Final Project Submission

- Student name: Kevin Spring
- Student pace: Flex
- Scheduled project review date/time: August 8, 2022 13:30 - 14:15 (CDT)
- Instructor name: Morgan Jones
- Blog post URL: https://medium.com/@kevinjspring/predicting-home-sale-prices-using-linear-regression-4ebf079a48e8

Summary

- Our client wants to be able to predict sales price, identify where to market in King County, WA, and how customers can improve their home to increase sale price.
- Ordinary least squares linear regression was used to create three models.
- The three models were compared using R^2, Prediction Intervals (PI), and Root Mean Squared Error (RMSE).
- Model 2 (M2) is the best model as it has the best predictive capabilities, R-squared of 0.88, low RMSE and PI, though the error is not normally distributed.

Actionable Recommendations

1. Improving the condition of a house by one-unit will increase the sale price by about 6%.
2. Adding an additional full bathroom would increase the sale price of a house by about 3.9%.
3. Marketing should be focused throughout King County, WA except in Zip codes 98002, 98003, 98023, 98032, 98042, and 98198 as the value of the homes sold in these Zip codes are well under the rest of King County, WA.
4. Market real estate services toward owners of waterfront properties as these sell for 59% more than homes not waterfront.

Table of Contents

Business Problem

Our client is a residential real estate broker in King County, WA interested in finding a solution for their customers. Many of their customers come to them needing to sell their home but are unsure of the market value. The client wants us to design and implement a model where they can take in the features of a seller's home and determine which price to begin listing discussion.

Stakeholders

- President of Bon Jovi Real Estate Advisors
- Bon Jovi real estate agents

Background

Our client wants us to predict a continuous value, sales price, from features of the house their customer gives them in the form of continuous and catagorical data. Regression analysis is a statistical process to estimate the relationship between a dependent variable (response) and a continuous independent variables (predictors).

In this analysis I will use ordinary least squares (OLS) linear regresion to assess the relationship between features of homes and sale price. OLS fits a linear model on data by minimizing the sum of the squared difference between the observed dependent variable and the predicted response ($\hat{y}$)

$$\text{To calculate } \hat{y},$$

$$\hat{y} = \hat{\beta}_0 + \sum_{i=1}^{n} x_n \hat{\beta}_n$$

where n is the number of predictors, β_0 is the intercept, $\hat{x}_n$ is the n^{th} predictor, and $\hat{y}$ are the predicted value associated with the dependent variables.

The linear equation that is returned can be used to predict the response value using new data.

To perform OLS linear regression the data needs to be clean with no missing values and catagorical data needs to be coded correctly. The assumptions of OLS linear regression are then checked and models are built. These models are compared using, coefficient of determination, prediction intervals, and Root Mean Squared Error to compare and determine wich model is the most suited for our client.

In [1]:
```
# Import libraries
from datetime import date

## Data analysis
import pandas as pd
import numpy as np

## Statistical analysis
from scipy import stats
from scipy.stats import norm
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols

## Model Validation
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

## Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

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

Data

Description

The data is a collection of single family homes in the King County, WA area sold between May 2014 and May 2015 (1). The data contains 21 variables and 21,597 records. This data will be suitable to create a model to predict sale price for homes within the paramaters of this dataset.

In [2]:
```
df.shape
```

Out[2]:
```
(21597, 21)
```

In [3]:
```
#inspect data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  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: 3.5+ MB
```

Table 1 Variable Names and Descriptions for King County Data Set

See the King County Assessor Website for further explanation of each condition code

Variable	Data Type	Description
`id`	catagorical	Unique identifier for a house
`date`	continuous	Date house was sold
`price`	continuous	Sale price (prediction target)
`bedrooms`	discrete	Number of bedrooms
`bathrooms`	discrete	Number of bathrooms
`sqft_living`	continuous	Square footage of living space in the home
`sqft_lot`	continuous	Square footage of the lot
`floors` -	discrete	Number of floors (levels) in house
`waterfront`	ordinal	Whether the house is on a waterfront
`view`	ordinal	Quality of view from house
`condition`	ordinal	How good the overall condition of the house is. Related to maintenance of house

Variable	Data Type	Description
`grade`	ordinal	Overall grade of the house. Related to the construction and design of the house
`sqft_above`	continuous	Square footage of house apart from basement
`sqft_basement`	continuous	Square footage of the basement
`yr_built`	catagorical	Year when house was built
`yr_renovated`	catagorical	Year when house was renovated
`zipcode`	catagorical	ZIP Code used by the United States Postal Service
`lat`	catagorical	Latitude coordinate
`long`	catagorical	Longitude coordinate
`sqft_living15`	continuous	The square footage of interior housing living space for the nearest 15 neighbors
`sqft_lot15`	continuous	The square footage of the land lots of the nearest 15 neighbors

Location of King County, WA home sales

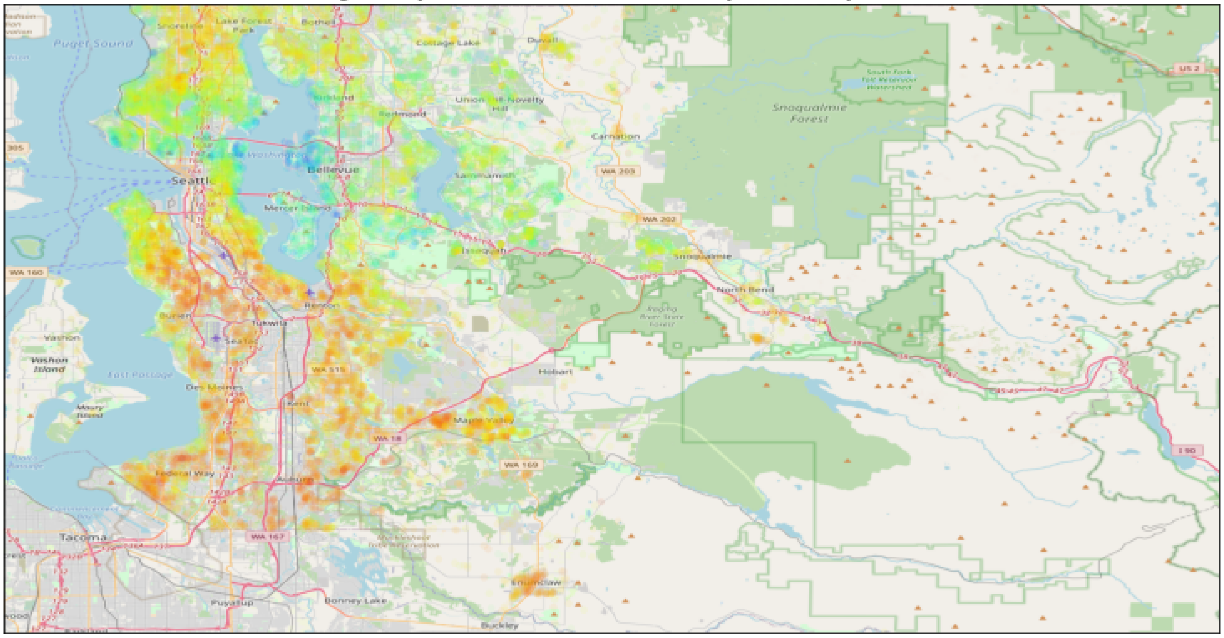
In [4]:

```
## Map of home sales between May 2014 and May 2015
# code adapted from
# Ahmed Qassim,
# https://towardsdatascience.com/easy-steps-to-plot-geographic-data-on-a-map-python-11

# Define bounding box
BBox = ((df.long.min(), df.long.max(),
         df.lat.min(), df.lat.max() ))

# Make scatterplot
fig, ax  = plt.subplots(figsize = (13,12))
ax.scatter(df.long, df.lat, c = np.log(df.price), alpha=.075,
           s=20, edgecolors='none',
           cmap= plt.cm.get_cmap('jet_r'))
# Plot paramaters
ax.set_title('King County, WA home sales between May 2014 - May 2015') # title
# Remove x, y ticks and labels
ax.tick_params(axis='both', which='both',
               bottom=False, top=False, left=False, right=False,
               labelbottom=False, labeltop=False, labelleft=False, labelright=False)

# Set x and y-axis limits to bounding box
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])

# Set area map
ruh_m=plt.imread('img/King_County_map.png')
ax.imshow(ruh_m, zorder=0, extent = BBox, aspect= 'equal') #

# plt.savefig('img/KC_home_sale_map.png', dpi=600) # save the image
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x200c7b8a820>
```

King County, WA home sales between May 2014 - May 2015



Data Limitations

- Data is only from 2014 to 2015. Models to predict future sales price would need to be updated with newer data.
- Some data might be missing, such as for-sale-by-owner or owner-financed sales.
- Ordinal data might be highly variable based on examinter's subjective experience.
- As the map shows, home sales are a mix of urban and rural houses, but much more homes are clustered together. The models may not be able to accurately predict rural house prices because of the lack of data for rural homes.

Data Cleanup

Identify and remove duplicated records

In [5]:
```
# Any dulplicated homes?
duplicates_len = len(df[df.duplicated(subset=['id'],
                                      keep=False)].sort_values(by='id'))

print(f"Results:\nThere are {duplicates_len} duplicated records.")
df[df.duplicated(subset=['id'], keep=False)].sort_values(by='id').head(4)
```

```
Results:
There are 353 duplicated records.
```

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
2495	1000102	4/22/2015	300000.0	6	3.0	2400	9373	2.0	NC
2494	1000102	9/16/2014	280000.0	6	3.0	2400	9373	2.0	NaN
16800	7200179	10/16/2014	150000.0	2	1.0	840	12750	1.0	NC
16801	7200179	4/24/2015	175000.0	2	1.0	840	12750	1.0	NC

4 rows × 21 columns

Duplicate home ID discussion

The duplicated records based on ID are from the same homes that sold within the same year. These homes have the same attributes except for sale `date`. These may be homes that were flipped or sold quickly after an initial sale. I will keep these records as I am interested in predicting a home's sale price and these give more data for the true value of a house.

Remove Unnecessary variables

The following variables will be deleted from this analysis as they are unnecessary to my analysis.

- `id` - This is an unique identifier for each home. Too unique.
- `date` - This is the sale date and time will not be analyzed due to the single year of the data.
- `lat` - This is the latitude of the home sold. Will use Zip code for location.
- `long` - same reasoning as `lat`

In [6]:

```
# delete unnecessary columns
df.drop(['id','date', 'lat', 'long'], axis=1, inplace=True)
```

Identify Missing data

In [7]:

```
# How many columns have NaN?
print(df.isna().sum())
```

```
price                0
bedrooms             0
bathrooms            0
sqft_living          0
sqft_lot             0
floors               0
waterfront        2376
view                63
condition            0
grade                0
sqft_above           0
sqft_basement        0
yr_built             0
yr_renovated      3842
zipcode              0
sqft_living15        0
sqft_lot15           0
dtype: int64
```

In [8]:

```
# Any placeholders?
# Look for top occuring values
print('King County, WA \n Home Sales Dataframe\n')
for col in df.columns:
    print(col, '\n', df[col].value_counts(normalize = True).head(10), '\n')
```

```
King County, WA
 Home Sales Dataframe

price
 450000.0    0.007964
350000.0    0.007964
550000.0    0.007362
500000.0    0.007038
425000.0    0.006945
325000.0    0.006853
400000.0    0.006714
375000.0    0.006390
300000.0    0.006158
525000.0    0.006066
Name: price, dtype: float64

bedrooms
 3     0.454878
4     0.318655
2     0.127796
5     0.074131
6     0.012594
1     0.009075
7     0.001760
8     0.000602
9     0.000278
10    0.000139
Name: bedrooms, dtype: float64

bathrooms
 2.50    0.248970
1.00    0.178312
1.75    0.141131
2.25    0.094782
2.00    0.089364
1.50    0.066907
2.75    0.054869
3.00    0.034866
3.50    0.033847
3.25    0.027272
Name: bathrooms, dtype: float64

sqft_living
 1300    0.006390
1400    0.006251
1440    0.006158
1800    0.005973
1660    0.005973
1010    0.005973
1820    0.005927
1480    0.005788
1720    0.005788
1540    0.005742
Name: sqft_living, dtype: float64

sqft_lot
 5000    0.016576
6000    0.013428
4000    0.011622
7200    0.010187
```

```
4800    0.005510
7500    0.005510
4500    0.005279
8400    0.005140
9600    0.005047
3600    0.004769
Name: sqft_lot, dtype: float64

floors
 1.0    0.494189
2.0    0.381303
1.5    0.088438
3.0    0.028291
2.5    0.007455
3.5    0.000324
Name: floors, dtype: float64

waterfront
 NO     0.992404
YES    0.007596
Name: waterfront, dtype: float64

view
 NONE         0.901923
AVERAGE      0.044441
GOOD         0.023591
FAIR         0.015325
EXCELLENT    0.014721
Name: view, dtype: float64

condition
 Average      0.649164
Good         0.262861
Very Good    0.078761
Fair         0.007871
Poor         0.001343
Name: condition, dtype: float64

grade
 7 Average        0.415521
8 Good           0.280826
9 Better         0.121082
6 Low Average    0.094365
10 Very Good     0.052507
11 Excellent     0.018475
5 Fair           0.011205
12 Luxury        0.004121
4 Low            0.001250
13 Mansion       0.000602
Name: grade, dtype: float64

sqft_above
 1300    0.009816
1010    0.009724
1200    0.009538
1220    0.008890
1140    0.008520
1400    0.008334
1060    0.008242
1180    0.008196
```

```
1340    0.008149
1250    0.008057
Name: sqft_above, dtype: float64

sqft_basement
 0.0       0.593879
?         0.021021
600.0     0.010048
500.0     0.009677
700.0     0.009631
800.0     0.009307
400.0     0.008520
1000.0    0.006853
900.0     0.006575
300.0     0.006575
Name: sqft_basement, dtype: float64

yr_built
 2014    0.025883
2006    0.020975
2005    0.020836
2004    0.020049
2003    0.019447
2007    0.019308
1977    0.019308
1978    0.017919
1968    0.017641
2008    0.016993
Name: yr_built, dtype: float64

yr_renovated
 0.0       0.958096
2014.0    0.004112
2013.0    0.001746
2003.0    0.001746
2007.0    0.001690
2000.0    0.001633
2005.0    0.001633
2004.0    0.001239
1990.0    0.001239
2009.0    0.001183
Name: yr_renovated, dtype: float64

zipcode
 98103    0.027874
98038    0.027272
98115    0.026994
98052    0.026578
98117    0.025605
98042    0.025328
98034    0.025235
98118    0.023475
98023    0.023105
98006    0.023059
Name: zipcode, dtype: float64

sqft_living15
 1540    0.009122
1440    0.009029
1560    0.008890
```

```
1500    0.008334
1460    0.007825
1580    0.007733
1610    0.007686
1720    0.007686
1800    0.007686
1620    0.007594
Name: sqft_living15, dtype: float64

sqft_lot15
 5000    0.019771
4000    0.016484
6000    0.013335
7200    0.009724
4800    0.006714
7500    0.006575
8400    0.005371
3600    0.005140
4500    0.005140
5100    0.005047
Name: sqft_lot15, dtype: float64
```

Missing value results

- `NaN`
  - `waterfront`
    - Binary categorical variable (`YES` or `NO`)
    - replace `NaN` with mode of `NO` as most likely these properties are not waterfront
  - `view`
    - Ordinal categorical variable
    - replace `NaN` with `NONE`
  - `yr_renovated`
    - Will be converted to a countable numerical variable
    - `0` is the most common value with over 95% of values.
    - Replace `NaN` with 0 value
- Placeholder
  - `yr_renovated` has `0` for missing or unknown values.
  - `sqft_basement` has `?` for missing or unknown values.

In [9]:

```
# replacing waterfront NaN with 'NO'
df['waterfront'].fillna('NO', inplace=True)

# replace yr_renovated NaN with 'Unknown'
df['yr_renovated'].fillna(0, inplace=True)

# replace `Nan` with `NONE` for column `view`
df['view'].fillna('NONE', inplace=True)
```

In [10]:

```
# Confirm no more NaN values
print(df.isna().sum())
```

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

Table 2: Coding ordinal, binary, and count data

variable	Data Type	Plan
`condition`	ordinal	Recode to dictionary. `{'Poor': 0, 'Fair': 1, 'Average': 2, 'Good': 3, 'Very Good': 4}`
`grade`	ordinal	Delete the descriptor, keep the number, and convert it to `int` datatype. Example: `7 Average` becomes `7`
`basement`	binary	If there is a basement (sq.ft > 0) the value will be set to `1`. No basement (sq.ft = 0) set to `0`. `?` makes up about 2% of values and the current value of `0` makes up almost 60%. Replace `?` with the mode of `0`.
`view`	oridinal	Recode to dictionary. `{'NONE': 0, 'FAIR': 1, 'AVERAGE': 2, 'GOOD': 3, 'EXCELLENT': 4}`
`waterfront`	binary	Recode to dictionary. `{NO': 0, 'YES': 1}`.
`home_age`	discrete	Create variable from `yr_built`. Subtract current year from `yr_built`. Drop `yr_built`
`yr_since_reno`	discrete	Create variable from `yr_renovated`. Subtract current year from `yr_renovated`. `0` is the most common value with over 95% of values. If never renovated then subtract from `yr_built`. Drop `yr_renovated`.

In [11]:
```
# --------------------------------------------------------------------------#
# Encoding ordinal, binary, and count variables

# Code condition to ordinal data
```

```python
# Map condition variable to dictionary
condition_dict = {'Poor': 0, 'Fair': 1, 'Average': 2,   # Map
                  'Good': 3, 'Very Good': 4}
df['condition'] = df['condition'].map(condition_dict)   # Use map to
                                                        # code values

# Code Grade to ordinal data
# Strip out by spaces and keep the first string, which is the value
df['grade'] = df['grade'].apply(lambda x: x.split(' ', 1)[0]).astype(int)

# Code sqft_basement to binary data
# sqft_basement has '?' as a placeholder. Set this to 0.
df['sqft_basement'].replace('?', 0, inplace=True)
# change to numerical type
df['sqft_basement'] = df['sqft_basement'].astype(float)
# With a basement then code as 1
df['sqft_basement'].loc[df['sqft_basement'] > 0] = 1
# rename column
df.rename(columns={'sqft_basement': 'basement'}, inplace=True)

# Code view to ordinal data
# Map ordinal variable to dictionary
view_dict = {'NONE': 0,
             'FAIR': 1,
             'AVERAGE': 2,
             'GOOD': 3,
             'EXCELLENT': 4} # map
df['view'] = df['view'].map(view_dict) # Recode

# Recode waterfront to binary data
# Map binary variable to dictionary
waterfront_dict = {'NO': 0, 'YES': 1} # map
df['waterfront'] = df['waterfront'].map(waterfront_dict) # Recode

# Recode home_age to discrete data
# Calculate home age
current_year = date.today().year # assign current year
df['home_age'] = current_year - df['yr_built'] # Calculate year since built
df.drop('yr_built', axis=1, inplace=True) # drop old column

# Recode yr_since_reno to discrete data
# subtraction function
def sub(a, b):
    return a - b
# Calculate years since last renovation
df['yr_since_reno'] = df.apply(
    lambda row : sub(current_year, row['yr_renovated']) # subtract
    if row['yr_renovated'] > 0 # if the property has been renovated
    else row['home_age'], axis = 1) # else the property has not been renovated
df.drop('yr_renovated', axis=1, inplace=True)
```

```
D:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Outliers

In [12]:
```
# Data description to identify outliers
df.describe().apply(lambda s: s.apply(lambda x: format(x, 'g')))
```

Out[12]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	cond
count	21597	21597	21597	21597	21597	21597	21597	21597	2
mean	540297	3.3732	2.11583	2080.32	15099.4	1.4941	0.0067602	0.233181	2.4
std	367368	0.926299	0.768984	918.106	41412.6	0.539683	0.0819439	0.764673	0.6
min	78000	1	0.5	370	520	1	0	0	
25%	322000	3	1.75	1430	5040	1	0	0	
50%	450000	3	2.25	1910	7618	1.5	0	0	
75%	645000	4	2.5	2550	10685	2	0	0	
max	7.7e+06	33	8	13540	1.65136e+06	3.5	1	4	

There is an outlier that may be due to a data entry mistake. One house has 33 bedrooms. I was expecting it to be a mansion but it has an average grade (7), 1.75 bathrooms, and only 1,620 square feet of living space. I think this house had a miskey and the number of bedrooms should be 3.

