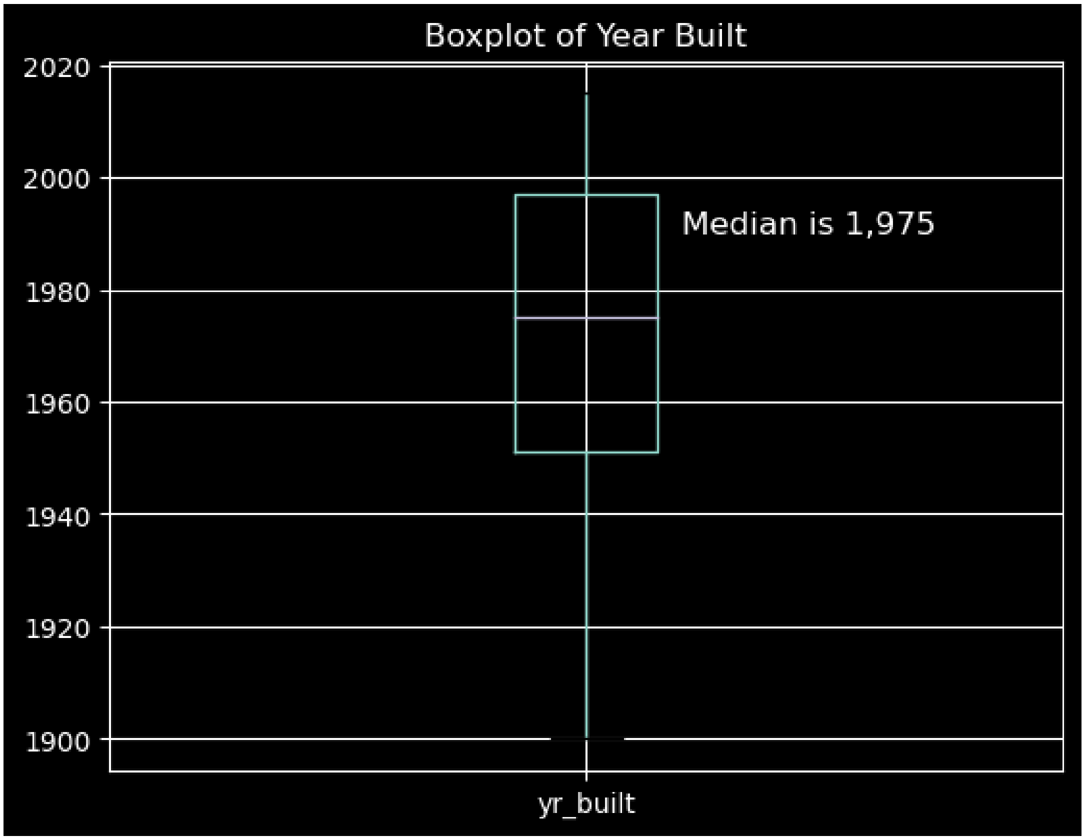


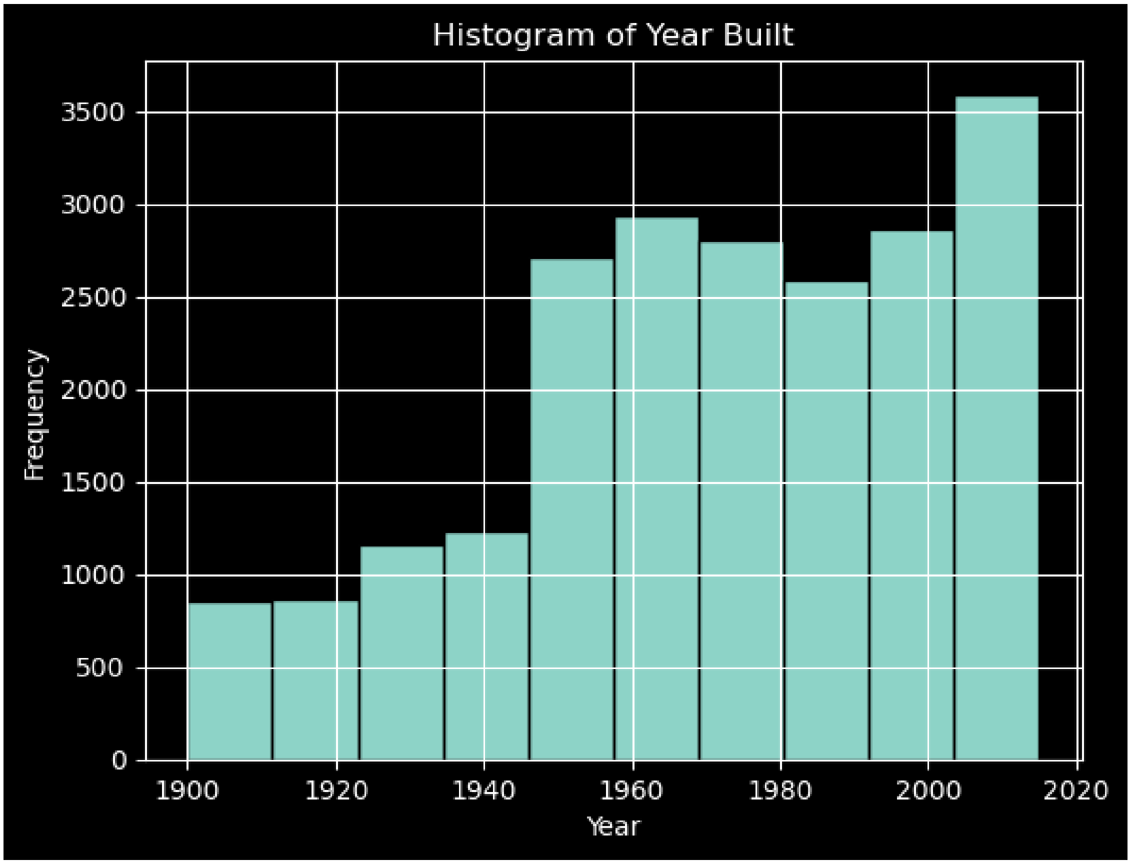
We can see that house prices are right skewed and most frequent around the 2000 area.

[22]:
```
print("Lastly lets look at the year built")

# find the median and 1st and 3rd quartiles
print("The quartiles and the median are:\n" + str((df.yr_built.quantile([0.25,0.
↪5,0.75]))))

# lets remove the outliers now for our plot

# find IQR
# find third and 1st quartile of price
# then subtract to find the difference
q3, q1 = np.percentile(df.yr_built, [75, 25])
iqr = q3 - q1
print("The IQR is " + str(iqr.round(3)))

# create the lower and upper outlier points
lower_outliers = q1 - 1.5*iqr
```

```
upper_outliers = q3 + 1.5*iqr

# create the new subset based on all rows that are greater than the lower␣
↪outliers and less than the upper outlier.
updated_year = df.loc[(df['yr_built'] > lower_outliers) & (df['yr_built'] <␣
↪upper_outliers)]

# do not display y axis in scientific notation
plt.ticklabel_format(style='plain')

# create boxplot
updated_year.boxplot(column=["yr_built"])
plt.title('Boxplot of Year Built')
plt.text(1.1, 1990, 'Median is 1,975', fontsize=12)
plt.show()
plt.close()

# create histogram
updated_year.hist(column='yr_built', ec='black')
plt.title('Histogram of Year Built')
plt.xlabel('Year')
plt.ylabel('Frequency')
plt.show()
plt.close()

print("We can see that house prices are left skewed and most frequent around␣
↪the 2020 area.")
```

```
Lastly lets look at the year built
The quartiles and the median are:
0.25    1951.0
0.50    1975.0
0.75    1997.0
Name: yr_built, dtype: float64
The IQR is 46.0
```


Boxplot of Year Built
2020
2000
1980
1960
1940
1920
1900
Median is 1,975
yr_built



We can see that house prices are left skewed and most frequent around the 2020 area.

