King County Housing Regression and Analysis

- Student name: Johnny Dryman
- Student pace: full time
- Scheduled project review date/time: 04/29/21, 2pm
- Instructor name: James Irving

INTRODUCTION

> For the Phase 2 Project, we will be analyzing housing sales data for King County (Seattle, WA area). We will be using multivariate linear regression to explore which features of the data have the greatest influence on price.

Business Problem

As home values continue to sky rocket in the pandemic era, many King County residents have inquired about how to increase the value of their homes. Fortunately, we have access to all homes sold in King County for roughly one year, from May 2014 - May 2015.

This data gives us access to a variety of important metrics both quantitative and qualitative.

After scrubbing the data and assuring quality, we will use multivariate linear regression to analyze our features and determine their relationship with sale price.

Finally, we will formulate our observations into useful recommendations to any resident interested in increasing their home value.

OBTAIN

We will begin by importing our packages for data exploration and load our .csv data into a pandas dataframe.

In [1532]:
```
import pandas as pd
import seaborn as sns
sns.set_theme(color_codes=True)
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv('data/kc_house_data.csv')

df.columns
```

Out[1532]:
```
Index(['id', 'date', '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'],
      dtype='object')
```

SCRUB

Data Preparation

We'll begin by getting a brief overview of our data and check for null values.

In [1533]:

```
df.head()

print(df.info())

print(df.isna().sum())
```

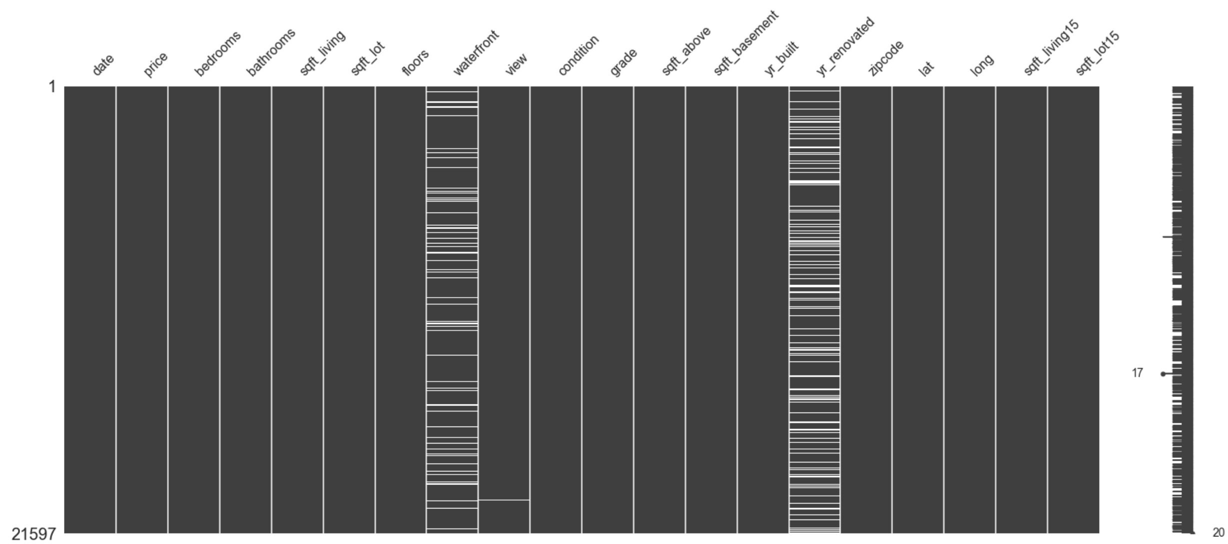
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
None
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

In [1534]:
```
df = df.set_index('id')
```

Using missingno package to visualize null values.

In [1535]:
```
import missingno as msno

msno.matrix(df)
```

Out[1535]: <AxesSubplot:>



Creating clean_column function, which will help us look at unique items. This will be useful for identifying any data that seems off or incorrect.

In [1536]:
```
def clean_column(column, unique_count=10):
    column_str = str(column)
    print('Datatype: ' + str(df[column].dtypes))
    print('Total unique itms: ' + str(df[column].nunique()))
    print('Displaying first ' + str(unique_count) + ':')
    print(df[column].unique()[0:unique_count])
    return column_str

def null_count(df):
    print('---Total Entries---')
    print(df.describe())
    print('---Non-Null Values---')
    print(df.notna().describe())
```

Let's take a look at our features that have null values.

We could conceivably estimate our null values, and that might be interesting for further analysis. Mapping could be used with 'latitude' and 'longitude' and potentially calculate distance to water. However, for this project, our safest bet will be to drop the null values.

In [1537]:
```
null_count(df['waterfront'])
```

```
---Total Entries---
count   19221.0000000000
mean        0.0075958587
std         0.0868248457
min         0.0000000000
25%         0.0000000000
50%         0.0000000000
75%         0.0000000000
max         1.0000000000
Name: waterfront, dtype: float64
---Non-Null Values---
count     21597
unique        2
top        True
freq      19221
Name: waterfront, dtype: object
```

In [1538]:
```
clean_column('waterfront')

df = df[df['waterfront'].notna()]

clean_column('waterfront')
```

```
Datatype: float64
Total unique itms: 2
Displaying first 10:
[nan  0.  1.]
Datatype: float64
Total unique itms: 2
Displaying first 10:
[0. 1.]
```

Out[1538]:
```
'waterfront'
```

View has very few null values, it is safe to remove them from the dataset.

In [1539]:
```
null_count(df['view'])
```

```
---Total Entries---
count    19164.0000000000
mean         0.2310582342
std          0.7633681938
min          0.0000000000
25%          0.0000000000
50%          0.0000000000
75%          0.0000000000
max          4.0000000000
Name: view, dtype: float64
---Non-Null Values---
count      19221
unique         2
top         True
freq       19164
Name: view, dtype: object
```

In [1540]:
```
clean_column('view')

df = df[df['view'].notna()]

clean_column('view')
```

```
Datatype: float64
Total unique itms: 5
Displaying first 10:
[ 0. nan  3.  4.  2.  1.]
Datatype: float64
Total unique itms: 5
Displaying first 10:
[0. 3. 4. 2. 1.]
```

Out[1540]:
```
'view'
```

Yr_renovated has ~3,500 null values. We would want to consider removing the column in this case, but yr_renovated indicates a renovation occurred with a year (e.g. 2007) and a renovation has never occurred with a zero (e.g. 0). The null values could also represent houses that have never been renovated, but we can't be sure.

In [1541]:
```
null_count(df['yr_renovated'])
```

```
---Total Entries---
count   15762.0000000000
mean       82.4402360107
std       397.2126256112
min         0.0000000000
25%         0.0000000000
50%         0.0000000000
75%         0.0000000000
max      2015.0000000000
Name: yr_renovated, dtype: float64
---Non-Null Values---
count     19164
unique        2
top        True
freq      15762
Name: yr_renovated, dtype: object
```

In [1542]:
```
clean_column('yr_renovated', unique_count=115)
```

```
Datatype: float64
Total unique itms: 70
Displaying first 115:
[1991.   nan    0. 2002. 2010. 1992. 2013. 1994. 1978. 2005. 2003. 1984.
 1954. 2014. 2011. 1983. 1990. 1988. 1977. 1981. 1995. 2000. 1999. 1998.
 1970. 1989. 2004. 1986. 2007. 1987. 2006. 1985. 2001. 1980. 1971. 1945.
 1979. 1997. 1950. 1969. 1948. 2009. 2015. 2008. 2012. 1968. 1963. 1951.
 1962. 1953. 1993. 1955. 1996. 1982. 1956. 1940. 1976. 1946. 1975. 1964.
 1973. 1957. 1959. 1960. 1965. 1967. 1934. 1972. 1944. 1958. 1974.]
```

Out[1542]:
```
'yr_renovated'
```

We will first remove rows with nan values from the dataset.

In [1543]:
```
df = df[df['yr_renovated'].notna()]

df.describe()


```

Out[1543]:

	price	bedrooms	bathrooms	sqft_living	sqf
count	15762.0000000000	15762.0000000000	15762.0000000000	15762.0000000000	15762.000000
mean	541317.1757391194	3.3789493719	2.1207968532	2084.5123715265	15280.821405
std	372225.8387270711	0.9353010799	0.7667716477	918.6176864783	41822.883323
min	82000.0000000000	1.0000000000	0.5000000000	370.0000000000	520.000000
25%	321000.0000000000	3.0000000000	1.7500000000	1430.0000000000	5048.500000
50%	450000.0000000000	3.0000000000	2.2500000000	1920.0000000000	7602.000000
75%	644875.0000000000	4.0000000000	2.5000000000	2550.0000000000	10720.000000
max	7700000.0000000000	33.0000000000	8.0000000000	13540.0000000000	1651359.000000

In [1544]:
```
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

In [1545]:
```
ren_df = df[df['yr_renovated'] != 0]

not_ren_df = df[df['yr_renovated'] == 0]

```

In [1546]:
```
ren_df['price'].describe()
```

Out[1546]:
```
count       651.00
mean     760872.06
std      637150.64
min      110000.00
25%      410000.00
50%      600000.00
75%      886250.00
max     7700000.00
Name: price, dtype: float64
```

In [1547]:
```
not_ren_df['price'].describe()
```

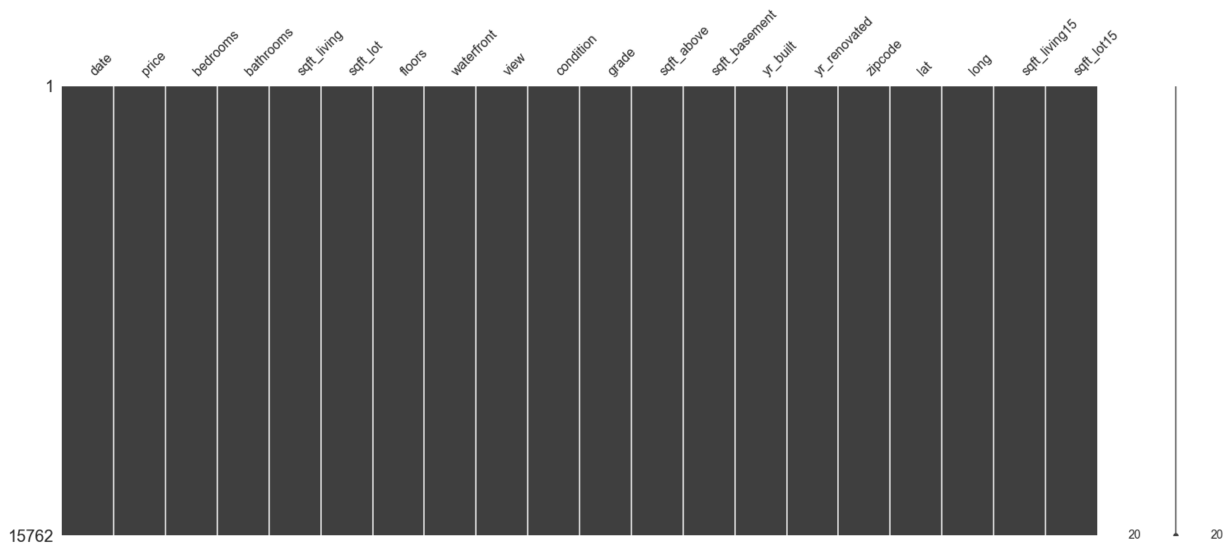
Out[1547]:
```
count     15111.00
mean     531858.49
std      353400.02
min       82000.00
25%      320000.00
50%      449000.00
75%      633000.00
max     6890000.00
Name: price, dtype: float64
```

The means, standard deviations, and medians for renovated and non-renovated houses are significant. We will revisit yr_renovated and potentially convert the column to a binary value.

Checking non-nulls again.

In [1548]:
```
msno.matrix(df)
```

Out[1548]: <AxesSubplot:>



Now we'll take a look at each column and see if anything needs correction.

Feature Review

In [1549]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15762 entries, 6414100192 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           15762 non-null  object 
 1   price          15762 non-null  float64
 2   bedrooms       15762 non-null  int64  
 3   bathrooms      15762 non-null  float64
 4   sqft_living    15762 non-null  int64  
 5   sqft_lot       15762 non-null  int64  
 6   floors         15762 non-null  float64
 7   waterfront     15762 non-null  float64
 8   view           15762 non-null  float64
 9   condition      15762 non-null  int64  
 10  grade          15762 non-null  int64  
 11  sqft_above     15762 non-null  int64  
 12  sqft_basement  15762 non-null  object 
 13  yr_built       15762 non-null  int64  
 14  yr_renovated   15762 non-null  float64
 15  zipcode        15762 non-null  int64  
 16  lat            15762 non-null  float64
 17  long           15762 non-null  float64
 18  sqft_living15  15762 non-null  int64  
 19  sqft_lot15     15762 non-null  int64  
dtypes: float64(8), int64(10), object(2)
memory usage: 2.5+ MB
None
```

We will define a few functions to more efficiently analyze individual features.

In [1550]:
```
def clean_column(column, unique_count=10):
    column_str = str(column)
    print('Datatype: ' + str(df[column].dtypes))
    print('Total unique itms: ' + str(df[column].nunique()))
    print('Displaying first ' + str(unique_count) + ':')
    print(df[column].unique()[0:unique_count])
    print(f"Minimum value: {df[column].min()}.  Maximum value: {df[column].m
    print(df[column].describe())
    return column_str

def regplot(column, df=df):
    return sns.regplot(data=df, x=column, y='price')

def hist(column):
    hist = df[column].hist()
    return plt.show()

def displot(column):
    return sns.displot(data=df, x=column, y='price')
```

Price

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Date

In [1551]:
```
df['date'] = df['date'].apply(pd.to_datetime)

```

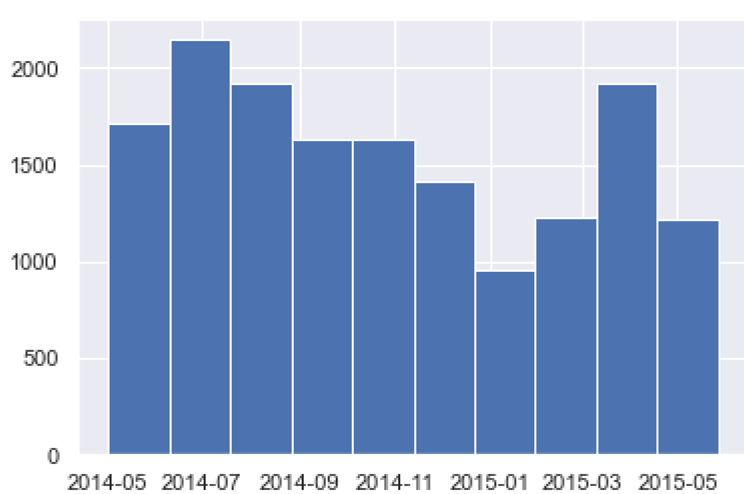
In [1552]:
```
clean_column('date')

df['bedrooms'].describe()
```

```
Datatype: datetime64[ns]
Total unique itms: 369
Displaying first 10:
['2014-12-09T00:00:00.000000000' '2015-02-18T00:00:00.000000000'
 '2014-05-12T00:00:00.000000000' '2014-06-27T00:00:00.000000000'
 '2015-04-15T00:00:00.000000000' '2015-03-12T00:00:00.000000000'
 '2014-05-27T00:00:00.000000000' '2014-10-07T00:00:00.000000000'
 '2015-01-24T00:00:00.000000000' '2014-07-31T00:00:00.000000000']
Minimum value: 2014-05-02 00:00:00.  Maximum value: 2015-05-27 00:00:00
count                     15762
unique                      369
top       2014-06-25 00:00:00
freq                        103
first     2014-05-02 00:00:00
last      2015-05-27 00:00:00
Name: date, dtype: object

<ipython-input-1550-17aaff753c74>:8: FutureWarning: Treating datetime data as c
ategorical rather than numeric in `.describe` is deprecated and will be removed
in a future version of pandas. Specify `datetime_is_numeric=True` to silence th
is warning and adopt the future behavior now.
  print(df[column].describe())
```

Out[1552]:
```
count    15762.00
mean         3.38
std          0.94
min          1.00
25%          3.00
50%          3.00
75%          4.00
max         33.00
Name: bedrooms, dtype: float64
```

In [1553]:
```
df['date'].hist()
```

Out[1553]: <AxesSubplot:>



Bedrooms

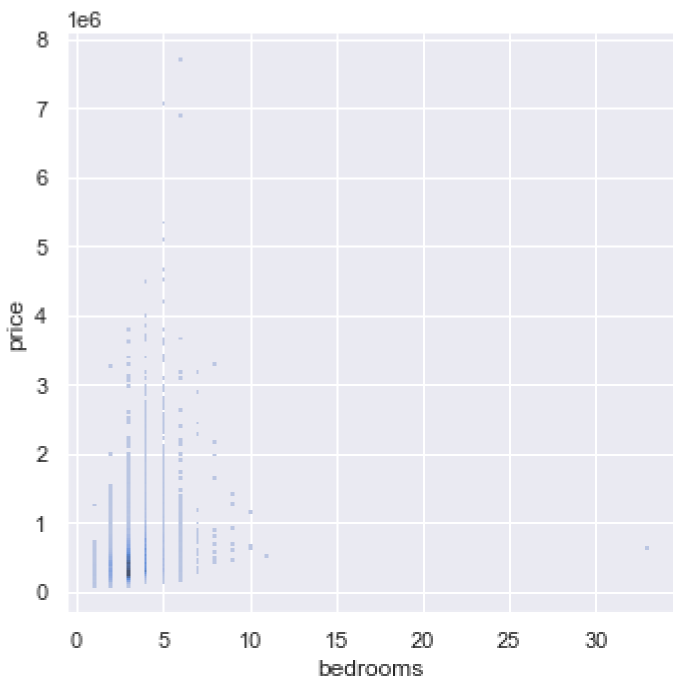
In [1554]:
```
clean_column('bedrooms', unique_count=20)
```

```
Datatype: int64
Total unique itms: 12
Displaying first 20:
[ 3  4  2  5  1  6  7  8  9 11 10 33]
Minimum value: 1.  Maximum value: 33
count    15762.00
mean         3.38
std          0.94
min          1.00
25%          3.00
50%          3.00
75%          4.00
max         33.00
Name: bedrooms, dtype: float64
```

Out[1554]: 'bedrooms'

In [1555]:
```
displot('bedrooms')
```

Out[1555]: <seaborn.axisgrid.FacetGrid at 0x2342078f100>



In [1556]:
```
df.loc[df['bedrooms'] == 33]
```

Out[1556]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
2402100895	2014-06-25	640000.00	33	1.75	1620	6000	1.00	0.00	0.00

Based on other stats, we assume the one entry with 33 bedrooms to actually be 3 bedrooms. Correcting below.

