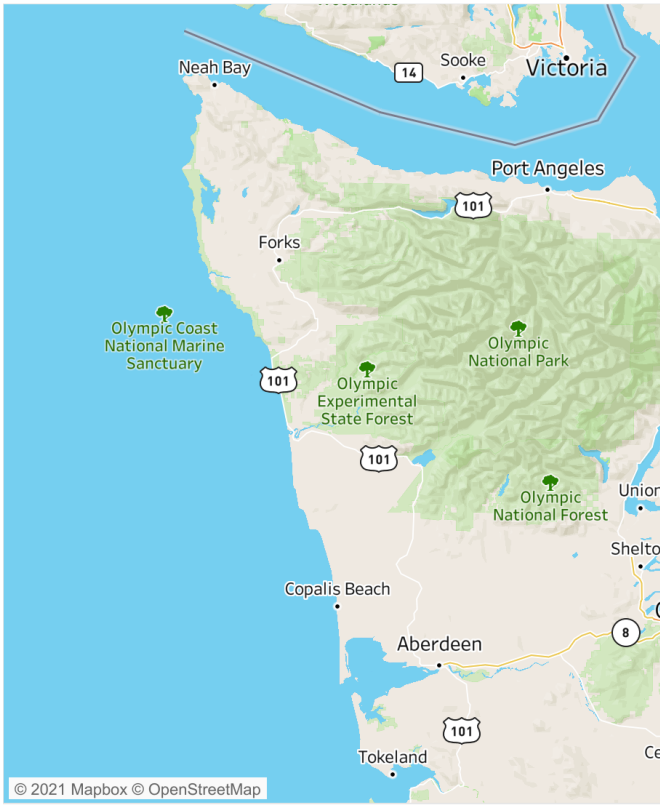
Contents

1 King County Housing Ch

King County Zip codes by Median Price



Using multiple linear regression analysis models to ir

Business problem:

King County home sales have been increasing as Se
live and work. Our real estate team has been tasked
ensure the price is accurate compared to the market

The model also guides clients on which features to p
suggestions within the owners control.

1.1 Data

- 21,597 rows by 21 columns
- CSV Formatted

1.2 Roadmap

Contents

- Scrub data to handle null values and duplicates
- Add additional features to better infer the price c
- Check for linearity and multicollinearity to make
- Perform outlier removal methods to better meet
- Use One Hot Encoding to handle categorical vai
- Provide accompanying visualizations to support
- Circle back to how the multiple linear regression

2 Scrub Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.set_option('display.max_columns'

df = pd.read_csv('data/kc_house_dat
df.head()
```

Out[1]:

	id	date	price	bedrooms	bat
0	7129300520	10/13/2014	221900.0	3	
1	6414100192	12/9/2014	538000.0	3	
2	5631500400	2/25/2015	180000.0	2	
3	2487200875	12/9/2014	604000.0	4	
4	1954400510	2/18/2015	510000.0	3	

2.1 Descriptions of columns

- **id** - unique identified for a house
- **dateDate** - house was sold
- **pricePrice** - is prediction target
- **bedroomsNumber** - of Bedrooms/House
- **bathroomsNumber** - of bathrooms/bedrooms
- **sqft_livingsquare** - footage of the home
- **sqft_lotsquare** - footage of the lot
- **floorsTotal** - floors (levels) in house
- **waterfront** - House which has a view to a water
- **view** - Quality of view
- **condition** - How good the condition is (Overall
- **grade** - overall grade given to the housing unit,
- **sqft_above** - square footage of house apart fror
- **sqft_basement** - square footage of the baseme
- **yr_built** - Built Year
- **yr_renovated** - Year when house was renovatec
- **zipcode** - zip

- **lat** - Latitude coordinate
- **long** - Longitude coordinate
- **sqft_living15** - The square footage of interior ho
- **sqft_lot15** - The square footage of the land lots

Contents

In [2]:
```
# Evaluating if type matches column

pd.set_option('display.float_format
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dty
---  ------         --------------  ---
 0   id             21597 non-null  int
 1   date           21597 non-null  obj
 2   price          21597 non-null  flo
 3   bedrooms       21597 non-null  int
 4   bathrooms      21597 non-null  flo
 5   sqft_living    21597 non-null  int
 6   sqft_lot       21597 non-null  int
 7   floors         21597 non-null  flo
 8   waterfront     19221 non-null  flo
 9   view           21534 non-null  flo
 10  condition      21597 non-null  int
 11  grade          21597 non-null  int
 12  sqft_above     21597 non-null  int
 13  sqft_basement  21597 non-null  obj
 14  yr_built       21597 non-null  int
 15  yr_renovated   17755 non-null  flo
 16  zipcode        21597 non-null  int
 17  lat            21597 non-null  flo
 18  long           21597 non-null  flo
 19  sqft_living15  21597 non-null  int
 20  sqft_lot15     21597 non-null  int
dtypes: float64(8), int64(11), object(2
memory usage: 3.5+ MB
```

- Date should be a datetime object
- Sqft basement should be an integer, not object

In [3]:
```
# Make date into datetime object

df['date'] = pd.to_datetime(df['dat
```

Contents

In [4]:
```
# Observe summary statistics

df.describe()
```

Out[4]:

	id	price	bedrooms	bathrc
count	21597.00	21597.00	21597.00	215
mean	4580474287.77	540296.57	3.37	
std	2876735715.75	367368.14	0.93	
min	1000102.00	78000.00	1.00	
25%	2123049175.00	322000.00	3.00	
50%	3904930410.00	450000.00	3.00	
75%	7308900490.00	645000.00	4.00	
max	9900000190.00	7700000.00	33.00	

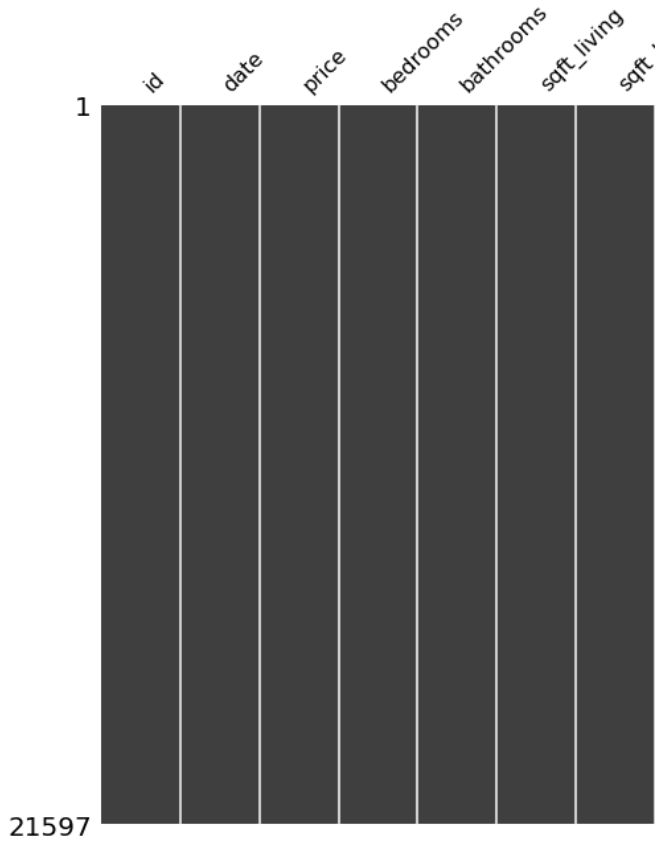
- ID is a random value so should not be evaluated
- Price has a large standard deviation and most lik
- Waterfront is a binary variable
- Floors, view, condition, and grade are discrete v
- Zipcode, latitude, and longitude are not continuc

2.2 Handling Null Values

Contents

In [5]:
```
# Visualize which columns contain n

import missingno
missingno.matrix(df)
```

Out[5]: <AxesSubplot:>



Waterfront, view, and yr_renovated contain null value

In [6]:
```
# Check how many null values are in

null = df.isna().sum()
null[null>1]
```

Out[6]:
```
waterfront      2376
view              63
yr_renovated    3842
dtype: int64
```

In [7]:
```
# Create formulat to impute null va

def impute_cat(df, col):
    '''
    Impute null value with value ba
    of occurring in the original co
    '''
    val_prob = dict(df[col].value_c
    prob = list(val_prob.values())
    val = list(val_prob.keys())
    np.random.choice(val, p=prob)
    df[col].fillna(np.random.choice
    return df
```

2.2.1 Fill in missing Values for 'view' c

Interpreting 'view' as quality of the view from the hor
of nature or the urban environmnt. A view of 0 would
natural characteristic.

Contents

Contents

In [8]:
```
print('Value Counts Normalized')
print(df['view'].value_counts(1, dr
print('-----------------------')
print('Value Counts Absolute')
print(df['view'].value_counts(dropn
```

```
Value Counts Normalized
0.00    0.90
2.00    0.04
3.00    0.02
1.00    0.02
4.00    0.01
nan     0.00
Name: view, dtype: float64
-----------------------
Value Counts Absolute
0.00    19422
2.00      957
3.00      508
1.00      330
4.00      317
nan        63
Name: view, dtype: int64
```

```
I have gone ahead and made the assu
I will use impute_cat to insert a v
distribution
- 90% chance of imputing a 0
- 2% chance of imputing a 1
- 4% of imputing a 2
- 2% of imputing a 3
- 1% of imputing a 4
```

Contents


```
In [9]:  1  impute_cat(df, 'view')
```

Out[9]:

	id	date	price	bedrooms	ba
0	7129300520	2014-10-13	221900.00	3	
1	6414100192	2014-12-09	538000.00	3	
2	5631500400	2015-02-25	180000.00	2	
3	2487200875	2014-12-09	604000.00	4	
4	1954400510	2015-02-18	510000.00	3	
...	...	...	...	...	
21592	263000018	2014-05-21	360000.00	3	
21593	6600060120	2015-02-23	400000.00	4	
21594	1523300141	2014-06-23	402101.00	2	
21595	291310100	2015-01-16	400000.00	3	
21596	1523300157	2014-10-15	325000.00	2	

21597 rows × 21 columns

Contents

In [10]:
```
# Confirm that there are no more nu

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

Out[10]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view              0
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
```

2.2.2 Fill in missing Values for 'yr_reno

Describes when the home was most recently renova

Contents

In [11]:
```
print('Value Counts Normalized')
print(df['yr_renovated'].value_coun
print('------------------------')
print('Value Counts Absolute')
print(df['yr_renovated'].value_coun
```

```
Value Counts Normalized
0.00       0.79
nan        0.18
2014.00    0.00
2003.00    0.00
2013.00    0.00
           ...
1944.00    0.00
1948.00    0.00
1976.00    0.00
1934.00    0.00
1953.00    0.00
Name: yr_renovated, Length: 71, dtype:
------------------------
Value Counts Absolute
0.00       17011
nan         3842
2014.00       73
2003.00       31
2013.00       31
           ...
1944.00        1
1948.00        1
1976.00        1
1934.00        1
1953.00        1
Name: yr_renovated, Length: 71, dtype:
```

Most of the values in yr_renovated are either 0 or nar
based on probability.

Since this value has ~79% 0 values, I will most likely

Contents

In [12]:
```
df[df['yr_renovated']>0].describe()
```

Out[12]:

	id	price	bedrooms	bathrc
count	744.00	744.00	744.00	74
mean	4418716401.67	768901.89	3.46	
std	2908265353.00	627125.79	1.07	
min	3600057.00	110000.00	1.00	
25%	1922984893.00	412250.00	3.00	
50%	3899100167.50	607502.00	3.00	
75%	7014200237.50	900000.00	4.00	
max	9829200250.00	7700000.00	11.00	

The most recent renovation took place in 2015. The

In [13]:
```
# Going to assume that a null value
# This is equivalent to a 0 which i

df['yr_renovated'].fillna(0, inplac
```

In [14]:
```
# Confirm that there are no more nu

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

Out[14]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
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

2.2.3 Fill in missing Values for 'waterfr

Waterfront is a binary variable. 1 means the home ha

- Based on national home prices, waterfront prop
  a lake, and enjoy easy access to bodies of wate
- I'd like to explore if homes prices at greater than
- I can then use this finding to subset the data bas

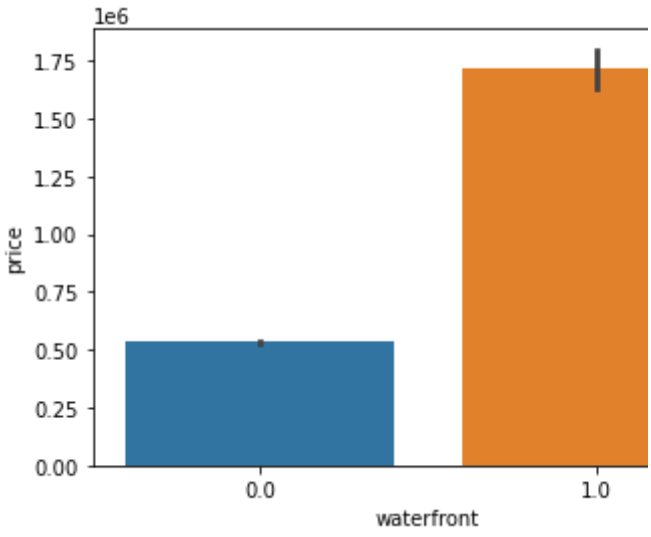
In [15]:
```
# Check if mean of price is greater

df.groupby('waterfront')[['price',
```

Out[15]:

waterfront	price	view
0.00	532641.99	0.20
1.00	1717214.73	3.76

As expected, waterfront homes have a greater mean
is easier to see the ocean, lake, or river which enhan

In [16]:
```
import seaborn as sns

sns.barplot(data=df, x='waterfront'
```



Clearly, waterfront homes are more expensive than n

Contents

In [17]:
```
mplus_water =len(df[(df['price']>10
print(f'Number of houses over $1,00
mminus_water = len(df[(df['price']<
print(f'Number of houses under $1,0
```

```
Number of houses over $1,000,000 with w
Number of houses under $1,000,000 with
```

In [18]:
```
# Observe ratio of waterfront homes

pd.set_option('display.float_format

print('$1M+ with waterfront');
print(df.loc[df['price']>1000000]['
print('---------------------------
# Prob of having waterfront view fo
print('$1M- with waterfront');
print(df.loc[df['price']<1000000]['
```

```
$1M+ with waterfront
0.00000   0.92666
1.00000   0.07334
Name: waterfront, dtype: float64
-----------------------------------
$1M- with waterfront
0.00000   0.99726
1.00000   0.00274
Name: waterfront, dtype: float64
```

- 7.3% of homes priced over $1 million have wate
- 0.02% of homes priced under $1 million have w
- As a result, I am going to subset the data by a $ because the more expensive homes are far mor waterfront view.

In [19]:
```
# Subset the data into two slices b

df_1mplus=df.loc[df['price']>100000
df_1mminus=df.loc[df['price']<10000
```

In [20]:
```
# Use impute_cat on homes over $1,0

df_1mplus =impute_cat(df_1mplus, 'w
```

```
/Users/ethankunin/opt/anaconda3/envs/le
yWarning:
A value is trying to be set on a copy o

See the caveats in the documentation: h
-a-view-versus-a-copy (https://pandas.p
a-copy)
  return super().fillna(
```

Contents

In [21]:
```
# Simply fill the missing waterfron
# probability of them having a wate

df_1mminus['waterfront'] =df_1mminu
```

```
<ipython-input-21-3a74373b56b6>:4: Sett
A value is trying to be set on a copy o
Try using .loc[row_indexer,col_indexer]

See the caveats in the documentation: h
-a-view-versus-a-copy (https://pandas.p
a-copy)
  df_1mminus['waterfront'] =df_1mminus[
```

In [22]:
```
# Join the data back together

df=pd.concat([df_1mminus, df_1mplus
```

In [23]:
```
# Confirm there are no more missing

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

Out[23]:
```
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
```

Missing values are handled, next step is to address c

2.3 Handling Duplicates

In [24]:
```
df[df.duplicated()]
```

Out[24]:

id	date	price	bedrooms	bathrooms	sqft_living

Initially, it appears that we don't have any duplicates matching. It may be produnt to check if there are any

In [25]:
```
# Check duplicats by 'id'
display(len(df))
df[df.duplicated(subset=['id'],keep

```

```
21565
```

Out[25]:

	id	date	price	bedrooms
93	6021501535	2014-07-25	430000.00000	3
94	6021501535	2014-12-23	700000.00000	3
324	7520000520	2014-09-05	232000.00000	2
325	7520000520	2015-03-11	240500.00000	2
345	3969300030	2014-07-23	165000.00000	4
...	...	...	...	...
14294	3528000040	2014-10-01	1690000.00000	3
14295	3528000040	2015-03-26	1800000.00000	3
15999	5536100020	2015-05-12	1190000.00000	3
18976	7856400300	2014-07-02	1410000.00000	2
18977	7856400300	2015-03-22	1510000.00000	2

353 rows × 21 columns

Here, we see that when we check for duplicates by ' price. This leads me to believe that the duplicate is s reflect the most recent change of value and accurate

Contents

Contents

In [26]:
```
df=df.drop_duplicates(subset=['id']
print(len(df))
```