[70]:
```python
print("Now let's diagnose for correlations between some variables and determine␣
↪independent and dependent variables")

print(np.corrcoef(df.sqft_living, df.price))
print("There is a strong positive correlation between price and square footage␣
↪of 0.7020")

print(np.corrcoef(df.yr_built, df.price))
print("There is a very weak or no positive correlation between price and year␣
↪built of 0.0540")

print(np.corrcoef(df.yr_built, df.sqft_living))
print("There is a weak positive correlation between square footage and year␣
↪built of 0.3180")
```

```
print("From the analysis, we can try to predict price based on multiple␣
↪variables such as the square footage and the year built. That means all the␣
↪independent or x variables (square feet and year built) together can be used␣
↪to try to predict or dependent or y variable (price).")
```

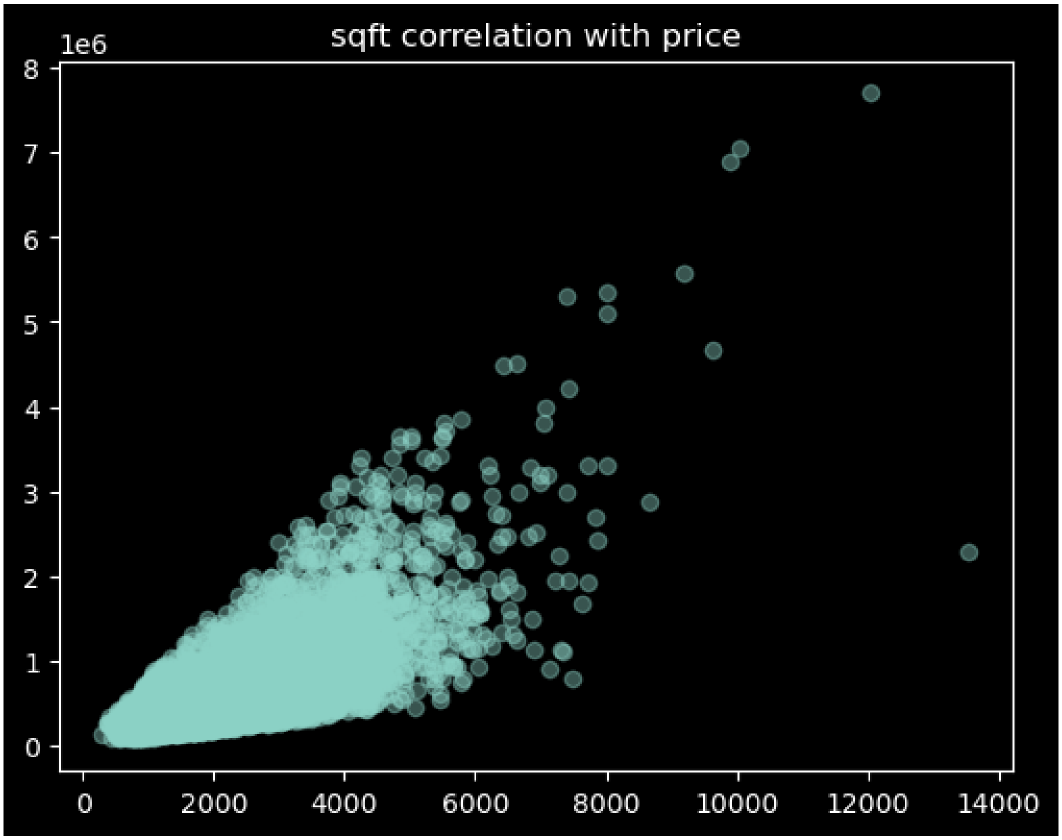
```
Now let's diagnose for correlations between some variables and determine
independent and dependent variables
[[1.         0.70203505]
 [0.70203505 1.        ]]
There is a strong positive correlation between price and square footage of
0.7020
[[1.         0.05401153]
 [0.05401153 1.        ]]
There is a very weak or no positive correlation between price and year built of
0.0540
[[1.         0.31804877]
 [0.31804877 1.        ]]
There is a weak positive correlation between square footage and year built of
0.3180
From the analysis, we can try to predict price based on multiple variables such
as the square footage and the year built. That means all the independent or x
variables (square feet and year built) together can be used to try to predict or
dependent or y variable (price).
```

