

# Linear Regression Modeling of King County Real Estate Sale Prices

**Authors:** Aisha Baitemirova-Othman, Angela Kim, Steven Addison, Wahaj Dar

**Instructor:** David Elliott

---

## Overview

This project analyzes residential real estate sales in King County, Washington, and uses the data to create a model that predicts price based on the parameters given.

## Business Problem

Windermere Real Estate, based in Seattle, Washington, wants to better serve home buyers by being able to accurately present a price point using features of a house (ie. number of bedrooms) that buyers are looking for.

## Data Understanding

This dataset contains information about residential real estate sales in King County between May 2014 - May 2015. It includes details such as number of bedrooms and bathrooms, square footage of the home, and various features regarding location.

## Data Preparation

```
In [1]: import pandas as pd
import numpy as np
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')

!pip install geopy
import geopy
from geopy import distance
import plotly.express as px

import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols

from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as mse
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFE
```

```
Requirement already satisfied: geopy in /opt/anaconda3/envs/learn-env/lib/python3.8/site-packages (2.2.0)
Requirement already satisfied: geographiclib<2,>=1.49 in /opt/anaconda3/envs/learn-env/lib/python3.8/site-packages (from geopy) (1.52)
```

Here we import our data and skim the first five rows to get a general idea of what the dataframe looks like. We also get an initial look at missing values and datatypes that need to be converted.

```
In [2]: df = pd.read_csv('data/kc_house_data.csv')
display(df.head())
display(df.info())
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO

5 rows × 21 columns

```
<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
```

None

## Cleaning &amp; preparing the data.

```
In [3]: # Drop the 'id' and 'date' columns
# Fill in missing data
# Convert all object datatype columns to numeric

df['yr_renovated'] = df['yr_renovated'].fillna(0)
df['waterfront'] = df['waterfront'].fillna('NO')
df['waterfront'] = df['waterfront'].str.replace('NO', '0')
df['waterfront'] = df['waterfront'].str.replace('YES', '1')
df['waterfront'] = pd.to_numeric(df['waterfront'])
df['view'] = df['view'].fillna('NONE')
df['grade'] = df['grade'].str.replace('7 Average', '7')
df['grade'] = df['grade'].str.replace('8 Good', '8')
df['grade'] = df['grade'].str.replace('9 Better', '9')
df['grade'] = df['grade'].str.replace('6 Low Average', '6')
df['grade'] = df['grade'].str.replace('10 Very Good', '10')
df['grade'] = df['grade'].str.replace('11 Excellent', '11')
df['grade'] = df['grade'].str.replace('5 Fair', '5')
df['grade'] = df['grade'].str.replace('12 Luxury', '12')
df['grade'] = df['grade'].str.replace('4 Low', '4')
df['grade'] = df['grade'].str.replace('13 Mansion', '13')
df['grade'] = df['grade'].str.replace('3 Poor', '3')
df['grade'] = pd.to_numeric(df['grade'])
if [df[df['sqft_basement']] == '?']:
    df['sqft_basement'] = df['sqft_living'] - df['sqft_above']
df['sqft_basement'] = pd.to_numeric(df['sqft_basement'])
df['bedrooms'].replace(33, 3, inplace=True)
df['date'] = pd.to_datetime(df['date'])
df['yr_sold'] = df['date'].dt.year
df['house_age'] = df['yr_sold'] - df['yr_built']
df.drop(labels=['id', 'date'], axis=1, inplace=True)
```

```
In [4]: # One-hot encoding 'condition' and 'view' columns
```

```
condition = df[['condition']]
ohe = OneHotEncoder(categories="auto", sparse=False, handle_unknown="ignore")
ohe.fit(condition)
condition_enc = ohe.transform(condition)
condition_enc = pd.DataFrame(condition_enc,
                             columns=['cond_avg', 'cond_fair', 'cond_good', 'c',
                             index=df.index)
df.drop('condition', axis=1, inplace=True)
df = pd.concat([df, condition_enc], axis=1)

view = df[['view']]
ohe.fit(view)
view_enc = ohe.transform(view)
view_enc = pd.DataFrame(view_enc,
                        columns=['view_avg', 'view_excellent', 'view_fair', 'v',
                        index=df.index)
df.drop('view', axis=1, inplace=True)
df = pd.concat([df, view_enc], axis=1)
```