```
21388
```

Our dataset went from 21,565 observations to 21,38

In [27]:
```
# Confirm that there are no more du

df.duplicated(subset=['id'],keep=Fa
```

Out[27]: 0

3 Exploratory Data Analys

- Explore the distribution of each variable and the
- Determine if variables are discrete or continuou
- Determine if variables are categoric or numeric
- Check if their skew in the distribution or normal
- Check if their are outlier values and if they appe

Contents

In [28]:
```
def distr_(df, col):
    """
    Produces a boxplot, scatterplot
    Produces summary statistics
    """
    fig, ax = plt.subplots(figsize=
    mean=df[col].mean()
    median=df[col].median()
    max_=df[col].max()
    min_=df[col].min()
    std_=df[col].std()
    sns.histplot(df[col],alpha=0.5,
    sns.kdeplot(df[col],color='gree
    ax[0].set_xlabel(col)
    ax[0].set_title(f'{col} Distrib
    ax[0].axvline(mean, label=f'Mea
    ax[0].axvline(median, label=f'm
    ax[0].legend()

    sns.boxplot(data=df, x=col, ax=

    sns.scatterplot(data=df, x=df[c

    fig.tight_layout();
    print(f'{col.capitalize()} Summ
    print(f'Median: {median}')
    print(f'Mean: {mean:.4}')
    print(f'Max: {max_}')
    print(f'Min: {min_}')
    print(f'Std: {std_:.4}')
    plt.show()
```

3.1 Handling Error in Basement

When initially running the EDA check, sqft_basement

In [29]:
```
# Check values

df['sqft_basement'].value_counts(0)
```

Out[29]:
```
0.0       12701
?           452
600.0       215
500.0       205
700.0       205
           ...
2130.0        1
861.0         1
3480.0        1
4130.0        1
516.0         1
Name: sqft_basement, Length: 304, dtype
```

452 values have a question mark. I am going to assu

Contents

data, changing it to 0 will not alter the original distrib
turn sqft_basement into a binary variable. 0 for no ba
difficult to interpret

In [30]:
```
# Replace the question mark with a

df['sqft_basement'].replace(to_repl
```

In [31]:
```
df['sqft_basement'].value_counts(1)
```

Out[31]:
```
0.0      0.61497
600.0    0.01005
700.0    0.00958
500.0    0.00958
800.0    0.00940
           ...
2130.0   0.00005
861.0    0.00005
3480.0   0.00005
4130.0   0.00005
516.0    0.00005
Name: sqft_basement, Length: 303, dtype
```

Question mark is gone. 61% of homes do not have a

In [32]:
```
# Use .map to make sqft_basement in

df['basementyes'] = (df['sqft_basem

```

In [33]:
```
# Confirm that original sqft_baseme

df['basementyes'].value_counts(1)
```

Out[33]:
```
0    0.61497
1    0.38503
Name: basementyes, dtype: float64
```

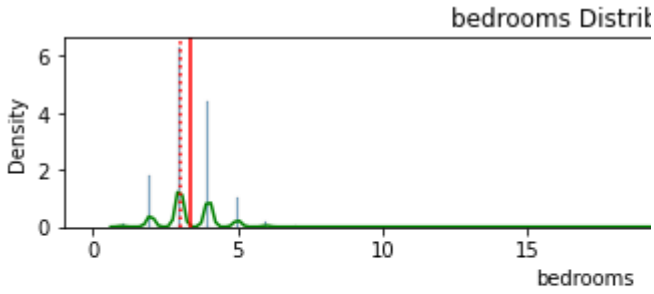
In [34]:
```
# Drop sqft_basement because we are
# or it does not)

df=df.drop('sqft_basement', axis=1)
```

3.2 Return to checking distribu

Contents

In [35]:
```
# Checking all variables except for

eda_check = ['price', 'bedrooms', '
       'sqft_lot', 'floors', 'water
       'sqft_above', 'yr_built', 'y
       'lat', 'long', 'sqft_living1
for col in eda_check:
    print(distr_(df, col))
    print('------------------------
```

```
None
--------------------------------------
Bedrooms Summary
Median: 3.0
Mean: 3.373
Max: 33
Min: 1
Std: 0.9252
```



3.2.1 Individual EDA Analysis

1. Price
   A. **Distribution**: Binomial, Right skewed
   B. **Outliers**: Outliers upper IQR threshold
   C. **Relationship with price**: NA
2. Bedrooms
   A. **Distribution**: Bimodal
   B. **Outliers**: Outliers upper IQR threshold. Extr
   C. **Relationship with price**: Linear until 5/6 be
   D. **Discrete or Continuous**: Discrete-Possibly
3. Bathrooms
   A. **Distribution**: Bimodal
   B. **Outliers**: Outliers upper IQR threshold.
   C. **Relationship with price**: Linear
   D. **Discrete or Continuous**: Discrete-Ordinal
4. Sqft_living
   A. **Distribution**: Bimodal, skewed
   B. **Outliers**: Outliers upper IQR threshold.
   C. **Relationship with price**: Linear
   D. **Discrete or Continuous**: Continuous

Contents

5. Sqft_lot
   A. **Distribution**: Binomial, Right skewed
   B. **Outliers**: Outliers upper IQR threshold.
   C. **Relationship with price**: Non-Linear
   D. **Discrete or Continuous**: Continuous
6. Floors
   A. **Distribution**: Bimodal, Right skewed
   B. **Outliers**: None
   C. **Relationship with price**: Non-Linear-Non C
   D. **Discrete or Continuous**: Discrete
7. Waterfront
   A. **Distribution**: Bernoulli
   B. **Outliers**: None
   C. **Relationship with price**: Unclear
   D. **Discrete or Continuous**: Discrete - Binary
8. View
   A. **Distribution**: Bernoulli
   B. **Outliers**: None
   C. **Relationship with price**: Unclear
   D. **Discrete or Continuous**: Discrete
9. Condition
   A. **Distribution**: Bernoulli
   B. **Outliers**: None
   C. **Relationship with price**: Unclear. Seems tc
   D. **Discrete or Continuous**: Discrete
10. Grade
    A. **Distribution**: Bernoulli
    B. **Outliers**: None
    C. **Relationship with price**: Linear
    D. **Discrete or Continuous**: Discrete
11. Sqft_above
    A. **Distribution**: Right skewed
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: Linear
    D. **Discrete or Continuous**: Continuous
12. Yr_Built
    A. **Distribution**: Left skewed
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: None
    D. **Discrete or Continuous**: Continuous
13. Yr_Renovated
    A. **Distribution**: Bernoulli
    B. **Outliers**: None
    C. **Relationship with price**: None
    D. **Discrete or Continuous**: Continuous
14. Sqft_living15
    A. **Distribution**: Right skewed
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: Linear

D. **Discrete or Continuous**: Continuous

15. Sqft_lot15
    A. **Distribution**: Right skewed
    B. **Outliers**: Outliers upper IQR threshold.
    C. **Relationship with price**: Linear
    D. **Discrete or Continuous**: Continuous
16. Basementyes
    A. **Distribution**: Binary
    B. **Outliers**: None
    C. **Relationship with price**: Unclear
    D. **Discrete or Continuous**: Discrete-Binary

3.2.2 Overall EDA Analysis

- Most of the continuous variables are right skewe
- Supported by the distribution and mean being g
- High outlier values on the upper IQR threshold
- Yr_renovated has a lot of 0 values so may be im
- Bedrooms has a mistaken entry (33 bedrooms)

3.2.2.1 Handle Bedroom error

In [36]:
```
# Find the observation where bedroo

df[df['bedrooms']>20]
```

Out[36]:

	id	date	price	bedrooms
15856	2402100895	2014-06-25	640000.00000	33

Appears to be a mistake because there is only 1.75 b

In [37]:
```
df.drop(index=15856, inplace=True)
```

In [38]:
```
# Confirm it has been removed

df[df['bedrooms']>20]
```

Out[38]:

id	date	price	bedrooms	bathrooms	sqft_living

3.2.3 Turn yr_renovated into binary val

- High amount of zero values, binary encoding wil
- Otherwise, the standard deviation will be very hi

Contents

Contents

In [39]:
```
df['yr_renovated'].value_counts(1)
```

Out[39]:
```
0.00000       0.96549
2014.00000    0.00341
2003.00000    0.00145
2013.00000    0.00145
2007.00000    0.00140
                 ...
1934.00000    0.00005
1971.00000    0.00005
1954.00000    0.00005
1950.00000    0.00005
1944.00000    0.00005
Name: yr_renovated, Length: 70, dtype:
```

Aprox. 97% of values suggest the home has not bee

In [40]:
```
df['renovated_yes'] = (df['yr_renov


```

In [41]:
```
# Confirm distribution has not chan

df['renovated_yes'].value_counts(1)
```

Out[41]:
```
0    0.96549
1    0.03451
Name: renovated_yes, dtype: float64
```

In [42]:
```
# Dropping yr_renovated because we

df.drop('yr_renovated', axis=1, inp
```

4 Feature Engineering

- Explore adding additional predictor values to the
- Feature engineering allows for us to broaden our
- Be cautious of multicollinearity because features

4.1 Total Rooms

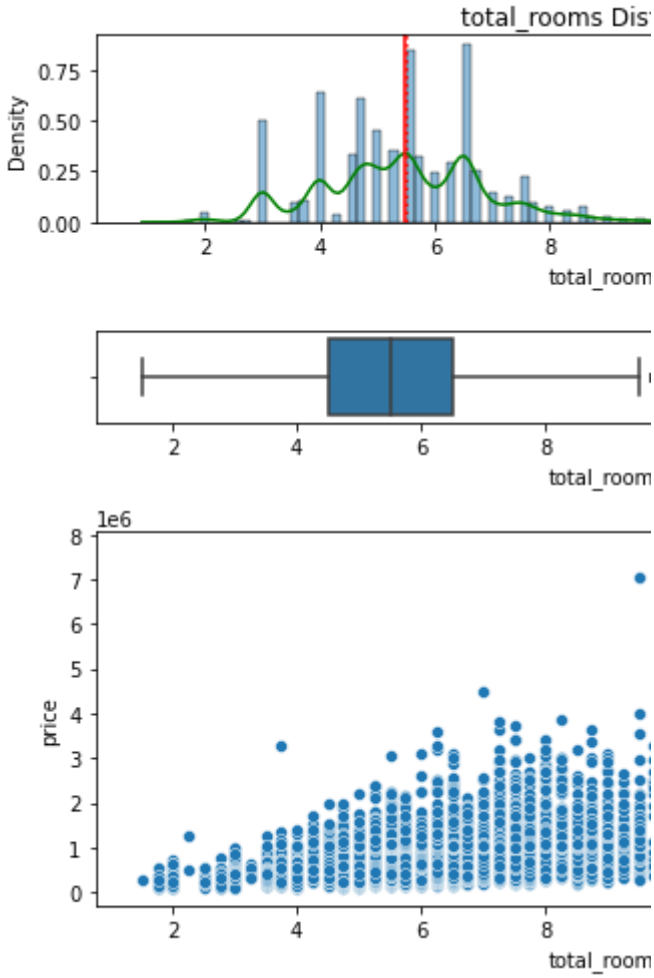
- Add bedrooms and bathrooms to create new co
- This will provide us a summary of the number of

In [43]:
```
# Combine bedrooms and bathrooms

df['total_rooms'] = df['bedrooms']+
```

Contents

In [44]:
```
1 distr_(df, 'total_rooms')
```

```
Total_rooms Summary
Median: 5.5
Mean: 5.49
Max: 16.5
Min: 1.5
Std: 1.463
```




```
1 - Distribution is not normal
2 - Looks like there are a significan
3 - Initially looks like there is a p
```

4.2 Backyard Size as a proporti

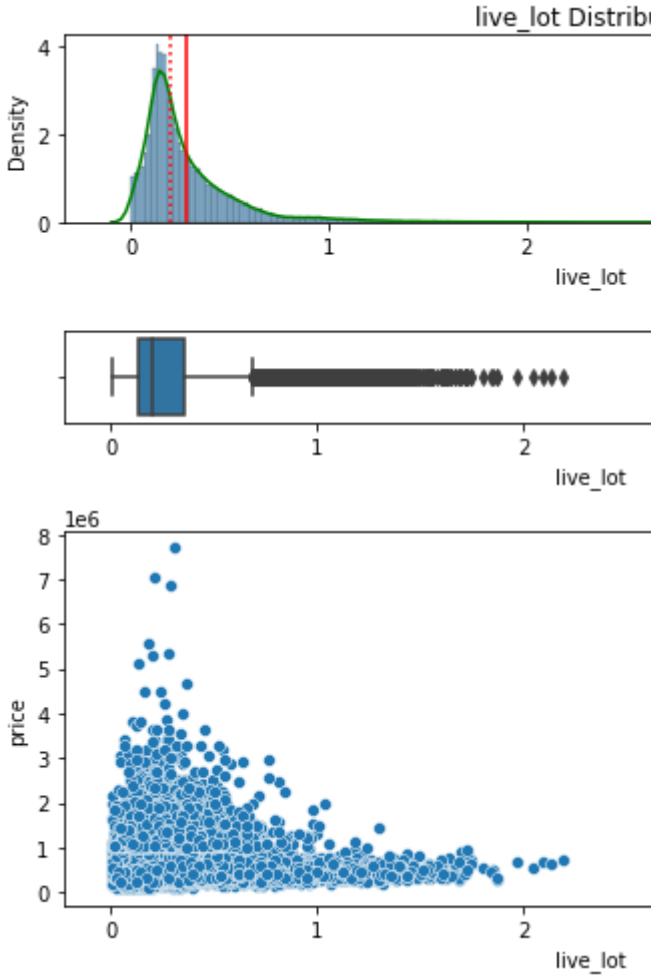
- Use sqft_above/sqft_lot as a proxy for backyard
- Essentially, we are capturing how big the home i
- A larger value means a relatively smaller backya

Contents

In [45]:
```
# Divide sqft_living/sqft_lot
# Using sqft_above as opposed to sq

df['live_lot'] = df['sqft_above']/d
```

In [46]:
```
distr_(df, 'live_lot')
```

```
Live_lot Summary
Median: 0.20274485339987525
Mean: 0.2808
Max: 4.653846153846154
Min: 0.0006095498431482305
Std: 0.2426
```



- Data is right skewed
- Significant number of outliers to the right of the
- Values may be greater than 1 because the living floors. In other words, it has a lot of sqft footage

4.3 Comparison of Square Foot

- Sqft_living15 represents the average living spac

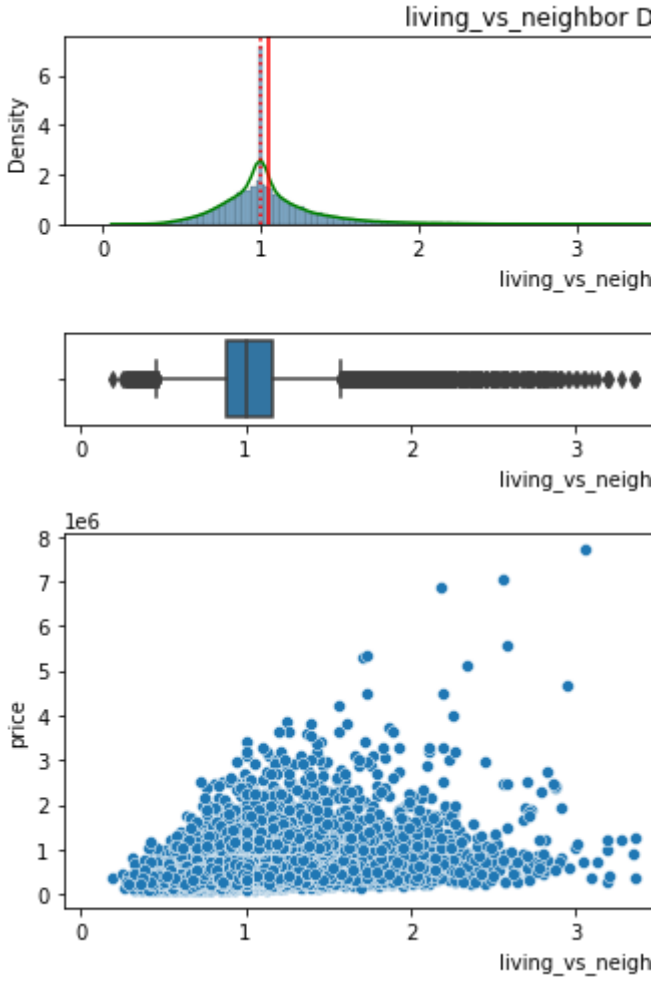
Contents

- Would like to compare how the living space of th

In [47]:
```
# SQF_living compared to neighbors
df['living_vs_neighbor'] = df['sqft
```

In [48]:
```
distr_(df, 'living_vs_neighbor')
```

```
Living_vs_neighbor Summary
Median: 1.0
Mean: 1.054
Max: 6.0
Min: 0.1872791519434629
Std: 0.3203
```




```
- Data looks pretty normally distri
- Outliers to the right of the uppe
- Unclear if there is a linear rela
- A lot of the much larger homes in
```

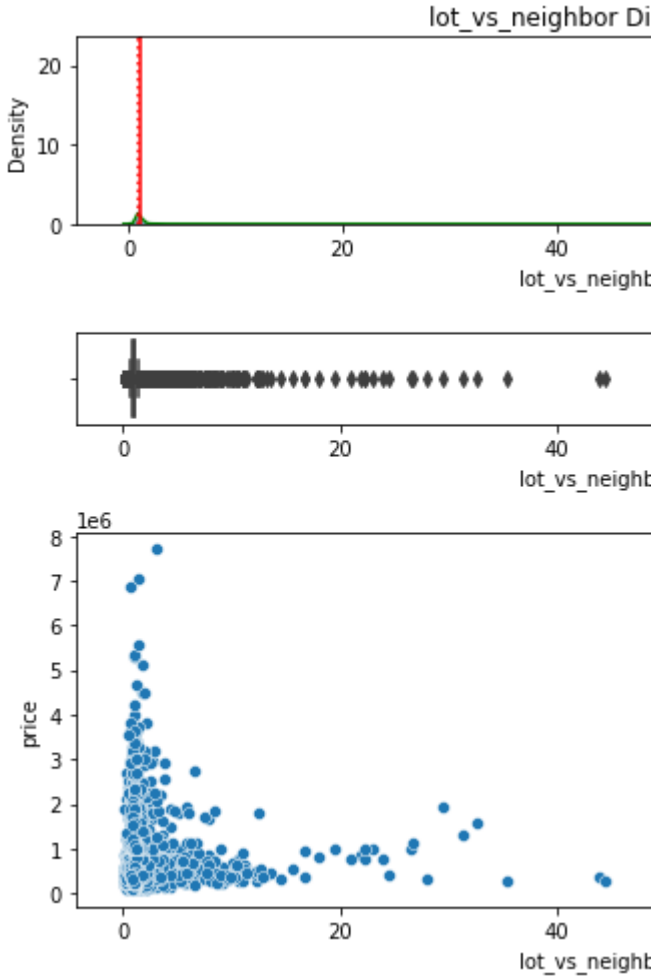
- Sqft_lot15 represents the average living space o
- Would like to compare how the lot size of the ob

Contents


