

In [1]:

```
1 %%html
2 <style>
3 h1 {padding: 9px; color: Green; border-bottom: 3px solid Green; text-align: center;}
4 h2 {padding: 9px; color: MediumBlue; border-bottom: 3px solid MediumBlue; text-align: center;}
5 h3 {padding: 9px; color: firebrick; border-bottom: 3px solid firebrick; text-align: center;}
6 summary {color: DarkOrange;}
7 td {text-align: center;}
8 </style>
```

## Project Overview

---

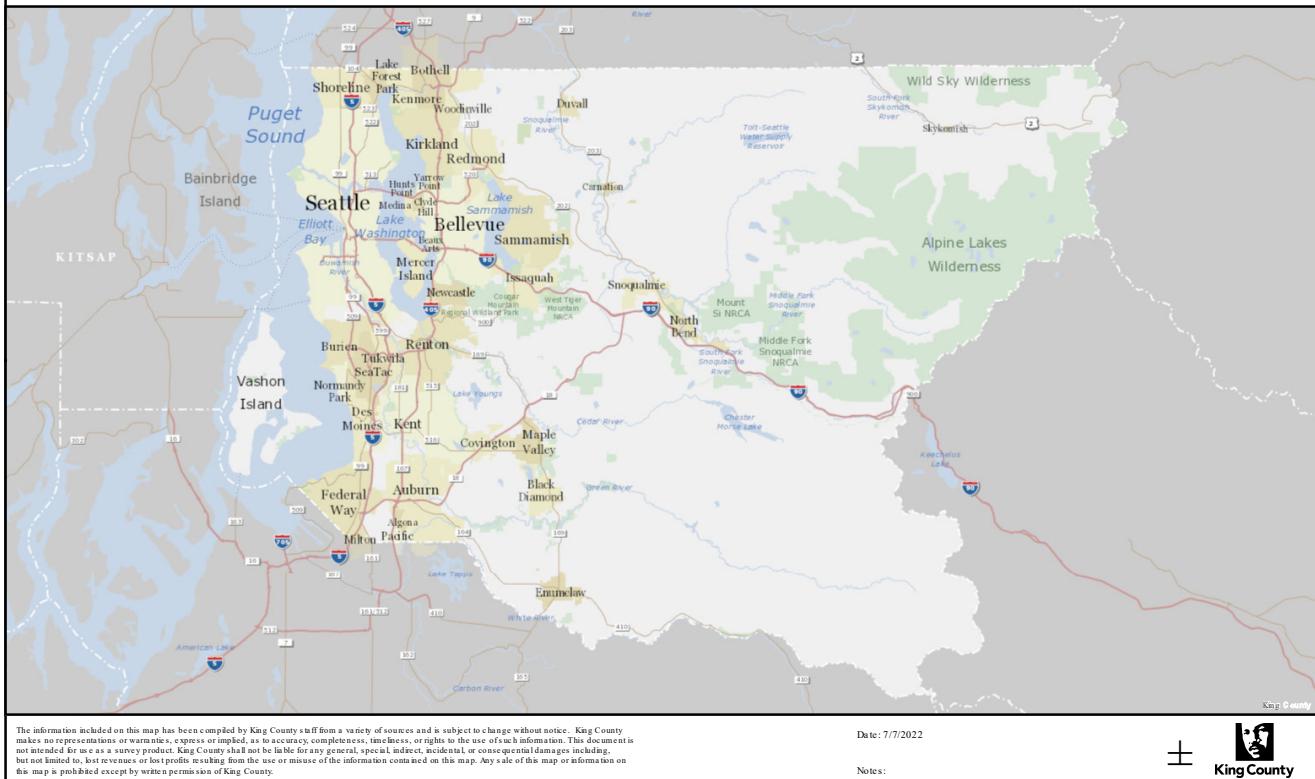
For this project, [Flatiron School](https://flatironschool.com/) (<https://flatironschool.com/>) provided me with a dataset of residential property sales in King County, Washington. We were instructed to create a hypothetical stakeholder, or client, and business problem, in contrast to the first project in which we were provided with both. For students who were struggling to define a stakeholder, they recommended a real estate agency that assists homeowners in selling their homes, with the specific business problem of providing the agency with the analysis necessary to tell their clients whether renovations really do increase the value of a house and if so, by how much.

Based on the project description, and prior to any exploration of the dataset, I already had real estate agency in mind as a client, but I didn't want to limit myself before I began my analysis. My initial thoughts were that areas with rapidly increasing home prices would be useful to a real estate agency, while areas with homes prices that were stagnant, or were perhaps even decreasing, would be useful to a state government agency, or maybe even some sort of charitable organization, as areas that may be in need of economic development or support. Even if I did end up going with a real estate agency, I was hoping to provide my hypothetical client with insights concerning other possible revelations that could be attained through a thorough analysis of the data. So, I began my analysis without identifying a client or business problem, with the purpose of seeing what was possible first.

Along with the dataset, Flatiron also provided me with a .md [file with the column names and a brief description of each column](#) ([https://github.com/sarnadpy32/king\\_county\\_development/blob/master/data/column\\_names.md](https://github.com/sarnadpy32/king_county_development/blob/master/data/column_names.md)). However, two of the columns, condition and grade, required me to go to a [glossary of terms](#) (<https://info.kingcounty.gov/assessor/esales/Glossary.aspx?type=1>) on the King County Assessor's website to see what the entities within the columns specifically meant, which can be seen in [this version of the file](#) ([https://github.com/sarnadpy32/king\\_county\\_development/blob/master/data/column\\_names\\_and\\_descriptions.md](https://github.com/sarnadpy32/king_county_development/blob/master/data/column_names_and_descriptions.md)). I've also included the columns and their descriptions below if you click on the collapsible section.

— [Click Here to see the Column Names and Descriptions.](#) —

I also used the opportunity to explore the site a little, and by doing so I was able to obtain a map of the county from the county government's [iMap](https://gismaps.kingcounty.gov/iMap/) (<https://gismaps.kingcounty.gov/iMap/>) feature.



## Table of Contents

- [Importing the Necessary Modules and Functions](#)
- [Exploring the Data](#)
- [Stakeholder and Business Problem Update](#)
- [EDA and Initial Preprocessing Steps](#)
- [Feature Distribution Visualizations](#)
- [Initial Correlation Examination](#)
- [Dummy Variable Creation](#)
- [Preprocessed DataFrames](#)
- [Base Models](#)
- [Full Models](#)
  - [All King County Full Model](#)
  - [Seattle Full Model](#)
  - [Outside Seattle Full Model](#)
- [Stakeholder and Business Problem Decision](#)
- [Insights and Conclusions](#)
- [Future Investigations](#)

## Importing the Necessary Modules and Functions

I started by importing the modules and functions that I knew I would most likely need. If I needed to import any others at any point, I would come back here and add them to this cell to keep everything organized. I also created some output formatters that I commonly use so that I wouldn't have to rewrite them each time.

In [2]:

```
1 import warnings
2 warnings.simplefilter(action='ignore', category=FutureWarning)
3
4 import os
5
6 import numpy as np
7 import pandas as pd
8 import scipy.stats as stats
9 import seaborn as sns
10 import matplotlib.pyplot as plt
11 import matplotlib.patches as mpatches
12 import matplotlib.lines as mlines
13 from matplotlib.colors import LinearSegmentedColormap
14 %matplotlib inline
15
16 import statsmodels.api as sm
17 from statsmodels.formula.api import ols
18 from statsmodels.stats.outliers_influence import variance_inflation_factor
19
20 from sklearn.impute import MissingIndicator
21 from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
22
23 from sklearn.feature_selection import RFECV
24 from sklearn.linear_model import LinearRegression
25 from sklearn.model_selection import ShuffleSplit
```

In [3]:

```
1 bold_red = '\033[31m\033[1m'
2 every_off = '\033[0m'
3
4 print(bold_red + 'Making sure ' + every_off + 'they worked ' + bold_red + 'as intended.' + every_off)
```

Making sure they worked as intended.

## Exploring the Data

---

I began my analysis by importing the dataset provided to me by [Flatiron School \(<https://flatironschool.com/>\)](https://flatironschool.com/). I decided to sort the dataframe by `id`, as it would make identifying any potential duplicates easier. During my analysis, I also found out that there were 70 different ZIP codes, so I downloaded the free ZIP code database available at [this page \(<https://www.unitedstateszipcodes.org/zip-code-database/>\)](https://www.unitedstateszipcodes.org/zip-code-database/), to replace the ZIP codes with the appropriate city names. This reduced the number of categories in the column to 24 and served as a more useful tool for analysis.

In [4]:

```

1 kc_house_data = pd.read_csv('data/kc_house_data.csv')
2 kc_house_data = kc_house_data.sort_values('id')
3
4 display(kc_house_data.head(3))
5 kc_house_data.info()

```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_
2495	1000102	4/22/2015	300000.0	6	3.00	2400	9373	2.0	NO	NONE	...	7	Average	2400
2494	1000102	9/16/2014	280000.0	6	3.00	2400	9373	2.0	NaN	NONE	...	7	Average	2400
6729	1200019	5/8/2014	647500.0	4	1.75	2060	26036	1.0	NaN	NONE	...	8	Good	1160

3 rows × 21 columns



```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 2495 to 15937
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id               21597 non-null   int64  
 1   date              21597 non-null   object  
 2   price             21597 non-null   float64 
 3   bedrooms          21597 non-null   int64  
 4   bathrooms         21597 non-null   float64 
 5   sqft_living       21597 non-null   int64  
 6   sqft_lot          21597 non-null   int64  
 7   floors            21597 non-null   float64 
 8   waterfront        19221 non-null   object  
 9   view              21534 non-null   object  
 10  condition         21597 non-null   object  
 11  grade             21597 non-null   object  
 12  sqft_above        21597 non-null   int64  
 13  sqft_basement    21597 non-null   object  
 14  yr_built          21597 non-null   int64  
 15  yr_renovated     17755 non-null   float64 
 16  zipcode           21597 non-null   int64  
 17  lat               21597 non-null   float64 
 18  long              21597 non-null   float64 
 19  sqft_living15     21597 non-null   int64  
 20  sqft_lot15        21597 non-null   int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 3.6+ MB

```

In [5]:

```

1 kc_house_data.zipcode.nunique()

```

Out[5]: 70

```
In [6]: 1 zip_code_data = pd.read_csv('data/zip_code_database.csv')
2
3 zip_code_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42724 entries, 0 to 42723
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
0    zip              42724 non-null   int64  
1    type             42724 non-null   object  
2    decommissioned   42724 non-null   int64  
3    primary_city     42724 non-null   object  
4    acceptable_cities 9287 non-null   object  
5    unacceptable_cities 11666 non-null   object  
6    state            42724 non-null   object  
7    county           41799 non-null   object  
8    timezone          41926 non-null   object  
9    area_codes        39698 non-null   object  
10   world_region     333 non-null    object  
11   country          42650 non-null   object  
12   latitude          42724 non-null   float64 
13   longitude         42724 non-null   float64 
14   irs_estimated_population 42724 non-null   int64  
dtypes: float64(2), int64(3), object(10)
memory usage: 4.9+ MB
```

```
In [7]: 1 zip_cities = zip_code_data[['zip', 'primary_city']].set_index('zip')
```

```
In [8]: 1 kc_house_data.zipcode = \
2 kc_house_data.zipcode.map(lambda x: zip_cities.loc[x]['primary_city'])
```

```
In [9]: 1 kc_house_data.rename(columns={'zipcode': 'city'}, inplace=True)
```

```
In [10]: 1 kc_house_data.city.nunique()
```

Out[10]: 24

## Duplicated ID's

---

I found out there were sales with duplicate `id`s. Before I could begin my analysis, I had to determine if these were resales or errors? I first changed the `date` column to make comparisons easier, and it also allowed me to easily see the period of time covered by the dataset, which was only a single year's worth of sales, specifically from May 2014 to May 2015. I then isolated all the duplicated `id`s into a single dataframe to investigate my concerns. After checking, they do seem to be resales and there were therefore no duplicates that needed to be dropped.

```
In [11]: 1 kc_house_data.id.nunique()
```

Out[11]: 21420

```
In [12]: 1 kc_house_data.date = pd.to_datetime(kc_house_data.date)
```

```
In [13]: 1 kc_house_data.date.describe(datetime_is_numeric=True)
```

```
Out[13]: count          21597
mean      2014-10-29 04:20:38.171968512
min      2014-05-02 00:00:00
25%      2014-07-22 00:00:00
50%      2014-10-16 00:00:00
75%      2015-02-17 00:00:00
max      2015-05-27 00:00:00
Name: date, dtype: object
```

```
In [14]: 1 dup_df = kc_house_data.loc[kc_house_data.id.duplicated(keep=False)].copy()
```

```
In [15]: 1 dup_df.id.value_counts().head(3)
```

```
Out[15]: 795000620    3
1000102     2
5430300171    2
Name: id, dtype: int64
```

In [16]:

```

1 for id_x in dup_df.id.unique():
2     uniq_id_df = dup_df.loc[dup_df.id==id_x].sort_values('date')
3
4     p_0, p_1 = uniq_id_df.price.iloc[0], uniq_id_df.price.iloc[1]
5
6     d_0, d_1 = uniq_id_df.date.iloc[0], uniq_id_df.date.iloc[1]
7
8     # Any duplicated `id`s that had a resale value of less than the orginal value was my primary
9     # concern that these could be errors instead of resales so I didn't feel the need to check
10    # if the resale value was greater, which would make sense if people were quickly flipping
11    # houses / properties
12    #-----
13    if p_1 < p_0: display(uniq_id_df)
14
15    if d_0 == d_1:
16        print('\n'+ bold_red +'X---Obvious Error---X'+ every_off +'\n')
17        display(uniq_id_df)
18
19    if len(uniq_id_df)==3: display(uniq_id_df)

```

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>...</b>	<b>grade</b>	<b>sqft_above</b>	<b>sqft_ba</b>
17588	795000620	2014-09-24	115000.0	3	1.0	1080	6250	1.0	NO	NONE	...	5	Fair	1080
17589	795000620	2014-12-15	124000.0	3	1.0	1080	6250	1.0	NO	NONE	...	5	Fair	1080
17590	795000620	2015-03-11	157000.0	3	1.0	1080	6250	1.0	NaN	NONE	...	5	Fair	1080

3 rows × 21 columns

--	--

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>...</b>	<b>grade</b>	<b>sqft_above</b>	<b>sqft_ba</b>
15263	2619920170	2014-10-01	772500.0	4	2.5	3230	4290	2.0	NO	NONE	...	9	Better	3230
15264	2619920170	2014-12-19	765000.0	4	2.5	3230	4290	2.0	NO	NONE	...	9	Better	3230

2 rows × 21 columns

--	--

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>...</b>	<b>grade</b>	<b>sqft_above</b>	<b>sqft_ba</b>
5587	2726049071	2014-12-11	510000.0	2	1.0	820	4206	1.0	NaN	NONE	...	5	Fair	820
5588	2726049071	2015-04-08	489950.0	2	1.0	820	4206	1.0	NO	NONE	...	5	Fair	820

2 rows × 21 columns

--	--

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>...</b>	<b>grade</b>	<b>sqft_above</b>	<b>sqft_ba</b>
10598	2767603612	2014-05-12	500000.0	2	2.25	1290	1334	3.0	NO	NONE	...	8	Good	1290
10599	2767603612	2015-01-13	489000.0	2	2.25	1290	1334	3.0	NO	NONE	...	8	Good	1290

2 rows × 21 columns

--	--

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	<b>...</b>	<b>grade</b>	<b>sqft_above</b>
--	-----------	-------------	--------------	-----------------	------------------	--------------------	-----------------	---------------	-------------------	-------------	------------	--------------	-------------------

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	...	<b>grade</b>	<b>sqft_above</b>
7838	4139420590	2014-05-20	1210000.0	4	3.5	4560	16643	1.0	NO	GOOD	...	12 Luxury	2230
7839	4139420590	2014-08-27	1200000.0	4	3.5	4560	16643	1.0	NO	GOOD	...	12 Luxury	2230

2 rows × 21 columns

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	...	<b>grade</b>	<b>sqft_above</b>	<b>sqft_bas</b>
13285	7167000040	2014-08-13	740000.0	4	3.0	3350	199253	2.0	Nan	NONE	...	10 Very Good	3350	
13286	7167000040	2015-03-05	700000.0	4	3.0	3350	199253	2.0	NO	NONE	...	10 Very Good	3350	

2 rows × 21 columns

	<b>id</b>	<b>date</b>	<b>price</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>sqft_living</b>	<b>sqft_lot</b>	<b>floors</b>	<b>waterfront</b>	<b>view</b>	...	<b>grade</b>	<b>sqft_above</b>	<b>sqft_bas</b>
836	8682262400	2014-07-18	430000.0	2	1.75	1350	4003	1.0	NO	NONE	...	8 Good	1350	
837	8682262400	2015-05-13	419950.0	2	1.75	1350	4003	1.0	NO	NONE	...	8 Good	1350	

2 rows × 21 columns

## Stakeholder and Business Problem Update

As I mentioned in the section above, I discovered that the period of time covered by the dataset was only a single year. My original idea of analyzing the prices of houses in certain areas over time was therefore not going to work, nor any other analysis of the changing effects of the other features over time. While my options for a client were still open, I was going to have to perform an analysis of the effect of the features themselves on the prices of houses in the year concerned.

## EDA and Initial Preprocessing Steps

I started by identifying the columns with missing data. Since there were only 63 entries missing a value in the `view` column, I decided to just drop those entries. I then used the `MissingIndicator()` function from `sklearn` to create an encoded column to be added to the DataFrame for each of the columns the held missing data. I could then fill in the `NaN`'s of those columns with something appropriate so that they wouldn't cause problems during my analysis. For the `waterfront` column, I then used the `OrdinalEncoder()` from `sklearn` to code and replace the column in a binary format.

```
In [17]: 1 kc_cols_missing = kc_house_data.isna().sum()
          2 kc_cols_missing.loc[kc_cols_missing != 0]
```

```
Out[17]: waterfront      2376
         view            63
         yr_renovated    3842
         dtype: int64
```

```

In [18]: 1 kc_house_data.dropna(subset=['view'], inplace=True)
2
3 for col in ['waterfront', 'yr_renovated']:
4
5     col_ser = kc_house_data[[col]].copy()
6     missing_indicator = MissingIndicator()
7     missing_indicator.fit(col_ser)
8     missing_ser = missing_indicator.transform(col_ser).astype(int)
9
10    if col == 'waterfront':
11        col_ser[col].fillna('NO', inplace=True)
12        ord_encoder = OrdinalEncoder(dtype='int64')
13        ord_encoder.fit(col_ser)
14        enc_ord_col_ser = ord_encoder.transform(col_ser).flatten()
15        kc_house_data[col] = enc_ord_col_ser
16
17    if col == 'yr_renovated': kc_house_data[col].fillna(0, inplace=True)
18
19    kc_house_data[col + '_Missing'] = missing_ser

```

I found out later that the `sqft_basement` column also contained a value that was basically equivalent to `NaN`. I came back here to deal with it in the same section.

```

In [19]: 1 kc_house_data.sqft_basement = \
2 kc_house_data.sqft_basement.map(lambda x: np.nan if x=='?' else x)
3
4 sqft_base_col = kc_house_data[['sqft_basement']]
5
6 missing_indicator = MissingIndicator()
7 missing_indicator.fit(sqft_base_col)
8 missing_ser = missing_indicator.transform(sqft_base_col).astype(int)
9
10 kc_house_data['sqft_basement_Missing'] = missing_ser
11
12 kc_house_data.sqft_basement = kc_house_data.sqft_basement.astype('float64')

```

I then dropped the `id`, `date`, `lat`, and `long` columns as they were no longer necessary for my analysis. The `date`, `lat`, and `long` columns were all inappropriate to use as parameters in the model at this point, and the `id` column is inappropriate for linear models in general.

```

In [20]: 1 kc_house_data.drop(['id', 'date', 'lat', 'long'], axis=1, inplace=True)

```

Then I split the data into the target (dependent variable) series and two separate dataframes for the dependent variables, one for the numerical dependent variables, and one for the categorical dependent variables.

```

In [21]: 1 numericals = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_living15', 'sqft_lot15', 'yr_built', 'yr_renov
2             'floors', 'sqft_basement', 'bedrooms', 'bathrooms']
3
4 categoricals = [col for col in kc_house_data.columns if col not in numericals and col != 'price']

```

```

In [22]: 1 kc_target = kc_house_data['price']
2 kc_nums = kc_house_data[numericals].copy()
3 kc_cats = kc_house_data[categoricals].copy()

```

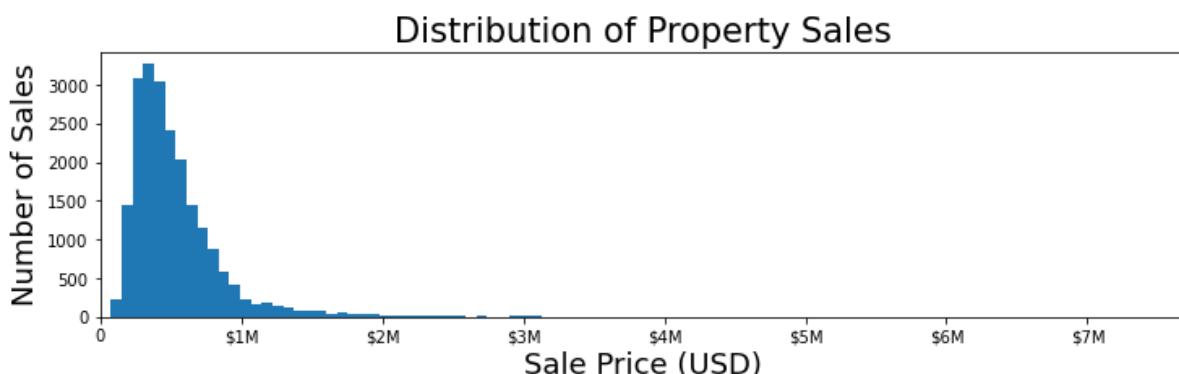
## Feature Distribution Visualizations

### Price Distribution Visualization

I first checked the distribution of price (the target variable). Before I created any visualizations, I wrote functions to properly format the ticks of any visualizations and any pandas outputs that contained currency information, as well a function to get a lighter version of a simple, prenamed color to use in visualizations as needed.

```
In [23]: 1 plt.rcParams.update({'mathtext.fontset':'dejavuserif', 'mathtext.bf':'serif:bold'})  
2  
3 def get_lighter_color(color_name, light_frac):  
4     color_map = LinearSegmentedColormap.from_list('', [color_name, 'w'], N=9)  
5     return color_map(light_frac)  
6  
7 def viz_percentage_formatter(y, pos):  
8     if y==0: c = '$\\mathbf{0\\%}$'  
9     else: c = '$\\mathbf{+' + '{:.2f}'.format(y * 100) + '\\%}$'  
10  
11    return c  
12  
13 def viz_currency_formatter(x, pos):  
14     if x==0: c = '0'  
15     if 0<np.abs(x)<1e3: c = '${}'.format(np.abs(x))  
16     if 1e3<=np.abs(x)<1e6: c = '${}K'.format(round(np.abs(x)*1e-3), 2)  
17     if 1e6<=np.abs(x)<1e9: c = '${}M'.format(round(np.abs(x)*1e-6), 2)  
18     if np.abs(x)>=1e9: c = '${}B'.format(round(np.abs(x)*1e-9), 2)  
19  
20     if x < 0: c = '-'+c  
21  
22    return c  
23  
24 def pd_currency_formatter(x):  
25     if x==0: c = '0'  
26     if 0<np.abs(x)<1e3: c = '${:f}'.format(np.abs(x))  
27     if 1e3<=np.abs(x)<1e6: c = '${:f}K'.format(round(np.abs(x)*1e-3), 2)  
28     if 1e6<=np.abs(x)<1e9: c = '${:f}M'.format(round(np.abs(x)*1e-6), 2)  
29     if np.abs(x)>=1e9: c = '${:f}B'.format(round(np.abs(x)*1e-9), 2)  
30  
31     if x < 0: c = '-'+c  
32  
33    return c
```

```
In [24]: 1 p_max = kc_target.max()  
2  
3 fig, ax = plt.subplots(figsize=(12, 3))  
4  
5 ax.hist(kc_target, bins=100)  
6  
7 ax.set_xbound(0, p_max)  
8 ax.xaxis.set_major_formatter(viz_currency_formatter)  
9  
10 ax.set_xlabel("Sale Price (USD)", size=18)  
11 ax.set_ylabel("Number of Sales", size=18)  
12 ax.set_title("Distribution of Property Sales", size=21)  
13  
14 fig.set_facecolor('w')  
15  
16 fig.savefig('visuals/price_distribution.png' , bbox_inches='tight')  
17  
18 plt.show()
```