```
In [5]: # Create 'distance_from_bellevue' column

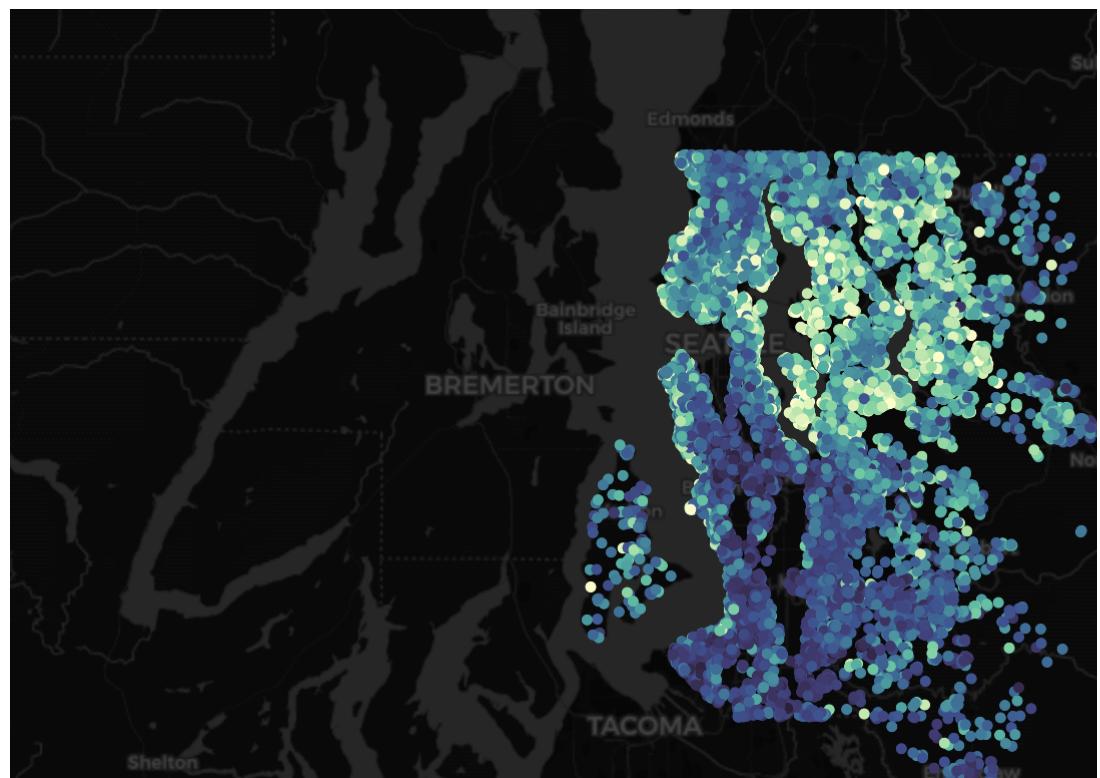
bellevue = (47.601, -122.2015)

def distancer(row):
    coords_1 = bellevue
    coords_2 = (row['lat'], row['long'])
    return geopy.distance.distance(coords_1, coords_2).miles

df['distance_from_bellevue'] = df.apply(distancer, axis=1)

# Plot distance map

distancemap = df[df['price'] <= 1000000]
fig = px.scatter_mapbox(data_frame = distancemap, lat='lat', lon='long', co
fig.update_geos(resolution=50)
fig.update_layout(mapbox_style="carto-darkmatter")
```



In [6]: # Examine correlations