In [1557]:
```
df['bedrooms'] = df['bedrooms'].replace([33],3)

df.loc[df['bedrooms'] == 33]
```

Out[1557]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gra

In [1558]:
```
df.loc[df['bedrooms'] == 11]
```

Out[1558]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
1773100755	2014-08-21	520000.00	11	3.00	3000	4960	2.00	0.00	0.00

The 11 bedroom house also seems unlikely based on square footage. Googling the ID '1773100755' revelas it to be a 4 bedroom house.

In [1559]:
```
df['bedrooms'] = df['bedrooms'].replace([11],4)

df.loc[df['bedrooms'] == 11]

```

Out[1559]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gra

In [1560]:
```
df.loc[df['bedrooms'] == 10]
```

Out[1560]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
627300145	2014-08-14	1150000.00	10	5.25	4590	10920	1.00	0.00	2.0(
5566100170	2014-10-29	650000.00	10	2.00	3610	11914	2.00	0.00	0.0(
8812401450	2014-12-29	660000.00	10	3.00	2920	3745	2.00	0.00	0.0(

Even though two of the 10 bedroom houses seem unlikely, a quick google shows that they are recorded as 9 bedroom houses on zillow. We will assume these entries were accurate at the time, and will not change.

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

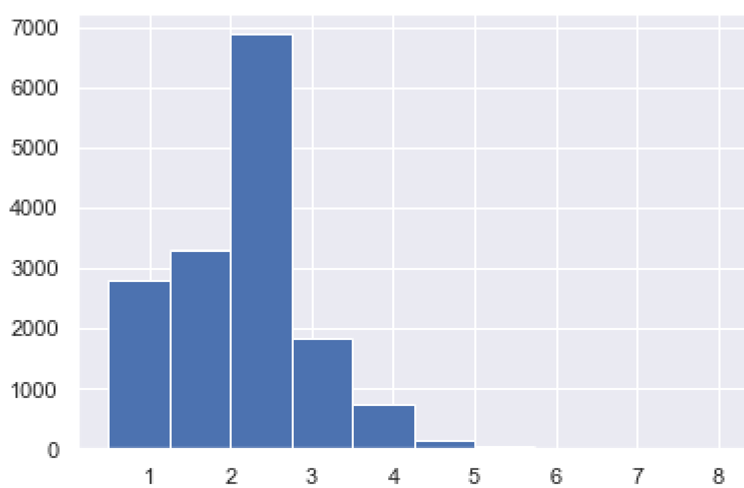
Bathrooms

In [1561]:
```
clean_column('bathrooms', unique_count=29)
```

```
Datatype: float64
Total unique itms: 27
Displaying first 29:
[2.25 3.   2.   4.5  1.   2.5  1.75 2.75 1.5  3.25 4.   3.5  0.75 5.
 4.25 3.75 1.25 5.25 4.75 0.5  5.5  6.   5.75 8.   6.75 7.5  7.75]
Minimum value: 0.5.  Maximum value: 8.0
count   15762.00
mean        2.12
std         0.77
min         0.50
25%         1.75
50%         2.25
75%         2.50
max         8.00
Name: bathrooms, dtype: float64
```

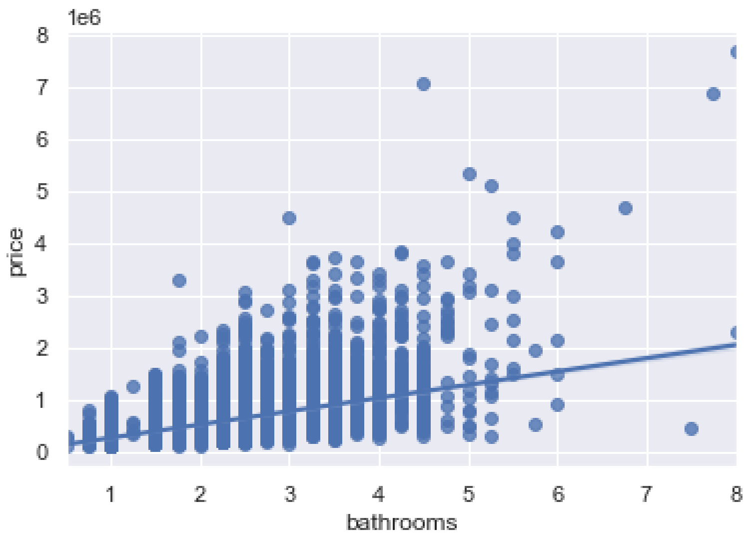
Out[1561]: 'bathrooms'

In [1562]:
```
hist('bathrooms')
```



In [1563]:
```
1 regplot('bathrooms')
```

Out[1563]: <AxesSubplot:xlabel='bathrooms', ylabel='price'>



Squarefoot - Living

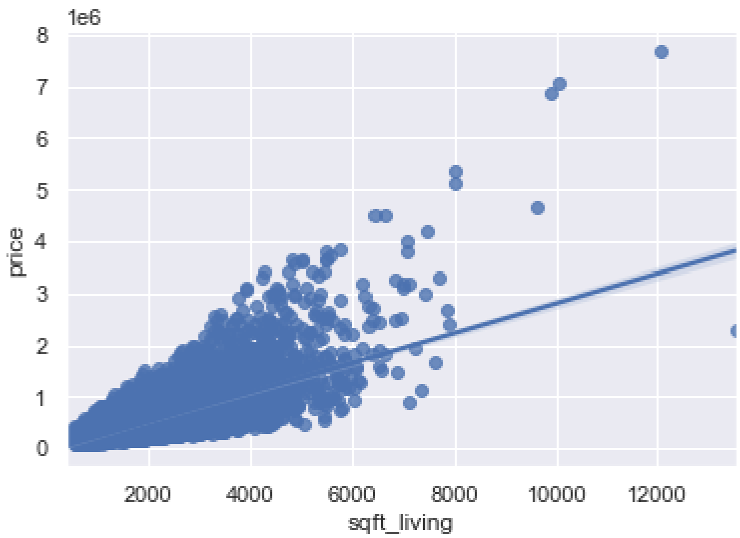
In [1564]:
```
1 clean_column('sqft_living')
```

```
Datatype: int64
Total unique itms: 912
Displaying first 10:
[2570 1960 1680 5420 1715 1780 1890 1160 1370 1810]
Minimum value: 370.  Maximum value: 13540
count   15762.00
mean     2084.51
std       918.62
min       370.00
25%      1430.00
50%      1920.00
75%      2550.00
max     13540.00
Name: sqft_living, dtype: float64
```

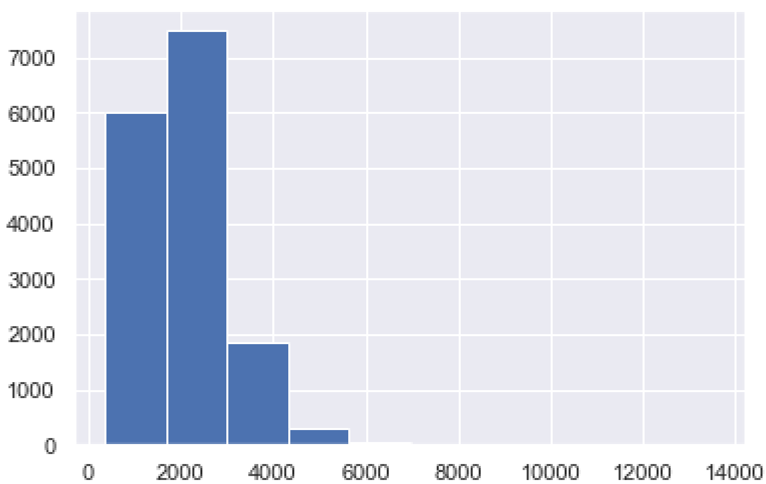
Out[1564]: 'sqft_living'

In [1565]:
```
regplot('sqft_living')
```

Out[1565]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



In [1566]:
```
hist('sqft_living')
```



There seem to be at least one unusual outlier for the price. We will want to take a look at the largest values to verify the quality of the data.

In [1567]:
```
df.sort_values(by=['sqft_living'], ascending=False).head(5)
```

Out[1567]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
1225069038	2014-05-05	2280000.00	7	8.00	13540	307752	3.00	0.00	4.0(
6762700020	2014-10-13	7700000.00	6	8.00	12050	27600	2.50	0.00	3.0(
9808700762	2014-06-11	7060000.00	5	4.50	10040	37325	2.00	1.00	2.0(
9208900037	2014-09-19	6890000.00	6	7.75	9890	31374	2.00	0.00	4.0(
1924059029	2014-06-17	4670000.00	5	6.75	9640	13068	1.00	1.00	4.0(

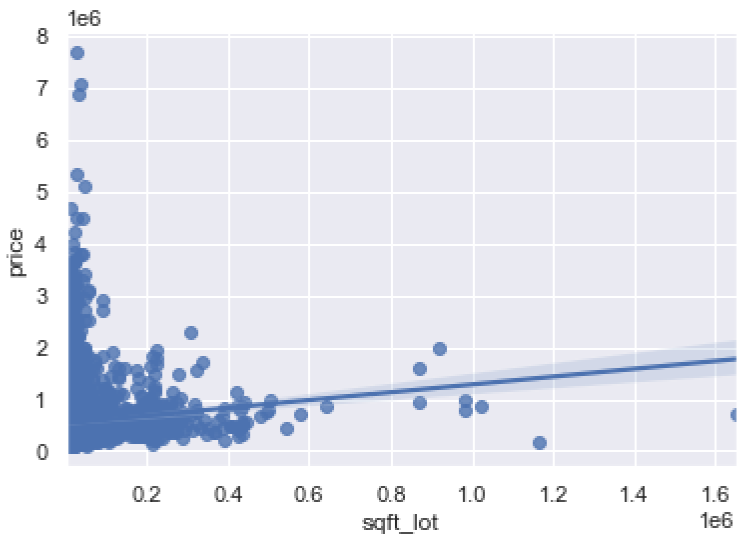
After reviewing the one outlier, it seems to be a compound in a rural area, and the sqft seems realistic.

Squarefoot - Lot

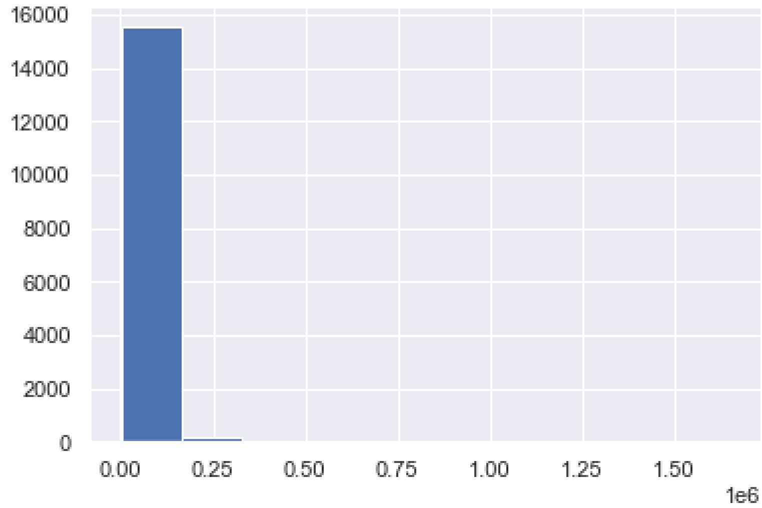
In [1568]:
```
clean_column('sqft_lot')
regplot('sqft_lot')
```

```
Datatype: int64
Total unique itms: 7927
Displaying first 10:
[  7242   5000   8080 101930   6819   7470   6560   6000   9680   4850]
Minimum value: 520.  Maximum value: 1651359
count      15762.00
mean       15280.82
std        41822.88
min          520.00
25%         5048.50
50%         7602.00
75%        10720.00
max      1651359.00
Name: sqft_lot, dtype: float64
```

Out[1568]: <AxesSubplot:xlabel='sqft_lot', ylabel='price'>



In [1569]:
```
hist('sqft_lot')
```



In [1570]:
```
df.sort_values(by=['sqft_lot'], ascending=False).head(5)

#outlier looks like a farm, will keep
```

Out[1570]:

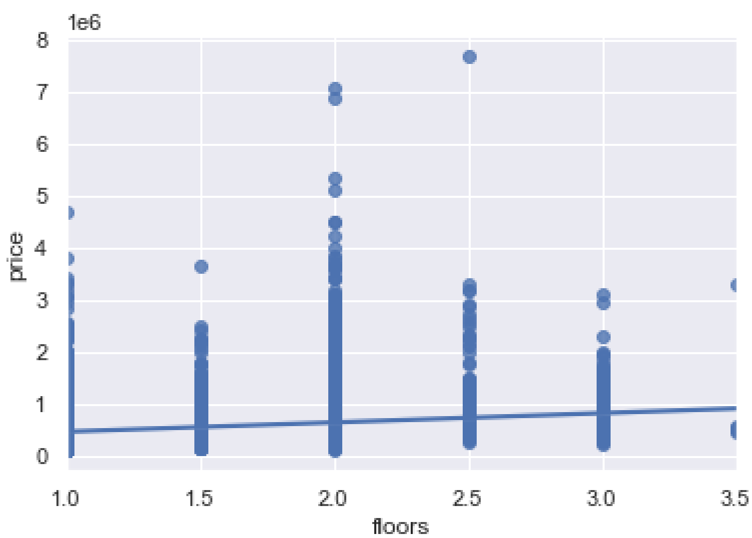
id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
1020069017	2015-03-27	700000.00	4	1.00	1300	1651359	1.00	0.00	3.00
3326079016	2015-05-04	190000.00	2	1.00	710	1164794	1.00	0.00	0.00
2323089009	2015-01-19	855000.00	4	3.50	4030	1024068	2.00	0.00	0.00
722069232	2014-09-05	998000.00	4	3.25	3770	982998	2.00	0.00	0.00
3626079040	2014-07-30	790000.00	2	3.00	2560	982278	1.00	0.00	0.00