[29]:

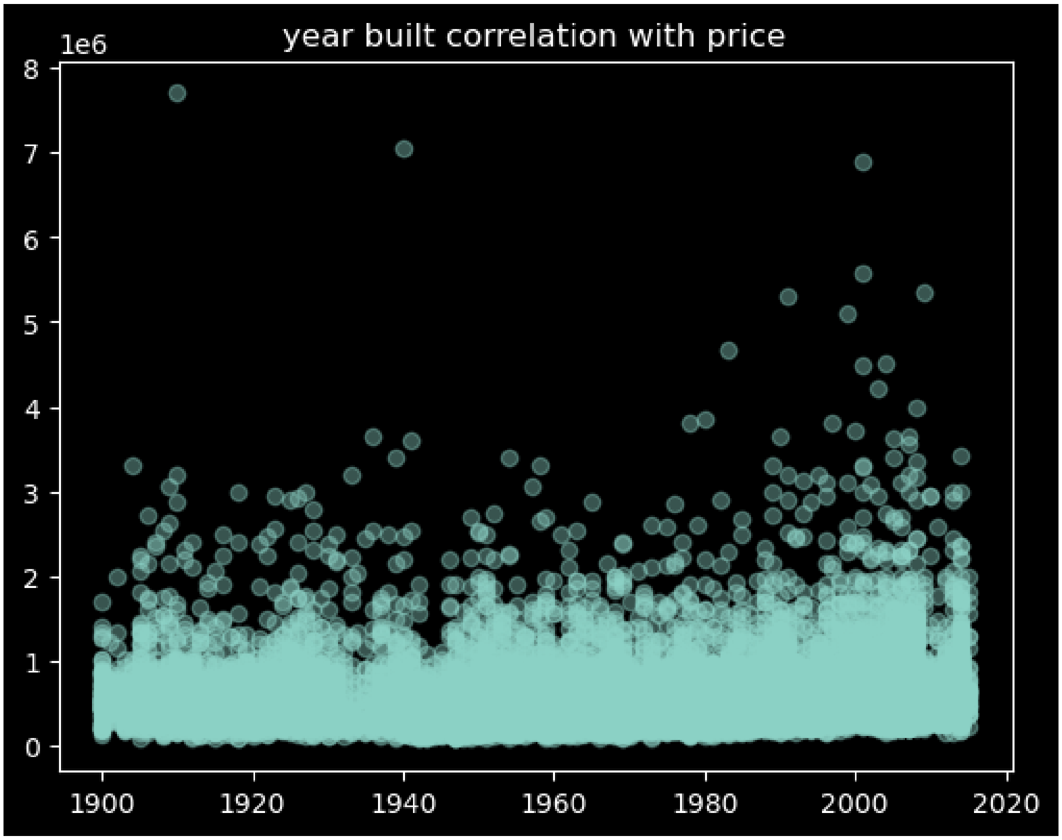
```
print("Finally let's perform exploratory analysis in combination with␣
↪visualization techniques to discover patterns and features of interest")
print("Let's look at the scatter plots and the correlations of price,␣
↪sqft_living, and yr_built")

# plot scatters to show the 3 attributes and their correlation with each other
plt.scatter(df[['sqft_living']], df[['price']], alpha=0.4)
plt.title("sqft correlation with price")
plt.show()
print("There seems to be a strong positive correlation between square feet and␣
↪price.\nThis confirms the previously calculated correlation of 0.7020")

plt.scatter(df[['yr_built']], df[['price']], alpha=0.4)
plt.title("year built correlation with price")
plt.show()
print("There seems to be a very small positive correlation between year built␣
↪and price.\nThis confirms the previously calculated correlation of 0.0540")

plt.scatter(df[['yr_built']], df[['sqft_living']], alpha=0.4)
plt.title("sqft correlation with year built")
plt.show()
```

```python
print("There seems to be a slightly positive correlation between floor year␣
↪built and square feet.\nThis confirms the previously calculated correlation␣
↪of 0.3180")

# create a heatmap to show correlations between all attributes
correlation_df = df[["price", "sqft_living", "yr_built", "bedrooms",␣
↪"bathrooms", "floors", "yr_renovated"]]

#Use the `.corr()` method on `df` to get the correlation matrix
correlation_matrix = correlation_df.corr()

# create heatmap, set hues for negative, positive areas of map and saturation␣
↪amount.
# create heatmap given: dataset, value range to anchor map with (-1 and 1),␣
↪colormap name set above, set title
red_blue = sns.diverging_palette(220, 20, as_cmap=True)
sns.heatmap(correlation_matrix, vmin = -1, vmax = 1, cmap=red_blue).
↪set(title="Correlation Heatmap")
plt.show()

print("With this heatmap, we can see how strongly a few select variables␣
↪correlate with each other. This can give us an idea of a few variables that␣
↪we can use to try to predict prices.\nFor example: Price correlates strongly␣
↪with square footage and number of bathrooms")
```