```
corr = df.corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr[mask] = np.nan
(corr
 .style
 .background_gradient(cmap='mako', axis=None, vmin=-1, vmax=1)
 .highlight_null(null_color='#f1f1f1') # Color NaNs grey
 .set_precision(2))
```

Out[6]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	grade
price	nan	nan	nan	nan	nan	nan	nan	nan
bedrooms	0.32	nan	nan	nan	nan	nan	nan	nan
bathrooms	0.53	0.53	nan	nan	nan	nan	nan	nan
sqft_living	0.70	0.59	0.76	nan	nan	nan	nan	nan
sqft_lot	0.09	0.03	0.09	0.17	nan	nan	nan	nan
floors	0.26	0.18	0.50	0.35	-0.00	nan	nan	nan
waterfront	0.26	-0.00	0.06	0.10	0.02	0.02	nan	nan
grade	0.67	0.37	0.67	0.76	0.11	0.46	0.08	nan
sqft_above	0.61	0.49	0.69	0.88	0.18	0.52	0.07	0.76
sqft_basement	0.32	0.31	0.28	0.44	0.02	-0.25	0.08	0.17
yr_built	0.05	0.16	0.51	0.32	0.05	0.49	-0.02	0.45
yr_renovated	0.12	0.02	0.05	0.05	0.00	0.00	0.07	0.02
zipcode	-0.05	-0.16	-0.20	-0.20	-0.13	-0.06	0.03	-0.19
lat	0.31	-0.01	0.02	0.05	-0.09	0.05	-0.01	0.11
long	0.02	0.14	0.22	0.24	0.23	0.13	-0.04	0.20
sqft_living15	0.59	0.40	0.57	0.76	0.14	0.28	0.08	0.71