Floors

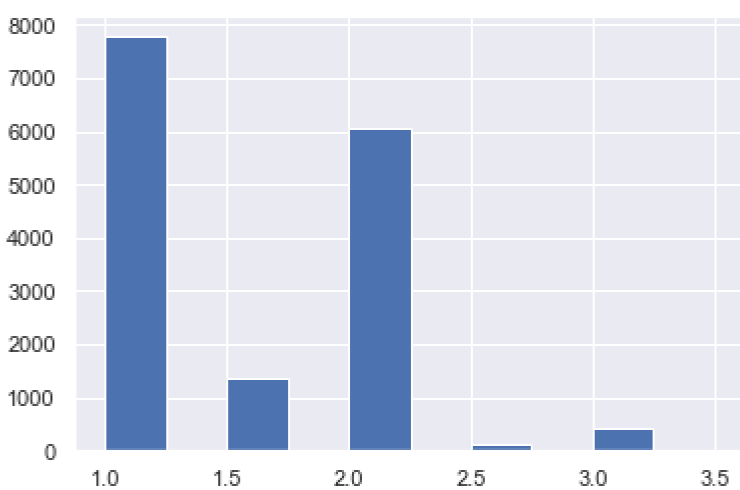
In [1571]:
```
clean_column('floors')

regplot('floors')
```

```
Datatype: float64
Total unique itms: 6
Displaying first 10:
[2.  1.  1.5 3.  2.5 3.5]
Minimum value: 1.0.  Maximum value: 3.5
count   15762.00
mean        1.50
std         0.54
min         1.00
25%         1.00
50%         1.50
75%         2.00
max         3.50
Name: floors, dtype: float64
```

Out[1571]: <AxesSubplot:xlabel='floors', ylabel='price'>



In [1572]:
```
hist('floors')
```

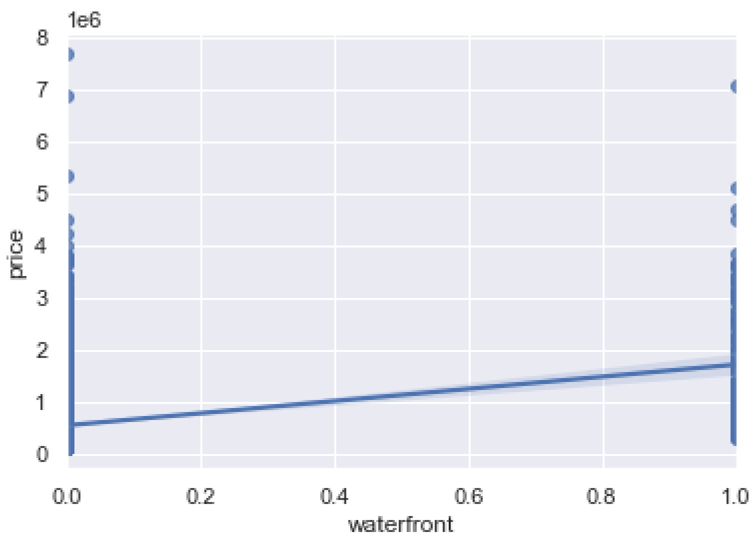


Waterfront

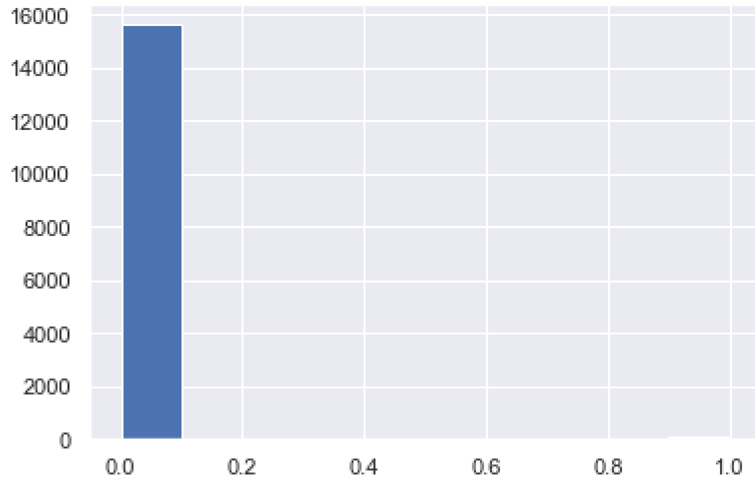
In [1573]:
```
clean_column('waterfront')

regplot('waterfront')
```

```
Datatype: float64
Total unique itms: 2
Displaying first 10:
[0. 1.]
Minimum value: 0.0.  Maximum value: 1.0
count   15762.00
mean        0.01
std         0.09
min         0.00
25%         0.00
50%         0.00
75%         0.00
max         1.00
Name: waterfront, dtype: float64
```

Out[1573]: <AxesSubplot:xlabel='waterfront', ylabel='price'>



In [1574]:
```
hist('waterfront')
```



View

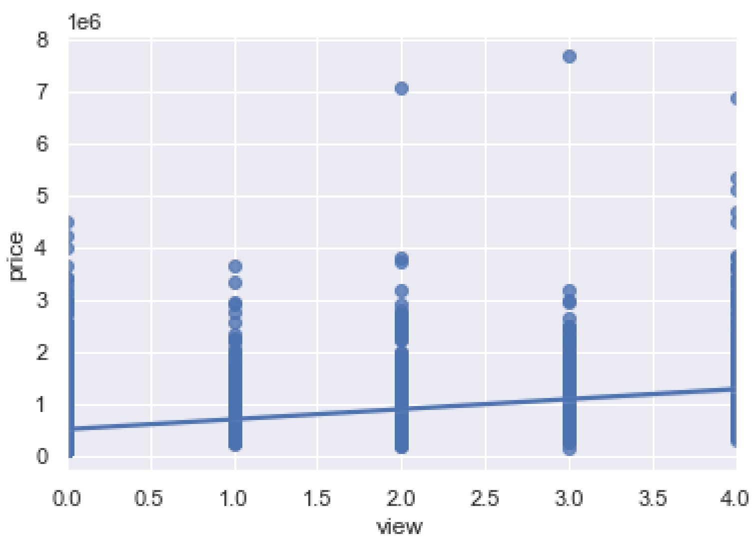
In [1575]:

```
clean_column('view')

regplot('view')
```

```
Datatype: float64
Total unique itms: 5
Displaying first 10:
[0. 3. 4. 2. 1.]
Minimum value: 0.0.  Maximum value: 4.0
count    15762.00
mean         0.23
std          0.76
min          0.00
25%          0.00
50%          0.00
75%          0.00
max          4.00
Name: view, dtype: float64
```

Out[1575]:

```
<AxesSubplot:xlabel='view', ylabel='price'>
```



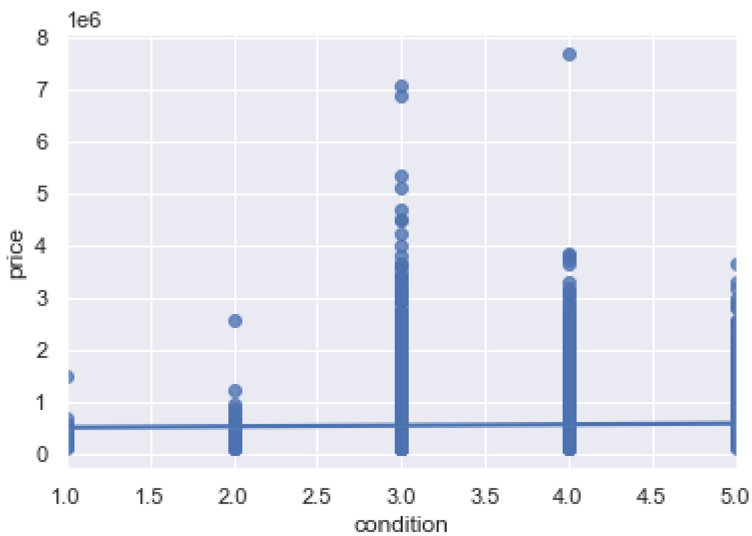
Condition

In [1576]:

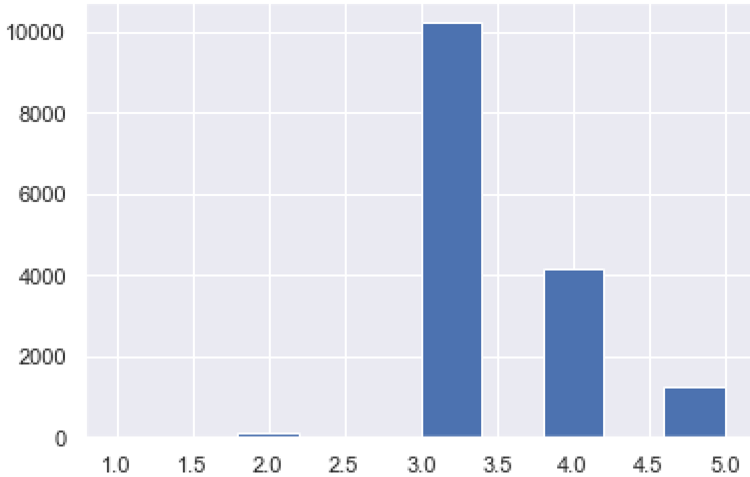
```
clean_column('condition')

regplot('condition')
```

```
Datatype: int64
Total unique itms: 5
Displaying first 10:
[3 5 4 1 2]
Minimum value: 1.  Maximum value: 5
count   15762.00
mean        3.41
std         0.65
min         1.00
25%         3.00
50%         3.00
75%         4.00
max         5.00
Name: condition, dtype: float64
```

Out[1576]: <AxesSubplot:xlabel='condition', ylabel='price'>



In [1577]:
```
hist('condition')
```



In [ ]:
```

```

In [ ]:
```

```

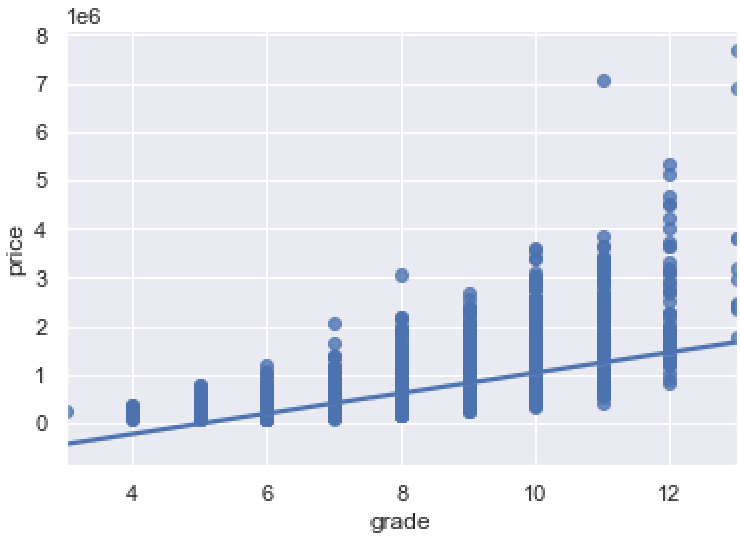
Grade

In [1578]:

```
clean_column('grade')

regplot('grade')
```

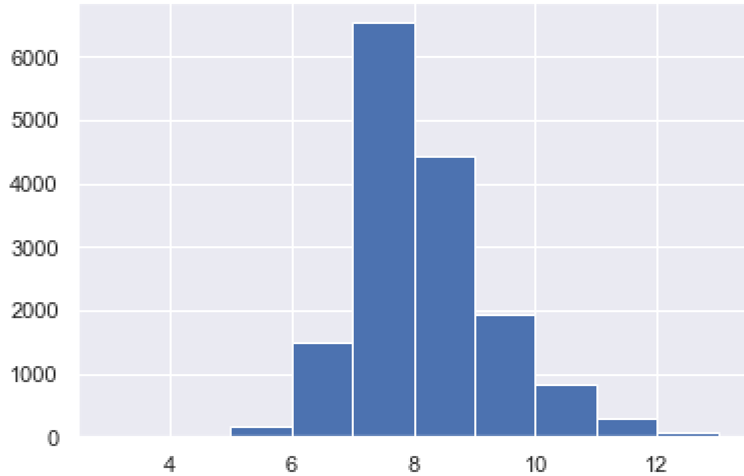
```
Datatype: int64
Total unique itms: 11
Displaying first 10:
[ 7  8 11  9  6  5 10 12  4  3]
Minimum value: 3.  Maximum value: 13
count    15762.00
mean         7.66
std          1.17
min          3.00
25%          7.00
50%          7.00
75%          8.00
max         13.00
Name: grade, dtype: float64
```

Out[1578]: <AxesSubplot:xlabel='grade', ylabel='price'>



In [1579]:

```
hist('grade')
```



In [1580]:
```
1 df[df['grade'] == 13]
```

Out[1580]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
9831200500	2015-03-04	2480000.00	5	3.75	6810	7500	2.50	0.00	0.0(
7237501190	2014-10-10	1780000.00	4	3.25	4890	13402	2.00	0.00	0.0(
1725059316	2014-11-20	2390000.00	4	4.00	6330	13296	2.00	0.00	2.0(
853200010	2014-07-01	3800000.00	5	5.50	7050	42840	1.00	0.00	2.0(
6762700020	2014-10-13	7700000.00	6	8.00	12050	27600	2.50	0.00	3.0(
1068000375	2014-09-23	3200000.00	6	5.00	7100	18200	2.50	0.00	0.0(
9208900037	2014-09-19	6890000.00	6	7.75	9890	31374	2.00	0.00	4.0(
3303850390	2014-12-12	2980000.00	5	5.50	7400	18898	2.00	0.00	3.0(
2426039123	2015-01-30	2420000.00	5	4.75	7880	24250	2.00	0.00	2.0(
4139900180	2015-04-20	2340000.00	4	2.50	4500	35200	1.00	0.00	0.0(
2303900100	2014-09-11	3800000.00	3	4.25	5510	35000	2.00	0.00	4.0(

In [ ]:
```
1
```

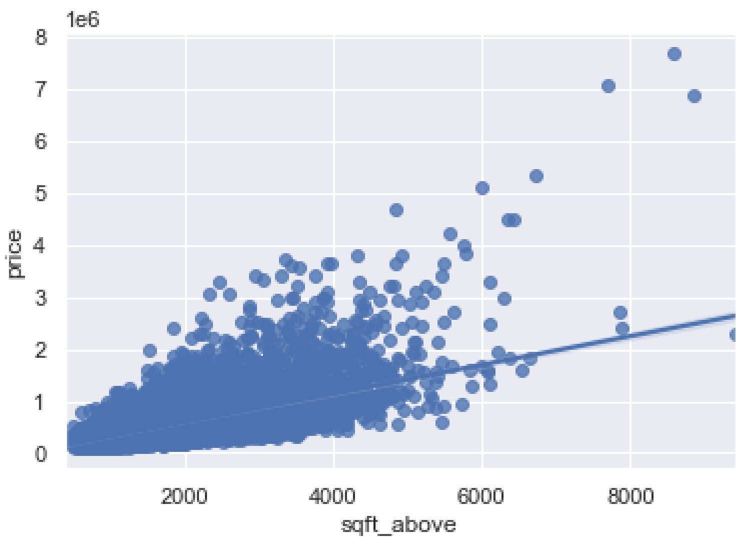
Squarefoot Above

In [1581]:

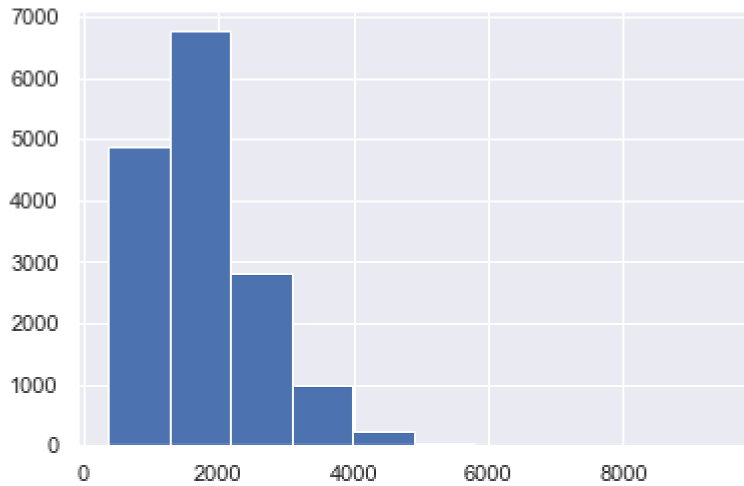
```
clean_column('sqft_above')

regplot('sqft_above')
```

```
Datatype: int64
Total unique itms: 835
Displaying first 10:
[2170 1050 1680 3890 1715 1890  860 1370 1810 1980]
Minimum value: 370.  Maximum value: 9410
count    15762.00
mean      1792.78
std        828.40
min        370.00
25%       1200.00
50%       1570.00
75%       2220.00
max       9410.00
Name: sqft_above, dtype: float64
```

Out[1581]: <AxesSubplot:xlabel='sqft_above', ylabel='price'>



In [1582]:
```
hist('sqft_above')
```



In [ ]:
```

```

In [ ]:
```

```

Squarefoot Basement

In [1583]:
```
clean_column('sqft_basement')

```

```
Datatype: object
Total unique itms: 283
Displaying first 10:
['400.0' '910.0' '0.0' '1530.0' '?' '730.0' '300.0' '970.0' '760.0'
 '720.0']
Minimum value: 0.0.  Maximum value: ?
count     15762
unique      283
top         0.0
freq       9362
Name: sqft_basement, dtype: object
```

Out[1583]: 'sqft_basement'

It seems there are some errors with question marks. Let's take a look.