```
In [49]:  1  df['lot_vs_neighbor'] = df['sqft_lo
```

```
In [50]:  1  distr_(df, 'lot_vs_neighbor')
```

```
Lot_vs_neighbor Summary
Median: 1.0
Mean: 1.134
Max: 87.52717948717948
Min: 0.054971997700810314
Std: 1.286
```



- Hard to tell distribution because there are clearly
- There are values greater than 80 which seem un
- Linearity seems unlinkely because as lot_vs_nei

5 Check Assumptions of L

- For the model to provide acccurate inferences, i

5.1 Check Assumption of Linea

Contents

- There must be a linear relationship between the
  - In our case, the predictor variable refers to t
- By linear relationship, we mean that as the x-val
- If we do not meet assumption of linearity, our m
- Must check each predictor that we are we going

In [51]:
```
def lin_check(df, cols, ncols=4, fi
    """
    Produces regplot for each featu
    """
    if ncols%4==0:
        fig, axes = plt.subplots(nr
        for ax, col in zip(axes.fla
            sns.regplot(data=df, x=
            ax.set_title(f'{col} vs
        fig.tight_layout()
    else:
        fig, axes = plt.subplots(nr
        for ax, col in zip(axes.fla
            sns.regplot(data=df, x=
            ax.set_title(f'{col} vs
        fig.tight_layout()
```

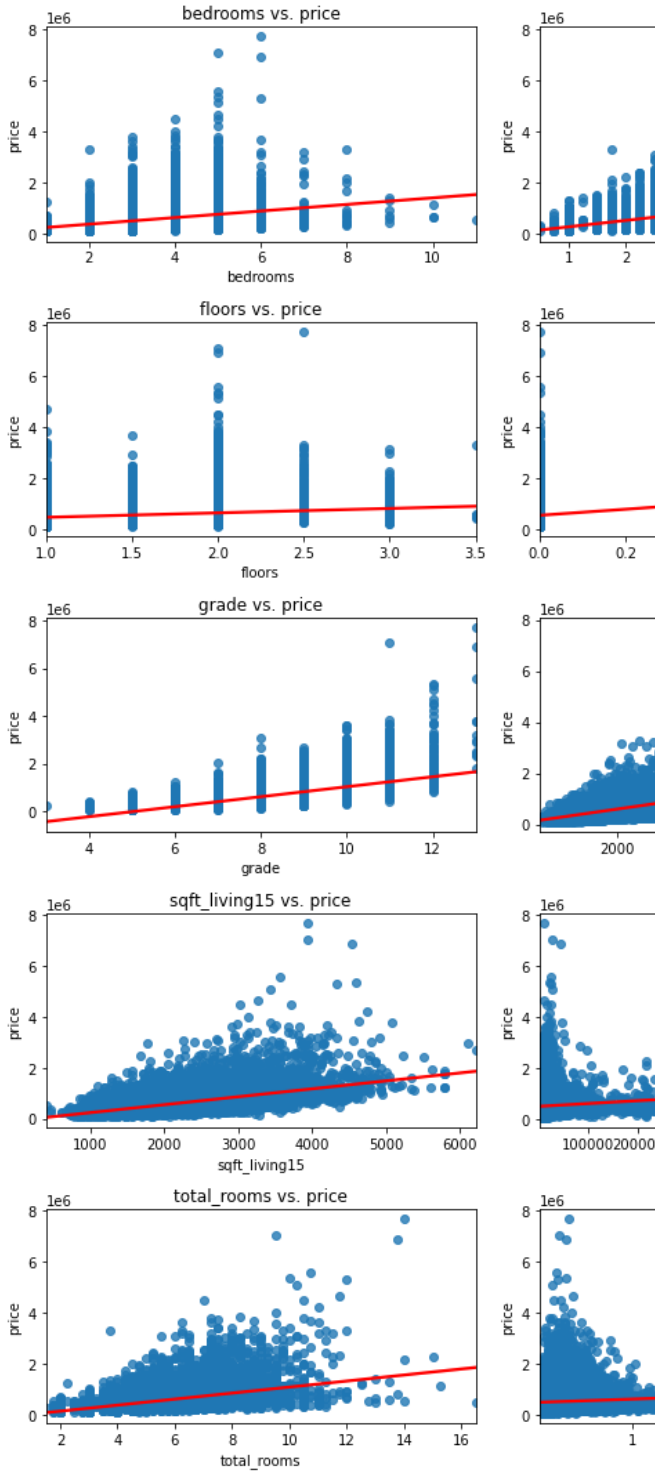
In [52]:
```
df.columns
```

Out[52]:
```
Index(['id', 'date', 'price', 'bedrooms
       'sqft_lot', 'floors', 'waterfron
       'sqft_above', 'yr_built', 'zipco
       'sqft_lot15', 'basementyes', 're
       'living_vs_neighbor', 'lot_vs_ne
      dtype='object')
```

In [53]:
```
# Choosing to remove latitude and l
# is a sufficient proxy for locatio
# will be easier for residents to u

cols_to_check = ['bedrooms', 'bathr
       'sqft_lot', 'floors', 'water
       'sqft_above', 'yr_built', 'z
       'sqft_lot15', 'basementyes',
       'living_vs_neighbor', 'lot_v
```

Contents

In [54]:
```
1 # Checking for linearity between pr
2
3 lin_check(df, cols_to_check)
```



Contents

The following cells do not have a linear relationship v

- Sqft_lot
- Floors (categorical)
- View (categorical)
- Condition (categorical)
- Yr_built
- Zip code (categorical)
- Basementyes (categorical)
- Renovatedyes (categorical)
- Sqft_lot15
- Live_lot

Sqft_lot, Yr_built, sqft_lot15, and live_lot are numeric
not have a linear relationship with price, will proceed
price, I will One Hot Encode them

In [55]:
```
df.columns
```

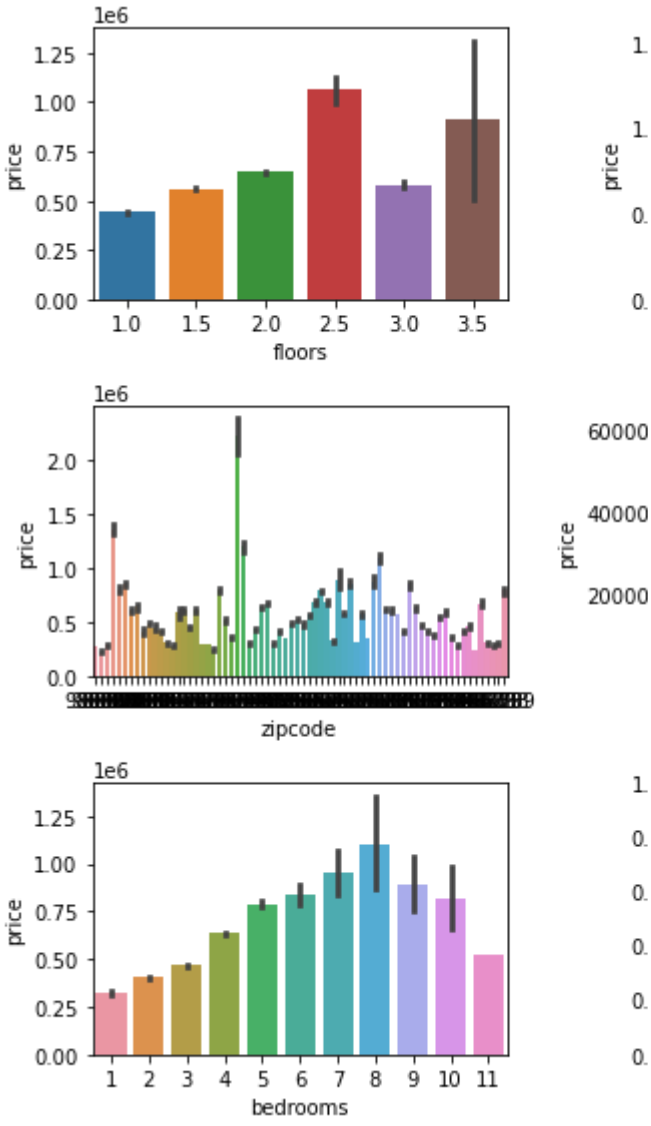
Out[55]:
```
Index(['id', 'date', 'price', 'bedrooms
       'sqft_lot', 'floors', 'waterfron
       'sqft_above', 'yr_built', 'zipco
       'sqft_lot15', 'basementyes', 're
       'living_vs_neighbor', 'lot_vs_ne
      dtype='object')
```

In [56]:
```
# These were continuous variables t

cols_to_drop = ['sqft_lot', 'sqft_l
df_lin = df.drop(cols_to_drop, axis
```

Contents

In [57]:

```
# Checking if categorical variables

cat_bars = ['floors', 'view', 'cond
fig, axes = plt.subplots(nrows=3, n
for ax, col in zip(axes.flatten(),
    sns.barplot(data=df, x=col, y='
    fig.tight_layout()

```



Contents

- Floors, and zipcode do not have linear relationsh
- Condition is close, but roughly has a linear relati
- Will turn floors, zip code, and bedrooms into OH

Conclusion:

- Will be dropping: Sqft_lot, Yr_built, sqft_lot15, a
- Will be One Hot Encoding: Floors, zipcode, and

5.2 Check Assumption of Multi

For a multiple linear regression model to be accurate
not only should the predictors have a linear relationsl
move very close together, then they are redundant fc
change in predictor values

RoadMap for checking assumptions of multicollinear

1. Run initial check of correlation with price
2. Observe heatmap triangle to see which predicto
3. Build table to show which variables have a corre
   - 0.75 is the norm for determing if predictor v

In [58]:
```
# Check correlation with price

def initial_corr_check(df, col='pri
    return df.corr()['price'].round
```

Contents

In [59]:
```
initial_corr_check(df_lin.drop(colu
```

Out[59]:
```
price                 1.00000
sqft_living           0.70000
grade                 0.67000
sqft_above            0.60000
sqft_living15         0.58000
bathrooms             0.53000
total_rooms           0.47000
view                  0.39000
bedrooms              0.32000
lat                   0.31000
living_vs_neighbor    0.30000
waterfront            0.27000
floors                0.25000
basementyes           0.18000
renovated_yes         0.12000
lot_vs_neighbor       0.04000
condition             0.03000
long                  0.02000
zipcode              -0.05000
Name: price, dtype: float64
```

Initial correlation check shows that sqft_living, grade

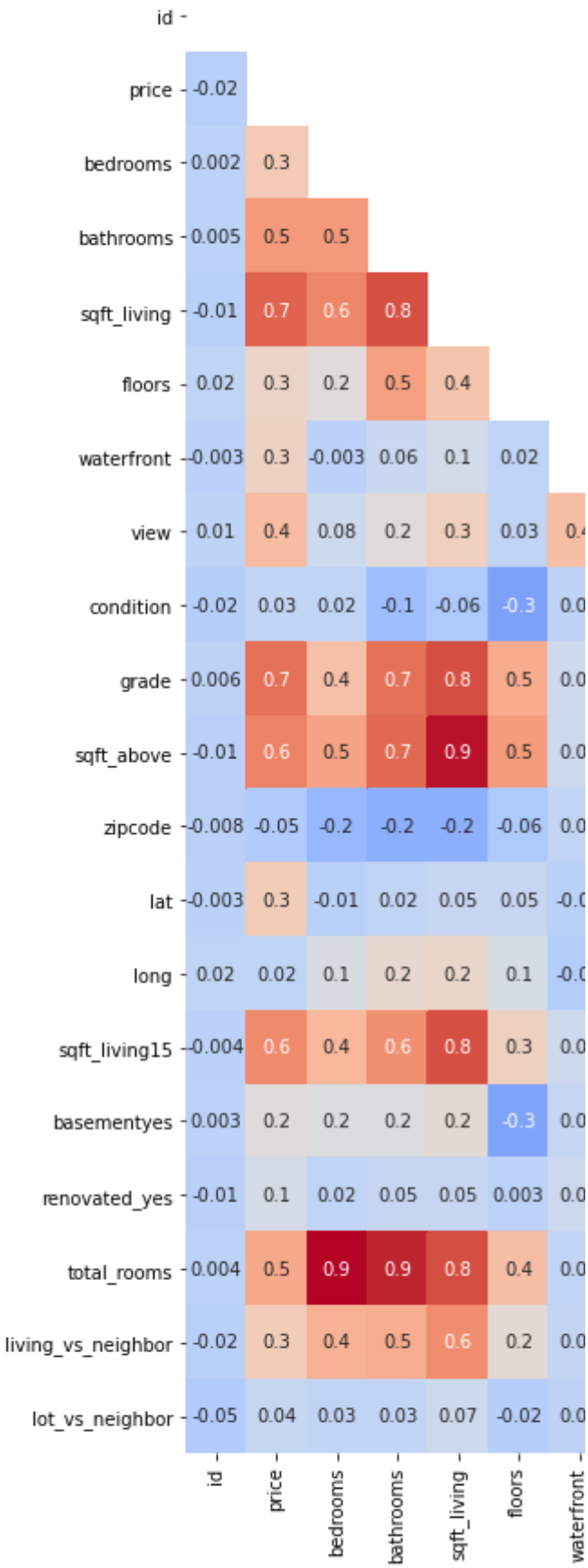
In [60]:
```
# Reference: https://heartbeat.frit

def corr_triangle(df):
    """
    Correlation heatcamp, including
    """
    corr2 = df.corr()
    fig, ax = plt.subplots(figsize=
    matrix = np.triu(corr2)
    return sns.heatmap(corr2,cmap="
```

```
In [61]: 1 corr_triangle(df_lin)
```

```
Out[61]: <AxesSubplot:>
```

Contents

Contents



There are a number of features that have strong mult

Contents

columns

- Total rooms correlates strongly with bedrooms a
- Sqft_living and sqft_living 15 are strongly correla
- Sqft_living and sqft_above and grade are strong
  - Because sqft_living and sqft_above will be t

In [62]:
```
# Reference:https://github.com/lear

def corr_finder(df):
    """
    Shows pairs of features that ha
    each other
    """
    df_corr = df.corr().abs().stack
    df_corr['pairs'] = list(zip(df_
    df_corr.set_index(['pairs'], in
    df_corr.drop(columns=['level_1'

    # # cc for correlation coeffici
    df_corr.columns = ['cc']
    df_corr.drop_duplicates(inplace

    return df_corr[(df_corr.cc>.75)
```

In [63]:
```
corr_finder(df_lin)
```

Out[63]:

pairs	cc
(total_rooms, bedrooms)	0.89510
(sqft_living, sqft_above)	0.87655
(bathrooms, total_rooms)	0.85186
(total_rooms, sqft_living)	0.76392
(sqft_living, grade)	0.76243
(sqft_living15, sqft_living)	0.75630
(grade, sqft_above)	0.75610
(bathrooms, sqft_living)	0.75581

Methodology to handle collinearity:

- For each pair, drop the feature that has the lowe
  - Maintaining the feature that has a stronger r
    - Bedrooms & Total_Rooms: Drop Total F between suggesting adding either bedr
    - Sqft_living & Sqft_above: Drop Sqft_ab represented with a binary variable. Nua
    - Total_Rooms and Bathrooms: Already e

- Total_Rooms and Sqft_living: Already e
- Sqft_living15 and Sqft_living: Already e
- Sqft_above and grade: Electing to keep
- Sqft_living and bathrooms: Already dro

Contents

In [64]:
```
cols_to_drop = ['total_rooms', 'sqf
df_linco = df_lin.drop(cols_to_drop
```

In [65]:
```
# Confirm no multicollinearity issu
# Elected to keep because they repr

corr_finder(df_linco)
```

Out[65]:

	cc
pairs	
(grade, sqft_living)	0.76243

Our model is now closer to meeting all the necessary

Next steps will be to handle outliers and then catego

6 Model 1: Baseline Model

- Now that assumptions of linearity and no multic
- Mainly observing R squared, Adjusted R squared
- Will check if certain variables are statistically ins outliers

What is R^2, Adjusted R^2, QQ Plot, and Homoskec

```
- R^2: Indicated how much variance i
 our 'goodness of fit' test. The hig
riance of the dependent model

- Adjusted R^2: Similar to R squared
s, it has a downward biance as the n

- Homoskedacicity: For our model to
t our residuals (actual value-predic
ance shuld not have a recognizable p

- QQ Plot: Helps us measure homosked
variance becomes non-uniform
```

Ref: https://statisticsbyjim.com/regression/interpret-

Contents

In [66]:

```
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as s

from scipy import stats
from sklearn.preprocessing import S
from statsmodels.formula.api import

from sklearn.datasets import make_r
from sklearn.linear_model import Li
import sklearn.metrics as metrics
```

In [67]:

```
def model_summary(df, X_targets, y,
    '''
    Produces OLS Linear Regression
    plot is displayed below the sum
    '''
    outcome = y
    x_cols = X_targets
    predictors = '+'.join(x_cols)
    formula = outcome + '~' + predi
    model = ols(formula=formula, da
    resid1 = model.resid
    display(model.summary())
    if qq==True:

        sm.graphics.qqplot(resid1,

    return model
```

In [68]:

```
from sklearn.datasets import make_r
from sklearn.linear_model import Li
import sklearn.metrics as metrics


def sked_show(df, X_cols, lr=None,
    '''
    Produces scatter plot showing m
    '''
    if lr is None:
        lr = LinearRegression()
        lr.fit(df[X_cols], df[val])

        y_hat = lr.predict(df[X_col
    else:
        y_hat = lr.predict(df)


    resid = (df[val] - y_hat)
    fig, ax= plt.subplots(figsize=(
    ax.scatter(x=y_hat,y=resid, alp
    ax.axhline(0, color='red')
    ax.set_xlabel('Price')
    ax.set_ylabel('Residual')
    return fig,ax
```

Contents

In [69]:

```
# Begin by using all columns as pre

x_targs = ['bedrooms', 'floors', 'w
       'view', 'condition', 'grade'
       'basementyes', 'renovated_ye
       'lot_vs_neighbor', 'sqft_liv
```

Contents

In [70]:
```
model_base = model_summary(df_linco
sked_show(df, x_targs, model_base)
```

OLS Regression Results

Dep. Variable:	price	**R-squared**	
Model:	OLS	**Adj. R-squared**	
Method:	Least Squares	**F-statistic**	
Date:	Wed, 05 May 2021	**Prob (F-statistic)**	
Time:	14:10:57	**Log-Likelihood**	
No. Observations:	21387	**AIC**	
Df Residuals:	21374	**BIC**	
Df Model:	12		
Covariance Type:	nonrobust		

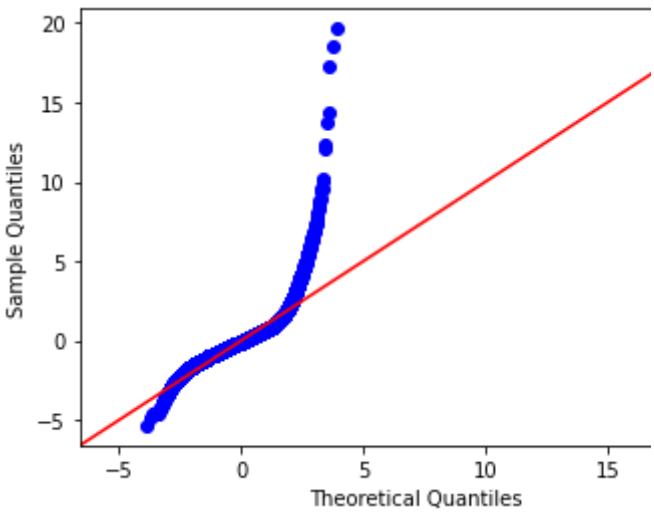
	coef	std err	t	P
Intercept	-5.254e+07	3.07e+06	-17.129	0.(
bedrooms	-3.239e+04	2204.113	-14.694	0.(
floors	-3302.0105	3555.724	-0.929	0.3
waterfront	6.124e+05	2.05e+04	29.829	0.(
view	4.962e+04	2361.401	21.012	0.(
condition	6.098e+04	2514.696	24.248	0.(
grade	9.624e+04	2284.261	42.133	0.(
zipcode	528.9805	31.267	16.918	0.(
basementyes	2.791e+04	3651.532	7.643	0.(
renovated_yes	1.524e+05	8613.485	17.697	0.(
living_vs_neighbor	-1.008e+05	6466.963	-15.591	0.(
lot_vs_neighbor	716.1896	1216.573	0.589	0.5
sqft_living	213.4715	3.621	58.950	0.(

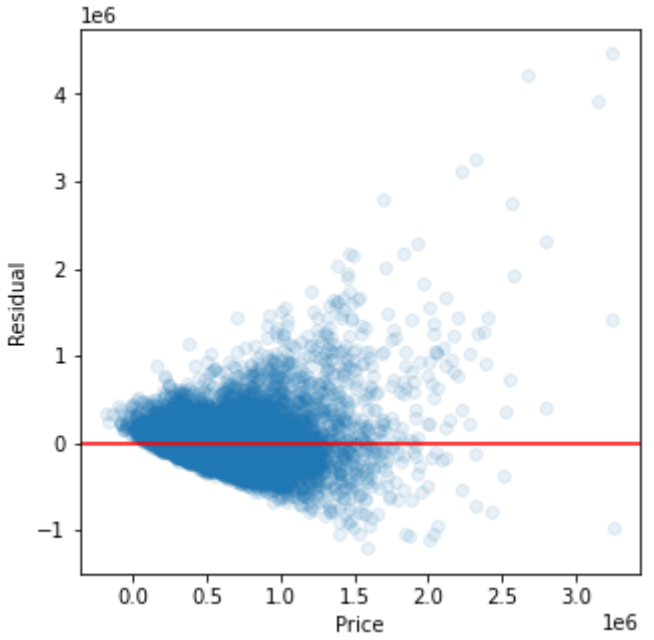
Omnibus:	15558.225	**Durbin-Watson:**	1
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	926735
Skew:	2.918	**Prob(JB):**	
Kurtosis:	34.716	**Cond. No.**	1.94(

Notes:
[1] Standard Errors assume that the covariance matri
[2] The condition number is large, 1.94e+08. This mig
strong multicollinearity or other numerical problems.

Contents

Out[70]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='R





Conclusions

- R^2: 0.618
- Adjusted R^2: 0.618
- QQ Plot: Deviates upwards at the 2nd/3rd quant
- Homoskedacicity: Becomes cone shaped aourn
- Non-Statistically Significant Preditors: Floors, lo
- Not dropping non-statistically significant values

7 Outlier Removal: IQR + Z

- Due to high outliers shown in the QQ Plot, next
- Will try using Z-Score and IQR method

- Will evaluate data loss to determine which meth

7.1 IQR Method

- The IQR method uses quantiles to determine if a
- The process:

    1. Determine IQR which is calcuated as Quant
    2. Upper Thresh: Quantile 3 * 1.5 *IQR
    3. Lower Thresh: Quantile 1 * 1.5 * -IQR
    4. Any values outside of the upper and lower t
- This method is more strict for evaluating outliers

7.1.1 IQR Method Accross All Columns

- In this approach, we are going to classify outlier
- For example, if an observation is has bedrooms
- This method is more strict for determining outlie
- CAUTION: we may have significant data loss

In [71]:
```
def find_outliers_IQR(data):
    """Detects outliers using the 1
    Returns a boolean Series where
    res = data.describe()
    q1 = res['25%']
    q3 = res['75%']
    thresh = 1.5*(q3-q1)
    idx_outliers =(data < (q1-thres
    return idx_outliers
```

In [72]:
```
df_linco.columns
```

Out[72]:
```
Index(['id', 'date', 'price', 'bedrooms
       'waterfront', 'view', 'condition
       'basementyes', 'renovated_yes',
       'lot_vs_neighbor'],
      dtype='object')
```

In [73]:
```
# Cols we are going to check with t
# Do not perform IQR check on binar

iqr_check = ['price', 'bedrooms',
             'sqft_living',
            'living_vs_neighbor', '

```

Contents

Contents

In [74]:
```
iqr_outliers = pd.DataFrame()
for col in iqr_check:
    iqr_outliers[col]=find_outliers
iqr_outliers['total'] = iqr_outlier
df_iqr = df_linco[~iqr_outliers['to
```

In [75]:
```
# Confirm that our data looks corre

df_iqr.head()
```

Out[75]:

	id	date	price	bedrooms	sqft
0	7129300520	2014-10-13	221900.00000	3	
1	6414100192	2014-12-09	538000.00000	3	
3	2487200875	2014-12-09	604000.00000	4	
4	1954400510	2015-02-18	510000.00000	3	
6	1321400060	2014-06-27	257500.00000	3	

In [76]:
```
df_iqr.describe()
```

Out[76]:

	id	price	bedroom
count	15475.00000	15475.00000	15475.0000
mean	4734745636.78559	467032.92775	3.3054
std	2878922190.79975	200357.29212	0.7680
min	1200019.00000	81000.00000	2.0000
25%	2297400055.00000	310000.00000	3.0000
50%	4046710050.00000	429000.00000	3.0000
75%	7504400730.00000	586750.00000	4.0000
max	9900000190.00000	1120000.00000	5.0000

With the new DataFrame, the range of values has be

- Price
  - Min: $81,000
  - Max: $1,120,000
- Bedrooms
  - Min: 2
  - Max: 5
- Floors
  - Min: 1
  - Max: 3.5

Contents

- Sqft_Living
  - Min: 560
  - Max: 4230

This constrains our dataset to only be able to provide
QQ Plot was trailing off aroun $1.25 million

In [77]:
```
print(f'Num observations before dro
print(f'Num observations after drop
print(f'Num observations removed: {
print(f'Num observations removed as
```

```
Num observations before dropping with I
Num observations after dropping with IQ
Num observations removed: 5,912
Num observations removed as percent of
```

With this type of outlier removal we have significant
performs with these constraints

7.1.1.1 Model 2: IQR All Outliers Removed

- Check to see if assumption of homoskedacicity
- Check if significant p-values has changed

In [78]:
```
# Ensure our DataFrame only include

df_iqr.describe()
```

Out[78]:

	id	price	bedroom
count	15475.00000	15475.00000	15475.0000
mean	4734745636.78559	467032.92775	3.3054
std	2878922190.79975	200357.29212	0.7680
min	1200019.00000	81000.00000	2.0000
25%	2297400055.00000	310000.00000	3.0000
50%	4046710050.00000	429000.00000	3.0000
75%	7504400730.00000	586750.00000	4.0000
max	9900000190.00000	1120000.00000	5.0000

In [79]:
```
x_targs = df_iqr.columns
x_targs = list(x_targs)
x_targs = [x for x in x_targs if x
```

Contents

In [80]:

```
1 model_iqra = model_summary(df_iqr,
2 sked_show(df_iqr, x_targs, model_iq
```

OLS Regression Results

Dep. Variable:	price	**R-squared**	
Model:	OLS	**Adj. R-squared**	
Method:	Least Squares	**F-statistic**	
Date:	Wed, 05 May 2021	**Prob (F-statistic)**	
Time:	14:10:57	**Log-Likelihood**	
No. Observations:	15475	**AIC**	
Df Residuals:	15460	**BIC**	
Df Model:	14		
Covariance Type:	nonrobust		

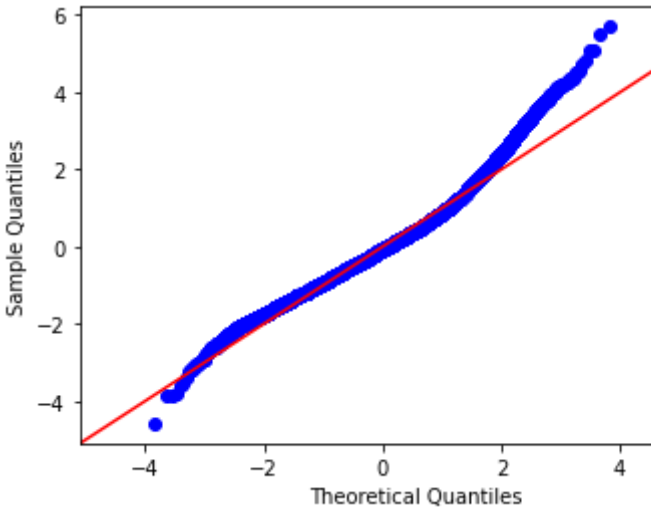
	coef	std err	t	P
Intercept	-2.777e+07	1.84e+06	-15.108	0.(
bedrooms	-8472.8199	1525.406	-5.554	0.(
sqft_living	134.7622	2.678	50.328	0.(
floors	1.37e+04	2184.062	6.273	0.(
waterfront	1.016e+05	3.01e+04	3.373	0.(
view	3.655e+04	1686.343	21.674	0.(
condition	4.756e+04	1535.022	30.983	0.(
grade	5.888e+04	1520.542	38.722	0.(
zipcode	-119.1025	21.897	-5.439	0.(
lat	5.857e+05	6849.178	85.512	0.(
long	-9.245e+04	8475.166	-10.909	0.(
basementyes	1.76e+04	2287.099	7.694	0.(
renovated_yes	9.912e+04	6055.472	16.368	0.(
living_vs_neighbor	-1.105e+05	5856.715	-18.869	0.(
lot_vs_neighbor	3044.9759	8182.017	0.372	0.1

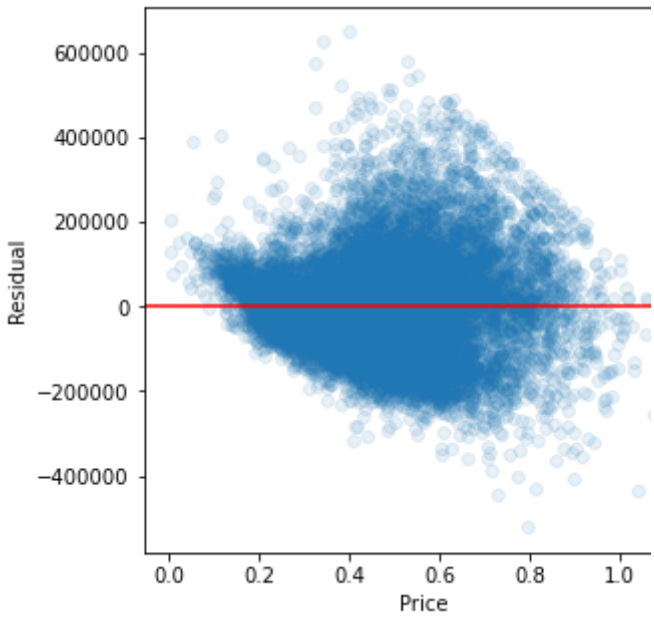
Omnibus:	1495.077	**Durbin-Watson:**	1.86
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2626.11
Skew:	0.679	**Prob(JB):**	0.0
Kurtosis:	4.494	**Cond. No.**	1.97e+0

Notes:

[1] Standard Errors assume that the covariance matri
[2] The condition number is large, 1.97e+08. This mig
strong multicollinearity or other numerical problems.

Out[80]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='R





Conclusions

- R^2: 0.676
- Adjusted R^2: 0.676
- QQ Plot: Meets assumption but treads up slightl
  assumption the scale would be -4 to 4

Contents

- Homoskedacicity: No cone shape, passes assur baseline model
- Non-Statistically Significant Predictors: lot_vs_ne
- Not going to use this model because there is toc

Contents

7.1.2 IQR Price Outliers Removed

- Rather than considering an observation an outlie
- This method reduces data loss because we are

In [81]:
```
# Finding the upper bound of the pr
# Since prices cannot be negative,

res=df_linco['price'].describe()
thresh = res['75%'] -res['25%']
u_bound=res['75%']+1.5*thresh
u_bound
```

Out[81]: 1125564.75

Maximum price for this model will be $1,125,564

In [82]:
```
# Subsetting the data to only inclu

df_iqrp = df_linco[df_linco['price'
```

In [83]:
```
# Ensure that observations have bee

print(len(df_iqrp))
print(df_iqrp['price'].max())
```

```
20235
1120000.0
```

Confirmed that we have less data than the original D

In [84]:
```
print(f'Num observations before dro
print(f'Num observations after drop
print(f'Num observations removed: {
print(f'Num observations removed as
```