In [13]:
```
df[df['bedrooms'] > 30]
```

Out[13]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gr
15856	640000.0	33	1.75	1620	6000	1.0	0	0	4	

In [14]:
```
df.at[15856, 'bedrooms'] = 3
df.loc[[15856]]
```

Out[14]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	gr
15856	640000.0	3	1.75	1620	6000	1.0	0	0	4	

Exploratory Data Analysis

Cleaned Data Description

In [15]:
```
df.describe().apply(lambda s: s.apply(lambda x: format(x, 'g')))
```

Out[15]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
count	21597	21597	21597	21597	21597	21597	21597	21597	
mean	540297	3.37181	2.11583	2080.32	15099.4	1.4941	0.0067602	0.233181	2.
std	367368	0.904096	0.768984	918.106	41412.6	0.539683	0.0819439	0.764673	0.6
min	78000	1	0.5	370	520	1	0	0	
25%	322000	3	1.75	1430	5040	1	0	0	
50%	450000	3	2.25	1910	7618	1.5	0	0	
75%	645000	4	2.5	2550	10685	2	0	0	
max	7.7e+06	11	8	13540	1.65136e+06	3.5	1	4	

About the data

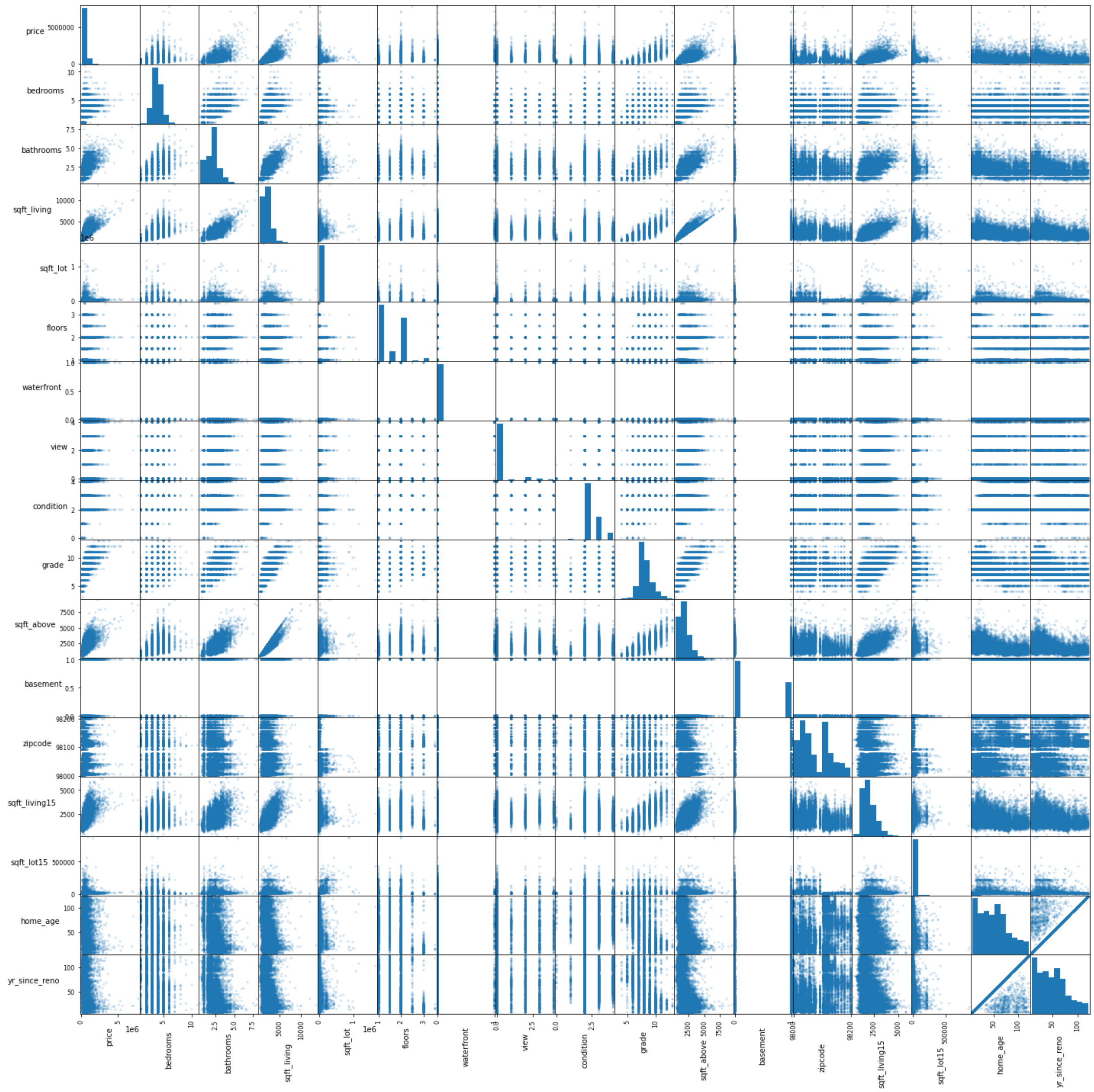
The median house sold in King County, WA between 2014 to 2015 was for $450,000. The median house sold was 1910 square feet, 3 bedroom, 2.25 bathrooms, and 47 years old. The home sale price range was \$78,000 to $7,700,000.

Variable scatter matrix

A scatter matrix will return both histograms and scatterplots lined up for each variable. The diaganal represents the histograms of that variable to visualize the distribution and the rest are scatterplots to visualize the relationships between variables.

In [16]:

```
# Visualize the data using scatter plot and histogram

import warnings
warnings.filterwarnings('ignore') # Ignore warnings

# Create scatter matrix
axes = pd.plotting.scatter_matrix(df, alpha = 0.2, figsize  = [20, 20])
for ax in axes.flatten():
    ax.xaxis.label.set_rotation(90)
    ax.yaxis.label.set_rotation(0)
    ax.yaxis.label.set_ha('right')

plt.tight_layout()
plt.gcf().subplots_adjust(wspace=0, hspace=0)
plt.show()
```



Scatter Matrix Results

Histogram

The diaganol plots are the histogram and indicate that most of the variables are right-skewed, including the dependent variable, price.

Scatterplot

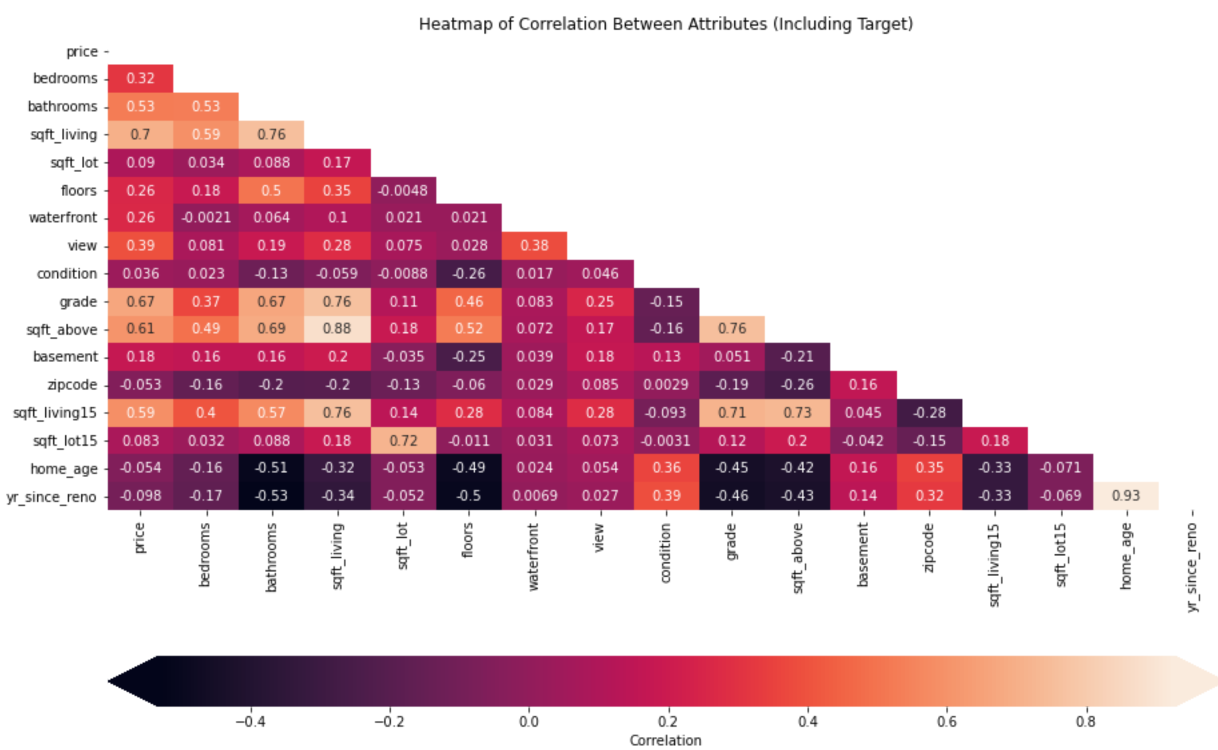
Looking at the first row, the variables with the strongest positive coorelation with price are for the number of bathrooms, grade, and square footage of the living space in the house.

Correlation Matrix Heatmap

A correlation heatmap calculates the Pearson correlation between variables. A Pearson correlation measures the relationship between two variables. A value of 1 means a complete positive correlation, a value of 0 means no correlation, and -1 means a negative correlation exists. This heatmap has a dark color for negatively correlated variables and a light color for positively correlated variables.

In [17]:

```
# Make heatmap
# Code adapted from Flatiron Data Science

# compute the correlation matrix
corr = df.corr()

# Set up figure and axes
fig, ax = plt.subplots(figsize=(15, 10))

# Plot a heatmap of the correlation matrix, with both
# numbers and colors indicating the correlations
sns.heatmap(
    # Specifies the data to be plotted
    data=corr,
    # The mask means we only show half the values,
    # instead of showing duplicates. It's optional.
    mask=np.triu(np.ones_like(corr, dtype=bool)),
    # Specifies that we should use the existing axes
    ax=ax,
    # Specifies that we want labels, not just colors
    annot=True,
    # Customizes colorbar appearance
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2, "extend'
)

# Customize the plot appearance
ax.set_title("Heatmap of Correlation Between Attributes (Including Target)");
```



Dummy variables for Zip code

Dummy variables or One-hot-encoding is a way to use catagorical variables with regression analysis. The variable `zipcode` is a catagorical variable and must be converted to a numerical data. Each Zip code will become its own variable and be either a 'no' (0) or 'yes' (1).