In [1584]:
```
df[df['sqft_basement'] == '?']
```

Out[1584]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
1321400060	2014-06-27	257500.00	3	2.25	1715	6819	2.00	0.00
16000397	2014-12-05	189000.00	2	1.00	1200	9850	1.00	0.00
7203220400	2014-07-07	861990.00	5	2.75	3595	5639	2.00	0.00
1531000030	2015-03-23	720000.00	4	2.50	3450	39683	2.00	0.00
2525310310	2014-09-16	272500.00	3	1.75	1540	12600	1.00	0.00
1909600046	2014-07-03	445838.00	3	2.50	2250	5692	2.00	0.00

It might be best to go ahead and make a "True" and "False" boolean column for 'has_basement.' We will also change all '?' values to zero (0) and conver the values into floats.

In [ ]:
```

```

In [1585]:
```
# df['sqft_basement'] = df['sqft_basement'].replace(['?'],0.0)

df['sqft_basement'] = df['sqft_living'] - df['sqft_above']

df.loc[df['sqft_basement'] == '?']
```

Out[1585]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gra

In [1586]:
```
df['sqft_basement'] = df['sqft_basement'].astype(float)
```

In [1587]:

```
# run this line if need to rerun and delete 'has_basement'
# del df['has_basement']

has_basement = np.where(df['sqft_basement'] > 0, 1, 0)

df.insert (12, 'has_basement', has_basement)

df.head(5)
```

Out[1587]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
id									
6414100192	2014-12-09	538000.00	3	2.25	2570	7242	2.00	0.00	0.0(
2487200875	2014-12-09	604000.00	4	3.00	1960	5000	1.00	0.00	0.0(
1954400510	2015-02-18	510000.00	3	2.00	1680	8080	1.00	0.00	0.0(
7237550310	2014-05-12	1230000.00	4	4.50	5420	101930	1.00	0.00	0.0(
1321400060	2014-06-27	257500.00	3	2.25	1715	6819	2.00	0.00	0.0(

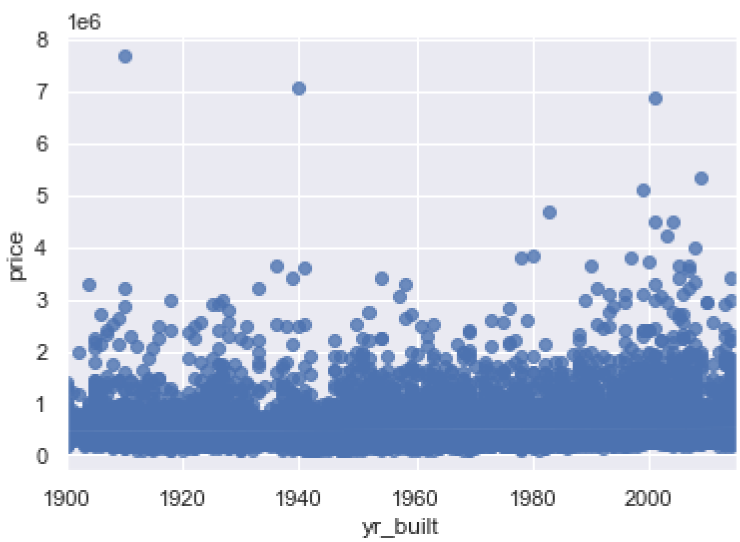
5 rows × 21 columns

Year Built

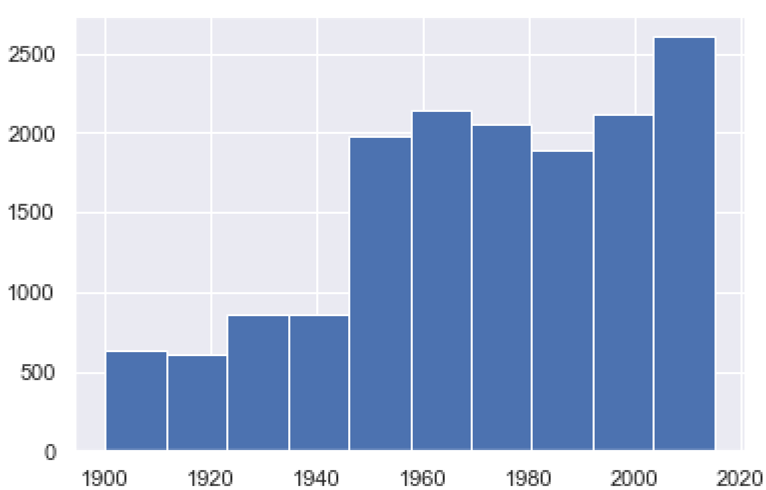
In [1588]:
```
clean_column('yr_built')

regplot('yr_built')
```

```
Datatype: int64
Total unique itms: 116
Displaying first 10:
[1951 1965 1987 2001 1995 1960 2003 1942 1977 1900]
Minimum value: 1900.  Maximum value: 2015
count   15762.00
mean     1971.11
std        29.34
min      1900.00
25%      1952.00
50%      1975.00
75%      1997.00
max      2015.00
Name: yr_built, dtype: float64
```

Out[1588]: <AxesSubplot:xlabel='yr_built', ylabel='price'>



In [1589]:
```
hist('yr_built')
```



We will convert this to age to more easily interpret this feature in our model.

In [1590]:

```
df['age'] = abs(df['yr_built'] - 2015)

df.head()
```

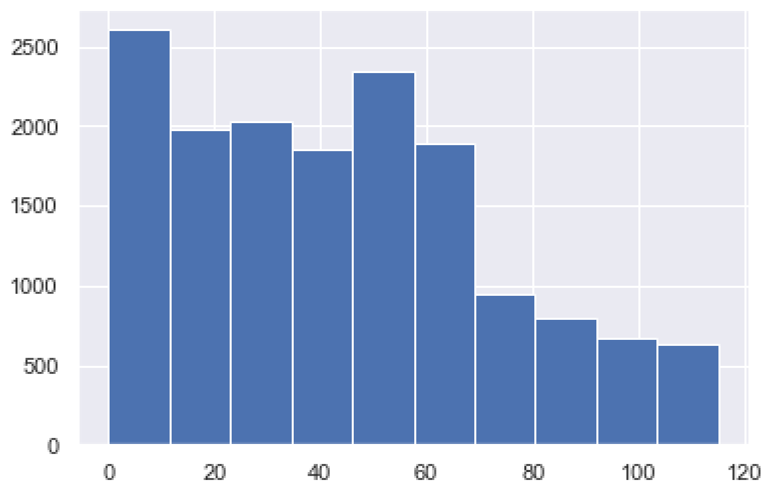
Out[1590]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
id									
6414100192	2014-12-09	538000.00	3	2.25	2570	7242	2.00	0.00	0.0(
2487200875	2014-12-09	604000.00	4	3.00	1960	5000	1.00	0.00	0.0(
1954400510	2015-02-18	510000.00	3	2.00	1680	8080	1.00	0.00	0.0(
7237550310	2014-05-12	1230000.00	4	4.50	5420	101930	1.00	0.00	0.0(
1321400060	2014-06-27	257500.00	3	2.25	1715	6819	2.00	0.00	0.0(

5 rows × 22 columns

In [1591]:
```
clean_column('age')

hist('age')
```

```
Datatype: int64
Total unique itms: 116
Displaying first 10:
[ 64  50  28  14  20  55  12  73  38 115]
Minimum value: 0.  Maximum value: 115
count   15762.00
mean       43.89
std        29.34
min         0.00
25%        18.00
50%        40.00
75%        63.00
max       115.00
Name: age, dtype: float64
```



In [1592]:
```
del df['yr_built']
```

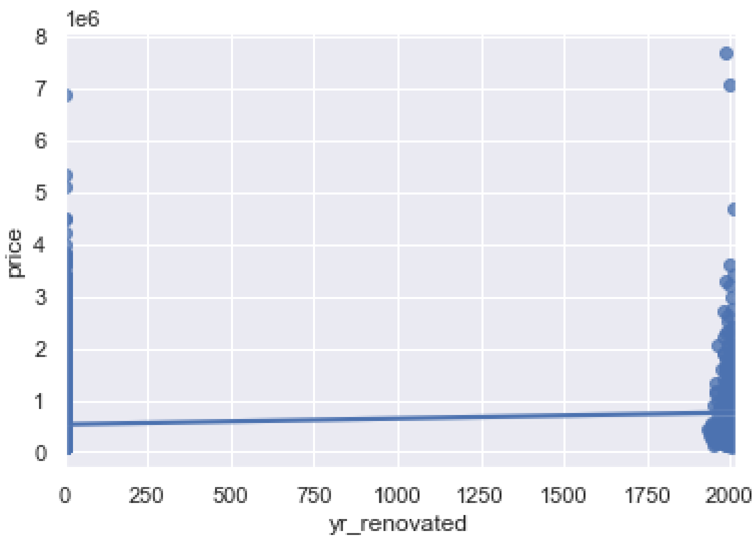
In [ ]:
```

```

Year Renovated

In [1593]:
```
clean_column('yr_renovated')

regplot('yr_renovated')
```

```
Datatype: float64
Total unique itms: 70
Displaying first 10:
[1991.    0. 2002. 2010. 1992. 2013. 1994. 1978. 2005. 2003.]
Minimum value: 0.0.  Maximum value: 2015.0
count   15762.00
mean       82.44
std       397.21
min         0.00
25%         0.00
50%         0.00
75%         0.00
max      2015.00
Name: yr_renovated, dtype: float64
```

Out[1593]: <AxesSubplot:xlabel='yr_renovated', ylabel='price'>



Convert to binary.

In [1594]:

```
renovated = np.where(df['yr_renovated'] > 0, 1, 0)

df['renovated'] = renovated

del df['yr_renovated']

df.head(5)
```

Out[1594]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
id									
6414100192	2014-12-09	538000.00	3	2.25	2570	7242	2.00	0.00	0.0(
2487200875	2014-12-09	604000.00	4	3.00	1960	5000	1.00	0.00	0.0(
1954400510	2015-02-18	510000.00	3	2.00	1680	8080	1.00	0.00	0.0(
7237550310	2014-05-12	1230000.00	4	4.50	5420	101930	1.00	0.00	0.0(
1321400060	2014-06-27	257500.00	3	2.25	1715	6819	2.00	0.00	0.0(

5 rows × 21 columns

In [1595]:

```
df.head()
```

Out[1595]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
id									
6414100192	2014-12-09	538000.00	3	2.25	2570	7242	2.00	0.00	0.0(
2487200875	2014-12-09	604000.00	4	3.00	1960	5000	1.00	0.00	0.0(
1954400510	2015-02-18	510000.00	3	2.00	1680	8080	1.00	0.00	0.0(
7237550310	2014-05-12	1230000.00	4	4.50	5420	101930	1.00	0.00	0.0(
1321400060	2014-06-27	257500.00	3	2.25	1715	6819	2.00	0.00	0.0(

5 rows × 21 columns

In [ ]:

```

```

Zipcode

In [1596]:
```
clean_column('zipcode')
```

```
Datatype: int64
Total unique itms: 70
Displaying first 10:
[98125 98136 98074 98053 98003 98146 98038 98115 98107 98126]
Minimum value: 98001.  Maximum value: 98199
count   15762.00
mean    98077.56
std        53.41
min     98001.00
25%     98033.00
50%     98065.00
75%     98117.00
max     98199.00
Name: zipcode, dtype: float64
```

Out[1596]: 'zipcode'

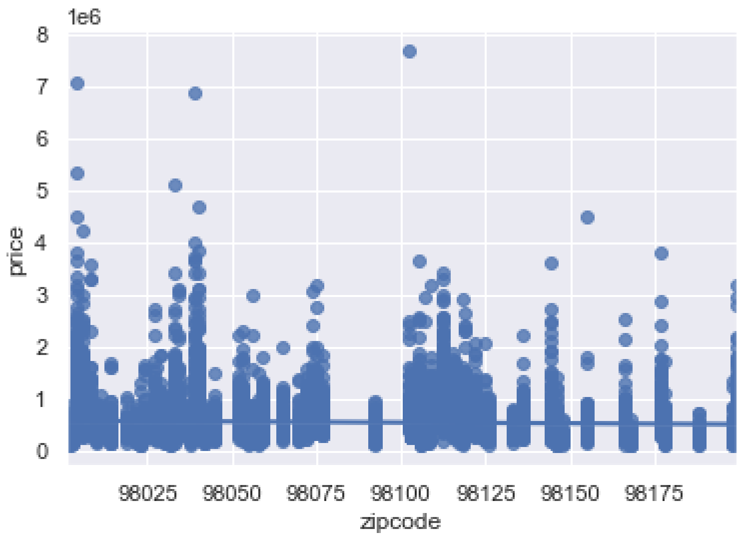
Zipcode should be integer for now, since there will be no decimals. It might be worth considering conversion to string as well further in the project.