```
Num observations before dropping with I
Num observations after dropping with IQ
Num observations removed: 1,152
Num observations removed as percent of
```

With this type of outlier removal we have much lowe performs with these constraints.

Contents

Compared to removing values based on a single feat
means that our model can provide inferences for a w

In [85]:
```
max_ip = df_iqrp['price'].max()
min_ip = df_iqrp['price'].min()

print(f'Min Price: ${min_ip:,}')
print(f'Max Price: ${max_ip:,}')
```

```
Min Price: $78,000.0
Max Price: $1,120,000.0
```

Our model provides inferential capabilities for homes

7.1.2.1 Model 3: IQR Price Outliers Removed

- Considering an observation an outlier if price is a
- This method reduces data loss compared to pre

Contents

In [86]:
```
model_iqrp = model_summary(df_iqrp,
sked_show(df_iqrp, x_targs, model_i
```

OLS Regression Results

Dep. Variable:	price	**R-squared**	
Model:	OLS	**Adj. R-squared**	
Method:	Least Squares	**F-statistic**	
Date:	Wed, 05 May 2021	**Prob (F-statistic)**	
Time:	14:10:57	**Log-Likelihood**	
No. Observations:	20235	**AIC**	
Df Residuals:	20220	**BIC**	
Df Model:	14		
Covariance Type:	nonrobust		

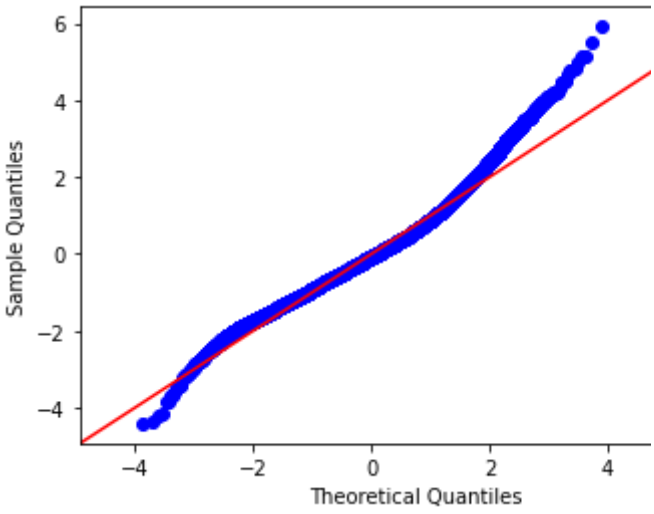
	coef	std err	t	P
Intercept	-2.723e+07	1.67e+06	-16.304	0.(
bedrooms	-7627.4038	1221.868	-6.242	0.(
sqft_living	121.3012	2.212	54.847	0.(
floors	1.412e+04	1936.055	7.293	0.(
waterfront	1.211e+05	1.76e+04	6.880	0.(
view	3.667e+04	1428.352	25.676	0.(
condition	4.31e+04	1374.848	31.352	0.(
grade	6.184e+04	1288.038	48.009	0.(
zipcode	-134.2122	20.068	-6.688	0.(
lat	5.895e+05	6321.978	93.251	0.(
long	-9.848e+04	7632.839	-12.902	0.(
basementyes	1.008e+04	2030.602	4.962	0.(
renovated_yes	8.853e+04	4918.509	18.000	0.(
living_vs_neighbor	-6.305e+04	3695.609	-17.062	0.(
lot_vs_neighbor	4833.7807	678.485	7.124	0.(

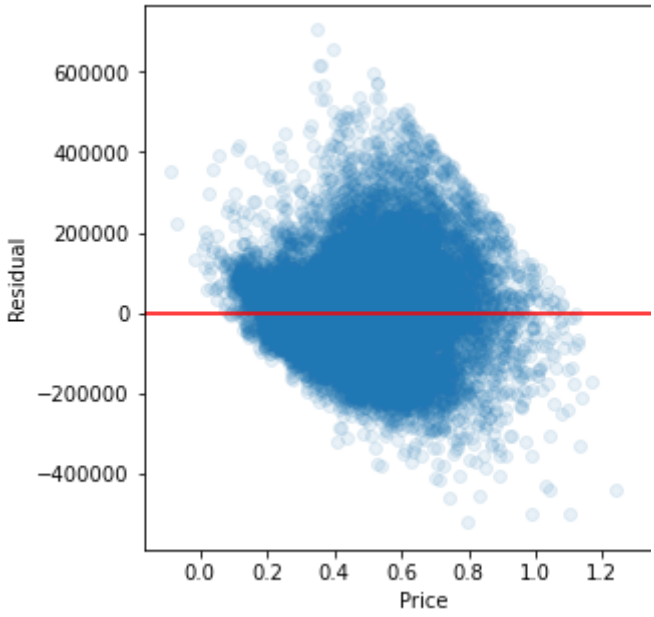
Omnibus:	1847.605	**Durbin-Watson:**	1.83
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3189.74
Skew:	0.655	**Prob(JB):**	0.0
Kurtosis:	4.438	**Cond. No.**	1.96e+0

Notes:

[1] Standard Errors assume that the covariance matri
[2] The condition number is large, 1.96e+08. This mig
strong multicollinearity or other numerical problems.

Out[86]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='R





Conclusions

- R^2: 0.667
- Adjusted R^2: 0.667
- QQ Plot: Meets assumption but treads up slightl
- Homoskedacicity: No cone shape, passes assur
- Non-Statistically Significant Predictors: None
- This model is highly preferable to removing all c
  wider range of inferences

Contents

Contents

7.2 Z-Score Method

- Z-score outlier removal is another method for cl
- To begin we standardize our values as z scores.
  1. This allows us to compare different features w
- We will not standardize Boolean values
- From there, determine if a value is an outlier bas
  - This rule of thumb comes from the empirica
  - Essentially, we are classifying a vale as an o
- This method is less strict at classifying outliers t

7.2.1 Z-Score Method Accross All Col

- In this approach, we are going to classify outlier
- For example, if an observation is has bedrooms
- This method is more strict for determining outlie
- CAUTION: we may have significant data loss

In [87]:
```
# Create scaler object

scaler = StandardScaler()
scaler
```

Out[87]:
```
StandardScaler()
```

In [88]:
```
# Create new DF to prepare for fit

df_z = df_linco.copy()
```

In [89]:
```
df_z.columns
```

Out[89]:
```
Index(['id', 'date', 'price', 'bedrooms
       'waterfront', 'view', 'condition
       'basementyes', 'renovated_yes',
       'lot_vs_neighbor'],
      dtype='object')
```

In [90]:
```
# Not scaling binary variables such
# Binary Variables are already enco
# 0 or 1
# Note: Scaling does not actually c

cols_to_scale = ['price', 'bedrooms
       'view', 'condition', 'grade'
       'living_vs_neighbor',
       'lot_vs_neighbor']
```

Contents

In [91]:
```
# Fit and transform original values

df_z[cols_to_scale] = scaler.fit_tr
df_z.describe()
```

Out[91]:

	id	price	bedrooms
count	21387.00000	21387.00000	21387.00000
mean	4581721940.59443	0.00000	0.00000
std	2876772841.46664	1.00002	1.00002
min	1000102.00000	-1.26066	-2.62718
25%	2124049194.50000	-0.58940	-0.41185
50%	3904930240.00000	-0.24815	-0.41185
75%	7309100170.00000	0.28260	0.69582
max	9900000190.00000	19.48493	8.44950

As we can see, the columns that we have scaled hav

In [92]:
```
# Create new DataFrame where we are
# less than 3 and greater than -3

outliers_z = pd.DataFrame()

for col in cols_to_scale:
    outliers_z[col] = df_z[col].abs

outliers_z['total'] = outliers_z.an
df_za = df_z[~outliers_z['total']].
```

Contents

In [93]:
```
print(len(df_za))
df_za.describe()
```

```
19735
```

Out[93]:

	id	price	bedrooms
count	19735.00000	19735.00000	19735.00000
mean	4608056409.49688	-0.14098	-0.05404
std	2876186313.51690	0.64731	0.94099
min	1000102.00000	-1.24841	-2.62718
25%	2140950145.00000	-0.61559	-0.41185
50%	3918400013.00000	-0.27945	-0.41185
75%	7334550735.00000	0.17373	0.69582
max	9900000190.00000	2.99080	2.91116

- We have created a new DataFrame that only inc
- Our mean and SD are no longer (0,1) because w
- In total, we now have 19,735 observations

In [94]:
```
# Formula that returns z-score back

def z_to_value(z, mu=df['price'].me
    """
    Converts z-score to original va
    """
    x = sd*z+mu
    return round(x,2)
```

In [95]:
```
print(f'Max price: ${z_to_value(2.9
print(f'Min price: ${z_to_value(-1.
```

```
Max price: $1,639,733.25
Min price: $82,490.54
```

The model will be able to infer prices of homes betw
score is less strict in terms of classifying outliers

Contents

In [96]:

```
1 print(f'Num observations before dro
2 print(f'Num observations after drop
3 print(f'Num observations removed: {
4 print(f'Num observations removed as
```

```
Num observations before dropping with I
Num observations after dropping with IQ
Num observations removed: 1,652
Num observations removed as percent of
```

With this type of outlier removal our data loss is apro
are met

This method will remove more data then when we or

7.2.2 Model 3: Z-Score All Outliers Rei

Contents

In [97]:
```
model_za = model_summary(df_za, x_t
sked_show(df_za, x_targs, model_za)
```

OLS Regression Results

Dep. Variable:	price	**R-squared**	
Model:	OLS	**Adj. R-squared**	
Method:	Least Squares	**F-statistic**	
Date:	Wed, 05 May 2021	**Prob (F-statistic)**	
Time:	14:10:58	**Log-Likelihood**	
No. Observations:	19735	**AIC**	
Df Residuals:	19720	**BIC**	
Df Model:	14		
Covariance Type:	nonrobust		

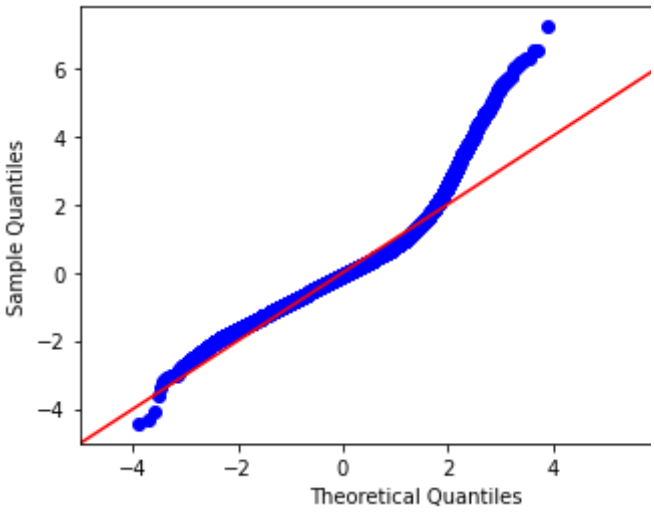
	coef	std err	t	P>|t
Intercept	-81.9507	5.385	-15.219	0.000
bedrooms	-0.0249	0.004	-6.775	0.000
sqft_living	0.3705	0.007	55.250	0.000
floors	0.0162	0.003	4.762	0.000
waterfront	0.6130	0.155	3.962	0.000
view	0.1000	0.005	19.771	0.000
condition	0.0888	0.003	30.751	0.000
grade	0.2319	0.005	46.912	0.000
zipcode	-0.0006	6.47e-05	-9.042	0.000
lat	1.6675	0.020	81.557	0.000
long	-0.4903	0.025	-19.848	0.000
basementyes	0.0290	0.007	4.421	0.000
renovated_yes	0.2978	0.016	18.476	0.000
living_vs_neighbor	-0.0819	0.004	-18.459	0.000
lot_vs_neighbor	0.0457	0.009	4.909	0.000

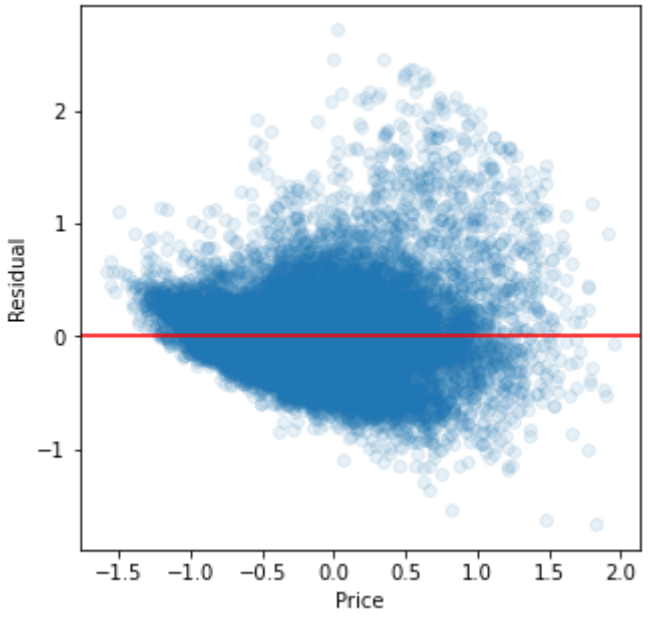
Omnibus:	5210.929	**Durbin-Watson:**	1.4
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21272.0
Skew:	1.256	**Prob(JB):**	0.
Kurtosis:	7.423	**Cond. No.**	1.96e+

Notes:

[1] Standard Errors assume that the covariance matri
[2] The condition number is large, 1.96e+08. This mig
strong multicollinearity or other numerical problems.

Out[97]: (<Figure size 360x360 with 1 Axes>,
<AxesSubplot:xlabel='Price', ylabel='R





Conclusions

- R^2: 0.660
- Adjusted R^2: 0.660
- QQ Plot: Does not do a good job at meeting ass
quantile
- Homoskedacicity: Cone shaped, especially as m
- Non-Statistically Significant Preditors: None
- Model does not meet all 4 assumptions, howeve

7.2.3 Z-Score Price Outliers Removed

Contents

- Rather than considering an observation an outlie
- This method reduces data loss because we are
- Will be using same scaled data from previous m

Contents

In [98]:
```
# Check to see if we have values gr
# Only need to worry about values g

df_z.describe()
```

Out[98]:

	id	price	bedrooms
count	21387.00000	21387.00000	21387.00000
mean	4581721940.59443	0.00000	0.00000
std	2876772841.46664	1.00002	1.00002
min	1000102.00000	-1.26066	-2.62718
25%	2124049194.50000	-0.58940	-0.41185
50%	3904930240.00000	-0.24815	-0.41185
75%	7309100170.00000	0.28260	0.69582
max	9900000190.00000	19.48493	8.44950

In [99]:
```
# Create new DataFrame only contain

df_zp = df_z[df_z['price'] < 3]
```

In [100]:
```
# Confirm that data has been remove
# Confirm that max price is less th

print(len(df_za))
df_zp['price'].max()
```

```
19735
```

Out[100]:
```
2.990795957515217
```

In [101]:
```
print(f'Max price: ${z_to_value(2.9
print(f'Min price: ${z_to_value(-1.
```

```
Max price: $1,639,733.25
Min price: $77,989.74
```

The model will be able to infer prices of homes betw

Contents

In [102]:

```
1 print(f'Num observations before dro
2 print(f'Num observations after drop
3 print(f'Num observations removed: {
4 print(f'Num observations removed as
```

```
Num observations before dropping with I
Num observations after dropping with IQ
Num observations removed: 404
Num observations removed as percent of
```

With this type of outlier removal our data loss is apro
are met

This method removes the least amount of data which
Second, Z-Score outlier removal is more strict than I

7.2.3.1 Model 4: Z-Score Price Outliers Removed

- Considering an observation an outlier if price is
- This method reduces data loss compared to pre

Contents

In [103]:
```
model_zp = model_summary(df_zp, x_t
sked_show(df_zp, x_targs, model_zp)
```

OLS Regression Results

Dep. Variable:	price	**R-squared**	
Model:	OLS	**Adj. R-squared**	
Method:	Least Squares	**F-statistic**	
Date:	Wed, 05 May 2021	**Prob (F-statistic)**	
Time:	14:10:58	**Log-Likelihood**	
No. Observations:	20983	**AIC**	
Df Residuals:	20968	**BIC**	
Df Model:	14		
Covariance Type:	nonrobust		

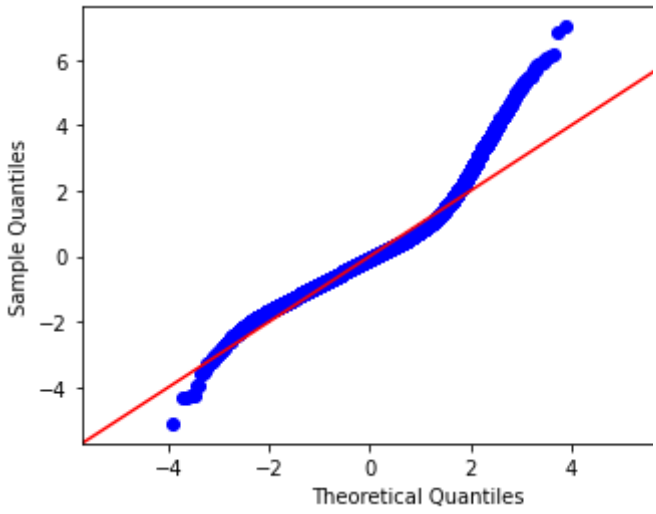
	coef	std err	t	P>|t
Intercept	-79.8958	5.424	-14.730	0.000
bedrooms	-0.0267	0.004	-7.542	0.000
sqft_living	0.3624	0.006	57.164	0.000
floors	0.0146	0.003	4.305	0.000
waterfront	0.5487	0.047	11.691	0.000
view	0.0991	0.003	30.002	0.000
condition	0.0860	0.003	29.774	0.000
grade	0.2411	0.005	50.040	0.000
zipcode	-0.0006	6.52e-05	-9.711	0.000
lat	1.7002	0.021	81.776	0.000
long	-0.4999	0.025	-20.061	0.000
basementyes	0.0215	0.007	3.271	0.001
renovated_yes	0.2953	0.016	19.038	0.000
living_vs_neighbor	-0.0637	0.004	-17.037	0.000
lot_vs_neighbor	0.0156	0.003	5.641	0.000

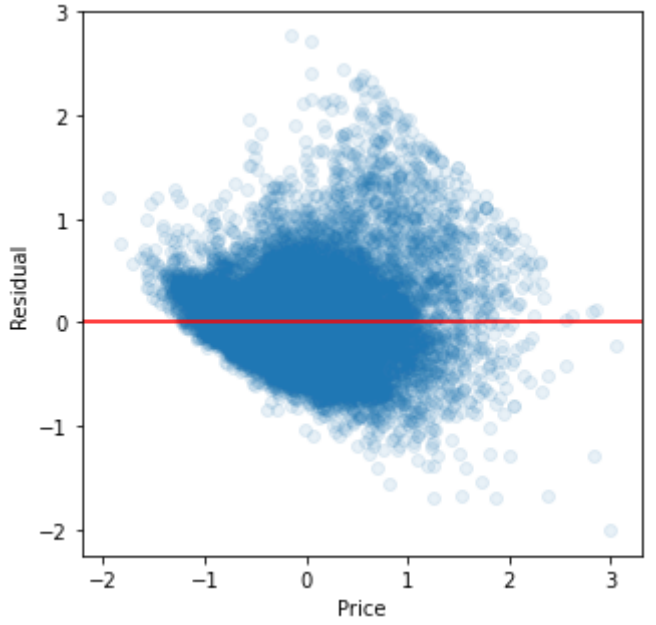
Omnibus:	4868.625	**Durbin-Watson:**	1.4
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	17767.8
Skew:	1.134	**Prob(JB):**	0.
Kurtosis:	6.896	**Cond. No.**	1.95e+

Notes:

[1] Standard Errors assume that the covariance matri
[2] The condition number is large, 1.95e+08. This mig
strong multicollinearity or other numerical problems.

Out[103]:
```
(<Figure size 360x360 with 1 Axes>,
 <AxesSubplot:xlabel='Price', ylabel='R
```





Conclusions

- R^2: 0.686
- Adjusted R^2: 0.686
- QQ Plot: Does not do a good job at meeting ass
- Homoskedacicity: Cone shaped, especially as m
- Non-Statistically Significant Preditors: Bathroom
- Model does not meet all 4 assumptions, howeve

Unfortunately, this model is not sufficient because it

Contents

Contents

7.3 Table to Compare 4 Outlier

- Want a simply way to evaluate all 4 models and

In [104]:
```
# Create DataFrame that compares al

d = {
    'Outlier Type': ['IQR-All', 'IQ
    'Data Loss %':[27.6,5.4,7.7,1.9
     'R^2':[0.676, 0.677, 0.660, 0.
    'Homoskedacicty':['Pass', 'Pass
    'QQ Plot':['Pass', 'Pass', 'Fai
    'Min Price':[81000, 78000.0, 82
    'Max Price':[1120000, 1120000,
    }
table_o = pd.DataFrame(d)
table_o.set_index('Outlier Type')
```

Out[104]:

Outlier Type	Data Loss %	R^2	Homoskedacicty
IQR-All	27.60000	0.67600	Pass
IQR-Price	5.40000	0.67700	Pass
Z-All	7.70000	0.66000	Fail
Z-Price	1.90000	0.68600	Fail

As we can see, each outlier removal type creates var
assumption of homoskedacicity. For that reason, I ar
price makes far more sense because the dat loss is r

In conclusion, will move forward modeling with **IQR-**

8 Handling Categorical Va

- Dummy variables (one hot encoded variables) m
multicollinearity. In other words, one of the dumr
- Our next step is to One Hot Encode the ordinal v
- These variables, when evaluated from an ordinal
- However, we will evaluate their P-Values to dete
  - Majority of OHE variables must be statistica
  - If not, can potentially feature engineer them

8.1 Check relationship of Non-l

Contents

In [105]:
```
def ordinal_check(df, col, val='pri
    """
    Produces stripplot and barplot
    the feature and price
    """
    fig, axes = plt.subplots(ncols=
    sns.stripplot(data=df, x=col, y
    sns.barplot(data=df, x=col, y=v

    fig.suptitle(f'Z-{col.upper()}
    plt.show()
    print('-----------------------
    print(df[col].value_counts(1))
```

In [106]:
```
# Based on our findings in our line
# Don't need to check binary variab

cat_bars = ['floors', 'view', 'cond
for col in cat_bars:
    ordinal_check(df_iqrp, col)
```