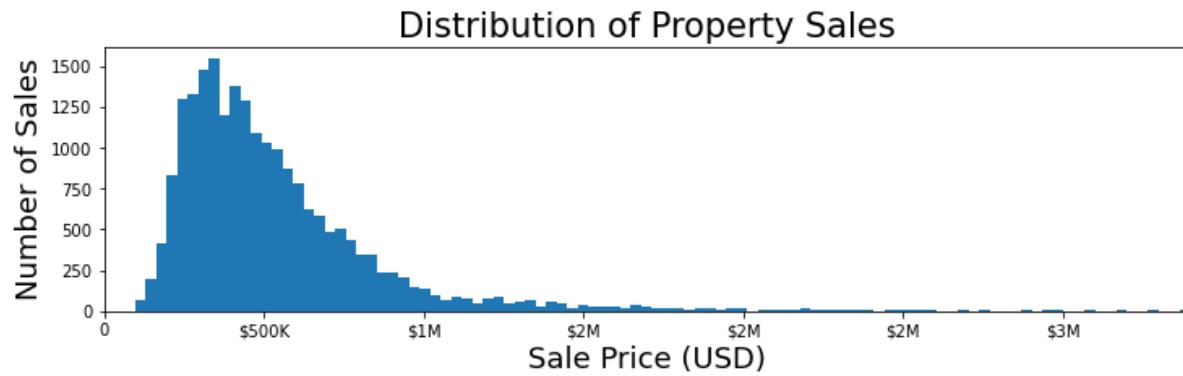
```
In [25]: 1 kc_target.describe().apply(pd_currency_formatter)
```

```
Out[25]: count      $22.000000K  
mean      $540.000000K  
std       $366.000000K  
min       $78.000000K  
25%      $322.000000K  
50%      $450.000000K  
75%      $645.000000K  
max       $8.000000M  
Name: price, dtype: object
```

There seemed to be a heavy skew which was confirmed by the large difference between the 75th quantile and the max value for the series. I eliminated the outliers so that the performance of the model would be improved.

```
In [26]: 1 kc_target = kc_target.loc[(kc_target >= kc_target.quantile(.001)) & (kc_target <= kc_target.quantile(.999))]
```

```
In [27]: 1 p_max = kc_target.max()  
2  
3 fig, ax = plt.subplots(figsize=(12, 3))  
4  
5 ax.hist(kc_target, bins=100)  
6  
7 ax.set_xbound(0, p_max)  
8 ax.xaxis.set_major_formatter(viz_currency_formatter)  
9  
10 ax.set_xlabel("Sale Price (USD)", size=18)  
11 ax.set_ylabel("Number of Sales", size=18)  
12 ax.set_title("Distribution of Property Sales", size=21)  
13  
14 fig.set_facecolor('w')  
15  
16 fig.savefig('visuals/price_distribution_2.png' , bbox_inches='tight')  
17  
18 plt.show()
```

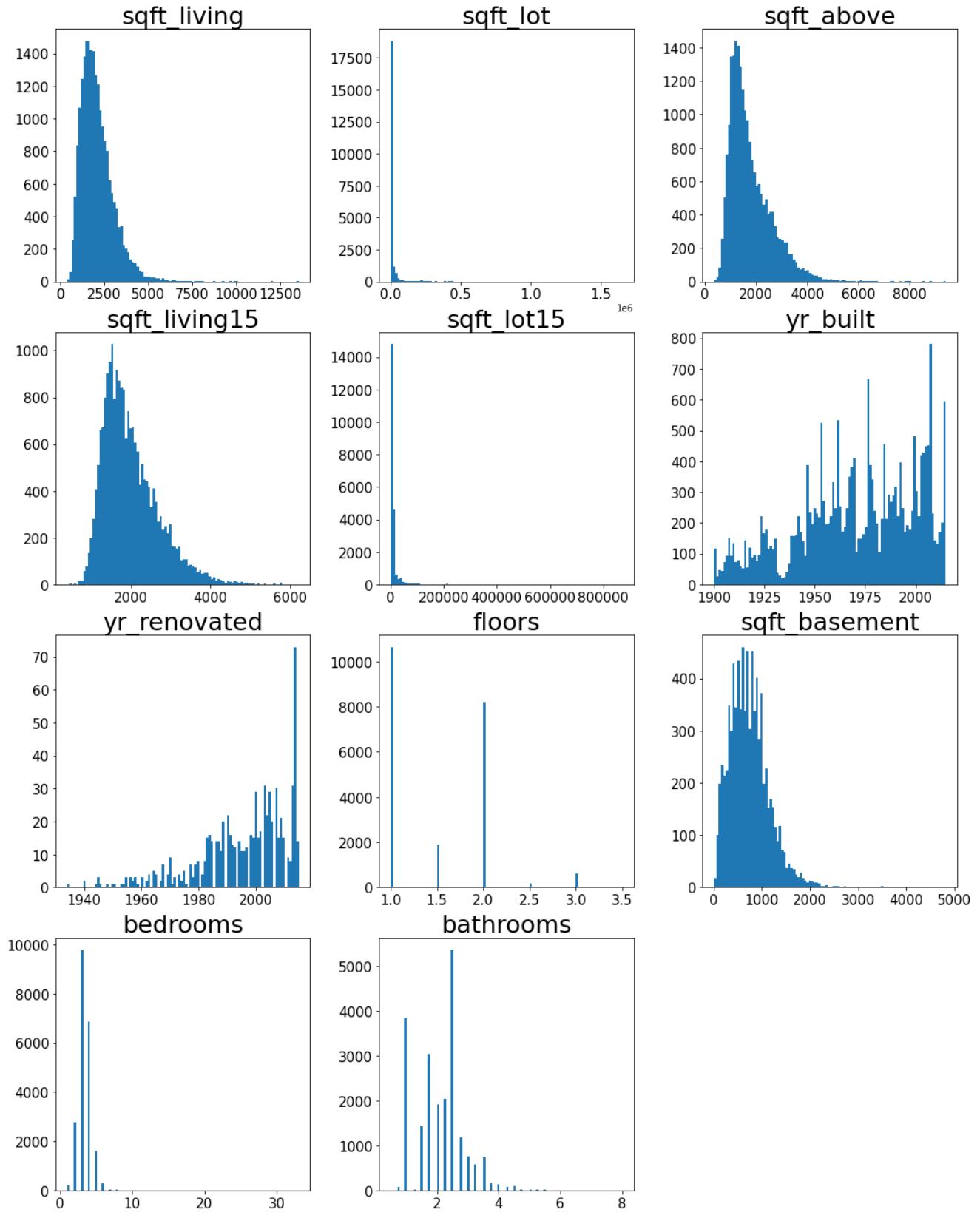


## Numerical Features Distribution Visualizations

I then created visualizations to explore the distributions of the numerical features.

In [28]:

```
1 nums = len(kc_nums.columns)
2 n_rows = int(nums / 2)#
3
4 fig, axs = plt.subplots(nrows=n_rows, ncols=3, figsize=(18, n_rows * 6),
5                         gridspec_kw={'wspace': .27})
6 axs = axs.flatten()
7
8 for c_i, col in enumerate(kc_nums.columns):
9
10    axs[c_i].hist(kc_nums[col], bins=100) # INITIAL EXPLORATION
11
12    # Comment the following IF...ELSE statement and uncomment the line above to see the original
13    # distributions
14    #-----
15    if col not in ['sqft_lot', 'sqft_basement', 'sqft_lot15', 'yr_renovated']:
16        axs[c_i].hist(kc_nums[col], bins=100)
17
18    else: axs[c_i].hist(kc_nums[col].loc[kc_nums[col] != 0], bins=100)
19    #-----
20    axs[c_i].set_title(col, size=27)
21    axs[c_i].tick_params('both', labelsize=15)
22
23 fig.set_facecolor('w')
24 [ax.set_visible(False) for ax in axs if not ax.has_data()]
25
26 fig.savefig('visuals/numeric_features_distributions.png' , bbox_inches='tight')
27
28 plt.show()
```



There seemed to be spikes at 0 for the columns `sqft_lot`, `sqft_basement`, `sqft_lot15`, and `yr_renovated`. By eliminating the 0's in the distribution visualizations for those columns, I determined that the `sqft_basement` and `yr_renovated` columns were heavily affected by removing the 0's, while the other two were not as clearly affected. It was therefore useful to turn the `sqft_basement` and `yr_renovated` into categorical features by either turning them into binary encoded features, or by splitting them into more complex categorical features.

I initially explored turning them into more complex categorical features, but in the end, I decided to just turn them into binary features.

```
In [29]: 1 kc_nums.sqft_basement = kc_nums.sqft_basement.map(lambda x: 1 if x!=0 else x).astype(int)
```

```
In [30]: 1 kc_nums.yr_renovated = kc_nums.yr_renovated.map(lambda x: 1 if x!=0 else x).astype(int)
```

I decided to turn the `yr_built` column into a categorical feature by splitting the values into groups of every 20 years. I also knew that I would be sorting some of the `xticklabels()` of the [Categorical Feature Distribution Visualizations](#) using dictionaries, as well as making sure the dummy variables were in the correct order when I encoded the categorical features, so I just created one for the `yr_built` column here. I will address this change again later in my [Insights and Conclusions](#) section.

```
In [31]: 1 built_min = kc_nums.yr_built.loc[kc_nums.yr_built!=0].min()
2 built_max = kc_nums.yr_built.max()
3
4 print(bold_red + 'min' + every_off + ' :', built_min, '\t' + bold_red + 'max' + every_off + ' :', built_max)
```

```
min : 1900      max : 2015
```

```
In [32]: 1 yr_range = np.arange(1900, 2001, 20)
2
3 built_dict = {}
4 for y_i, yr in enumerate(yr_range):
5     yr_20 = yr + 20
6     yr_str = str(yr) + '_to_' + str(yr_20) + '_s'
7     built_dict[yr_str] = \
8         kc_nums.yr_built.map(lambda x: True if yr <= x < yr_20 else np.nan).count()
9
10 built_dict
```

```
Out[32]: {'1900_to_1920_s': 1446,
 '1920_to_1940_s': 1717,
 '1940_to_1960_s': 4198,
 '1960_to_1980_s': 4932,
 '1980_to_2000_s': 4498,
 '2000_to_2020_s': 4743}
```

```
In [33]: 1 yr_range = np.arange(1900, 2001, 20)
2
3 built_dict = {}
4 for y_i, yr in enumerate(yr_range):
5     yr_20 = yr + 20
6     yr_str = str(yr) + '_to_' + str(yr_20) + '_s'
7     built_dict[yr_str] = y_i + 1
8     kc_nums.yr_built = \
9         kc_nums.yr_built.map(lambda x: yr_str if type(x)!=str and yr <= x < yr_20 else x)
```

I then switched the three columns I had restructured from the numerical features dataframe to the categorical features dataframe.

```
In [34]: 1 kc_cats.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21534 entries, 2495 to 15937
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   waterfront      21534 non-null    int64  
 1   view            21534 non-null    object 
 2   condition       21534 non-null    object 
 3   grade           21534 non-null    object 
 4   city            21534 non-null    object 
 5   waterfront_Missing  21534 non-null    int32  
 6   yr_renovated_Missing 21534 non-null    int32  
 7   sqft_basement_Missing 21534 non-null    int32  
dtypes: int32(3), int64(1), object(4)
memory usage: 1.2+ MB
```

```
In [35]: 1 kc_cats.insert(5, 'basement', kc_nums.sqft_basement)
2 kc_cats.insert(6, 'yr_built', kc_nums.yr_built)
3 kc_cats.insert(7, 'renovated', kc_nums.yr_renovated)
4
5 kc_nums.drop(['sqft_basement', 'yr_built', 'yr_renovated'], axis=1, inplace=True)
```

I again eliminated any outliers to improve the performance of the model.

```
In [36]: 1 kc_nums.describe()
```

Out[36]:

	sqft_living	sqft_lot	sqft_above	sqft_living15	sqft_lot15	floors	bedrooms	bathrooms
<b>count</b>	21534.000000	2.153400e+04	21534.000000	21534.000000	21534.000000	21534.000000	21534.000000	21534.000000
<b>mean</b>	2079.827854	1.509060e+04	1788.557537	1986.299944	12751.079502	1.494126	3.373038	2.115712
<b>std</b>	917.446520	4.138021e+04	827.745641	685.121001	27255.483308	0.539806	0.926410	0.768602
<b>min</b>	370.000000	5.200000e+02	370.000000	399.000000	651.000000	1.000000	1.000000	0.500000
<b>25%</b>	1430.000000	5.040000e+03	1190.000000	1490.000000	5100.000000	1.000000	3.000000	1.750000
<b>50%</b>	1910.000000	7.617000e+03	1560.000000	1840.000000	7620.000000	1.500000	3.000000	2.250000
<b>75%</b>	2550.000000	1.068775e+04	2210.000000	2360.000000	10083.000000	2.000000	4.000000	2.500000
<b>max</b>	13540.000000	1.651359e+06	9410.000000	6210.000000	871200.000000	3.500000	33.000000	8.000000

```
In [37]: 1 for n_col in kc_nums.columns:
```

```
2     kc_nums = \
3     kc_nums.loc[(kc_nums[n_col] >= kc_nums[n_col].quantile(.001)) & (kc_nums[n_col] <= kc_nums[n_col].quantile(.999))]
```

```
In [38]: 1 kc_nums.describe()
```

Out[38]:

	sqft_living	sqft_lot	sqft_above	sqft_living15	sqft_lot15	floors	bedrooms	bathrooms
<b>count</b>	21279.000000	21279.000000	21279.000000	21279.000000	21279.000000	21279.000000	21279.000000	21279.000000
<b>mean</b>	2067.101744	13951.586400	1777.229334	1982.012595	12090.615113	1.490883	3.370929	2.106831
<b>std</b>	869.249044	30476.339079	789.855994	668.130370	22379.965811	0.537670	0.884038	0.744401
<b>min</b>	570.000000	713.000000	570.000000	750.000000	914.000000	1.000000	1.000000	0.750000
<b>25%</b>	1430.000000	5041.000000	1200.000000	1490.000000	5100.000000	1.000000	3.000000	1.750000
<b>50%</b>	1910.000000	7603.000000	1560.000000	1840.000000	7620.000000	1.500000	3.000000	2.250000
<b>75%</b>	2540.000000	10584.000000	2200.000000	2360.000000	10035.000000	2.000000	4.000000	2.500000
<b>max</b>	7100.000000	493534.000000	5530.000000	4920.000000	283140.000000	3.000000	8.000000	5.000000

## Categorical Features Distribution Visualizations

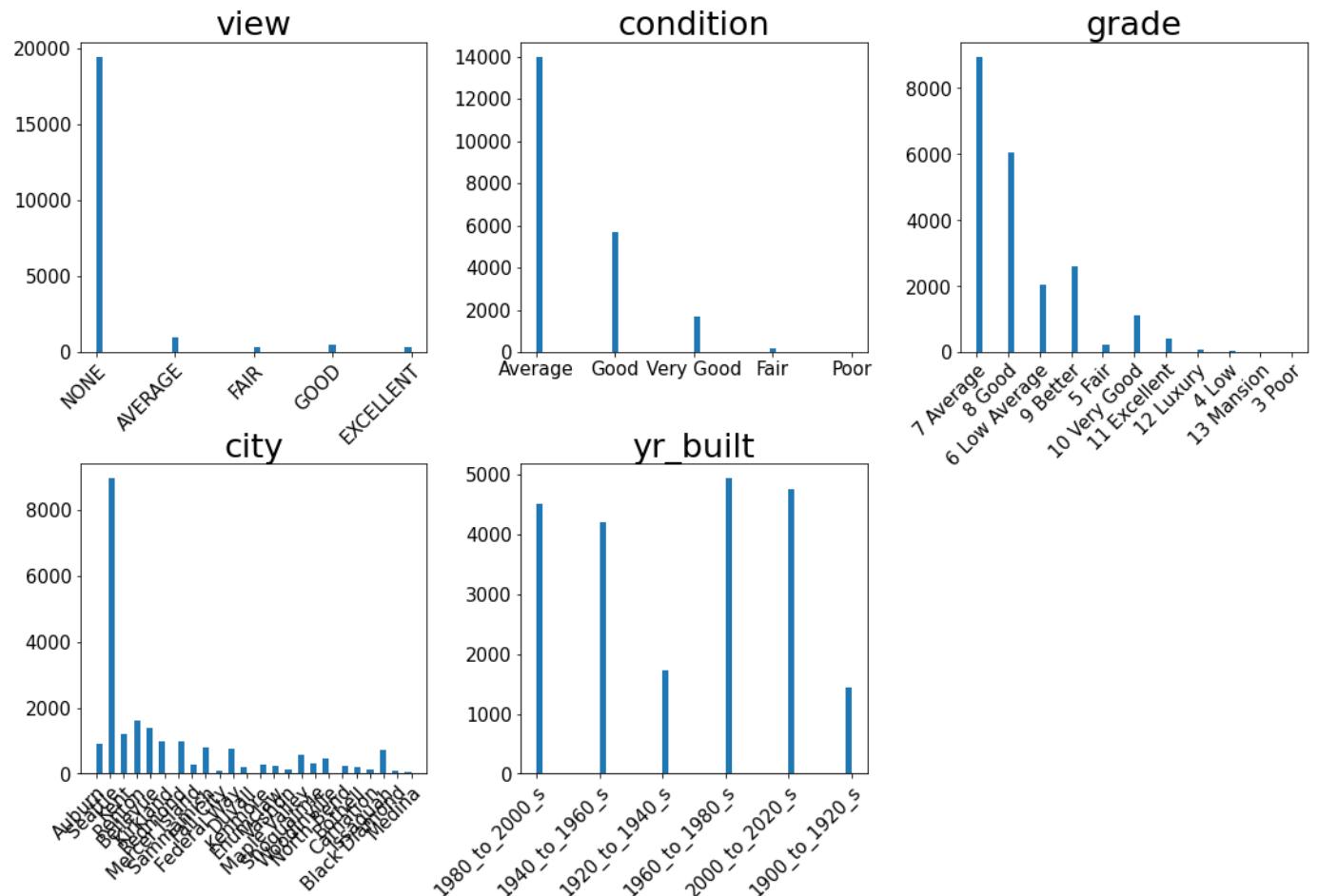
After the numerical features were taken care of, I checked the distribution of the categorical variables. These would have to be encoded in an appropriate manner to be used in my model.

In [39]:

```

1 cats = len(kc_cats.columns)
2 n_rows = int(cats / 2)
3
4 fig, axs = plt.subplots(nrows=n_rows, ncols=3, figsize=(18, n_rows * 6),
5                         gridspec_kw={'wspace': .27, 'hspace': .36})
6 axs = axs.flatten()
7
8 only_2_vals_df = pd.DataFrame()
9 empty_i = 0
10 for c_i, cat in enumerate(kc_cats.columns):
11
12     ax_i = c_i if empty_i == 0 else c_i - empty_i
13
14     if kc_cats[cat].nunique() > 2:
15         axs[ax_i].hist(kc_cats[cat], bins=50)
16         axs[ax_i].set_title(cat, size=27)
17         axs[ax_i].tick_params('both', labelsize=15)
18
19     if cat in ['grade', 'view', 'city', 'yr_renovated', 'yr_built', 'sqft_basement']:
20         plt.setp(axs[ax_i].get_xticklabels(), rotation=45, rotation_mode='anchor', ha='right', va='top')
21     else:
22         empty_i += 1
23     only_2_vals_df = pd.concat([only_2_vals_df, kc_cats[cat].value_counts()], axis=1)
24
25 fig.set_facecolor('w')
26 [ax.set_visible(False) for ax in axs if not ax.has_data()]
27
28 fig.savefig('visuals/categorical_features_distributions.png', bbox_inches='tight')
29
30 plt.show()
31
32 display(only_2_vals_df)

```



	waterfront	basement	renovated	waterfront_Missing	yr_renovated_Missing	sqft_basement_Missing
0	21389	12798	20791	19164	17704	21082
1	145	8736	743	2370	3830	452

I found out later that empty spaces in the values of a column caused problems during my analysis. Also, if you look at the visualizations above, you can see that the `xticklabels` in the visualizations for the distributions of `grade`, `condition`, `view`, and `yr_built` seem to be out of order.

I therefore figured this would be an appropriate place to eliminate the spaces in the values of the `grade`, `condition`, and `city` columns, as well as creating the sorting dictionaries I previously mentioned for the `grade`, `condition`, and `view` columns so that I could sort their `xticklabels`, as well as their dummy columns.

In [40]:

```
1 kc_cats.grade = kc_cats.grade.map(lambda x: x.replace(' ', '_'))
2 kc_cats.condition = kc_cats.condition.map(lambda x: x.replace(' ', '_'))
3 kc_cats.city = kc_cats.city.map(lambda x: x.replace(' ', '_'))
```

In [41]:

```
1 grade_dict = {'3_Poor':1, '4_Low':2, '5_Fair':3, '6_Low_Average':4, '7_Average':5, '8_Good':6,
2             '9_Better':7, '10_Very_Good':8, '11_Excellent':9, '12_Luxury':10, '13_Mansion':11}
3
4 cond_dict = {'Poor':1, 'Fair':2, 'Average':3, 'Good':4, 'Very_Good':5}
5
6 view_dict = {'NONE':1, 'FAIR':2, 'AVERAGE':3, 'GOOD':4, 'EXCELLENT':5}
7
8 all_dicts = {'grade': grade_dict, 'condition': cond_dict, 'view': view_dict, 'yr_built': built_dict}
```

By analyzing the `value_counts()` for each categorical column, I found that there were some categories with less than 10 entries associated with them. I thought of these as the outliers of the categorical features, and I dropped them accordingly.

```
In [42]: 1 low_freq_cats = {}
2
3 for c_i, cat in enumerate(kc_cats.columns):
4     n_cats = kc_cats[cat].nunique()
5     cat_type = kc_cats[cat].dtype
6
7     if cat_type!=object and n_cats>2:
8         num_cats = kc_cats[cat].value_counts().sort_index()
9         low_cats = num_cats.loc[num_cats < 10].index
10        for l_cat in low_cats:
11            if cat not in low_freq_cats.keys(): low_freq_cats[cat] = [l_cat]
12            else: low_freq_cats[cat].append(l_cat)
13
14        print(bold_red + str(c_i) + ' - ' + cat + every_off +':\tnum_vals = ' + str(n_cats), cat_type)
15        print(list(num_cats.index), '\n')
16
17    if cat_type==object:
18        if cat!='city':
19            cat_order = list(all_dicts[cat].keys())
20            obj_cats = \
21                kc_cats[cat].value_counts().sort_index(key=lambda cat_col: cat_col.map(lambda x: cat_order.index(x)))
22        else: obj_cats = kc_cats[cat].value_counts()
23
24        low_cats = obj_cats.loc[obj_cats < 10].index
25        if not low_cats.empty:
26            for l_cat in low_cats:
27                if cat not in low_freq_cats.keys(): low_freq_cats[cat] = [l_cat]
28                else: low_freq_cats[cat].append(l_cat)
29
30        print(bold_red + str(c_i) + ' - ' + cat + every_off +':\tnum_vals = ' + str(n_cats) +'\tdtype =', cat_type)
31        print(list(obj_cats.index))
32        print()
```

```
1 - view:      num_vals = 5      dtype = object
['NONE', 'FAIR', 'AVERAGE', 'GOOD', 'EXCELLENT']

2 - condition: num_vals = 5      dtype = object
['Poor', 'Fair', 'Average', 'Good', 'Very_Good']

3 - grade:     num_vals = 11     dtype = object
['3_Poor', '4_Low', '5_Fair', '6_Low_Average', '7_Average', '8_Good', '9_Better', '10_Very_Good', '11_Excellen
t', '12_Luxury', '13_Mansion']

4 - city:      num_vals = 24     dtype = object
['Seattle', 'Renton', 'Bellevue', 'Kent', 'Kirkland', 'Redmond', 'Auburn', 'Sammamish', 'Federal_Way', 'Issaquah', 'Maple_Valley', 'Woodinville', 'Snoqualmie', 'Kenmore', 'Mercer_Island', 'Enumclaw', 'North_Bend', 'Bothell', 'Duvall', 'Carnation', 'Vashon', 'Black_Diamond', 'Fall_City', 'Medina']

6 - yr_built:  num_vals = 6      dtype = object
['1900_to_1920_s', '1920_to_1940_s', '1940_to_1960_s', '1960_to_1980_s', '1980_to_2000_s', '2000_to_2020_s']
```

```
In [43]: 1 low_freq_cats
```

```
Out[43]: {'grade': ['3_Poor']}
```

```
In [44]: 1 for cat, low_cats in low_freq_cats.items():
2     for l_cat in low_cats:
3         low_index = list(kc_cats.loc[kc_cats[cat] == l_cat].index)
4         if low_index: kc_cats.drop(low_index, inplace=True)
```

As I mentioned earlier, I found out later that empty spaces in the values of a column would cause problems in my analysis. This was also the case with .'.s. Once they were in str format, I could then use the .replace() function to replace the .'s with \_'s.