sqft_lot15	0.08	0.03	0.09	0.18	0.72	-0.01	0.03	0.12
yr_sold	0.00	-0.01	-0.03	-0.03	0.01	-0.02	-0.01	-0.03
house_age	-0.05	-0.16	-0.51	-0.32	-0.05	-0.49	0.02	-0.45
cond_avg	0.01	0.01	0.19	0.10	-0.01	0.32	-0.02	0.20
cond_fair	-0.05	-0.05	-0.08	-0.06	0.04	-0.06	-0.00	-0.08
cond_good	-0.03	-0.01	-0.17	-0.08	0.01	-0.26	0.01	-0.14
cond_poor	-0.02	-0.03	-0.04	-0.03	0.01	-0.02	0.01	-0.05
cond_verygood	0.06	0.02	-0.03	-0.02	-0.01	-0.12	0.01	-0.08
view_avg	0.15	0.05	0.09	0.13	0.04	0.01	0.00	0.12
view_excellent	0.30	0.03	0.11	0.17	0.02	0.03	0.57	0.15

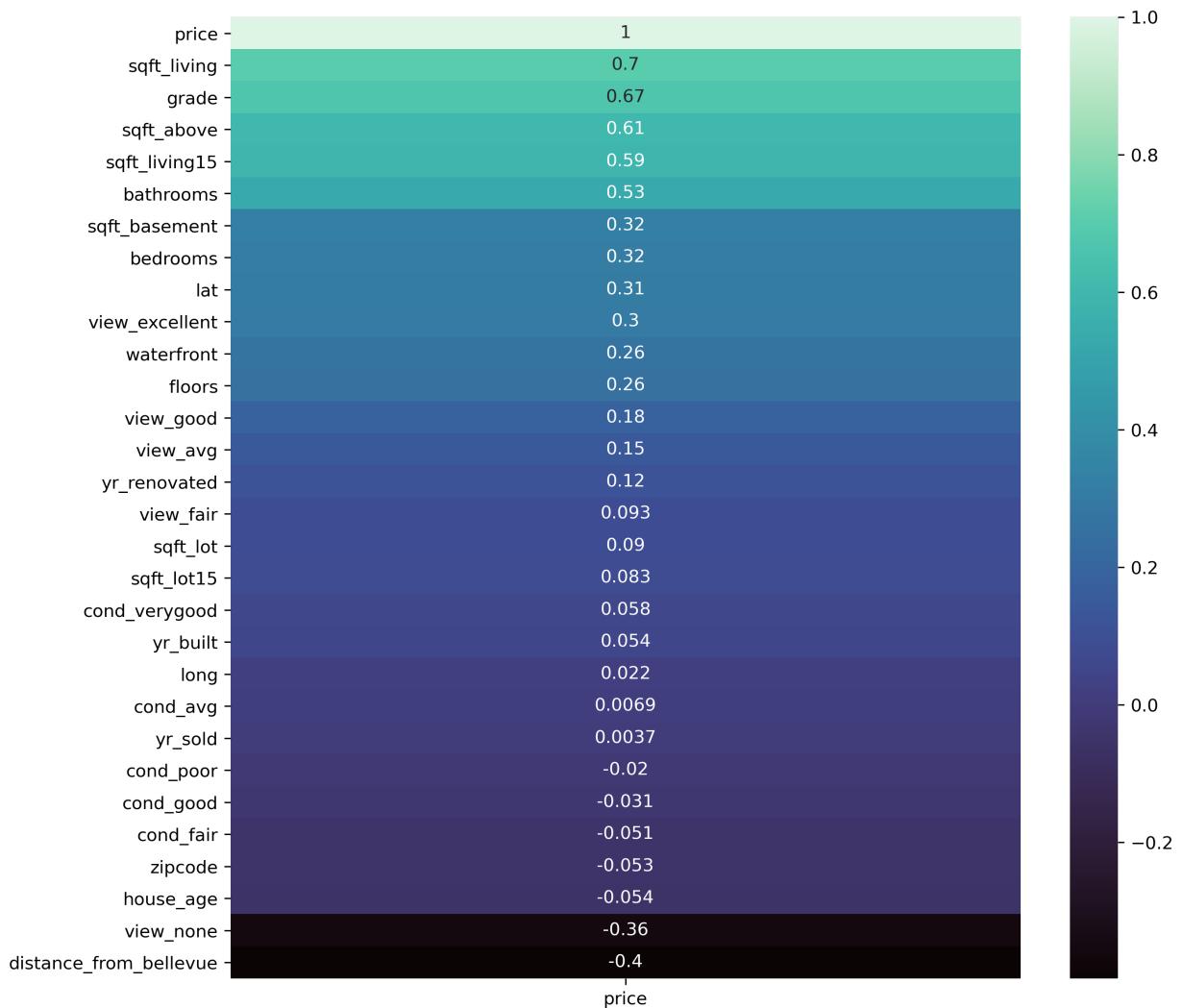
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	grade
<b>view_fair</b>	0.09	0.02	0.04	0.07	-0.01	-0.02	-0.01	0.05
<b>view_good</b>	0.18	0.05	0.11	0.16	0.07	0.02	0.04	0.14
<b>view_none</b>	-0.36	-0.08	-0.18	-0.27	-0.07	<b>-0.02</b>	-0.25	-0.24
<b>distance_from_bellevue</b>	-0.40	-0.06	-0.06	-0.11	<b>0.18</b>	<b>-0.03</b>	<b>-0.01</b>	-0.16

'sqft\_living' has the highest correlation with 'price' at 0.70. We also see high multicollinearity.