In [1597]:
```
df['zipcode'] = df['zipcode'].astype(int)
```

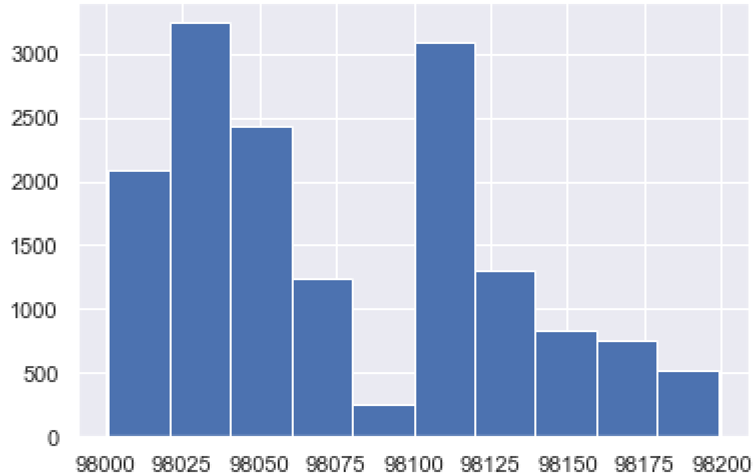
In [1598]:
```
clean_column('zipcode')

regplot('zipcode')
```

```
Datatype: int32
Total unique itms: 70
Displaying first 10:
[98125 98136 98074 98053 98003 98146 98038 98115 98107 98126]
Minimum value: 98001.  Maximum value: 98199
count    15762.00
mean     98077.56
std         53.41
min      98001.00
25%      98033.00
50%      98065.00
75%      98117.00
max      98199.00
Name: zipcode, dtype: float64
```

Out[1598]: <AxesSubplot:xlabel='zipcode', ylabel='price'>



In [1599]:
```
1 hist('zipcode')
```



In [ ]:
```
1
```

In [ ]:
```
1
```

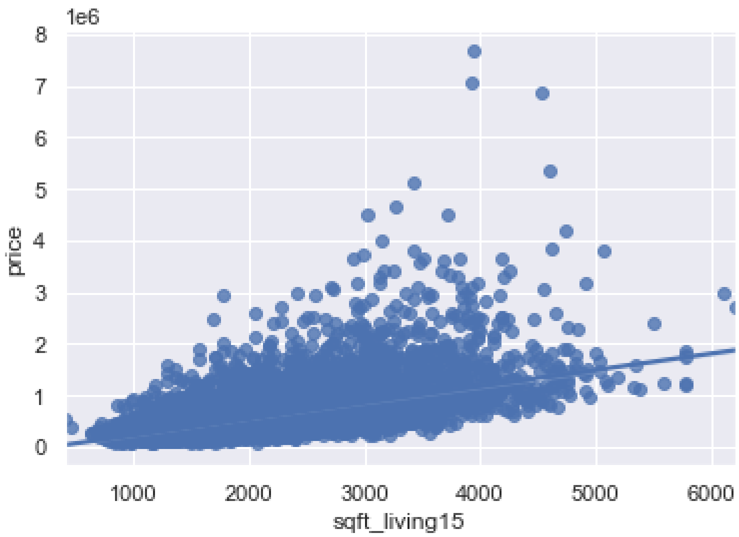
'sqft_living15'

The square footage of interior housing living space for the nearest 15 neighbors

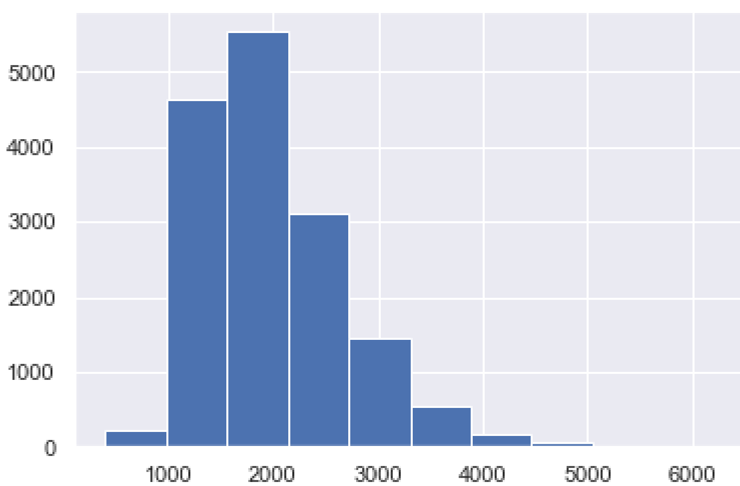
In [1600]:
```
clean_column('sqft_living15')

regplot('sqft_living15')
```

```
Datatype: int64
Total unique itms: 694
Displaying first 10:
[1690 1360 1800 4760 2238 1780 2390 1330 1370 2140]
Minimum value: 399.  Maximum value: 6210
count   15762.00
mean     1990.22
std       684.14
min       399.00
25%      1490.00
50%      1846.00
75%      2370.00
max      6210.00
Name: sqft_living15, dtype: float64
```

Out[1600]: <AxesSubplot:xlabel='sqft_living15', ylabel='price'>



In [1601]:
```
hist('sqft_living15')
```



In [ ]:
```
1
```

In [ ]:
```
1
```

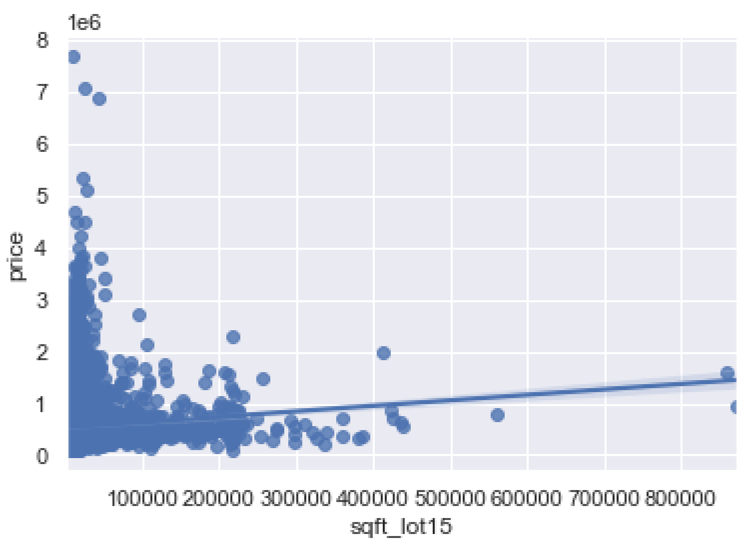
'sqft_lot15'

The square footage of the land lots of the nearest 15 neighbors

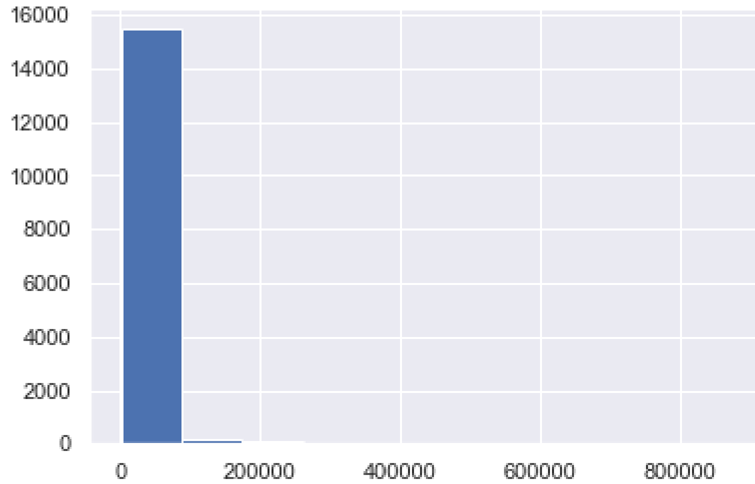
In [1602]:
```
clean_column('sqft_lot15')

regplot('sqft_lot15')
```

```
Datatype: int64
Total unique itms: 7126
Displaying first 10:
[  7639    5000    7503 101930    6819    8113    7570    6000   10208    4850]
Minimum value: 659.  Maximum value: 871200
count     15762.00
mean      12900.42
std       27977.23
min         659.00
25%        5100.00
50%        7620.00
75%       10107.50
max      871200.00
Name: sqft_lot15, dtype: float64
```

Out[1602]: <AxesSubplot:xlabel='sqft_lot15', ylabel='price'>



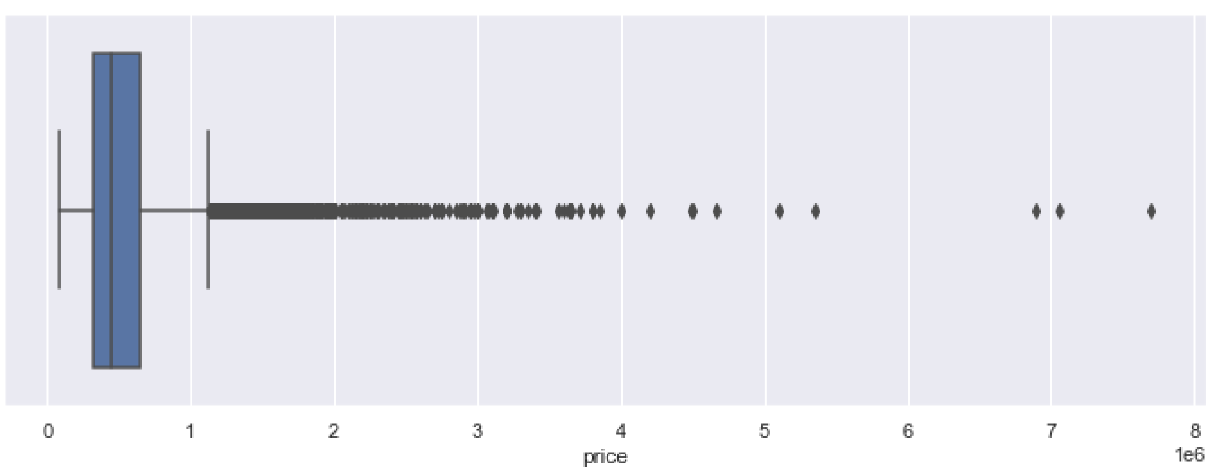
In [1603]:
```
hist('sqft_lot15')
```



Remove Outliers

In [1604]:
```
fig, ax = plt.subplots(figsize=(12, 4))
ax = sns.boxplot(df['price'])
```

```
C:\Users\Johnny\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.
py:36: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```



In [ ]:
```

```

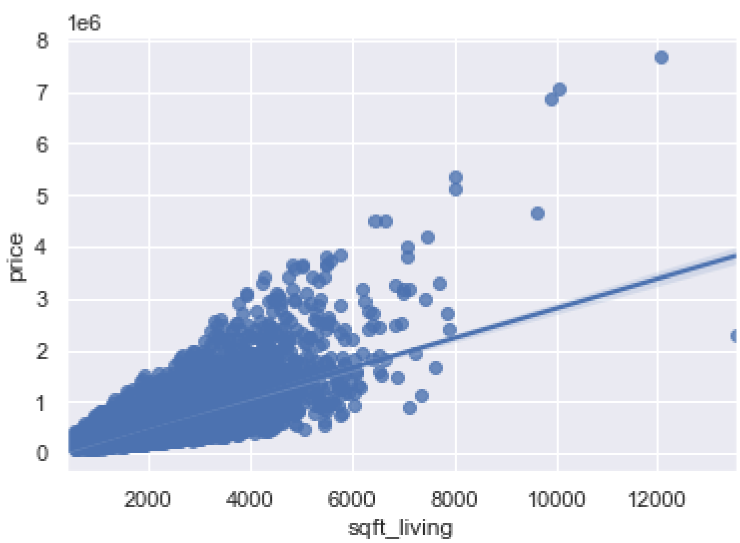
Target Variable (price)

In [1605]:
```
column = 'price'

describe = df.describe()[column]

q1 = describe['25%']
q3 = describe['75%']

iqr = q3 - q1

outlier_index = (df[column] > (q3 + 1.5 * iqr)) | (df[column] < (q1 - 1.5 *

df[outlier_index].shape

```

Out[1605]: (831, 21)

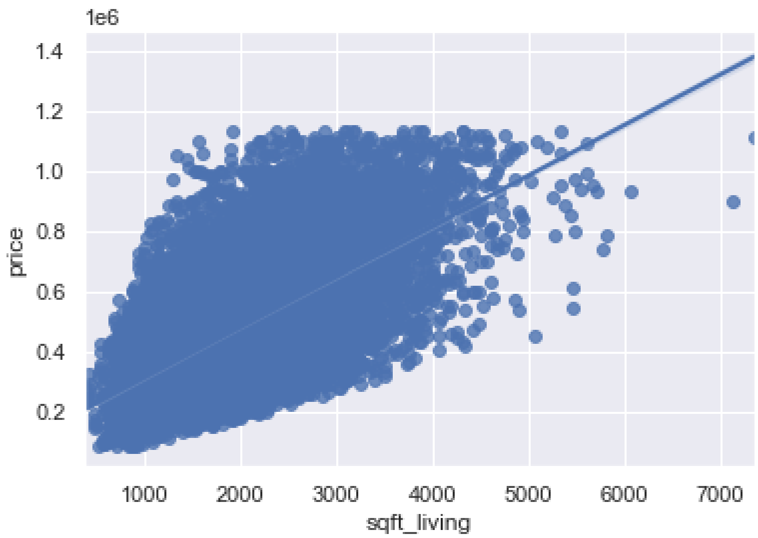
In [1606]:
```
regplot('sqft_living', df=df)
```

Out[1606]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



In [1607]:
```
regplot('sqft_living', df=df[~outlier_index])
```

Out[1607]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



In [1608]:
```
df = df[~outlier_index]
```

In [ ]:
```

```

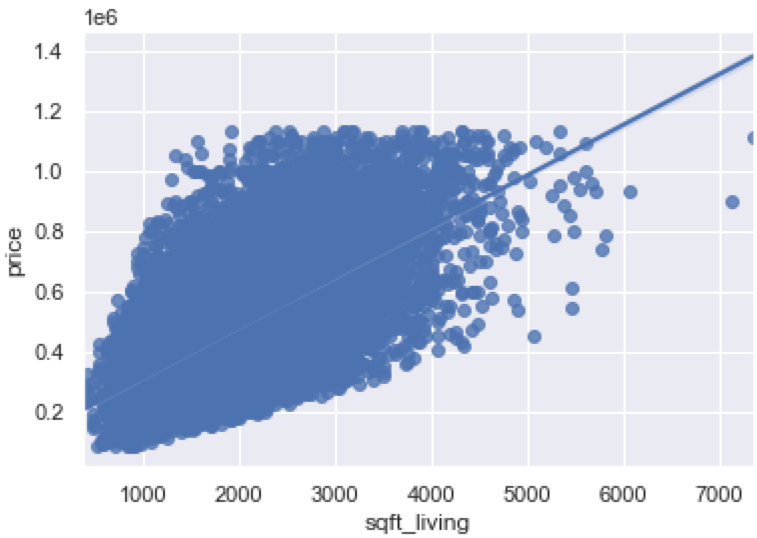
Squarefoot Living

In [1609]:
```
column = 'sqft_living'

describe = df.describe()[column]

q1 = describe['25%']
q3 = describe['75%']

iqr = q3 - q1

outlier_index = (df[column] > (q3 + 1.5 * iqr)) | (df[column] < (q1 - 1.5 *

df[outlier_index].shape

```

Out[1609]: (216, 21)

In [1610]:
```
regplot('sqft_living', df=df)
```

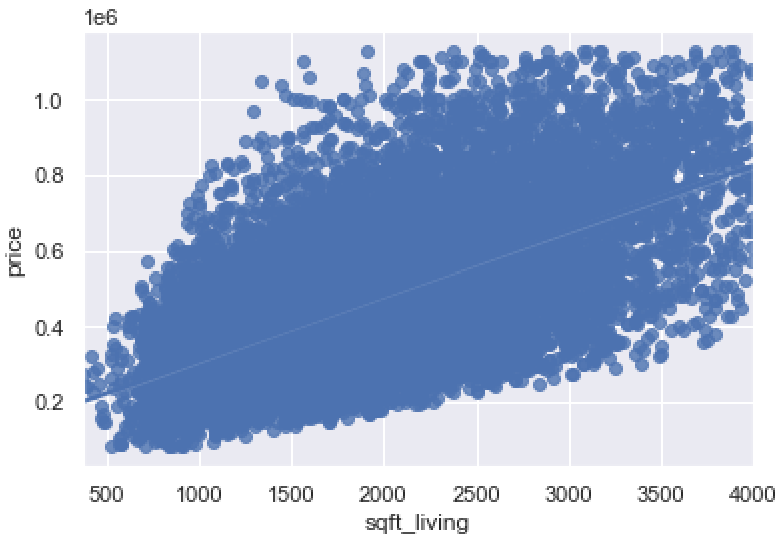
Out[1610]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



In [1611]:

```
regplot('sqft_living', df=df[~outlier_index])
```

Out[1611]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



In [1612]:

```
df = df[~outlier_index]
```

Squarefoot Lot

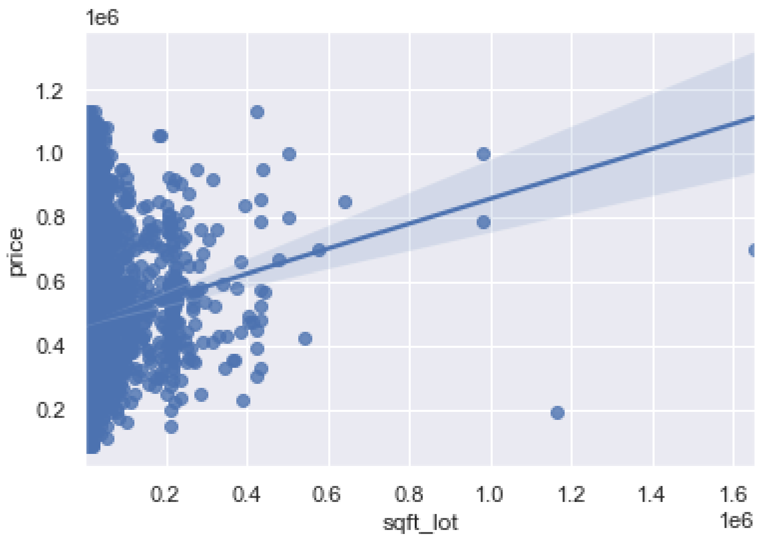
In [1613]:

```
column = 'sqft_lot'

describe = df.describe()[column]

q1 = describe['25%']
q3 = describe['75%']

iqr = q3 - q1

outlier_index = (df[column] > (q3 + 1.5 * iqr)) | (df[column] < (q1 - 1.5 *

df[outlier_index].shape

```

Out[1613]: (1587, 21)

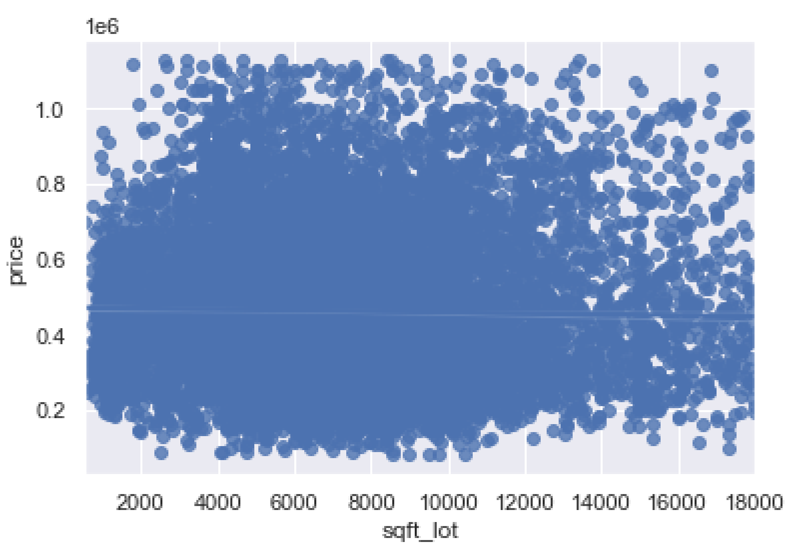
In [1614]:
```
regplot('sqft_lot', df=df)
```

Out[1614]: <AxesSubplot:xlabel='sqft_lot', ylabel='price'>



In [1615]:
```
regplot('sqft_lot', df=df[~outlier_index])
```

Out[1615]: <AxesSubplot:xlabel='sqft_lot', ylabel='price'>



In [1616]:
```
df = df[~outlier_index]
```

In [ ]:
```

```

EXPLORE

Now that we're comfortable that we have quality data, it's time to determine which columns we'll want to analyze for our primary analysis.

First we'll review which columns we have to work with:

Feature Selection

Based on prior anaysis and scrubbing, we'll categorize our columns into three sections:

Continuous variables:

- price
- sqft_living
- sqft_lot
- sqft_above
- sqft_basement
- yr_built
- sqft_living15
- sqft_lot15

Categorical variables - while some of these may appear continuous, their values represent integers and fractions that are more categorical even if they are for a specific count.

- bedrooms
- bathrooms
- floors
- condition
- grade
- waterfront
- renovated
- zipcode
- has_basement

Remove from model:

- date - date of sale could be interesting to analyze if we had a longer time horizon. Home prices could sell for more less based on season, and this could be interesting for further analysis
- lat - will not have a linear relationship

- long - will not have a linear relationship
- view - while we have a range of values, the column description reads "Has been viewed" which should be binary. Seems like there could be an error, further review could make this column eligible for future analysis

In [1617]:
```
df.drop(['date', 'lat', 'long', 'view'],axis=1,inplace=True)
```

In [1618]:
```
df.head()
```

Out[1618]:

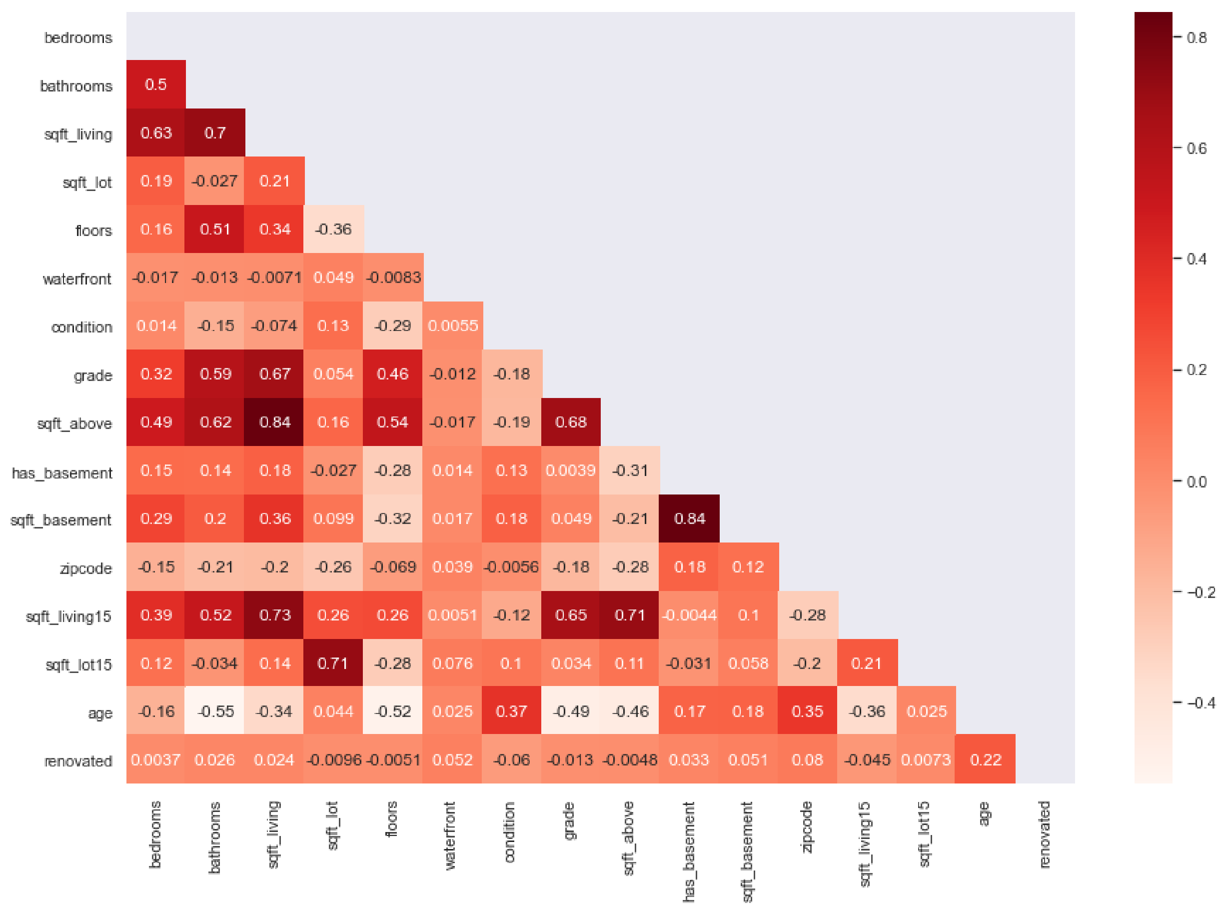
id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	g
6414100192	538000.00	3	2.25	2570	7242	2.00	0.00	3	
2487200875	604000.00	4	3.00	1960	5000	1.00	0.00	5	
1954400510	510000.00	3	2.00	1680	8080	1.00	0.00	3	
1321400060	257500.00	3	2.25	1715	6819	2.00	0.00	3	
2414600126	229500.00	3	1.00	1780	7470	1.00	0.00	3	

Multicollinearity

We will create a heat map to identify multicollinearity.

In [1619]:
```
def heatmap(df_name, figsize=(15,10), cmap='Reds'):
    corr = df_name.drop('price',axis=1).corr()
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    fig, ax = plt.subplots(figsize=figsize)
    sns.heatmap(corr, annot=True, cmap=cmap, mask=mask)
    return fig, ax

heatmap(df)
```

Out[1619]: (<Figure size 1080x720 with 2 Axes>, <AxesSubplot:>)



We will drop the following:

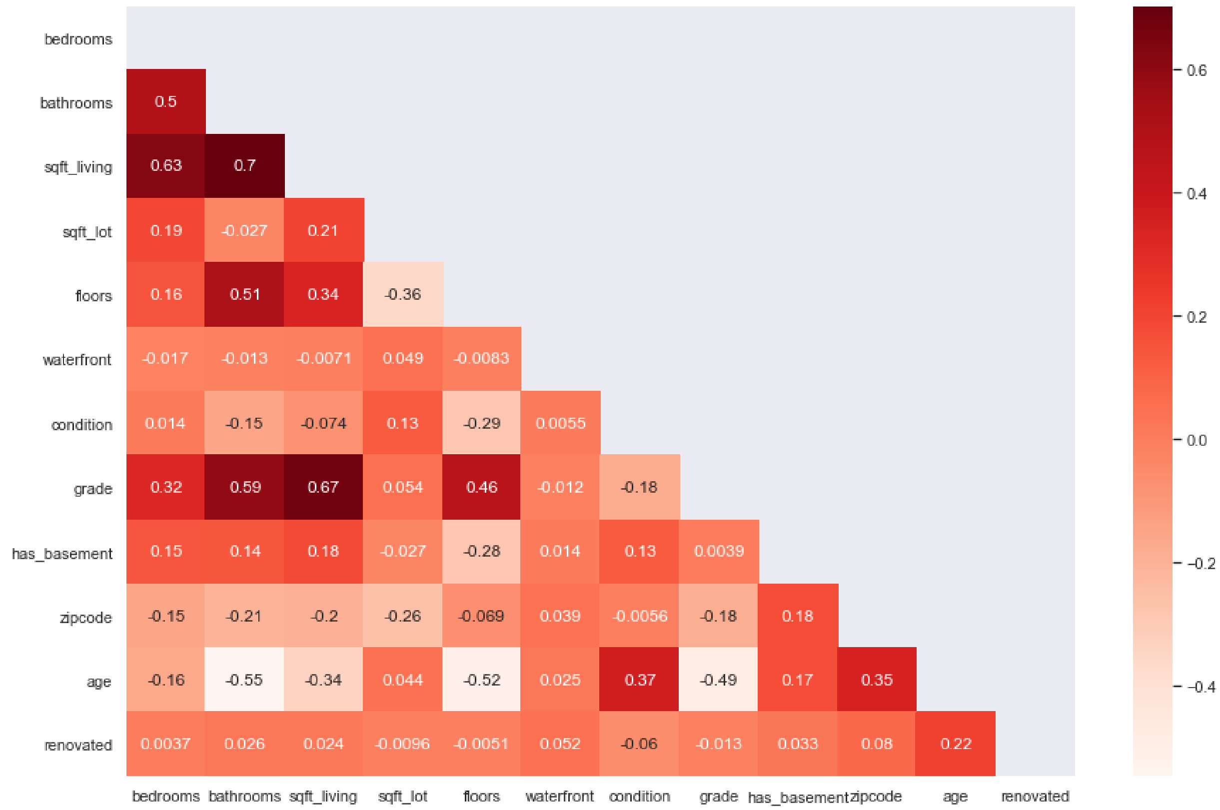
sqft_above + sqft_basement - these are duplicative of sqft_living.

sqft_lot15 and sqft_living15 - these could be more interesting for broader analysis of areas. Since there is high multicolinearity, we can save these for when we look at zip, lat, and long.

In [1620]:
```
del df['sqft_above']
del df['sqft_basement']
del df['sqft_lot15']
del df['sqft_living15']
```

In [1621]:
```
heatmap(df)
```

Out[1621]: (<Figure size 1080x720 with 2 Axes>, <AxesSubplot:>)



Sqft_living, bathrooms, and grade appear to have potential for multicollinearity. This issue should be remedied by encoding grade and bathrooms as categorical variables, which we will do next in the modeling stage.

MODEL

Data Modeling

Describe and justify the process for analyzing or modeling the data.

Questions to consider:

- How did you analyze or model the data?

- How did you iterate on your initial approach to make it better?
- Why are these choices appropriate given the data and the business problem?

In [1622]:

```
df.head()
```

Out[1622]:

id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	g
6414100192	538000.00	3	2.25	2570	7242	2.00	0.00	3	
2487200875	604000.00	4	3.00	1960	5000	1.00	0.00	5	
1954400510	510000.00	3	2.00	1680	8080	1.00	0.00	3	
1321400060	257500.00	3	2.25	1715	6819	2.00	0.00	3	
2414600126	229500.00	3	1.00	1780	7470	1.00	0.00	3	

Installing stats and modeling packages:

In [1623]:

```
!pip install -U fsds
from scipy import stats
from fsds.imports import *

import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
import scipy.stats as stats

import scipy.stats as stats
import statsmodels.api as sms
```

```
Requirement already up-to-date: fsds in c:\users\johnny\anaconda3\envs\learn-
env\lib\site-packages (0.3.2)
Requirement already satisfied, skipping upgrade: seaborn>=0.11.0 in c:\users
\johnny\anaconda3\envs\learn-env\lib\site-packages (from fsds) (0.11.0)
Requirement already satisfied, skipping upgrade: ipywidgets in c:\users\johnn
y\anaconda3\envs\learn-env\lib\site-packages (from fsds) (7.5.1)
Requirement already satisfied, skipping upgrade: pandas>=1.1.0 in c:\users\jo
hnny\anaconda3\envs\learn-env\lib\site-packages (from fsds) (1.1.3)
Requirement already satisfied, skipping upgrade: missingno in c:\users\johnny
\anaconda3\envs\learn-env\lib\site-packages (from fsds) (0.4.2)
Requirement already satisfied, skipping upgrade: scikit-learn>=0.23.1 in c:\u
sers\johnny\anaconda3\envs\learn-env\lib\site-packages (from fsds) (0.23.2)
Requirement already satisfied, skipping upgrade: scipy in c:\users\johnny\ana
conda3\envs\learn-env\lib\site-packages (from fsds) (1.5.0)
Requirement already satisfied, skipping upgrade: IPython in c:\users\johnny\a
naconda3\envs\learn-env\lib\site-packages (from fsds) (7.18.1)
Requirement already satisfied, skipping upgrade: tzlocal in c:\users\johnny\a
naconda3\envs\learn-env\lib\site-packages (from fsds) (2.1)
Requirement already satisfied, skipping upgrade: pyperclip in c:\users\johnny
```

Initial Model

We'll go ahead and define our categorical variables so that we can implement the code into our model function:

In [1624]:
```
categoricals = ['bedrooms',
                'bathrooms',
                'floors',
                'waterfront',
                'condition',
                'grade',
                'has_basement',
                'zipcode',
                'renovated']

```

Function to draw a QQ plot and a homoscedasticity check.

In [1625]:
```
def check_model(model):

    resids = model.resid

    fig,ax = plt.subplots(ncols=2,figsize=(12,5))
    sms.qqplot(resids, stats.distributions.norm, fit=True, line='45',ax=ax[0
    xs = np.linspace(0,1,len(resids))

    y_hat = model.predict(df)
    y = df['price']
    resid = y - y_hat
    plot = plt.scatter(x=y_hat, y=resid)
    plt.axhline(0)

    ax[1].scatter(x=y_hat,y=resid)

    return fig,ax

# check_model(model1)
```

Function to run the model and output summary statistics and graphs.

In [1626]:
```
def make_model(df_name, categoricals=categoricals):

    features = ' + '.join(df.drop('price',axis=1).columns)
    for variable in categoricals:
        features = features.replace(variable, ("C(" + variable + ")"))

    f  = "price~"+features

    model = smf.ols(f, df_name).fit()
    display(model.summary())

    fig,ax = check_model(model)
    plt.show()

    return model

model1 = make_model(df)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.831
Model:	OLS	**Adj. R-squared:**	0.829
Method:	Least Squares	**F-statistic:**	528.1
Date:	Thu, 22 Apr 2021	**Prob (F-statistic):**	0.00
Time:	15:44:51	**Log-Likelihood:**	-1.6728e+05
No. Observations:	13128	**AIC:**	3.348e+05
Df Residuals:	13006	**BIC:**	3.357e+05
Df Model:	121		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-9.876e+04	1.01e+05	-0.982	0.326	-2.96e+05	9.83e+04

Our first model has a fairly strong R-squared at 0.831. The QQ plot indicates that there might be some outliers that we could remove to further refine our model. The homoscedasticity graph also shows some outliers, but the graph has a noticeable cone shape indicating we are mostly on track with our current refinement of the overall dataset.

Refining based on P-values

Next, we'll want to look at the features that have a P-value greater than 0.05. Removing these features will help us isolate the most statistically significant variables of our model.

