1 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:

2 Table of Contents

3 INTRODUCTION

3.1 Business Problem

Potential real estate tycoons are looking to purchase houses so that they can sell it out to future house owners. You are equipped with the data and you need to advise the stakeholders on the affect of different parameters that affect the value of houses in the King County.

4 OBTAIN

Data Understanding:

Questions to consider:

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

4.1 Importing Libraries

In [4611]:

```
# Your code here - remember to use markdown cells for comments as well!
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")
```

4.2 Importing Dataset

In [4612]:

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

Out[4612]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0
...	...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	0.0
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	0.0
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	0.0

5 SCRUB

5.1 Data Cleaning

In [4613]:

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

Out[4613]:

```
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 [4614]:

```
from sklearn.impute import SimpleImputer
## Use imputer variable to clean features
imputer = SimpleImputer(missing_values = np.NaN, strategy="most_frequent")
```

In [4615]:

```
df["yr_renovated"] = imputer.fit_transform(df["yr_renovated"].values.reshape(-1, 1))[:,
df["yr_renovated"].value_counts()
```

Out[4615]:

```
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
```

In [4616]:

```
df["view"] = imputer.fit_transform(df["view"].values.reshape(-1, 1))[:, 0]
df["view"].value_counts()
```

Out[4616]:

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

In [4617]:

```
df["waterfront"] = imputer.fit_transform(df["waterfront"].values.reshape(-1, 1))[:, 0]
df["waterfront"].value_counts()
```

Out[4617]:

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

In [4618]:

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

Out[4618]:

```
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
```

In [4619]:

```
df["sqft_basement"] = df["sqft_basement"].map(lambda x: x.replace("?", "0.0"))
```

5.2 Feature Engineering

In [4620]:

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

Out[4620]:

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

In [4621]:

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

Out[4621]:

	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

In [4622]:

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

Out[4622]:

```
1      559
9      453
10     450
11     433
12     420
      ...
82      30
114     29
113     27
80      24
81      21
Name: how_old, Length: 116, dtype: int64
```

In [4623]:

```
df["renovated"] = df["yr_renovated"] != 0
df["renovated"].value_counts()
```

Out[4623]:

```
False    20853
True       744
Name: renovated, dtype: int64
```

In [4722]:

```
df["has_basement"] = df["sqft_basement"] != 0.0
df["has_basement"].value_counts()
```

Out[4722]:

```
False    13280
True      8317
Name: has_basement, dtype: int64
```

In [4723]:

```
df["sqft_living_comparison"] = df["sqft_living"] > df["sqft_living15"]
```

In [4724]:

```
df["sqft_basement"] = df["sqft_basement"].astype("float64")
df["view"] = df["view"].astype("int64")
df["floors"] = df["floors"].astype("int64")
df["waterfront"] = df["waterfront"].astype("int64")
df["renovated"] = df["renovated"].astype("int64")
df["large_home"] = df["large_home"].astype("int64")
df["sqft_living_comparison"] = df["sqft_living_comparison"].astype("int64")
```

In [4725]:

```
df
```

Out[4725]:

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	3	1.00	1180	5650	1	0
1	6414100192	12/9/2014	3	2.25	2570	7242	2	0
2	5631500400	2/25/2015	2	1.00	770	10000	1	0
3	2487200875	12/9/2014	4	3.00	1960	5000	1	0
4	1954400510	2/18/2015	3	2.00	1680	8080	1	0
...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	3	2.50	1530	1131	3	0
21593	6600060120	2/23/2015	4	2.50	2310	5813	2	0
21594	1523300141	6/23/2014	2	0.75	1020	1350	2	0
21595	291310100	1/16/2015	3	2.50	1600	2388	2	0
21596	1523300157	10/15/2014	2	0.75	1020	1076	2	0

21597 rows × 26 columns

5.3 Plotting relationships between dependent and independent variable

In [4726]:

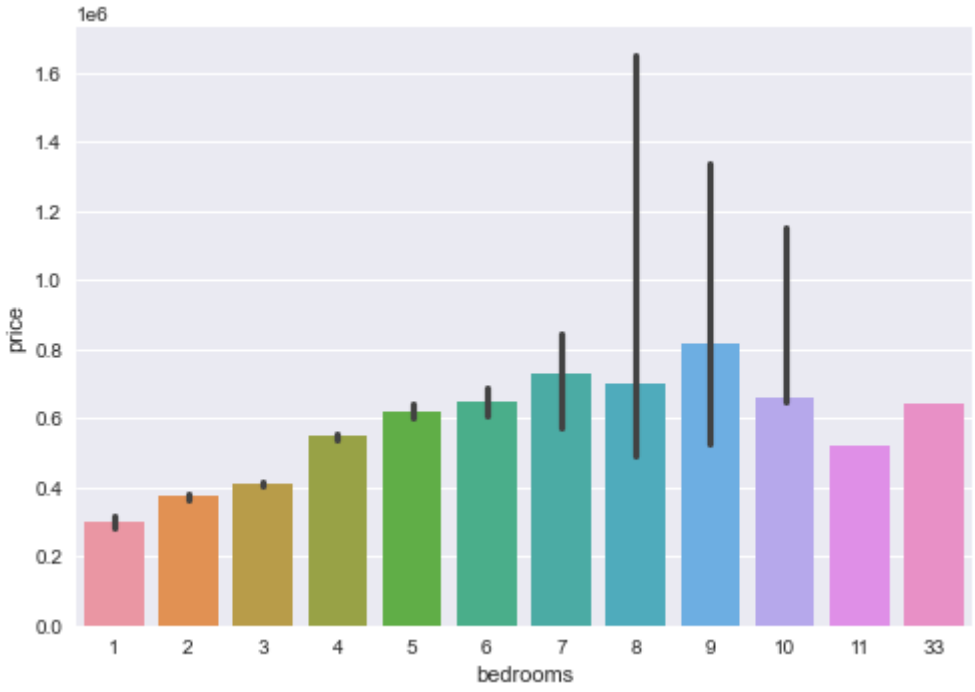
```
def histogram(column):
    '''
    returns histogram of a column
    in the dataframe df
    '''
    hist = df[column].hist()
    return plt.show()

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

5.3.1 Bedroom

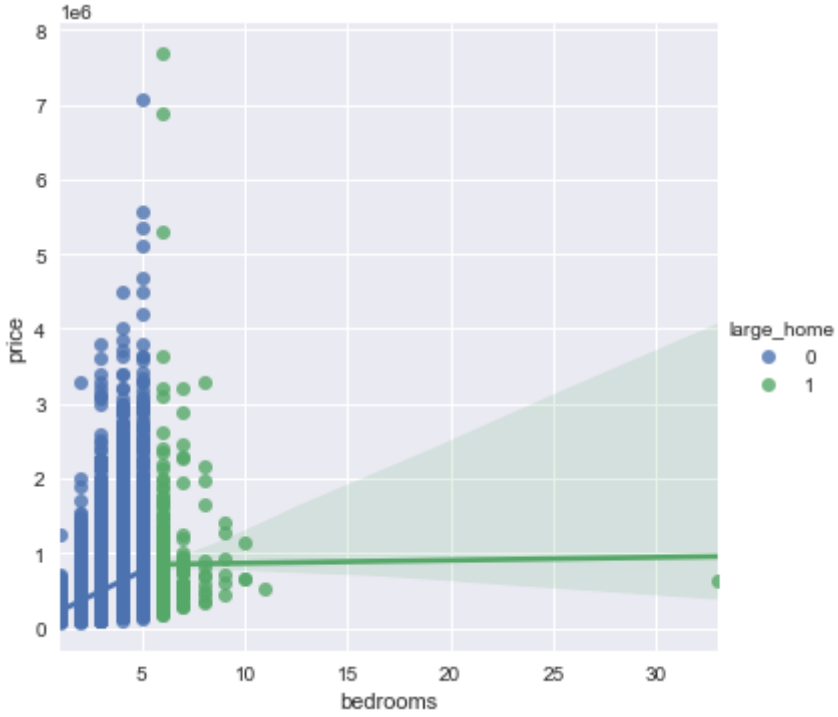
In [4727]:

```
sns.barplot(data=df, x="bedrooms", y="price", ci=95, estimator=np.median)
plt.show()
```



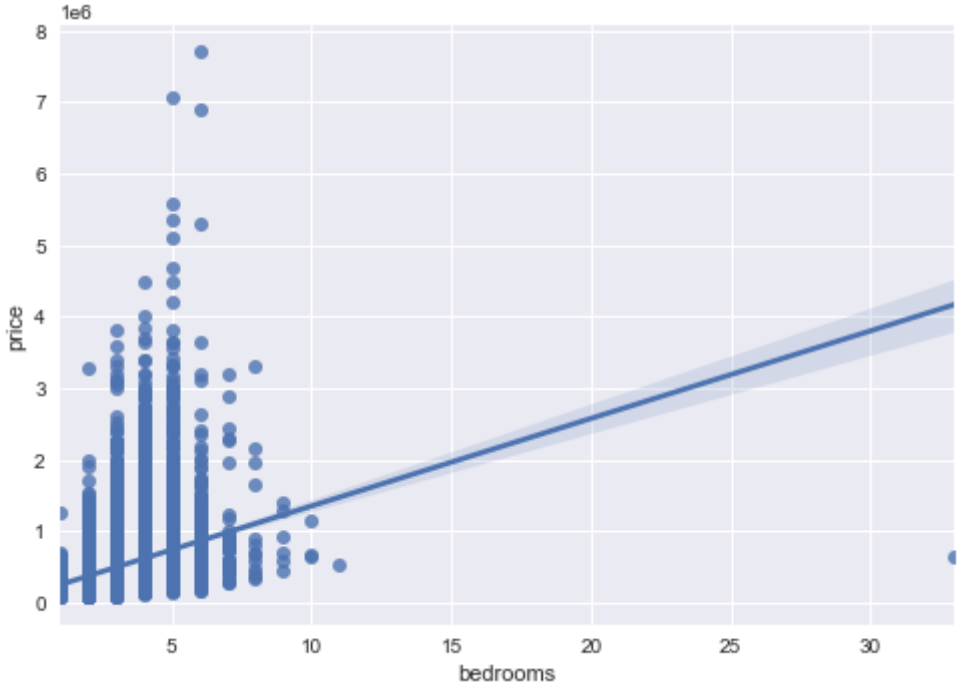
In [4728]:

```
1 sns.lmplot(x="bedrooms", y="price", hue="large_home", data=df)
2 plt.show()
```



In [4729]:

```
reg("bedrooms", df)
plt.show()
```



In [4730]:

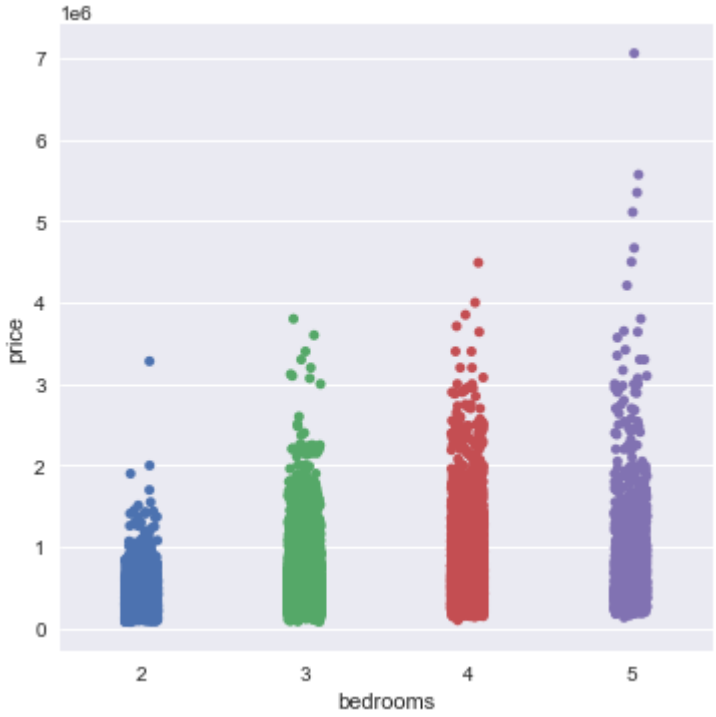
```
def find_outliers(data):
    """
     Detects outliers using the 1.5*IQR thresholds.
    Returns a boolean Series where True=outlier

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

outliers_bedrooms = find_outliers(df["bedrooms"])
```

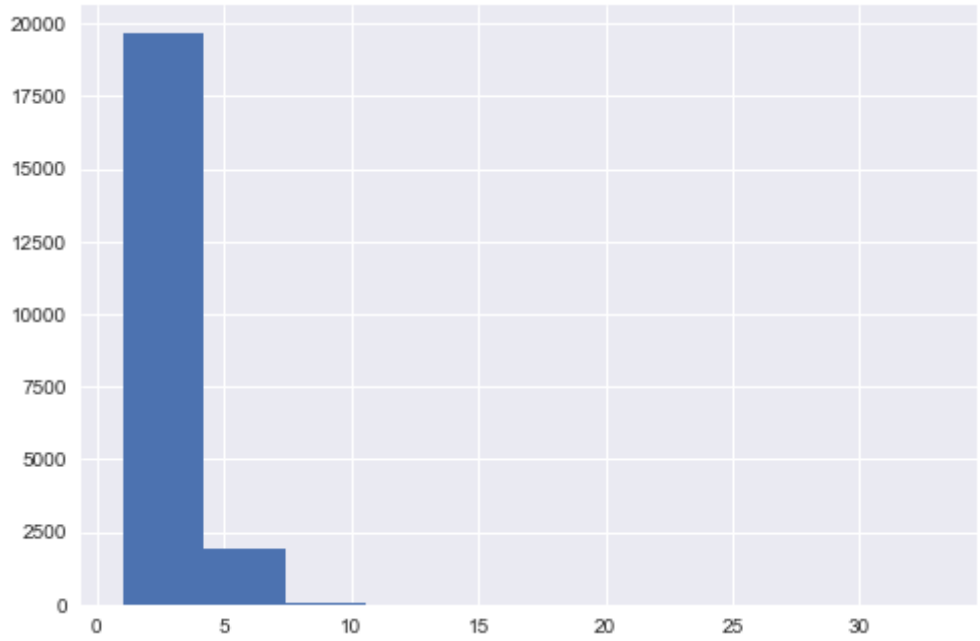
In [4731]:

```
sns.catplot(data=df[~outliers_bedrooms], x="bedrooms", y="price")
plt.show()
```



In [4732]:

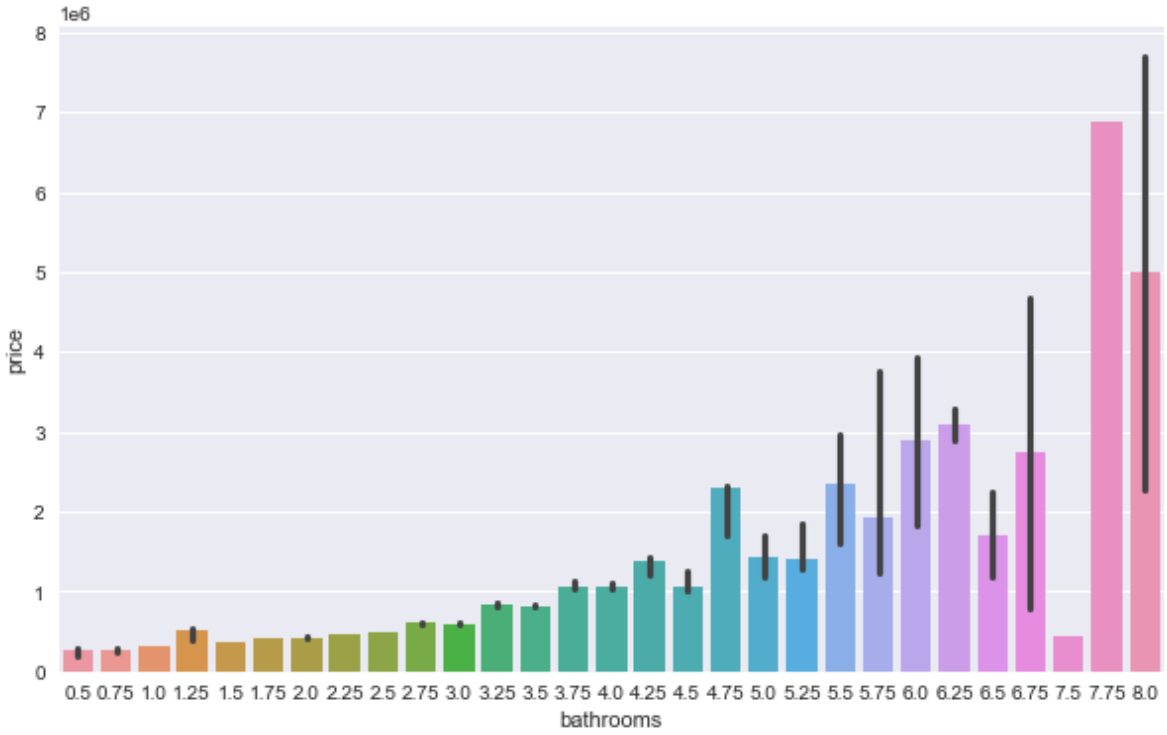
```
1 histogram("bedrooms")
```



5.3.2 Bathroom

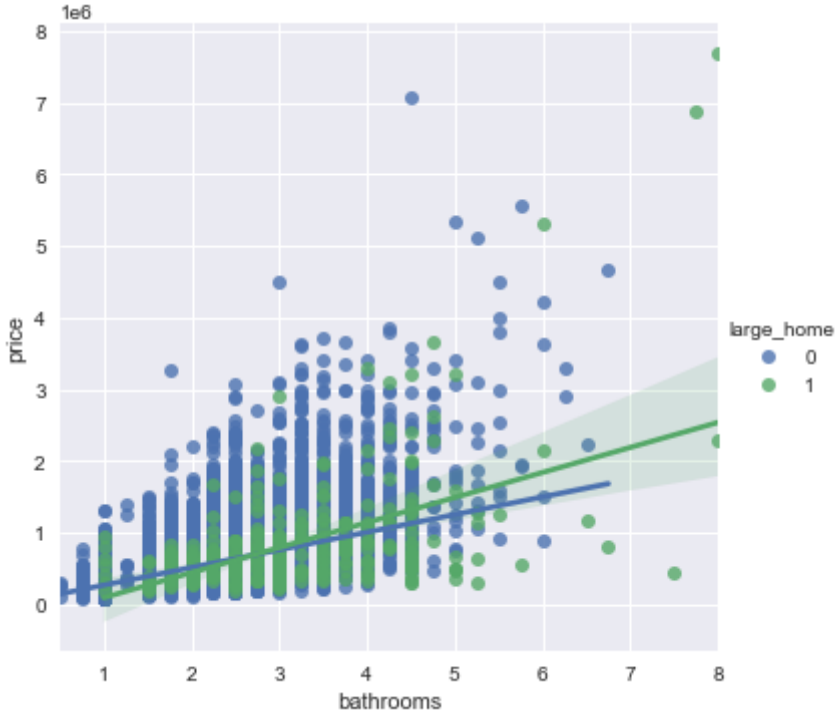
In [4733]:

```
fig, ax = plt.subplots(figsize=(10, 6))
sns.barplot(x="bathrooms", y="price", data=df, ci=68, estimator=np.median, ax=ax)
plt.show()
```



In [4734]:

```
1 sns.lmplot(x="bathrooms", y="price", data=df, hue="large_home")
2 plt.show()
```



In [4735]:

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

Out[4735]:

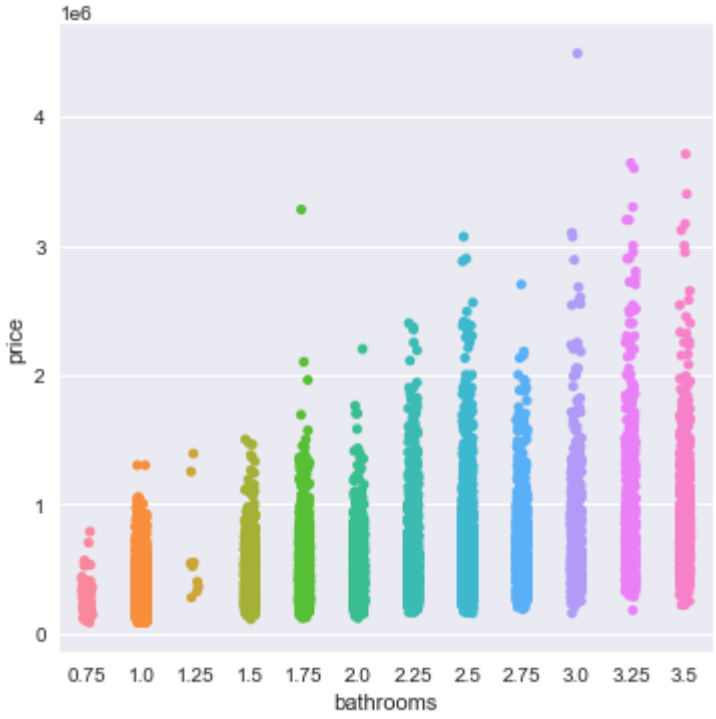
```
False    21036
True       561
Name: bathrooms, dtype: int64
```

In [4736]:

```
1 sns.catplot(data=df[~outliers_bathroom], x="bathrooms", y="price")
```

Out[4736]:

```
<seaborn.axisgrid.FacetGrid at 0x20674dce190>
```

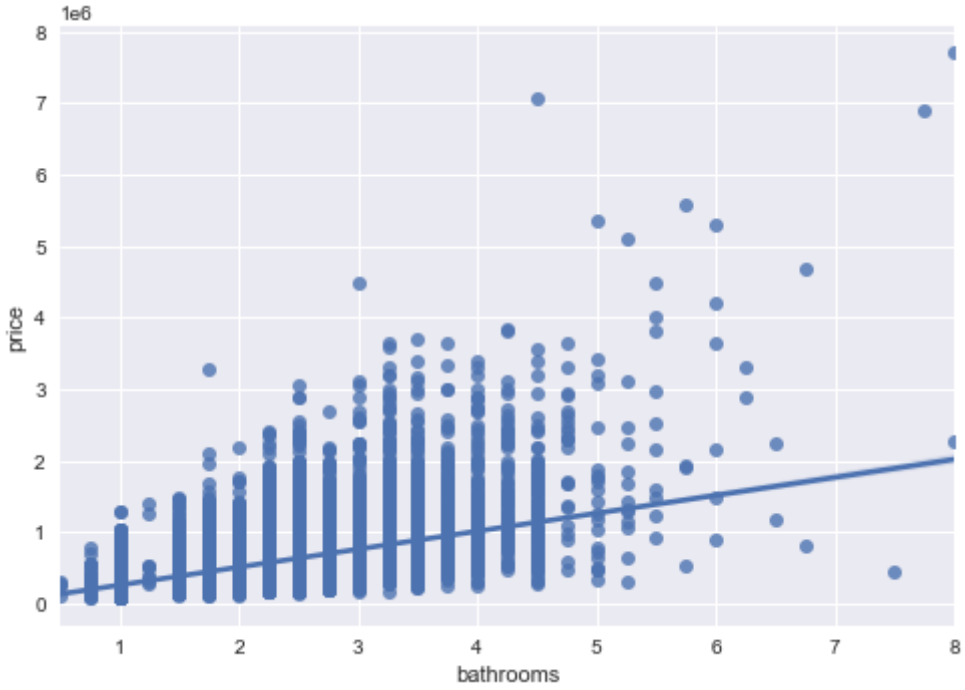


In [4737]:

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

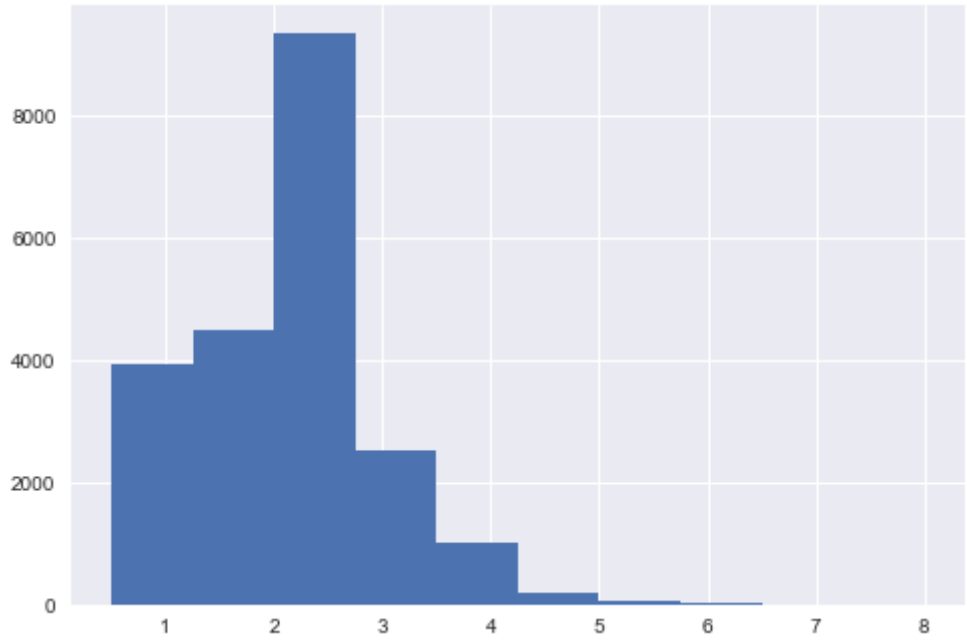
Out[4737]:

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



In [4738]:

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



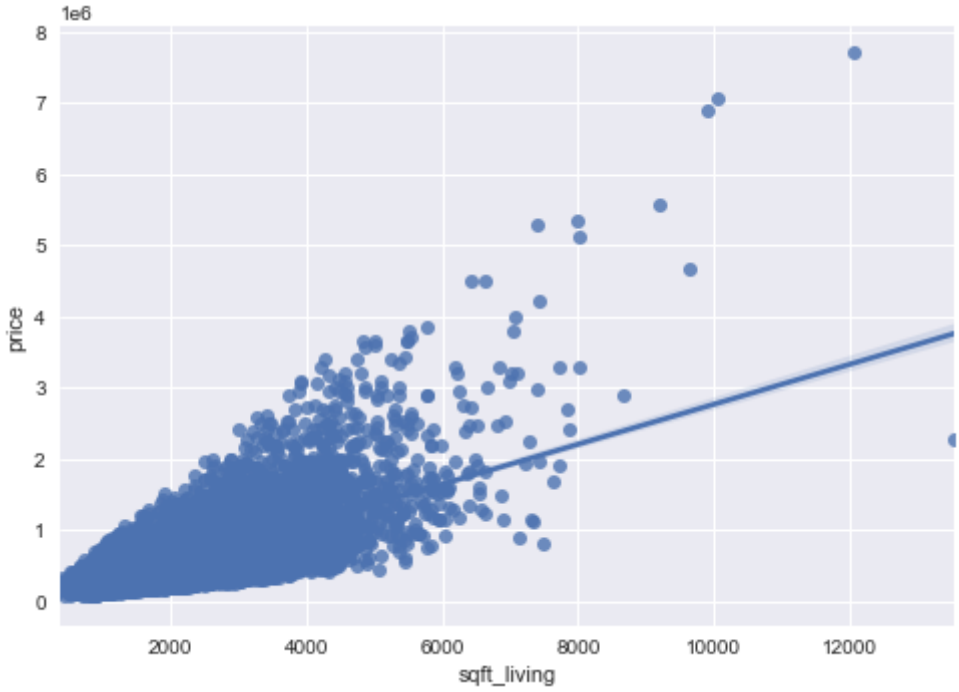
5.3.3 SQFT - living

In [4739]:

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

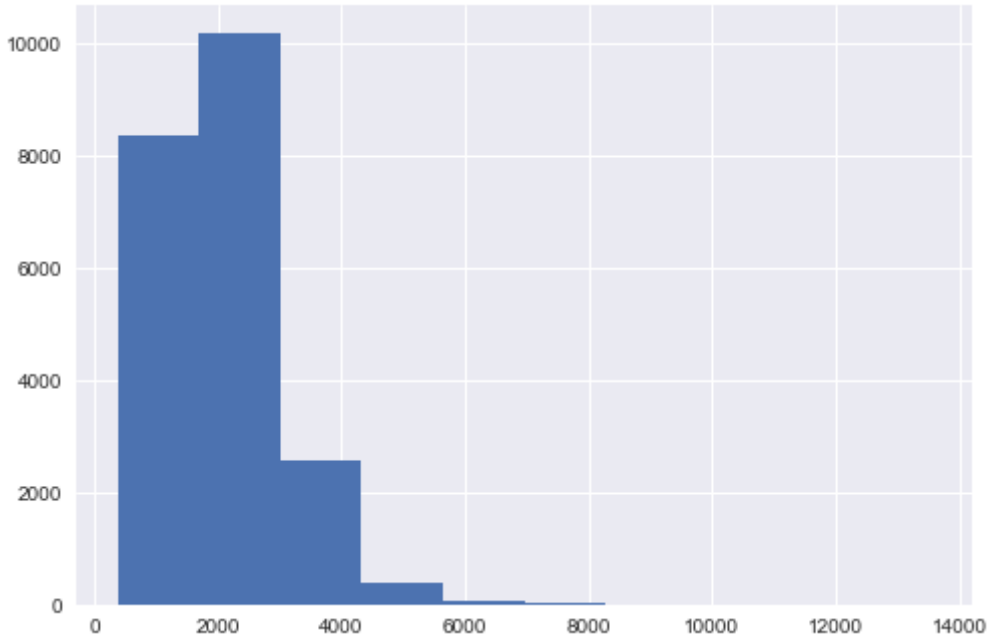
Out[4739]:

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



In [4740]:

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



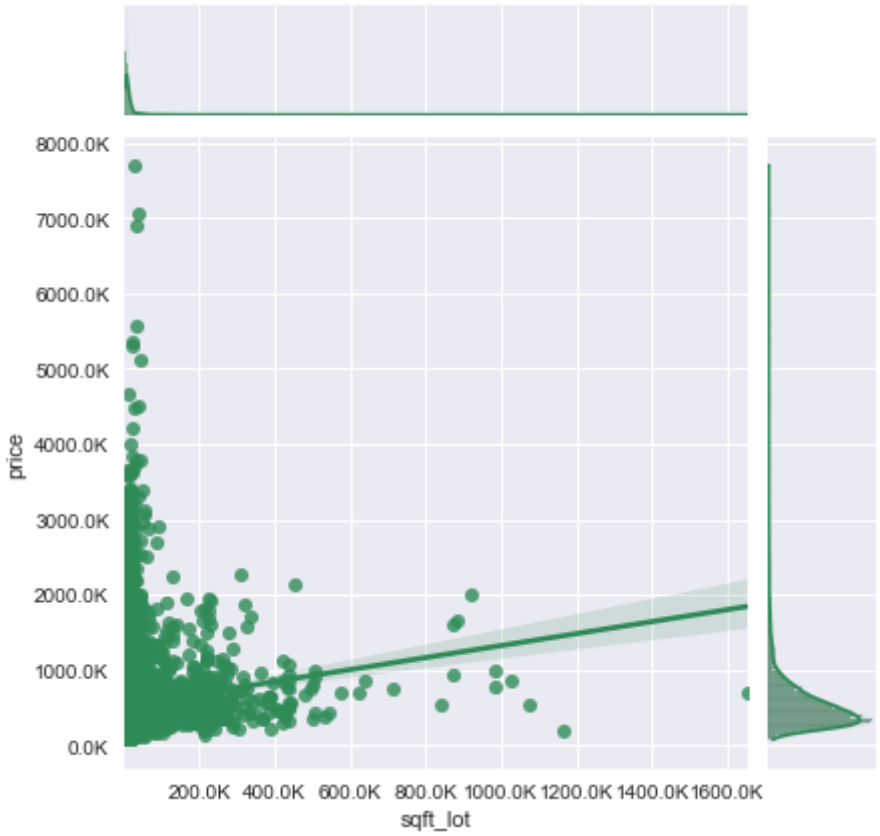
5.3.4 SQFT-lot

In [4741]:

```
1 def thousands(x, pos):
2     return "{:1.1f}K".format(x* 1e-3)
```

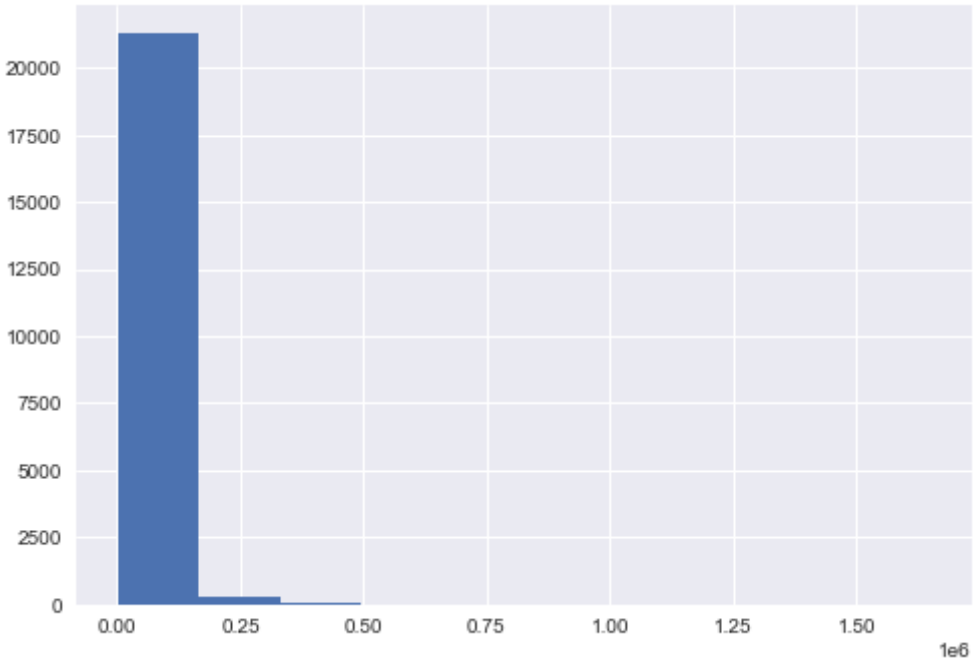
In [4742]:

```
fig3 = sns.jointplot(x="sqft_lot", y="price", data=df, kind="reg", color="seagreen")
fig3.ax_joint.yaxis.set_major_formatter(FuncFormatter(thousands))
fig3.ax_joint.xaxis.set_major_formatter(FuncFormatter(thousands))
```



In [4743]:

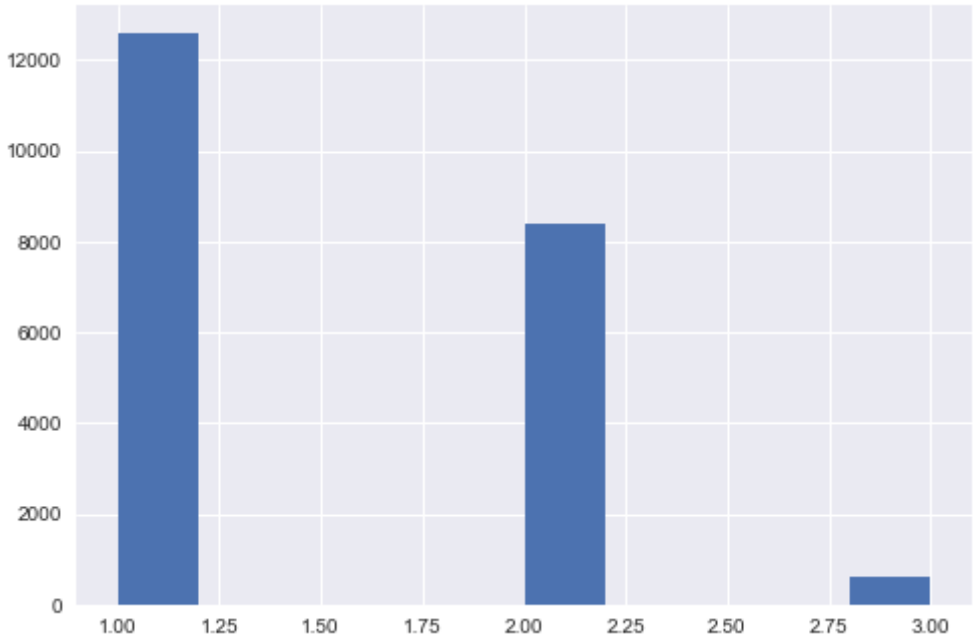
```
1 histogram("sqft_lot")
```



5.3.5 Floors

In [4744]:

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

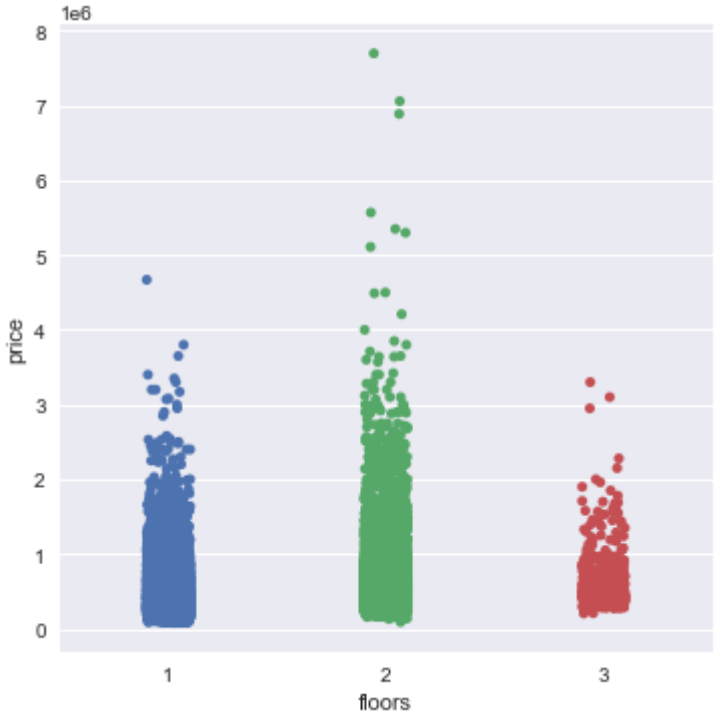


In [4745]:

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

Out[4745]:

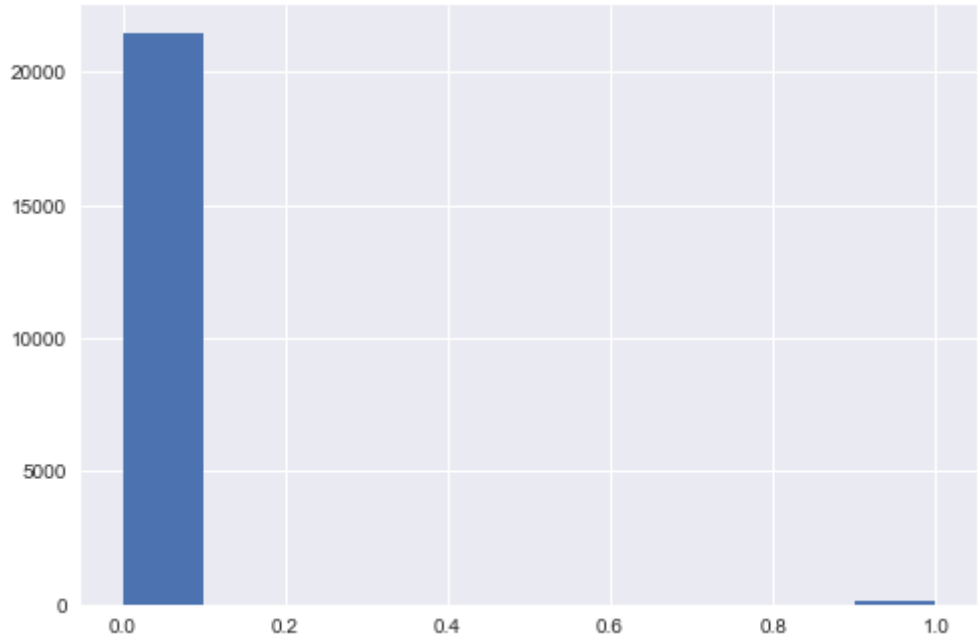
```
<seaborn.axisgrid.FacetGrid at 0x206a9486460>
```



5.3.6 Waterfront

In [4746]:

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

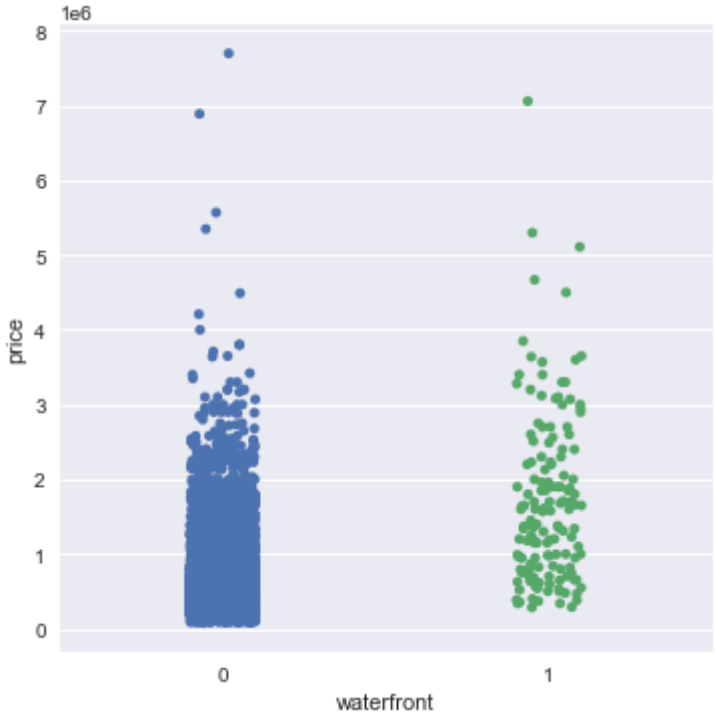


In [4747]:

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

Out[4747]:

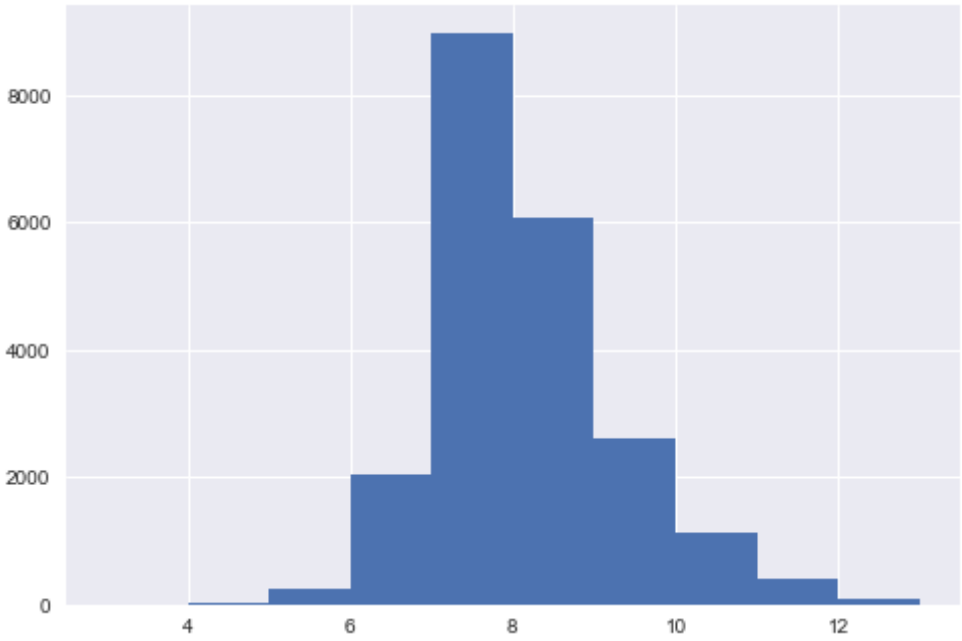
```
<seaborn.axisgrid.FacetGrid at 0x206016ebc70>
```



5.3.7 Grade

In [4748]:

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

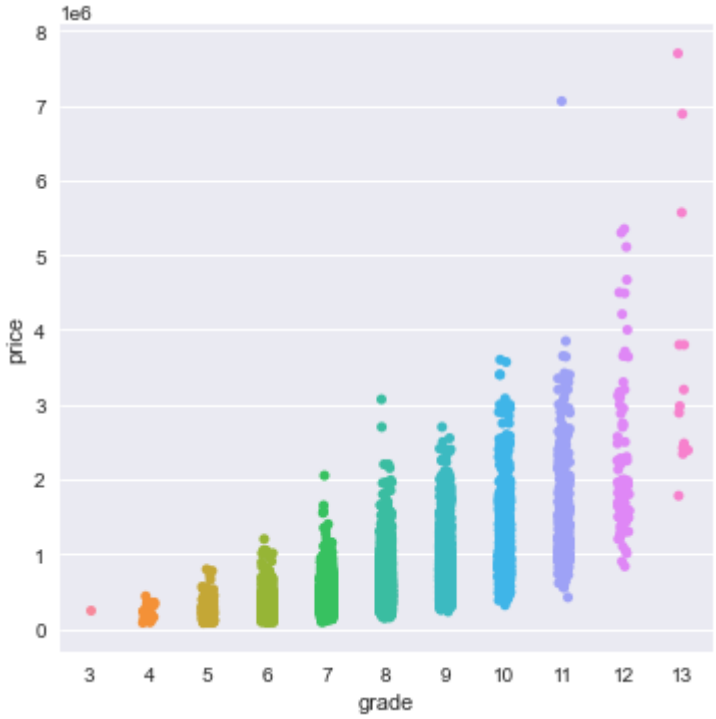


In [4749]:

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

Out[4749]:

```
<seaborn.axisgrid.FacetGrid at 0x20603894580>
```

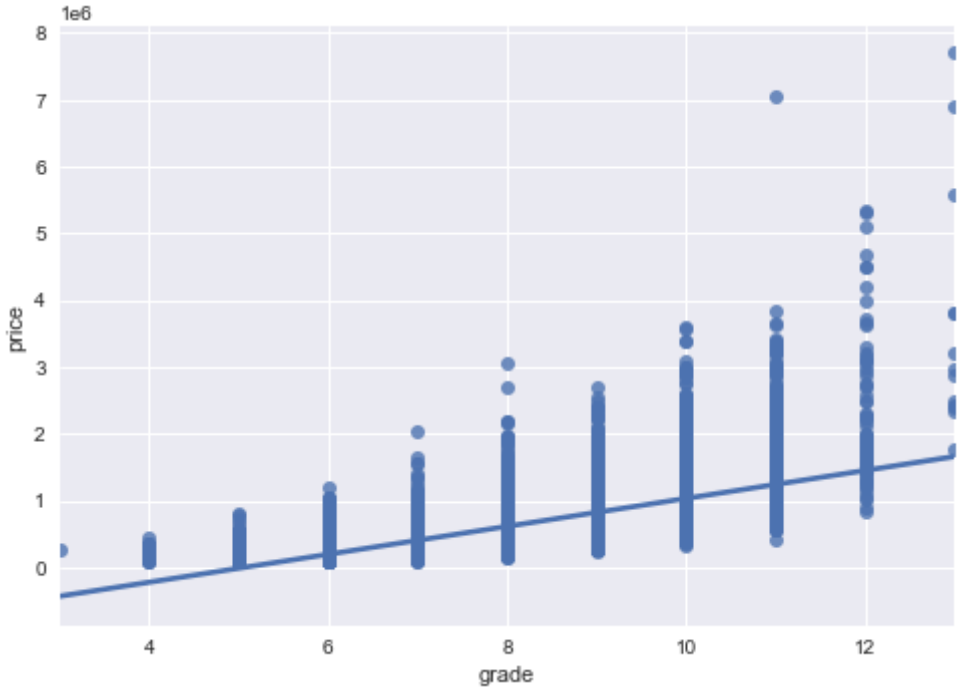


In [4750]:

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

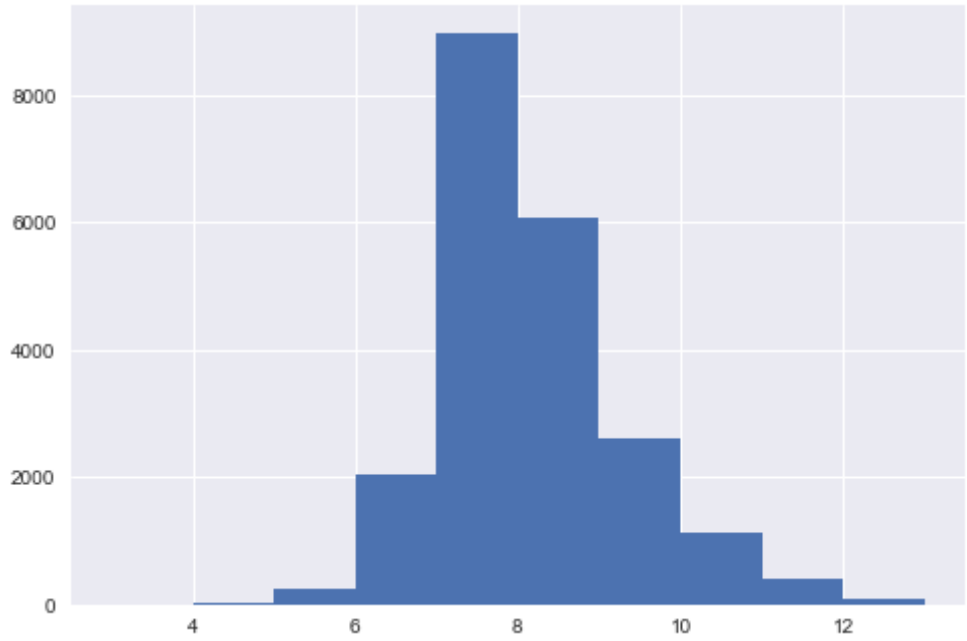
Out[4750]:

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



In [4751]:

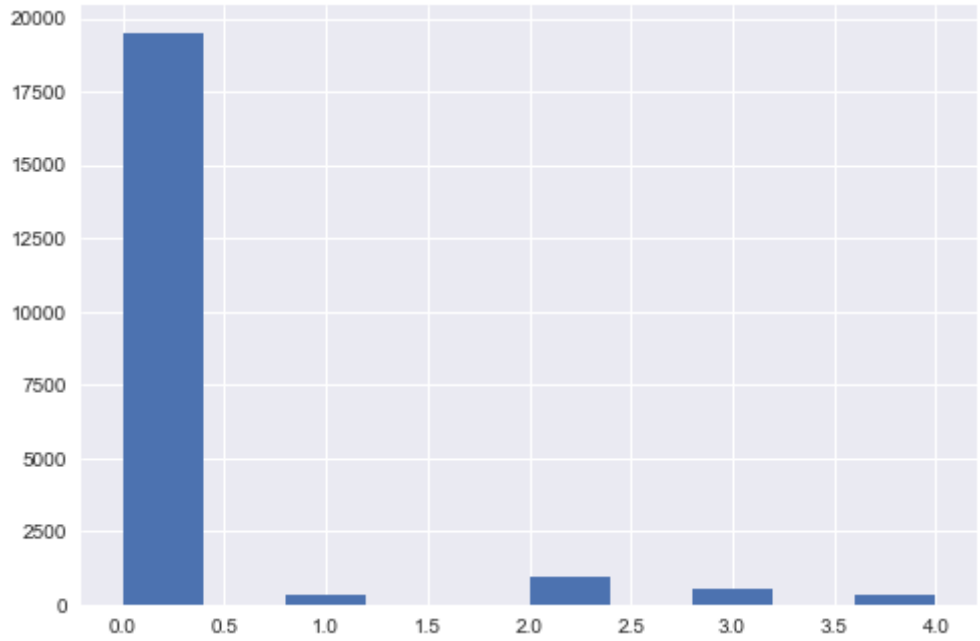
```
1 histogram("grade")
```



5.3.8 View

In [4752]:

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

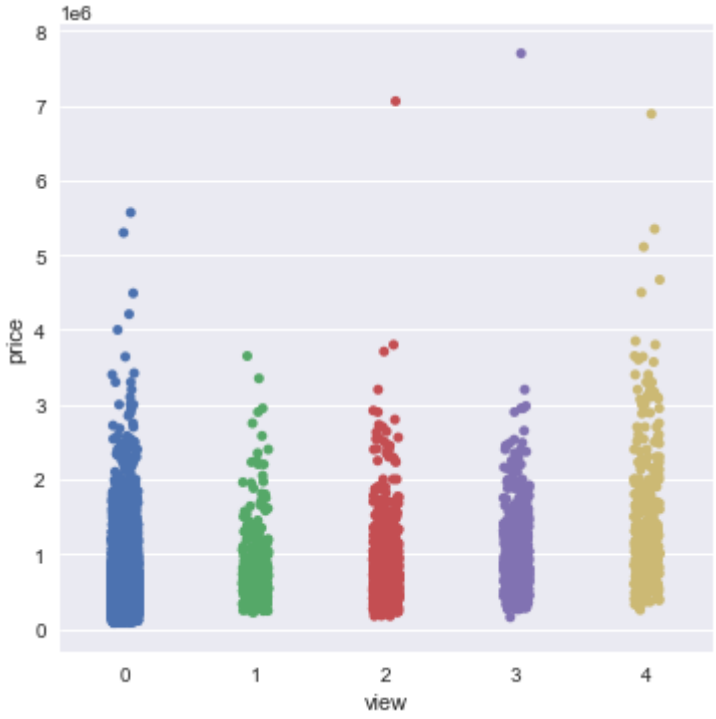


In [4753]:

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

Out[4753]:

```
<seaborn.axisgrid.FacetGrid at 0x2049a7aa130>
```

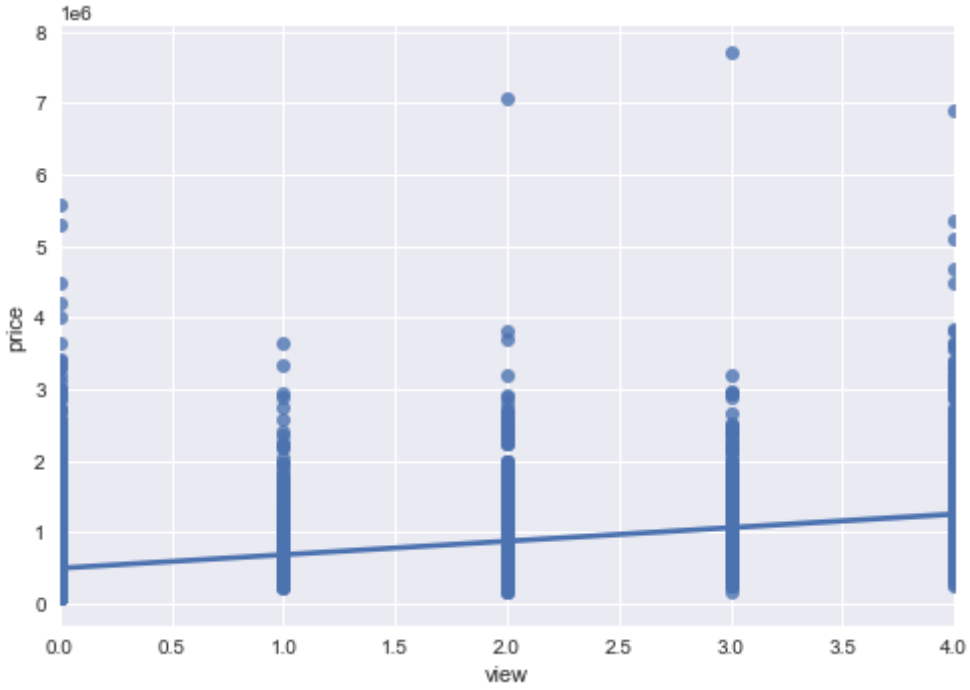


In [4754]:

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

Out[4754]:

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



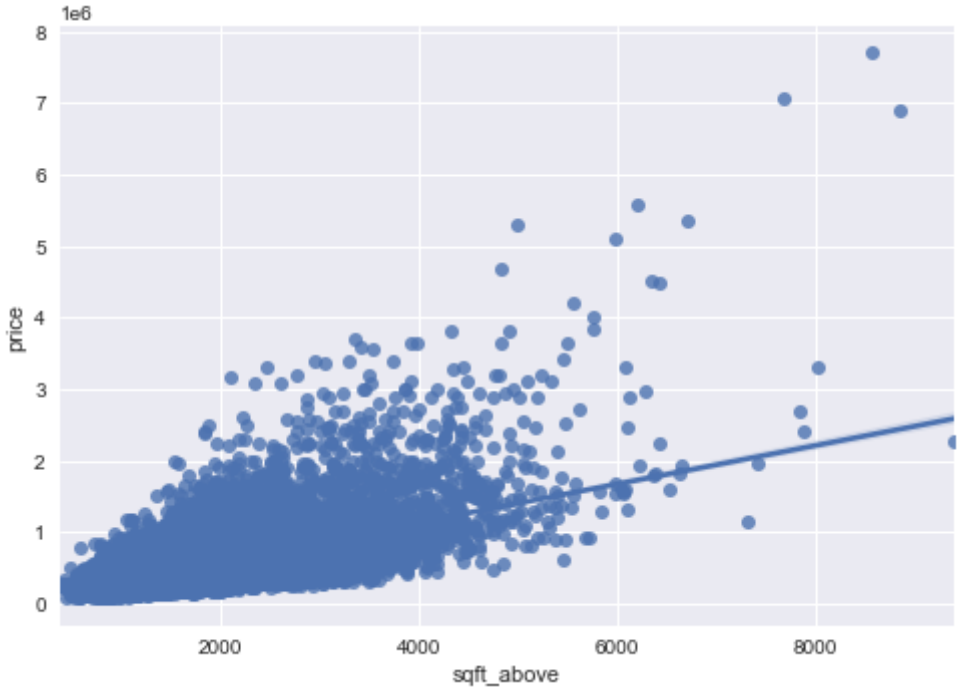
5.3.9 SQFT above

In [4755]:

```
1 reg("sqft_above")
```

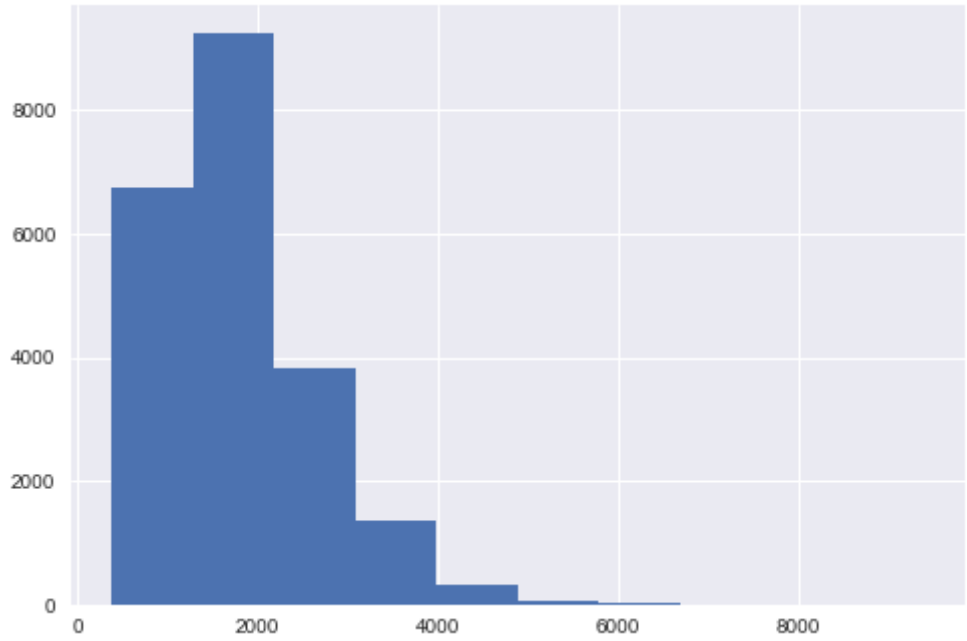
Out[4755]:

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



In [4756]:

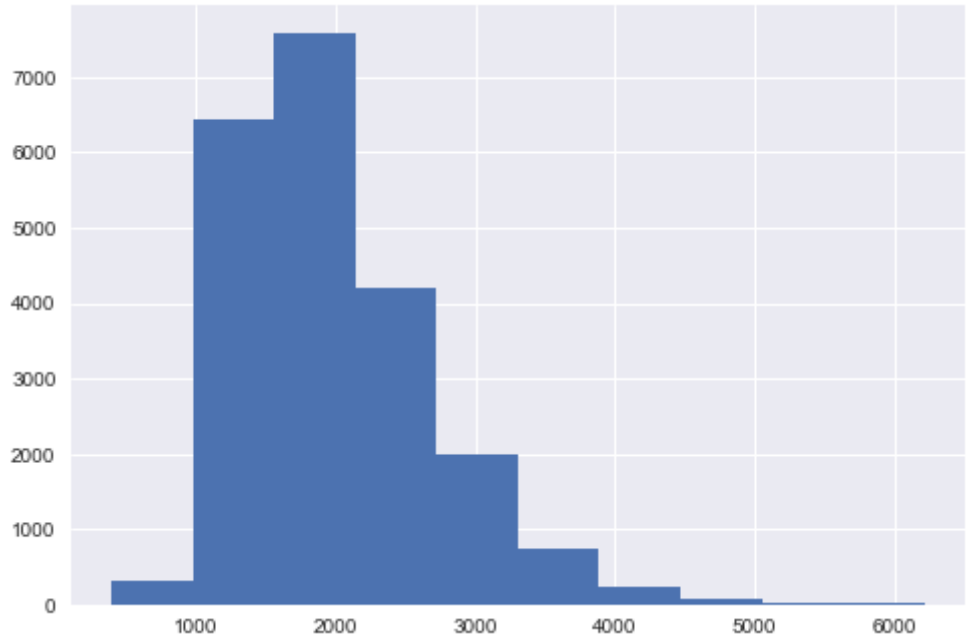
```
1 histogram("sqft_above")
```



5.3.10 SQFT-living15

In [4757]:

```
1 histogram("sqft_living15")
```

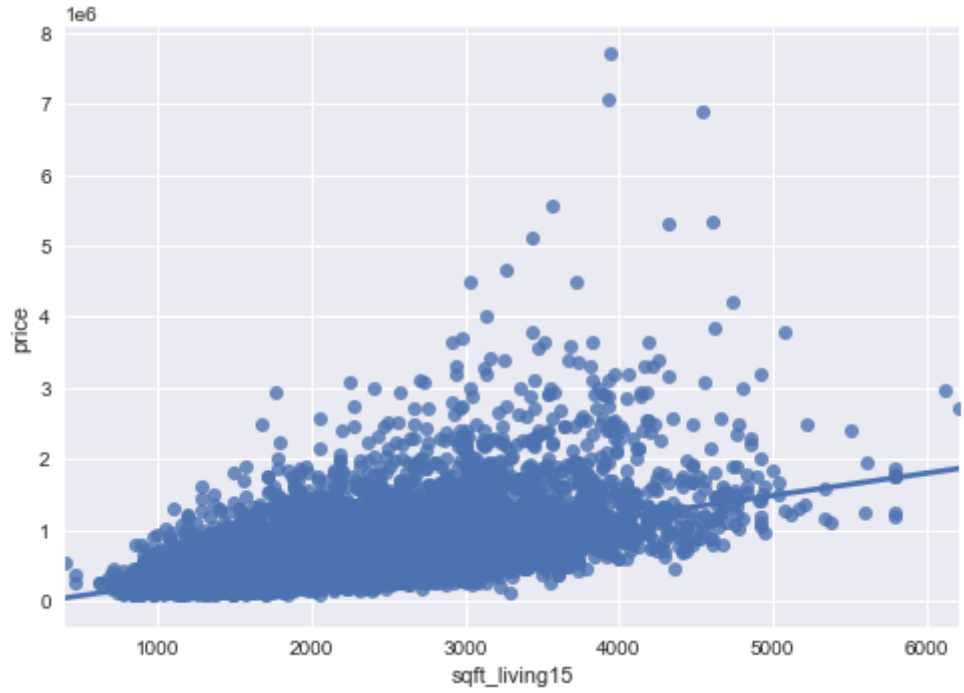


In [4758]:

```
1 reg("sqft_living15")
```

Out[4758]:

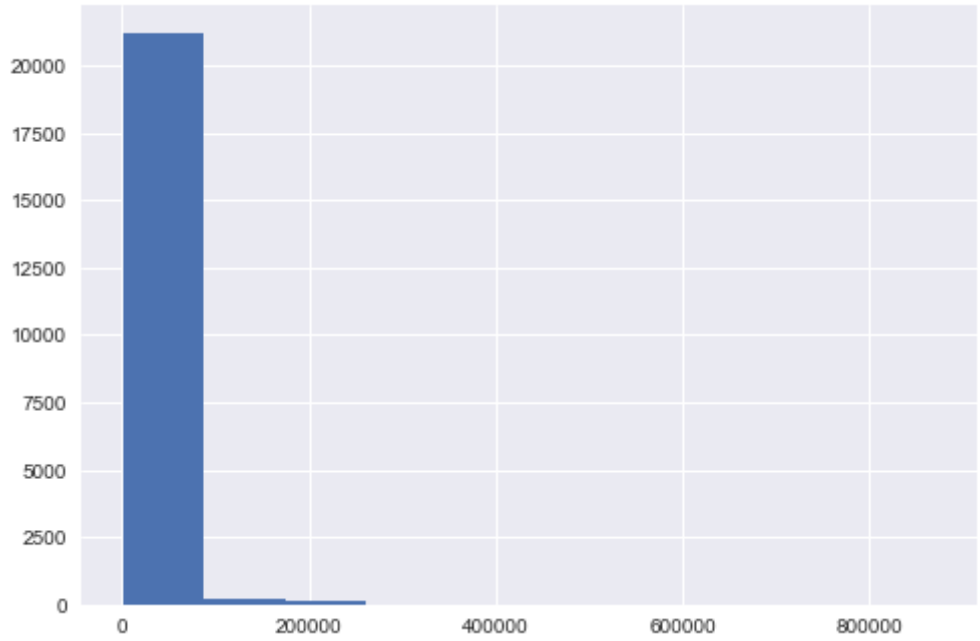
```
<AxesSubplot:xlabel='sqft_living15', ylabel='price'>
```



5.3.11 SQFT_lot15

In [4759]:

```
1 histogram("sqft_lot15")
```

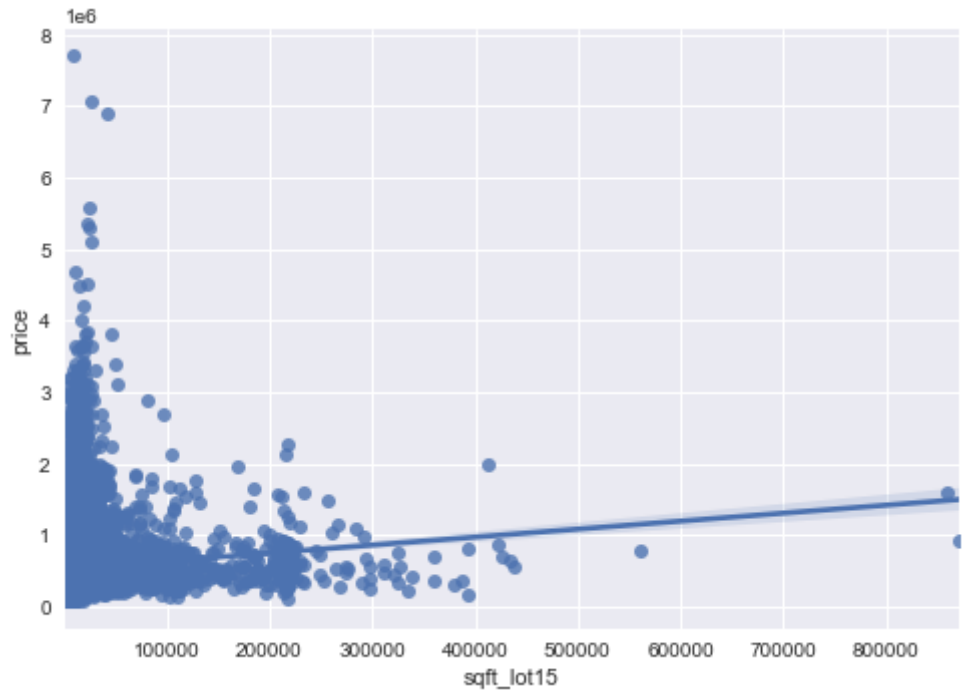


In [4760]:

```
1 reg("sqft_lot15")
```

Out[4760]:

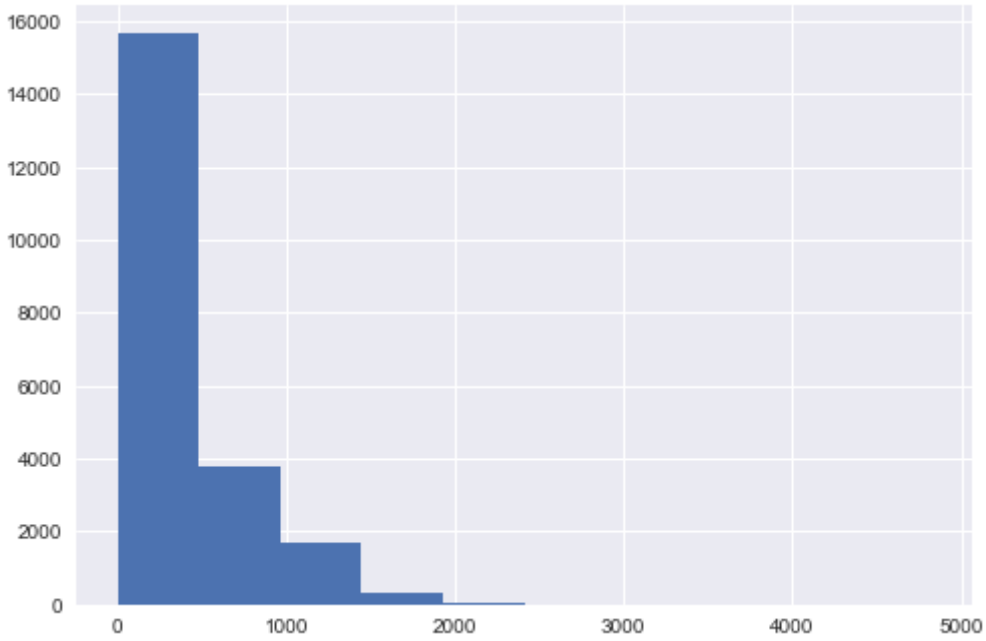
```
<AxesSubplot:xlabel='sqft_lot15', ylabel='price'>
```



5.3.12 SQFT_basement

In [4761]:

```
1 histogram("sqft_basement")
```

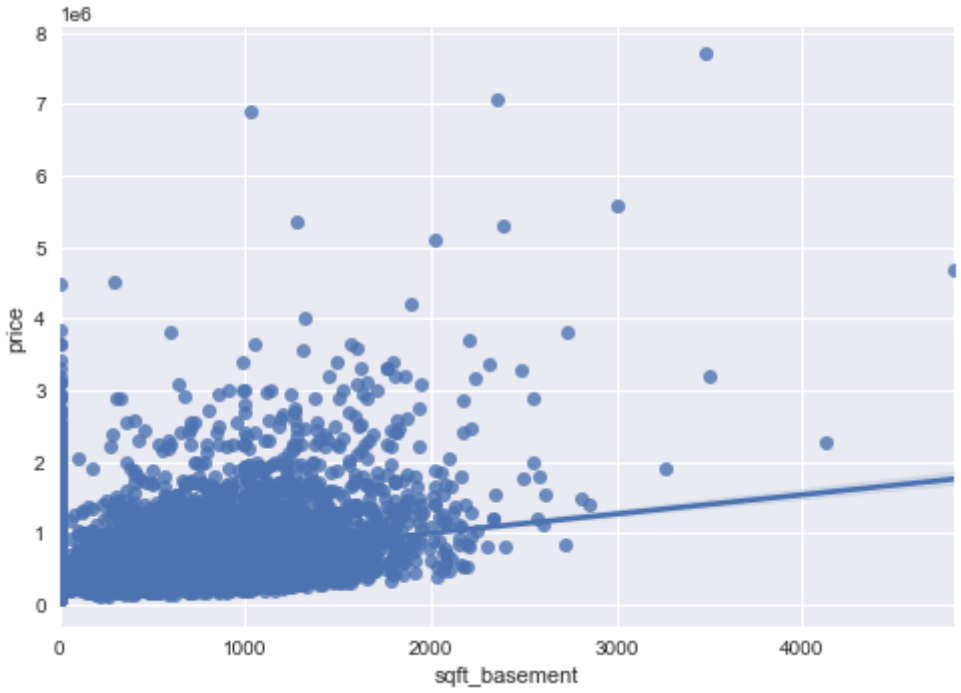


In [4762]:

```
1 reg("sqft_basement")
```

Out[4762]:

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



5.4 Shifting dependent variable to the end

In [4763]:

```
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 df
    df = df[cols + ["price"]]
    return df

df = move_price_col(df)
df
```

Out[4763]:

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	3	1.00	1180	5650	1	0
1	6414100192	12/9/2014	3	2.25	2570	7242	2	0
2	5631500400	2/25/2015	2	1.00	770	10000	1	0
3	2487200875	12/9/2014	4	3.00	1960	5000	1	0
4	1954400510	2/18/2015	3	2.00	1680	8080	1	0
...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	3	2.50	1530	1131	3	0
21593	6600060120	2/23/2015	4	2.50	2310	5813	2	0
21594	1523300141	6/23/2014	2	0.75	1020	1350	2	0
21595	291310100	1/16/2015	3	2.50	1600	2388	2	0
21596	1523300157	10/15/2014	2	0.75	1020	1076	2	0

21597 rows × 26 columns

6 Explore

In [4764]:

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

Out[4764]:

```
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 [4765]:

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

Out[4765]:

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

In [4766]:

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

Out[4766]:

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

In [4767]:

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

Out[4767]:

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

In [4768]:

```
categorical = ["zipcode"]
```

6.0.1 Taking care of numeric data

In [4769]:

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

In [4770]:

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

6.0.2 Taking care of categorical data

In [4771]:

```
1 df[categorical]
```

Out[4771]:

	zipcode
0	98178
1	98125
2	98028
3	98136
4	98074
...	...
21592	98103
21593	98146
21594	98144
21595	98027
21596	98144

21597 rows × 1 columns

In [4772]:

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

Out[4772]:

	zipcode_98002	zipcode_98003	zipcode_98004	zipcode_98005	zipcode_98006	zipcode_98
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	
21592	0.0	0.0	0.0	0.0	0.0	
21593	0.0	0.0	0.0	0.0	0.0	
21594	0.0	0.0	0.0	0.0	0.0	
21595	0.0	0.0	0.0	0.0	0.0	
21596	0.0	0.0	0.0	0.0	0.0	

21597 rows × 69 columns

In [4773]:

```
modeling_df = pd.concat([df.drop(categorical, axis=1), cat_df], axis=1)
```

In [4774]:

```
modeling_df.to_csv("kc_cleaned.csv", index=False)
```

6.0.3 Initial Model before Multicollinearity check

In [4775]:

```
def modeling(df, target="price"):
    features = " + ".join(df.drop(target, axis=1).columns)
    f = target + "~" + features
    model = smf.ols(formula = f, data=df).fit()
    display(model.summary())

    fig, ax = plt.subplots(ncols=2, figsize=(10, 6))
    sm.graphics.qqplot(model.resid, line="45", fit=True, ax=ax[0])
    sns.scatterplot(x=model.predict(df, transform=True), y=model.resid, ax=ax[1])
    ax[1].set_ylabel('Residuals')
    ax[1].set_xlabel('Predicted')
    return model

initial_model = modeling(modeling_df)

```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.816
Model:	OLS	**Adj. R-squared:**	0.812
Method:	Least Squares	**F-statistic:**	203.3
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:07:33	**Log-Likelihood:**	-2.8911e+05
No. Observations:	21597	**AIC:**	5.791e+05
Df Residuals:	21135	**BIC:**	5.828e+05
Df Model:	461		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	28.5748	26.935	1.061	0.289	-81.369	24.220

6.0.4 Multicollinearity check

In [4776]:

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

Out[4776]:

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
id	1.000000	0.001150	0.005162	-0.012241	-0.131911	0.019229	-0.0
bedrooms	0.001150	1.000000	0.514508	0.578212	0.032471	0.158065	-0.0
bathrooms	0.005162	0.514508	1.000000	0.755758	0.088373	0.520922	0.0
sqft_living	-0.012241	0.578212	0.755758	1.000000	0.173453	0.353372	0.1
sqft_lot	-0.131911	0.032471	0.088373	0.173453	1.000000	-0.008603	0.0
floors	0.019229	0.158065	0.520922	0.353372	-0.008603	1.000000	0.0
waterfront	-0.003599	-0.002127	0.063629	0.104637	0.021459	0.018321	1.0
view	0.011772	0.078354	0.186016	0.281715	0.075054	0.023711	0.3
condition	-0.023803	0.026496	-0.126479	-0.059445	-0.008830	-0.293463	0.0
grade	0.008188	0.356563	0.665838	0.762779	0.114731	0.473273	0.0
sqft_above	-0.010799	0.479386	0.686668	0.876448	0.184139	0.518037	0.0
sqft_basement	-0.004359	0.297229	0.278485	0.428660	0.015031	-0.231754	0.0
yr_built	0.021617	0.155670	0.507173	0.318152	0.052946	0.578549	-0.0
yr_renovated	-0.010612	0.017900	0.047177	0.051060	0.004979	-0.009505	0.0
zipcode	-0.008211	-0.154092	-0.204786	-0.199802	-0.129586	-0.097146	0.0
lat	-0.001798	-0.009951	0.024280	0.052155	-0.085514	0.029218	-0.0
long	0.020672	0.132054	0.224903	0.241214	0.230227	0.159481	-0.0
sqft_living15	-0.002701	0.393406	0.569884	0.756402	0.144763	0.296797	0.0
sqft_lot15	-0.138557	0.030690	0.088303	0.184342	0.718204	-0.012766	0.0
large_home	-0.009057	0.406468	0.172900	0.178745	0.008984	0.023155	0.0
how_old	-0.021617	-0.155670	-0.507173	-0.318152	-0.052946	-0.578549	0.0
renovated	-0.010621	0.017635	0.046742	0.050829	0.005091	-0.009580	0.0
has_basement	0.003495	0.158412	0.159863	0.201198	-0.034889	-0.246878	0.0
sqft_living_comparison	-0.004408	0.365105	0.342441	0.470834	0.047754	0.109445	0.0
price	-0.016772	0.308787	0.525906	0.701917	0.089876	0.237264	0.2

25 rows × 25 columns

In [4777]:

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

Out[4777]:

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	v
id	True	False	False	False	False	False	False	Fa
bedrooms	False	True	False	False	False	False	False	Fa
bathrooms	False	False	True	True	False	False	False	Fa
sqft_living	False	False	True	True	False	False	False	Fa
sqft_lot	False	False	False	False	True	False	False	Fa
floors	False	False	False	False	False	True	False	Fa
waterfront	False	False	False	False	False	False	True	Fa
view	False	False	False	False	False	False	False	T
condition	False	False	False	False	False	False	False	Fa
grade	False	False	False	True	False	False	False	Fa
sqft_above	False	False	False	True	False	False	False	Fa
sqft_basement	False	False	False	False	False	False	False	Fa
yr_built	False	False	False	False	False	False	False	Fa
yr_renovated	False	False	False	False	False	False	False	Fa
zipcode	False	False	False	False	False	False	False	Fa
lat	False	False	False	False	False	False	False	Fa
long	False	False	False	False	False	False	False	Fa
sqft_living15	False	False	False	True	False	False	False	Fa
sqft_lot15	False	False	False	False	False	False	False	Fa
large_home	False	False	False	False	False	False	False	Fa
how_old	False	False	False	False	False	False	False	Fa
renovated	False	False	False	False	False	False	False	Fa
has_basement	False	False	False	False	False	False	False	Fa
sqft_living_comparison	False	False	False	False	False	False	False	Fa
price	False	False	False	False	False	False	False	Fa

25 rows × 25 columns

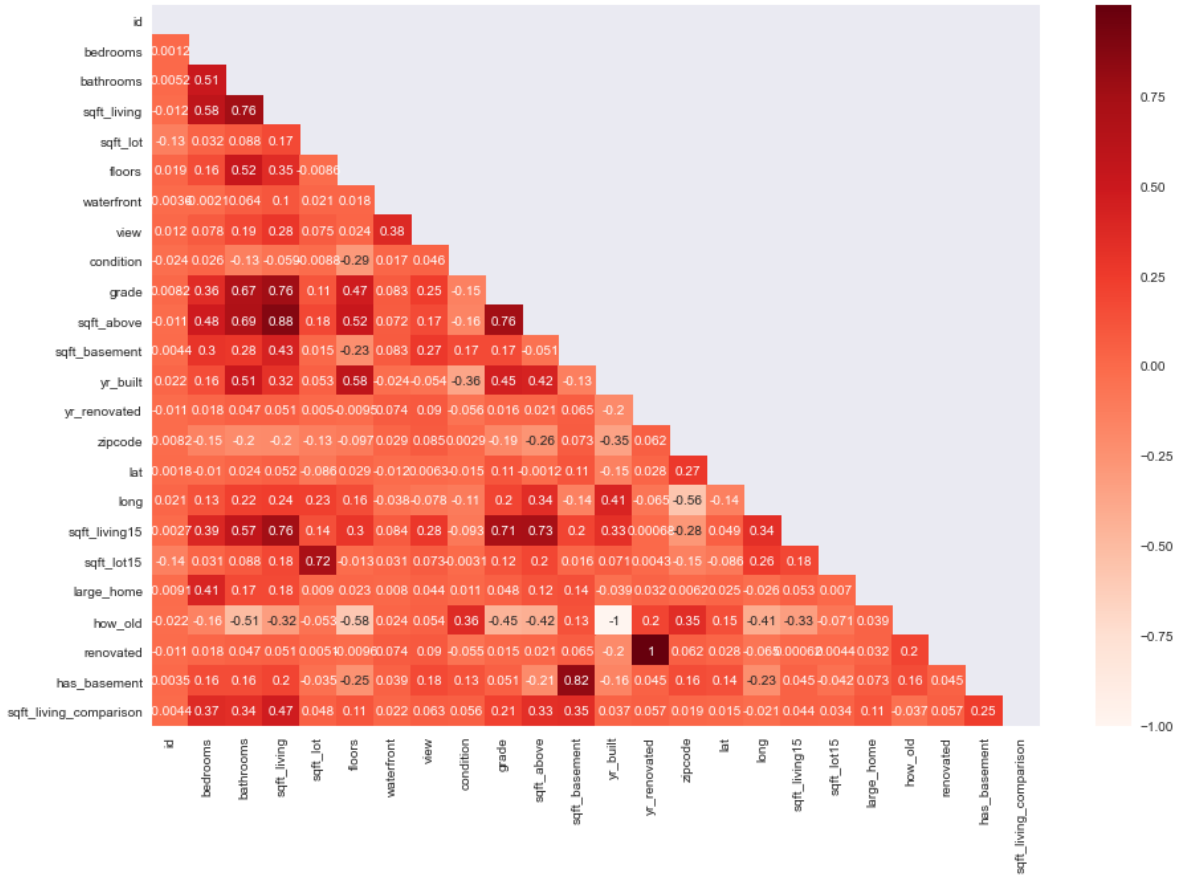
In [4778]:

```
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, mask=mask)
    return fig, ax

heatmap(df)
```

Out[4778]:

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



In [4779]:

```
## Making a dataframe with all the non-duplicate correlation coefficient
df_corr = df.corr().abs().stack().reset_index().sort_values(0, ascending=False)

df_corr["pairs"] = list(zip(df_corr["level_0"], df_corr["level_1"]))

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

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

df_corr.columns = ["cc"]

df_corr[(df_corr["cc"] > 0.75) & (df_corr["cc"] < 1)]

```

Out[4779]:

pairs	cc
(renovated, yr_renovated)	0.999968
(yr_renovated, renovated)	0.999968
(sqft_living, sqft_above)	0.876448
(sqft_above, sqft_living)	0.876448
(has_basement, sqft_basement)	0.820893
(sqft_basement, has_basement)	0.820893
(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
(sqft_living, bathrooms)	0.755758
(bathrooms, sqft_living)	0.755758

As we can see yr_renovated and sqft_living are highly correlated variables, we shall drop these columns from our main dataframe to avoid modeling issues

In [4780]:

```
modeling_df.drop(["yr_renovated", "sqft_living"], axis=1, inplace=True)
```

In [4781]:

```
modeling_df
```

Out[4781]:

	id	date	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condit
0	7129300520	10/13/2014	3	1.00	5650	1	0	0	
1	6414100192	12/9/2014	3	2.25	7242	2	0	0	
2	5631500400	2/25/2015	2	1.00	10000	1	0	0	
3	2487200875	12/9/2014	4	3.00	5000	1	0	0	
4	1954400510	2/18/2015	3	2.00	8080	1	0	0	
...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	3	2.50	1131	3	0	0	
21593	6600060120	2/23/2015	4	2.50	5813	2	0	0	
21594	1523300141	6/23/2014	2	0.75	1350	2	0	0	
21595	291310100	1/16/2015	3	2.50	2388	2	0	0	
21596	1523300157	10/15/2014	2	0.75	1076	2	0	0	

21597 rows × 92 columns

7 Modeling

7.1 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 [4782]:

```
modeling_df.drop(["id", "date"], axis=1, inplace=True)
modeling_df
```

Out[4782]:

	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
0	3	1.00	5650	1	0	0	3	7	1180
1	3	2.25	7242	2	0	0	3	7	2170
2	2	1.00	10000	1	0	0	3	6	770
3	4	3.00	5000	1	0	0	5	7	1050
4	3	2.00	8080	1	0	0	3	8	1680
...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1131	3	0	0	3	8	1530
21593	4	2.50	5813	2	0	0	3	8	2310
21594	2	0.75	1350	2	0	0	3	7	1020
21595	3	2.50	2388	2	0	0	3	8	1600
21596	2	0.75	1076	2	0	0	3	7	1020

21597 rows × 90 columns

In [4783]:

```
modeling(modeling_df)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.810
Model:	OLS	**Adj. R-squared:**	0.809
Method:	Least Squares	**F-statistic:**	1042.
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:07:40	**Log-Likelihood:**	-2.8946e+05
No. Observations:	21597	**AIC:**	5.791e+05
Df Residuals:	21508	**BIC:**	5.798e+05
Df Model:	88		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	13.4118	3.021	4.439	0.000	19.333	7.490

The p-value of long, sqft_lot and how_old is greater than our 0.05 alpha level. So we shall drop these columns as they are statistically insignificant.

In [4784]:

```
drop_cols = ["long", "sqft_lot15", "how_old"]
modeling_df.drop(drop_cols, axis=1, inplace=True)
modeling_df
```

Out[4784]:

	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
0	3	1.00	5650	1	0	0	3	7	1180
1	3	2.25	7242	2	0	0	3	7	2170
2	2	1.00	10000	1	0	0	3	6	770
3	4	3.00	5000	1	0	0	5	7	1050
4	3	2.00	8080	1	0	0	3	8	1680
...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1131	3	0	0	3	8	1530
21593	4	2.50	5813	2	0	0	3	8	2310
21594	2	0.75	1350	2	0	0	3	7	1020
21595	3	2.50	2388	2	0	0	3	8	1600
21596	2	0.75	1076	2	0	0	3	7	1020

21597 rows × 87 columns

In [4785]:

```
modeling(modeling_df)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.810
Model:	OLS	**Adj. R-squared:**	0.809
Method:	Least Squares	**F-statistic:**	1065.
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:07:42	**Log-Likelihood:**	-2.8946e+05
No. Observations:	21597	**AIC:**	5.791e+05
Df Residuals:	21510	**BIC:**	5.798e+05
Df Model:	86		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9.001e+06	2.99e+06	3.008	0.003	1.49e+07	3.14e+06

7.2 Removing outliers from the Dataset to complete homoscedasticity assumptions

In [4786]:

```
numeric_cols = ['bedrooms','bathrooms','sqft_lot','view','condition','grade',"sqft_base
```

In [4787]:

```
for col in numeric_cols:
    modeling_df = modeling_df[find_outliers(modeling_df[col]) == False]

modeling_df
```

Out[4787]:

	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
0	3	1.00	5650	1	0	0	3	7	1180
1	3	2.25	7242	2	0	0	3	7	2170
2	2	1.00	10000	1	0	0	3	6	770
3	4	3.00	5000	1	0	0	5	7	1050
4	3	2.00	8080	1	0	0	3	8	1680
...	...	...	...	...	...	...	...	...	...
21592	3	2.50	1131	3	0	0	3	8	1530
21593	4	2.50	5813	2	0	0	3	8	2310
21594	2	0.75	1350	2	0	0	3	7	1020
21595	3	2.50	2388	2	0	0	3	8	1600
21596	2	0.75	1076	2	0	0	3	7	1020

15188 rows × 87 columns

In [4788]:

```
def plot(df, target="price"):
    fig, ax = plt.subplots(nrows=len(df.columns), figsize=(8, 196))
    for i, cols in enumerate(df.columns):
        ax[i].scatter(df[cols], df[target])
        ax[i].set_xlabel(cols)
        ax[i].set_ylabel(target)
        ax[i].set_title(f"{col} vs. {target}")
    plt.tight_layout()
    return plt.show()

plot(modeling_df, target="price")
```

In [4789]:

```
1 model = modeling(modeling_df)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.827
Model:	OLS	**Adj. R-squared:**	0.826
Method:	Least Squares	**F-statistic:**	869.3
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:07:58	**Log-Likelihood:**	-1.9116e+05
No. Observations:	15188	**AIC:**	3.825e+05
Df Residuals:	15104	**BIC:**	3.831e+05
Df Model:	83		
Covariance Type:	nonrobust		