```
In [7]: ix = df.corr().sort_values('price', ascending=False).index
```

```
df_sorted = df.loc[:, ix]

plt.figure(figsize=(10,10), dpi=300)
sns.heatmap(df_sorted.corr()[['price']],
            cmap="mako",
            annot=True);
```



```
In [8]: heatmap = df[['price','sqft_living','grade','sqft_above','distance_from_bellview','view_none','house_age','zipcode']]  
heatmap
```

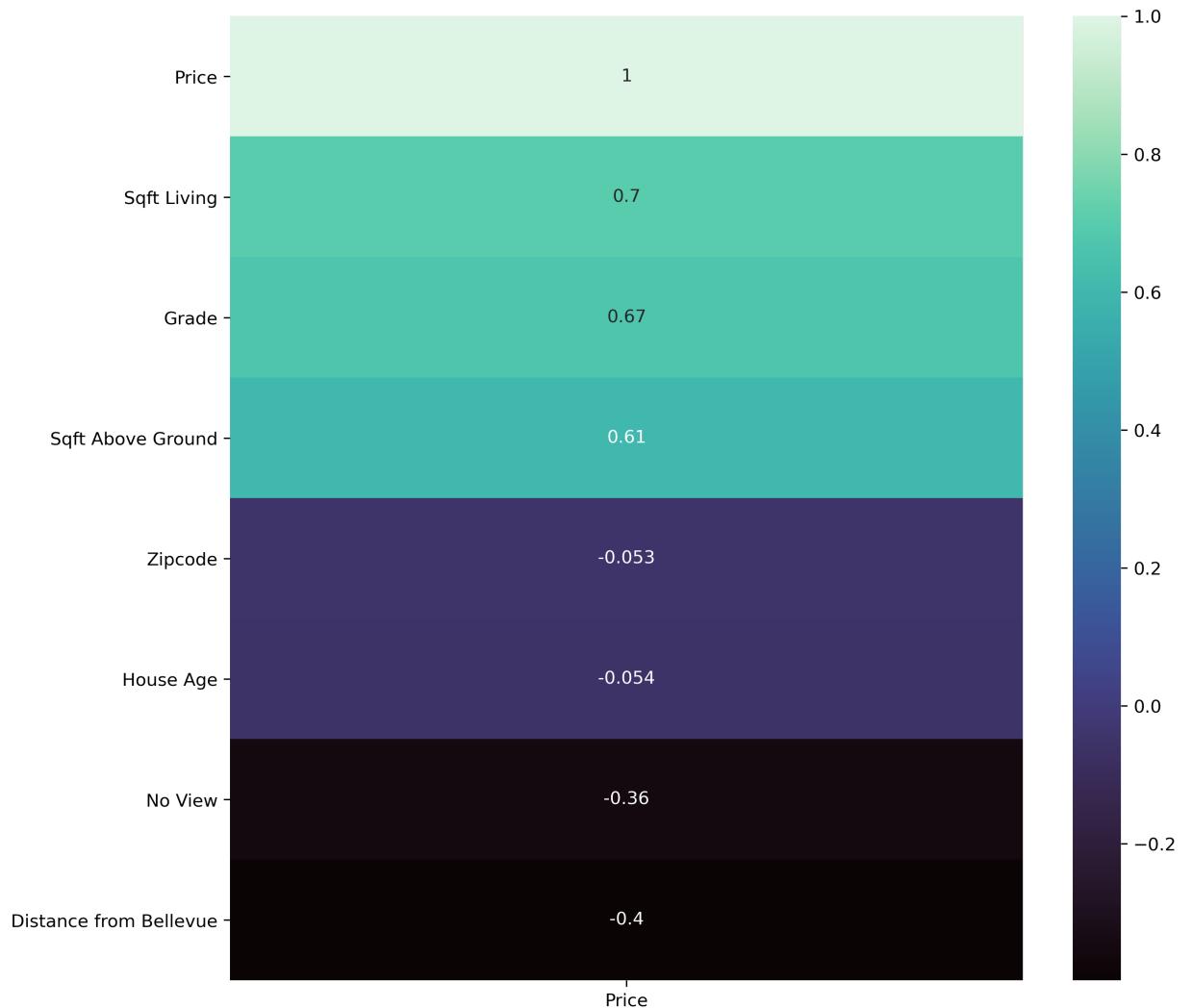
Out[8]:

	price	sqft_living	grade	sqft_above	distance_from_bellview	view_none	house_age	zipcode
0	221900.0	1180	7	1180	6.724913	1.0	59	98033
1	538000.0	2570	7	2170	9.940080	1.0	63	98033
2	180000.0	770	6	770	9.571487	1.0	82	98033
3	604000.0	1960	7	1050	10.530416	1.0	49	98033
4	510000.0	1680	8	1680	7.392620	1.0	28	98033
...	...	...	...	...	...	...	...	...
21592	360000.0	1530	8	1530	9.572031	1.0	5	98033
21593	400000.0	2310	8	2310	9.760026	1.0	1	98033
21594	402101.0	1020	7	1020	4.578875	1.0	5	98033
21595	400000.0	1600	8	1600	7.712755	1.0	11	98033
21596	325000.0	1020	7	1020	4.580999	1.0	6	98033

21597 rows × 8 columns

```
In [9]: # heatmap for presentation
ix2 = heatmap.corr().sort_values('price', ascending=False).index
df_sorted2 = df.loc[:, ix2]