In [1627]:
```
model1.pvalues

pvals = model1.pvalues

pvals[pvals > 0.05]
# pvals[pvals > 0.05].index
```

Out[1627]:
```
Intercept               0.33
C(bedrooms)[T.5]        0.69
C(bedrooms)[T.6]        0.35
C(bedrooms)[T.8]        0.15
C(bedrooms)[T.9]        0.47
C(bedrooms)[T.10]       0.15
C(bathrooms)[T.0.75]    0.14
C(bathrooms)[T.1.0]     0.26
C(bathrooms)[T.1.25]    0.90
C(bathrooms)[T.1.5]     0.26
C(bathrooms)[T.1.75]    0.20
C(bathrooms)[T.2.0]     0.25
C(bathrooms)[T.2.25]    0.15
C(bathrooms)[T.2.5]     0.12
C(bathrooms)[T.2.75]    0.08
C(bathrooms)[T.3.0]     0.11
C(bathrooms)[T.3.25]    0.06
C(bathrooms)[T.4.5]     0.38
C(bathrooms)[T.4.75]    0.63
C(bathrooms)[T.5.0]     0.39
C(bathrooms)[T.5.25]    0.34
C(bathrooms)[T.5.75]    0.94
C(floors)[T.2.0]        0.07
C(floors)[T.3.5]        0.10
C(grade)[T.4]           0.22
C(grade)[T.5]           0.11
C(grade)[T.6]           0.13
C(grade)[T.7]           0.24
C(grade)[T.8]           0.55
C(grade)[T.9]           0.70
C(grade)[T.10]          0.40
C(zipcode)[T.98002]     0.25
C(zipcode)[T.98003]     0.39
C(zipcode)[T.98022]     0.61
C(zipcode)[T.98030]     0.55
C(zipcode)[T.98031]     0.05
C(zipcode)[T.98032]     0.93
C(zipcode)[T.98042]     0.10
dtype: float64
```

It seems that certain bedroom numbers don't have a significant effect. Bathrooms have very little effect. 1.5 and 3.5 floors might not have an effect, likely due to low representaion in dataset. Some conditions seems important, grade seems negligible, and a 12 of the 69 zip codes are not significant.

We will remove the following features since more than half of their representative categoricals are not significant:

- bedrooms
- bathrooms
- grade

In [ ]:

```

```

In [1628]:

```
df = df.drop(['bedrooms', 'bathrooms', 'grade'], axis=1)
```

In [1629]:

```
df.head()
```

Out[1629]:

id	price	sqft_living	sqft_lot	floors	waterfront	condition	has_basement	zipcode
6414100192	538000.00	2570	7242	2.00	0.00	3	1	98125
2487200875	604000.00	1960	5000	1.00	0.00	5	1	98136
1954400510	510000.00	1680	8080	1.00	0.00	3	0	98074
1321400060	257500.00	1715	6819	2.00	0.00	3	0	98003
2414600126	229500.00	1780	7470	1.00	0.00	3	1	98146

In [1630]:

```
categoricals = [
#                  'bedrooms',
#                  'bathrooms',
                'floors',
                'waterfront',
                'condition',
#                  'grade',
                'has_basement',
                'zipcode',
                'renovated'
                ]
```

In [1631]:
```
model1 = make_model(df, categoricals)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.799
Model:	OLS	**Adj. R-squared:**	0.798
Method:	Least Squares	**F-statistic:**	617.7
Date:	Thu, 22 Apr 2021	**Prob (F-statistic):**	0.00
Time:	15:44:54	**Log-Likelihood:**	-1.6841e+05
No. Observations:	13128	**AIC:**	3.370e+05
Df Residuals:	13043	**BIC:**	3.376e+05
Df Model:	84		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-1.882e+05	2.64e+04	-7.127	0.000	-2.4e+05	-1.36e+05

In [1632]:
```
model1.pvalues

pvals = model1.pvalues

pvals[pvals > 0.05]
# pvals[pvals > 0.05].index
```

Out[1632]:
```
C(floors)[T.2.5]      0.92
C(floors)[T.3.5]      0.32
C(zipcode)[T.98002]   0.74
C(zipcode)[T.98022]   0.32
C(zipcode)[T.98023]   1.00
C(zipcode)[T.98030]   0.22
C(zipcode)[T.98032]   0.35
C(zipcode)[T.98042]   0.14
C(zipcode)[T.98092]   0.54
dtype: float64
```

In [1633]:
```
pd.set_option('display.float_format', lambda x: '%.10f' % x)

type(model1.pvalues)
```

Out[1633]:
```
pandas.core.series.Series
```

In [1634]:

```
dfp = model1.pvalues.to_frame()

dfp.sort_values(by=[0])

# dfp
```

Out[1634]:

	0
C(zipcode)[T.98103]	0.0000000000
sqft_living	0.0000000000
C(zipcode)[T.98105]	0.0000000000
C(zipcode)[T.98199]	0.0000000000
C(zipcode)[T.98117]	0.0000000000
C(zipcode)[T.98107]	0.0000000000
C(zipcode)[T.98116]	0.0000000000
C(zipcode)[T.98033]	0.0000000000
C(zipcode)[T.98004]	0.0000000000
C(zipcode)[T.98115]	0.0000000000
C(zipcode)[T.98040]	0.0000000000
C(zipcode)[T.98119]	0.0000000000

In [1635]:

```
pvals = model1.pvalues
pvals_list = pvals.sort_values(ascending=True)

pvals_df = pvals_list.to_frame()

# type(model1.params)
pd.options.display.max_rows = 999
pvals_df = pvals_df.reset_index()
pvals_df = pvals_df.rename(columns={'index': 'Variable', 0: 'P_Value'})

# coeff_df['Dollar Impact'] = coeff_df['Dollar Impact'].apply(lambda x: "${:
# coeff_df['Dollar Impact'] = coeff_df['Dollar Impact'].apply(lambda x: "{:,

pvals_df[~pvals_df['Variable'].str.contains("zipcode")]

# coeff_df[~coeff_df['Variable'].str.contains("zipcode")].to_csv('data/coeff

# pvals_df
```

Out[1635]:

	Variable	P_Value
1	sqft_living	0.0000000000
35	C(waterfront)[T.1.0]	0.0000000000
48	C(has_basement)[T.1]	0.0000000000
49	sqft_lot	0.0000000000
51	C(renovated)[T.1]	0.0000000000
54	Intercept	0.0000000000
55	C(condition)[T.5]	0.0000000000
58	C(condition)[T.4]	0.0000000085
62	C(condition)[T.3]	0.0000003648
66	C(floors)[T.2.0]	0.0000224120
68	C(floors)[T.3.0]	0.0001514949
69	age	0.0004193930
70	C(condition)[T.2]	0.0011577263
71	C(floors)[T.1.5]	0.0044800397
78	C(floors)[T.3.5]	0.3210375106
83	C(floors)[T.2.5]	0.9151773929

After running our p-value check again, some zip codes are still insignificant, but not enough to remove zip codes from the model. The 2.5 and 3.5 floors are insignificant, but that is likely due to half-floors having little representation in our dataset.

In [1636]:

```
# df = df.drop(['floors'], axis=1)
```

In [1637]:

```
# categoricals = [
# #                 'bedrooms',
# #                 'bathrooms',
# #                 'floors',
#                 'waterfront',
#                 'condition',
# #                 'grade',
#                 'has_basement',
#                 'zipcode',
#                 'renovated'
#                 ]
```

In [1638]:

```
# model1 = make_model(df, categoricals)
# model1.summary()
```

In [1639]:
```
coeffs = model1.params
coeffs_list = coeffs.sort_values(ascending=False).round(2)

coeff_df = coeffs_list.to_frame()

# type(model1.params)
pd.options.display.max_rows = 999
coeff_df = coeff_df.reset_index()
coeff_df = coeff_df.rename(columns={'index': 'Variable', 0: 'Dollar Impact'}

# coeff_df['Dollar Impact'] = coeff_df['Dollar Impact'].apply(lambda x: "${:
coeff_df['Dollar Impact'] = coeff_df['Dollar Impact'].apply(lambda x: "{:,}"

coeff_df[~coeff_df['Variable'].str.contains("zipcode")]

# coeff_df[~coeff_df['Variable'].str.contains("zipcode")].to_csv('data/coeff
```

Out[1639]:

	Variable	Dollar Impact
17	C(waterfront)[T.1.0]	337,780.33
35	C(condition)[T.5]	174,135.47
41	C(condition)[T.4]	145,846.97
45	C(condition)[T.3]	128,815.16
53	C(condition)[T.2]	87,360.03
63	C(renovated)[T.1]	42,260.43
69	C(floors)[T.2.0]	11,448.94
73	C(floors)[T.1.5]	9,218.91
75	C(floors)[T.2.5]	1,225.86
76	sqft_living	159.48
78	sqft_lot	3.45
79	age	-168.92
81	C(floors)[T.3.0]	-22,470.55
82	C(has_basement)[T.1]	-23,634.14
83	C(floors)[T.3.5]	-40,433.65
84	Intercept	-188,190.8

INTERPRET

Before looking at zipcode, let's take a look at our feature coefficients, which represent price impact.

Waterfront is the most impactful, adding $338k to price.

Condition lines up with our expectations. The greater the condition, the more valuable the home. Improving the condition from 1 to 5 would add an estimated $174,135 to a home owner's value.

Renovated homes seem to fetch a larger price of approximately $42,260, which aligns with expectations.

Floors is a bit counterintuitive. While 2 floors seems to increase the value by $11.5k, a third floor decreases value by $22.5k, 3.5 floors decreases by $40.5k. Considering the cost of adding an additional floor would likely be much more expensive than these coefficients, this might indicate that expanding the square footage of a home within floors that already exist might be a more sensible investment.

Sqft_living gives us an estimated value of $159 for every additional square foot of space.

On the surface, sqft_lot looks like it has a relatively lower impact on price. However, it is still relevant when comparing properties with significant differences in size. One acre is 43,560 square feet. Our model predicts that with a $3.45 impact to price for every square foot, an additional acre would add $150,282 to the value of two otherwise identical properties.

Age doesn't seem to have a great impact. Despite having a P-value greater than 0.05, a house will lose $168 in value every year. Even in the case of our oldest houses, age can only have a maximum price impact of $19,425.

Perhaps counterintuitively, the presence of a basement decreases the value of a home by $23,634. This might require further examination.

In [1640]:

```
print('Most valuable zip codes:')
print(coeff_df[coeff_df['Variable'].str.contains("zipcode")].head(5))
print('Least valuable zip codes:')
print(coeff_df[coeff_df['Variable'].str.contains("zipcode")].tail(5))
```

```
Most valuable zip codes:
              Variable Dollar Impact
0  C(zipcode)[T.98039]    628,000.93
1  C(zipcode)[T.98004]    553,256.56
2  C(zipcode)[T.98112]     494,473.0
3  C(zipcode)[T.98119]    481,396.09
4  C(zipcode)[T.98109]    473,148.05
Least valuable zip codes:
               Variable Dollar Impact
71  C(zipcode)[T.98022]     10,614.08
72  C(zipcode)[T.98032]      10,500.9
74  C(zipcode)[T.98002]      3,273.34
77  C(zipcode)[T.98023]         25.98
80  C(zipcode)[T.98092]     -5,509.67
```

Depending on the location, zip codes can have the most dramatic impact on price. The most valuable zip codes are those closest to the metropolitan city center (Seattle, Bellevue, and Mercer Island). The impact on price in the top 5 zip codes is an estimated $473-628k.

Other than the least valuable zip code, our model functions in a way that doesn't subtract estimated value from homes. The bottom 5 zip codes are located in Kent, near the southern end of King County. While not the furthest from the city center, they are significantly further than our most valuable zip codes.

Model Evaluation

Our model has a semi-strong fit with an adjusted R-squared of 0.798. This means it has a predictive power of roughly 79.8%.

Additional steps could be taken to improve predictive power. Standardization and logistic normalization would theoretically improve R-squared and allow us to make more accurate predictions. We did not incorporate these processes into our model because we were more interested in the practical recommendations it could provide, and inferences are difficult to interpret after normalization.

There are likely improvements that we could make to hone in on accuracy. For our residential clients interest in improving the value of their homes, a 79.8% confidence level seems strong enough to make at least some base line recommendations.

CONCLUSIONS & RECOMMENDATIONS

Our model generated some interesting insights about what drives price in the King County housing market. Here are our major takeaways about the most influential factors in determining a house's price:

Insights

- Location is the most prized quality of a property. Certain zip codes are highly sought after. The top 5 most valuable zip codes will influence property value by an average of $473k-$628k. These zip codes are generally closer to the metropolitan area. Homes located further from the city to the south are less valuable.
- Similar to location, waterfront properties are also much more more valuable and add an average $337k to property value.
- One might assume that additional bedrooms and bathrooms are more valuable. However, according to our model, what actually drives value is total living area square footage. Understanding this, we can intuitively assume that with additional square footage comes additional bedrooms and bathrooms (on average), but our model does not see the bed/bath count as significant.
- The home condition also has a significant impact on price. Before analysis, we assumed that King County's 'Grade' system might behave similarly, but our model determine that the grade system was not a driver of price.

Recommendations to Home Owners

Many of the insights generated by analyzing our model did not lead to practical recommendations for home owners. It isn't exactly practical or possible in most cases to uproot a home and move it to a new area or by the water. But we did notice two key ways that an owner can improve their value:

- Adding square footage through home construction is the most practical recommendation we can offer to improve value. Each additional square foot of living space adds an estimated $159.48 in home value. Adding a second floor gives a small bonus and adding a basement

gives a small penalty. However, when factoring in the added square footage of projects like these, the penalties will most likely be absorbed by the added value.
- Renovating also gives a noticeable bump to price, especially if that renovation improves the condition. Home owners should maintain the condition of their home, or it will decrease in value.

Further Analysis and Modeling

The goal of this project was to develop a very general understanding of the most influential factors in property value. Given more time for data review, we might be able to implement the 'view' feature if we can get a better understanding of what it represents. Sqft_living15, sqft_lot15, and Year Renovated might be interesting to explore. Lat and long can be used to heatmap our dataset to visualize home values on a map of King County.

We could implement standardization and normalization to improve our model's predictive quality. We would also like to implement a train / test split for similar purposes.

It might be helpful to build dynamic splitting of our data. For example, how specifically could the owner of a 2 story, 4 bedroom house in Bellevue improve their home value? Would the coefficients of our features change if we ran our model using only houses that matched that criteria? Dynamic splitting could be useful for generating tailored recommendations to clients who might be willing to pay a premium for such services.

VISUALS

Zipcode Graph

In [1641]:

```
coeffs = model1.params
coeffs_list = coeffs.sort_values(ascending=False).round(2)

coeff_df = coeffs_list.to_frame()

# type(model1.params)
pd.options.display.max_rows = 999
coeff_df = coeff_df.reset_index()
coeff_df = coeff_df.rename(columns={'index': 'Variable', 0: 'Dollar Impact'}