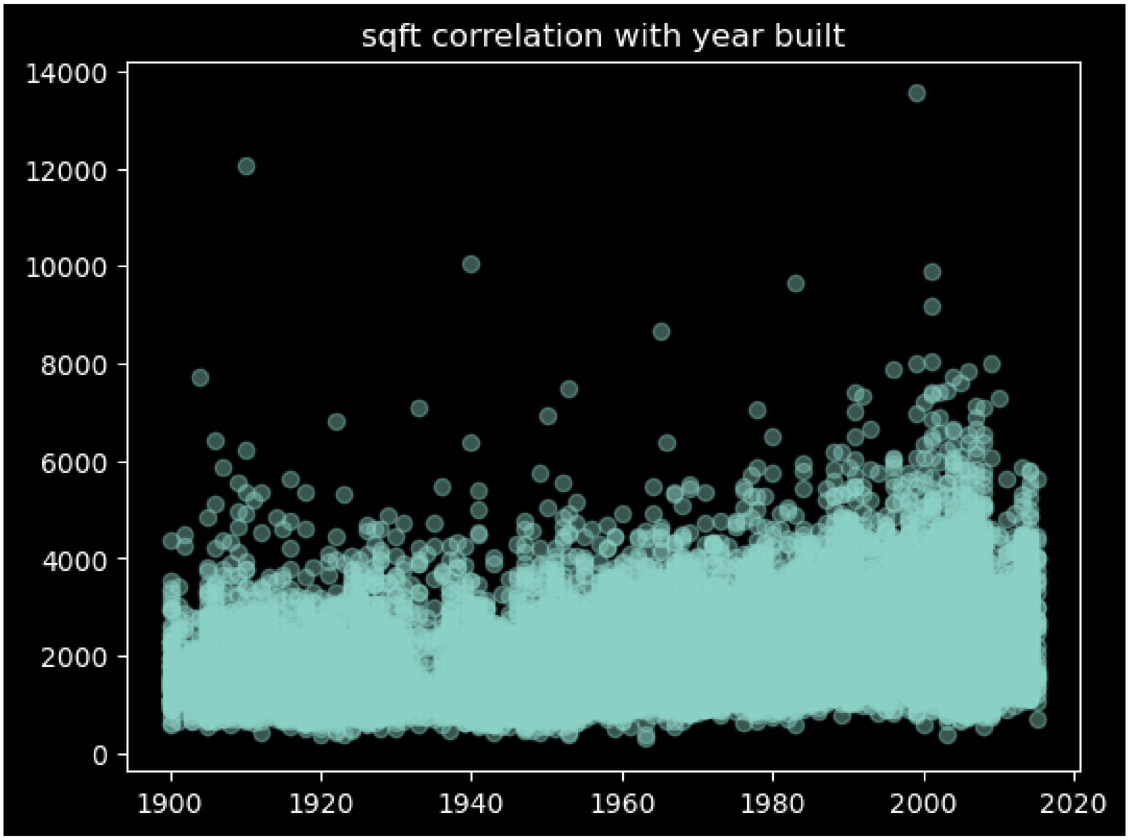
Finally let's perform exploratory analysis in combination with visualization techniques to discover patterns and features of interest
Let's look at the scatter plots and the correlations of price, sqft_living, and yr_built