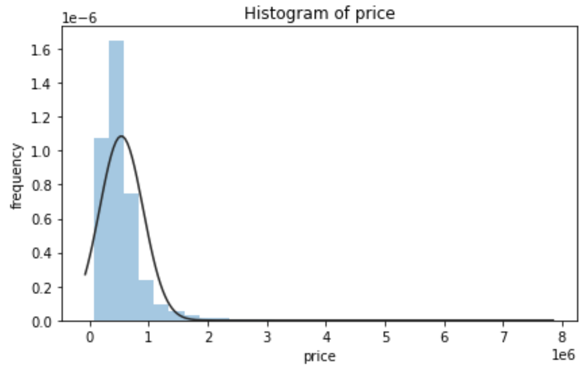
In [18]:
```
# Convert zip code variable to Dummy variables
df_clean = df.copy()
cat_col = ['zipcode']

# label columns as category
df[cat_col] = df[cat_col].astype('category')
ohe_df = pd.get_dummies(df[cat_col], drop_first=True)

# merge ohe_df with df_clean and drop old zip_code column
df_clean = pd.concat([df_clean, ohe_df], axis=1)
df_clean.drop(cat_col, axis=1, inplace=True)
```

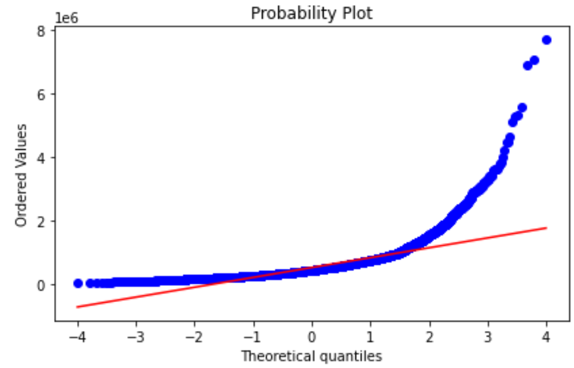
Distirubtion of `price`

In [19]:
```
def hist_plot(data, Y):
    '''
    Histogram plot function
    Input:
          data: pandas dataframe
          Y: column of variable for the histogram
    Output:
          Histogram plot
          Skewness and kurtosis value
    Citation:
         Atanu Dan
             https://medium.com/@atanudan/kurtosis-skew-function-in-pandas-aa63d72e20c
'''
    y = data[Y]
    # Plot code
    fig, ax = plt.subplots(1,2, figsize=(15,4))
    sns.distplot(y, fit=norm, bins=30, kde=False, ax=ax[0]);
    ax[0].title.set_text(f'Histogram of {Y}')
    ax[0].set(xlabel=f'{Y}', ylabel='frequency')
    res = stats.probplot(df_clean['price'], plot=ax[1])

    ## Skewness and Kurtosis
    print(f'EDA of {Y} variable')
    print(f'Skewness: {y.skew()}')
    print(f'Kurtosis: {y.kurt()}')
```

In [20]:
```
#histogram of price
hist_plot(df_clean, 'price')
```

```
EDA of price variable
Skewness: 4.023364652271239
Kurtosis: 34.54135857673376
```





Interpretation

The data in the variable `price` is highly right-skewed and does not follow a normal distribution as shown in the histogram and QQ-plot. This may result in a high level of **heteroskedasticity** because there are many orders of magnitude between the lowest and highest sale price. Heteroskedasticity results when variance is not equal across the range of the dependent variable. This may cause higher variance in high sale price houses in contrast to low sale price houses. In other words, the variance is unequal as it is changing porportionally with the variable.

Model Specification

In this section I will specify which variable to include in the model to predict sales price.

Model	Description
M1	Use variables highly correlated (>0.6) with sale price
M2	A backward stepwise regression to choose variables. This procedure will result in almost all independent variables, excluding some the dummy variables for `zipcode` , being chosen.
M3	M1 with interaction effects

Table 3: OLS Model Assumptions (2)

Assumption	Description
1	The regression model is linear in the coefficients and error term
2	There is a random sampling of observations
3	Error term has a population mean of zero
4	There is no multi-collinearity (or perfect collinearity)
5	The error term has a constant variance (no heteroskedasticity)
6	The error term is normally distributed. This allows statistical hypothesis testing to be done.

Assumption 2 is met with the collection of the data. The data may lack private sales by owner but the majority of house sales occur through real estate agents. Assumptions 1, 3, 4, 5 and 6 will be checked with plots.

Specifing Model 1 (M1)

I will specify M1 variables using the independent variables with a Pearson's correlation with `price` of 0.6 or greater.

In [21]:
```
# Features correlated
# Source doe from Flatiron
features = []
correlations = []
for idx, correlation in corr['price'].T.iteritems():
    if correlation >= .6 and idx != 'price':
        features.append(idx)
        correlations.append(correlation)
corr_with_price = pd.DataFrame({'Correlations':correlations, 'Features': features})

print('Table 4: Independent Variables Highly Correlated (>0.6) With Price')
display(corr_with_price)
```

```
Table 4: Independent Variables Highly Correlated (>0.6) With Price
```

	Correlations	Features
0	0.701917	sqft_living
1	0.667951	grade
2	0.605368	sqft_above

Specifying Model 2 (M2)

M2 uses an automated stepwise backward elemination feature selection strategy. All the variables are fed into the model and fitted. The independent variable with highest p-value is removed if the p-value is greater than 0.05. A p-value greater than 0.05 indicates that variable's effect is not statistically significant. This is repeated until all the p-values of the predictor variables are less than 0.05, meaning they are significant.

In [22]:
```
#Backward Elimination function
def backward_elimination(df, y):
    '''
    Backward Elimination
    Feed all the possible features to the model at first. We check
    the performance of the model and then iteratively remove the worst
    performing features one by one till the overall performance of the
    model comes in acceptable range. The performance metric used here to
    evaluate feature performance is p-value. If the pvalue is above 0.05
    then we remove the feature, else we keep it.

    Input:
          df = Pandas dataframe of your data
```

```
        y = dependent variable
    Output:
        list of independent variables in the model

    Citation:
        Abhini Shetye
            https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b
    '''
    X = df.copy()
    X.drop(y.columns, axis=1, inplace=True)
    cols = list(X.columns) # get all of the column names
    pmax = 1
    while (len(cols)>0): # while there are entries in the cols list
        p= [] # initialize p-value list
        X_1 = X[cols] # create new dataframe
        X_1 = sm.add_constant(X_1) # add constant for y-intercept
        model = sm.OLS(y,X_1).fit() # OLS regression model on the data
        p = pd.Series(model.pvalues.values[1:],index = cols) # save the p-values
        pmax = max(p) # assign maximum p-value
        feature_with_p_max = p.idxmax() # get that p-value's index
        if(pmax > 0.05): # check if the max p-value is greater than 5
            cols.remove(feature_with_p_max) # if it is then remove it from the cols li
        else:
            break # otherwise all p-values are less than 0.05
    return cols # return the features
```

Specifying Model 3 (M3)

Interaction effects occur when the effect of one predictor variable depends on the value of another variable. For example, condition of a home may be dependent on the age of the home. There may be a dependency between the view and whether the house is on waterfront property. M3 builds on M1 by adding interaction effects to the main effects. It also removes `sqft_above` as a predictor variable due to high correlation between `sqft_above` and `sqft_living`.

The graphs below are interaction plots. This plot displays the fitted values of the dependent variable (`price`) on the y-axis and one of the predictor variables on the x-axis. The lines between the points represents the other predictor variable. If the lines remain parallel to each other then there is not an interaction between these variables. If the lines cross that indicates there could be an interaction.

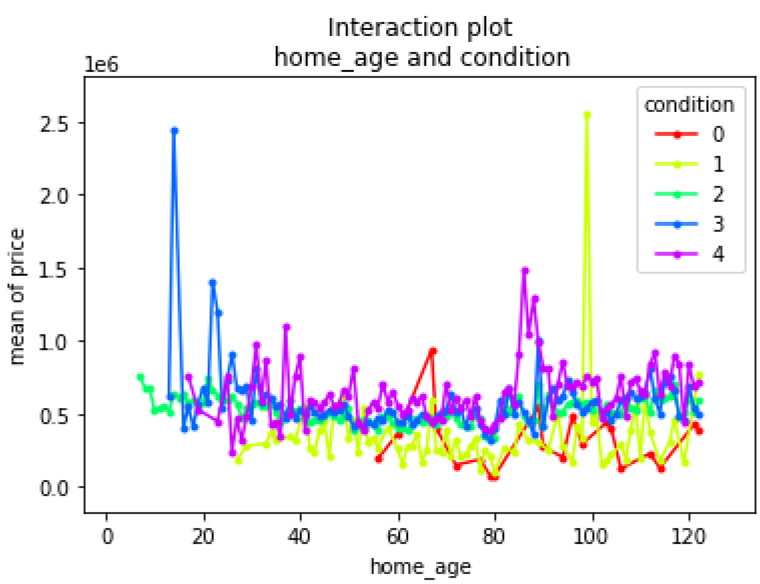
Interaction Effect Plot Helper Function

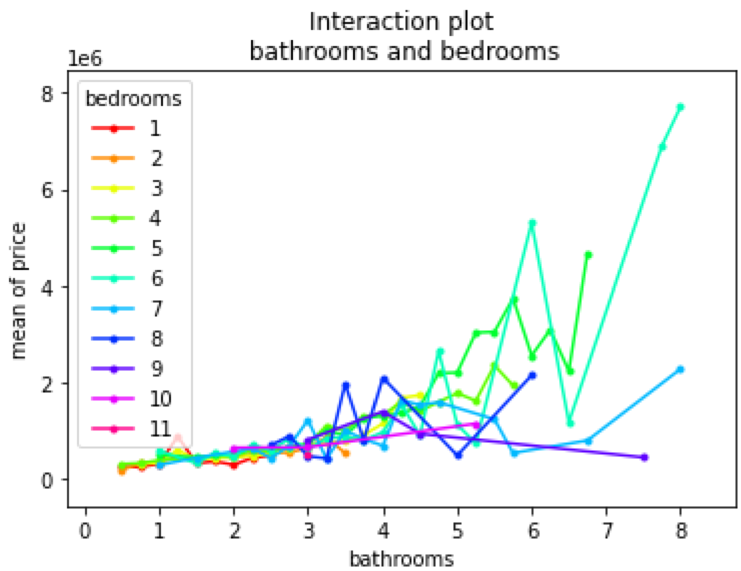
In [23]:
```
def interaction_analysis(data, var1, var2):
    '''
    Produces an interaction plot with programically determined
    title.
    Input:
         data: Pandas dataframe
         var1: column name of first variable
         var2: column name of second variable
    Output:
         interaction plot with title using var1 and var2
```