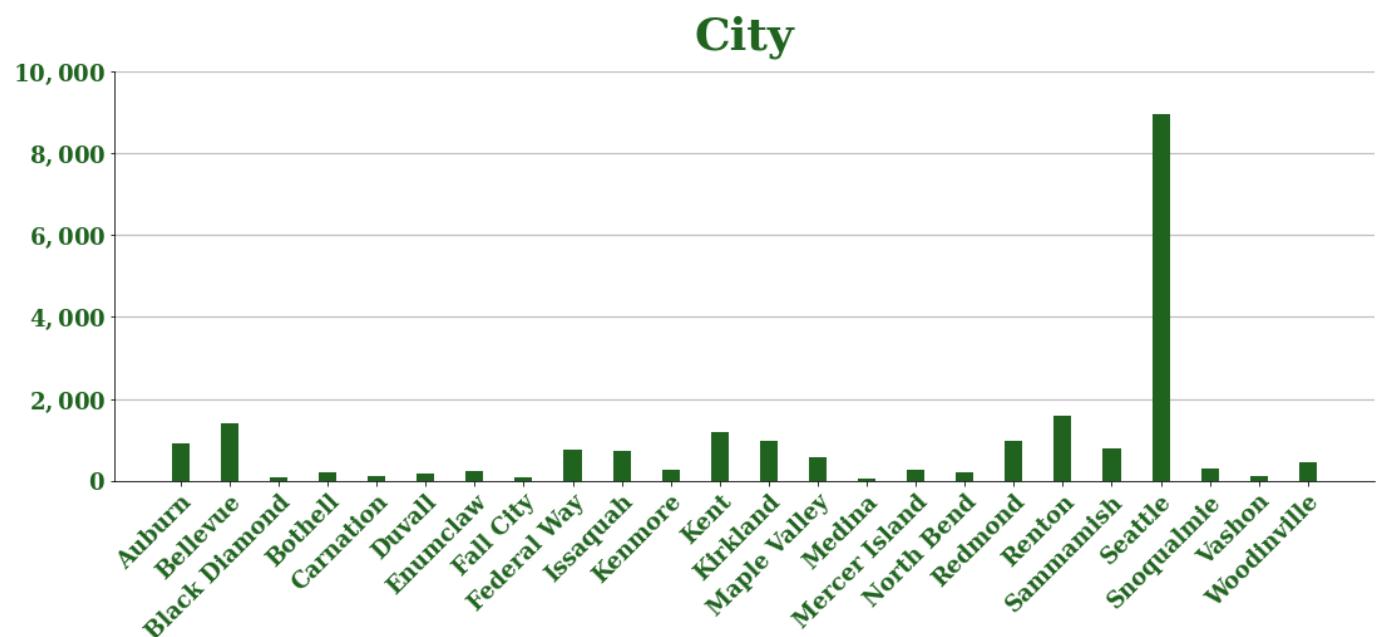
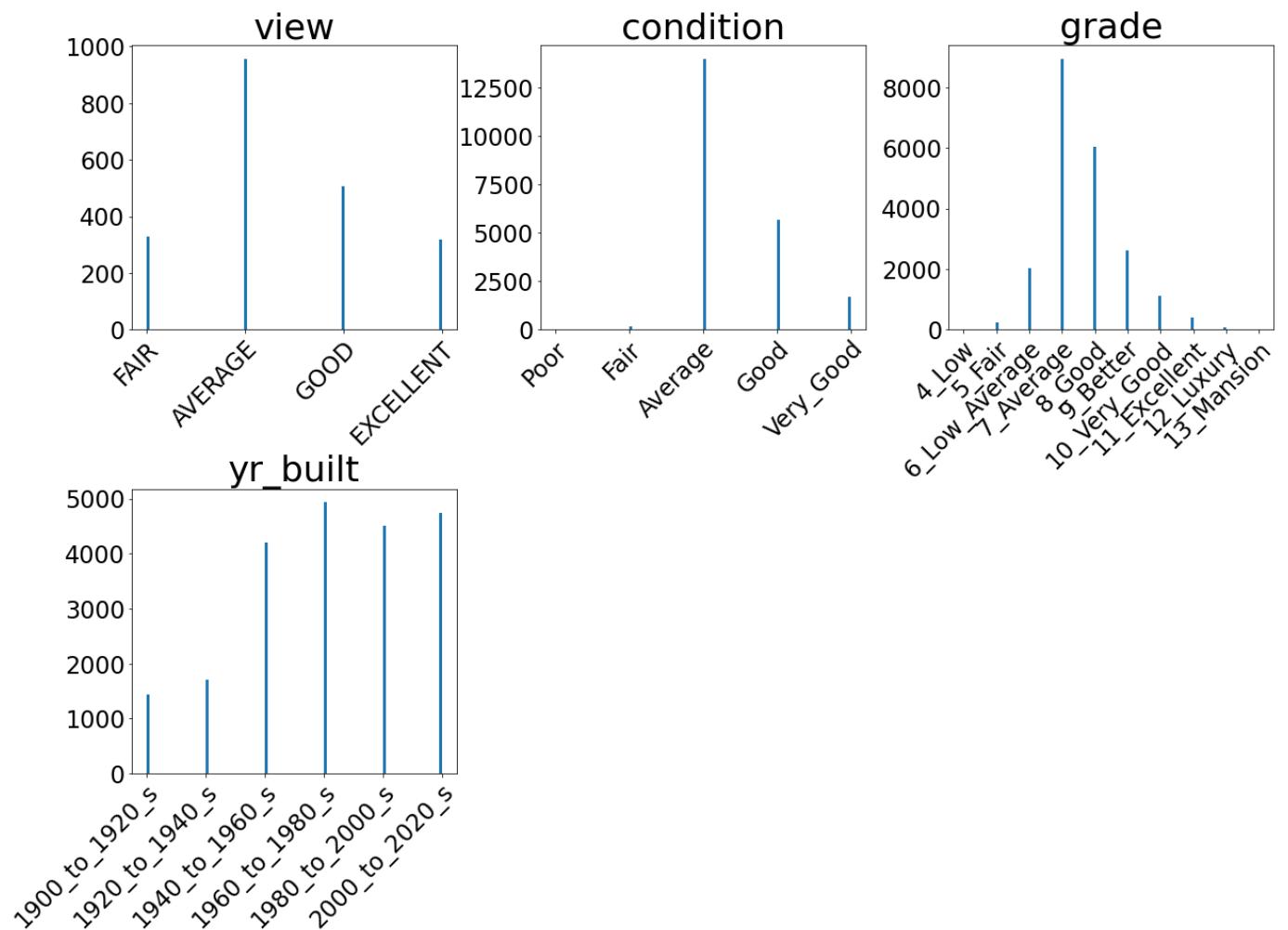
In [45]:

```

1 cats = len(kc_cats.columns)
2 n_rows = int(cats / 2)
3
4 fig, axs = plt.subplots(nrows=n_rows, ncols=3, figsize=(18, n_rows * 6), )
5 fig2, city_ax = plt.subplots(figsize=(18, 6))
6 axs = axs.flatten()
7
8 only_2_vals_df = pd.DataFrame()
9 empty_i = 0
10 city_i = 0
11 for c_i, cat in enumerate(kc_cats.columns):
12
13     ax_i = c_i if empty_i==0 else c_i - empty_i if city_i==0 else c_i - empty_i - city_i
14
15     if kc_cats[cat].nunique() > 2:
16         plot_col = kc_cats[cat].copy()
17
18     if cat != 'city':
19         cat_dict = all_dicts[cat].copy()
20
21     if cat == 'view':
22         plot_col = plot_col.where(plot_col != 'NONE').dropna()
23         del cat_dict['NONE']
24
25     col_order = sorted(cat_dict.keys(), key=lambda k: cat_dict[k])
26     plot_col = \
27         plot_col.sort_values(key=lambda c_col: c_col.map(lambda x: col_order.index(x)))
28     axs[ax_i].hist(plot_col, bins=100)
29
30     axs[ax_i].tick_params('both', labelsize=15)
31     plt.setp(axs[ax_i].get_xticklabels(), rotation=45, rotation_mode='anchor', ha='right', va='top')
32
33     axs[ax_i].set_title(cat, size=36)
34
35 else:
36     city_i += 1
37     c_step = 2000
38
39     plot_col = plot_col.sort_values().map(lambda x: ' '.join(x.split('_')) if '_' in x else x)
40
41     plot_col = plot_col.value_counts().sort_index()
42
43     city_ax.bar(x=plot_col.index, height=plot_col.values, width=.36, color=get_lighter_color(.12,
44             zorder=9)
45
46     city_ax.grid(True, axis='y', lw=1.2, alpha=.81)
47     [city_ax.spines[side].set_visible(False) for side in ['top', 'right']]
48     city_ax.tick_params('both', labelsize=18, labelcolor=(.12, .39, .12))
49     c_ticks = np.arange(0, plot_col.max() + c_step, c_step)
50     def c_y_ticks(y, pos):
51         return '$\\mathbf{' + '{:,}'.format(y) + '}$'
52     city_ax.set_yticks(c_ticks)
53     city_ax.yaxis.set_major_formatter(c_y_ticks)
54     plt.setp(city_ax.get_yticklabels(), weight='bold')
55     plt.setp(city_ax.get_xticklabels(), rotation=45, rotation_mode='anchor', ha='right', va='top',
56             weight='bold')
57
58     city_ax.set_title(cat.title(), size=36, pad=18, weight='bold', family='serif', color=(.12, .39,
59
60     if cat=='yr_built': axs[ax_i].tick_params('both', labelsize=21)
61
62 else:
63     empty_i += 1
64     only_2_vals_df = pd.concat([only_2_vals_df, kc_cats[cat].value_counts()], axis=1)
65
66 fig.tight_layout(h_pad=1.5, w_pad=3)
67 fig.set_facecolor('w')
68 fig2.set_facecolor('w')
69
70 [ax.set_visible(False) for ax in axs if not ax.has_data()]
71 [ax.tick_params('both', labelsize=24) for ax in [fig.axes[0]] + fig.axes[1:5] + fig.axes[6:7]]
72
73 fig.savefig('visuals/categorical_features_distributions_2.png', bbox_inches='tight')
74 fig2.savefig('visuals/city_distribution.png', bbox_inches='tight')
75

```

```
76 plt.show()  
77  
78 display(only_2_vals_df)
```



	waterfront	basement	renovated	waterfront_Missing	yr_renovated_Missing	sqft_basement_Missing
0	21388	12797	20790	19163	17703	21081
1	145	8736	743	2370	3830	452

## Initial Correlation Examination

---

The basic point of building a regression model is to find what impact an increase in one predictor, or **independent** variable, has on the target, or **dependent** variable, holding everything else constant. Eliminating predictors that are already highly correlated with each other attempts to achieve that goal as closely as possible. The predictors that are eliminated in this section could not be considered independent and would lead to unreliable statistical measurements.

```
In [46]: 1 kc_corr_test = pd.concat([kc_target, kc_nums, kc_cats], join='inner', axis=1)
```

```
In [47]: 1 kc_corr_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21252 entries, 2495 to 15937
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   price            21252 non-null   float64
 1   sqft_living      21252 non-null   int64  
 2   sqft_lot         21252 non-null   int64  
 3   sqft_above       21252 non-null   int64  
 4   sqft_living15    21252 non-null   int64  
 5   sqft_lot15       21252 non-null   int64  
 6   floors           21252 non-null   float64
 7   bedrooms         21252 non-null   int64  
 8   bathrooms        21252 non-null   float64
 9   waterfront        21252 non-null   int64  
 10  view             21252 non-null   object 
 11  condition        21252 non-null   object 
 12  grade            21252 non-null   object 
 13  city             21252 non-null   object 
 14  basement         21252 non-null   int32  
 15  yr_built         21252 non-null   object 
 16  renovated         21252 non-null   int32  
 17  waterfront_Missing  21252 non-null   int32  
 18  yr_renovated_Missing 21252 non-null   int32  
 19  sqft_basement_Missing 21252 non-null   int32  
dtypes: float64(3), int32(5), int64(7), object(5)
memory usage: 3.0+ MB
```

```
In [48]: 1 restruc_cols = list(kc_corr_test.columns[10:14]) + list(kc_corr_test.columns[15:16])
 2
 3 for col in restruc_cols:
 4     if col != 'city':
 5         col_dict = all_dicts[col]
 6         kc_corr_test[col] = kc_corr_test[col].map(lambda x: col_dict[x])
 7     else:
 8         col_dict = {}
 9         for c_i, city in enumerate(kc_corr_test[col].unique()): col_dict[city] = c_i
10         kc_corr_test[col] = kc_corr_test[col].map(lambda x: col_dict[x])
```

```

In [49]: 1 corr_df = kc_corr_test.corr()
2
3 # Variables' Correlations with the Target
4 #####
5 price_corr = corr_df[['price']].copy()
6
7 price_corr.sort_values('price', ascending=False, inplace=True)
8
9 most_correlated_feature = price_corr.index[1]
10 most_correlated_value = price_corr.iloc[1][0]
11
12 print('\033[31m\033[4m\033[1mFeature Most Correlated with Price\033[0m:\t'+ \
13     most_correlated_feature)
14 print("\033[31m\033[4m\033[1mMost Correlated Feature's Value\033[0m:\t\t",
15     round(most_correlated_value, 4))
16
17 # Variables' Correlation with Each Other
18 #####
19 corr_df = corr_df.abs().stack().reset_index().sort_values(0, ascending=False)
20 corr_df['pairs'] = list(zip(corr_df.level_0, corr_df.level_1))
21
22 corr_df.set_index(['pairs'], inplace=True)
23 corr_df.drop(['level_0', 'level_1'], axis=1, inplace=True)
24 corr_df.columns = ['correlation']
25
26 corr_df = corr_df[(corr_df.correlation > .8) & (corr_df.correlation < 1)]
27
28 display(corr_df)

```

Feature Most Correlated with Price: sqft\_living  
Most Correlated Feature's Value: 0.681

correlation	
pairs	
(sqft_living, sqft_above)	0.865301
(sqft_above, sqft_living)	0.865301

## Removing Highly Correlated Predictors

Based on my initial correlation examination, I decided to drop the `sqft_above` column from the `kc_nums` dataframe.

```

In [50]: 1 kc_nums.drop('sqft_above', axis=1, inplace=True)

```

## Dummy Variable Creation

I used the `OneHotEncoder()` on the appropriate columns to create a properly encoded version of `kc_cats`. I also created a dictionary with the original categorical columns as the keys and the dummy columns that were created from each of them as the values for each key. Finally, I rearranged the columns of `kc_cats` in a way that I found more logical.

```
In [51]: 1 orig_cat_cols = list(kc_cats.columns)
2 [print(c_i, col) for c_i, col in enumerate(orig_cat_cols)];
```

```
0 waterfront
1 view
2 condition
3 grade
4 city
5 basement
6 yr_built
7 renovated
8 waterfront_Missing
9 yr_renovated_Missing
10 sqft_basement_Missing
```

```
In [52]: 1 missing_cols = orig_cat_cols[-3:]
2 orig_cat_cols = orig_cat_cols[:-3]
3
4 missing_cols
```

```
Out[52]: ['waterfront_Missing', 'yr_renovated_Missing', 'sqft_basement_Missing']
```

```
In [53]: 1 num_cols = list(kc_nums.columns)
2
3 print(num_cols)
```

```
['sqft_living', 'sqft_lot', 'sqft_living15', 'sqft_lot15', 'floors', 'bedrooms', 'bathrooms']
```

```
In [54]: 1 print(orig_cat_cols)
```

```
['waterfront', 'view', 'condition', 'grade', 'city', 'basement', 'yr_built', 'renovated']
```

```
In [55]: 1 new_cat_order = ['basement', 'grade', 'view', 'waterfront', 'condition', 'renovated', 'yr_built', 'city']
2
3 assert len(new_cat_order)==len(orig_cat_cols)
```

```
In [56]: 1 kc_coded_cats = pd.DataFrame()
2 cat_dummy_dict = {}
3 for n_cat in num_cols:
4     cat_dummy_dict[n_cat] = n_cat
5
6 for c_i, cat in enumerate(new_cat_order):
7     if kc_cats[cat].nunique() <= 2:
8         cat_df = kc_cats[cat]
9         cat_dummy_dict[cat] = cat
10    else:
11        cat_col = kc_cats[[cat]].applymap(lambda x: cat + '_' + x)
12
13    if cat != 'city':
14        cat_order = [cat + '_' + sub_cat for sub_cat in list(all_dicts[cat].keys())]
15
16        ohe_encoder = OneHotEncoder(categories=[cat_order], drop='first', sparse=False)
17        ohe_encoder.fit(cat_col)
18        coded_cat = ohe_encoder.transform(cat_col)
19
20        cat_df = pd.DataFrame(coded_cat, columns=list(ohe_encoder.categories_[0])[1:], index=kc_cats.index)
21
22        cat_dummy_dict[cat] = list(ohe_encoder.categories_[0])[1:]
23
24    if cat == 'city':
25        ohe_encoder = OneHotEncoder(categories='auto', drop='first', sparse=False)
26        ohe_encoder.fit(cat_col)
27        coded_cat = ohe_encoder.transform(cat_col)
28
29        cat_df = pd.DataFrame(coded_cat, columns=list(ohe_encoder.categories_[0])[1:], index=kc_cats.index)
30
31        cat_dummy_dict[cat] = list(ohe_encoder.categories_[0])[1:]
32
33 kc_coded_cats = pd.concat([kc_coded_cats, cat_df], axis=1)
```

```
In [57]: 1 cat_dummy_dict['Missing'] = missing_cols
```

```
In [58]: 1 # [print(cat, '\n', dum_cols, '\n') for cat, dum_cols in cat_dummy_dict.items();]
```

## Preprocessed DataFrames

---

The last visualization that I created shows that there were significantly more entries within the city of Seattle compared to the rest of the cities. To explore the difference between the sales of residential properties inside Seattle vs. outside Seattle, which predictors were important for each, as well as the difference between the coefficients of the important predictors shared by both, I created a separate preprocessed dataframe for each. Of course, I also created a preprocessed dataframe for all of King County. Beyond the intrinsic value of building a model for the entire county, it also had the added benefit of serving as a useful comparison to the results of the separated dataframes.

For each of the preprocessed dataframes, I first removed the dummy columns created from the `city` column. I did this because I created the entirely separate models to analyze the difference between the prices of sales inside Seattle with those outside Seattle and including them at this point would've just added unnecessary noise to my models. I then split the preprocessed dataframes into their `x` and `y` components and performed one final check to make sure there were no categorical variables present with low frequencies.

```
In [59]: 1 seattle_entries = kc_coded_cats.loc[kc_coded_cats.city_Seattle == 1].shape[0]
2 out_seattle_entries = kc_coded_cats.loc[kc_coded_cats.city_Seattle == 0].shape[0]
```

```
In [60]: 1 print(bold_red + 'Number of Entries Inside Seattle' + every_off +':\t'+ str(seattle_entries))
2 print(bold_red + 'Number of Entries Outside Seattle' + every_off +':\t'+ str(out_seattle_entries))
```

```
Number of Entries Inside Seattle:      8946
Number of Entries Outside Seattle:    12587
```

```
In [61]: 1 kc_prepoc = pd.concat([kc_target, kc_nums, kc_coded_cats], join='inner', axis=1)
```

```
In [62]: 1 kc_prepoc = kc_prepoc.loc[:, list(kc_prepoc.columns.map(lambda x: 'city' not in x))]
```

```
In [63]: 1 not any(kc_prepoc.columns.map(lambda x: 'city' in x))
```

```
Out[63]: True
```

```
In [64]: 1 kc_X = kc_prepoc.drop('price', axis=1)
2 kc_y = kc_prepoc['price']
```

```
In [65]: 1 kc_X_sums = kc_X[[col for col in kc_prepoc.columns if col in kc_coded_cats.columns]].sum()
2 kc_X_sums = pd.DataFrame(kc_X_sums.values, index=kc_X_sums.index)
```

```
In [66]: 1 kc_less_10_entries = kc_X_sums.loc[kc_X_sums[0] < 10].sort_values(0)
2
3 kc_X.drop(list(kc_less_10_entries.index), axis=1, inplace=True)
```

```
In [67]: 1 assert kc_X.shape[0]==kc_y.shape[0]
```

## Seattle Preprocessed DataFrame

```
In [68]: 1 kc_coded_cats_seattle = kc_coded_cats.loc[kc_coded_cats.city_Seattle == 1].copy()
```

```
In [69]: 1 kc_coded_cats_seattle =
2 kc_coded_cats_seattle.loc[:, list(kc_coded_cats_seattle.columns.map(lambda x: 'city' not in x))]
```

```
In [70]: 1 not any(kc_coded_cats_seattle.columns.map(lambda x: 'city' in x))
```

```
Out[70]: True
```

```
In [71]: 1 kc_prepoc_seattle = pd.concat([kc_target, kc_nums, kc_coded_cats_seattle], join='inner', axis=1)
```

```
In [72]: 1 kc_X_seattle = kc_prepoc_seattle.drop('price', axis=1)
2 kc_y_seattle = kc_prepoc_seattle['price']
```

```
In [73]: 1 seattle_X_sums = kc_X_seattle[kc_coded_cats_seattle.columns].sum()
2 seattle_X_sums = pd.DataFrame(seattle_X_sums.values, index=seattle_X_sums.index)
```

```
In [74]: 1 seattle_less_10_entries = seattle_X_sums.loc[seattle_X_sums[0] < 10].sort_values(0)
2
3 kc_X_seattle.drop(list(seattle_less_10_entries.index), axis=1, inplace=True)
```

```
In [75]: 1 assert kc_X_seattle.shape[0]==kc_y_seattle.shape[0]
```

## Outside Seattle Preprocessed DataFrame

```
In [76]: 1 kc_coded_cats_out_seattle = kc_coded_cats.loc[kc_coded_cats.city_Seattle == 0].copy()
```

```
In [77]: 1 kc_coded_cats_out_seattle = \
2 kc_coded_cats_out_seattle.loc[:, list(kc_coded_cats_out_seattle.columns.map(lambda x: 'city' not in x))]
```

```
In [78]: 1 not any(kc_coded_cats_seattle.columns.map(lambda x: 'city' in x))
```

Out[78]: True

```
In [79]: 1 kc_preproc_out_seattle = \
2 pd.concat([kc_target, kc_nums, kc_coded_cats_out_seattle], join='inner', axis=1)
```

```
In [80]: 1 kc_X_out_seattle = kc_preproc_out_seattle.drop('price', axis=1)
2 kc_y_out_seattle = kc_preproc_out_seattle['price']
```

```
In [81]: 1 out_seattle_X_sums = kc_X_out_seattle[kc_coded_cats_out_seattle.columns].sum()
2 out_seattle_X_sums = pd.DataFrame(out_seattle_X_sums.values, index=out_seattle_X_sums.index)
```

```
In [82]: 1 out_seattle_less_10_entries = out_seattle_X_sums.loc[out_seattle_X_sums[0] < 10].sort_values(0)
2
3 kc_X_out_seattle.drop(list(out_seattle_less_10_entries.index), axis=1, inplace=True)
```

```
In [83]: 1 assert kc_X_out_seattle.shape[0]==kc_y_out_seattle.shape[0]
```

## Base Models

---

The base models for each preprocessed dataframe are just the relationship between the most correlated feature, `sqft_living`, and the target variable, `price`, which will be used to judge the performance of future models.

### All King County Base Model

---

```
In [84]: 1 kc_base_const = sm.add_constant(kc_X[most_correlated_feature])
2 kc_base_model = sm.OLS(endog=kc_y, exog=kc_base_const).fit()
```

### Seattle Base Model

---

```
In [85]: 1 base_const_seattle = sm.add_constant(kc_X_seattle[most_correlated_feature])
2 seattle_base_model = sm.OLS(endog=kc_y_seattle, exog=base_const_seattle).fit()
```

### Outside Seattle Base Model

---

```
In [86]: 1 base_const_out_seattle = sm.add_constant(kc_X_out_seattle[most_correlated_feature])
2 out_seattle_base_model = sm.OLS(endog=kc_y_out_seattle, exog=base_const_out_seattle).fit()
```

## Full Models

---

After getting a baseline for each preprocessed dataframe, I created models for each with all the predictors available. I then began eliminating them to create a final model with only the best predictors. First, I removed predictors based on their Variance Inflation Factor (VIF) scores to eliminate predictors with high levels of multicollinearity missed by my initial correlation examination. I then used the RFEcv feature selection method to eliminate all but the best predictors. Finally, I eliminated all predictors with pvalues less than the standard confidence level of 0.05 , to only include the predictors of the highest statistical significance.

Once the predictors for each model were finalized, I could investigate the Linearity, Normality, and Homoscedasticity of the predicted values generated by each model. For each of the models, a log transformation was required for the model to meet the assumptions required when building multiple linear regression models. I then had a final equation that I could analyze to produce my [Insights and Conclusions](#). I transformed the coefficients in the equation to make understanding them easier.

I also created simple dataframes with the coefficients, the  $r^2$ , and the adjusted  $r^2$  scores from each model to make any comparisons between the models easier.