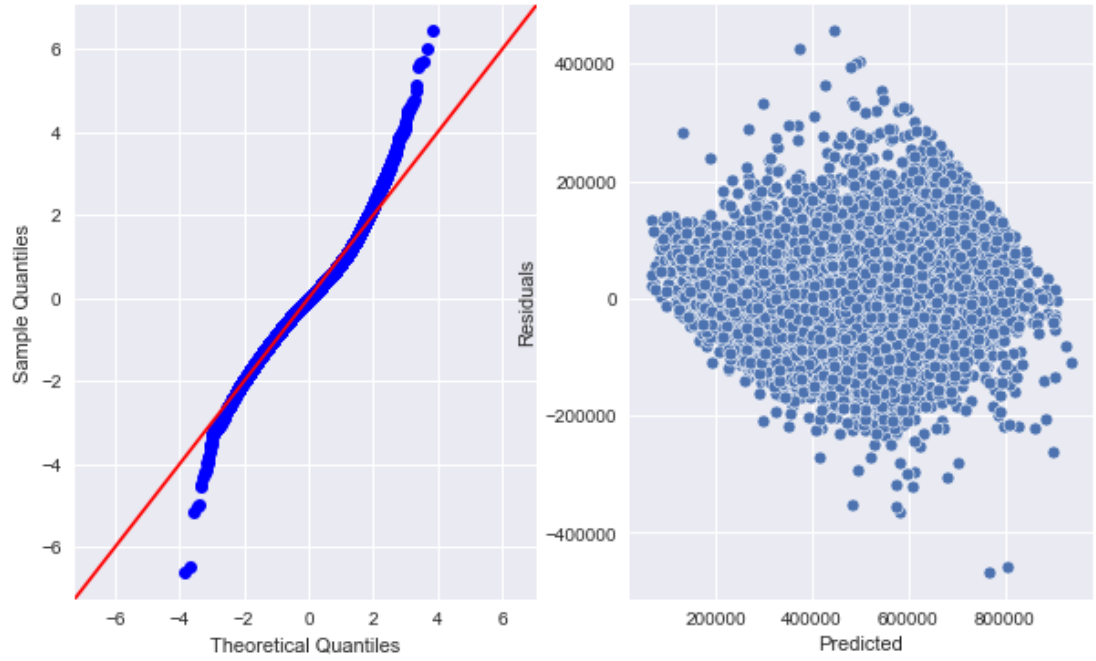
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.57e+05	1.76e+06	0.089	0.929	-3.28e+06	3.6e+06
has_basement[T.True]	5316.3825	2670.357	1.991	0.047	82.160	1.06e+04
bedrooms	-355.9379	1002.750	-0.355	0.723	-2321.449	1609.573
bathrooms	1.292e+04	1544.545	8.367	0.000	9895.406	1.6e+04
sqft_lot	1.1607	0.242	4.805	0.000	0.687	1.634
floors	-1.771e+04	1834.101	-9.658	0.000	-2.13e+04	-1.41e+04
waterfront	3.819e-06	2.23e-07	17.119	0.000	3.38e-06	4.26e-06
view	5.36e-06	3.13e-07	17.144	0.000	4.75e-06	5.97e-06
condition	2.12e+04	1049.047	20.210	0.000	1.91e+04	2.33e+04
grade	3.847e+04	1169.573	32.896	0.000	3.62e+04	4.08e+04
sqft_above	111.6518	2.471	45.177	0.000	106.807	116.496
sqft_basement	65.0496	4.358	14.926	0.000	56.507	73.592
yr_built	-398.4092	37.776	-10.546	0.000	-472.456	-324.363
lat	6004.5802	3.71e+04	0.162	0.871	-6.66e+04	7.87e+04
sqft_living15	31.1552	2.228	13.986	0.000	26.789	35.521
large_home	-1.538e-09	8.81e-11	-17.448	0.000	-1.71e-09	-1.37e-09
renovated	2.794e+04	3879.412	7.202	0.000	2.03e+04	3.55e+04
sqft_living_comparison	-1182.7929	1676.179	-0.706	0.480	-4468.306	2102.721
zipcode_98002	7812.0293	6873.015	1.137	0.256	-5659.912	2.13e+04
zipcode_98003	3851.5531	6309.896	0.610	0.542	-8516.606	1.62e+04
zipcode_98004	4.829e+05	1.42e+04	34.095	0.000	4.55e+05	5.11e+05
zipcode_98005	3.145e+05	1.38e+04	22.822	0.000	2.88e+05	3.42e+05
zipcode_98006	2.544e+05	1.11e+04	22.831	0.000	2.33e+05	2.76e+05
zipcode_98007	2.453e+05	1.37e+04	17.853	0.000	2.18e+05	2.72e+05

zipcode_98008	2.325e+05	1.31e+04	17.752	0.000	2.07e+05	2.58e+05
zipcode_98010	7.31e+04	1.21e+04	6.019	0.000	4.93e+04	9.69e+04
zipcode_98011	1.504e+05	1.79e+04	8.418	0.000	1.15e+05	1.85e+05
zipcode_98014	9.87e+04	1.82e+04	5.433	0.000	6.31e+04	1.34e+05
zipcode_98019	9.518e+04	1.74e+04	5.457	0.000	6.1e+04	1.29e+05
zipcode_98022	1.093e+04	8645.682	1.264	0.206	-6014.963	2.79e+04
zipcode_98023	-1.354e+04	5449.133	-2.485	0.013	-2.42e+04	-2862.555
zipcode_98024	1.276e+05	1.77e+04	7.221	0.000	9.29e+04	1.62e+05
zipcode_98027	2.325e+05	1.07e+04	21.659	0.000	2.12e+05	2.54e+05
zipcode_98028	1.331e+05	1.76e+04	7.547	0.000	9.86e+04	1.68e+05
zipcode_98029	2.356e+05	1.11e+04	21.217	0.000	2.14e+05	2.57e+05
zipcode_98030	7612.4918	6669.103	1.141	0.254	-5459.758	2.07e+04
zipcode_98031	1.318e+04	7116.492	1.852	0.064	-766.467	2.71e+04
zipcode_98032	-588.8188	8587.742	-0.069	0.945	-1.74e+04	1.62e+04
zipcode_98033	2.94e+05	1.51e+04	19.514	0.000	2.64e+05	3.24e+05
zipcode_98034	1.811e+05	1.62e+04	11.200	0.000	1.49e+05	2.13e+05
zipcode_98038	3.953e+04	5701.170	6.934	0.000	2.84e+04	5.07e+04
zipcode_98039	5.92e+05	4.29e+04	13.796	0.000	5.08e+05	6.76e+05
zipcode_98040	4.034e+05	1.29e+04	31.318	0.000	3.78e+05	4.29e+05
zipcode_98042	1.05e+04	5843.635	1.796	0.073	-959.040	2.19e+04
zipcode_98045	9.936e+04	9570.879	10.382	0.000	8.06e+04	1.18e+05
zipcode_98052	2.533e+05	1.47e+04	17.267	0.000	2.25e+05	2.82e+05
zipcode_98053	2.525e+05	1.56e+04	16.149	0.000	2.22e+05	2.83e+05
zipcode_98055	3.832e+04	8399.716	4.562	0.000	2.19e+04	5.48e+04
zipcode_98056	1.037e+05	9478.626	10.939	0.000	8.51e+04	1.22e+05
zipcode_98058	3.518e+04	7570.480	4.648	0.000	2.03e+04	5e+04
zipcode_98059	9.055e+04	8743.447	10.356	0.000	7.34e+04	1.08e+05
zipcode_98065	1.401e+05	1.04e+04	13.462	0.000	1.2e+05	1.61e+05
zipcode_98070	8.802e+04	1.83e+04	4.822	0.000	5.22e+04	1.24e+05
zipcode_98072	1.526e+05	1.79e+04	8.507	0.000	1.17e+05	1.88e+05
zipcode_98074	2.221e+05	1.3e+04	17.016	0.000	1.96e+05	2.48e+05
zipcode_98075	2.36e+05	1.29e+04	18.354	0.000	2.11e+05	2.61e+05
zipcode_98077	1.472e+05	2.11e+04	6.972	0.000	1.06e+05	1.89e+05
zipcode_98092	-1.776e+04	6199.883	-2.864	0.004	-2.99e+04	-5603.037
zipcode_98102	3.773e+05	1.56e+04	24.120	0.000	3.47e+05	4.08e+05
zipcode_98103	3.181e+05	1.47e+04	21.618	0.000	2.89e+05	3.47e+05
zipcode_98105	3.51e+05	1.53e+04	22.995	0.000	3.21e+05	3.81e+05
zipcode_98106	1.231e+05	1.03e+04	11.933	0.000	1.03e+05	1.43e+05
zipcode_98107	3.186e+05	1.49e+04	21.356	0.000	2.89e+05	3.48e+05
zipcode_98108	1.222e+05	1.14e+04	10.700	0.000	9.98e+04	1.45e+05

zipcode_98109	3.902e+05	1.58e+04	24.690	0.000	3.59e+05	4.21e+05
zipcode_98112	3.994e+05	1.42e+04	28.030	0.000	3.71e+05	4.27e+05
zipcode_98115	3.117e+05	1.5e+04	20.776	0.000	2.82e+05	3.41e+05
zipcode_98116	2.914e+05	1.17e+04	24.799	0.000	2.68e+05	3.14e+05
zipcode_98117	3.093e+05	1.51e+04	20.539	0.000	2.8e+05	3.39e+05
zipcode_98118	1.597e+05	1.04e+04	15.395	0.000	1.39e+05	1.8e+05
zipcode_98119	3.893e+05	1.49e+04	26.206	0.000	3.6e+05	4.18e+05
zipcode_98122	3.009e+05	1.3e+04	23.212	0.000	2.75e+05	3.26e+05
zipcode_98125	1.918e+05	1.62e+04	11.872	0.000	1.6e+05	2.23e+05
zipcode_98126	1.902e+05	1.06e+04	18.027	0.000	1.7e+05	2.11e+05
zipcode_98133	1.535e+05	1.67e+04	9.205	0.000	1.21e+05	1.86e+05
zipcode_98136	2.475e+05	1.11e+04	22.254	0.000	2.26e+05	2.69e+05
zipcode_98144	2.292e+05	1.21e+04	18.962	0.000	2.06e+05	2.53e+05
zipcode_98146	9.584e+04	9591.726	9.992	0.000	7.7e+04	1.15e+05
zipcode_98148	5.013e+04	1.16e+04	4.317	0.000	2.74e+04	7.29e+04
zipcode_98155	1.378e+05	1.74e+04	7.920	0.000	1.04e+05	1.72e+05
zipcode_98166	9.299e+04	9104.579	10.213	0.000	7.51e+04	1.11e+05
zipcode_98168	4.743e+04	9329.905	5.084	0.000	2.91e+04	6.57e+04
zipcode_98177	1.955e+05	1.75e+04	11.184	0.000	1.61e+05	2.3e+05
zipcode_98178	5.196e+04	9683.586	5.366	0.000	3.3e+04	7.09e+04
zipcode_98188	3.597e+04	9484.157	3.792	0.000	1.74e+04	5.46e+04
zipcode_98198	2.654e+04	7375.326	3.598	0.000	1.21e+04	4.1e+04
zipcode_98199	3.4e+05	1.44e+04	23.695	0.000	3.12e+05	3.68e+05

Omnibus:	946.351	**Durbin-Watson:**	1.994
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3025.198
Skew:	0.285	**Prob(JB):**	0.00
Kurtosis:	5.111	**Cond. No.**	1.24e+17

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.87e-23. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.



In [4790]:

```
1 model.pvalues[model.pvalues > 0.05]
```

Out[4790]:

```
Intercept                0.928721
bedrooms                 0.722623
lat                      0.871310
sqft_living_comparison   0.480418
zipcode_98002            0.255714
zipcode_98003            0.541607
zipcode_98022            0.206105
zipcode_98030            0.253698
zipcode_98031            0.063985
zipcode_98032            0.945337
zipcode_98042            0.072514
dtype: float64
```

from p values, we can see floors and lat are insignificant, hence we can get rid of them.

In [4791]:

```
1 modeling_df.drop(["floors", "lat"], axis=1, inplace=True)
```

In [4792]:

```
1 modeling_df
```

Out[4792]:

	bedrooms	bathrooms	sqft_lot	waterfront	view	condition	grade	sqft_above	sqft_bas
0	3	1.00	5650	0	0	3	7	1180	
1	3	2.25	7242	0	0	3	7	2170	
2	2	1.00	10000	0	0	3	6	770	
3	4	3.00	5000	0	0	5	7	1050	
4	3	2.00	8080	0	0	3	8	1680	
...	...	...	...	...	...	...	...	...	
21592	3	2.50	1131	0	0	3	8	1530	
21593	4	2.50	5813	0	0	3	8	2310	
21594	2	0.75	1350	0	0	3	7	1020	
21595	3	2.50	2388	0	0	3	8	1600	
21596	2	0.75	1076	0	0	3	7	1020	

15188 rows × 85 columns

In [4793]:

```
1 model = modeling(modeling_df)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.826
Model:	OLS	**Adj. R-squared:**	0.825
Method:	Least Squares	**F-statistic:**	884.3
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:08:01	**Log-Likelihood:**	-1.9120e+05
No. Observations:	15188	**AIC:**	3.826e+05
Df Residuals:	15106	**BIC:**	3.832e+05
Df Model:	81		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.126e+05	6.8e+04	10.475	0.000	5.79e+05	8.46e+05

In [4794]:

```
modeling_df.drop(["large_home","renovated", "bedrooms"], axis=1, inplace=True)
```

dropping large_home and renovated from the dataset since they are insignificant

In [4795]:

```
model = modeling(modeling_df)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.825
Model:	OLS	**Adj. R-squared:**	0.824
Method:	Least Squares	**F-statistic:**	903.7
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:08:03	**Log-Likelihood:**	-1.9123e+05
No. Observations:	15188	**AIC:**	3.826e+05
Df Residuals:	15108	**BIC:**	3.832e+05
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	8.253e+05	6.56e+04	12.586	0.000	6.97e+05	9.54e+05

In [4796]:

```
1 modeling_df
```

Out[4796]:

	bathrooms	sqft_lot	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_l
0	1.00	5650	0	0	3	7	1180	0.0	1
1	2.25	7242	0	0	3	7	2170	400.0	1
2	1.00	10000	0	0	3	6	770	0.0	1
3	3.00	5000	0	0	5	7	1050	910.0	1
4	2.00	8080	0	0	3	8	1680	0.0	1
...	...	...	...	...	...	...	...	...	
21592	2.50	1131	0	0	3	8	1530	0.0	2
21593	2.50	5813	0	0	3	8	2310	0.0	2
21594	0.75	1350	0	0	3	7	1020	0.0	2
21595	2.50	2388	0	0	3	8	1600	0.0	2
21596	0.75	1076	0	0	3	7	1020	0.0	2

15188 rows × 82 columns

7.2.1 Outliers based on price

In [4797]:

```
1 outliers_price = modeling_df[find_outliers(modeling_df["price"]) == False]
```

In [4798]:

```
1 modeling(outliers_price)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.824
Model:	OLS	**Adj. R-squared:**	0.823
Method:	Least Squares	**F-statistic:**	894.0
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:08:05	**Log-Likelihood:**	-1.9018e+05
No. Observations:	15121	**AIC:**	3.805e+05
Df Residuals:	15041	**BIC:**	3.811e+05
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	8.029e+05	6.5e+04	12.359	0.000	6.76e+05	9.3e+05

In [4799]:

```
1 outliers_price.drop(["sqft_living_comparison"], axis=1, inplace=True)
```

```
C:\Users\Vinayak Modgil\Anaconda3\lib\site-packages\pandas\core\frame.py:416
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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

In [4800]:

```
1 outliers_price["has_basement"].value_counts()
```

Out[4800]:

```
False    9996
True     5125
Name: has_basement, dtype: int64
```

7.3 Scaling

In [4801]:

```
cols_numeric = [col for col in outliers_price.columns if (col.startswith('zipcode')==Fa
cols_numeric
```

Out[4801]:

```
['bathrooms',
 'sqft_lot',
 'waterfront',
 'view',
 'condition',
 'grade',
 'sqft_above',
 'sqft_basement',
 'yr_built',
 'sqft_living15']
```

In [4802]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = outliers_price.copy()
df_scaled[cols_numeric] = scaler.fit_transform(df_scaled[cols_numeric])
```

In [4803]:

```
model = modeling(df_scaled)
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.824
Model:	OLS	**Adj. R-squared:**	0.824
Method:	Least Squares	**F-statistic:**	905.5
Date:	Fri, 23 Apr 2021	**Prob (F-statistic):**	0.00
Time:	13:08:08	**Log-Likelihood:**	-1.9018e+05
No. Observations:	15121	**AIC:**	3.805e+05
Df Residuals:	15042	**BIC:**	3.811e+05
Df Model:	78		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	2.647e+05	4296.427	61.616	0.000	2.56e+05	2.73e+05

In [4804]:

```
df_scaled.drop(["waterfront", "has_basement"], axis=1, inplace=True)
```

8 Interpret

Evaluate how well your work solves the stated business problem.

Questions to consider:

- How do you interpret the results?
- How well does your model fit your data? How much better is this than your baseline model?
- How confident are you that your results would generalize beyond the data you have?
- How confident are you that this model would benefit the business if put into use?