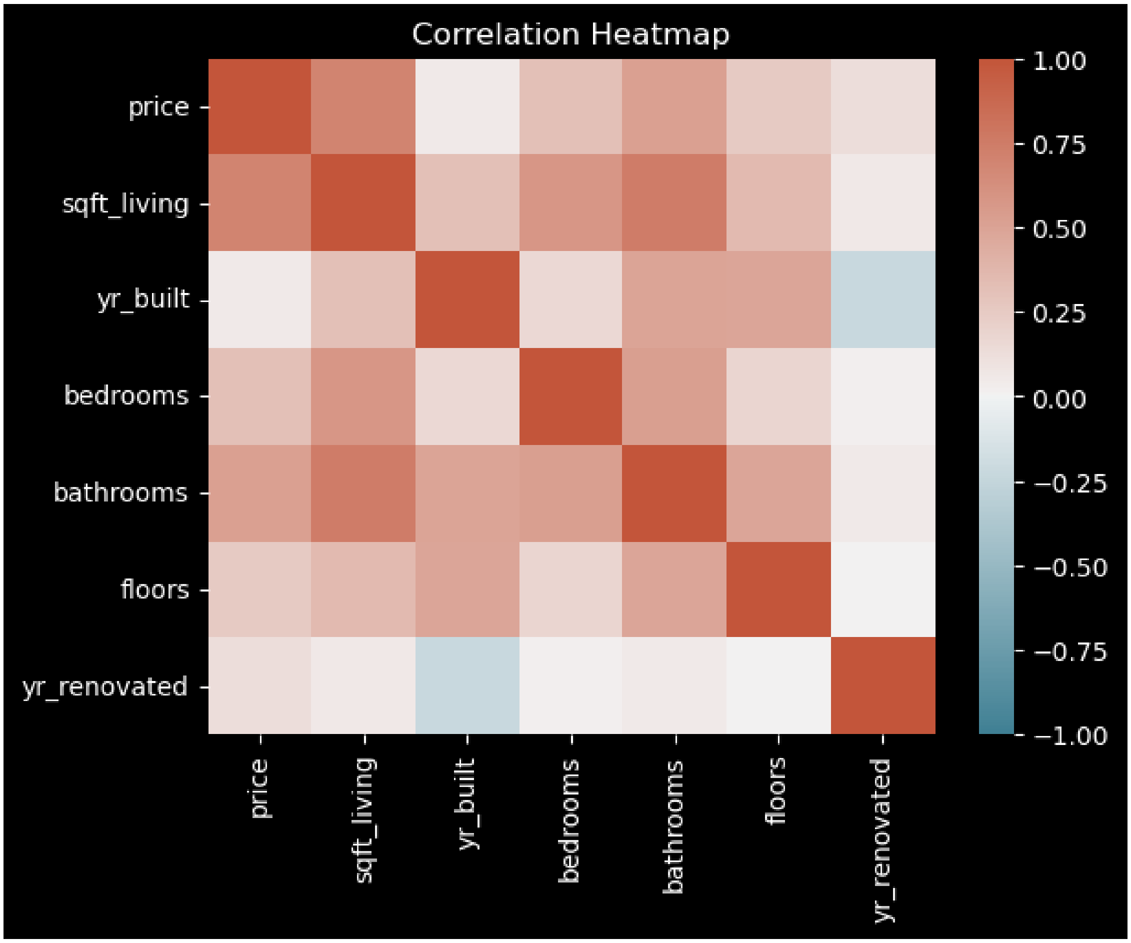
There seems to be a strong positive correlation between square feet and price.
This confirms the previously calculated correlation of 0.7020




```
There seems to be a very small positive correlation between year built and
price.
This confirms the previously calculated correlation of 0.0540
```