## All King County Full Model

---

```
In [87]: 1 kc_X_full = kc_X.copy()
```

```
In [88]: 1 kc_full_const = sm.add_constant(kc_X_full)
2 kc_full_model = sm.OLS(endog=kc_y, exog=kc_full_const).fit()
```

## kc\_VIF

---

In [89]:

```
1 vif_compl = 0
2 while not vif_compl:
3     high_cols_dict = {}
4     mid_cols_dict = {}
5
6     full_const = sm.add_constant(kc_X_full)
7     kc_full_VIF_model = sm.OLS(endog=kc_y, exog=full_const).fit()
8
9     vif = [variance_inflation_factor(full_const.values, i) for i in range(full_const.shape[1])]
10    vif_df = pd.DataFrame(vif, index=full_const.columns, columns=["Variance Inflation Factor"])
11
12    high_vif_cols = list(vif_df.loc[vif_df['Variance Inflation Factor'] > 10].index)
13    high_vif_vals = list(vif_df.loc[vif_df['Variance Inflation Factor'] > 10].values)
14    if 'const' in high_vif_cols:
15        h_i = high_vif_cols.index('const')
16        high_vif_cols.remove('const')
17        high_vif_vals.pop(h_i)
18
19    mid_vif_cols = \
20    list(vif_df.loc[(vif_df['Variance Inflation Factor'] > 5) & (vif_df['Variance Inflation Factor'] < 10)])
21
22    mid_vif_vals = \
23    list(vif_df.loc[(vif_df['Variance Inflation Factor'] > 5) & (vif_df['Variance Inflation Factor'] < 10)])
24
25    if 'const' in mid_vif_cols:
26        m_i = mid_vif_cols.index('const')
27        mid_vif_cols.remove('const')
28        mid_vif_vals.pop(m_i)
29
30    for g_i, col_grp in enumerate([high_vif_cols, mid_vif_cols]):
31        if g_i==0: grp_dict, grp_vals = high_cols_dict, high_vif_vals
32        if g_i==1: grp_dict, grp_vals = mid_cols_dict, mid_vif_vals
33        if col_grp:
34            for c_i, col in enumerate(col_grp):
35                if col in grp_dict.keys(): grp_dict[col] += grp_vals[c_i]
36                else: grp_dict[col] = grp_vals[c_i]
37
38    high_cols_df = pd.DataFrame(high_cols_dict.values(), high_cols_dict.keys(), columns=['VIF'])
39    mid_VIF_cols_df = pd.DataFrame(mid_cols_dict.values(), mid_cols_dict.keys(), columns=['VIF'])
40
41    kc_VIF = high_cols_df.sort_values('VIF', ascending=False)
42    mid_VIF_cols_df = mid_VIF_cols_df.sort_values('VIF', ascending=False)
43
44    VIF_drop_cols = []
45    for o_col in list(cat_dummy_dict.keys())[:-1]:
46        if any(kc_VIF.index.map(lambda x: o_col in x)):
47            first_pred_col = kc_VIF.loc[kc_VIF.index.map(lambda x: o_col in x)].index[0]
48            VIF_drop_cols.append(first_pred_col)
49
50    if VIF_drop_cols:
51        kc_X_full.drop(VIF_drop_cols, axis=1, inplace=True)
52        print(bold_red + 'High VIF Dropped Columns'+ every_off, '\n', VIF_drop_cols, '\n')
53
54    if not VIF_drop_cols and not mid_VIF_cols_df.empty:
55        mid_VIF_drop_cols = []
56        for o_col in list(cat_dummy_dict.keys())[:-1]:
57            if any(mid_VIF_cols_df.index.map(lambda x: o_col in x)):
58                first_pred_col = mid_VIF_cols_df.loc[mid_VIF_cols_df.index.map(lambda x: o_col in x)].index[0]
59                mid_VIF_drop_cols.append(first_pred_col)
60
61    if mid_VIF_drop_cols:
62        kc_X_full.drop(mid_VIF_drop_cols, axis=1, inplace=True)
63        print(bold_red + 'Mid VIF Dropped Columns'+ every_off, '\n', mid_VIF_drop_cols, '\n')
64
65    if not VIF_drop_cols and mid_VIF_cols_df.empty: vif_compl = 1
66
67 print(bold_red +'VIF Elimination Finished'+ every_off)
```

High VIF Dropped Columns

['grade\_7\_Average', 'condition\_Average']

Mid VIF Dropped Columns

['sqft\_living']

VIF Elimination Finished

## kc\_RFECV

---

In [90]:

```
1 best_model_found = 0
2 RFECV_round = 0
3 while not best_model_found:
4
5     full_const = sm.add_constant(kc_X_full)
6     kc_full_RFECV_model = sm.OLS(endog=kc_y, exog=full_const).fit()
7
8     splitter = ShuffleSplit(n_splits=6, test_size=0.2, random_state=36)
9     kc_X_for_RFECV = StandardScaler().fit_transform(kc_X_full)
10    model_for_RFECV = LinearRegression()
11    selector = RFECV(model_for_RFECV, cv=splitter)
12    selector.fit(kc_X_for_RFECV, kc_y)
13
14    RFECV_bad_cols = [col for c_i, col in enumerate(kc_X_full.columns) if not selector.support_[c_i]]
15
16    if RFECV_bad_cols:
17        RFECV_drop_cols = []
18        for o_col in list(cat_dummy_dict.keys()):
19            o_df = pd.DataFrame()
20            for r_col in RFECV_bad_cols:
21                if r_col.startswith(o_col) or r_col.endswith(o_col):
22                    r_prob, r_coef = kc_full_RFECV_model.pvalues[r_col], kc_full_RFECV_model.params[r_col]
23                    r_df = pd.DataFrame([r_prob, r_coef], columns=[r_col], index=['p_value', 'coeff'])
24                    o_df = pd.concat([o_df, r_df])
25
26                if not o_df.empty:
27                    o_max_col = o_df.loc[o_df.p_value == o_df.p_value.max()].index[0]
28                    RFECV_drop_cols.append(o_max_col)
29
30    RFECV_round += 1
31
32    kc_X_full.drop(RFECV_drop_cols, axis=1, inplace=True)
33    print(bold_red + 'RFECV Dropped Columns' + every_off, '\n', RFECV_drop_cols, '\n')
34
35    else: best_model_found = 1
36
37 print(bold_red + 'RFECV Finished' + every_off)
```

RFECV Dropped Columns  
['bedrooms', 'condition\_Fair']

RFECV Finished

## kc\_pvals

---

In [91]:

```
1 high_pvals = 1
2
3 while high_pvals:
4
5     kc_pval_const = sm.add_constant(kc_X_full)
6     kc_pval_model = sm.OLS(endog=kc_y, exog=kc_pval_const).fit()
7
8     high_pval_list = list(kc_pval_model.pvalues.loc[kc_pval_model.pvalues >= .05].sort_values(ascending=False))
9
10    if high_pval_list:
11        high_pval_drop_col = high_pval_list[0]
12        kc_X_full.drop(high_pval_drop_col, axis=1, inplace=True)
13        print(str(high_pvals) + ' - ' + bold_red + 'High p-value Column Dropped' + every_off + ': \n', high_pval_list)
14        high_pvals -= 1
15
16    else: high_pvals = 0
17
18 print(bold_red + 'High p_value Elimination Finsihed' + every_off)
```

High p\_value Elimination Finsihed

In

```
[92]: 1 kc_fin_const = kc_pval_const  
2 kc_fin_model = kc_pval_model
```

## Investigating the Linearity, Normality and Homoscedasticity of kc\_fin\_model

---

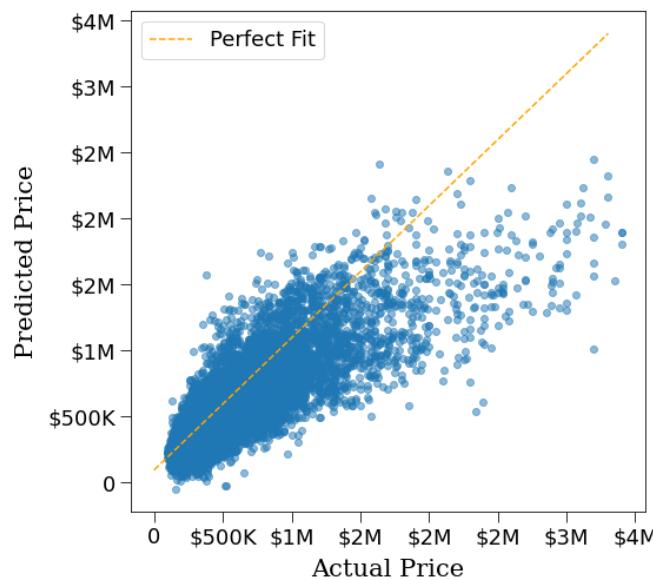
In [93]:

```
1 plt.rc('figure', figsize=(18, 18), facecolor='w')  
2 plt.rc('axes', titlesize=24, titlepad=18, titleweight='bold', labelsize=21, labelpad=9)  
3 plt.rc('xtick', labelsize=18)  
4 plt.rc('xtick.major', size=9)  
5 plt.rc('ytick', labelsize=18)  
6 plt.rc('ytick.major', size=9)  
7 plt.rc('legend', fontsize=18)
```

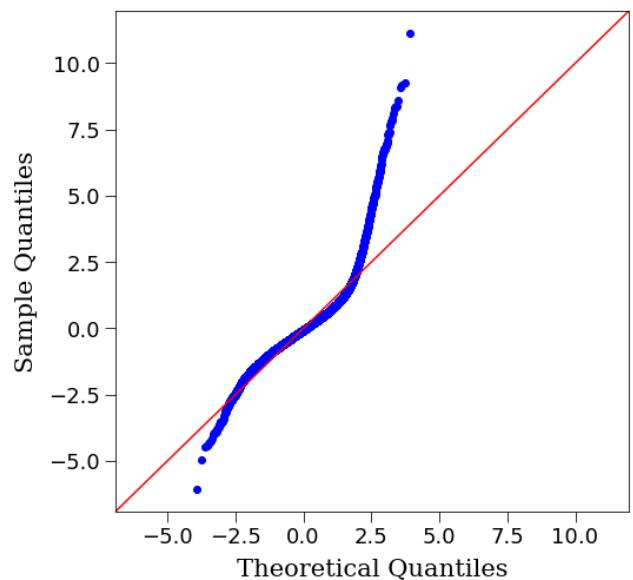
In [94]:

```
1 fig = plt.figure()
2 gs = fig.add_gridspec(2, 2, wspace=.3, hspace=.3)
3
4 lin_ax = fig.add_subplot(gs[0, 0])
5 norm_ax = fig.add_subplot(gs[0, 1])
6 homo_ax = fig.add_subplot(gs[1, :])
7
8 kc_preds = kc_fin_model.predict(kc_fin_const)
9 perfect_line = np.arange(kc_y.min(), kc_y.max())
10 kc_resids = (kc_y - kc_preds)
11
12 lin_ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
13 lin_ax.scatter(kc_y, kc_preds, alpha=0.5)
14 lin_ax.set_xlabel("Actual Price", family='serif')
15 lin_ax.set_ylabel("Predicted Price", family='serif')
16 lin_ax.set_title('Linearity Check', family='serif')
17 lin_ax.legend()
18
19 sm.graphics.qqplot(kc_resids, dist=stats.norm, line='45', fit=True, ax=norm_ax)
20 norm_ax.set_title('Normality Check', family='serif')
21
22 homo_ax.scatter(kc_preds, kc_resids, alpha=0.5)
23 homo_ax.plot(kc_preds, [0 for i in range(len(kc_X_full))])
24 homo_ax.set_xlabel("Predicted Price", family='serif')
25 homo_ax.set_ylabel("Actual Price - Predicted Price", family='serif')
26 homo_ax.set_title('Homoscedasticity Check', family='serif')
27
28 for a_i, ax in enumerate([lin_ax, norm_ax, homo_ax]):
29     if a_i!=1:
30         ax.xaxis.set_major_formatter(viz_currency_formatter)
31         ax.yaxis.set_major_formatter(viz_currency_formatter)
32     else:
33         ax.set_xlabel(ax.get_xlabel(), family='serif')
34         ax.set_ylabel(ax.get_ylabel(), family='serif')
35
36 fig.savefig('visuals/kc_model_check.png' , bbox_inches='tight')
37
38 plt.show()
```

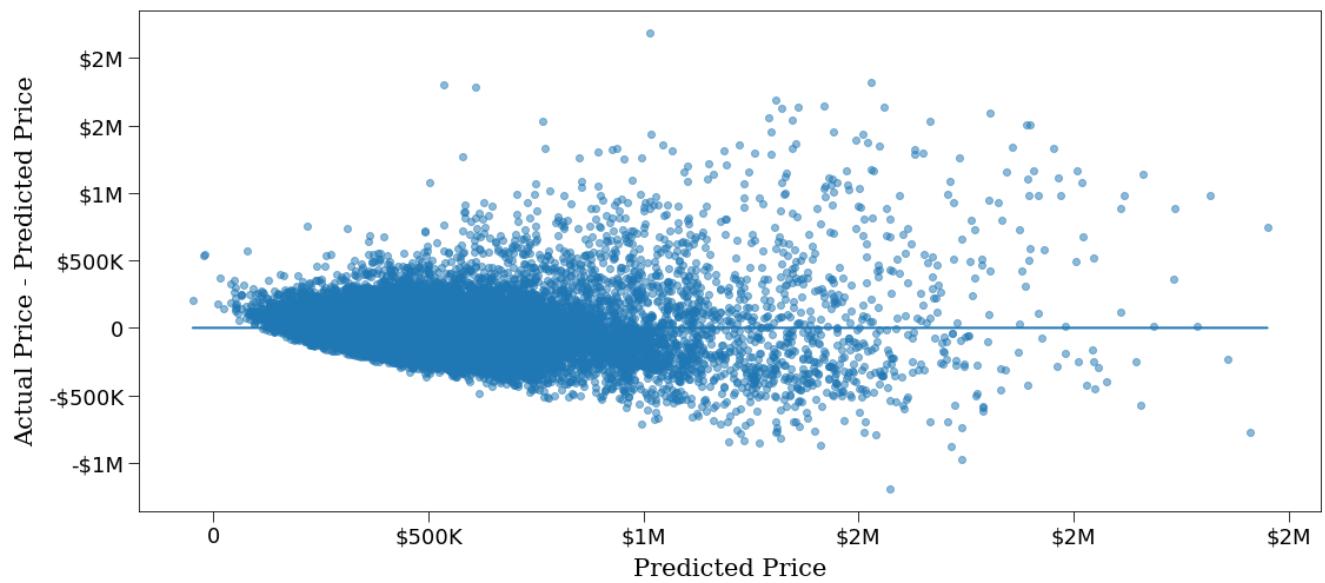
## Linearity Check



## Normality Check



## Homoscedasticity Check



## kc\_log\_model

```
In [95]: 1 kc_log_y = np.log(kc_y)
```

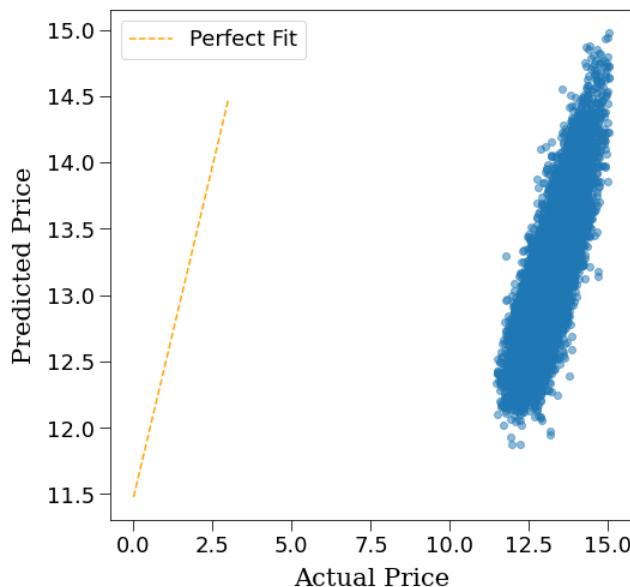
```
In [96]: 1 kc_log_const = sm.add_constant(kc_X_full)
2 kc_log_model = sm.OLS(endog=kc_log_y, exog=kc_log_const).fit()
```

Investigating the Linearity, Normality and Homoscedasticity of `kc_log_model`

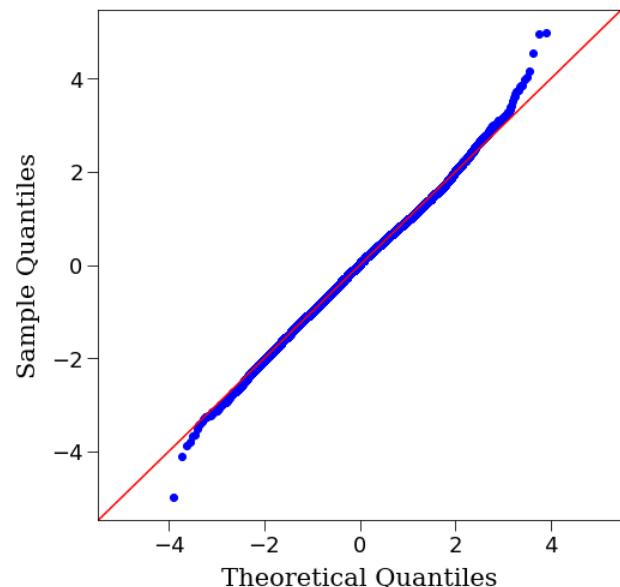
In [97]:

```
1 fig = plt.figure()
2 gs = fig.add_gridspec(2, 2, wspace=.3, hspace=.3)
3
4 lin_ax = fig.add_subplot(gs[0, 0])
5 norm_ax = fig.add_subplot(gs[0, 1])
6 homo_ax = fig.add_subplot(gs[1, :])
7
8 kc_preds = kc_log_model.predict(kc_log_const)
9 perfect_line = np.arange(kc_log_y.min(), kc_log_y.max())
10 kc_resids = (kc_log_y - kc_preds)
11
12 lin_ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
13 lin_ax.scatter(kc_log_y, kc_preds, alpha=0.5)
14 lin_ax.set_xlabel("Actual Price", family='serif')
15 lin_ax.set_ylabel("Predicted Price", family='serif')
16 lin_ax.set_title('Linearity Check', family='serif')
17 lin_ax.legend()
18
19 sm.graphics.qqplot(kc_resids, dist=stats.norm, line='45', fit=True, ax=norm_ax)
20 norm_ax.set_title('Normality Check', family='serif')
21 norm_ax.set_xlabel(norm_ax.get_xlabel(), family='serif')
22 norm_ax.set_ylabel(norm_ax.get_ylabel(), family='serif')
23
24 homo_ax.scatter(kc_preds, kc_resids, alpha=0.5)
25 homo_ax.plot(kc_preds, [0 for i in range(len(kc_X_full))])
26 homo_ax.set_xlabel("Predicted Price", family='serif')
27 homo_ax.set_ylabel("Actual Price - Predicted Price", family='serif')
28 homo_ax.set_title('Homoscedasticity Check', family='serif')
29
30 fig.savefig('visuals/kc_model_check_log.png' , bbox_inches='tight')
31
32 plt.show()
```

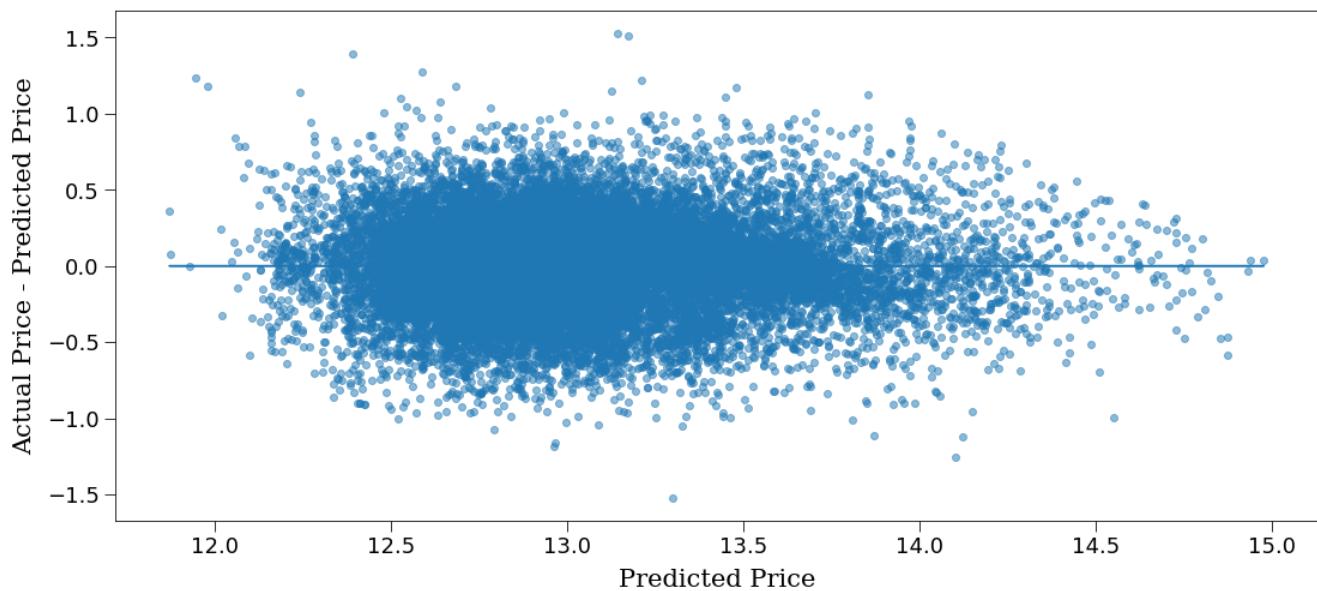
## Linearity Check



## Normality Check



## Homoscedasticity Check



## Comparing kc Models

```
In [98]: 1 kc_model_list = [kc_base_model, kc_full_model, kc_full_VIF_model, kc_full_RFECV_model, kc_fin_model, kc_log
2 kc_model_names = ['base', 'full', 'full_VIF', 'full_RFECV', 'fin', 'log']
```

```
In [99]: 1 kc_r_vals = {}
2 kc_r_adj_vals = {}
3 kc_compare_params_df = pd.DataFrame()
4 for ols_i, (ols_model, model_name) in enumerate(zip(kc_model_list, kc_model_names)):
5     model_params = ols_model.params
6     model_params.name = model_name
7     kc_r_vals[model_name] = ols_model.rsquared
8     kc_r_adj_vals[model_name] = ols_model.rsquared_adj
9
10    kc_compare_params_df = pd.concat([kc_compare_params_df, model_params], axis=1)
11
12 for r_i, r_dict in enumerate([kc_r_vals, kc_r_adj_vals]):
13     r_name = 'r_score' if r_i==0 else 'r_adj_score'
14     r_ser = pd.DataFrame(r_dict.values(), index=r_dict.keys(), columns=[r_name]).transpose()
15
16    kc_compare_params_df = pd.concat([kc_compare_params_df, r_ser])
```

In [100]:

```

1 for col in kc_compare_params_df.columns[:-1]:
2     kc_compare_params_df[col] = kc_compare_params_df[col].round(2)
3
4 display(kc_compare_params_df)

```

		base	full	full_VIF	full_RFECV	fin	log
	const	-7280.35	1851260.75	181853.63	184086.86	184086.86	1.247509e+01
	sqft_living	261.48	101.41	NaN	NaN	NaN	NaN
	sqft_lot	NaN	0.00	0.16	0.16	0.16	5.308354e-07
	sqft_living15	NaN	43.95	84.43	85.00	85.00	1.677819e-04
	sqft_lot15	NaN	-0.41	-0.34	-0.34	-0.34	-3.228181e-07
	floors	NaN	32242.39	38391.34	38596.57	38596.57	7.979150e-02
	bedrooms	NaN	-20563.99	1876.15	NaN	NaN	NaN
	bathrooms	NaN	43649.55	77424.92	78637.52	78637.52	1.102396e-01
	basement	NaN	28271.09	46838.03	46921.40	46921.40	1.023296e-01
	grade_4_Low	NaN	-1845862.71	-202612.19	-205199.69	-205199.69	-7.329981e-01
	grade_5_Fair	NaN	-1806822.56	-156429.53	-158160.98	-158160.98	-4.922216e-01
	grade_6_Low_Average	NaN	-1751618.09	-91864.21	-92576.01	-92576.01	-2.792341e-01
	grade_7_Average	NaN	-1666481.28	NaN	NaN	NaN	NaN
	grade_8_Good	NaN	-1581692.34	96157.10	96053.56	96053.56	2.231718e-01
	grade_9_Better	NaN	-1440300.37	267735.68	267709.22	267709.22	4.668456e-01
	grade_10_Very_Good	NaN	-1248361.03	491630.80	491261.71	491261.71	6.667515e-01
	grade_11_Excellent	NaN	-979479.94	796781.64	796239.55	796239.55	8.406650e-01
	grade_12_Luxury	NaN	-732027.42	1073845.32	1072828.69	1072828.69	9.651193e-01
	view_FAIR	NaN	108804.10	112440.75	112164.38	112164.38	1.442479e-01
	view_AVERAGE	NaN	52721.43	57588.41	57406.37	57406.37	7.810800e-02
	view_GOOD	NaN	109097.91	112817.17	112393.29	112393.29	9.707906e-02
	view_EXCELLENT	NaN	277423.43	283667.93	283371.66	283371.66	2.578018e-01
	waterfront	NaN	391638.36	403652.08	402814.20	402814.20	3.096941e-01
	condition_Fair	NaN	18184.77	-26034.46	NaN	NaN	NaN
	condition_Average	NaN	41651.75	NaN	NaN	NaN	NaN
	condition_Good	NaN	70256.35	31452.89	32015.22	32015.22	5.174069e-02
	condition_Very_Good	NaN	109975.58	71782.15	72384.31	72384.31	1.135529e-01
	renovated	NaN	43209.95	50488.77	50507.01	50507.01	4.283010e-02
	yr_built_1920_to_1940_s	NaN	-21387.52	-22770.59	-22905.56	-22905.56	-5.179860e-02
	yr_built_1940_to_1960_s	NaN	-90311.26	-91173.55	-91062.34	-91062.34	-1.928097e-01
	yr_built_1960_to_1980_s	NaN	-189001.30	-197283.50	-196955.07	-196955.07	-3.973160e-01
	yr_built_1980_to_2000_s	NaN	-267220.17	-281701.49	-282060.51	-282060.51	-4.949009e-01
	yr_built_2000_to_2020_s	NaN	-244943.85	-256734.33	-257185.94	-257185.94	-4.560393e-01
	r_score	0.46	0.67	0.65	0.65	0.65	6.452154e-01
	r_adj_score	0.46	0.67	0.65	0.65	0.65	6.447640e-01