plt.figure(figsize=(10,10), dpi=300)
sns.heatmap(df_sorted2.corr()[['price']],
            xticklabels=['Price'],
            yticklabels=['Price', 'Sqft Living', 'Grade', 'Sqft Above Ground',
                        'Zipcode', 'House Age', 'No View', 'Distance from',
                        'Price'],
            cmap="mako",
            annot=True)
plt.yticks(rotation=0);
```



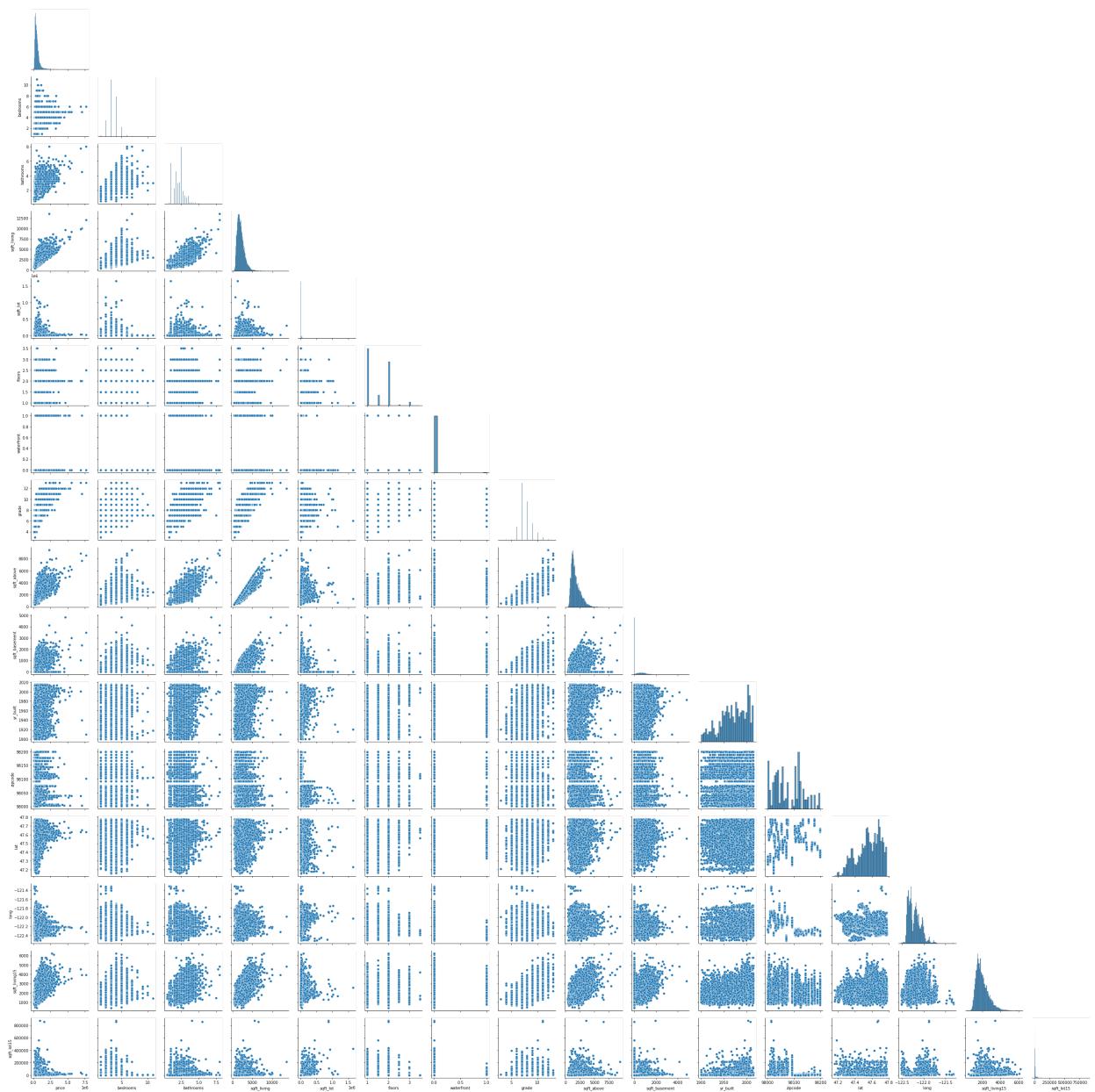


In [10]: # Scatter matrix

```
scatter_columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_living15',
                   'floors', 'waterfront', 'grade', 'sqft_above', 'sqft_basement',
                   'yr_built', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_basement']

df_scatter = df[scatter_columns]

sns.pairplot(df_scatter, corner=True);
```



Scatter matrix shows many non-normal distributions.

## Inferential Modeling

In [11]: # Analyzing OLS results