In [4805]:

```
coefficients = model.params.sort_values().to_frame('coefficients')
coefficients['abs'] = coefficients['coefficients'].abs()
coefficients.sort_values('abs', ascending=False, inplace=True)
```

In [4806]:

```
coefficients[~coefficients.index.str.startswith('zipcode')]
```

Out[4806]:

	coefficients	abs
Intercept	2.647292e+05	2.647292e+05
sqft_above	6.366172e+04	6.366172e+04
grade	3.041832e+04	3.041832e+04
sqft_basement	2.113136e+04	2.113136e+04
yr_built	-1.736469e+04	1.736469e+04
sqft_living15	1.690048e+04	1.690048e+04
condition	1.317124e+04	1.317124e+04
has_basement[T.True]	6.867278e+03	6.867278e+03
bathrooms	6.750996e+03	6.750996e+03
sqft_lot	5.678813e+03	5.678813e+03
waterfront	3.434058e-09	3.434058e-09
view	-2.246917e-09	2.246917e-09

We see that the top 3 parameters that affect the sale price of homes are has_basement, sqft_above and grade. We shall draw conclusions from the these three parameters.

In [4807]:

```
sqft_above_df = outliers_price.loc[:, ["sqft_above", "price"]]
sqft_above_df = sqft_above_df.groupby("sqft_above").mean().reset_index()
sqft_above_df
```

Out[4807]:

	sqft_above	price
0	480	119900.0
1	490	280300.0
2	560	249900.0
3	570	230000.0
4	580	279475.0
...	...	...
653	4073	510000.0
654	4140	720000.0
655	4180	673200.0
656	4190	460000.0
657	4210	884744.0

658 rows × 2 columns

In [4808]:

```
sqft_above_df["sqft_above"].max()
```

Out[4808]:

```
4210
```

In [4809]:

```
sqft_above_df["sqft_above"].min()
```

Out[4809]:

```
480
```

8.1 Data Visualization

In [4810]:

```
1 sns.lmplot(x="sqft_above", y="price", data=sqft_above_df)
```

Out[4810]:

```
<seaborn.axisgrid.FacetGrid at 0x205aa54bb20>
```

In [4811]:

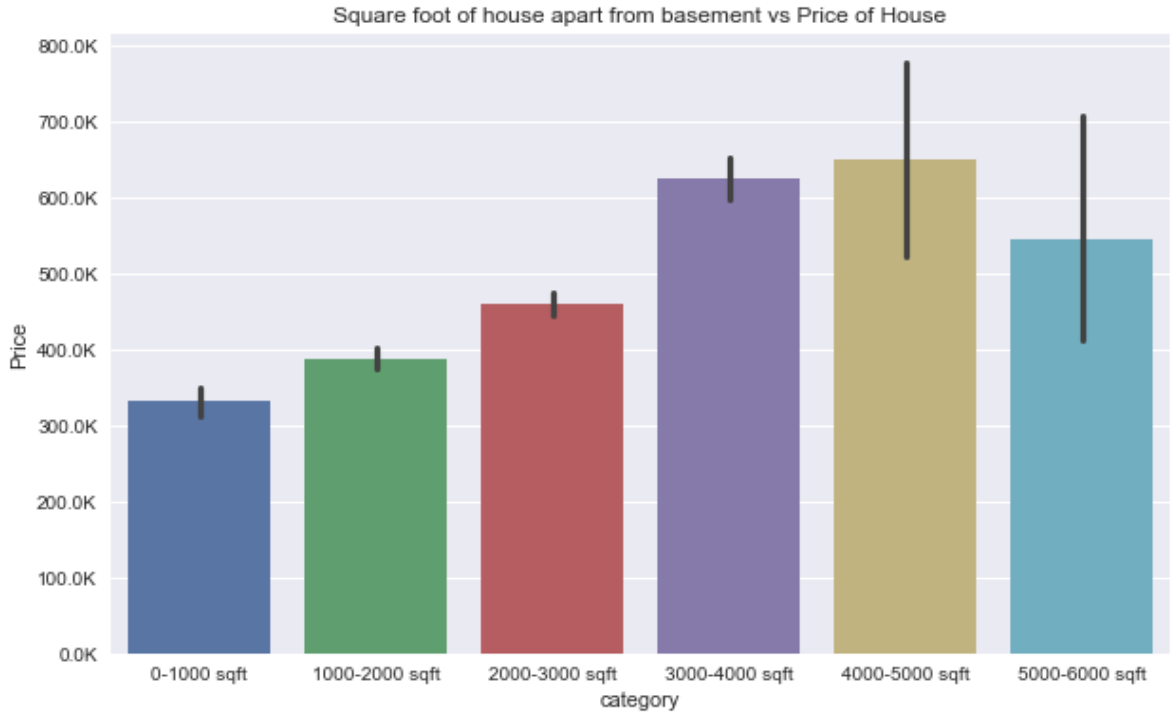
```
1  def types_of_houses(feature):
2      if feature > 0 and feature < 1000:
3          return "0-1000 sqft"
4      elif feature > 1000 and feature < 2000:
5          return "1000-2000 sqft"
6      elif feature > 2000 and feature < 3000:
7          return "2000-3000 sqft"
8      elif feature > 3000 and feature < 4000:
9          return "3000-4000 sqft"
10     elif feature > 4000 and feature < 5000:
11         return "4000-5000 sqft"
12     else:
13         return "5000-6000 sqft"
```

In [4813]:

```
sqft_above_df["category"] = sqft_above_df["sqft_above"].map(lambda x: types_of_houses(x
```

In [4814]:

```
fig, ax = plt.subplots(figsize=(10, 6))
order = ["0-1000 sqft", "1000-2000 sqft", "2000-3000 sqft", "3000-4000 sqft", "4000-500
sns.barplot(x="category", y="price", data=sqft_above_df, order=order, ci=95, ax=ax)
ax.set_label("SQFT above ground")
ax.set_ylabel("Price")
ax.set_title("Square foot of house apart from basement vs Price of House")
ax.yaxis.set_major_formatter(FuncFormatter(thousands))
plt.show()
```



In [4815]:

```
mean_price_per_category = sqft_above_df.groupby("category")["price"].mean().reset_index
mean_price_per_category
```

Out[4815]:

	category	price
0	0-1000 sqft	331466.561171
1	1000-2000 sqft	387755.069363
2	2000-3000 sqft	458789.411111
3	3000-4000 sqft	624586.888098
4	4000-5000 sqft	649588.800000
5	5000-6000 sqft	545343.019294

In [4816]:

```
outliers_price["has_basement"] = outliers_price["has_basement"].astype("bool")
```

```
<ipython-input-4816-31a9642a2216>:1: 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://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  outliers_price["has_basement"] = outliers_price["has_basement"].astype("bo
ol")
```

In [4857]:

```
plt.figure(figsize=(15, 8))
sns.lmplot(x="sqft_above", y="price", hue="has_basement", data=outliers_price,
           markers="*", palette="husl")
ax = plt.gca()
ax.set_xlabel("SQFT above ground")
ax.set_ylabel("Price of houses")
ax.set_title("Sqft vs price of houses for both with basements and without")
ax.yaxis.set_major_formatter(FuncFormatter(thousands))
```

```
<Figure size 1080x576 with 0 Axes>
```

In [4847]:

```
df_grade = outliers_price.loc[:, ["grade", "price"]]
df_grade
```

Out[4847]:

	grade	price
0	7	221900.0
1	7	538000.0
2	6	180000.0
3	7	604000.0
4	8	510000.0
...	...	...
21592	8	360000.0
21593	8	400000.0
21594	7	402101.0
21595	8	400000.0
21596	7	325000.0

15121 rows × 2 columns

In [4848]:

```
df_grade_sqft = pd.concat([df_grade, outliers_price["sqft_above"]], axis=1)
df_grade_sqft
```

Out[4848]:

	grade	price	sqft_above
0	7	221900.0	1180
1	7	538000.0	2170
2	6	180000.0	770
3	7	604000.0	1050
4	8	510000.0	1680
...	...	...	...
21592	8	360000.0	1530
21593	8	400000.0	2310
21594	7	402101.0	1020
21595	8	400000.0	1600
21596	7	325000.0	1020

15121 rows × 3 columns

In [4849]:

```
def grade_score(feature):
    if feature < 7:
        return "Low Score"
    elif feature > 8:
        return "High Score"
    else:
        return "Average Score"
```

In [4850]:

```
df_grade_sqft["categories"] = df_grade_sqft["grade"].map(lambda x: grade_score(x))
```

In [4851]:

```
df_grade_group = df_grade_sqft.groupby("categories")["price"].mean().reset_index()
df_grade_group
```

Out[4851]:

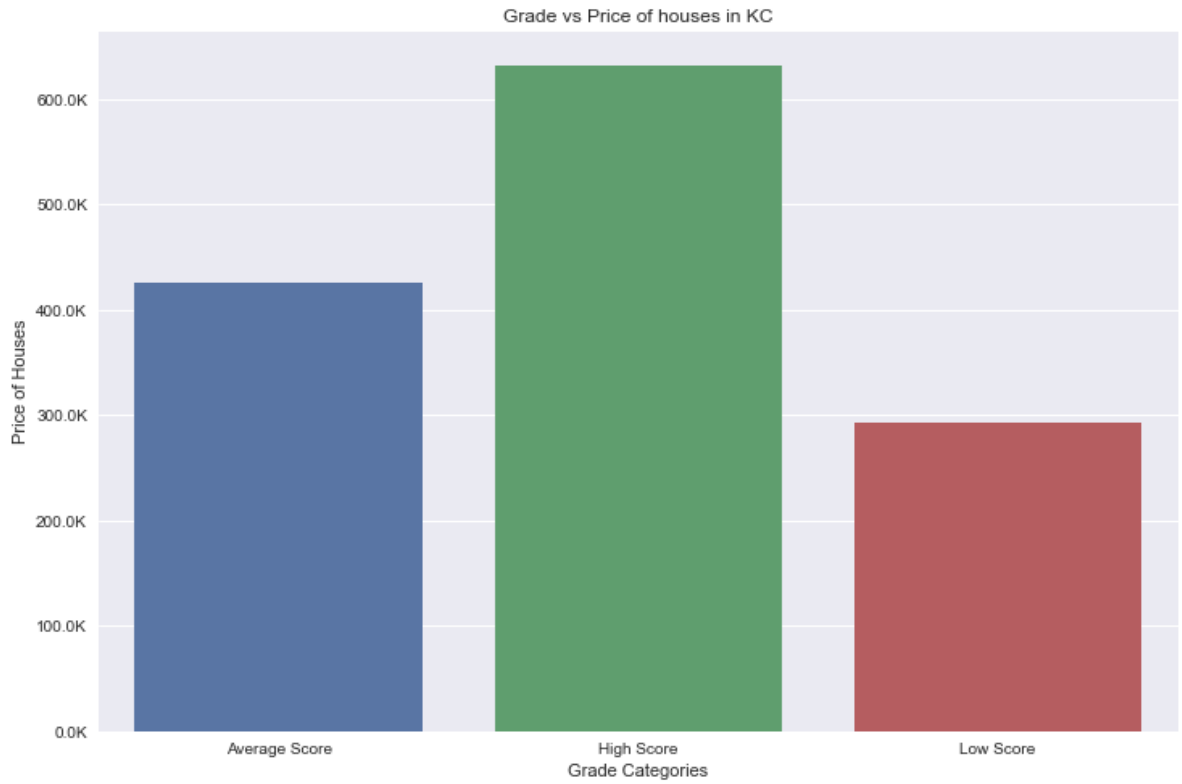
	categories	price
0	Average Score	425442.880252
1	High Score	632470.022140
2	Low Score	292656.627030

In [4852]:

```
fig, ax = plt.subplots(figsize=(12, 8))
sns.barplot(x="categories", y="price", data=df_grade_group, ax=ax)
ax.yaxis.set_major_formatter(FuncFormatter(thousands))
ax.set_xlabel("Grade Categories")
ax.set_ylabel("Price of Houses")
ax.set_title("Grade vs Price of houses in KC")
```

Out[4852]:

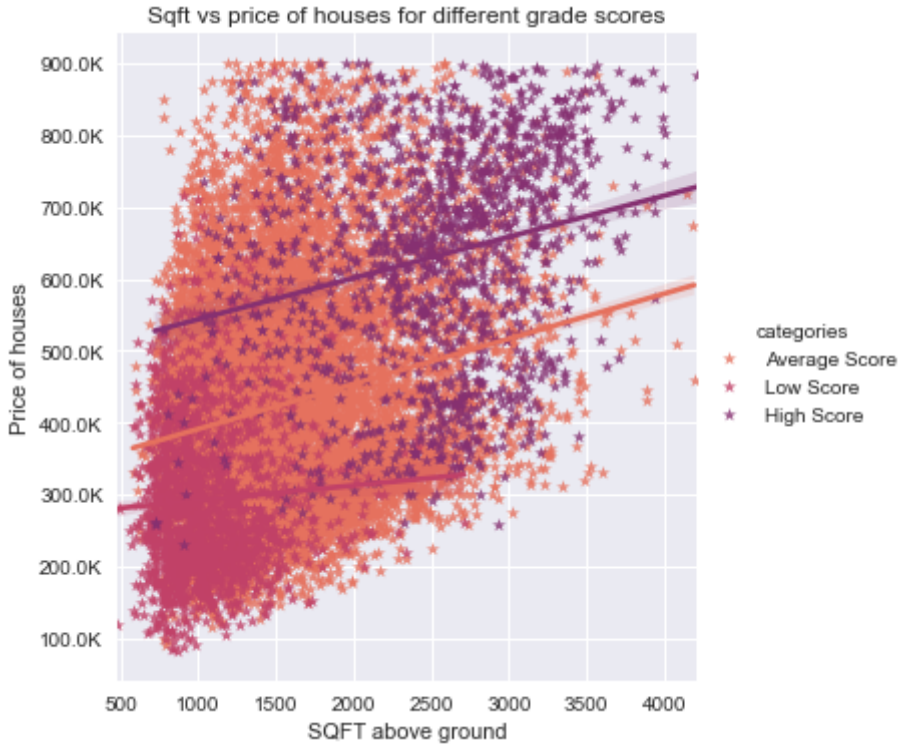
```
Text(0.5, 1.0, 'Grade vs Price of houses in KC')
```



In [4858]:

```
plt.figure(figsize=(15, 8))
sns.lmplot(x="sqft_above", y="price", hue="categories", data= df_grade_sqft,
           markers="*", palette = "flare")
ax = plt.gca()
ax.set_xlabel("SQFT above ground")
ax.set_ylabel("Price of houses")
ax.set_title("Sqft vs price of houses for different grade scores")
ax.yaxis.set_major_formatter(FuncFormatter(thousands))
```

```
<Figure size 1080x576 with 0 Axes>
```



9 Conclusion and Recommendation

Provide your conclusions about the work you've done, including any limitations or next steps.

Questions to consider:

- What would you recommend the business do as a result of this work?
- What are some reasons why your analysis might not fully solve the business problem?
- What else could you do in the future to improve this project?

9.1 The following conclusions can be drawn from our analysis:

- The area above the basement is key when purchasing houses. The main area to focus is on purchasing houses with adequate amount of area excluding the basement.
- Basements also do play a key role in valuing houses in King County. If a house has a basement, it tends to have more value.
- Grading done according to the KC grading system plays a significant role in valuing houses in KC.

9.2 Here are the recommendations based on our findings:

> - Firstly, we recommend purchasing houses with 4000-5000 sqft area above basement, as they hold the most value.
> - Secondly, purchasing houses with 4000-5000 sqft and having a basement will hold tremendous value in the future.
> - Lastly, Highly scored houses (according to KC grading system) but not the largest (in terms of sqft above ground) will be profitable while selling the houses to future house owners.

In [ ]:

```
1
```