## kc\_log\_eq

In [101]:

```

1 kc_log_int = int(np.exp(kc_compare_params_df.log.dropna()[0]))
2 kc_log_int

```

Out[101]: 261735

```
In [102]: 1 kc_log_percs = kc_compare_params_df.log.dropna()[1:-2] * 100
2 kc_log_percs.astype(int)
```

```
Out[102]: sqft_lot          0
sqft_living15        0
sqft_lot15          0
floors              7
bathrooms           11
basement             10
grade_4_Low          -73
grade_5_Fair         -49
grade_6_Low_Average -27
grade_8_Good          22
grade_9_Better        46
grade_10_Very_Good   66
grade_11_Excellent    84
grade_12_Luxury       96
view_FAIR            14
view_AVERAGE          7
view_GOOD             9
view_EXCELLENT        25
waterfront            30
condition_Good        5
condition_Very_Good   11
renovated             4
yr_built_1920_to_1940_s -5
yr_built_1940_to_1960_s -19
yr_built_1960_to_1980_s -39
yr_built_1980_to_2000_s -49
yr_built_2000_to_2020_s -45
Name: log, dtype: int32
```

```
In [103]: 1 kc_fin_percs = list(kc_log_percs.values.astype(int))
2 kc_fin_preds = list(kc_log_percs.index)
```

```
In [104]: 1 kc_perc_eq = ['{}% * {}'.format(perc, pred) for perc, pred in zip(kc_fin_percs, kc_fin_preds)]
```

```
In [105]: 1 model_eq = 'price = '
2 model_int_string = str(kc_log_int) + ' +\n'
3 model_var_string = ' +\n'.join(kc_perc_eq)
```

In [106]:

```
1 kc_log_eq = model_eq + model_int_string + model_var_string  
2  
3 print(kc_log_eq)
```

```
price = 261735 +  
0% * sqft_lot +  
0% * sqft_living15 +  
0% * sqft_lot15 +  
7% * floors +  
11% * bathrooms +  
10% * basement +  
-73% * grade_4_Low +  
-49% * grade_5_Fair +  
-27% * grade_6_Low_Average +  
22% * grade_8_Good +  
46% * grade_9_Better +  
66% * grade_10_Very_Good +  
84% * grade_11_Excellent +  
96% * grade_12_Luxury +  
14% * view_FAIR +  
7% * view_AVERAGE +  
9% * view_GOOD +  
25% * view_EXCELLENT +  
30% * waterfront +  
5% * condition_Good +  
11% * condition_Very_Good +  
4% * renovated +  
-5% * yr_built_1920_to_1940_s +  
-19% * yr_built_1940_to_1960_s +  
-39% * yr_built_1960_to_1980_s +  
-49% * yr_built_1980_to_2000_s +  
-45% * yr_built_2000_to_2020_s
```

## Seattle Full Model

---

In [107]:

```
1 seattle_full_const = sm.add_constant(kc_X_seattle)  
2 seattle_full_model = sm.OLS(endog=kc_y_seattle, exog=seattle_full_const).fit()
```

## seattle\_VIF

---

In [108]:

```
1 vif_compl = 0
2 while not vif_compl:
3     high_cols_dict = {}
4     mid_cols_dict = {}
5
6     full_const = sm.add_constant(kc_X_seattle)
7     seattle_full_VIF_model = sm.OLS(endog=kc_y_seattle, exog=full_const).fit()
8
9     vif = [variance_inflation_factor(full_const.values, i) for i in range(full_const.shape[1])]
10    vif_df = pd.DataFrame(vif, index=full_const.columns, columns=["Variance Inflation Factor"])
11
12    high_vif_cols = list(vif_df.loc[vif_df['Variance Inflation Factor'] > 10].index)
13    high_vif_vals = list(vif_df.loc[vif_df['Variance Inflation Factor'] > 10].values)
14    if 'const' in high_vif_cols:
15        h_i = high_vif_cols.index('const')
16        high_vif_cols.remove('const')
17        high_vif_vals.pop(h_i)
18
19    mid_vif_cols = \
20    list(vif_df.loc[(vif_df['Variance Inflation Factor'] > 5) & (vif_df['Variance Inflation Factor'] < 10)])
21
22    mid_vif_vals = \
23    list(vif_df.loc[(vif_df['Variance Inflation Factor'] > 5) & (vif_df['Variance Inflation Factor'] < 10)])
24
25    if 'const' in mid_vif_cols:
26        m_i = mid_vif_cols.index('const')
27        mid_vif_cols.remove('const')
28        mid_vif_vals.pop(m_i)
29
30    for g_i, col_grp in enumerate([high_vif_cols, mid_vif_cols]):
31        if g_i==0: grp_dict, grp_vals = high_cols_dict, high_vif_vals
32        if g_i==1: grp_dict, grp_vals = mid_cols_dict, mid_vif_vals
33        if col_grp:
34            for c_i, col in enumerate(col_grp):
35                if col in grp_dict.keys(): grp_dict[col] += grp_vals[c_i]
36                else: grp_dict[col] = grp_vals[c_i]
37
38    high_cols_df = pd.DataFrame(high_cols_dict.values(), high_cols_dict.keys(), columns=['VIF'])
39    mid_VIF_cols_df = pd.DataFrame(mid_cols_dict.values(), mid_cols_dict.keys(), columns=['VIF'])
40
41    seattle_VIF = high_cols_df.sort_values('VIF', ascending=False)
42    mid_VIF_cols_df = mid_VIF_cols_df.sort_values('VIF', ascending=False)
43
44    VIF_drop_cols = []
45    for o_col in list(cat_dummy_dict.keys())[:-1]:
46        if any(seattle_VIF.index.map(lambda x: o_col in x)):
47            first_pred_col = seattle_VIF.loc[seattle_VIF.index.map(lambda x: o_col in x)].index[0]
48            VIF_drop_cols.append(first_pred_col)
49
50    if VIF_drop_cols:
51        kc_X_seattle.drop(VIF_drop_cols, axis=1, inplace=True)
52        print(bold_red + 'High VIF Dropped Columns'+ every_off, '\n', VIF_drop_cols, '\n')
53
54    if not VIF_drop_cols and not mid_VIF_cols_df.empty:
55        mid_VIF_drop_cols = []
56        for o_col in list(cat_dummy_dict.keys())[:-1]:
57            if any(mid_VIF_cols_df.index.map(lambda x: o_col in x)):
58                first_pred_col = mid_VIF_cols_df.loc[mid_VIF_cols_df.index.map(lambda x: o_col in x)].index[0]
59                mid_VIF_drop_cols.append(first_pred_col)
60
61    if mid_VIF_drop_cols:
62        kc_X_seattle.drop(mid_VIF_drop_cols, axis=1, inplace=True)
63        print(bold_red + 'Mid VIF Dropped Columns'+ every_off, '\n', mid_VIF_drop_cols, '\n')
64
65    if not VIF_drop_cols and mid_VIF_cols_df.empty: vif_compl = 1
66
67 print(bold_red + 'VIF Elimination Finished'+ every_off)
```

High VIF Dropped Columns  
['grade\_7\_Average', 'condition\_Average']

VIF Elimination Finished

In [109]:

```

1 best_model_found = 0
2 RFECV_round = 0
3 while not best_model_found:
4
5     full_const = sm.add_constant(kc_X_seattle)
6     seattle_full_RFECV_model = sm.OLS(endog=kc_y_seattle, exog=full_const).fit()
7
8     splitter = ShuffleSplit(n_splits=6, test_size=0.2, random_state=36)
9     seattle_X_for_RFECV = StandardScaler().fit_transform(kc_X_seattle)
10    model_for_RFECV = LinearRegression()
11    selector = RFECV(model_for_RFECV, cv=splitter)
12    selector.fit(seattle_X_for_RFECV, kc_y_seattle)
13
14    RFECV_bad_cols = [col for c_i, col in enumerate(kc_X_seattle.columns) if not selector.support_[c_i]]
15
16    if RFECV_bad_cols:
17        RFECV_drop_cols = []
18        for o_col in list(cat_dummy_dict.keys()):
19            o_df = pd.DataFrame()
20            for r_col in RFECV_bad_cols:
21                if r_col.startswith(o_col) or r_col.endswith(o_col):
22                    r_prob, r_coef = seattle_full_RFECV_model.pvalues[r_col], seattle_full_RFECV_model.params[r_col]
23                    r_df = pd.DataFrame([r_prob, r_coef], columns=[r_col], index=['p_value', 'coeff'])
24                    o_df = pd.concat([o_df, r_df])
25            if not o_df.empty:
26                o_max_col = o_df.loc[o_df.p_value == o_df.p_value.max()].index[0]
27                RFECV_drop_cols.append(o_max_col)
28        RFECV_round += 1
29
30        kc_X_seattle.drop(RFECV_drop_cols, axis=1, inplace=True)
31        print(bold_red + 'RFECV Dropped Columns' + every_off, '\n', RFECV_drop_cols, '\n')
32
33    else: best_model_found = 1
34
35
36 print(bold_red + 'RFECV Finished' + every_off)

```

RFECV Dropped Columns

['view\_AVERAGE', 'condition\_Fair', 'renovated', 'yr\_built\_1920\_to\_1940\_s']

RFECV Finished

---

seattle\_pvals

---

In [110]:

```
1 high_pvals = 1
2
3 while high_pvals:
4     seattle_pval_const = sm.add_constant(kc_X_seattle)
5     seattle_pval_model = sm.OLS(endog=kc_y_seattle, exog=seattle_pval_const).fit()
6
7     high_pval_list = list(seattle_pval_model.pvalues.loc[seattle_pval_model.pvalues >= .05].sort_values(ascending=True))
8
9     if high_pval_list:
10         high_pval_drop_col = high_pval_list[0]
11         kc_X_seattle.drop(high_pval_drop_col, axis=1, inplace=True)
12         print(str(high_pvals) + ' - ' + bold_red + 'High p-value Column Dropped' + every_off + ':\\n', high_pval_list)
13         high_pvals += 1
14
15     else: high_pvals = 0
16
17 print(bold_red + 'High p_value Elimination Finsihed'+ every_off)
```

1 - High p-value Column Dropped:  
view\_FAIR

2 - High p-value Column Dropped:  
basement

High p\_value Elimination Finsihed

## seattle\_fin\_model

---

In [111]:

```
1 seattle_fin_const = seattle_pval_const
2 seattle_fin_model = seattle_pval_model
```

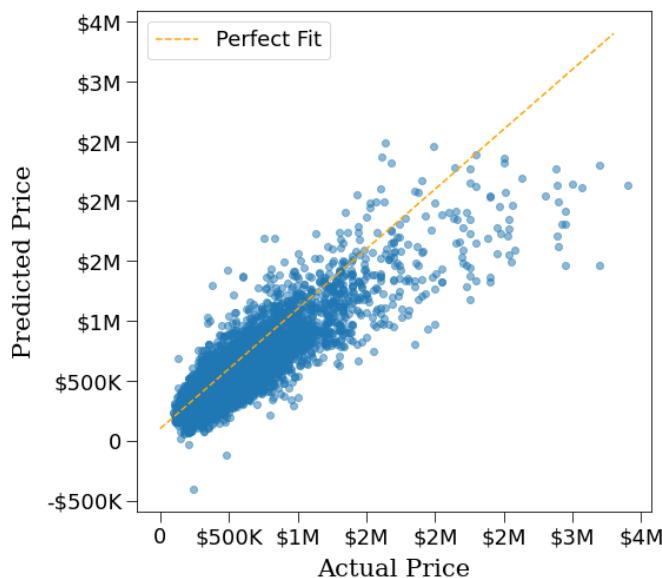
## Investigating the Linearity, Normality and Homoscedasticity of seattle\_fin\_model

---

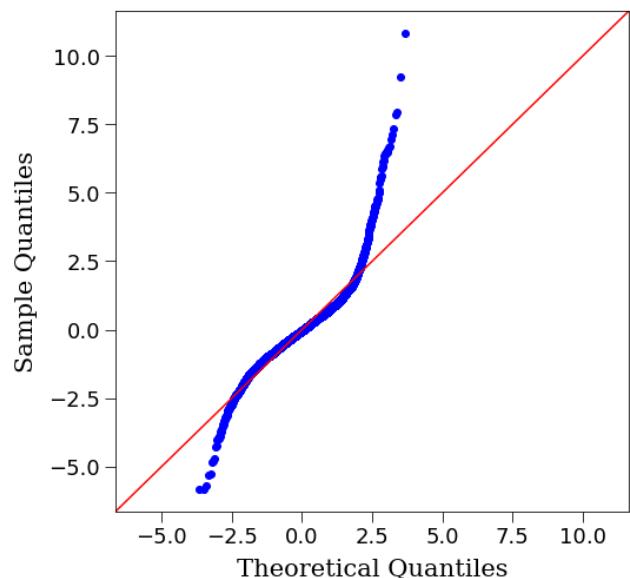
In [112]:

```
1 fig = plt.figure()
2 gs = fig.add_gridspec(2, 2, wspace=.3, hspace=.3)
3
4 lin_ax = fig.add_subplot(gs[0, 0])
5 norm_ax = fig.add_subplot(gs[0, 1])
6 homo_ax = fig.add_subplot(gs[1, :])
7
8 seattle_preds = seattle_fin_model.predict(seattle_fin_const)
9 perfect_line = np.arange(kc_y_seattle.min(), kc_y_seattle.max())
10 seattle_resids = (kc_y_seattle - seattle_preds)
11
12 lin_ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
13 lin_ax.scatter(kc_y_seattle, seattle_preds, alpha=0.5)
14 lin_ax.set_xlabel("Actual Price", family='serif')
15 lin_ax.set_ylabel("Predicted Price", family='serif')
16 lin_ax.set_title('Linearity Check', family='serif')
17 lin_ax.legend()
18
19 sm.graphics.qqplot(seattle_resids, dist=stats.norm, line='45', fit=True, ax=norm_ax)
20 norm_ax.set_title('Normality Check', family='serif')
21
22 homo_ax.scatter(seattle_preds, seattle_resids, alpha=0.5)
23 homo_ax.plot(seattle_preds, [0 for i in range(len(kc_X_seattle))])
24 homo_ax.set_xlabel("Predicted Price", family='serif')
25 homo_ax.set_ylabel("Actual Price - Predicted Price", family='serif')
26 homo_ax.set_title('Homoscedasticity Check', family='serif')
27
28 for a_i, ax in enumerate([lin_ax, norm_ax, homo_ax]):
29     if a_i!=1:
30         ax.xaxis.set_major_formatter(viz_currency_formatter)
31         ax.yaxis.set_major_formatter(viz_currency_formatter)
32     else:
33         ax.set_xlabel(ax.get_xlabel(), family='serif')
34         ax.set_ylabel(ax.get_ylabel(), family='serif')
35
36 fig.savefig('visuals/seattle_model_check.png' , bbox_inches='tight')
37
38 plt.show()
```

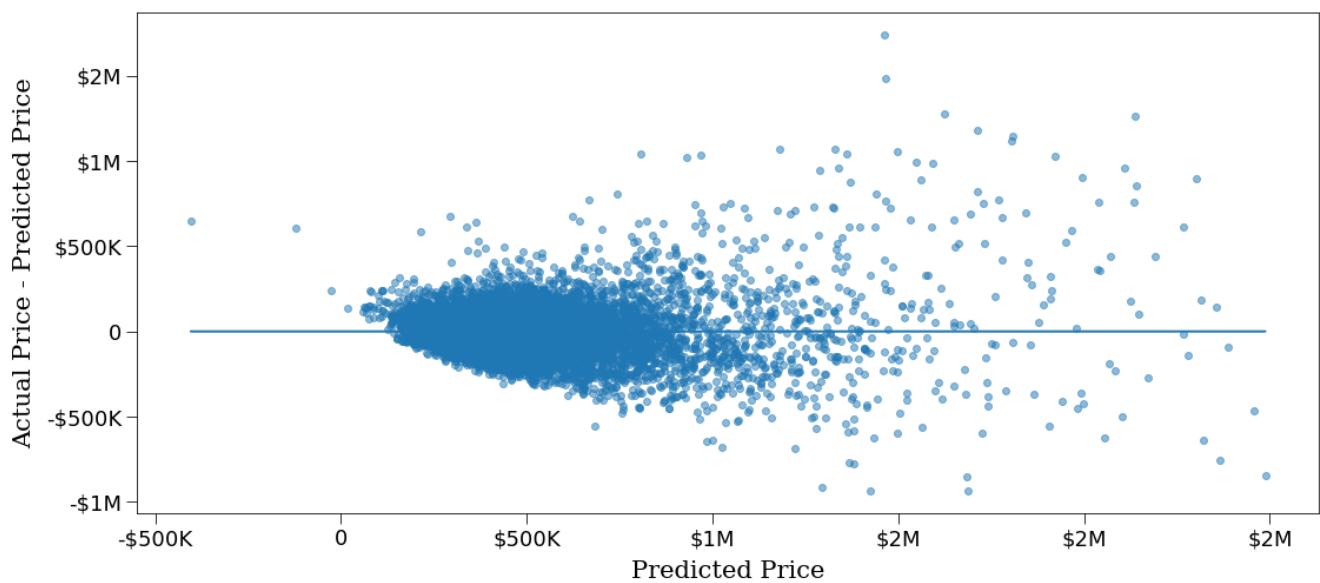
## Linearity Check



## Normality Check



## Homoscedasticity Check



## seattle\_log\_model

```
In [113]: 1 seattle_log_y = np.log(kc_y_seattle)
```

```
In [114]: 1 seattle_log_const = sm.add_constant(kc_X_seattle)
2 seattle_log_model = sm.OLS(endog=seattle_log_y, exog=seattle_log_const).fit()
```

In [115]: 1 seattle\_log\_model.summary()

Out[115]: OLS Regression Results

<b>Dep. Variable:</b>	price	<b>R-squared:</b>	0.698				
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.697				
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	883.9				
<b>Date:</b>	Thu, 18 Aug 2022	<b>Prob (F-statistic):</b>	0.00				
<b>Time:</b>	19:17:06	<b>Log-Likelihood:</b>	-1063.4				
<b>No. Observations:</b>	8828	<b>AIC:</b>	2175.				
<b>Df Residuals:</b>	8804	<b>BIC:</b>	2345.				
<b>Df Model:</b>	23						
<b>Covariance Type:</b>	nonrobust						
		<b>coef</b>	<b>std err</b>	<b>t</b>	<b>P&gt; t </b>	<b>[0.025</b>	<b>0.975]</b>
const	12.5694	0.019	649.446	0.000	12.531	12.607	
sqft_living	0.0002	7.7e-06	26.167	0.000	0.000	0.000	
sqft_lot	-6.238e-06	7.96e-07	-7.838	0.000	-7.8e-06	-4.68e-06	
sqft_living15	0.0002	8.56e-06	21.406	0.000	0.000	0.000	
sqft_lot15	-1.758e-05	1.08e-06	-16.309	0.000	-1.97e-05	-1.55e-05	
floors	0.0254	0.008	3.274	0.001	0.010	0.041	
bedrooms	-0.0288	0.004	-7.046	0.000	-0.037	-0.021	
bathrooms	0.0455	0.007	6.998	0.000	0.033	0.058	
grade_5_Fair	-0.4120	0.031	-13.255	0.000	-0.473	-0.351	
grade_6_Low_Average	-0.2459	0.009	-26.408	0.000	-0.264	-0.228	
grade_8_Good	0.1949	0.008	22.966	0.000	0.178	0.211	
grade_9_Better	0.4383	0.014	31.096	0.000	0.411	0.466	
grade_10_Very_Good	0.5654	0.023	24.704	0.000	0.521	0.610	
grade_11_Excellent	0.7060	0.041	17.165	0.000	0.625	0.787	
grade_12_Luxury	1.1849	0.094	12.670	0.000	1.002	1.368	
view_GOOD	0.0447	0.018	2.474	0.013	0.009	0.080	
view_EXCELLENT	0.2145	0.027	8.048	0.000	0.162	0.267	
waterfront	0.3810	0.043	8.958	0.000	0.298	0.464	
condition_Good	0.0705	0.007	9.533	0.000	0.056	0.085	
condition_Very_Good	0.1024	0.010	10.001	0.000	0.082	0.123	
yr_built_1940_to_1960_s	-0.1483	0.008	-19.015	0.000	-0.164	-0.133	
yr_built_1960_to_1980_s	-0.3214	0.011	-28.962	0.000	-0.343	-0.300	
yr_built_1980_to_2000_s	-0.3039	0.014	-22.264	0.000	-0.331	-0.277	
yr_built_2000_to_2020_s	-0.3417	0.013	-26.932	0.000	-0.367	-0.317	
<b>Omnibus:</b>	217.004	<b>Durbin-Watson:</b>	1.367				
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	292.734				
<b>Skew:</b>	-0.291	<b>Prob(JB):</b>	2.71e-64				
<b>Kurtosis:</b>	3.676	<b>Cond. No.</b>	3.59e+05				

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

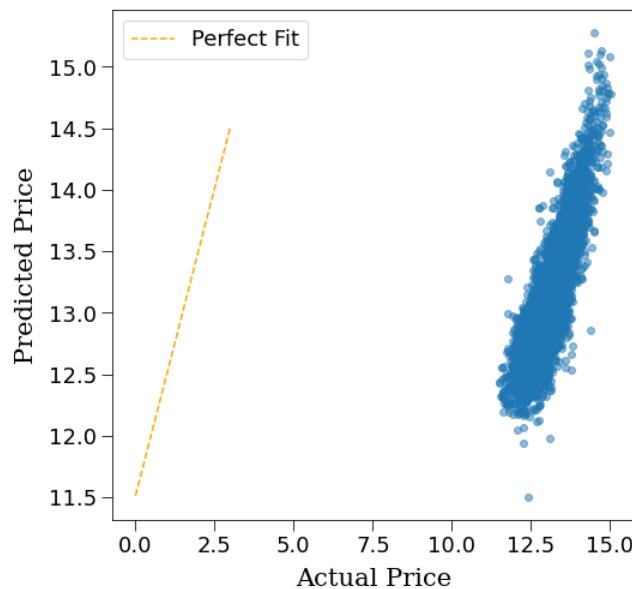
[2] The condition number is large, 3.59e+05. This might indicate that there are strong multicollinearity or other numerical problems.