There seems to be a slightly positive correlation between floor year built and square feet.
This confirms the previously calculated correlation of 0.3180




```
With this heatmap, we can see how strongly a few select variables correlate with
each other. This can give us an idea of a few variables that we can use to try
to predict prices.
For example: Price correlates strongly with square footage and number of
bathrooms
```

[ ]:

```

```

3 Building a Model

3.1 Test Different Options & Choose One with Lowest RMSE

[16]:
```
# Choose independent & dependent variables for model
X =␣
↪df[['date','bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront',
       'view','condition','grade','sqft_above','sqft_basement','yr_built',
       'yr_renovated','zipcode','lat','long']]
Y = df[['price']]
```

[17]:
```
# Split data into 70% training and 30% test data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,␣
↪random_state = 222)
```

[18]:
```
# Linear Regression
lin_model = LinearRegression().fit(x_train, y_train)
lin_pred = lin_model.predict(x_test)

# Linear Regression RMSE
lin_rmse = np.sqrt(MSE(y_test, lin_pred))
print(lin_rmse)
```

```
211274.4353263405
```

[19]:
```
# XG Boost
xgb = xg.XGBRegressor(objective = 'reg:squarederror', n_estimators = 10, seed =␣
↪222)
xgb_model = xgb.fit(x_train, y_train)
xgb_pred = xgb_model.predict(x_test)

# XGB RMSE
xgb_rmse = np.sqrt(MSE(y_test, xgb_pred))
print(xgb_rmse)
```

```
144182.06142324655
```

[20]:
```
# Random Forest
rf_model = RandomForestClassifier().fit(x_train,y_train.values.ravel())
rf_pred = rf_model.predict(x_test)

# RF RMSE
rf_rmse = np.sqrt(MSE(y_test, rf_pred))
print(rf_rmse)
```

```
185379.738334969
```

[ ]:
```

```

3.2 Optimize Hyperparameters of Model with Lowest RMSE (XG Boost)

[26]:
```
# Set parameters to search through combinations of
params = { 'max_depth': [3,6,10],
           'learning_rate': [0.01, 0.05, 0.1],
           'n_estimators': [100, 500, 1000]}

# Initialize XG Boost Model & have grid_search iterate through all combinations␣
↪of parameters
xgb = xg.XGBRegressor(seed = 222)
grid_search = GridSearchCV(estimator=xgb,
                           param_grid=params,
                           scoring='neg_mean_squared_error',
                           verbose=1)
grid_search.fit(x_train, y_train)

# Print out best parameters & lowest RMSE
print("Best parameters:", grid_search.best_params_)
print("Lowest RMSE: ", (-grid_search.best_score_)**(1/2.0))
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Best parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 1000}
Lowest RMSE:  119993.12021454785
```