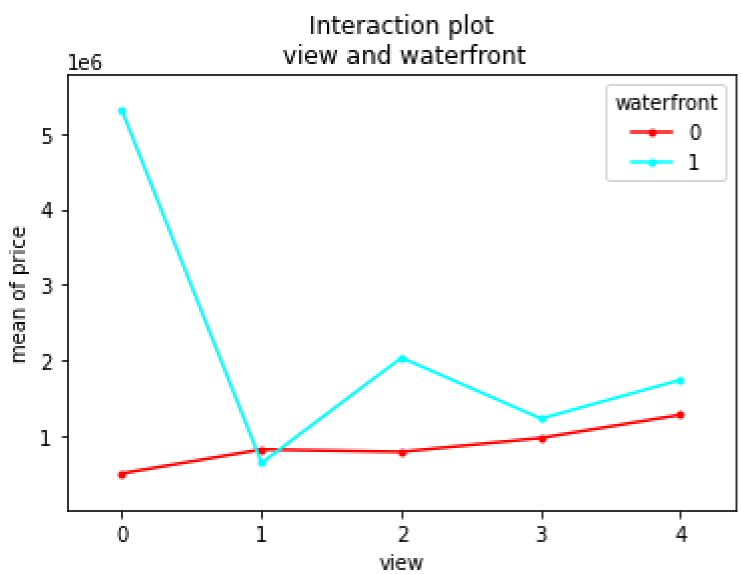
```
    '''
    # Import library
    from statsmodels.graphics.factorplots import interaction_plot

    # Make interaction plot
    fig = interaction_plot(data[var1], df_clean[var2], data['price'])
    plt.title(f'Interaction plot\n {var1} and {var2}') # set title
```

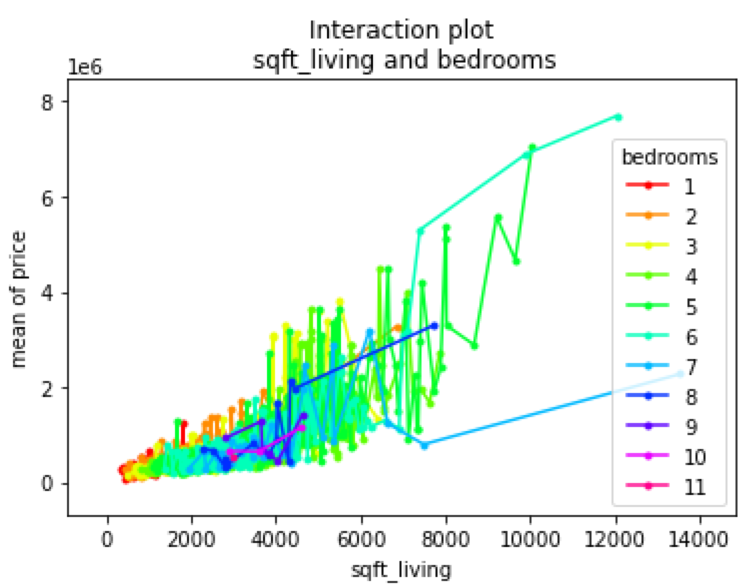
Interaction Plots

In [24]:
```
import pandas as pd
from statsmodels.graphics.factorplots import interaction_plot

# Interaction of home age and condition
interaction_analysis(df_clean, 'home_age', 'condition')

# Interaction of bathrooms and bedrooms
interaction_analysis(df_clean, 'bathrooms', 'bedrooms')

# interaction of view and waterfront
interaction_analysis(df_clean, 'view', 'waterfront')

# Interaction of sqft_living and bedrooms
interaction_analysis(df_clean, 'sqft_living', 'bedrooms')
```









Interaction Effects Interpretation

On an interaction plot, parallel lines indicate that there is no interaction effect while different slopes and lines that cross suggest that an interaction may be present. `home_age` and `condition`, `bathrooms` and `bedrooms`, and `sqft_living` and `bedrooms` each show interaction effects. This means a third variable influences the relationship between a dependent and independent variable. The relationship changes depending on the value of the third variable.

`view` and `waterfront` shows a slight interactive effect but only for view of 1 with no waterfront property. This may because the view variable is measuring the view of the mountains in Washington and not the view of a waterfront property. This data also may be too subjective or variable from different observers.

Modeling and Regression Results

In this section I run the linear regression analysis and report the results.

Residual plots

A residual plot is produced for the independent variables with the highest correlation to `price`. Residual plots show the residual error plotted against the actual sale price. This will allow me to assess heteroskedasticity as residual plots will have a random pattern around 0 with homoskedasticity but a cone shape pattern with heteroskedasticity.

In [25]:

```
# Residual plot
def residual_plot(data, X, Y, xlogged=False, ylogged=False):
    '''
    Residual Plot Function
    Input:
          data = Pandas dataframe
          X: independent variable
          Y: dependent variable
          logged: Is the axis logged?
    Output:
          Residual plot
    '''
    x = data[X]
    y = data[Y]

    # set label for x-axis if ind var is logged
    if xlogged:
        label_x = f'log({X})'
    else: # not logged
        label_x = f'{X}'
    # set label for y-axis if dep var is logged
    if ylogged:
        label_y = f'log({Y})\n$USD'
    else: # not logged
        label_y = f'{Y}\n$USD'

    # make plots
```

```python
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,4))
    ax1.scatter(x, y)
    m, b = np.polyfit(x, y, 1) # regression line
    ax1.plot(x, m*x+b, color='red') # plot regression line
    ax1.set(xlabel=label_x, ylabel=label_y)
    sns.residplot(x=x, y = y, ax=ax2) # residual plot
    # title of scatterplot
    ax1.title.set_text(f'Scatterplot: {X} versus {Y}')
    # title of residual plot
    ax2.title.set_text(f'Residual plot: {X} versus {Y}')
    ax2.set(ylabel='residual', xlabel=label_x) # residual plot y-axis label
```

Linear Regression Helper Function

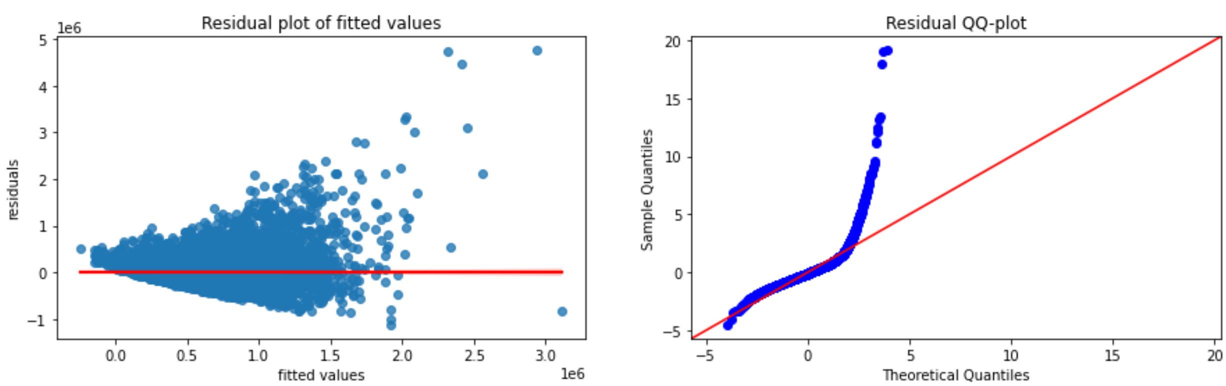
In [26]:

```python
def lin_reg_model(data, features, model_name, formula):
    '''
    Runs OLS linear regression
    Input:
        - data: clean data
        - features: independent variables that will be included in model
        - formula: regression formula in R-style
        - model_name: Name you will call the model (ex. Model 1, Model 2)
    Output:
        - OLS summary
        - Residual QQ-plot
        - OLS model object
        - Prediction interval
        - R-squared
        - Root Mean Squared Error
    '''
    target = 'price'
    y = data[target] # outcome data
    X = data[features]

    # Linear Regression using statsmodel library
    data = sm.add_constant(data)
    model = sm.OLS.from_formula(formula=formula, data=data).fit()

    # Predict values from the model
    y_predict = model.predict(X)

    # Create K-Fold cross-validation object
    kf = KFold(n_splits=5, shuffle=True) #K-Fold of 5, shuffle data, 20% test data

    # Regression model using sklearn
    lm = LinearRegression()

    # Cross-validated R-squared calculation
    r2_scores = cross_val_score(lm, X, y, cv=kf, scoring = 'r2')
    r2 = np.mean(np.absolute(r2_scores)) # calculate the mean r-squared

    # Calculate prediction interval
    sum_errs = np.sum((y - y_predict) ** 2) # Sum of errors
    stdev = np.sqrt(1/(len(y)-2) * sum_errs) # Standard deviation
    interval = 1.96 * stdev # Prediction interval

    # Cross-validated Root Mean Squared Error
```

```
    scores = cross_val_score(lm, X, y, cv=kf, scoring = 'neg_mean_squared_error')
    RMSE = np.sqrt(np.mean(np.absolute(scores)))

    # Plotting
    fig, ax = plt.subplots(1, 2, figsize=(15,4))

    # Residual plot
    sns.regplot(x=model.fittedvalues, y=model.resid, ax=ax[0], line_kws={'color':'r'})
    ax[0].title.set_text('Residual plot of fitted values')
    ax[0].set(ylabel='residuals',xlabel='fitted values')
    ax[1].title.set_text('Residual QQ-plot')

    # Plot residual qq-plot
    sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True, ax=ax[1])
    plt.show() # see https://github.com/statsmodels/statsmodels/issues/5493 for bug
    #fig.suptitle(f'{model_name} Residual QQ plot')

    print(model.summary())
    print('\n')
    print(f'R-squared: {r2:.2f}')
    print(f'Prediction Interval: {interval:.2f}')
    print(f'Root Mean Squared Error: {RMSE:.2f}')

    return model, r2, interval, RMSE
```