# coeff_df[coeff_df['Variable'].str.contains("zipcode")].head()
```

In [1642]:
```
zip_df = coeff_df[coeff_df['Variable'].str.contains("zipcode")]
zip_df.head()
```

Out[1642]:

	Variable	Dollar Impact
0	C(zipcode)[T.98039]	628000.9300000001
1	C(zipcode)[T.98004]	553256.5600000001
2	C(zipcode)[T.98112]	494473.0000000000
3	C(zipcode)[T.98119]	481396.0900000000
4	C(zipcode)[T.98109]	473148.0500000000

In [1643]:
```
zipcodes = []

for row in zip_df['Variable']:
    old = row
    old = old.replace("C(zipcode)[T.", "")
    old = old.replace("]", "")
    zipcodes.append(old)

zip_df['Zip Code'] = zipcodes

zip_df.head()
```

```
<ipython-input-1643-1481b7f7e852>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  zip_df['Zip Code'] = zipcodes
```

Out[1643]:

	Variable	Dollar Impact	Zip Code
0	C(zipcode)[T.98039]	628000.9300000001	98039
1	C(zipcode)[T.98004]	553256.5600000001	98004
2	C(zipcode)[T.98112]	494473.0000000000	98112
3	C(zipcode)[T.98119]	481396.0900000000	98119
4	C(zipcode)[T.98109]	473148.0500000000	98109

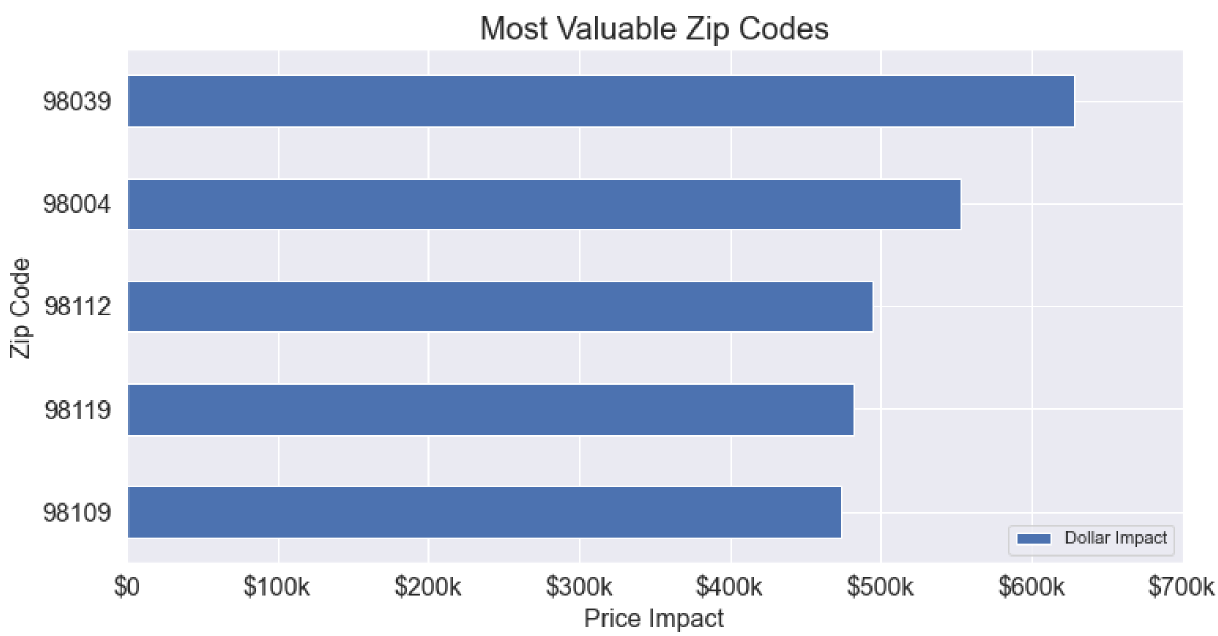
In [1644]:

```
zip_df_top5 = zip_df.head()

zip_df_bottom5 = zip_df.tail()

zip_df['Zip Code'] = zip_df['Zip Code'].astype(int)
zip_df['Dollar Impact'] = zip_df['Dollar Impact'].astype(float)

print(zip_df['Zip Code'].describe())
```

```
count       69.0000000000
mean     98078.4057971015
std         56.2707004631
min      98002.0000000000
25%      98030.0000000000
50%      98070.0000000000
75%      98118.0000000000
max      98199.0000000000
Name: Zip Code, dtype: float64

<ipython-input-1644-a822e190af78>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  zip_df['Zip Code'] = zip_df['Zip Code'].astype(int)
<ipython-input-1644-a822e190af78>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  zip_df['Dollar Impact'] = zip_df['Dollar Impact'].astype(float)
```

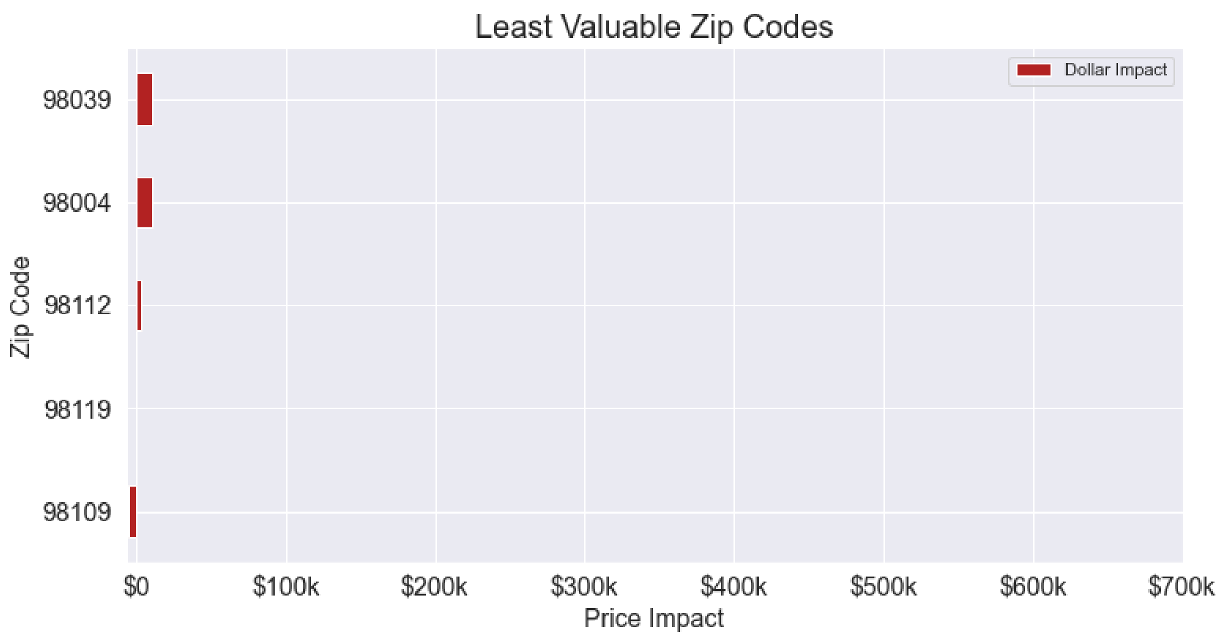
In [1645]:

```
import pandas as pd
import matplotlib.pyplot as plt

data = zip_df_top5
data.plot(x="Zip Code", y="Dollar Impact", kind="barh",figsize=(12, 6))

# plt.legend((["Film Budget ($)", "Box Office Revenue ($)"]))
ax = plt.gca()
ax.set_xticks([0, 100000, 200000, 300000, 400000, 500000, 600000, 700000])
ax.set_xticklabels(['$0', '$100k', '$200k', '$300k', '$400k', '$500k', '$600
ax.set_yticklabels(zip_df_top5['Zip Code'], fontsize=16)
ax.set_title('Most Valuable Zip Codes', fontsize=20)
plt.ylabel('Zip Code', fontsize = 16)
plt.xlabel('Price Impact',fontsize = 16)

ax.invert_yaxis()

plt.savefig('images/top_zips.png')

plt.show()
```



In [1646]:

```
import pandas as pd
import matplotlib.pyplot as plt

data = zip_df_bottom5
data.plot(x="Zip Code", y="Dollar Impact", kind="barh",figsize=(12, 6), colo

# plt.legend((["Film Budget ($)", "Box Office Revenue ($)"]))
ax = plt.gca()
ax.set_xticks([0, 100000, 200000, 300000, 400000, 500000, 600000, 700000])
ax.set_xticklabels(['$0', '$100k', '$200k', '$300k', '$400k', '$500k', '$600
ax.set_yticklabels(zip_df_top5['Zip Code'], fontsize=16)
ax.set_title('Least Valuable Zip Codes', fontsize=20)
plt.ylabel('Zip Code', fontsize = 16)
plt.xlabel('Price Impact',fontsize = 16)

ax.invert_yaxis()

plt.savefig('images/bottom_zips.png')

plt.show()
```



Renovation & Condition Improvements

In [1647]:
```
coeff_df[~coeff_df['Variable'].str.contains("zipcode")]
```

Out[1647]:

	Variable	Dollar Impact
17	C(waterfront)[T.1.0]	337780.3300000000
35	C(condition)[T.5]	174135.4700000000
41	C(condition)[T.4]	145846.9700000000
45	C(condition)[T.3]	128815.1600000000
53	C(condition)[T.2]	87360.0300000000
63	C(renovated)[T.1]	42260.4300000000
69	C(floors)[T.2.0]	11448.9400000000
73	C(floors)[T.1.5]	9218.9100000000
75	C(floors)[T.2.5]	1225.8600000000
76	sqft_living	159.4800000000
78	sqft_lot	3.4500000000
79	age	-168.9200000000
81	C(floors)[T.3.0]	-22470.5500000000
82	C(has_basement)[T.1]	-23634.1400000000
83	C(floors)[T.3.5]	-40433.6500000000
84	Intercept	-188190.8000000000

In [1648]:
```
labels = ['Reonvated Bonus', 'Condition 1 to 2', 'Condition 2 to 3', 'Condit

labels
```

Out[1648]:
```
['Reonvated Bonus',
 'Condition 1 to 2',
 'Condition 2 to 3',
 'Condition 3 to 4',
 'Condition 4 to 5']
```

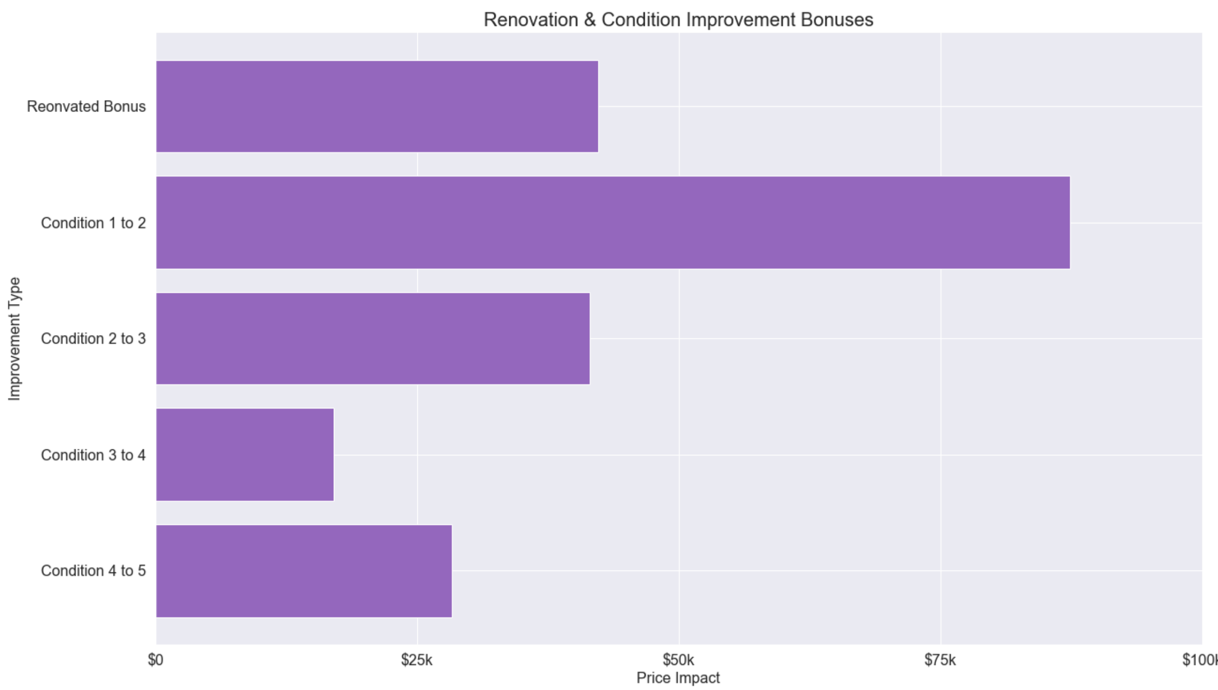
In [1649]:
```
renovated = 42260.43
onetwo = 87360.03
twothree = 128815.16 - 87360.03
threefour = 145846.97 - 128815.16
fourfive = 174135.47 - 145846.97

print(twothree)
print(threefour)
print(fourfive)
```

```
41455.130000000005
17031.809999999998
28288.5
```

In [1650]:
```
values = [42260.43, 87360.03, 41455.13, 17031.80, 28288.5]
```

In [1651]:
```
fig = plt.figure(figsize=(20, 12))
ax = plt.gca()

ax.barh(labels,values, color='tab:purple')

ax.set_title('Renovation & Condition Improvement Bonuses', fontsize=20)
plt.ylabel('Improvement Type', fontsize=16)
plt.xlabel('Price Impact', fontsize=16)

ax.set_xticks([0, 25000, 50000, 75000, 100000])
ax.set_xticklabels(['$0', '$25k', '$50k', '$75k', '$100k'], fontsize=16)
ax.set_yticklabels(labels, fontsize=16)

ax.invert_yaxis()

plt.savefig('images/renovation.png')

plt.show()
```

```
<ipython-input-1651-50a986fac75a>:12: UserWarning: FixedFormatter should only b
e used together with FixedLocator
  ax.set_yticklabels(labels, fontsize=16)
```



Home Addition Bonuses

In [1652]:
```
coeff_df[~coeff_df['Variable'].str.contains("zipcode")]
```

Out[1652]:

	Variable	Dollar Impact
17	C(waterfront)[T.1.0]	337780.3300000000
35	C(condition)[T.5]	174135.4700000000
41	C(condition)[T.4]	145846.9700000000
45	C(condition)[T.3]	128815.1600000000
53	C(condition)[T.2]	87360.0300000000
63	C(renovated)[T.1]	42260.4300000000
69	C(floors)[T.2.0]	11448.9400000000
73	C(floors)[T.1.5]	9218.9100000000
75	C(floors)[T.2.5]	1225.8600000000
76	sqft_living	159.4800000000
78	sqft_lot	3.4500000000
79	age	-168.9200000000
81	C(floors)[T.3.0]	-22470.5500000000
82	C(has_basement)[T.1]	-23634.1400000000
83	C(floors)[T.3.5]	-40433.6500000000
84	Intercept	-188190.8000000000

In [1653]:
```
labels = ['500 Sqft', '1000 Sqft', 'Second Floor', 'Finished Basement',]

labels
```

Out[1653]:
```
['500 Sqft', '1000 Sqft', 'Second Floor', 'Finished Basement']
```

For this graph, we we will be making recommendations to home owners with a one story house with an unfinished basement. The 'Finished Basement' idea is a bit flawed, since we cannot tell from our data whether or not a basement is finished or unfinished. But we will assume that this add-on will toggle 'has_basement' from 0 to 1 and add the additional living space to 'sqft_living.'

In [1654]:
```
f1_df = df[(df['floors'] == 1.00) & (df['has_basement'] == 0)]

f1_df['sqft_living'].describe()
```

Out[1654]:
```
count   3315.0000000000
mean    1284.1710407240
std      402.6186829600
min      370.0000000000
25%      990.0000000000
50%     1240.0000000000
75%     1510.0000000000
max     3430.0000000000
Name: sqft_living, dtype: float64
```

The median sqft_living for a 1 floor house that currently has no basement is 1240 sqft. For the sake of example, we will assume that a basement or second floor addon will be the same sqft as the first floor.

In [1655]:
```
sqft = 159.48

sqft500 = sqft * 500
sqft1000 = sqft * 1000
second_floor = (sqft * 1240) + 11448.94
finished_basement = (sqft * 1240) - 23634.14

print('sqft500 = ' + str(sqft500))
print('sqft1000 = ' + str(sqft1000))
print('second_floor = ' + str(second_floor))
print('finished_basement = ' + str(finished_basement))

1240 * sqft

```

```
sqft500 = 79740.0
sqft1000 = 159480.0
second_floor = 209204.13999999998
finished_basement = 174121.06
```

Out[1655]:
```
197755.19999999998
```

In [1656]:
```
values = [79740.0, 159480.0, 209204.13999999998, 174121.06]

values
```

Out[1656]:
```
[79740.0, 159480.0, 209204.13999999998, 174121.06]
```

In [1657]:

```
fig = plt.figure(figsize=(20, 12))
ax = plt.gca()

ax.barh(labels,values, color='tab:green')

ax.set_title('Home Addition Bonuses', fontsize=20)
plt.ylabel('Addition Type', fontsize=16)
plt.xlabel('Price Impact', fontsize=16)

ax.set_xticks([0, 50000, 100000, 150000, 200000, 250000])
ax.set_xticklabels(['$0', '$50k', '$100k', '$150k', '$200k', '$250k'], fonts
ax.set_yticklabels(labels, fontsize=16)

ax.invert_yaxis()

plt.savefig('images/additions.png')

plt.show()
```

```
<ipython-input-1657-8fa3b030f5d6>:12: UserWarning: FixedFormatter should only b
e used together with FixedLocator
  ax.set_yticklabels(labels, fontsize=16)
```