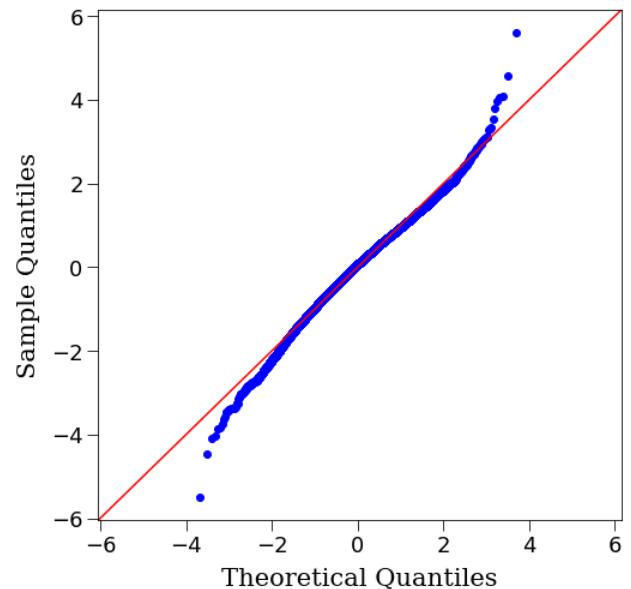
In [116]:

```
1 fig = plt.figure()
2 gs = fig.add_gridspec(2, 2, wspace=.3, hspace=.3)
3
4 lin_ax = fig.add_subplot(gs[0, 0])
5 norm_ax = fig.add_subplot(gs[0, 1])
6 homo_ax = fig.add_subplot(gs[1, :])
7
8 seattle_preds = seattle_log_model.predict(seattle_log_const)
9 perfect_line = np.arange(seattle_log_y.min(), seattle_log_y.max())
10 seattle_resids = (seattle_log_y - seattle_preds)
11
12 lin_ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
13 lin_ax.scatter(seattle_log_y, seattle_preds, alpha=0.5)
14 lin_ax.set_xlabel("Actual Price", family='serif')
15 lin_ax.set_ylabel("Predicted Price", family='serif')
16 lin_ax.set_title('Linearity Check', family='serif')
17 lin_ax.legend()
18
19 sm.graphics.qqplot(seattle_resids, dist=stats.norm, line='45', fit=True, ax=norm_ax)
20 norm_ax.set_title('Normality Check', family='serif')
21 norm_ax.set_xlabel(norm_ax.get_xlabel(), family='serif')
22 norm_ax.set_ylabel(norm_ax.get_ylabel(), family='serif')
23
24 homo_ax.scatter(seattle_preds, seattle_resids, alpha=0.5)
25 homo_ax.plot(seattle_preds, [0 for i in range(len(kc_X_seattle))])
26 homo_ax.set_xlabel("Predicted Price", family='serif')
27 homo_ax.set_ylabel("Actual Price - Predicted Price", family='serif')
28 homo_ax.set_title('Homoscedasticity Check', family='serif')
29
30 fig.savefig('visuals/seattle_model_check_log.png' , bbox_inches='tight')
31
32 plt.show()
```

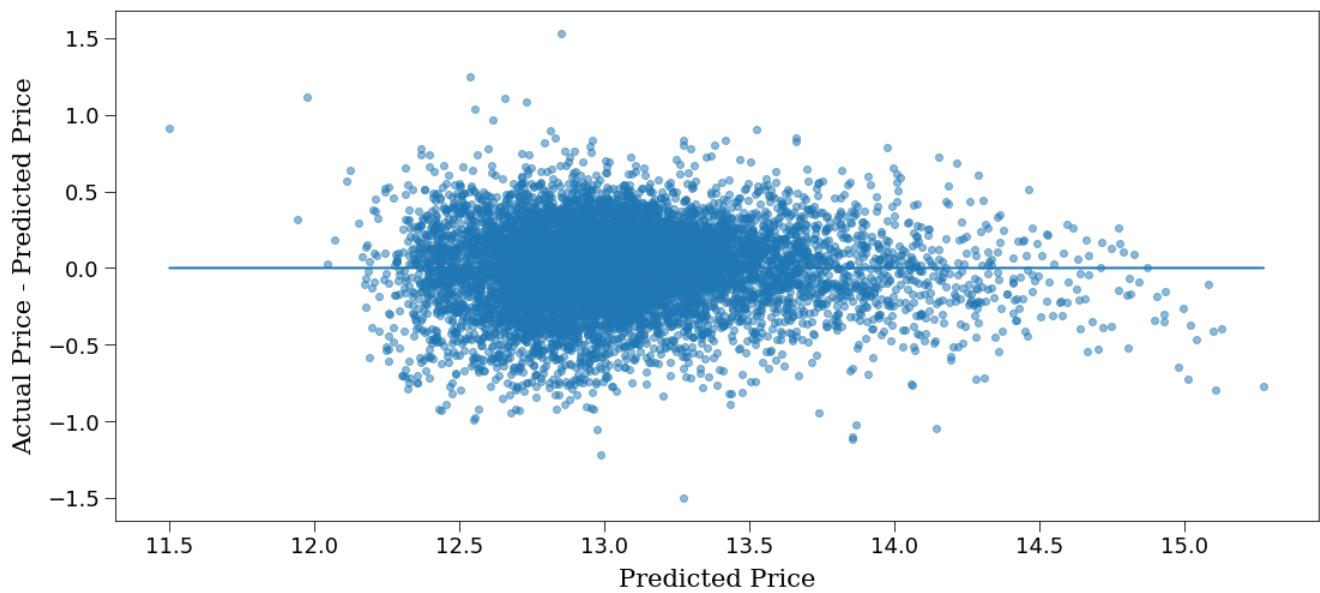
## Linearity Check



## Normality Check



## Homoscedasticity Check



## Comparing seattle Models

```
In [117]: 1 seattle_model_list = [seattle_base_model, seattle_full_model, seattle_full_VIF_model, seattle_full_RFECV_mo  
2                         seattle_fin_model, seattle_log_model]  
3  
4 seattle_model_names = ['base', 'full', 'full_VIF', 'full_RFECV', 'fin', 'log']
```

In [118]:

```
1 seattle_r_vals = {}
2 seattle_r_adj_vals = {}
3 seattle_compare_params_df = pd.DataFrame()
4 for ols_i, (ols_model, model_name) in enumerate(zip(seattle_model_list, seattle_model_names)):
5     model_params = ols_model.params
6     model_params.name = model_name
7     seattle_r_vals[model_name] = ols_model.rsquared
8     seattle_r_adj_vals[model_name] = ols_model.rsquared_adj
9
10    seattle_compare_params_df = pd.concat([seattle_compare_params_df, model_params], axis=1)
11
12 for r_i, r_dict in enumerate([seattle_r_vals, seattle_r_adj_vals]):
13     r_name = 'r_score' if r_i==0 else 'r_adj_score'
14     r_ser = pd.DataFrame(r_dict.values(), index=r_dict.keys(), columns=[r_name]).transpose()
15
16    seattle_compare_params_df = pd.concat([seattle_compare_params_df, r_ser])
```

In [119]:

```

1 for col in seattle_compare_params_df.columns[:-1]:
2     seattle_compare_params_df[col] = seattle_compare_params_df[col].round(2)
3
4 display(seattle_compare_params_df)

```

	base	full	full_VIF	full_RFECV	fin	log
	const	7148.41	491438.27	158847.89	155356.89	150633.28
<b>sqft_living</b>	293.17	135.15	136.21	136.39	134.33	0.000201
<b>sqft_lot</b>	NaN	-2.51	-2.51	-2.54	-2.48	-0.000006
<b>sqft_living15</b>	NaN	131.77	131.11	131.46	131.75	0.000183
<b>sqft_lot15</b>	NaN	-10.48	-10.53	-10.54	-10.41	-0.000018
<b>floors</b>	NaN	26516.04	26529.99	26635.76	29577.71	0.025384
<b>bedrooms</b>	NaN	-20287.84	-20608.57	-20596.79	-20490.59	-0.028837
<b>bathrooms</b>	NaN	22132.33	22340.04	22311.97	21428.51	0.045524
<b>basement</b>	NaN	-6765.95	-7093.05	-7230.78	NaN	NaN
<b>grade_5_Fair</b>	NaN	-553126.46	-85545.22	-87652.26	-85594.76	-0.411990
<b>grade_6_Low_Average</b>	NaN	-536149.09	-62415.59	-62528.72	-61946.55	-0.245897
<b>grade_7_Average</b>	NaN	-474582.53	NaN	NaN	NaN	NaN
<b>grade_8_Good</b>	NaN	-393683.57	80506.23	80651.86	80722.49	0.194855
<b>grade_9_Better</b>	NaN	-190465.58	283354.67	283346.02	284470.63	0.438257
<b>grade_10_Very_Good</b>	NaN	63240.34	536847.91	536678.08	538149.36	0.565386
<b>grade_11_Excellent</b>	NaN	451839.39	925217.63	925367.66	927254.11	0.705951
<b>grade_12_Luxury</b>	NaN	786496.10	1261792.74	1262217.00	1261689.34	1.184853
<b>view_FAIR</b>	NaN	13241.93	14928.80	14573.30	NaN	NaN
<b>view_AVERAGE</b>	NaN	2284.04	2245.71	NaN	NaN	NaN
<b>view_GOOD</b>	NaN	40582.96	40385.02	39867.15	38758.08	0.044745
<b>view_EXCELLENT</b>	NaN	151865.47	150499.34	150208.06	148506.59	0.214510
<b>waterfront</b>	NaN	288776.95	287809.10	287742.45	287184.80	0.380997
<b>condition_Fair</b>	NaN	123737.20	-17472.86	NaN	NaN	NaN
<b>condition_Average</b>	NaN	142630.60	NaN	NaN	NaN	NaN
<b>condition_Good</b>	NaN	164661.54	22273.24	22633.66	22578.01	0.070457
<b>condition_Very_Good</b>	NaN	197201.90	54683.97	55144.57	54934.00	0.102440
<b>renovated</b>	NaN	-1184.31	-374.69	NaN	NaN	NaN
<b>yr_built_1920_to_1940_s</b>	NaN	-5859.97	-4599.04	NaN	NaN	NaN
<b>yr_built_1940_to_1960_s</b>	NaN	-70917.57	-69411.88	-66702.13	-66135.54	-0.148344
<b>yr_built_1960_to_1980_s</b>	NaN	-170904.35	-169779.18	-166989.49	-166736.97	-0.321357
<b>yr_built_1980_to_2000_s</b>	NaN	-197923.68	-197007.95	-194238.87	-193683.66	-0.303857
<b>yr_built_2000_to_2020_s</b>	NaN	-202398.48	-201521.89	-198890.26	-200151.41	-0.341686
<b>r_score</b>	0.49	0.75	0.75	0.75	0.75	0.697796
<b>r_adj_score</b>	0.49	0.75	0.75	0.75	0.75	0.697006

## seattle\_log\_eq

In [120]:

```

1 seattle_log_int = int(np.exp(seattle_compare_params_df.log.dropna()[0]))
2 seattle_log_int

```

Out[120]: 287626

```
In [121]: 1 seattle_log_percs = seattle_compare_params_df.log.dropna()[1:-2] * 100
2 seattle_log_percs.astype(int)
```

```
Out[121]: sqft_living      0
sqft_lot          0
sqft_living15     0
sqft_lot15         0
floors            2
bedrooms          -2
bathrooms         4
grade_5_Fair      -41
grade_6_Low_Average -24
grade_8_Good       19
grade_9_Better     43
grade_10_Very_Good 56
grade_11_Excellent 70
grade_12_Luxury    118
view_GOOD          4
view_EXCELLENT     21
waterfront         38
condition_Good     7
condition_Very_Good 10
yr_built_1940_to_1960_s -14
yr_built_1960_to_1980_s -32
yr_built_1980_to_2000_s -30
yr_built_2000_to_2020_s -34
Name: log, dtype: int32
```

```
In [122]: 1 seattle_fin_percs = list(seattle_log_percs.values.astype(int))
2 seattle_fin_preds = list(seattle_log_percs.index)
```

```
In [123]: 1 seattle_perc_eq = ['{}% * {}'.format(perc, pred) for perc, pred in zip(seattle_fin_percs, seattle_fin_preds)]
```

```
In [124]: 1 model_eq = 'price = '
2 model_int_string = str(seattle_log_int) + '\n'
3 model_var_string = ' +\n'.join(seattle_perc_eq)
```

```
In [125]: 1 seattle_log_eq = model_eq + model_int_string + model_var_string
2
3 print(seattle_log_eq)
```

```
price = 287626 +
0% * sqft_living +
0% * sqft_lot +
0% * sqft_living15 +
0% * sqft_lot15 +
2% * floors +
-2% * bedrooms +
4% * bathrooms +
-41% * grade_5_Fair +
-24% * grade_6_Low_Average +
19% * grade_8_Good +
43% * grade_9_Better +
56% * grade_10_Very_Good +
70% * grade_11_Excellent +
118% * grade_12_Luxury +
4% * view_GOOD +
21% * view_EXCELLENT +
38% * waterfront +
7% * condition_Good +
10% * condition_Very_Good +
-14% * yr_built_1940_to_1960_s +
-32% * yr_built_1960_to_1980_s +
-30% * yr_built_1980_to_2000_s +
-34% * yr_built_2000_to_2020_s
```

In [126]:

```
1 out_seattle_full_const = sm.add_constant(kc_X_out_seattle)
2 out_seattle_full_model = sm.OLS(endog=kc_y_out_seattle, exog=out_seattle_full_const).fit()
```

**out\_seattle\_VIF**

---

In [127]:

```
1 vif_compl = 0
2 while not vif_compl:
3     high_cols_dict = {}
4     mid_cols_dict = {}
5
6     full_const = sm.add_constant(kc_X_out_seattle)
7     out_seattle_full_VIF_model = sm.OLS(endog=kc_y_out_seattle, exog=full_const).fit()
8
9     vif = [variance_inflation_factor(full_const.values, i) for i in range(full_const.shape[1])]
10    vif_df = pd.DataFrame(vif, index=full_const.columns, columns=["Variance Inflation Factor"])
11
12    high_vif_cols = list(vif_df.loc[vif_df['Variance Inflation Factor'] > 10].index)
13    high_vif_vals = list(vif_df.loc[vif_df['Variance Inflation Factor'] > 10].values)
14    if 'const' in high_vif_cols:
15        h_i = high_vif_cols.index('const')
16        high_vif_cols.remove('const')
17        high_vif_vals.pop(h_i)
18
19    mid_vif_cols = \
20        list(vif_df.loc[(vif_df['Variance Inflation Factor'] > 5) & (vif_df['Variance Inflation Factor'] < 10)])
21
22    mid_vif_vals = \
23        list(vif_df.loc[(vif_df['Variance Inflation Factor'] > 5) & (vif_df['Variance Inflation Factor'] < 10)])
24
25    if 'const' in mid_vif_cols:
26        m_i = mid_vif_cols.index('const')
27        mid_vif_cols.remove('const')
28        mid_vif_vals.pop(m_i)
29
30    for g_i, col_grp in enumerate([high_vif_cols, mid_vif_cols]):
31        if g_i==0: grp_dict, grp_vals = high_cols_dict, high_vif_vals
32        if g_i==1: grp_dict, grp_vals = mid_cols_dict, mid_vif_vals
33        if col_grp:
34            for c_i, col in enumerate(col_grp):
35                if col in grp_dict.keys(): grp_dict[col] += grp_vals[c_i]
36                else: grp_dict[col] = grp_vals[c_i]
37
38    high_cols_df = pd.DataFrame(high_cols_dict.values(), high_cols_dict.keys(), columns=['VIF'])
39    mid_VIF_cols_df = pd.DataFrame(mid_cols_dict.values(), mid_cols_dict.keys(), columns=['VIF'])
40
41    out_seattle_VIF = high_cols_df.sort_values('VIF', ascending=False)
42    mid_VIF_cols_df = mid_VIF_cols_df.sort_values('VIF', ascending=False)
43
44    VIF_drop_cols = []
45    for o_col in list(cat_dummy_dict.keys())[:-1]:
46        if any(out_seattle_VIF.index.map(lambda x: o_col in x)):
47            first_pred_col = out_seattle_VIF.loc[out_seattle_VIF.index.map(lambda x: o_col in x)].index[0]
48            VIF_drop_cols.append(first_pred_col)
49
50    if VIF_drop_cols:
51        kc_X_out_seattle.drop(VIF_drop_cols, axis=1, inplace=True)
52        print(bold_red + 'High VIF Dropped Columns'+ every_off, '\n', VIF_drop_cols, '\n')
53
54    if not VIF_drop_cols and not mid_VIF_cols_df.empty:
55        mid_VIF_drop_cols = []
56        for o_col in list(cat_dummy_dict.keys())[:-1]:
57            if any(mid_VIF_cols_df.index.map(lambda x: o_col in x)):
58                first_pred_col = mid_VIF_cols_df.loc[mid_VIF_cols_df.index.map(lambda x: o_col in x)].index[0]
59                mid_VIF_drop_cols.append(first_pred_col)
60
61    if mid_VIF_drop_cols:
62        kc_X_out_seattle.drop(mid_VIF_drop_cols, axis=1, inplace=True)
63        print(bold_red + 'Mid VIF Dropped Columns'+ every_off, '\n', mid_VIF_drop_cols, '\n')
64
65    if not VIF_drop_cols and mid_VIF_cols_df.empty: vif_compl = 1
66
67 print(bold_red + 'VIF Elimination Finished'+ every_off)
```

High VIF Dropped Columns

```
['grade_7_Average', 'condition_Average', 'yr_built_1980_to_2000_s']
```

Mid VIF Dropped Columns

```
['sqft_living']
```

**out\_seattle\_RFECV**

In [128]:

```

1 best_model_found = 0
2 RFECV_round = 0
3 while not best_model_found:
4
5     full_const = sm.add_constant(kc_X_out_seattle)
6     out_seattle_full_RFECV_model = sm.OLS(endog=kc_y_out_seattle, exog=full_const).fit()
7
8     splitter = ShuffleSplit(n_splits=6, test_size=0.2, random_state=36)
9     out_seattle_X_for_RFECV = StandardScaler().fit_transform(kc_X_out_seattle)
10    model_for_RFECV = LinearRegression()
11    selector = RFECV(model_for_RFECV, cv=splitter)
12    selector.fit(out_seattle_X_for_RFECV, kc_y_out_seattle)
13
14    RFECV_bad_cols = [col for c_i, col in enumerate(kc_X_out_seattle.columns) if not selector.support_[c_i]]
15
16    if RFECV_bad_cols:
17        RFECV_drop_cols = []
18        for o_col in list(cat_dummy_dict.keys()):
19            o_df = pd.DataFrame()
20            for r_col in RFECV_bad_cols:
21                if r_col.startswith(o_col) or r_col.endswith(o_col):
22                    r_prob, r_coef = out_seattle_full_RFECV_model.pvalues[r_col], out_seattle_full_RFECV_mo
23                    r_df = pd.DataFrame([r_prob, r_coef], columns=[r_col], index=['p_value', 'coeff'])
24                    o_df = pd.concat([o_df, r_df])
25                if not o_df.empty:
26                    o_max_col = o_df.loc[o_df.p_value == o_df.p_value.max()].index[0]
27                    RFECV_drop_cols.append(o_max_col)
28        RFECV_round += 1
29
30        kc_X_out_seattle.drop(RFECV_drop_cols, axis=1, inplace=True)
31        print(bold_red + 'RFECV Dropped Columns' + every_off, '\n', RFECV_drop_cols, '\n')
32
33    else: best_model_found = 1
34
35
36 print(bold_red + 'RFECV Finished' + every_off)

```

RFECV Finished

**out\_seattle\_pvals**

In [129]:

```
1 high_pvals = 1
2
3 while high_pvals:
4     out_seattle_pval_const = sm.add_constant(kc_X_out_seattle)
5     out_seattle_pval_model = sm.OLS(endog=kc_y_out_seattle, exog=out_seattle_pval_const).fit()
6
7     high_pval_list = list(out_seattle_pval_model.pvalues.loc[out_seattle_pval_model.pvalues >= .05].sort_values(ascending=False))
8
9     if high_pval_list:
10        high_pval_drop_col = high_pval_list[0]
11        kc_X_out_seattle.drop(high_pval_drop_col, axis=1, inplace=True)
12        print(str(high_pvals) + ' - ' + bold_red + 'High p-value Column Dropped' + every_off + ':\\n', high_pval_list)
13        high_pvals += 1
14
15    else: high_pvals = 0
16
17 print(bold_red + 'High p_value Elimination Finsihed'+ every_off)
```

1 - High p-value Column Dropped:  
grade\_4\_Low

2 - High p-value Column Dropped:  
condition\_Fair

High p\_value Elimination Finsihed

### out\_seattle\_fin\_model

---

In [130]:

```
1 out_seattle_fin_const = out_seattle_pval_const
2 out_seattle_fin_model = out_seattle_pval_model
```

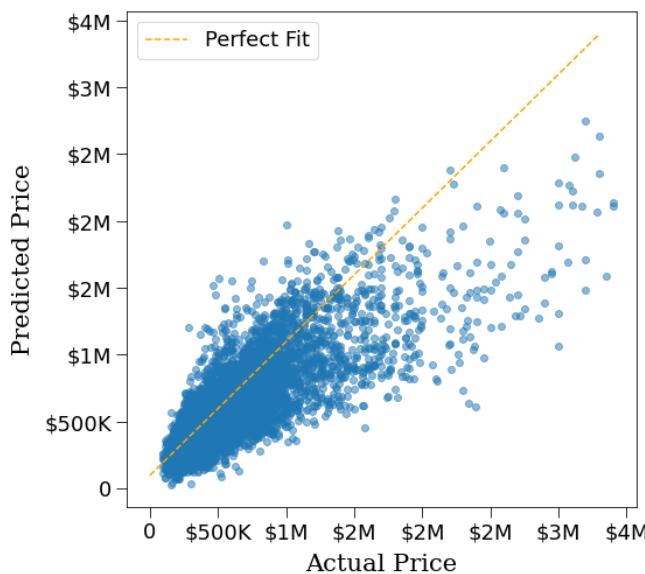
## Investigating the Linearity, Normality and Homoscedasticity of out\_seattle\_fin\_model

---

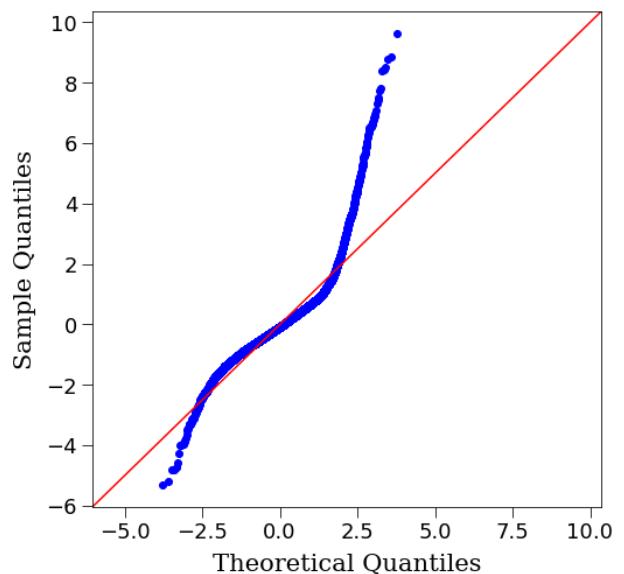
In [131]:

```
1 fig = plt.figure()
2 gs = fig.add_gridspec(2, 2, wspace=.3, hspace=.3)
3
4 lin_ax = fig.add_subplot(gs[0, 0])
5 norm_ax = fig.add_subplot(gs[0, 1])
6 homo_ax = fig.add_subplot(gs[1, :])
7
8 out_seattle_preds = out_seattle_fin_model.predict(out_seattle_fin_const)
9 perfect_line = np.arange(kc_y_out_seattle.min(), kc_y_out_seattle.max())
10 out_seattle_resids = (kc_y_out_seattle - out_seattle_preds)
11
12 lin_ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
13 lin_ax.scatter(kc_y_out_seattle, out_seattle_preds, alpha=0.5)
14 lin_ax.set_xlabel("Actual Price", family='serif')
15 lin_ax.set_ylabel("Predicted Price", family='serif')
16 lin_ax.set_title('Linearity Check', family='serif')
17 lin_ax.legend()
18
19 sm.graphics.qqplot(out_seattle_resids, dist=stats.norm, line='45', fit=True, ax=norm_ax)
20 norm_ax.set_title('Normality Check', family='serif')
21
22 homo_ax.scatter(out_seattle_preds, out_seattle_resids, alpha=0.5)
23 homo_ax.plot(out_seattle_preds, [0 for i in range(len(kc_X_out_seattle))])
24 homo_ax.set_xlabel("Predicted Price", family='serif')
25 homo_ax.set_ylabel("Actual Price - Predicted Price", family='serif')
26 homo_ax.set_title('Homoscedasticity Check', family='serif')
27
28 for a_i, ax in enumerate([lin_ax, norm_ax, homo_ax]):
29     if a_i!=1:
30         ax.xaxis.set_major_formatter(viz_currency_formatter)
31         ax.yaxis.set_major_formatter(viz_currency_formatter)
32     else:
33         ax.set_xlabel(ax.get_xlabel(), family='serif')
34         ax.set_ylabel(ax.get_ylabel(), family='serif')
35
36 fig.savefig('visuals/out_seattle_model_check.png' , bbox_inches='tight')
37
38 plt.show()
```

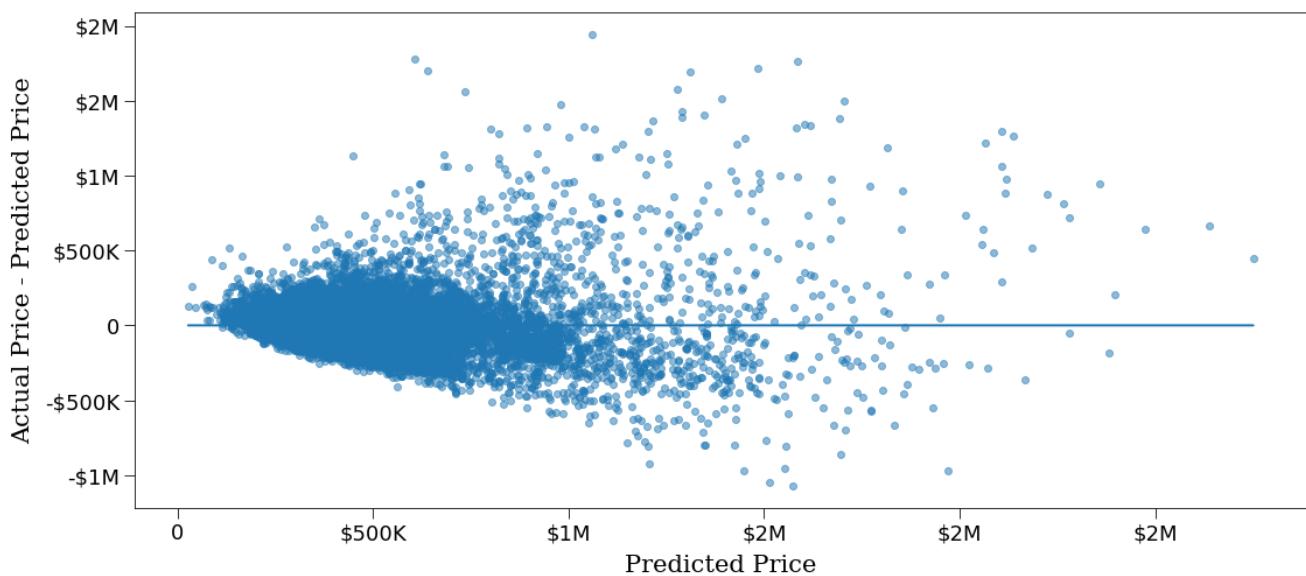
## Linearity Check



## Normality Check



## Homoscedasticity Check



## out\_seattle\_log\_model

```
In [132]: 1 out_seattle_log_y = np.log(kc_y_out_seattle)
```

```
In [133]: 1 out_seattle_log_const = sm.add_constant(kc_X_out_seattle)
2 out_seattle_log_model = sm.OLS(endog=out_seattle_log_y, exog=out_seattle_log_const).fit()
```

In [134]: 1 out\_seattle\_log\_model.summary()

