Contents

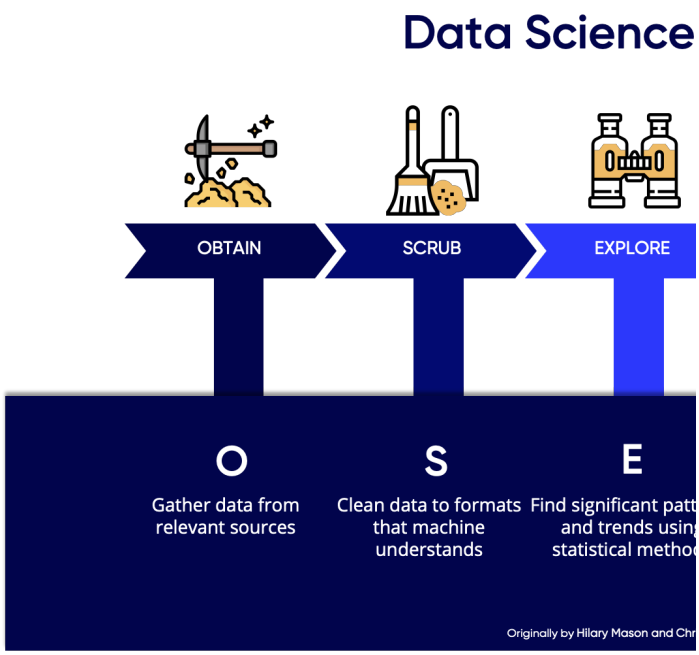
1 RESOURCES FOR YOU

(Delete from final notebook)

- OVERVIEW OF OSEMiN
- PROCESS-CHECKLIST
  - Can actually keep this part if you'd like.
- LINKS FOR MOD 1 PROJECT

1.1 Study Group Recordings Playlist

- Project Specific:
  - Intro to Mod Project from 100719PT Cohort (htt
- 4-Part Lessons for Regression (100719PT cohort)
  - https://www.youtube.com/playlist?list=PLFknVe
    (https://www.youtube.com/playlist?list=PLFknVe



(https://www.kdnuggets.com/2018/02/data-science-

Contents

> The Data Science Process we'll be us
> OSEMiN (pronounced "OH-sum", rhy
> the most straightforward of the Data S
> so far. **Note that during this proces
> stages often blur together.*** It is cor
> ***often a best practice!) to float bac
> as you learn new things about your pr
> requirements, etc.
> It's quite common to get to the modeli
> need to scrub your data a bit more or
> and jump back to the "Scrub" stage, c
> "Obtain" stage when you realize your
> solve this problem. As with any of the
> meant to be treated as guidelines, no

1.2 OSEMN DETAILS

OBTAIN

- This step involves understanding stakeholder requir
  problem, and finally sourcing data that we think will

SCRUB

- During this stage, we'll focus on preprocessing our d
  removing null values, dealing with outliers, normaliz
  selection are handled around this stage. The line wi
  stage, as it is common to only realize that certain co
  result of the visualzations and explorations done du
- Note that although technically, categorical data shou
  practice, it's usually done after data exploration. Thi
  to visualize and explore a few columns containing ca
  different dummy columns that have been one-hot er

EXPLORE

- This step focuses on getting to know the dataset yo
  step tends to blend with the *Scrub* step mentioned a
  visualizations to really get a feel for your dataset. Yo
  the distribution of different columns, checking for mu
  project is a classification task, you may check the ba
  If your problem is a regression task, you may check
  necessary for a regression task.
- At the end of this step, you should have a dataset re
  explored and are extremely familiar with.

MODEL

- This step, as with the last two frameworks, is also pr
  and tuning models using all the tools you have in yo
  means defining a threshold for success, selecting m
  project, and tuning the ones that show promise to tr

stages, it is both common and accepted to realize s
Scrub or *Explore*, and make some changes to see h

iNTERPRET

- During this step, you'll interpret the results of your m
stakeholders. As with the other frameworks, commu
stage, you may come to realize that further investig
fine--figure out what's needed, go get it, and start th
to all stakeholders involved, you may also go from t
and automating processes necessary to support it.

Note: Delete this markdown cell from your final project n

2 PROCESS CHECKLIST

1. **OBTAIN**

- Import data, inspect, check for datatypes to convert
- Display header and info.
- Drop any unneeded columns, if known (`df.drop([`

2. **SCRUB**

- Recast data types, identify outliers, check for multicc
- Check and cast data types
  - ☐ Check for #'s that are store as objects (`df.i`
    - when converting to #'s, look for odd values
    - Decide how to deal weird/null values (`df.u`
    - `df.fillna(subset=['col_with_nulls'`
  - ☐ Check for categorical variables stored as inte
    - May be easier to tell when you make a scat
- ☐

  Check for missing values (df.isna().sum())
  - Can drop rows or colums
  - For missing numeric data with median or bin/co
  - For missing categorical data: make NaN own ca
- ☐

  Check for multicollinearity
  - Use seaborn to make correlation matrix plot
  - Good rule of thumb is anything over 0.75 corr is
  with the largest # of variables
- ☐

  Normalize data (may want to do after some explorin
  - Most popular is Z-scoring (but won't fix skew)
  - Can log-transform to fix skewed data

3. **EXPLORE**

   - ☐ Check distributions, outliers, etc**
   - ☐ Check scales, ranges (df.describe())
   - ☐ Check histograms to get an idea of distributions (
     - Can also do kernel density estimates
   - ☐ Use scatter plots to check for linearity and possibl
     - categoricals will look like vertical lines
   - ☐ Use `pd.plotting.scatter_matrix(df)` to visu
   - ☐ Check for linearity.

4. **MODEL**

   - **Fit an initial model:**
     - Run an initial model and get results
   - **Holdout validation / Train/test split**
     - use sklearn `train_test_split`

5. **iNTERPRET**

   - **Assessing the model:**
     - Assess parameters (slope,intercept)
     - Check if the model explains the variation in the
     - *Are the coeffs, slopes, intercepts in appropriate*
     - *Whats the impact of collinearity? Can we ignore*
   - **Revise the fitted model**
     - Multicollinearity is big issue for lin regression an
     - Use the predictive ability of model to test it (like
     - Check for missed non-linearity

6. **Interpret final model and draw >=3 conclusions and r**

DELETE ABOVE THIS LINE FROM YOUR FINAL NOTE

---

3 Final Project Submission

Please fill out:

- Student name: Vinayak Modgil
- Student pace: self paced / part time / full time : full time
- Scheduled project review date/time: TBD
- Instructor name: James Irving
- Blog post URL:

4 Table of Contents

5 INTRODUCTION

5.1 Business Problem

Potential real estate tycoons are looking to purchase houses
are equipped with the data and you need to convince the stak
Return on Investment based on the size, area and the locatio

6 OBTAIN

Data Understanding:

Questions to consider:

- What are the business's pain points related to this projec
- How did you pick the data analysis question(s) that you c
- Why are these questions important from a business pers

6.1 Importing Libraries

In [377]:

```
# Your code here - remember to use markdown c
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import seaborn as sns

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

import scipy.stats as stats
plt.style.use("seaborn")
```

6.2 Importing Dataset

In [378]:

```
1 df = pd.read_csv("data/kc_house_data.csv")
2 df
```

]:

	id	date	price	bedrooms	bathrooms	sqft_
	7129300520	10/13/2014	221900.0	3	1.00	
	5414100192	12/9/2014	538000.0	3	2.25	
	5631500400	2/25/2015	180000.0	2	1.00	
	2487200875	12/9/2014	604000.0	4	3.00	
	1954400510	2/18/2015	510000.0	3	2.00	
	...	...	...	...	...	
	263000018	5/21/2014	360000.0	3	2.50	
	6600060120	2/23/2015	400000.0	4	2.50	

Contents

7 SCRUB

7.1 Data Cleaning

Contents

In [379]:

```
df.isnull().sum()
```

Out[379]:

```
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 [380]:

```
from sklearn.impute import SimpleImputer
## Use imputer variable to clean features
imputer = SimpleImputer(missing_values = np.N
```

In [381]:

```
df["yr_renovated"] = imputer.fit_transform(df
df["yr_renovated"].value_counts()
```

Out[381]:

```
0.0       20853
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

Contents

In [382]:

```
df["view"] = imputer.fit_transform(df["view"]
df["view"].value_counts()
```

Out[382]:

```
0.0    19485
2.0      957
3.0      508
1.0      330
4.0      317
Name: view, dtype: int64
```

In [383]:

```
df["waterfront"] = imputer.fit_transform(df["
df["waterfront"].value_counts()
```

Out[383]:

```
0.0    21451
1.0      146
Name: waterfront, dtype: int64
```

In [384]:

```
df.isnull().sum()
```

Out[384]:

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

Contents

In [385]:

```
df["sqft_basement"] = df["sqft_basement"].map
```

In [386]:

```
df["sqft_basement"] = df["sqft_basement"].ast
df["view"] = df["view"].astype("int64")
df["floors"] = df["floors"].astype("int64")
df["waterfront"] = df["waterfront"].astype("i
```

7.2 Feature Engineering

In [387]:

```
df["large_home"] = df["bedrooms"] > 5
df["large_home"].value_counts()
```

Out[387]:

```
False    21263
True       334
Name: large_home, dtype: int64
```

In [388]:

```
latlong = df[["lat", "long"]]
latlong
```

Out[388]:

	lat	long
0	47.5112	-122.257
1	47.7210	-122.319
2	47.7379	-122.233
3	47.5208	-122.393
4	47.6168	-122.045
...	...	...
21592	47.6993	-122.346
21593	47.5107	-122.362
21594	47.5944	-122.299
21595	47.5345	-122.069
21596	47.5941	-122.299

21597 rows × 2 columns

Contents

In [502]:

```
df["how_old"] = abs(df["yr_built"] - 2015)
```

In [503]:

```
df["renovated"] = df["yr_renovated"] != 0
```

In [504]:

```
df
```

Out[504]:

's	waterfront	view	condition	grade	sqft_above	sqft_basem
1	0	0	3	7	1180	
2	0	0	3	7	2170	40
1	0	0	3	6	770	
1	0	0	5	7	1050	91
1	0	0	3	8	1680	
...	...	...	...	...	...	
3	0	0	3	8	1530	
2	0	0	3	8	2310	
2	0	0	3	7	1020	
2	0	0	3	8	1600	
2	0	0	3	7	1020	

7.3 Plotting relationships between depende variable

Contents

In [505]:

```
def histogram(column):
    '''
    returns histogram of a column
    in the dataframe df
    '''
    hist = df[column].hist()
    return plt.show()

def reg(column, df = df):
    return sns.regplot(x=column, y="price", d
```

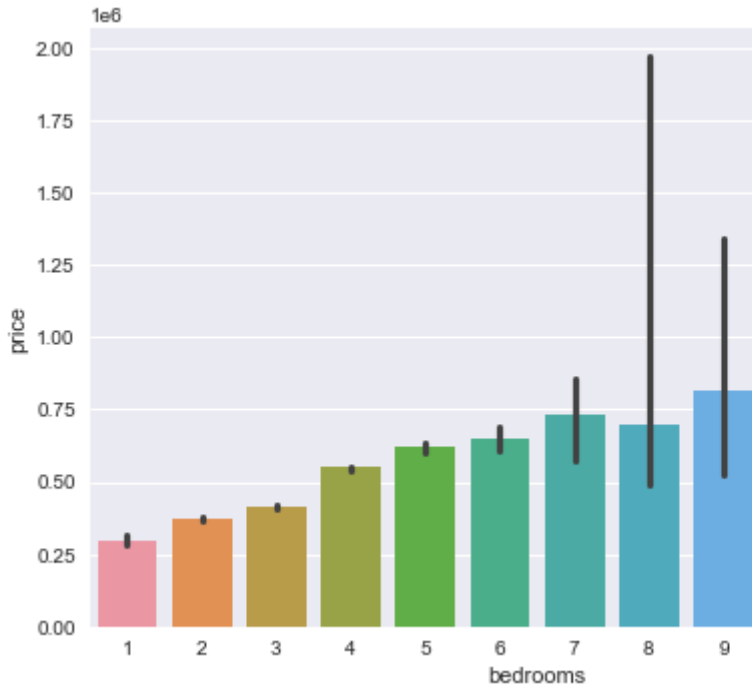
7.3.1 Bedroom

In [506]:

```
sns.barplot(data=df, x="bedrooms", y="price",
```

Out[506]:

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

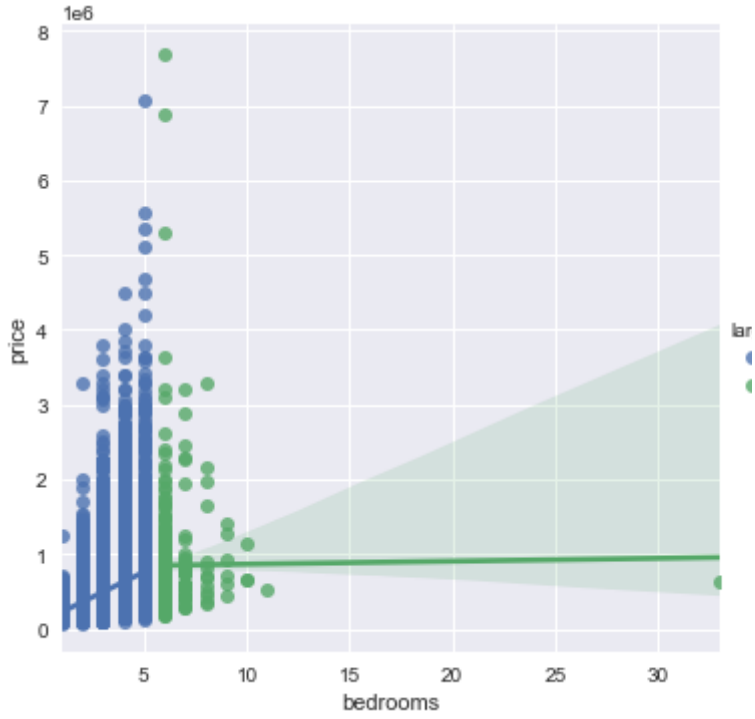


In [507]:

```
1 sns.lmplot(x="bedrooms", y="price", hue="larg
```

Out[507]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4fda2e280>
```



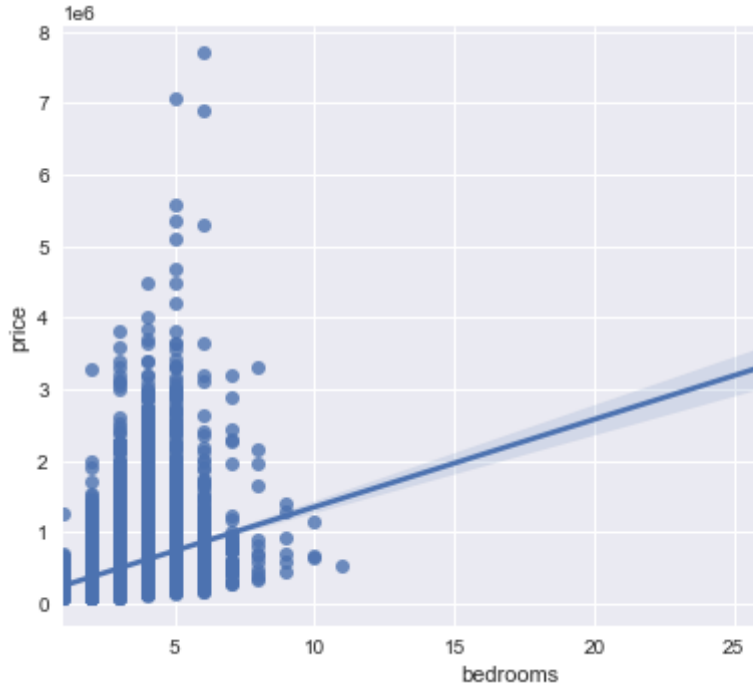
Contents

In [508]:

```
1 reg("bedrooms", df)
```

Out[508]:

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



Contents

Contents

In [509]:

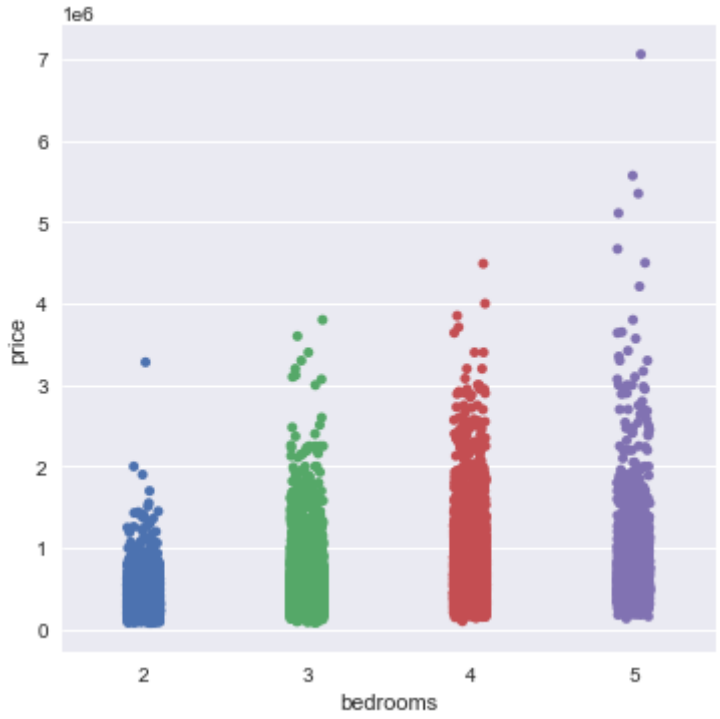
```
def find_outliers(data):
    """
     Detects outliers using the 1.5*IQR thres
    Returns a boolean Series where True=outli