M1 Results

In [27]:
```
# Model 1 features (highest correlated)
model_1_features = corr_with_price['Features'].values
formula_1 = 'price ~' + '+'.join(model_1_features) # model formula

# get regression model results, prediction interval, and RMSE score
model_1_reg, model_1_r2, model_1_pi, m1_RMSE = lin_reg_model(
    df_clean, model_1_features, 'Model 1', formula_1)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.541
Model:                            OLS   Adj. R-squared:                  0.541
Method:                 Least Squares   F-statistic:                     8494.
Date:                Tue, 09 Aug 2022   Prob (F-statistic):               0.00
Time:                        02:00:50   Log-Likelihood:            -2.9897e+05
No. Observations:               21597   AIC:                         5.980e+05
Df Residuals:                   21593   BIC:                         5.980e+05
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept   -6.564e+05   1.36e+04    -48.298      0.000   -6.83e+05    -6.3e+05
sqft_living   234.5900      4.039     58.075      0.000     226.672     242.508
grade        1.108e+05   2325.608     47.637      0.000    1.06e+05    1.15e+05
sqft_above    -78.0959      4.427    -17.642      0.000     -86.773     -69.419
==============================================================================
Omnibus:                    17102.886   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1062513.676
Skew:                           3.332   Prob(JB):                         0.00
Kurtosis:                      36.709   Cond. No.                     2.43e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 2.43e+04. This might indicate that there are
strong multicollinearity or other numerical problems.


R-squared: 0.54
Prediction Interval: 487671.26
Root Mean Squared Error: 248906.83
```

Interpretation of M1 price variable

The p-values indicate that each of the variables chosen has a statistical significant relationship with the dependent variable, the R^2 value is low at 0.54.

The residual plot shows a cone like pattern indicating the homoskedasticity assumption is not valid. Also the assumption that the error term is normally distributed is not met as the residuals are not normally distributed. To make accurate inferences these assumptions need to be met.

As the dependent variable, `price` is not normally distributed, I will log transform `price` to keep the variance constant and not change porportionally with the magnitude of `price`.

Log transforming `price` variable

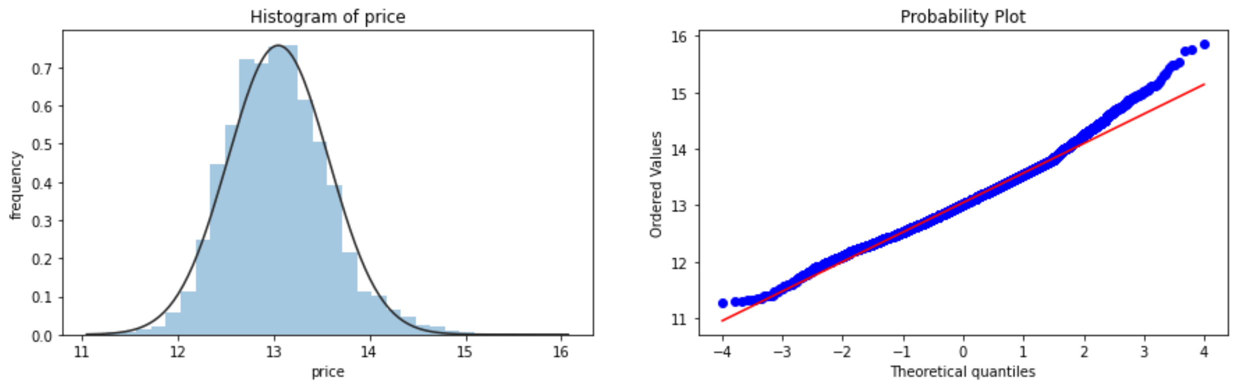
The distirubtion of the dependent variable, `price`, is normally distributed after being log transformed.

In [28]:
```
# log transform price data
```

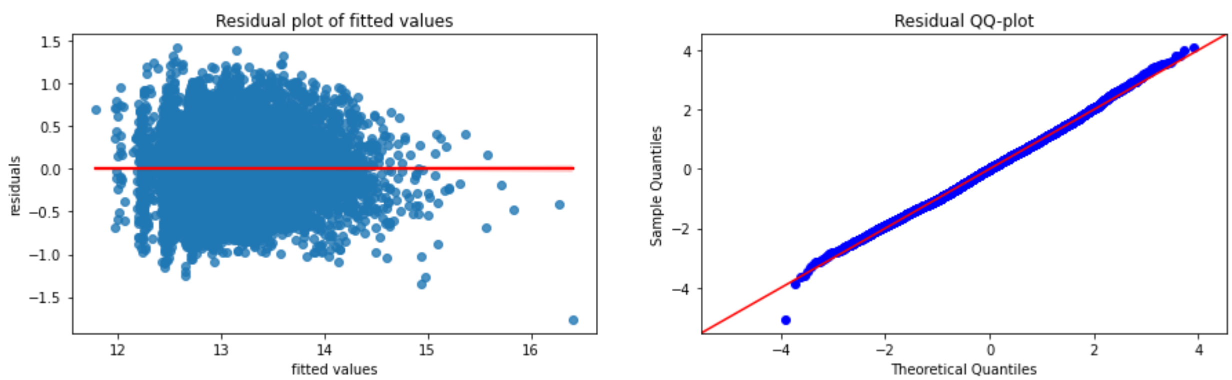
```python
df_clean['price'] = np.log(df_clean['price'])
```

In [29]:
```python
# Histogram of log transformed price independent variable
hist_plot(df_clean, 'price')
```

```
EDA of price variable
Skewness: 0.4310041773299232
Kurtosis: 0.691048515911131
```



In [30]:
```python
# Regression analysis using log transformed dependent variable
# get regression model results, prediction interval, and RMSE score
model_1_reg, model_1_r2, model_1_pi, m1_RMSE = lin_reg_model(
    df_clean, model_1_features, 'Model 1', formula_1)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.564
Model:                            OLS   Adj. R-squared:                  0.564
Method:                 Least Squares   F-statistic:                     9322.
Date:                Tue, 09 Aug 2022   Prob (F-statistic):               0.00
Time:                        02:00:52   Log-Likelihood:                -7820.6
No. Observations:               21597   AIC:                         1.565e+04
Df Residuals:                   21593   BIC:                         1.568e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      11.0806      0.019    583.615      0.000      11.043      11.118
sqft_living     0.0003   5.64e-06     53.520      0.000       0.000       0.000
grade           0.2055      0.003     63.266      0.000       0.199       0.212
sqft_above     -0.0001   6.18e-06    -21.209      0.000      -0.000      -0.000
==============================================================================
Omnibus:                       37.044   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               37.179
Skew:                           0.101   Prob(JB):                     8.45e-09
Kurtosis:                       3.029   Cond. No.                     2.43e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 2.43e+04. This might indicate that there are
strong multicollinearity or other numerical problems.


R-squared: 0.56
Prediction Interval: 0.68
Root Mean Squared Error: 0.35
```

M1 Interpretation

The p-values ($\alpha < 0.05$) indicate that each of the variables chosen has a statistical significant relationship with the dependent variable. The residual plot has a random pattern and the QQ-plot of the residuals indicate that they are normally distributed. Log transforming `price` has worked and the assumptions are met. There does seem to be a high level of multicollinearity.

The linear function of M1 is:

$$\hat{y}_{\underset{price}{sales}} = 0.0003X_{\underset{living}{sqft}} + 0.201X_{grade} - 0.0001X_{\underset{above}{sqft}} + 11.08$$

To interpret the coefficients we need to take into account that `price` has been log transformed (3).

$$(e^{\beta_0} - 1) \times 100, \text{where } \beta_0 \text{ is the coefficient}$$

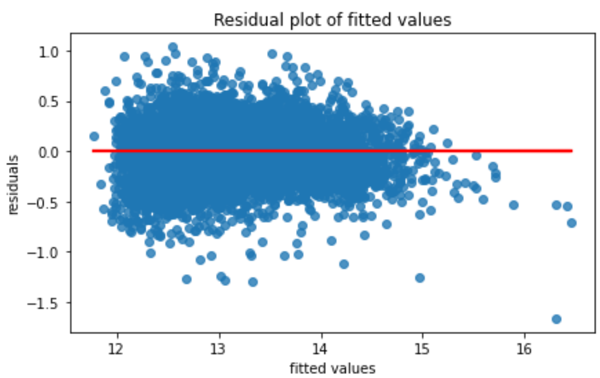
The coefficient for `grade` of a home is 0.201. This means for every one-unit increase in the `grade` of the home, there is a 22.8% increase in sales price. The coefficient for `sqft_living` is 0.0003. This means for every square foot increase in the living space in a house there is a

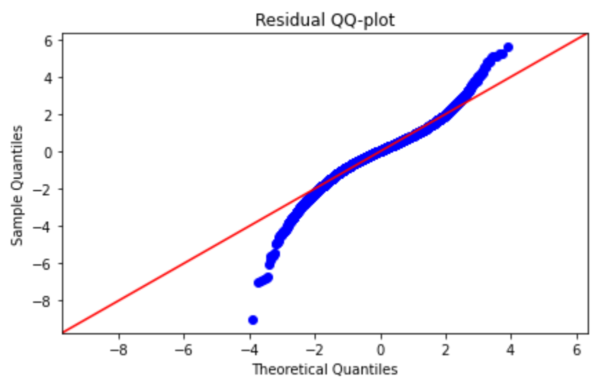
0.03% increase in sale price. Lastly, the coefficient for `sqft_above` is 0.0001 so for every square foot increase in the above ground area of a house there is a 0.01% reduction in sale price. Interestingly, the area of the space above the home is penalized in this model. This may be due to multicolinearity between the `sqft_living` and `sqft_above`. These values are so low as to be insignificant.

R^2 value is low at 0.56, RMSE is 0.35, and prediciton interval is 0.68.