Out[134]: OLS Regression Results

<b>Dep. Variable:</b>	price	<b>R-squared:</b>	0.656				
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.655				
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	908.0				
<b>Date:</b>	Thu, 18 Aug 2022	<b>Prob (F-statistic):</b>	0.00				
<b>Time:</b>	19:17:14	<b>Log-Likelihood:</b>	-3026.0				
<b>No. Observations:</b>	12424	<b>AIC:</b>	6106.				
<b>Df Residuals:</b>	12397	<b>BIC:</b>	6307.				
<b>Df Model:</b>	26						
<b>Covariance Type:</b>	nonrobust						
		<b>coef</b>	<b>std err</b>	<b>t</b>	<b>P&gt; t </b>	<b>[0.025</b>	<b>0.975]</b>
	<b>const</b>	11.9124	0.020	604.705	0.000	11.874	11.951
	<b>sqft_lot</b>	7.593e-07	1.17e-07	6.514	0.000	5.31e-07	9.88e-07
	<b>sqft_living15</b>	0.0002	6.65e-06	32.640	0.000	0.000	0.000
	<b>sqft_lot15</b>	-1.465e-07	1.6e-07	-0.916	0.360	-4.6e-07	1.67e-07
	<b>floors</b>	0.0063	0.009	0.719	0.472	-0.011	0.023
	<b>bedrooms</b>	0.0154	0.004	3.605	0.000	0.007	0.024
	<b>bathrooms</b>	0.1255	0.007	17.883	0.000	0.112	0.139
	<b>basement</b>	0.0435	0.007	5.978	0.000	0.029	0.058
	<b>grade_5_Fair</b>	-0.2987	0.029	-10.434	0.000	-0.355	-0.243
	<b>grade_6_Low_Average</b>	-0.1998	0.014	-14.716	0.000	-0.226	-0.173
	<b>grade_8_Good</b>	0.2084	0.008	27.235	0.000	0.193	0.223
	<b>grade_9_Better</b>	0.4119	0.011	37.473	0.000	0.390	0.433
	<b>grade_10_Very_Good</b>	0.6038	0.015	40.782	0.000	0.575	0.633
	<b>grade_11_Excellent</b>	0.7448	0.023	33.032	0.000	0.701	0.789
	<b>grade_12_Luxury</b>	0.8752	0.049	17.902	0.000	0.779	0.971
	<b>view_FAIR</b>	0.2116	0.027	7.830	0.000	0.159	0.265
	<b>view_AVERAGE</b>	0.0795	0.016	5.065	0.000	0.049	0.110
	<b>view_GOOD</b>	0.0668	0.021	3.116	0.002	0.025	0.109
	<b>view_EXCELLENT</b>	0.2614	0.034	7.604	0.000	0.194	0.329
	<b>waterfront</b>	0.4678	0.047	10.055	0.000	0.377	0.559
	<b>condition_Good</b>	0.0491	0.007	6.820	0.000	0.035	0.063
	<b>condition_Very_Good</b>	0.1325	0.013	10.432	0.000	0.108	0.157
	<b>renovated</b>	0.1626	0.020	8.233	0.000	0.124	0.201
	<b>yr_built_1920_to_1940_s</b>	0.1454	0.023	6.203	0.000	0.099	0.191
	<b>yr_built_1940_to_1960_s</b>	0.2579	0.012	21.048	0.000	0.234	0.282
	<b>yr_built_1960_to_1980_s</b>	0.0878	0.009	10.247	0.000	0.071	0.105
	<b>yr_built_2000_to_2020_s</b>	0.0279	0.008	3.463	0.001	0.012	0.044
	<b>Omnibus:</b>	60.696	<b>Durbin-Watson:</b>	0.891			
	<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	63.893			
	<b>Skew:</b>	0.148	<b>Prob(JB):</b>	1.34e-14			
	<b>Kurtosis:</b>	3.190	<b>Cond. No.</b>	9.88e+05			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 9.88e+05. This might indicate that there are

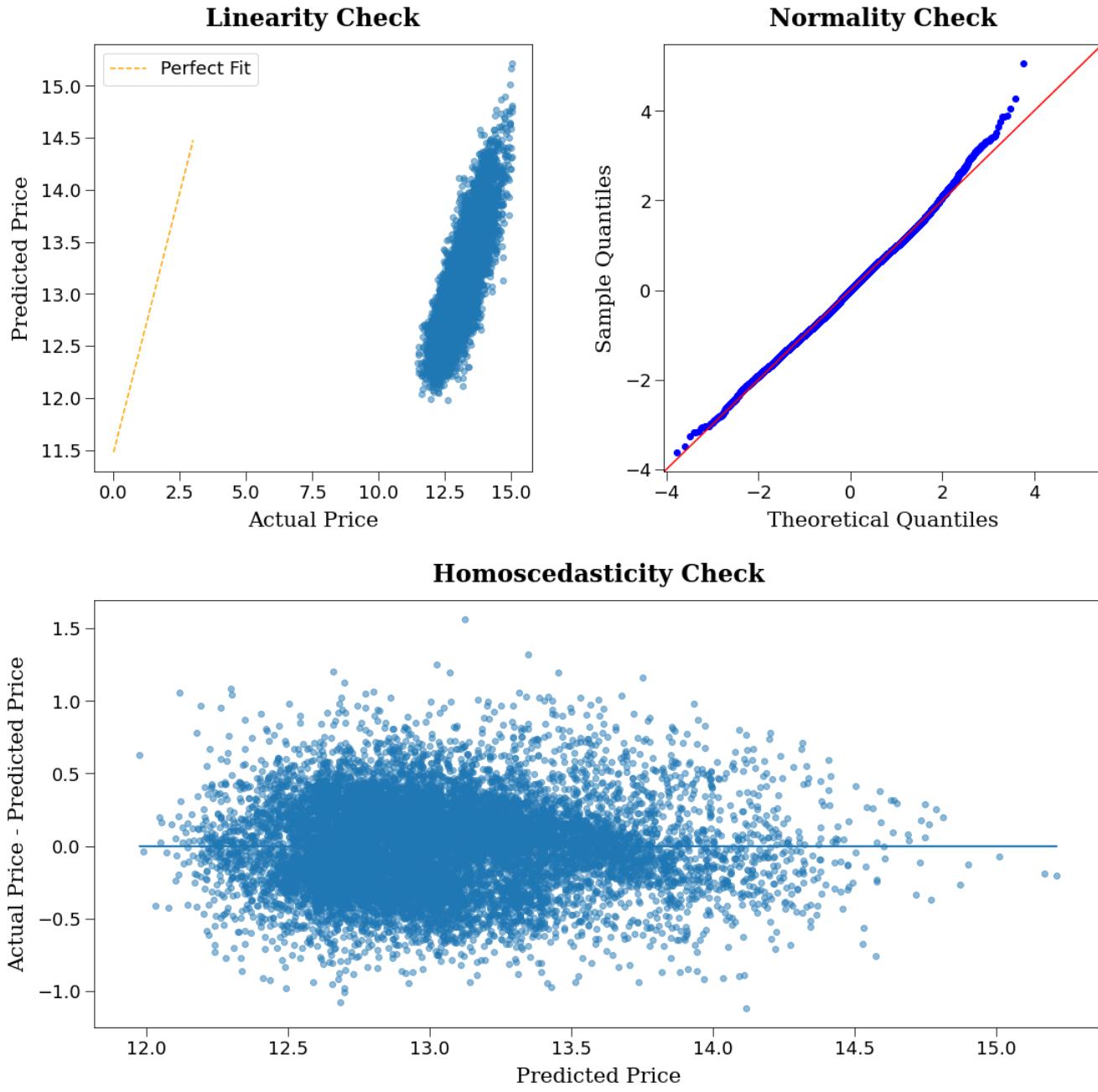
strong multicollinearity or other numerical problems.

## **Investigating the Linearity, Normality and Homoscedasticity of `out_seattle_log_model`**

---

In [135]:

```
1 fig = plt.figure()
2 gs = fig.add_gridspec(2, 2, wspace=.3, hspace=.3)
3
4 lin_ax = fig.add_subplot(gs[0, 0])
5 norm_ax = fig.add_subplot(gs[0, 1])
6 homo_ax = fig.add_subplot(gs[1, :])
7
8 out_seattle_preds = out_seattle_log_model.predict(out_seattle_log_const)
9 perfect_line = np.arange(out_seattle_log_y.min(), out_seattle_log_y.max())
10 out_seattle_resids = (out_seattle_log_y - out_seattle_preds)
11
12 lin_ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
13 lin_ax.scatter(out_seattle_log_y, out_seattle_preds, alpha=0.5)
14 lin_ax.set_xlabel("Actual Price", family='serif')
15 lin_ax.set_ylabel("Predicted Price", family='serif')
16 lin_ax.set_title('Linearity Check', family='serif')
17 lin_ax.legend()
18
19 sm.graphics.qqplot(out_seattle_resids, dist=stats.norm, line='45', fit=True, ax=norm_ax)
20 norm_ax.set_title('Normality Check', family='serif')
21 norm_ax.set_xlabel(norm_ax.get_xlabel(), family='serif')
22 norm_ax.set_ylabel(norm_ax.get_ylabel(), family='serif')
23
24 homo_ax.scatter(out_seattle_preds, out_seattle_resids, alpha=0.5)
25 homo_ax.plot(out_seattle_preds, [0 for i in range(len(kc_X_out_seattle))])
26 homo_ax.set_xlabel("Predicted Price", family='serif')
27 homo_ax.set_ylabel("Actual Price - Predicted Price", family='serif')
28 homo_ax.set_title('Homoscedasticity Check', family='serif')
29
30 fig.savefig('visuals/out_seattle_model_check_log.png' , bbox_inches='tight')
31
32 plt.show()
```



## Comparing out\_seattle Models

```
In [136]: 1 out_seattle_model_list = [out_seattle_base_model, out_seattle_full_model, out_seattle_full_VIF_model,
2                               out_seattle_full_RFECV_model, out_seattle_fin_model, out_seattle_log_model]
3 out_seattle_model_names = ['base', 'full', 'full_VIF', 'full_RFECV', 'fin', 'log']
```

```
In [137]: 1 out_seattle_r_vals = {}
2 out_seattle_r_adj_vals = {}
3 out_seattle_compare_params_df = pd.DataFrame()
4 for ols_i, (ols_model, model_name) in enumerate(zip(out_seattle_model_list, out_seattle_model_names)):
5     model_params = ols_model.params
6     model_params.name = model_name
7     out_seattle_r_vals[model_name] = ols_model.rsquared
8     out_seattle_r_adj_vals[model_name] = ols_model.rsquared_adj
9
10    out_seattle_compare_params_df = pd.concat([out_seattle_compare_params_df, model_params], axis=1)
11
12 for r_i, r_dict in enumerate([out_seattle_r_vals, out_seattle_r_adj_vals]):
13     r_name = 'r_score' if r_i==0 else 'r_adj_score'
14     r_ser = pd.DataFrame(r_dict.values(), index=r_dict.keys(), columns=[r_name]).transpose()
15
16     out_seattle_compare_params_df = pd.concat([out_seattle_compare_params_df, r_ser])
```

In [138]:

```

1 for col in out_seattle_compare_params_df.columns[:-1]:
2     out_seattle_compare_params_df[col] = out_seattle_compare_params_df[col].round(2)
3
4 display(out_seattle_compare_params_df)

```

		base	full	full_VIF	full_RFECV	fin	log
	<b>const</b>	-88031.96	1623854.61	-71551.43	-71551.43	-72877.59	1.191244e+01
	<b>sqft_living</b>	274.79	117.21	NaN	NaN	NaN	NaN
	<b>sqft_lot</b>	NaN	0.09	0.28	0.28	0.27	7.592585e-07
	<b>sqft_living15</b>	NaN	59.57	100.06	100.06	100.09	2.170949e-04
	<b>sqft_lot15</b>	NaN	-0.29	-0.22	-0.22	-0.22	-1.465068e-07
	<b>floors</b>	NaN	-33303.69	-17430.15	-17430.15	-17429.06	6.300600e-03
	<b>bedrooms</b>	NaN	-17308.27	7787.86	7787.86	7859.85	1.535420e-02
	<b>bathrooms</b>	NaN	50836.90	88325.45	88325.45	88620.09	1.254691e-01
	<b>basement</b>	NaN	-12974.73	15322.16	15322.16	15217.88	4.353457e-02
	<b>grade_4_Low</b>	NaN	-1546358.38	-68083.09	-68083.09	NaN	NaN
	<b>grade_5_Fair</b>	NaN	-1542575.16	-70039.11	-70039.11	-69234.79	-2.987118e-01
	<b>grade_6_Low_Average</b>	NaN	-1512696.03	-44490.69	-44490.69	-44259.22	-1.998210e-01
	<b>grade_7_Average</b>	NaN	-1460580.49	NaN	NaN	NaN	NaN
	<b>grade_8_Good</b>	NaN	-1396660.56	79979.87	79979.87	80128.48	2.084065e-01
	<b>grade_9_Better</b>	NaN	-1290985.95	218543.60	218543.60	218659.91	4.119348e-01
	<b>grade_10_Very_Good</b>	NaN	-1126670.65	421548.82	421548.82	421580.38	6.038023e-01
	<b>grade_11_Excellent</b>	NaN	-895490.94	694388.83	694388.83	694329.46	7.448251e-01
	<b>grade_12_Luxury</b>	NaN	-620163.85	1008847.43	1008847.43	1008671.92	8.751767e-01
	<b>view_FAIR</b>	NaN	188561.25	195214.73	195214.73	194746.14	2.115879e-01
	<b>view_AVERAGE</b>	NaN	58891.43	69604.22	69604.22	69726.34	7.951031e-02
	<b>view_GOOD</b>	NaN	85552.92	88884.54	88884.54	88931.12	6.684284e-02
	<b>view_EXCELLENT</b>	NaN	351261.56	367216.01	367216.01	367194.68	2.613644e-01
	<b>waterfront</b>	NaN	584099.57	597325.25	597325.25	597656.29	4.678498e-01
	<b>condition_Fair</b>	NaN	-119905.87	-29974.66	-29974.66	NaN	NaN
	<b>condition_Average</b>	NaN	-92324.19	NaN	NaN	NaN	NaN
	<b>condition_Good</b>	NaN	-62367.90	35968.72	35968.72	36503.41	4.907338e-02
	<b>condition_Very_Good</b>	NaN	-26097.65	77089.86	77089.86	77828.57	1.324609e-01
	<b>renovated</b>	NaN	139378.96	160733.11	160733.11	161218.90	1.626134e-01
	<b>yr_built_1920_to_1940_s</b>	NaN	341.04	71212.78	71212.78	69159.60	1.454178e-01
	<b>yr_built_1940_to_1960_s</b>	NaN	74516.72	158097.61	158097.61	157501.35	2.579424e-01
	<b>yr_built_1960_to_1980_s</b>	NaN	-20379.14	60449.98	60449.98	60335.75	8.779453e-02
	<b>yr_built_1980_to_2000_s</b>	NaN	-78898.06	NaN	NaN	NaN	NaN
	<b>yr_built_2000_to_2020_s</b>	NaN	-67913.07	23061.71	23061.71	23224.04	2.788341e-02
	<b>r_score</b>	0.50	0.67	0.65	0.65	0.65	6.556790e-01
	<b>r_adj_score</b>	0.50	0.67	0.65	0.65	0.65	6.549568e-01

### out\_seattle\_log\_eq

In [139]:

```

1 out_seattle_log_int = int(np.exp(out_seattle_compare_params_df.log.dropna()[0]))
2 out_seattle_log_int

```

Out[139]: 149110

```
In [140]: 1 out_seattle_log_percs = out_seattle_compare_params_df.log.dropna()[1:-2] * 100
2 out_seattle_log_percs.astype(int)
```

```
Out[140]: sqft_lot          0
sqft_living15        0
sqft_lot15          0
floors              0
bedrooms            1
bathrooms           12
basement             4
grade_5_Fair         -29
grade_6_Low_Average -19
grade_8_Good          20
grade_9_Better         41
grade_10_Very_Good      60
grade_11_Excellent       74
grade_12_Luxury          87
view_FAIR            21
view_AVERAGE          7
view_GOOD             6
view_EXCELLENT         26
waterfront            46
condition_Good          4
condition_Very_Good       13
renovated              16
yr_built_1920_to_1940_s     14
yr_built_1940_to_1960_s     25
yr_built_1960_to_1980_s      8
yr_built_2000_to_2020_s      2
Name: log, dtype: int32
```

```
In [141]: 1 out_seattle_fin_percs = list(out_seattle_log_percs.values.astype(int))
2 out_seattle_fin_preds = list(out_seattle_log_percs.index)
```

```
In [142]: 1 out_seattle_perc_eq = ['{}% * {}'.format(perc, pred) for perc, pred in zip(out_seattle_fin_percs, out_seatt
```

```
In [143]: 1 model_eq = 'price = '
2 model_int_string = str(out_seattle_log_int) + ' +\n'
3 model_var_string = ' +\n'.join(out_seattle_perc_eq)
```

In [144]:

```
1 out_seattle_log_eq = model_eq + model_int_string + model_var_string
2
3 print(out_seattle_log_eq)

price = 149110 +
0% * sqft_lot +
0% * sqft_living15 +
0% * sqft_lot15 +
0% * floors +
1% * bedrooms +
12% * bathrooms +
4% * basement +
-29% * grade_5_Fair +
-19% * grade_6_Low_Average +
20% * grade_8_Good +
41% * grade_9_Better +
60% * grade_10_Very_Good +
74% * grade_11_Excellent +
87% * grade_12_Luxury +
21% * view_FAIR +
7% * view_AVERAGE +
6% * view_GOOD +
26% * view_EXCELLENT +
46% * waterfront +
4% * condition_Good +
13% * condition_Very_Good +
16% * renovated +
14% * yr_built_1920_to_1940_s +
25% * yr_built_1940_to_1960_s +
8% * yr_built_1960_to_1980_s +
2% * yr_built_2000_to_2020_s
```

## Stakeholder and Business Problem Decision

---

Based on the results obtained through the three models I created, I chose a real estate developer as the stakeholder for this project. While I could have chosen a real estate agency, I felt a developer could make better use of the insights I gained through my analysis. Real estate agencies would be limited by the desires of their client and the physical location of the client's property. Developers have more freedom in their decision making, both in terms what changes to make to the properties they acquire, and what properties to acquire in the first place. They may ultimately rely on investors to acquire the property, but they will need an analysis like this, to convince those investors of a property's / design's value. A real estate developer could also take on clients simply wanting renovations, or even remodeling services. My analysis would just be a step in a process, of course, which I will discuss more in the [Future Investigations](#) section. I named my hypothetical client King County Development.



For the specific business problem, I chose to provide the real estate developer with key insights into which property features were the most relevant in predicting the sales price, and how much the sales price would be affected by either a one-unit increase in those predictors, or if a property were to have certain categorical features.

## Insights and Conclusions

---

By creating the separate models, I was able to gain valuable insights. I created a dataframe and visualizations with the coefficients of the three log-transformed models to identify the differences between the predictors that were included in each model and their values more easily.

#### Intercepts

- The Seattle model had the highest intercept, the entire King County model had the second highest, and the Outside Seattle model had the lowest.
- Seattle Model - \$287,626.85
- Entire King County Model - \$261,735.06
- Outside Seattle Model - \$149,110.47

#### sqft Columns

- These columns were not important on a per unit basis, but I could still judge them by whether they were included, whether they were positive or negative, and if they were = 0.02 in the dataframe / visualizations, or  $\approx 0$ .
- `sqft_living`
  - It was only even included for the prices of residential property inside Seattle, and the fact that it was 0.02 and positive, showing that living space is very important inside Seattle.
- `sqft_living15`
  - It was positive 0.02, or very important, in all three models.
- `sqft_lot`
  - It was positive  $\approx 0$  in the entire King County and the Outside Seattle models, and negative  $\approx 0$  in the Seattle model, meaning that it was beneficial to the former, and detrimental to the latter, but not as important or harmful as if it was 0.02.
- `sqft_lot15`
  - It was negative  $\approx 0$ , or detrimental, in all three models.

#### floors

- This was interesting. It was most important in the entire King County model. In the Seattle model, it was still important, but in the Outside Seattle model, it was barely important at all.

#### bedrooms

- Another interesting predictor. It wasn't even included in the entire King County model, and it actually had a negative effect in the Seattle model. However, it did have a positive effect in the Outside Seattle model.

#### bathrooms

- It was very important in the entire King County model, and even more so in the Outside Seattle model. In the Seattle model it was important, especially compared to the previous predictors, but not nearly as much as in the other models.

#### basement

- It wasn't even included in the Seattle model. It was very important in the entire King County model. In the Outside Seattle model, it was important, and its worth noting that having a basement adds more value than additional floors outside of Seattle.

#### grade

- This seems to be the most important feature when determining the price of a property. The categories in this feature resulted in the largest changes in price, both positively and negatively. Anything below `7_Average` had a negative effect if it was included at all, and the lower the `grade`, the worse the effect, while `7_Average` itself was not included in any of the models. Anything above `7_Average` had a positive effect, and the size of the effect increases as `grade` does. A few things that are worth noting, is how much of an increase in value is gained by increasing the `grade` from `8_Good` to `9_Better`, as well as the increase from `9_Better` to `10_Very_Good`, and another is how much a `grade` of `12_Luxury` increases the price for Seattle properties.
- Specific Recommendation
  - Create a database of example pictures of properties with each `grade` value (*as many as can be gathered*)
  - These could be used to guide King County Development as they design new properties, or act as a guide, or as an incentive in any promotional material, for any clients seeking renovation or remodeling services.

#### view

- The FAIR and AVERAGE dummy columns weren't even included in the Seattle model, while even having a FAIR view was very important in the Outside Seattle model. It is also important in the entire King County model, although less so. Having an AVERAGE view added a similar increase in value in both the Outside Seattle and entire King County models, so an AVERAGE view could possibly be the standard / baseline in Seattle, as the `grade_7_Average` column seemed to be the standard / baseline in the `grade` column. It is also especially worth noting how much of increase having an EXCELLENT view has on the price.
- Specific Recommendation
  - Again, create a database of example pictures of properties with each `view` value, which could be used for the purposes I've previously described.

#### waterfront

- Having a waterfront property understandably adds significant value to the price, in all three models, and is the most significant in the Outside Seattle model.

#### condition

- The Fair and Average dummy columns weren't included in any of the models, and the Average column again seemed to be standard / baseline. Having a `condition` of `Good` was slightly more important inside of Seattle, while having a `condition` of `Very_Good` was slightly more important outside of Seattle.

- Specific Recommendation
    - Once again, create a database of example pictures of properties with each condition value, which could be used for the purposes I've previously described.
- renovated
- An extremely important insight was gained for this predictor. It wasn't even included in the Seattle model, and it corresponded to a much smaller increase in value in the entire King County model than in the Outside Seattle model, where it was very important.
- yr\_built
- This was the most difficult predictor to analyze. In the entire King County model, the effect was negative for each dummy column and, generally, was less negative the further back in time the property was built. In the Seattle model, the 1920s to 1940s dummy column wasn't even included, neither was the 1980s to 2000s in the Outside Seattle model. Both the Seattle and Outside Seattle models showed that older built properties seemed to be more valuable in general, although the trend was much clearer Outside Seattle. Just to clarify that there was an overall negative effect as the yr\_built increases in each model, I created a [copy of this notebook in which I did not restructure the yr\\_built column](#) ([https://github.com/sarnadpy32/king\\_county\\_development/blob/master/Phase%202%20-%20Project%20-20yr\\_built%20changed.ipynb](https://github.com/sarnadpy32/king_county_development/blob/master/Phase%202%20-%20Project%20-20yr_built%20changed.ipynb)), which confirmed my hypothesis. I included the yr\_built results from the other notebook below ([sub\\_sect\\_other\\_yr\\_built\\_results](#)).