    """
    stats = data.describe()
    q1 = stats["25%"]
    q3 = stats["75%"]
    thresh = 1.5*(q3 - q1)
    idx_outliers = (data < (q1-thresh)) | (da
    return idx_outliers

outliers_bedrooms = find_outliers(df["bedroom
```

In [510]:

```
sns.catplot(data=df[~outliers_bedrooms], x="b
```

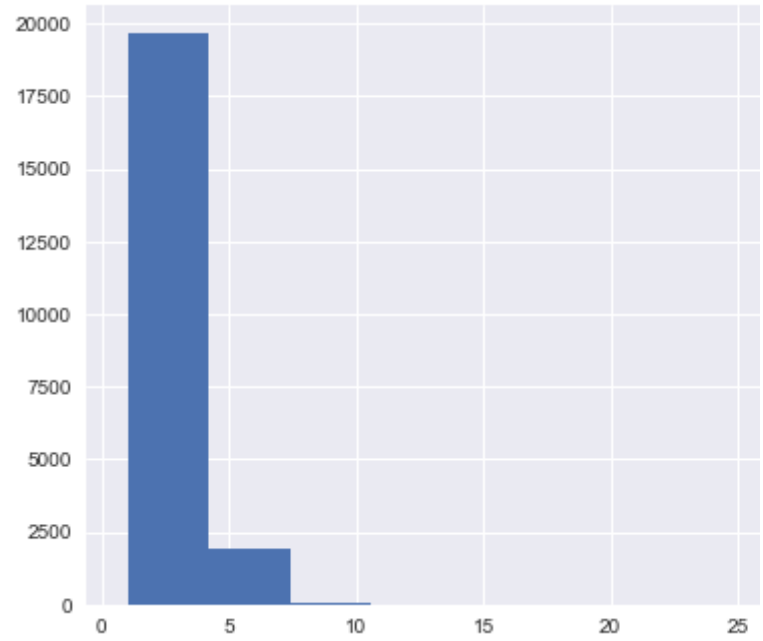
Out[510]:

```
<seaborn.axisgrid.FacetGrid at 0x2b48316c370>
```



In [511]:

```
1 histogram("bedrooms")
```



Contents

7.3.2 Bathroom

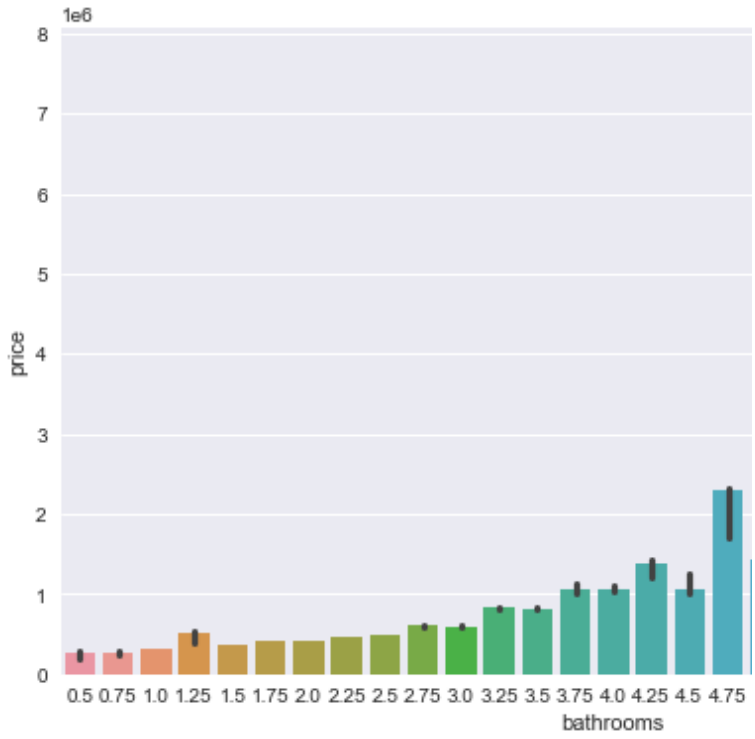
Contents

In [512]:

```
1 fig, ax = plt.subplots(figsize=(10, 6))
2 sns.barplot(x="bathrooms", y="price", data=df
```

Out[512]:

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

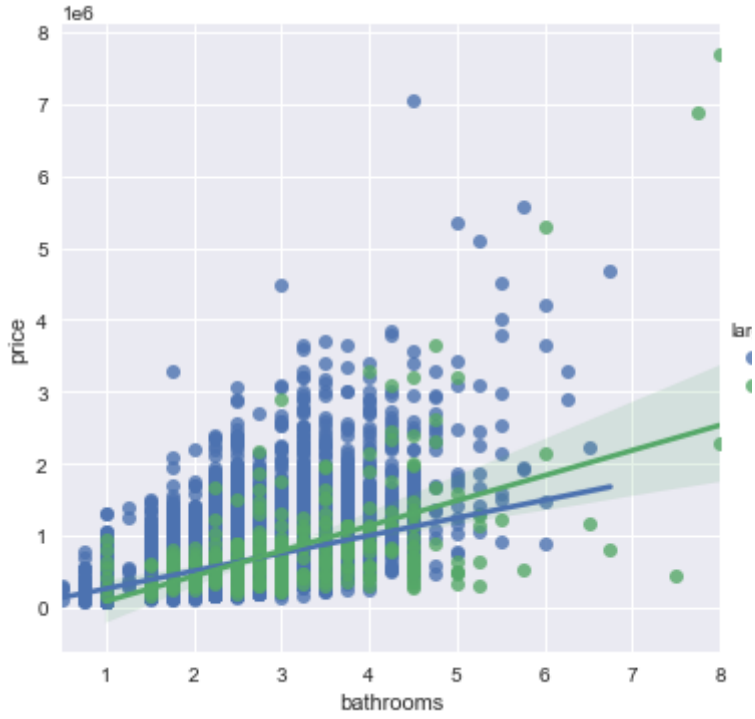


In [513]:

```
1 sns.lmplot(x="bathrooms", y="price", data=df,
```

Out[513]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4fc89b790>
```



Contents

Contents

In [514]:

```
1 outliers_bathroom = find_outliers(df["bathroo
2 outliers_bathroom.value_counts()
```

Out[514]:

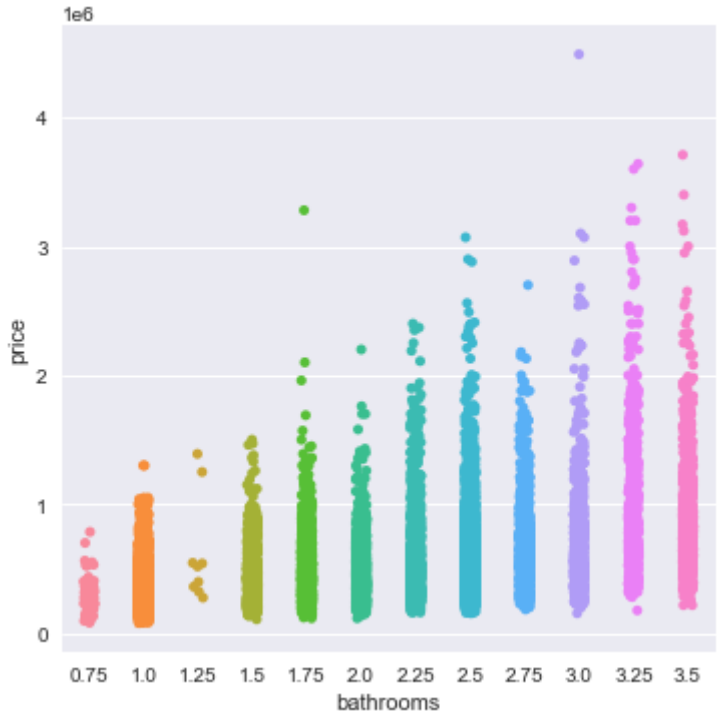
```
False    21036
True       561
Name: bathrooms, dtype: int64
```