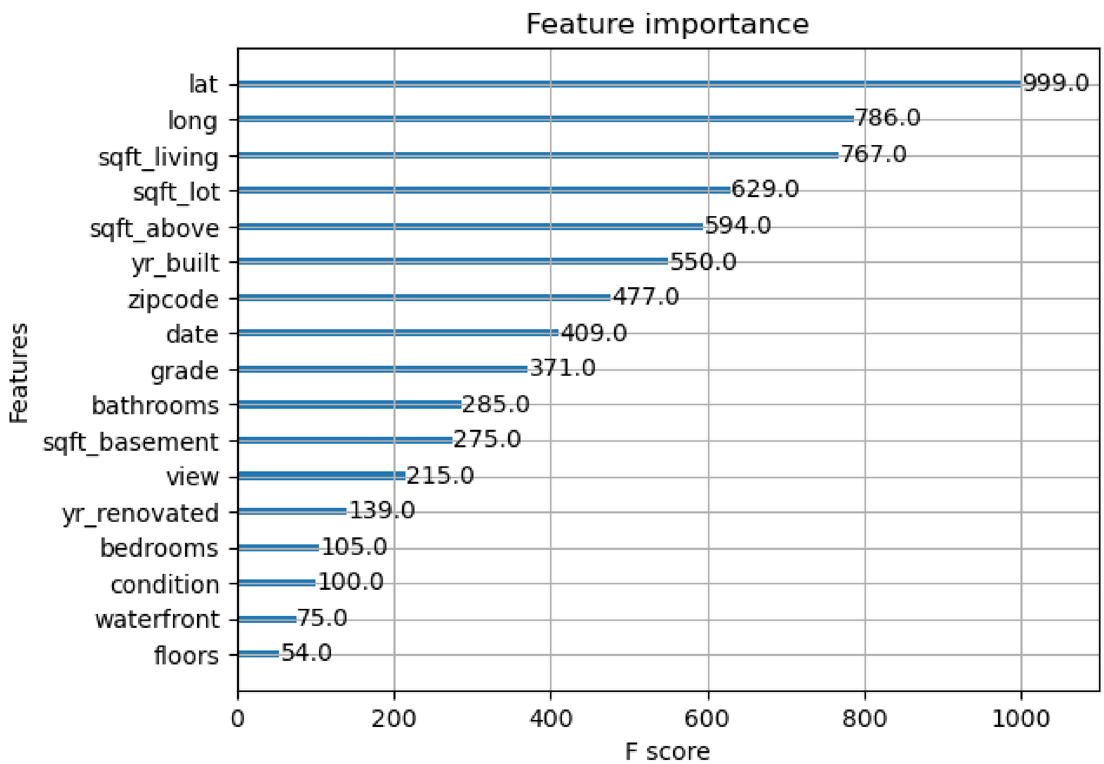
[28]:
```
# Build Final Model with Optimized Hyperparameters
xgb_final = xg.XGBRegressor(objective = 'reg:squarederror', n_estimators =␣
↪1000, max_depth = 3,
                            learning_rate = 0.1, seed = 222)
final_model = xgb_final.fit(x_train, y_train)
final_pred = final_model.predict(x_test)

# Final Model RMSE
final_rmse = np.sqrt(MSE(y_test, final_pred))
print(final_rmse)
```

```
123890.49542354565
```

[ ]:
```

```

3.3 Visualize Feature Importance

[33]:
```
# Visualize importance of features in models
xg.plot_importance(final_model)
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()

%matplotlib inline
```



3.4 Get Predictions for Example House

[65]:
```
# Example House: https://www.zillow.com/homedetails/
 ↪516-E-Denny-Way-Seattle-WA-98122/49011225_zpid/
bedrooms = 4
bathrooms = 3
sqft_living = 2380
sqft_above = 2380
sqft_basement = 0
sqft_lot = 2099
floors = 1
waterfront = 0
view = 0
yr_built = 1919
yr_renovated = 1919
date = 20221210
lat = 47.618680
long = -122.324470
zipcode = 98122
grade = 7 # made this up as I don't know how grade was defined in our dataset
condition = 4 # made this up as I don't know how condition was defined in our␣
 ↪dataset
```

```
# Create 1-row dataframe that will play role of "x_test" data
ex_house = pd.DataFrame([[date, bedrooms, bathrooms, sqft_living, sqft_lot,␣
↪floors, waterfront,
                         view, condition, grade, sqft_above, sqft_basement,␣
↪yr_built,
                         yr_renovated, zipcode, lat, long]],
                       columns=["date", "bedrooms", "bathrooms", "sqft_living",␣
↪"sqft_lot", "floors", "waterfront",
                         "view", "condition", "grade", "sqft_above",␣
↪"sqft_basement", "yr_built",
                         "yr_renovated", "zipcode", "lat", "long"])
```

[69]:
```
# Get predicted house value for our example house
print("The cost of this house is estimated to be: $" + str(final_model.
↪predict(ex_house)[0]))
```

```
The cost of this house is estimated to be: $878823.56
```

[ ]:
```
```