Model 2 Results

In [31]:
```
# Model 2 features (Stepwise backward Design)
model_2_features = backward_elimination(df_clean, df_clean[['price']])
formula_2 = 'price ~' + '+'.join(model_2_features) # model formula

# get regression model results, prediction interval, and RMSE score
model_2_reg, model_2_r2, model_2_pi, m2_RMSE = lin_reg_model(df_clean, model_2_feature
```





OLS Regression Results

Dep. Variable:	price	R-squared:	0.877
Model:	OLS	Adj. R-squared:	0.876
Method:	Least Squares	F-statistic:	1938.
Date:	Tue, 09 Aug 2022	Prob (F-statistic):	0.00
Time:	02:00:55	Log-Likelihood:	5818.9
No. Observations:	21597	AIC:	-1.148e+04
Df Residuals:	21517	BIC:	-1.084e+04
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	11.0932	0.016	692.273	0.000	11.062	11.125
bathrooms	0.0383	0.003	12.792	0.000	0.032	0.044
sqft_living	9.116e-05	5.44e-06	16.751	0.000	8.05e-05	0.000
sqft_lot	6.304e-07	3.32e-08	18.998	0.000	5.65e-07	6.95e-07
floors	-0.0293	0.004	-8.027	0.000	-0.036	-0.022
waterfront	0.4626	0.017	27.368	0.000	0.429	0.496
view	0.0610	0.002	30.593	0.000	0.057	0.065
condition	0.0587	0.002	26.477	0.000	0.054	0.063
grade	0.0921	0.002	44.088	0.000	0.088	0.096
sqft_above	0.0001	6.12e-06	18.722	0.000	0.000	0.000
basement	0.0437	0.005	9.260	0.000	0.034	0.053
sqft_living15	8.428e-05	3.32e-06	25.365	0.000	7.78e-05	9.08e-05
home_age	0.0018	0.000	14.790	0.000	0.002	0.002
yr_since_reno	-0.0013	0.000	-11.027	0.000	-0.002	-0.001
zipcode_98002	-0.0494	0.014	-3.471	0.001	-0.077	-0.022
zipcode_98004	1.0693	0.012	89.714	0.000	1.046	1.093
zipcode_98005	0.7004	0.015	45.408	0.000	0.670	0.731
zipcode_98006	0.5915	0.010	58.200	0.000	0.572	0.611
zipcode_98007	0.6236	0.017	37.693	0.000	0.591	0.656
zipcode_98008	0.6205	0.012	50.433	0.000	0.596	0.645
zipcode_98010	0.2268	0.019	11.751	0.000	0.189	0.265
zipcode_98011	0.4266	0.014	29.751	0.000	0.398	0.455
zipcode_98014	0.2828	0.018	16.061	0.000	0.248	0.317
zipcode_98019	0.3132	0.014	21.610	0.000	0.285	0.342
zipcode_98023	-0.0547	0.010	-5.538	0.000	-0.074	-0.035
zipcode_98024	0.3948	0.022	18.346	0.000	0.353	0.437
zipcode_98027	0.4779	0.011	44.852	0.000	0.457	0.499
zipcode_98028	0.3997	0.012	32.598	0.000	0.376	0.424
zipcode_98029	0.5740	0.012	48.820	0.000	0.551	0.597
zipcode_98030	0.0367	0.013	2.882	0.004	0.012	0.062
zipcode_98031	0.0540	0.012	4.350	0.000	0.030	0.078
zipcode_98032	-0.0552	0.017	-3.165	0.002	-0.089	-0.021
zipcode_98033	0.7527	0.010	72.111	0.000	0.732	0.773
zipcode_98034	0.5155	0.010	53.736	0.000	0.497	0.534
zipcode_98038	0.1547	0.009	16.521	0.000	0.136	0.173
zipcode_98039	1.1889	0.027	44.069	0.000	1.136	1.242
zipcode_98040	0.8255	0.013	65.594	0.000	0.801	0.850
zipcode_98042	0.0402	0.010	4.207	0.000	0.021	0.059
zipcode_98045	0.3099	0.014	22.810	0.000	0.283	0.337
zipcode_98052	0.6092	0.009	64.215	0.000	0.591	0.628
zipcode_98053	0.5565	0.011	51.626	0.000	0.535	0.578
zipcode_98055	0.1139	0.013	9.087	0.000	0.089	0.139
zipcode_98056	0.2907	0.011	27.302	0.000	0.270	0.312
zipcode_98058	0.1405	0.010	13.787	0.000	0.121	0.160
zipcode_98059	0.3103	0.010	30.544	0.000	0.290	0.330
zipcode_98065	0.3713	0.012	31.010	0.000	0.348	0.395

```
zipcode_98070     0.3129      0.018     17.111      0.000       0.277       0.349
zipcode_98072     0.4663      0.012     37.410      0.000       0.442       0.491
zipcode_98074     0.5258      0.010     50.175      0.000       0.505       0.546
zipcode_98075     0.5187      0.011     45.402      0.000       0.496       0.541
zipcode_98077     0.4149      0.014     28.818      0.000       0.387       0.443
zipcode_98102     0.8992      0.019     46.257      0.000       0.861       0.937
zipcode_98103     0.7773      0.010     78.771      0.000       0.758       0.797
zipcode_98105     0.8937      0.014     64.466      0.000       0.867       0.921
zipcode_98106     0.2966      0.012     25.509      0.000       0.274       0.319
zipcode_98107     0.7925      0.013     61.047      0.000       0.767       0.818
zipcode_98108     0.3206      0.015     21.721      0.000       0.292       0.350
zipcode_98109     0.9393      0.019     49.576      0.000       0.902       0.976
zipcode_98112     0.9905      0.013     75.090      0.000       0.965       1.016
zipcode_98115     0.7785      0.010     79.822      0.000       0.759       0.798
zipcode_98116     0.7169      0.012     60.352      0.000       0.694       0.740
zipcode_98117     0.7658      0.010     76.917      0.000       0.746       0.785
zipcode_98118     0.4185      0.010     41.460      0.000       0.399       0.438
zipcode_98119     0.9274      0.015     60.927      0.000       0.898       0.957
zipcode_98122     0.7558      0.013     59.629      0.000       0.731       0.781
zipcode_98125     0.5371      0.011     49.958      0.000       0.516       0.558
zipcode_98126     0.4999      0.011     43.581      0.000       0.477       0.522
zipcode_98133     0.4257      0.010     42.415      0.000       0.406       0.445
zipcode_98136     0.6385      0.013     49.597      0.000       0.613       0.664
zipcode_98144     0.6166      0.012     52.688      0.000       0.594       0.640
zipcode_98146     0.2485      0.012     20.230      0.000       0.224       0.273
zipcode_98148     0.1315      0.025      5.227      0.000       0.082       0.181
zipcode_98155     0.4002      0.010     38.690      0.000       0.380       0.421
zipcode_98166     0.2795      0.013     21.680      0.000       0.254       0.305
zipcode_98168     0.0438      0.013      3.465      0.001       0.019       0.069
zipcode_98177     0.5603      0.013     43.356      0.000       0.535       0.586
zipcode_98178     0.1148      0.013      8.982      0.000       0.090       0.140
zipcode_98188     0.0745      0.017      4.433      0.000       0.042       0.107
zipcode_98198     0.0369      0.012      2.987      0.003       0.013       0.061
zipcode_98199     0.8144      0.012     67.498      0.000       0.791       0.838
==============================================================================
Omnibus:                     1934.611   Durbin-Watson:                   2.010
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8714.901
Skew:                          -0.339   Prob(JB):                         0.00
Kurtosis:                       6.037   Cond. No.                     1.62e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 1.62e+06. This might indicate that there are
strong multicollinearity or other numerical problems.


R-squared: 0.88
Prediction Interval: 0.36
Root Mean Squared Error: 0.19
```

M2 Interpretation

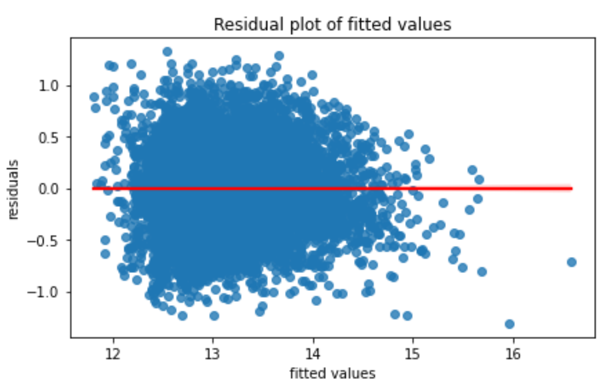
There are many more independent variables used in M2 as compared to M1, which may overfit the data. This model used all the independent variables expect for some of the Zip codes after recoding Zip code to dummy variables. `price` is still log transformed. Though there are many variables, the p-values indicate that each of the variables chosen has a statistical significant relationship with the dependent variable, `price`. The residual plot has a random pattern so the homoskedasticity assumption is met. The QQ-plot of the residuals indicate that they are not normally distributed. R^2 is 0.88 for M2. This is tremendous improvement over M1. RMSE is 0.19 and the prediction interval is 0.36.

The coefficient for `condition` for M2 is 0.0587. This means that for ever one-unit increase in condition of a house the sale price is expected to increase on average 6%. Adding an additional full bathroom would increase the sale price of a home by about 3.9%.

M2 can tell us which customers to target when marketing. Waterfront properties have a sale price over 59% more homes not on water. A 1-unit increase in the score of a view increases the sale price by about 6%. Some of the highest coefficients with M2 is in the Zip code variables. For example, with all other independent variables held constant a home in 98039 would sell for 228% more than Zip code 98003. The real estate agents should avoid marketing to property owners in 98002, 98003, 98023, 98032, 98042, and 98198 as these sell for much less than other Zip codes in King County, WA as these sell for much less than other Zip codes in King County, WA.