In [145]:

```

1 kc_log = kc_compare_params_df.log.copy()
2 seattle_log = seattle_compare_params_df.log.copy()
3 out_seattle_log = out_seattle_compare_params_df.log.copy()
4
5 kc_log.name = 'kc'
6 seattle_log.name = 'seattle'
7 out_seattle_log.name = 'out_seattle'
8
9 log_compare_df = pd.concat([kc_log, seattle_log, out_seattle_log], axis=1)
10
11 log_comp_df = log_compare_df.copy()
12
13 log_compare_df = log_compare_df.fillna('-')
```

In [146]:

```
1 log_compare_df.loc['const'] = log_compare_df.loc['const'].apply(lambda x: np.exp(x)).round(2)
```

In [147]:

```
1 log_compare_df.iloc[1:-2] = log_compare_df.iloc[1:-2].applymap(lambda x: '{:.2f}%'.format(x*100) if type(x)
```

In [148]:

```
1 log_compare_df.iloc[-2:] = log_compare_df.iloc[-2:].applymap(lambda x: round(x, 3))
```

In [149]:

```
1 # print(log_compare_df.to_markdown(colalign=['center']*4))
```

## This Jupyter NB's Results

---

	<b>kc</b>	<b>seattle</b>	<b>out_seattle</b>
const	\$262K	\$288K	\$149K
sqft_living	-	0.02%	-
sqft_lot	0.00%	-0.00%	0.00%
sqft_living15	0.02%	0.02%	0.02%
sqft_lot15	-0.00%	-0.00%	-0.00%
floors	7.98%	2.54%	0.63%
bedrooms	-	-2.88%	1.54%
bathrooms	11.02%	4.55%	12.55%
basement	10.23%	-	4.35%
grade_4_Low	-73.30%	-	-
grade_5_Fair	-49.22%	-41.20%	-29.87%
grade_6_Low_Average	-27.92%	-24.59%	-19.98%
grade_7_Average	-	-	-
grade_8_Good	22.32%	19.49%	20.84%
grade_9_Better	46.68%	43.83%	41.19%
grade_10_Very_Good	66.68%	56.54%	60.38%
grade_11_Excellent	84.07%	70.60%	74.48%
grade_12_Luxury	96.51%	118.49%	87.52%
view_FAIR	14.42%	-	21.16%
view_AVERAGE	7.81%	-	7.95%
view_GOOD	9.71%	4.47%	6.68%
view_EXCELLENT	25.78%	21.45%	26.14%
waterfront	30.97%	38.10%	46.78%
condition_Fair	-	-	-
condition_Average	-	-	-
condition_Good	5.17%	7.05%	4.91%
condition_Very_Good	11.36%	10.24%	13.25%
renovated	4.28%	-	16.26%
yr_built_1920_to_1940_s	-5.18%	-	14.54%
yr_built_1940_to_1960_s	-19.28%	-14.83%	25.79%
yr_built_1960_to_1980_s	-39.73%	-32.14%	8.78%
yr_built_1980_to_2000_s	-49.49%	-30.39%	-
yr_built_2000_to_2020_s	-45.60%	-34.17%	2.79%
r_score	0.645	0.698	0.656
r_adj_score	0.645	0.697	0.655

**yr\_built Results from the other Jupyter NB**  
[\(https://github.com/sarnadpy32/king\\_county\\_development/blob/master/Phase%20Project%20-%20yr\\_built%20changed.ipynb\)](https://github.com/sarnadpy32/king_county_development/blob/master/Phase%20Project%20-%20yr_built%20changed.ipynb)

---

<b>kc</b>	<b>seattle</b>	<b>out_seattle</b>
yr_built	-0.58%	-0.44%



**Visualizations of the Coefficients of the Three Log Models**

---

```
In [150]: 1 print(bold_red +'num_cols:' + every_off, num_cols)
2 print(bold_red +'new_cat_order:' + every_off, new_cat_order)

num_cols: ['sqft_living', 'sqft_lot', 'sqft_living15', 'sqft_lot15', 'floors', 'bedrooms', 'bathrooms']
new_cat_order: ['basement', 'grade', 'view', 'waterfront', 'condition', 'renovated', 'yr_built', 'city']
```

```
In [151]: 1 sqft_cols = num_cols[:1] + num_cols[2:3] + num_cols[1:2] + num_cols[3:4]
2 fl_bed_bath_base_cols = num_cols[4:] + ['basement']
3 perc_groups = \
4 [sqft_cols, fl_bed_bath_base_cols, ['grade'], ['view', 'waterfront'], ['condition', 'renovated'], ['yr_buil
```

In [152]:

```

1  for p_i, perc_group in enumerate(perc_groups):
2
3      if 'grade' in perc_group or 'view' in perc_group or 'condition' in perc_group or 'yr_built' in perc_group:
4          r_col = 'grade' if 'grade' in perc_group else 'view' if 'view' in perc_group \
5              else 'condition' if 'condition' in perc_group else 'yr_built'
6
7      plot_df = log_comp_df.loc[log_comp_df.index.map(lambda x: r_col in x or x in perc_group)]
8
9  else: plot_df = log_comp_df.loc[perc_group]
10
11 plot_df = plot_df.dropna(how='all')
12
13 p_min = plot_df.min(axis=1).max()
14 p_max = plot_df.max(axis=1).max()
15 p_idx = plot_df.index
16
17 plot_df = plot_df.reset_index().rename(columns={'index': 'cat_col'})
18
19 c_wrap = 5 if len(plot_df) % 5 == 0 else 4 if len(plot_df) % 4 == 0 else 3
20 sep_x = 0
21
22 g = sns.catplot(col='cat_col', data=plot_df, saturation=.5, kind="bar", ci=None, col_wrap=c_wrap,
23                  palette=[get_lighter_color(c, .09) for c in [(0, .3, 0), (0, .6, 0), (0, .9, 0)]])
24
25 g.fig.dpi = 300
26 g.fig.subplots_adjust(wspace=0)
27
28 for p_col in p_idx:
29     p_ax = g.axes_dict[p_col]
30
31     if p_i==0: p_lim, p_step = .0011, .0001
32     else: p_lim, p_step = 1.5, .25
33
34     p_m = [m for m in np.arange(0, p_lim, p_step) if m >= np.abs(p_min) and m >= p_max]
35     if p_m:
36         p_m = p_m[0]
37
38     if p_m==.25:
39         p_step = .125
40         p_ax.set_ylim(-p_m, p_m)
41
42     else: p_ax.set_ylim(-p_m - p_step, p_m + p_step)
43
44     if p_i!=0:
45         if p_m==.25: ticks = np.arange(0, p_m + .01, p_step)
46         elif p_i==2: ticks = np.arange(0, 1.51, .5)
47         else: ticks = np.arange(0, p_m + p_step + .01, p_step)
48
49         ticks = np.append(np.delete(np.flip(ticks), -1) * -1, ticks)
50         p_ax.yaxis.set_ticks(ticks)
51
52     p_rects = p_ax.findobj(mpatches.Rectangle)[-1]
53     for rect in p_rects:
54         rect.set_zorder(9)
55         r_val, r_wid = rect.get_height(), rect.get_width()
56         r_x = rect.get_xy()[0] + (r_wid / 2)
57
58         annot_kws = dict(textcoords='offset pixels', size=15, ha='center', arrowprops={'arrowstyle': '[->]',
59                                         'family':'serif', 'weight':'bold', 'color=(.12, .39, .12)'})
60
61         if np.isnan(r_val):
62             p_ax.annotate('Not\nIncluded', xy=(r_x, 0), xytext=(0, 9), va='bottom', **annot_kws)
63         else:
64             if r_val > 0: r_low, r_off, r_va = '<.01', 9, 'bottom'
65             if r_val < 0: r_low, r_off, r_va = '>-.01', -9, 'top'
66
67             if round(np.abs(r_val), 4) < .0001: r_text = '$\\mathbf{'+ r_low +'\\%}$'
68             elif r_val==0: r_text = '$\\mathbf{0\\%}$'
69             else: r_text = '$\\mathbf{'+ '{:.2f}'.format(round(r_val, 4) * 100) +'\\%}$'
70
71             p_ax.annotate(r_text, xy=(r_x, r_val), xytext=(0, r_off), va=r_va, **annot_kws)
72
73         if '_' in p_col:
74             p_col_list = p_col.split('_')
75

```

```

76     if p_i==0:
77         fig_tit = 'Square Footage'
78         if '15' not in p_col_list[1]:
79             if 'living' in p_col: p_ax_tit = p_col_list[1].title() + ' Area'
80             else: p_ax_tit = p_col_list[1].title()
81         else:
82             if 'living' in p_col: p_ax_tit = p_col_list[1][:-2].title() + ' Area'
83             else: p_ax_tit = p_col_list[1][:-2].title()
84
85     if p_i!=0 and p_i!=5:
86         fig_tit = p_col_list[0].title()
87         if p_i==2:
88             if len(p_col_list)==4: p_ax_tit = ' - '.join(p_col_list[1:3]) + ' + ' + p_col_list[3]
89             else: p_ax_tit = ' - '.join(p_col_list[1:])
90         if p_i==3: p_ax_tit = p_col_list[1].title()
91         if p_i==4:
92             if p_col=='Good': p_ax_tit = p_col_list[1]
93             else: p_ax_tit = ' '.join(p_col_list[1:])
94
95     if p_i==5: fig_tit, p_ax_tit = 'Year Built', p_col_list[2]+"'s " + ' '.join(p_col_list[3:5]) +
96
97     if '_' not in p_col: p_ax_tit = p_col.title()
98
99     p_ax.axhline(0, color='k', lw=.9)
100    p_ax.yaxis.set_major_formatter(viz_percentage_formatter)
101    plt.setp(p_ax.get_yticklabels(), weight='bold', color=(.12, .39, .12))
102    p_ax.tick_params('x', length=0, labelbottom=False)
103    p_ax.grid(True, 'major', 'y', lw=1.2, alpha=.81)
104    if p_col!=p_idx[0]: p_ax.tick_params('y', length=0)
105
106    if p_col not in ['sqft_living15', 'sqft_lot15']: p_ax.set_title(p_ax_tit, family='serif', color=(.12, .39, .12))
107    else:
108        p_tit = p_ax.set_title(p_ax_tit, family='serif', pad=45, color=(.12, .39, .12))
109
110    renderer = g.fig.canvas.get_renderer()
111    p_txt_x_max, p_txt_x_min = p_tit.get_tightbbox(renderer).max[0], p_tit.get_tightbbox(renderer).min[0]
112    p_txt_x = ((p_txt_x_max - p_txt_x_min) / 2) + p_txt_x_min
113    p_txt_y = p_tit.get_tightbbox(renderer).min[1]
114    p_txt_x, p_txt_y = g.fig.transFigure.inverted().transform((p_txt_x, p_txt_y))
115
116    p_txt = '$\mathbf{15}$ Nearest Neighbors'
117    g.fig.text(p_txt_x, p_txt_y - .03, p_txt, size=18, family='serif', weight='bold', ha='center',
118                color=(.12, .39, .12))
119
120    if p_i in [3, 4]:
121        tit_i, sep_i = 1 if p_i==3 else 0, len(plot_df) - 2
122
123        if p_col==list(g.axes_dict.keys())[tit_i]:
124            fig_tit_x = g.fig.transFigure.inverted().transform(p_ax.get_tightbbox(renderer).max)[0]
125        if p_col==list(g.axes_dict.keys())[sep_i]:
126            sep_x_1 = g.fig.transFigure.inverted().transform(p_ax.get_tightbbox(renderer).max)[0]
127            sep_y = g.fig.transFigure.inverted().transform(p_ax.get_tightbbox(renderer).min)[1] - .0005
128        if p_col==list(g.axes_dict.keys())[sep_i + 1]:
129            sep_x_2 = g.fig.transFigure.inverted().transform(p_ax.get_tightbbox(renderer).min)[0]
130            sep_x = ((sep_x_2 - sep_x_1) / 2) + sep_x_1
131
132    fig_box = g.fig.dpi_scale_trans.transform((g.fig.get_tightbbox(renderer).width, g.fig.get_tightbbox(renderer).height))
133    fig_hlf = g.fig.transFigure.inverted().transform(fig_box)[0] / 2
134
135    leg_txts = ["All King County", "Seattle", "Outside Seattle"]
136    leg_txt_prop = dict(family='serif', weight='bold', size=21)
137    leg_x = fig_hlf
138
139    g.fig.legend(p_rects, leg_txts, loc=8, bbox_to_anchor=(leg_x, -.03), ncol=3, prop=leg_txt_prop,
140                  labelcolor=(.12, .39, .12), frameon=False)
141
142    if p_i!=1:
143        sup_x = fig_hlf
144        if p_i==0: sup_x, sup_y = sup_x + .03, 1.11
145        elif p_i==2: sup_y = 1.02
146        elif p_i in [3, 4]: sup_x = fig_tit_x
147        else: sup_y = 1.05
148
149        g.fig.suptitle(r''+ fig_tit +'', x=sup_x, y=sup_y, size=30, family='serif', weight='bold', va='bottom',
150                      color=(.12, .39, .12))
151

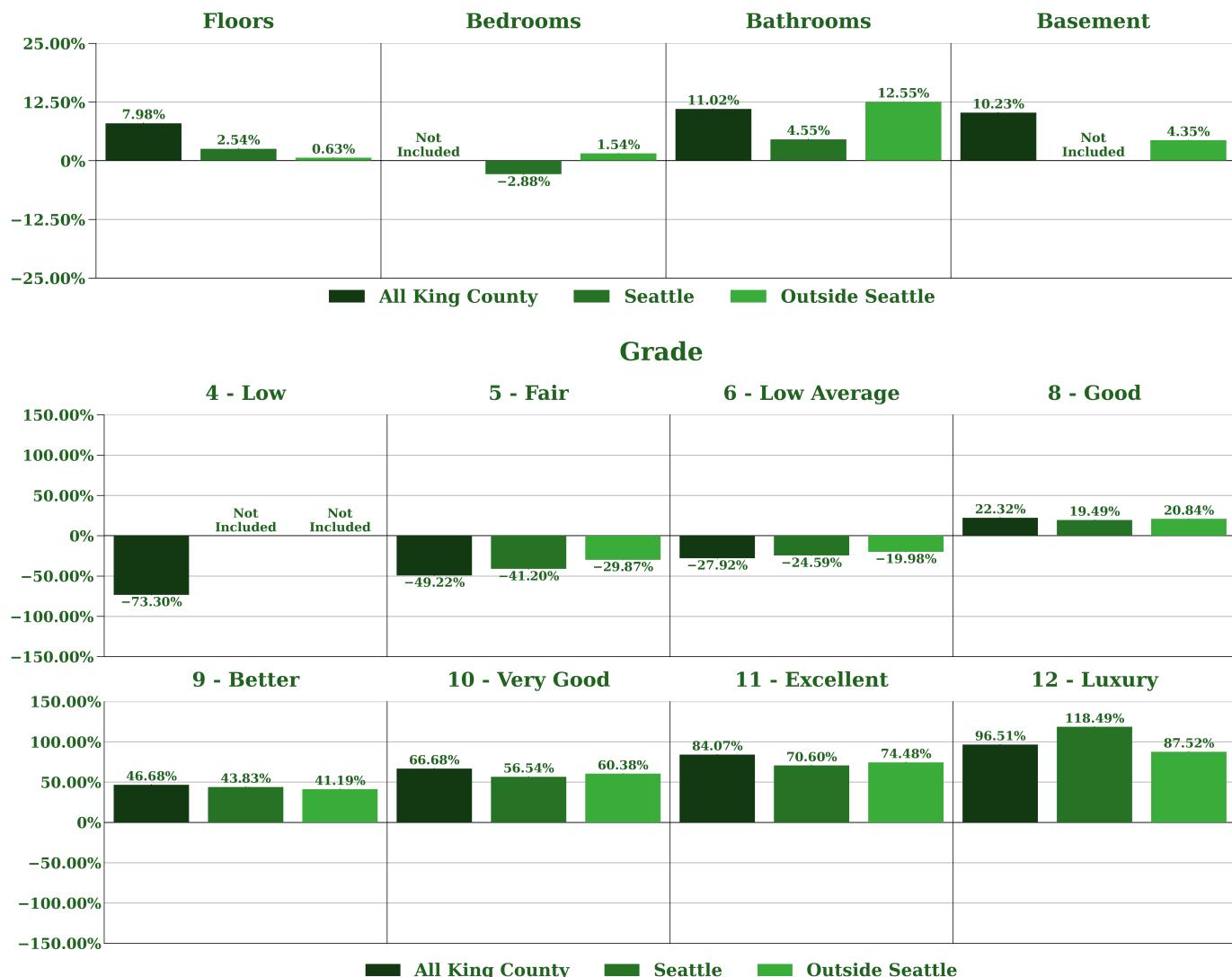
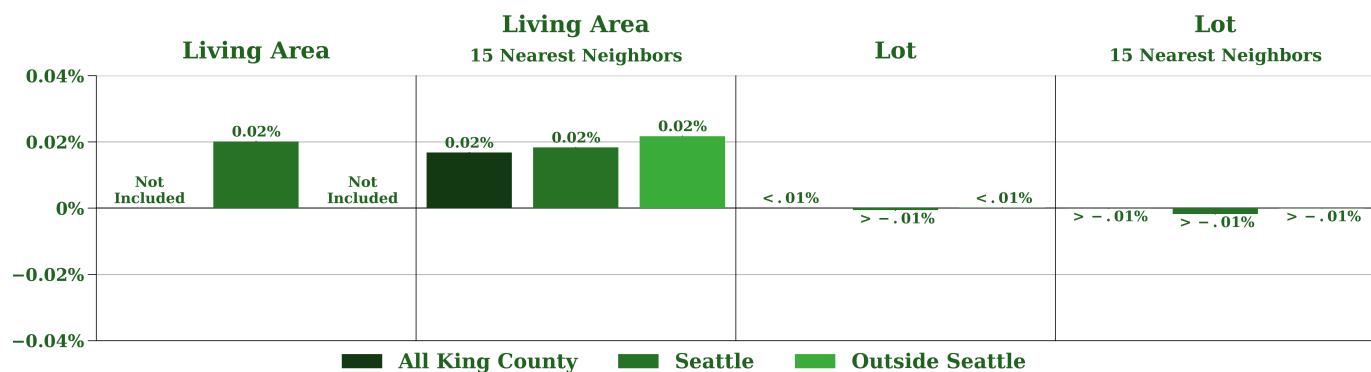
```

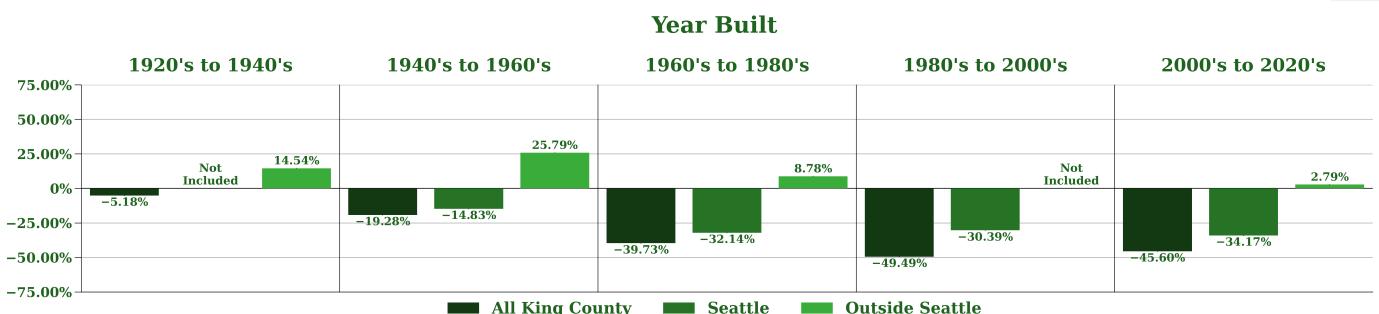
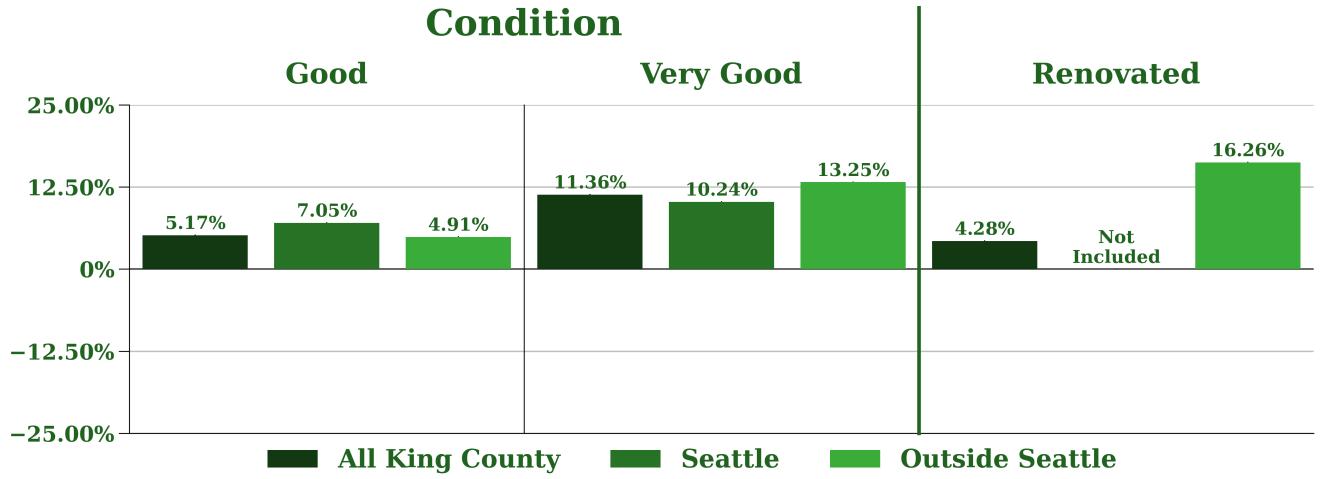
```

152 fig_box = g.fig.dpi_scale_trans.transform((g.fig.get_tightbbox(renderer).width, g.fig.get_tightbbox(renderer).height))
153 fig_hght = g.fig.transFigure.inverted().transform(fig_box)[1]
154 if p_i in [3, 4]:
155     g.fig.add_artist(mlines.Line2D([sep_x, sep_x], [sep_y, fig_hght], lw=3, color=(.12, .39, .12)))
156
157 g.fig.savefig('visuals/presentation_pic_'+ str(p_i + 1), bbox_inches='tight')

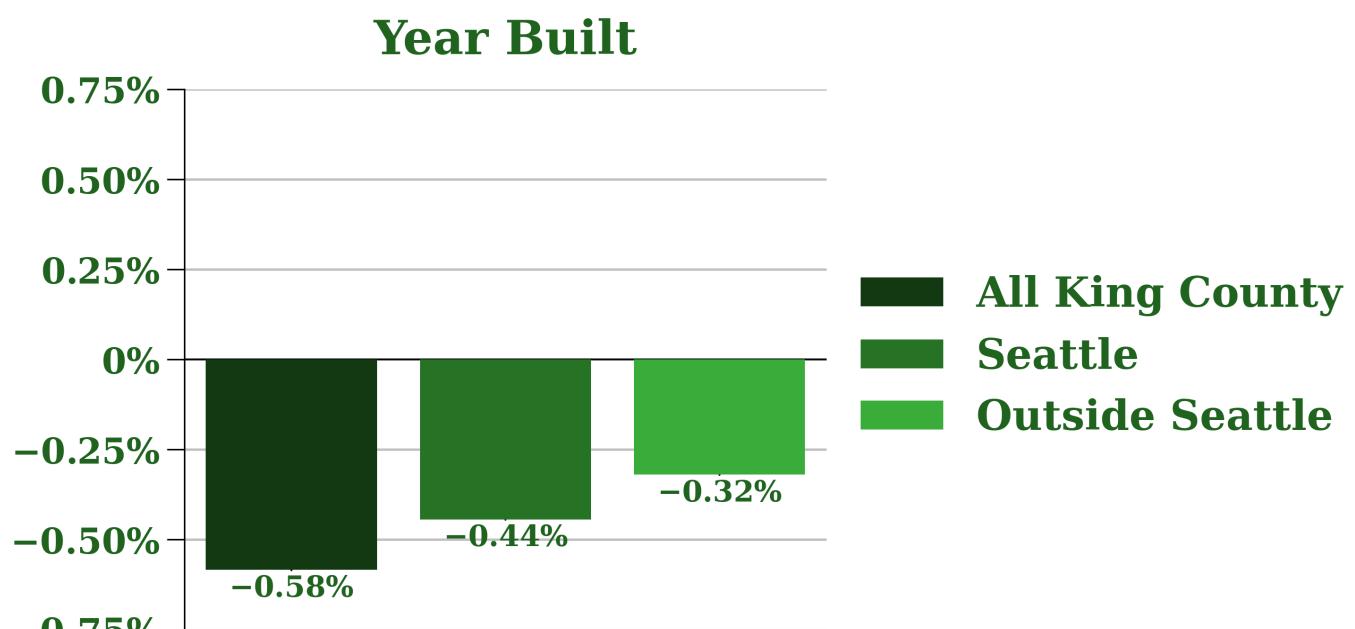
```

## Square Footage





**yr\_built Results from the other Jupyter NB**  
[\(\[https://github.com/sarnadpy32/king\\\_county\\\_development/blob/master/Phase%20Project%20-%20yr\\\_builtinchanged.ipynb\]\(https://github.com/sarnadpy32/king\_county\_development/blob/master/Phase%20Project%20-%20yr\_builtinchanged.ipynb\)\)](https://github.com/sarnadpy32/king_county_development/blob/master/Phase%20Project%20-%20yr_builtinchanged.ipynb)



## Future Investigations

Besides the picture databases I mentioned regarding the `grade`, `view`, and `condition` features, there is a lot more work that King County Development will need to perform if they want to be profitable, efficient in their spending, and if they want to provide the highest quality services to their clients and / or investors. There is more valuable information that can no doubt be gleaned from the dataset that was provided to me for this project, including testing interactions and / or building polynomial regression models, but the dataset only

covered one year's worth of sales, so any further investigation would still be limited. A much larger dataset, possibly with even more predictors to build models with, can undoubtedly be obtained ([https://kingcountyexec.govqa.us/WEBAPP/\\_rs/](https://kingcountyexec.govqa.us/WEBAPP/_rs/)). At that point there should be enough data to build separate models for each city in King County, and to also analyze the various ZIP codes within each city. They should then develop maps of each city and / or ZIP code, separated with the appropriate zoning laws, to see what is even possible in every inch of King County. Areas of high potential should be identified, and the owners of any potentially lucrative properties should be approached to gauge their interest in either their property being acquired, or if they are at least interested in any renovation or remodeling services. If not, King County Development will at least be fully prepared should any enticing properties become available, or if they are approached for their other services. The image below is from a video (<https://www.youtube.com/watch?v=ZeRd3aurWz8>) from the Real Estate for Noobs (<https://www.youtube.com/c/RealEstateforNoobs>) YouTube channel, and even that is only a piece of the puzzle when it comes to real estate development.