In [515]:

```
1 sns.catplot(data=df[~outliers_bathroom], x="b
```

Out[515]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4fff39f10>
```



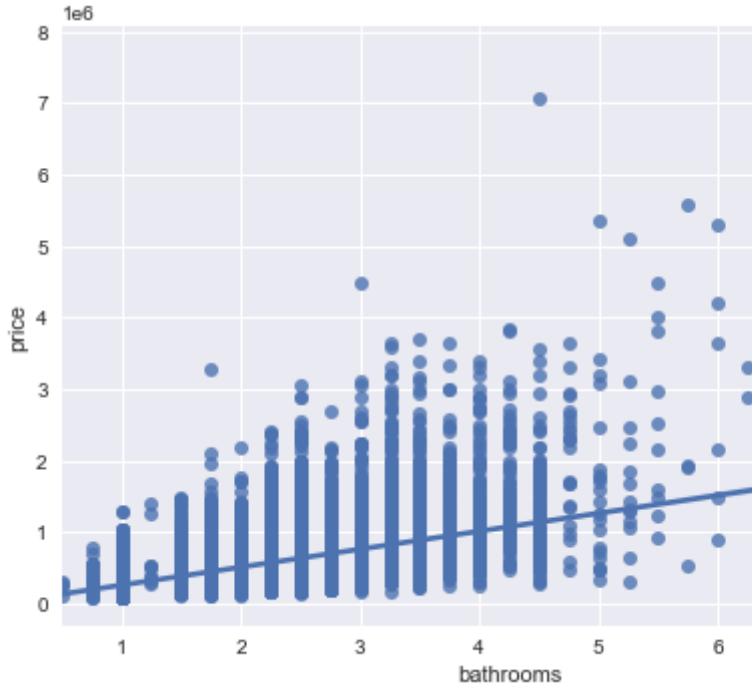
Contents

In [516]:

```
1 reg("bathrooms")
```

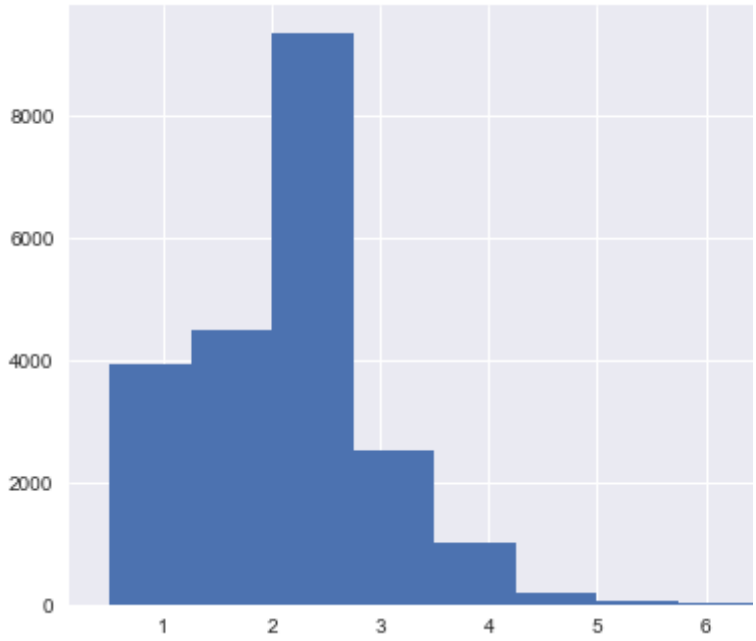
Out[516]:

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



In [517]:

```
1 histogram("bathrooms")
```



Contents

7.3.3 SQFT - living

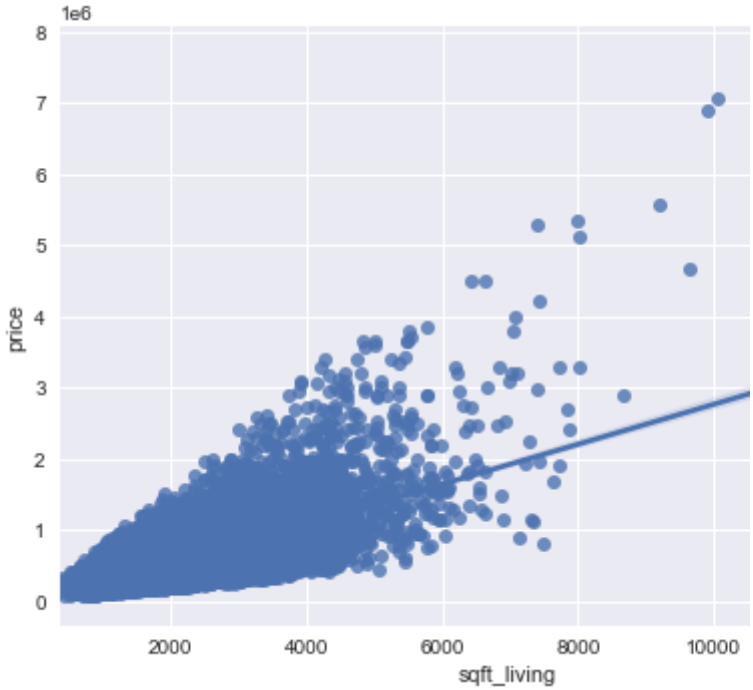
Contents

In [518]:

```
1 reg("sqft_living")
```

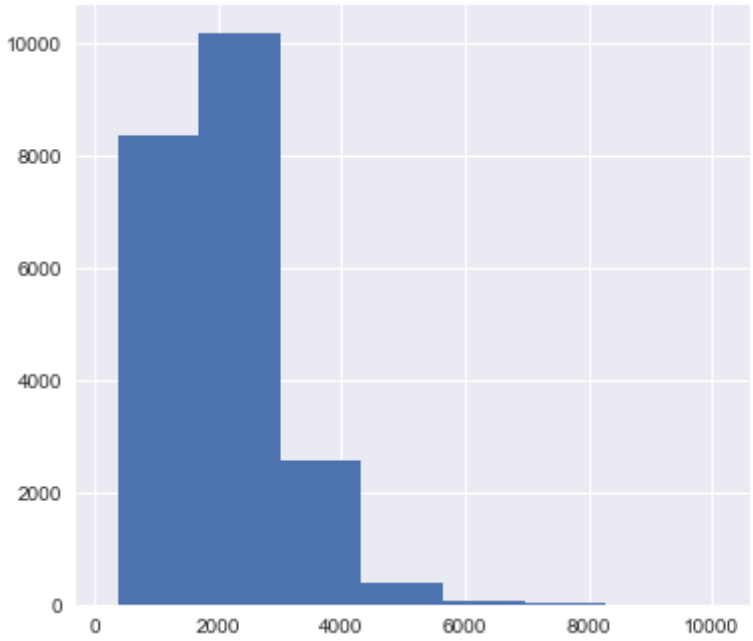
Out[518]:

```
<AxesSubplot:xlabel='sqft_living', ylabel='price'
```



In [519]:

```
1 histogram("sqft_living")
```



Contents

7.3.4 SQFT-lot

In [520]:

```
1 def thousands(x, pos):
2     return "{:1.1f}K".format(x* 1e-4)
```

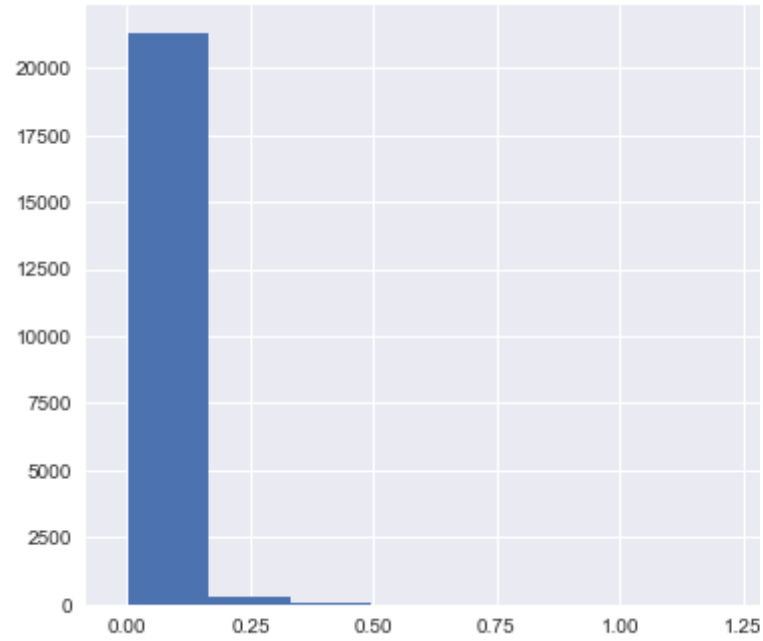
Contents

In [521]:

```
1 fig3 = sns.jointplot(x="sqft_lot", y="price",
2 fig3.ax_joint.yaxis.set_major_formatter(FuncF
3 fig3.ax_joint.xaxis.set_major_formatter(FuncF
```

In [522]:

```
1 histogram("sqft_lot")
```

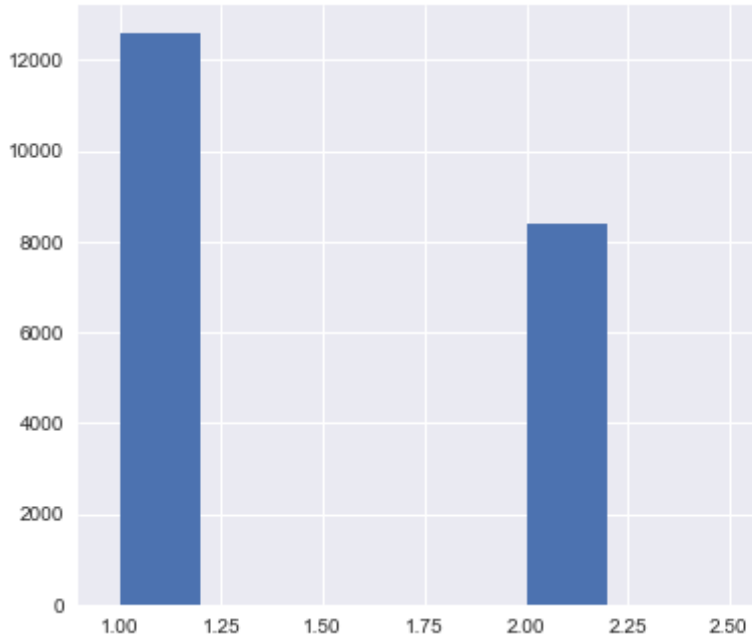


Contents

7.3.5 Floors

In [523]:

```
1 histogram("floors")
```



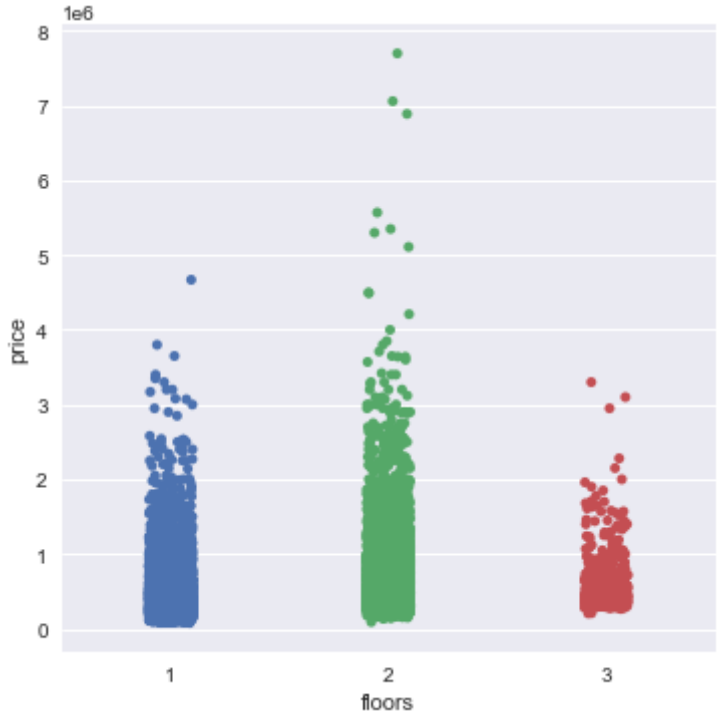
Contents

In [524]:

```
1 sns.catplot(x="floors", y="price", data=df)
```

Out[524]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4dc690cd0>
```

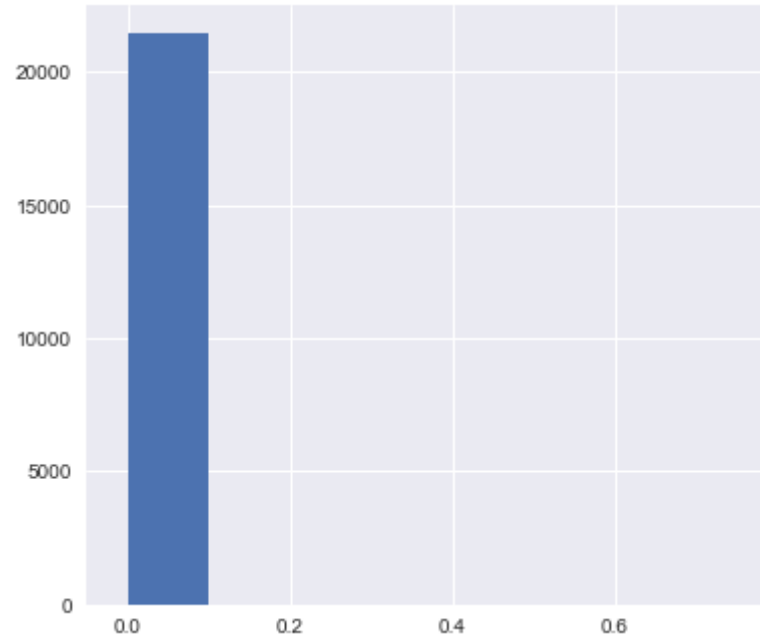


Contents

7.3.6 Waterfront

In [525]:

```
1 histogram("waterfront")
```



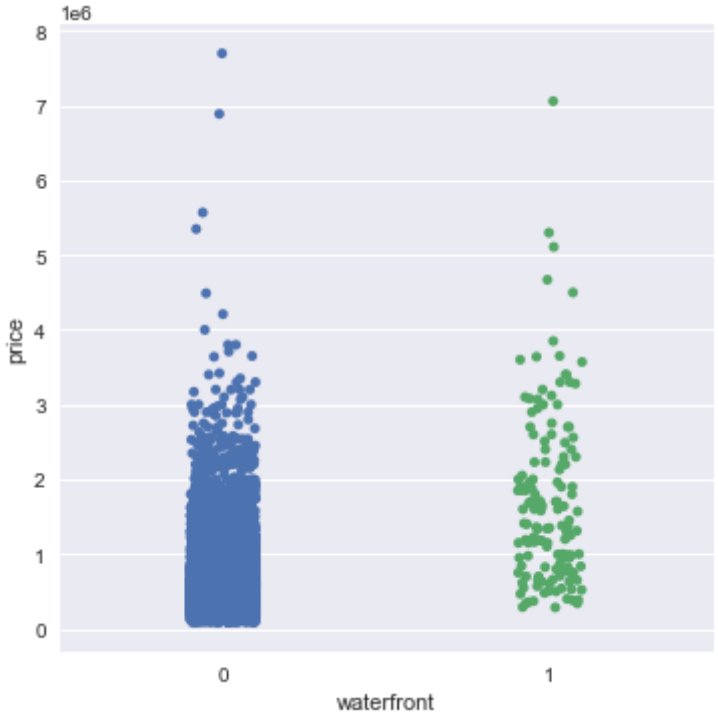
Contents

In [526]:

```
1 sns.catplot(x="waterfront", y="price", data=d
```

Out[526]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4dba5ff40>
```



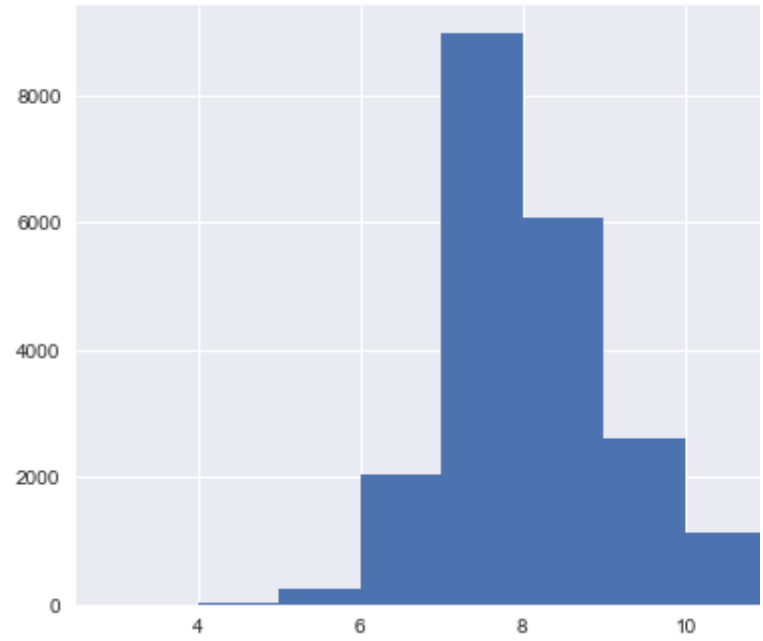
7.3.7 Grade

Contents

Contents

In [527]:

```
1 histogram("grade")
```

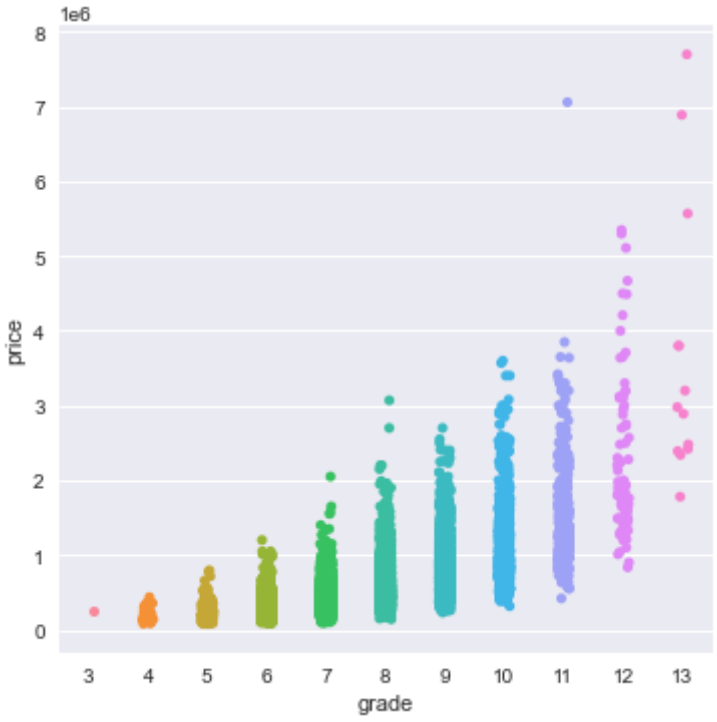


In [528]:

```
1 sns.catplot(x="grade", y="price", data=df)
```

Out[528]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4dbeae130>
```

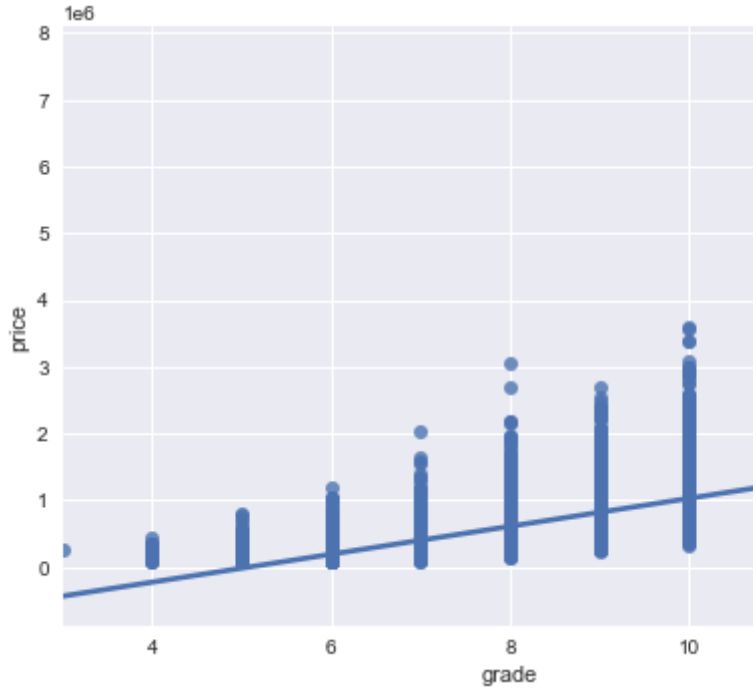


In [529]:

```
1 reg("grade")
```

Out[529]:

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



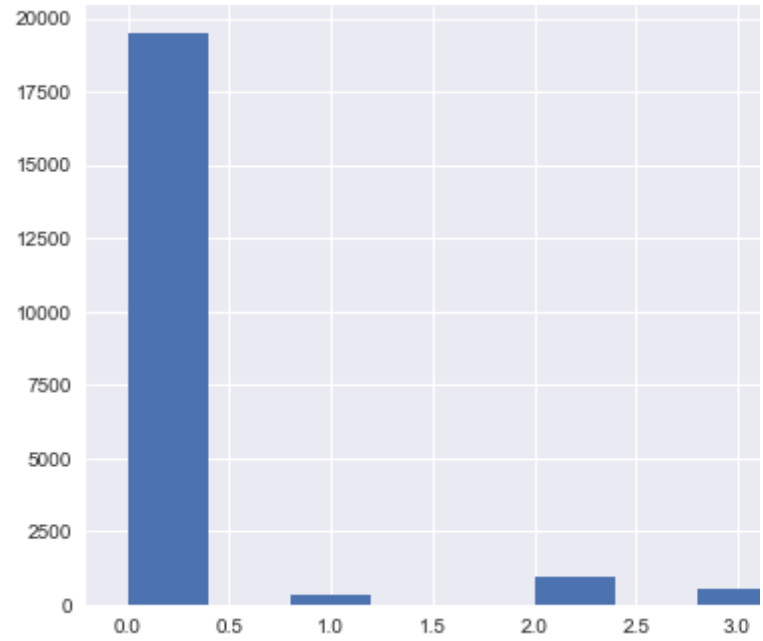
7.3.8 View

Contents

Contents

In [530]:

```
1 histogram("view")
```

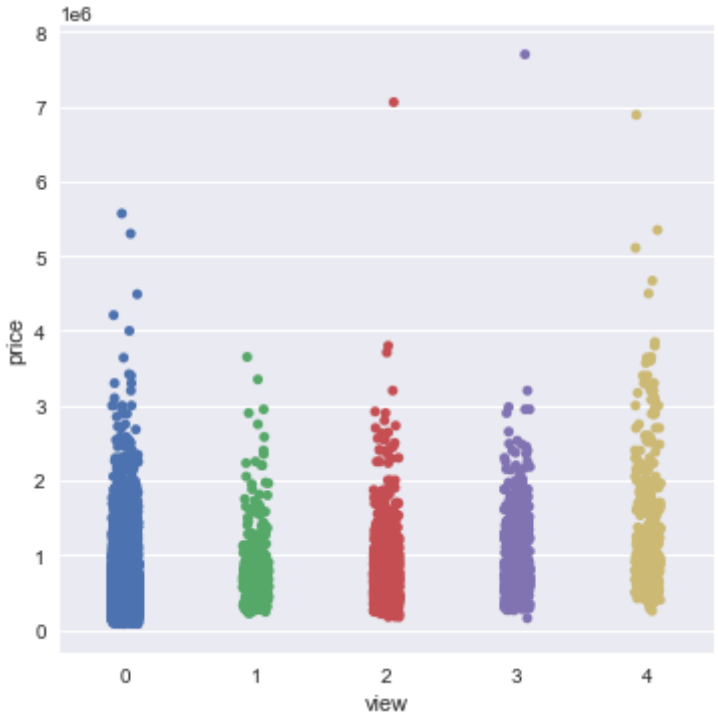


In [531]:

```
1 sns.catplot(x="view", y="price", data=df)
```

Out[531]:

```
<seaborn.axisgrid.FacetGrid at 0x2b4869c2760>
```

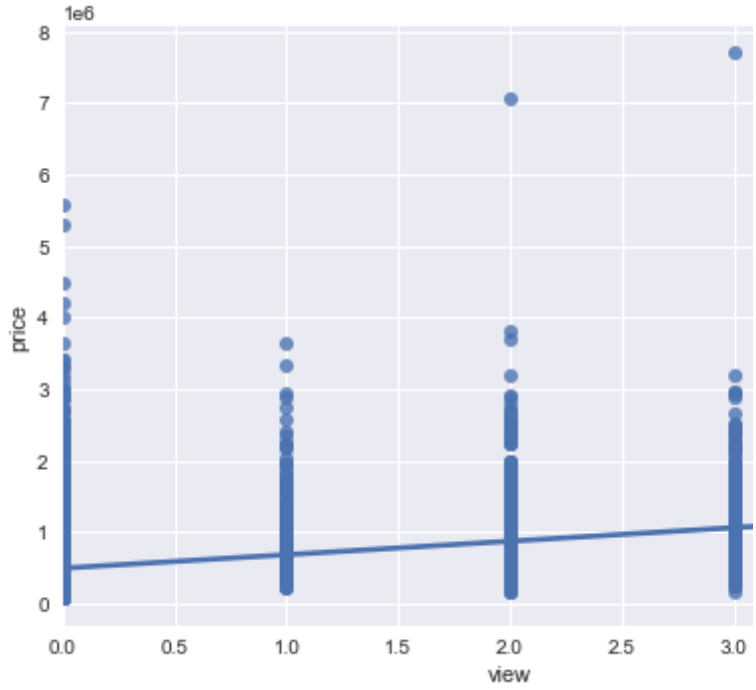


In [532]:

```
1 reg("view")
```

Out[532]:

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



Contents

Contents

In [533]:

```
1 df.drop(["id", "date", "lat", "long"],axis=1,
2 df
```

```
---------------------------------------------
KeyError                                  Traceba
<ipython-input-533-41bd6353d693> in <module>
----> 1 df.drop(["id", "date", "lat", "long"],axi
      2 df

~\Anaconda3\lib\site-packages\pandas\core\frame.py
s, index, columns, level, inplace, errors)
   4161                 weight  1.0     0.8
   4162         """
-> 4163         return super().drop(
   4164             labels=labels,
   4165             axis=axis,

~\Anaconda3\lib\site-packages\pandas\core\generic
xis, index, columns, level, inplace, errors)
   3885         for axis, labels in axes.items():
   3886             if labels is not None:
-> 3887                 obj = obj._drop_axis(labe
rs=errors)
   3888
   3889         if inplace:

~\Anaconda3\lib\site-packages\pandas\core\generic
els, axis, level, errors)
   3919                 new_axis = axis.drop(labe
ors)
   3920             else:
-> 3921                 new_axis = axis.drop(labe
   3922             result = self.reindex(**{axis
   3923

~\Anaconda3\lib\site-packages\pandas\core\indexes
ls, errors)
   5280         if mask.any():
   5281             if errors != "ignore":
-> 5282                 raise KeyError(f"{labels[
   5283             indexer = indexer[~mask]
   5284         return self.delete(indexer)

KeyError: "['id' 'date' 'lat' 'long'] not found i
```

Contents

In [ ]:

```
def move_price_col(df):
    '''
    takes the dataframe as a parameter
    -------------------
    returns the updated dataframe with
    dependent variable in the end
    '''
    # store values of all the columns in cols
    cols = list(df.columns.values)

    # pop the price index from cols
    cols.pop(cols.index("price"))

    # add the price column to the dataframe d
    df = df[cols + ["price"]]
    return df

df = move_price_col(df)
df
```

8 Explore

We are comfortable about having cleaned data, now we can r
we can work with.

8.0.1 Checking for correlation and collinearity

Contents

In [346]:

```
1 df.corr()
```

Out[346]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors
oms	1.000000	0.514508	0.578212	0.032471	0.158065
oms	0.514508	1.000000	0.755758	0.088373	0.520922
ving	0.578212	0.755758	1.000000	0.173453	0.353372
t_lot	0.032471	0.088373	0.173453	1.000000	-0.008603
oors	0.158065	0.520922	0.353372	-0.008603	1.000000
front	-0.002127	0.063629	0.104637	0.021459	0.018321
view	0.078354	0.186016	0.281715	0.075054	0.023711
ition	0.026496	-0.126479	-0.059445	-0.008830	-0.293463
rade	0.356563	0.665838	0.762779	0.114731	0.473273
ove	0.479386	0.686668	0.876448	0.184139	0.518037
ment	0.297229	0.278485	0.428660	0.015031	-0.231754
built	0.155670	0.507173	0.318152	0.052946	0.578549
ated	0.017900	0.047177	0.051060	0.004979	-0.009505
code	-0.154092	-0.204786	-0.199802	-0.129586	-0.097146
ng15	0.393406	0.569884	0.756402	0.144763	0.296797
ot15	0.030690	0.088303	0.184342	0.718204	-0.012766
ome	0.406468	0.172900	0.178745	0.008984	0.023155
rice	0.308787	0.525906	0.701917	0.089876	0.237264

Contents

In [347]:

```
1 abs(df.corr()) > 0.75
```

Out[347]:

	bedrooms	bathrooms	sqft_living	sqft_lot
bedrooms	True	False	False	False
bathrooms	False	True	True	False
sqft_living	False	True	True	False
sqft_lot	False	False	False	True
floors	False	False	False	False
waterfront	False	False	False	False
view	False	False	False	False
condition	False	False	False	False
grade	False	False	True	False
sqft_above	False	False	True	False
sqft_basement	False	False	False	False
yr_built	False	False	False	False
yr_renovated	False	False	False	False
zipcode	False	False	False	False
sqft_living15	False	False	True	False
sqft_lot15	False	False	False	False
large_home	False	False	False	False
price	False	False	False	False

Contents

In [348]:

```
## Making a dataframe with all the non-duplic
df_corr = df.corr().abs().stack().reset_index

df_corr["pairs"] = list(zip(df_corr["level_0"

## Setting index to the new column "pairs" cr
df_corr.set_index(["pairs"], inplace=True)

## Dropping the columns "level_1" and "level_
df_corr.drop(columns=["level_1", "level_0"],

df_corr.columns = ["cc"]

df_corr[(df_corr["cc"] > 0.75) & (df_corr["cc
```

Out[348]:

pairs	cc
(sqft_living, sqft_above)	0.876448
(sqft_above, sqft_living)	0.876448
(sqft_living, grade)	0.762779
(grade, sqft_living)	0.762779
(sqft_living15, sqft_living)	0.756402
(sqft_living, sqft_living15)	0.756402
(sqft_above, grade)	0.756073
(grade, sqft_above)	0.756073
(bathrooms, sqft_living)	0.755758
(sqft_living, bathrooms)	0.755758

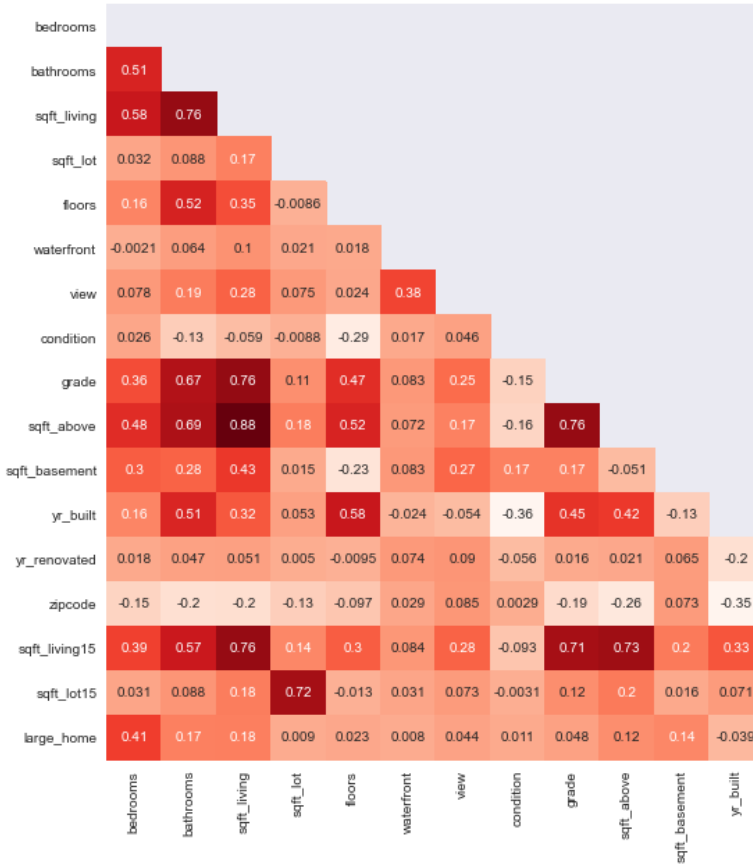
Contents

In [349]:

```
def heatmap(df, figsize=(15, 10), cmap="Reds"
    corr = df.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, m
    return fig, ax

heatmap(df)
```

Out[349]:

```
(<Figure size 1080x720 with 2 Axes>, <AxesSubplot
```



Since sqft_above and sqft_living are the common variables w
less than 1, we shall drop them from our required dataframe.

In [350]:

```
1 df.drop(columns =["sqft_living", "sqft_above"
```

```
C:\Users\Vinayak Modgil\Anaconda3\lib\site-packag
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice
```

```
See the caveats in the documentation: https://pan
stable/user_guide/indexing.html#returning-a-view-
as.pydata.org/pandas-docs/stable/user_guide/index
ersus-a-copy)
  return super().drop(
```

In [351]:

```
1 df
```

Out[351]:

	bedrooms	bathrooms	sqft_lot	floors	waterfront	v
0	3	1.00	5650	1	0	
1	3	2.25	7242	2	0	
2	2	1.00	10000	1	0	
3	4	3.00	5000	1	0	
4	3	2.00	8080	1	0	
...	...	...	...	...	...	
21592	3	2.50	1131	3	0	
21593	4	2.50	5813	2	0	
21594	2	0.75	1350	2	0	
21595	3	2.50	2388	2	0	
21596	2	0.75	1076	2	0	

21597 rows × 16 columns

Contents

Contents

In [352]:

```
1 df["bedrooms"].value_counts()
```

Out[352]:

```
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

In [353]:

```
1 df["view"].value_counts()
```

Out[353]:

```
0    19485
2      957
3      508
1      330
4      317
Name: view, dtype: int64
```

In [354]:

```
1 df["floors"].value_counts()
```

Out[354]:

```
1    12583
2     8396
3      618
Name: floors, dtype: int64
```

Contents

In [355]:

```
df["condition"].value_counts()
```

Out[355]:

```
3    14020
4     5677
5     1701
2      170
1       29
Name: condition, dtype: int64
```

In [544]:

```
categoricals = ["bedrooms", "bathrooms","floo
numeric =  ["sqft_lot", "sqft_basement", "how
           "sqft_living15", "sqft_lot15", "pr
cat1 = ["large_home", "renovated"]
```

8.0.2 Taking care of numeric data

In [545]:

```
def standardize(feature):
    '''
    takes a feature in the df as the paramete
    ------------------------------------
    returns the standardized value of the fea
    '''
    return (feature - feature.mean()) / featu
```

In [546]:

```
#df_stdized = df[numeric].apply(standardize)
#df_stdized
```

8.0.3 Taking care of categorical data

In [547]:

```
1 df[cat1]
```

Out[547]:

	large_home	renovated
0	False	False
1	False	True
2	False	False
3	False	False
4	False	False
...	...	...
21592	False	False
21593	False	False
21594	False	False
21595	False	False
21596	False	False

21597 rows × 2 columns

Contents

Contents

In [548]:

```
from sklearn.preprocessing import OneHotEncod
ohe = OneHotEncoder(sparse=False, drop="first
arr = ohe.fit_transform(df[cat1])
cat_df = pd.DataFrame(arr, columns= ohe.get_f
cat_df
```

Out[548]:

	large_home_True	renovated_True
0	0.0	0.0
1	0.0	1.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
...	...	...
21592	0.0	0.0
21593	0.0	0.0
21594	0.0	0.0
21595	0.0	0.0
21596	0.0	0.0

21597 rows × 2 columns

In [585]:

```
modeling_df = pd.concat([df[numeric], cat_df,
```

In [586]:

```
1 modeling_df
```

Out[586]:

	sqft_lot	sqft_basement	how_old	sqft_living15	sqft_
0	5650	0.0	60	1340	
1	7242	400.0	64	1690	
2	10000	0.0	82	2720	
3	5000	910.0	50	1360	
4	8080	0.0	28	1800	
...	...	...	...	...	
21592	1131	0.0	6	1530	
21593	5813	0.0	1	1830	
21594	1350	0.0	6	1020	
21595	2388	0.0	11	1410	
21596	1076	0.0	7	1020	

21597 rows × 15 columns

In [587]:

```
1 modeling_df.to_csv("kc_cleaned.csv", index=Fa
```

Contents

9 Modeling

9.1 Data Modeling

Describe and justify the process for analyzing or modeling the

---

Questions to consider:

- How did you analyze or model the data?
- How did you iterate on your initial approach to make it be
- Why are these choices appropriate given the data and th

Contents

In [588]:

```
#Define the problem
outcome = "price"
x_cols = list(modeling_df.columns)
x_cols.pop(x_cols.index("price"))

```

Out[588]:

```
'price'
```

In [589]:

```
from statsmodels.formula.api import ols
```

In [594]:

```
def check_model(model):
    '''

    '''

    resids = model.resid

    sm.graphics.qqplot(resids, stats.distribu
    xs = np.linspace(0,1,len(resids))

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

    fig, ax = plt.subplots()
    ax.scatter(x=y_hat,y=resid)

    return fig,ax
```

Contents

In [595]:

```
def make_model(df_name, categoricals=categori
    '''

    '''

    features = ' + '.join(df_name.drop('price
    for variable in categoricals:
        features = features.replace(variable,

    f  = "price~"+features

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

    check_model(model)
    plt.show()

    return model

model1 = make_model(modeling_df)
```

C(zipcode)[T.98199]	3.586e+05	1.25e+04	28.623	0.000	3
sqft_lot	0.3642	0.038	9.547	0.000	
sqft_basement	54.1142	3.201	16.903	0.000	
how_old	731.2009	67.520	10.829	0.000	
sqft_living15	84.9631	2.709	31.358	0.000	
sqft_lot15	-0.0815	0.060	-1.356	0.175	
large_home_True	9513.9504	3.44e+04	0.277	0.782	-5
renovated_True	7.008e+04	6394.787	10.959	0.000	5

Omnibus:	15342.961	**Durbin-Watson:**	1.986
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1618798.509
Skew:	2.624	**Prob(JB):**	0.00
Kurtosis:	45.088	**Cond. No.**	1.11e+16

In [563]:

```
X = modeling_df.drop("price", axis=1)
y = modeling_df["price"]
```

Contents

In [564]:

```
1 X
```

Out[564]:

	sqft_lot	sqft_basement	how_old	sqft_living15	sqft_
0	5650	0.0	60	1340	
1	7242	400.0	64	1690	
2	10000	0.0	82	2720	
3	5000	910.0	50	1360	
4	8080	0.0	28	1800	
...	...	...	...	...	
21592	1131	0.0	6	1530	
21593	5813	0.0	1	1830	
21594	1350	0.0	6	1020	
21595	2388	0.0	11	1410	
21596	1076	0.0	7	1020	

21597 rows × 14 columns

In [565]:

```
1 y
```

Out[565]:

```
0         221900.0
1         538000.0
2         180000.0
3         604000.0
4         510000.0
            ...
21592     360000.0
21593     400000.0
21594     402101.0
21595     400000.0
21596     325000.0
Name: price, Length: 21597, dtype: float64
```

In [566]:

```
1 from sklearn.model_selection import train_tes
```

Contents

In [567]:

```
1 X_train, X_test, y_train, y_test = train_test
```

In [568]:

```
1 X_train
```

Out[568]:

	sqft_lot	sqft_basement	how_old	sqft_living15	sqft_
5931	7800	720.0	58	1450	
19984	1102	0.0	8	1320	
13637	4959	1060.0	50	1590	
11462	1168	260.0	13	1650	
19036	9514	0.0	46	1040	
...	...	...	...	...	
10955	7548	1050.0	48	2150	
17289	5750	0.0	34	1060	
5192	13002	0.0	31	1620	
12172	49928	560.0	30	2620	
235	28040	0.0	32	3430	

17277 rows × 14 columns

Contents

In [569]:

```
1 X_test
```

Out[569]:

	sqft_lot	sqft_basement	how_old	sqft_living15	sqft_
16729	8864	0.0	30	1510	
10996	7920	300.0	64	1140	
12089	1824	0.0	8	1460	
554	8280	0.0	59	1570	
16075	7102	750.0	69	1620	
...	...	...	...	...	
13541	9465	0.0	55	1530	
10735	7577	0.0	32	1430	
11018	10500	0.0	72	950	
13521	8164	0.0	65	1340	
8369	5421	0.0	24	1570	

4320 rows × 14 columns

In [570]:

```
1 y_train
```

Out[570]:

```
5931      261000.0
19984     445000.0
13637     350000.0
11462     370350.0
19036     299000.0
           ...
10955     282000.0
17289     317000.0
5192      492450.0
12172     429000.0
235      1030000.0
Name: price, Length: 17277, dtype: float64
```

Contents

In [571]:

```
1 y_test
```

Out[571]:

```
16729    244500.0
10996    190000.0
12089    348500.0
554      396000.0
16075    665000.0
           ...
13541    252000.0
10735    194000.0
11018    235000.0
13521    332500.0
8369     300000.0
Name: price, Length: 4320, dtype: float64
```

In [572]:

```
1 from sklearn.linear_model import LinearRegres
2 linreg = LinearRegression()
3 linreg.fit(X_train, y_train)
```

Out[572]:

```
LinearRegression()
```

In [575]:

```
1 linreg.intercept_
```

Out[575]:

```
-815443.3944476284
```

In [576]:

```
1 linreg.coef_
```

Out[576]:

```
array([ 2.05262135e-01,  9.13176546e+01,  4.17918
       -4.04017663e-01,  7.07845789e+04,  3.50426
        9.31820303e+04,  4.91492716e+04,  8.02675
        1.60509169e+05, -5.65226031e+00])
```

Contents

In [577]:

```
cdf = pd.DataFrame(linreg.coef_, X.columns, c
cdf
```

Out[577]:

	Coefficients
sqft_lot	0.205262
sqft_basement	91.317655
how_old	4179.186498
sqft_living15	95.678314
sqft_lot15	-0.404018
large_home_True	70784.578928
renovated_True	35042.633522
bedrooms	-18481.608326
bathrooms	93182.030301
floors	49149.271612
waterfront	802675.462114
condition	20231.407918
grade	160509.168879
zipcode	-5.652260

In [578]:

```
y_hat_test = linreg.predict(X_test)
y_hat_train = linreg.predict(X_train)
```

In [579]:

```
from sklearn.metrics import mean_squared_erro

train_mse = mean_squared_error(y_hat_train, y
test_mse = mean_squared_error(y_hat_test, y_t

print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error:', test_mse)
```

```
Train Mean Squared Error: 52223590078.687325
Test Mean Squared Error: 46425215441.21558
```

In [ ]:

```

```

Contents