M3 Results

In [32]:
```
model_1_features_del_sqftabove = np.delete(model_1_features, 2)
model_3_features =  np.append(model_1_features_del_sqftabove, ['bedrooms', 'home_age',
formula_3 = 'price ~' + '+'.join(model_3_features) + '+ home_age*condition + sqft_livi
model_3_reg, model_3_r2, model_3_pi, m3_RMSE = lin_reg_model(df_clean, model_3_feature
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.625
Model:                            OLS   Adj. R-squared:                  0.625
Method:                 Least Squares   F-statistic:                     5141.
Date:                Tue, 09 Aug 2022   Prob (F-statistic):               0.00
Time:                        02:00:56   Log-Likelihood:                -6199.3
No. Observations:               21597   AIC:                         1.241e+04
Df Residuals:                   21589   BIC:                         1.248e+04
Df Model:                           7
Covariance Type:            nonrobust
=====================================================================================
===
                          coef    std err          t      P>|t|      [0.025      0.9
75]
-------------------------------------------------------------------------------------
---
Intercept              10.6119      0.035    306.963      0.000      10.544      10.
680
sqft_living             0.0003   9.76e-06     28.011      0.000       0.000       0.
000
grade                   0.2421      0.003     77.287      0.000       0.236       0.
248
bedrooms                0.0064      0.005      1.193      0.233      -0.004       0.
017
home_age                0.0011      0.000      3.344      0.001       0.000       0.
002
condition              -0.0718      0.009     -7.563      0.000      -0.090      -0.
053
home_age:condition      0.0016      0.000     12.406      0.000       0.001       0.
002
sqft_living:bedrooms -1.339e-05   2.19e-06     -6.115      0.000   -1.77e-05    -9.1e
-06
==============================================================================
Omnibus:                       50.969   Durbin-Watson:                   1.961
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               56.642
Skew:                          -0.077   Prob(JB):                     5.02e-13
Kurtosis:                       3.197   Cond. No.                     1.48e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 1.48e+05. This might indicate that there are
strong multicollinearity or other numerical problems.


R-squared: 0.62
Prediction Interval: 0.63
Root Mean Squared Error: 0.32
```

M3 Interpretation

M3 includes interaction effects along with main effects. The interaction effect of `bathrooms` and `bedrooms` was not included as the p-value was above 0.05, indicating that the interaction with `price` is not statistically significant. Likewise, the variable `bedrooms` is not have a statistically significant main interaction with `price`. The residual plot has a random pattern so there is homoskedasticity. The QQ-plot of the residuals indicate that they are normally distributed. These results show the model meets the assumptions of OLS. There is multicollinearity between some of the variables.

R^2 is 0.62. This is an improvement over M1 but is less than M2. RMSE is 0.32 and the prediction interval is 0.63.

The coefficient for `sqft_living` is 0.0003, the same as M1. The coefficient for `condition` with M3 is -0.077 so for each one-unit increase in the condition of a house the sale price is expected to **decrease** about 8% on average.

Conclusions

Model Analysis and Comparisons

I will use coefficient of determination (R^2), prediciton intervals, and root means squared error (RMSE) to compare the models and determine which model meets the needs of our client. This includes the most accurate predictive power with a tight range in possibilities.

In [33]:
```
# Table of model's R-squared, PI, and Cross-validated RMSE
r2 = [model_1_reg.rsquared, model_2_reg.rsquared, model_3_reg.rsquared]
pi = [model_1_pi, model_2_pi, model_3_pi]
rmse = [m1_RMSE, m2_RMSE, m3_RMSE]
results = pd.DataFrame({'R_squared': r2,
                        'PI': pi,
                        'RMSE': rmse},
                       index = ['M1', 'M2', 'M3'])
print('Table 5: Regression Results Table')
results
```

```
Table 5: Regression Results Table
```

Out[33]:

	R_squared	PI	RMSE
M1	0.564301	0.681244	0.347598
M2	0.876794	0.362265	0.185685
M3	0.625043	0.631976	0.323954

Coefficient of Determination (R^2)

R^2 is a statistical estimate of how close the observed data is to the regression line of each model. It is the porportion of variation in the dependent variable that is predictive from the independent variable. I measured R^2 using 5-fold cross-validation.

$$R^2 = 1 - \frac{\text{Residual Sum of Squares (RSS)}}{\text{Total Sum of Squares (TSS)}}$$

$$= 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

As Table 5 above shows, the model with the highest R^2 is M2 at 0.88. This is a great score for a predictive model, the higher the better. M1 has the lowest R^2 score at 0.55 and M3 is 0.63.

Prediction Interval

A prediction interval (PI) is the range where a single new observation is likely to fall given specific values of the indpendent variables. The prediciton interval can be use to assess if the predicitons are sufficiently in a narrow range to satisfy the client's requirement. Prediction intervals can be compared across models. Smaller intervals indicate tighter predictive range. Large prediction intervals tell us the model could have a wide range in its predictions and would not meet the client's needs.

The prediction interval is calculated by,

$$PI = 1.96 \times s,$$
$$\text{where s is the sample standard deviation calculated by}$$

$$s = \sqrt{\frac{1}{N-2} \times RSS}$$

M2 also has the lowest predictive intervals. A prediction interval is the range where a single new observation is likely (95%) to fall given specific values of the indpendent variables. The smaller the predictive interval the more confidence the true sale price is in that region. M1 and M2 had resonably similar prediction intervals and twice the value as M2.

Root Mean Squared Error

RMSE is a measure of the mean error rate of a regression model that penalizes larger errors. The smaller the RMSE value, the closer the fitted line from the linear equation is to the actual data. It is the square root of the average squared difference between the predicted dependent value and the actual values in the dataset. Like Mean Squared Error (MSE), this statistic squares the residual error before it is averaged, which gives a high weight to large errors, but because it the square root is taken, the statistic is in the same units as the dependent variable, sale price ($USD). The lower the score of RMSE the closer the model fits the data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2},$$
$$\text{where } (\hat{y_1}, \hat{y_2}, \ldots, \hat{y_n}) \text{ are the predicted values,}$$
$$(y_1, y_2, \ldots, y_n) \text{ are the observed values}$$
$$\text{and } (n) \text{is the number of observations}$$

RMSE is estimated using K-Fold cross validation. M2 is almost half the RMSE score of M1 and M2 indicating it produces better prediction between the actual and predicted values.

Discussion on the best model

Althought M2 does not meet all the assumptions of OLS linear regression analysis, it is the best model of the three as it has a high R^2, low RMSE, and low PI. M2 also has the most interpretable variables. M1's is almost entirely dependent on the `grade` of the house. The `grade` of a home cannot be changed unless the house is completely remodeled. This model would not help our clients as it has low predictive power and does not have variables that could be improved to increase the sale price of the house.

According to M3, increasing the condition of a house will decrease the sale price of a home, which doesn't make a lot of sense.

M2 has many variables that can be interpeted to improve the sale price of the house like improving the condition and number of bathrooms. M2 is driven heavily by the location of the house in the form of the Zip code. These coefficents will be used to make actionable recommendations to real estate agents for targeted marketing.

Prediction Application

The prediction of Model 2 seems the most reliable of the three models. Below is the `price_predictor` function that input one of the three models, the new home features, and the prediction interval. This function returns the predicted house sale price and 95% prediction interval range according to that model and new data.

In [34]:
```
def price_predictor(model_no, new_data, pi):
    '''
    This function takes in a model and Pandas series object and returns
    an estimated price range for the house to be listed.

    Model 1: price ~ ln(sqft_living) + ln(sqft_above) + grade

    Model 2: price ~ bedrooms+bathrooms+sqft_living+sqft_lot+floors+waterfront+
                     view+condition+grade+sqft_above+basement+sqft_living15+sqft_lot15
                     home_age+yr_since_reno+zipcode

    Model 3: price ~ sqft_living+grade+sqft_above+ home_age*condition + sqft_living*be

    Input:
          model_no: Statsmodel linear regression model results
          new_data: Pandas Series with variables needed for the regression model
          pi: prediction interval for that model
    Output:
          Predicted sale price
          Predicted sale price range
    '''
    price_ln = model_no.predict(new_data).values[0] # predicted price
    price = round(np.exp(price_ln))
    price_low = round(np.exp(price_ln - pi)) # prediction lower bound
    price_up = round(np.exp(price_ln + pi)) # prediction upper bound

    return f'Predicted price: {price}, range: {price_low} - {price_up}'
```

Demonstration

In [35]:
```
#data = df_clean.iloc[8677] # for testing purposes
```

In [48]:
```
data = df_clean.sample().squeeze() # Randomly sample data for prediction
data.head(10) # new house features
```

Out[48]:
```
price             13.122363
bedrooms           3.000000
bathrooms          2.500000
sqft_living     1210.000000
sqft_lot        1200.000000
floors             3.000000
waterfront         0.000000
view               0.000000
condition          2.000000
grade              8.000000
Name: 20464, dtype: float64
```

In [49]:
```
# Model 1 prediction
data_new = data.drop('price') # Remove actual price
print(price_predictor(model_1_reg, data_new, model_1_pi)) # Run prediciton function
sale_price = round(np.exp(data['price'])) # assign actual sale price
print(f'Actual sale price: {sale_price}') # print out actual sale price
```

```
Predicted price: 413149, range: 209048 - 816520
Actual sale price: 500000
```

In [50]:
```
# Model 2 Prediction function
print(price_predictor(model_2_reg, data_new, model_2_pi))
print(f'Actual sale price: {sale_price}')
```

```
Predicted price: 484124, range: 336998 - 695483
Actual sale price: 500000
```

In [51]:
```
# Model 3 prediction function
print(price_predictor(model_3_reg, data_new, model_3_pi))
print(f'Actual sale price: {sale_price}')
```

```
Predicted price: 350551, range: 186332 - 659499
Actual sale price: 500000
```

Recommendations

Summary

- Our client wants to be able to predict sales price, identify where to market in King County, WA, and how customers can improve their home to increase sale price.
- Ordinary least squares linear regression was used to create three models.
- The three models were compared using R^2, Prediction Intervals (PI), and Root Mean Squared Error (RMSE).
- Model 2 (M2) is the best model as it has the best predictive capabilities, R-squared of 0.88, low RMSE and PI, though the error is not normally distributed.

Actionable Recommendations

1. Improving the condition of a house by one-unit will increase the sale price by about 6%.
2. Adding an additional full bathroom would increase the sale price of a house by about 3.9%.
3. Marketing should be focused throughout King County, WA except in Zip codes 98002, 98003, 98023, 98032, 98042, and 98198 as the value of the homes sold in these Zip codes are well under the rest of King County, WA.
4. Market real estate services toward owners of waterfront properties as these sell for 59% more than homes not waterfront.

Next Steps

This model could be improved to make better predictions by adding more data and additional variables, such as crime rate in the geographic location of the home, the zoned public school ranking, and time the house was on the market until it was sold. The GPS coordinates of the sold house could be used to collect the first two of these variables. The Multiple Listing Service may be a source for more recent data and on how long a house was on the market from day of listing to closing date.

Another source of data could be in the internal data of our brokerage client. They possibly have data of properties they have sold or bid on, this would include the data of the asking and bidding price of the property.

References

1. Kaggle, Kaggle, House Sales in King County, USA

2. Albert, Key Assumptions of OLS: Econometrics Review

3. University of Virginia, Interpreting Log Transformations in a Linear Model