```
Name: zipcode, Length: 70, dtype: float
```

Based on the results we are going to One Hot Encod

- Floors: Increases up to 2.5 and then decreases
- Zipcode: Completely random
- Bedrooms: Increases up to 6 and then decrease
- Condition: Increases at 3 and then decreases at

This is because they do not appear ordinal. In other
Check if they are statistically significant

8.2 One Hot Encode Categorica

Contents

In [107]:
```
from sklearn.preprocessing import O
encoder = OneHotEncoder(sparse=Fals
encoder
```

Out[107]:
```
OneHotEncoder(drop='first', sparse=Fals
```

In [108]:
```
# Seperate the columns we are going

cat_cols=['floors', 'zipcode', 'bed

```

In [109]:
```
# Fit and transform categorical col
# Turn matrix into DataFrame

encoder.fit(df_linco[cat_cols])

ohe_vars = encoder.transform(df_iqr
encoder.get_feature_names(cat_cols)
cat_vars = pd.DataFrame(ohe_vars,co
```

In [110]:
```
# Confirm variables are OHE

cat_vars
```

Out[110]:

	floors_1.5	floors_2.0	floors_2.5	floors_3.0	f
0	0.00000	0.00000	0.00000	0.00000	
1	0.00000	1.00000	0.00000	0.00000	
2	0.00000	0.00000	0.00000	0.00000	
3	0.00000	0.00000	0.00000	0.00000	
4	0.00000	0.00000	0.00000	0.00000	
...	...	...	...	...	
20230	0.00000	1.00000	0.00000	0.00000	
20231	0.00000	1.00000	0.00000	0.00000	
20232	0.00000	1.00000	0.00000	0.00000	
20233	0.00000	1.00000	0.00000	0.00000	
20234	0.00000	1.00000	0.00000	0.00000	

20235 rows × 88 columns

Contents

In [111]:
```
# OLS Formula does not accept '.' s

name_dict = {}
for col in cat_vars.columns:
    name_dict[col]=col.replace('.',
name_dict
```

Out[111]:
```
{'floors_1.5': 'floors_1_5',
 'floors_2.0': 'floors_2_0',
 'floors_2.5': 'floors_2_5',
 'floors_3.0': 'floors_3_0',
 'floors_3.5': 'floors_3_5',
 'zipcode_98002': 'zipcode_98002',
 'zipcode_98003': 'zipcode_98003',
 'zipcode_98004': 'zipcode_98004',
 'zipcode_98005': 'zipcode_98005',
 'zipcode_98006': 'zipcode_98006',
 'zipcode_98007': 'zipcode_98007',
 'zipcode_98008': 'zipcode_98008',
 'zipcode_98010': 'zipcode_98010',
 'zipcode_98011': 'zipcode_98011',
 'zipcode_98014': 'zipcode_98014',
 'zipcode_98019': 'zipcode_98019',
 'zipcode_98022': 'zipcode_98022',
 'zipcode_98023': 'zipcode_98023',
 'zipcode_98024': 'zipcode_98024',
 'zipcode_98027': 'zipcode_98027',
 'zipcode_98028': 'zipcode_98028',
 'zipcode_98029': 'zipcode_98029',
 'zipcode_98030': 'zipcode_98030',
 'zipcode_98031': 'zipcode_98031',
 'zipcode_98032': 'zipcode_98032',
 'zipcode_98033': 'zipcode_98033',
 'zipcode_98034': 'zipcode_98034',
 'zipcode_98038': 'zipcode_98038',
 'zipcode_98039': 'zipcode_98039',
 'zipcode_98040': 'zipcode_98040',
 'zipcode_98042': 'zipcode_98042',
 'zipcode_98045': 'zipcode_98045',
 'zipcode_98052': 'zipcode_98052',
 'zipcode_98053': 'zipcode_98053',
 'zipcode_98055': 'zipcode_98055',
 'zipcode_98056': 'zipcode_98056',
 'zipcode_98058': 'zipcode_98058',
 'zipcode_98059': 'zipcode_98059',
 'zipcode_98065': 'zipcode_98065',
 'zipcode_98070': 'zipcode_98070',
 'zipcode_98072': 'zipcode_98072',
 'zipcode_98074': 'zipcode_98074',
 'zipcode_98075': 'zipcode_98075',
 'zipcode_98077': 'zipcode_98077',
 'zipcode_98092': 'zipcode_98092',
 'zipcode_98102': 'zipcode_98102',
 'zipcode_98103': 'zipcode_98103',
 'zipcode_98105': 'zipcode_98105',
 'zipcode_98106': 'zipcode_98106',
 'zipcode_98107': 'zipcode_98107',
```

```
'zipcode_98108': 'zipcode_98108',
'zipcode_98109': 'zipcode_98109',
'zipcode_98112': 'zipcode_98112',
'zipcode_98115': 'zipcode_98115',
'zipcode_98116': 'zipcode_98116',
'zipcode_98117': 'zipcode_98117',
'zipcode_98118': 'zipcode_98118',
'zipcode_98119': 'zipcode_98119',
'zipcode_98122': 'zipcode_98122',
'zipcode_98125': 'zipcode_98125',
'zipcode_98126': 'zipcode_98126',
'zipcode_98133': 'zipcode_98133',
'zipcode_98136': 'zipcode_98136',
'zipcode_98144': 'zipcode_98144',
'zipcode_98146': 'zipcode_98146',
'zipcode_98148': 'zipcode_98148',
'zipcode_98155': 'zipcode_98155',
'zipcode_98166': 'zipcode_98166',
'zipcode_98168': 'zipcode_98168',
'zipcode_98177': 'zipcode_98177',
'zipcode_98178': 'zipcode_98178',
'zipcode_98188': 'zipcode_98188',
'zipcode_98198': 'zipcode_98198',
'zipcode_98199': 'zipcode_98199',
'bedrooms_2': 'bedrooms_2',
'bedrooms_3': 'bedrooms_3',
'bedrooms_4': 'bedrooms_4',
'bedrooms_5': 'bedrooms_5',
'bedrooms_6': 'bedrooms_6',
'bedrooms_7': 'bedrooms_7',
'bedrooms_8': 'bedrooms_8',
'bedrooms_9': 'bedrooms_9',
'bedrooms_10': 'bedrooms_10',
'bedrooms_11': 'bedrooms_11',
'condition_2': 'condition_2',
'condition_3': 'condition_3',
'condition_4': 'condition_4',
'condition_5': 'condition_5'}
```

Contents

Contents

In [112]:
```
# Rename cat_vars DF with new names

cat_vars.rename(columns=name_dict,
cat_vars
```

Out[112]:

	floors_1_5	floors_2_0	floors_2_5	floors_3_0
0	0.00000	0.00000	0.00000	0.00000
1	0.00000	1.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	0.00000
...	...	...	...	...
20230	0.00000	1.00000	0.00000	0.00000
20231	0.00000	1.00000	0.00000	0.00000
20232	0.00000	1.00000	0.00000	0.00000
20233	0.00000	1.00000	0.00000	0.00000
20234	0.00000	1.00000	0.00000	0.00000

20235 rows × 88 columns

In [113]:
```
# Join OHE DataFrame back with orig
# Ensure each DataFrame has the sam
# Begin by resetting index

df_iqrp=df_iqrp.reset_index()
```

Contents

In [114]:
```
# Check number of rows and ensure t
# Should be 20235

df_iqrp
```

Out[114]:

	index	id	date	price	be
0	0	7129300520	2014-10-13	221900.00000	
1	1	6414100192	2014-12-09	538000.00000	
2	2	5631500400	2015-02-25	180000.00000	
3	3	2487200875	2014-12-09	604000.00000	
4	4	1954400510	2015-02-18	510000.00000	
...	...	...	...	...	
20230	21453	1245002281	2014-05-12	1050000.00000	
20231	21461	7010700308	2014-11-12	1010000.00000	
20232	21532	8835770330	2014-08-19	1060000.00000	
20233	21577	8672200110	2015-03-17	1090000.00000	
20234	21590	7936000429	2015-03-26	1010000.00000	

20235 rows × 18 columns

In [115]:
```
# Check to ensure that concat was d

df_ohe = pd.concat([df_iqrp, cat_va
```

Contents

In [116]:
```
# Ensure we do not have any missing

df_ohe.isna().sum()
```

Out[116]:
```
index           0
id              0
date            0
price           0
bedrooms        0
               ..
bedrooms_11     0
condition_2     0
condition_3     0
condition_4     0
condition_5     0
Length: 106, dtype: int64
```

We now have a new DataFrame called df_ohe that in
meet assumptions of no multicollinearity, and all inde
from DataFrame so they are not double counted

OHE variables are interpreted as such: With respect
coefficient. For example, if floors_1_5 had a coefficie

In [117]:
```
# Drop non-OHE variables so they ar

cols_to_drop = ['floors', 'zipcode'
df_ohe.drop(cols_to_drop, axis=1, i
```

In [118]:
```
# Ensure we do not have extra colum

df_ohe.describe()
```

Out[118]:

	index	id	pric
count	20235.00000	20235.00000	20235.0000
mean	10778.21596	4605299135.81389	477281.0373
std	6231.88071	2877559932.98793	206564.7901
min	0.00000	1000102.00000	78000.0000
25%	5381.50000	2136600137.50000	316000.0000
50%	10758.00000	3905081500.00000	439000.0000
75%	16173.00000	7338220140.00000	600000.0000
max	21596.00000	9900000190.00000	1120000.0000

Contents

In [119]:
```
1 # Confirm non-OHE variables have be
2
3 df_ohe.head()
```

Out[119]:

	index	id	date	price	sqft_livir
0	0	7129300520	2014-10-13	221900.00000	118
1	1	6414100192	2014-12-09	538000.00000	257
2	2	5631500400	2015-02-25	180000.00000	77
3	3	2487200875	2014-12-09	604000.00000	196
4	4	1954400510	2015-02-18	510000.00000	168

Contents

In [120]:
```
# Create list of columns excluding

x_targs = df_ohe.columns
x_targs = list(x_targs)
x_targs = [x for x in x_targs if x
x_targs
```

Out[120]:
```
['sqft_living',
 'waterfront',
 'view',
 'grade',
 'lat',
 'long',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_1_5',
 'floors_2_0',
 'floors_2_5',
 'floors_3_0',
 'floors_3_5',
 'zipcode_98002',
 'zipcode_98003',
 'zipcode_98004',
 'zipcode_98005',
 'zipcode_98006',
 'zipcode_98007',
 'zipcode_98008',
 'zipcode_98010',
 'zipcode_98011',
 'zipcode_98014',
 'zipcode_98019',
 'zipcode_98022',
 'zipcode_98023',
 'zipcode_98024',
 'zipcode_98027',
 'zipcode_98028',
 'zipcode_98029',
 'zipcode_98030',
 'zipcode_98031',
 'zipcode_98032',
 'zipcode_98033',
 'zipcode_98034',
 'zipcode_98038',
 'zipcode_98039',
 'zipcode_98040',
 'zipcode_98042',
 'zipcode_98045',
 'zipcode_98052',
 'zipcode_98053',
 'zipcode_98055',
 'zipcode_98056',
 'zipcode_98058',
 'zipcode_98059',
 'zipcode_98065',
 'zipcode_98070',
```

Contents


```
'zipcode_98072',
'zipcode_98074',
'zipcode_98075',
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199',
'bedrooms_2',
'bedrooms_3',
'bedrooms_4',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_9',
'bedrooms_10',
'bedrooms_11',
'condition_2',
'condition_3',
'condition_4',
'condition_5']
```

8.2.1 Model 5: OHE Iteration 1

Contents

In [121]:
```
model_ohe = model_summary(df_ohe, x
sked_show(df_ohe, x_targs, model_oh
```

sqft_living	135.6679	1.717	79.017	0.(
waterfront	1.644e+05	1.31e+04	12.581	0.(
view	3.255e+04	1057.566	30.775	0.(
grade	4.433e+04	987.127	44.906	0.(
lat	1.483e+05	3.49e+04	4.246	0.(
long	-5.204e+04	2.48e+04	-2.096	0.(
basementyes	-2.049e+04	1541.175	-13.296	0.(
renovated_yes	4.836e+04	3577.607	13.516	0.(
living_vs_neighbor	-4.941e+04	2794.428	-17.681	0.(
lot_vs_neighbor	5681.0665	492.684	11.531	0.(
floors_1_5	1.535e+04	2386.316	6.431	0.(
floors_2_0	-7407.2962	1778.178	-4.166	0.(

Conclusions

- R^2: 0.828
- Adjusted R^2: 0.827
- QQ Plot: Does a good job of meeting assumptio
- Homoskedacicity: Meets assumptions
- Non-Statistically Significant Preditors:
  - Maintaining bedroom because a majority of
  - Floors are majority statistically significant. W
  - Majority of zipcodes are statistically signific

Next step will be to turn floors into integer values

8.2.2 Model 5: Iteration 2 - Handling Fl

- In this iteration we are going to turn half floors in
  may not classify as an additional floor.

In [122]:
```
df_iqrp['floors'].value_counts(1)
```

Out[122]:
```
1.00000   0.50971
2.00000   0.36768
1.50000   0.08891
3.00000   0.02797
2.50000   0.00544
3.50000   0.00030
Name: floors, dtype: float64
```

- ~51% of homes are 1 story
- ~37% are 2 stories
- ~3% are 3 stories

Going to add the half story to their respective integer

In [123]:
```
# Function to adjust floors to thei

df_iqrp['floors'] = df_iqrp['floors
```

In [124]:
```
# Ensure transformation worked

df_iqrp['floors'].value_counts(1)
```

Out[124]:
```
1    0.59862
2    0.37312
3    0.02827
Name: floors, dtype: float64
```

As we can see, the 0.5 values were added to the resp

Now, we can try OHE with floors and the number of

Contents

Contents

In [125]:

```
from sklearn.preprocessing import O
encoder = OneHotEncoder(sparse=Fals
encoder

# Seperate the columns we are going

cat_cols=[]

# Fit and transform categorical col
# Turn matrix into DataFrame

def onehotencoder(df, cat_cols):
    encoder.fit(df[cat_cols])

    ohe_vars = encoder.transform(df
    encoder.get_feature_names(cat_c
    cat_vars = pd.DataFrame(ohe_var

    # OLS Formula does not accept '

    name_dict = {}
    for col in cat_vars.columns:
        name_dict[col]=col.replace(

    # Rename cat_vars DF with new n

    cat_vars.rename(columns=name_di

    # Join OHE DataFrame back with
    # Ensure each DataFrame has the
    # Begin by resetting index

    df=df.reset_index()

    df_ohe = pd.concat([df, cat_var

    # Drop original column names
    cols_to_drop = cat_cols
    df_ohe.drop(cols_to_drop, axis=
    return df_ohe
```

In [126]:

```
# Use function to re-OHE with new f


cat_cols = ['floors', 'zipcode', 'b
df_ohefloors = onehotencoder(df_iqr
```

Contents

In [127]:
```
# Should see floors_2 and floors_3

df_ohefloors.head()
```

Out[127]:

ated_yes	living_vs_neighbor	lot_vs_neighbor	floors_2
0	0.88060	1.00000	0.00000
1	1.52071	0.94803	1.00000
0	0.28309	1.24039	0.00000
0	1.44118	1.00000	0.00000
0	0.93333	1.07690	0.00000

In [128]:
```
# Confirm that we do not have any n

df_ohefloors.isna().sum().any()
```

Out[128]: False

Contents

In [129]:
```
# Create list of columns excluding
# Remove additional indices created
# Removing lat and long because zip

x_targs = df_ohefloors.columns
x_targs = list(x_targs)
x_targs = [x for x in x_targs if x
x_targs
```

Out[129]:
```
['sqft_living',
 'waterfront',
 'view',
 'grade',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_2',
 'floors_3',
 'zipcode_98002',
 'zipcode_98003',
 'zipcode_98004',
 'zipcode_98005',
 'zipcode_98006',
 'zipcode_98007',
 'zipcode_98008',
 'zipcode_98010',
 'zipcode_98011',
 'zipcode_98014',
 'zipcode_98019',
 'zipcode_98022',
 'zipcode_98023',
 'zipcode_98024',
 'zipcode_98027',
 'zipcode_98028',
 'zipcode_98029',
 'zipcode_98030',
 'zipcode_98031',
 'zipcode_98032',
 'zipcode_98033',
 'zipcode_98034',
 'zipcode_98038',
 'zipcode_98039',
 'zipcode_98040',
 'zipcode_98042',
 'zipcode_98045',
 'zipcode_98052',
 'zipcode_98053',
 'zipcode_98055',
 'zipcode_98056',
 'zipcode_98058',
 'zipcode_98059',
 'zipcode_98065',
 'zipcode_98070',
 'zipcode_98072',
 'zipcode_98074',
 'zipcode_98075',
```

Contents


```
'zipcode_98077',
'zipcode_98092',
'zipcode_98102',
'zipcode_98103',
'zipcode_98105',
'zipcode_98106',
'zipcode_98107',
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199',
'bedrooms_2',
'bedrooms_3',
'bedrooms_4',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_9',
'bedrooms_10',
'bedrooms_11',
'condition_2',
'condition_3',
'condition_4',
'condition_5']
```

Targets look correct with new OHE variables and no

Next step is to run Model 5 Iteration 2 and check if fl

Contents

In [130]:
```
model_ohefloors = model_summary(df_
sked_show(df_ohefloors, x_targs, mo
```

floors_3	-6.452e+04	4129.581	-15.623	0.(
zipcode_98002	1.167e+04	7660.497	1.523	0.
zipcode_98003	-4413.3594	6900.304	-0.640	0.5
zipcode_98004	5.169e+05	8431.130	61.312	0.(
zipcode_98005	3.309e+05	8446.732	39.169	0.(
zipcode_98006	2.714e+05	6362.838	42.655	0.(
zipcode_98007	2.625e+05	8732.050	30.061	0.(
zipcode_98008	2.44e+05	6999.033	34.861	0.(
zipcode_98010	9.473e+04	9781.390	9.685	0.(
zipcode_98011	1.474e+05	7690.627	19.160	0.(
zipcode_98014	1.249e+05	9099.142	13.730	0.(
zipcode_98019	1.06e+05	7742.104	13.693	0.(

As we can see, floors are statistically significant now

Conclusions

- R^2: 0.827
- Adjusted R^2: 0.827
- QQ Plot: Does a good job of meeting assumptio
- Homoskedacicity: Meets assumptions
- Non-Statistically Significant Preditors:
  - Maintaining bedroom because a majority of
  - Majority of zipcodes are statistically signific

- **Model 5 Iteration 2 will act as our final model**
  - It does not have any statistically insignificar
  - R^2 of 0.83 which means that the predictor
  - Meets all assumptions:
    - Homoskedacicity
    - Predictor variables have a linear relatio
    - No multicollinearity between predictor

9 Intepretation

- Share what the results of our multiple linear regr
- Support these points with visualizations from the

Contents

9.1 Standardize data for intepre

For our data to be on the same unit, we must use a S
standardizing data will change the range and distrib
across predictors

In [131]:
```
df_zf = df_ohefloors.copy()
```

In [132]:
```
# Predictor values, need to evaluat
# standard scaler

x_scale = df_zf.columns
x_scale
```

Out[132]:
```
Index(['level_0', 'index', 'id', 'date'
       'view', 'grade', 'lat', 'long',
       'living_vs_neighbor', 'lot_vs_ne
       'zipcode_98002', 'zipcode_98003'
       'zipcode_98006', 'zipcode_98007'
       'zipcode_98011', 'zipcode_98014'
       'zipcode_98023', 'zipcode_98024'
       'zipcode_98029', 'zipcode_98030'
       'zipcode_98033', 'zipcode_98034'
       'zipcode_98040', 'zipcode_98042'
       'zipcode_98053', 'zipcode_98055'
       'zipcode_98059', 'zipcode_98065'
       'zipcode_98074', 'zipcode_98075'
       'zipcode_98102', 'zipcode_98103'
       'zipcode_98107', 'zipcode_98108'
       'zipcode_98115', 'zipcode_98116'
       'zipcode_98119', 'zipcode_98122'
       'zipcode_98133', 'zipcode_98136'
       'zipcode_98148', 'zipcode_98155'
       'zipcode_98177', 'zipcode_98178'
       'zipcode_98199', 'bedrooms_2', '
       'bedrooms_6', 'bedrooms_7', 'bed
       'bedrooms_11', 'condition_2', 'c
       'condition_5'],
      dtype='object')
```

We don't scale our target variable (price) for interpret

In [133]:
```
# Not going to scale OHE variables
# Not going to scale binary variabl

x_scale = [x for x in x_targs if x

```

In [134]:
```
x_scale
```

Out[134]:
```
['sqft_living', 'view', 'grade', 'livin
```

We are ready to scale all of our numeric data

Contents

In [135]:
```
df_zf=df_zf.drop(['level_0', 'index
```

Dropping lat, and long because they are difficult to ir

In [136]:
```
# Fit and transform original values

df_zf[x_scale] = scaler.fit_transfo
df_zf.describe()
```

Out[136]:

	id	price	sqft_livin
count	20235.00000	20235.00000	20235.0000
mean	4605299135.81389	477281.03736	0.0000
std	2877559932.98793	206564.79018	1.0000
min	1000102.00000	78000.00000	-2.0760
25%	2136600137.50000	316000.00000	-0.7451
50%	3905081500.00000	439000.00000	-0.1507
75%	7338220140.00000	600000.00000	0.5986
max	9900000190.00000	1120000.00000	7.1108

As we can observe, the column that we have used th
been converted into Z-scores which allows us to cor
1 bedroom increase and a 1 sqft_living increase now

9.2 View which predictors mak

- When evaluating coefficients, we are looking at t
  1. As we increase this predictor variable, does
  2. How large is the coefficient. As we increase
     increase means a 1 standard deviation incre
  3. Coefficient values are interpreted as 'with re

Contents

In [137]:
```
1 x_targs
```

Out[137]:
```
['sqft_living',
 'waterfront',
 'view',
 'grade',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_2',
 'floors_3',
 'zipcode_98002',
 'zipcode_98003',
 'zipcode_98004',
 'zipcode_98005',
 'zipcode_98006',
 'zipcode_98007',
 'zipcode_98008',
 'zipcode_98010',
 'zipcode_98011',
 'zipcode_98014',
 'zipcode_98019',
 'zipcode_98022',
 'zipcode_98023',
 'zipcode_98024',
 'zipcode_98027',
 'zipcode_98028',
 'zipcode_98029',
 'zipcode_98030',
 'zipcode_98031',
 'zipcode_98032',
 'zipcode_98033',
 'zipcode_98034',
 'zipcode_98038',
 'zipcode_98039',
 'zipcode_98040',
 'zipcode_98042',
 'zipcode_98045',
 'zipcode_98052',
 'zipcode_98053',
 'zipcode_98055',
 'zipcode_98056',
 'zipcode_98058',
 'zipcode_98059',
 'zipcode_98065',
 'zipcode_98070',
 'zipcode_98072',
 'zipcode_98074',
 'zipcode_98075',
 'zipcode_98077',
 'zipcode_98092',
 'zipcode_98102',
 'zipcode_98103',
 'zipcode_98105',
 'zipcode_98106',
 'zipcode_98107',
```

Contents


```
'zipcode_98108',
'zipcode_98109',
'zipcode_98112',
'zipcode_98115',
'zipcode_98116',
'zipcode_98117',
'zipcode_98118',
'zipcode_98119',
'zipcode_98122',
'zipcode_98125',
'zipcode_98126',
'zipcode_98133',
'zipcode_98136',
'zipcode_98144',
'zipcode_98146',
'zipcode_98148',
'zipcode_98155',
'zipcode_98166',
'zipcode_98168',
'zipcode_98177',
'zipcode_98178',
'zipcode_98188',
'zipcode_98198',
'zipcode_98199',
'bedrooms_2',
'bedrooms_3',
'bedrooms_4',
'bedrooms_5',
'bedrooms_6',
'bedrooms_7',
'bedrooms_8',
'bedrooms_9',
'bedrooms_10',
'bedrooms_11',
'condition_2',
'condition_3',
'condition_4',
'condition_5']
```

Contents

In [138]:

```
# Run model to get parameter (coeff

model_final = model_summary(df_zf,
sked_show(df_zf, x_targs, model_fin
```

OLS Regression Results

Dep. Variable:	price	**R-squared**	
Model:	OLS	**Adj. R-squared**	
Method:	Least Squares	**F-statistic**	
Date:	Wed, 05 May 2021	**Prob (F-statistic)**	
Time:	14:11:05	**Log-Likelihood**	
No. Observations:	20235	**AIC**	
Df Residuals:	20141	**BIC**	
Df Model:	93		
Covariance Type:	nonrobust		

	coef	std err	t	P
Intercept	2.431e+05	1.82e+04	13.339	0.(

Confirmed that assumptions and values look the san

Scaling data does not make a difference

In [139]:

```
# Use statsmodels .params to extrac
# Include coefficient and absolute

df_coeff=pd.DataFrame({'coeff': mod
```

In [140]:
```
1 df_coeff
```

Out[140]:

	coeff	abs_coeff
Intercept	243052.98038	243052.98038
sqft_living	105564.12643	105564.12643
waterfront	164587.53385	164587.53385
view	20736.15964	20736.15964
grade	45704.32075	45704.32075
...	...	...
bedrooms_11	11786.85452	11786.85452
condition_2	48276.34358	48276.34358
condition_3	66106.77111	66106.77111
condition_4	87080.87703	87080.87703
condition_5	122763.55463	122763.55463

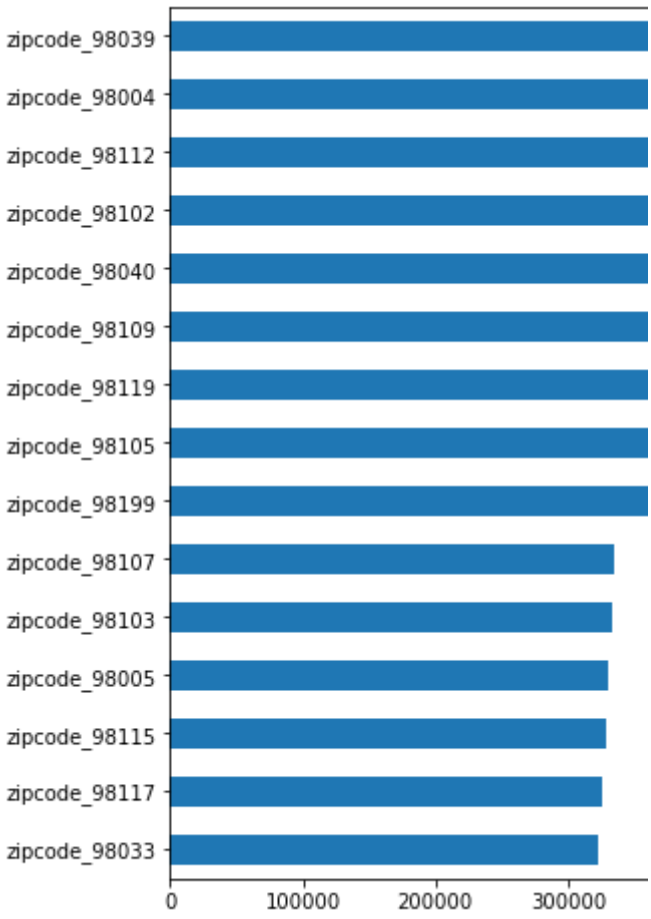
94 rows × 2 columns

Contents

Contents

In [141]:
```
# Create horizontal bar plot to vie
# absolute perspective

df_coeff.drop('Intercept', axis=0).
```

Out[141]: <AxesSubplot:>




```
As we can see, the 15 most impactfu
the zipcodes can explain ~$600,000

98039, 98004, and 98112 are the top
Check that later

Almost all zip codes are statistica
```

9.2.1 Digging Deepr Into Zip codes

- Evaluating how the top zipcodes compare to the

Contents

In [142]:
```
# Gather data on the top 10 zipcode

top_10_zips = df_iqrp.groupby('zipc
top_10_zips
```

Out[142]:

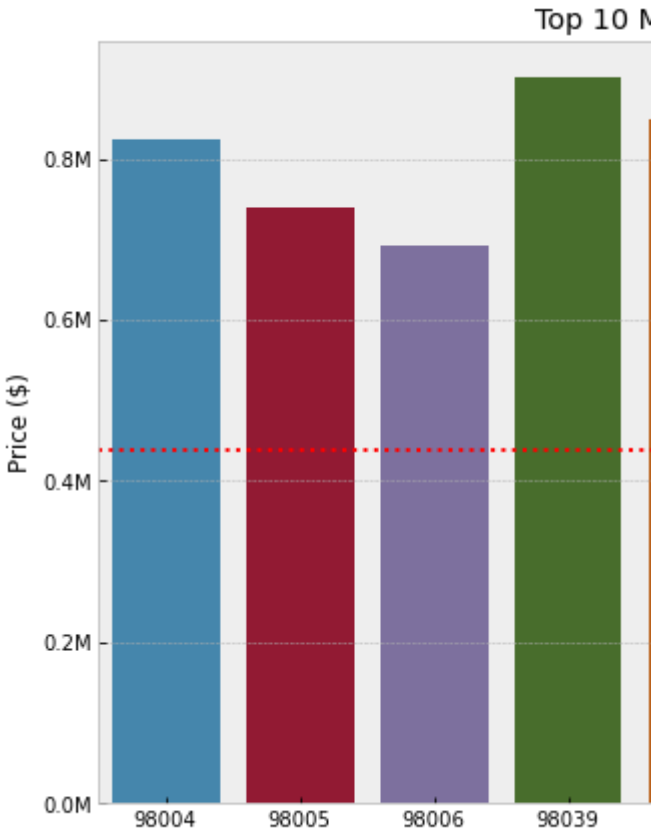
	zipcode	price
0	98039	901250.00000
1	98040	850000.00000
2	98004	825000.00000
3	98005	740000.00000
4	98075	725393.00000
5	98112	714250.00000
6	98109	700000.00000
7	98006	691100.00000
8	98119	667500.00000
9	98102	667475.00000

Using median in the groupby statement to handle ou

Contents

In [143]:

```
# Comparing top 10 zip codes by med

# Function to display values in mil
from matplotlib.ticker import FuncF

def millions(x, pos):
    return '%1.1fM' % (x * 1e-6)

# Kings County median home price
median = int(df_iqrp['price'].media

# Construct visualization
with plt.style.context('bmh'):
    fig, ax = plt.subplots(figsize=
    sns.barplot(data=top_10_zips, x
    ax.set_xlabel('Zip Code')
    ax.set_ylabel('Price ($)')
    ax.set_title('Top 10 Most Expen
    ax.axhline(median, color='r', l
    ax.legend()

    formatter = FuncFormatter(milli
    ax.yaxis.set_major_formatter(fo
```



The median home price in King County is $439,000.
The most expensive zip codes tend to be situated ar

Contents

9.2 Evaluating the predictors n

- Evaluate how the predictor values outside of zip

In [144]:
```
# Create list of features excluding

indices = df_coeff.index
non_zips = []
for ind in indices:
    if ind.startswith('zip'):
        pass
    else:
        non_zips.append(ind)
```

In [145]:
```
# These are the predictors that are

non_zips
```

Out[145]:
```
['Intercept',
 'sqft_living',
 'waterfront',
 'view',
 'grade',
 'basementyes',
 'renovated_yes',
 'living_vs_neighbor',
 'lot_vs_neighbor',
 'floors_2',
 'floors_3',
 'bedrooms_2',
 'bedrooms_3',
 'bedrooms_4',
 'bedrooms_5',
 'bedrooms_6',
 'bedrooms_7',
 'bedrooms_8',
 'bedrooms_9',
 'bedrooms_10',
 'bedrooms_11',
 'condition_2',
 'condition_3',
 'condition_4',
 'condition_5']
```

In [146]:
```
# Select the non-zip rows

df_coeff2 = df_coeff.loc[non_zips]
```

In [147]:
```
# Remove intercept because it is no

df_coeff2.drop('Intercept', axis=0,
```

Contents

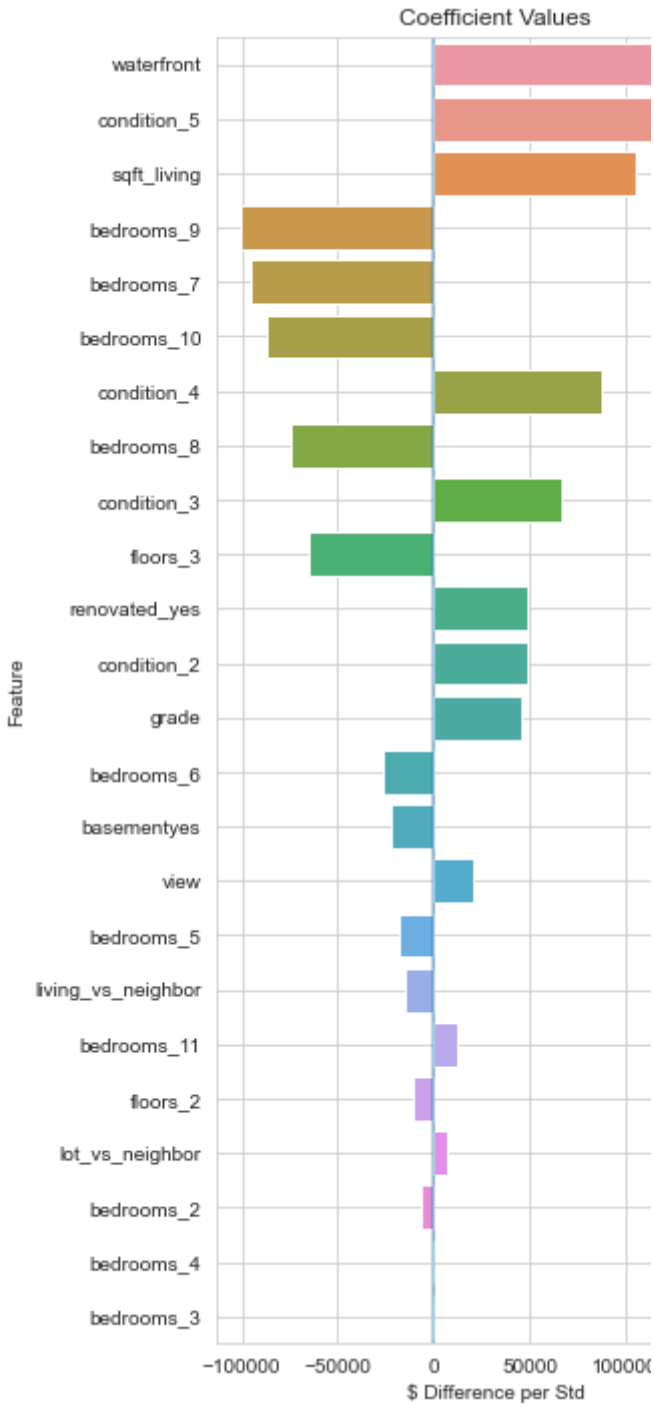
In [148]:
```
# Reset index so that we can sort b

df_coeff2=df_coeff2.reset_index()
```

In [149]:
```
# Sort in descending order so the v

df_coeff2=df_coeff2.sort_values(by=
```

Contents

In [150]:

```
sns.set_style('whitegrid')
fig, ax = plt.subplots(figsize=(5,1

sns.barplot(data=df_coeff2, y='inde
ax.set_title('Coefficient Values')
ax.set_xlabel('$ Difference per Std
ax.set_ylabel('Feature')

ax.axvline(0, alpha=0.5)
ax.grid(True)
```



Contents

Based on the graph above, waterfront (binary variabl
~$150,000 more. Next comes condition, all conditior
Condition represents the overall state of the house. I
have structural integrity, be generally well kept, and l
price. A 1 Standard Deviation increase in sqft_living r
standard deviation above the mean sqft_living will cc

Bedrooms 9,7, and 10 all have negative impacts on p
needed to OHE them.

All floors with respect to 1 floor have a negative relat

Grade has a positive impact on price, which is how t
good condition.

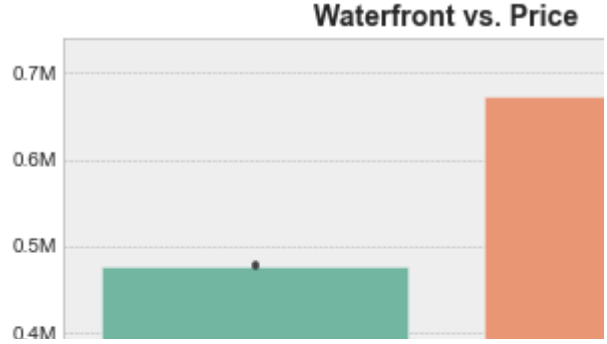
Homes with basements have a lower price on averag

View has a positive relationship with price

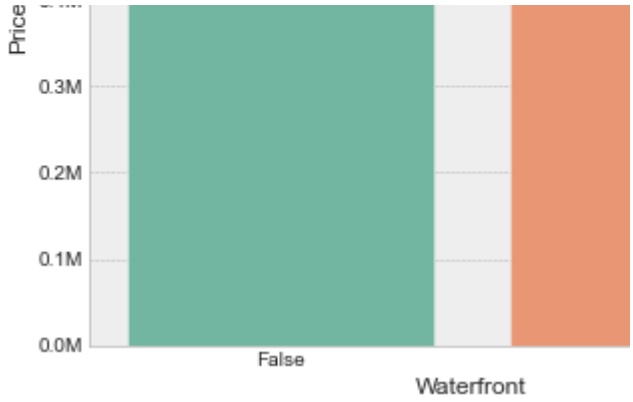
Contents

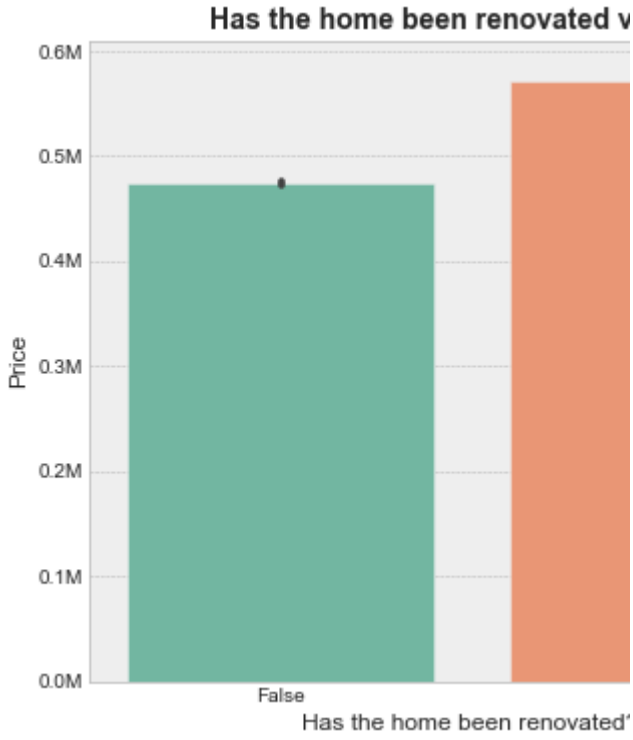
In [158]:

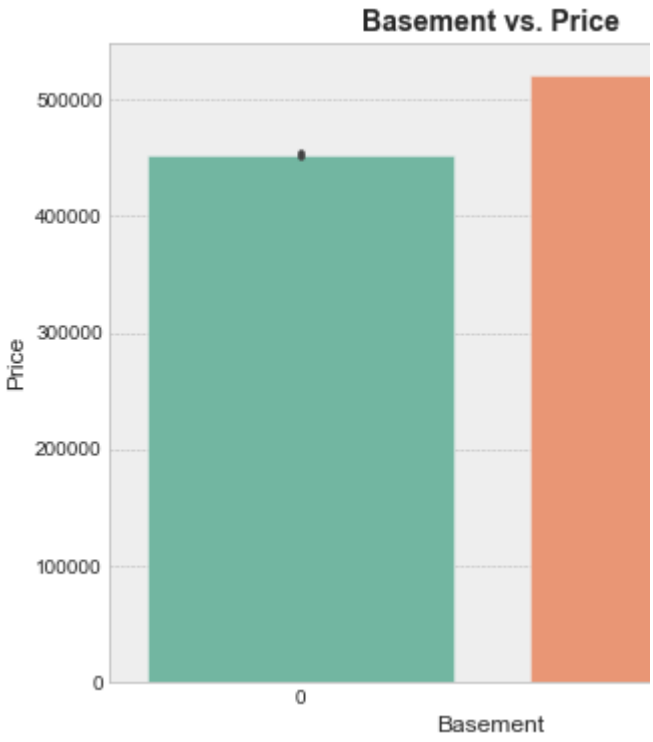
```
with plt.style.context('bmh'):

    fig, axs=plt.subplots(nrows=3,

    sns.barplot(data=df_ohefloors,
    sns.barplot(data=df_ohefloors,
    sns.barplot(data=df_ohefloors,
    sns.barplot(data=df_ohefloors,
    sns.barplot(data=df_ohefloors,
    sns.barplot(data=df_iqrp, x='co


    tf_labs = ['False', 'True']

    axs[0,0].set_title('Waterfront
    axs[0,0].set_xlabel('Waterfront
    axs[0,0].set_ylabel('Price')
    axs[0,0].set_xticklabels(tf_lab

    axs[0,1].set_title('View vs. Pr
    axs[0,1].set_xlabel('View')
    axs[0,1].set_ylabel('Price')

    axs[1,0].set_title('Has the hom
    axs[1,0].set_xlabel('Has the ho
    axs[1,0].set_ylabel('Price')
    axs[1,0].set_xticklabels(tf_lab

    axs[1,1].set_title('Grade vs. P
    axs[1,1].set_xlabel('Grade')
    axs[1,1].set_ylabel('Price')

    axs[2,0].set_title('Basement vs
    axs[2,0].set_xlabel('Basement')
    axs[2,0].set_ylabel('Price')

    axs[2,1].set_title('Condition v
    axs[2,1].set_xlabel('Condition'
    axs[2,1].set_ylabel('Price')


    formatter = FuncFormatter(milli
    axs[0,0].yaxis.set_major_format
    axs[0,1].yaxis.set_major_format
    axs[1,0].yaxis.set_major_format
    axs[1,1].yaxis.set_major_format
```



Contents







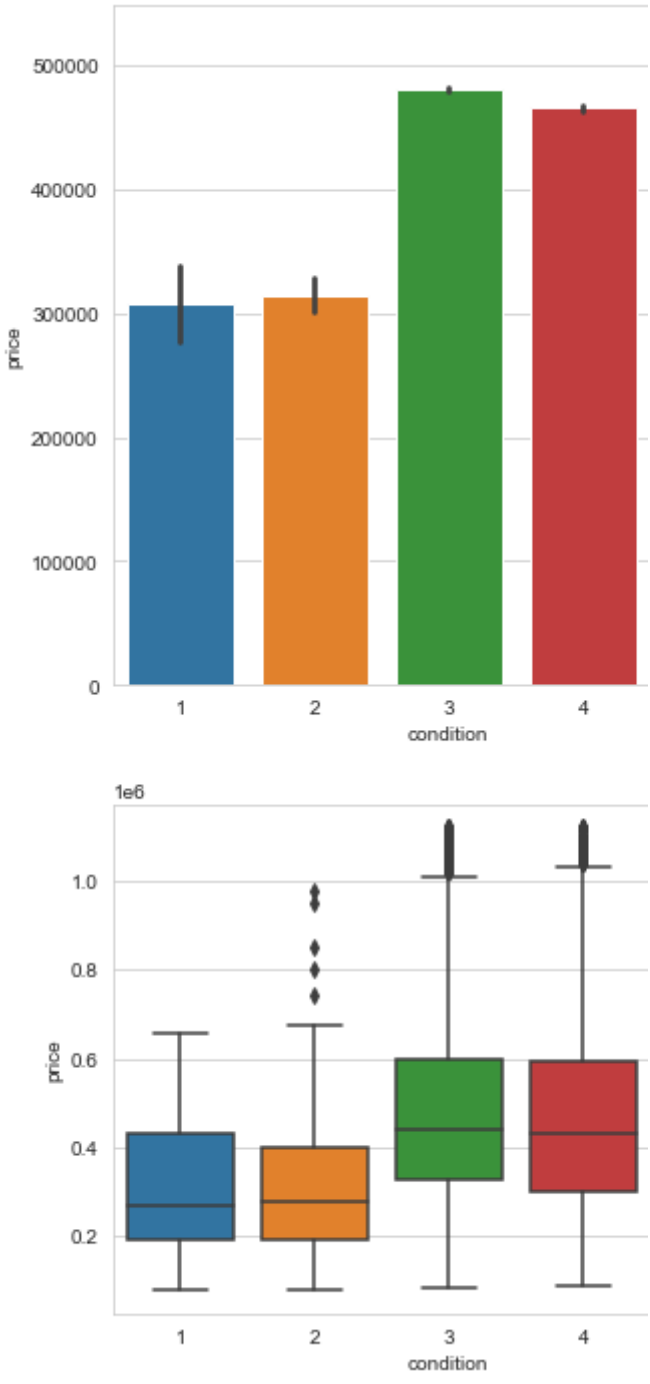
Contents

- Waterfront homes are more expensive than non-
- View increases in a semi linear fashion
- Homes that have been renovated are more expe
- Condition has a linear relationship with price
- Homes with basements are marginally more exp
- Condition does not have a linear relationship wit

10 Models for Presentatio

In [152]:
```
1 fig, axs=plt.subplots(nrows=2, ncol
2
3 sns.barplot(data=df_iqrp, x='condit
4 sns.boxplot(data=df_iqrp, x='condit
```

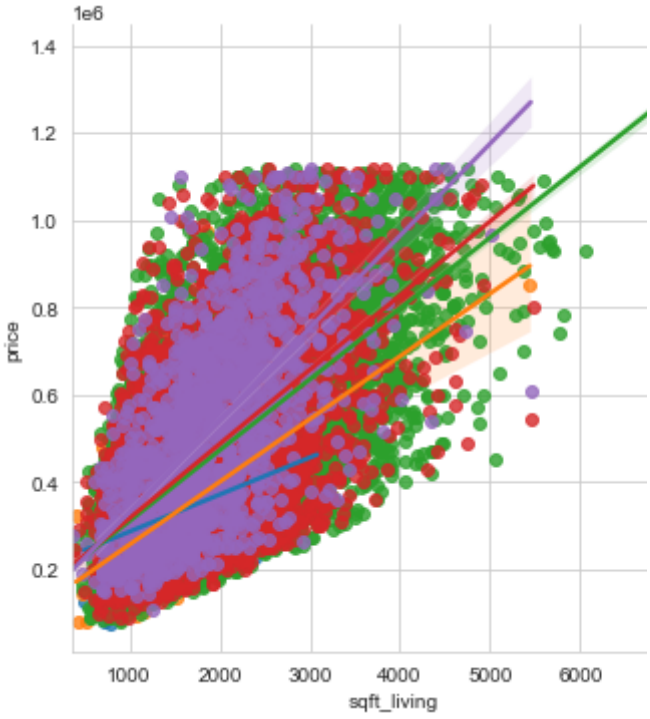
Out[152]: <AxesSubplot:xlabel='condition', ylabel



Contents

Contents

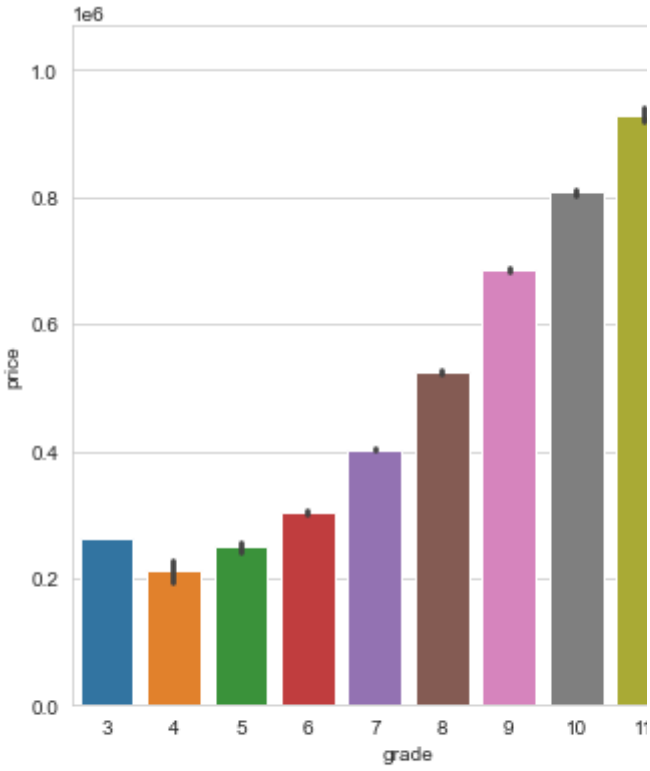
In [153]:
```
1 g = sns.lmplot(data=df_iqrp, x="sqf
```

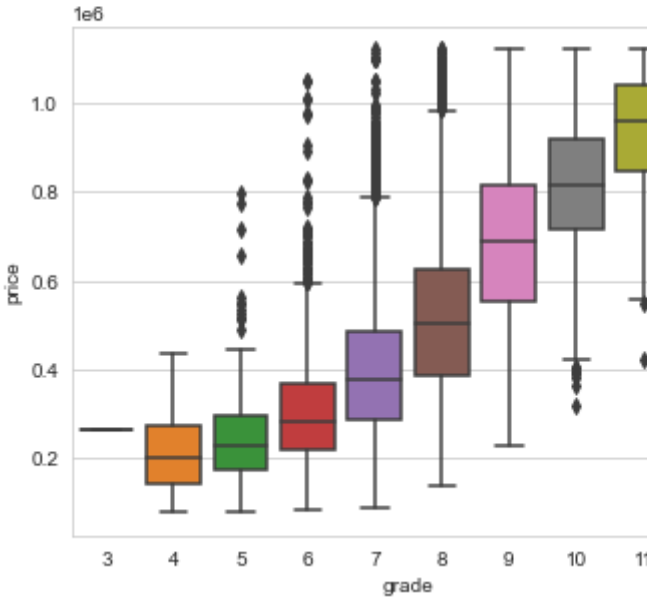


Contents

In [154]:
```
fig, axs=plt.subplots(nrows=2, ncol

sns.barplot(data=df_iqrp, x='grade'
sns.boxplot(data=df_iqrp, x='grade'
```

Out[154]: <AxesSubplot:xlabel='grade', ylabel='pr





Contents

In [ ]:
```
1
```