```
outcome = 'price'
dfx = df.drop('price', axis=1)
predictors = '+'.join(dfx.columns)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=df).fit()
model.summary()
```

Out[11]: OLS Regression Results

<b>Dep. Variable:</b>	price	<b>R-squared:</b>	0.725			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.724			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	2270.			
<b>Date:</b>	Sun, 21 Nov 2021	<b>Prob (F-statistic):</b>	0.00			
<b>Time:</b>	15:57:45	<b>Log-Likelihood:</b>	-2.9347e+05			
<b>No. Observations:</b>	21597	<b>AIC:</b>	5.870e+05			
<b>Df Residuals:</b>	21571	<b>BIC:</b>	5.872e+05			
<b>Df Model:</b>	25					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	-2.869e+07	4.53e+06	-6.328	0.000	-3.76e+07	-1.98e+07
<b>bedrooms</b>	-4.093e+04	1899.880	-21.544	0.000	-4.47e+04	-3.72e+04
<b>bathrooms</b>	4.038e+04	3139.238	12.864	0.000	3.42e+04	4.65e+04
<b>sqft_living</b>	111.5017	2.194	50.811	0.000	107.200	115.803
<b>sqft_lot</b>	0.2136	0.046	4.639	0.000	0.123	0.304
<b>floors</b>	2372.2405	3463.560	0.685	0.493	-4416.592	9161.073
<b>waterfront</b>	5.355e+05	1.96e+04	27.378	0.000	4.97e+05	5.74e+05
<b>grade</b>	8.844e+04	2087.403	42.369	0.000	8.43e+04	9.25e+04
<b>sqft_above</b>	78.3929	2.171	36.108	0.000	74.137	82.648
<b>sqft_basement</b>	33.1190	2.548	12.999	0.000	28.125	38.113
<b>yr_built</b>	8639.5763	939.648	9.194	0.000	6797.797	1.05e+04
<b>yr_renovated</b>	24.9298	3.824	6.520	0.000	17.435	32.425
<b>zipcode</b>	-507.6069	31.836	-15.944	0.000	-570.008	-445.206
<b>lat</b>	3.072e+05	1.26e+04	24.379	0.000	2.83e+05	3.32e+05
<b>long</b>	-1.492e+05	1.27e+04	-11.719	0.000	-1.74e+05	-1.24e+05
<b>sqft_living15</b>	8.8410	3.328	2.656	0.008	2.318	15.364
<b>sqft_lot15</b>	-0.1242	0.071	-1.759	0.079	-0.263	0.014
<b>yr_sold</b>	1.954e+04	1877.232	10.410	0.000	1.59e+04	2.32e+04

<b>house_age</b>	1.09e+04	938.899	11.611	0.000	9061.481	1.27e+04
<b>cond_avg</b>	-5.748e+06	9.07e+05	-6.337	0.000	-7.53e+06	-3.97e+06
<b>cond_fair</b>	-5.746e+06	9.07e+05	-6.334	0.000	-7.52e+06	-3.97e+06
<b>cond_good</b>	-5.725e+06	9.07e+05	-6.314	0.000	-7.5e+06	-3.95e+06
<b>cond_poor</b>	-5.785e+06	9.07e+05	-6.376	0.000	-7.56e+06	-4.01e+06
<b>cond_verygood</b>	-5.686e+06	9.07e+05	-6.272	0.000	-7.46e+06	-3.91e+06
<b>view_avg</b>	-5.792e+06	9.07e+05	-6.387	0.000	-7.57e+06	-4.01e+06
<b>view_excellent</b>	-5.552e+06	9.07e+05	-6.121	0.000	-7.33e+06	-3.77e+06
<b>view_fair</b>	-5.765e+06	9.07e+05	-6.356	0.000	-7.54e+06	-3.99e+06
<b>view_good</b>	-5.714e+06	9.07e+05	-6.300	0.000	-7.49e+06	-3.94e+06
<b>view_none</b>	-5.867e+06	9.07e+05	-6.472	0.000	-7.64e+06	-4.09e+06
<b>distance_from_bellevue</b>	-1.331e+04	328.984	-40.459	0.000	-1.4e+04	-1.27e+04

**Omnibus:** 19227.699    **Durbin-Watson:** 1.989  
**Prob(Omnibus):** 0.000    **Jarque-Bera (JB):** 2315048.410  
**Skew:** 3.794    **Prob(JB):** 0.00  
**Kurtosis:** 53.150    **Cond. No.** 1.01e+16

#### Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The smallest eigenvalue is 2.16e-18. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

The p-values for 'floors' and 'sqft\_lot15' are not statistically significant. JB is very high, indicating non-normal distributions. There is strong multicollinearity.

Previously, we saw that 'price' and 'sqft\_living' have the strongest correlation, but the scatter matrix reveals that they are not normally distributed.

```
In [12]: # OLS between 'price' and 'sqft_living'
f = 'price-sqft_living'
model = ols(f, df).fit()
model.summary()
```

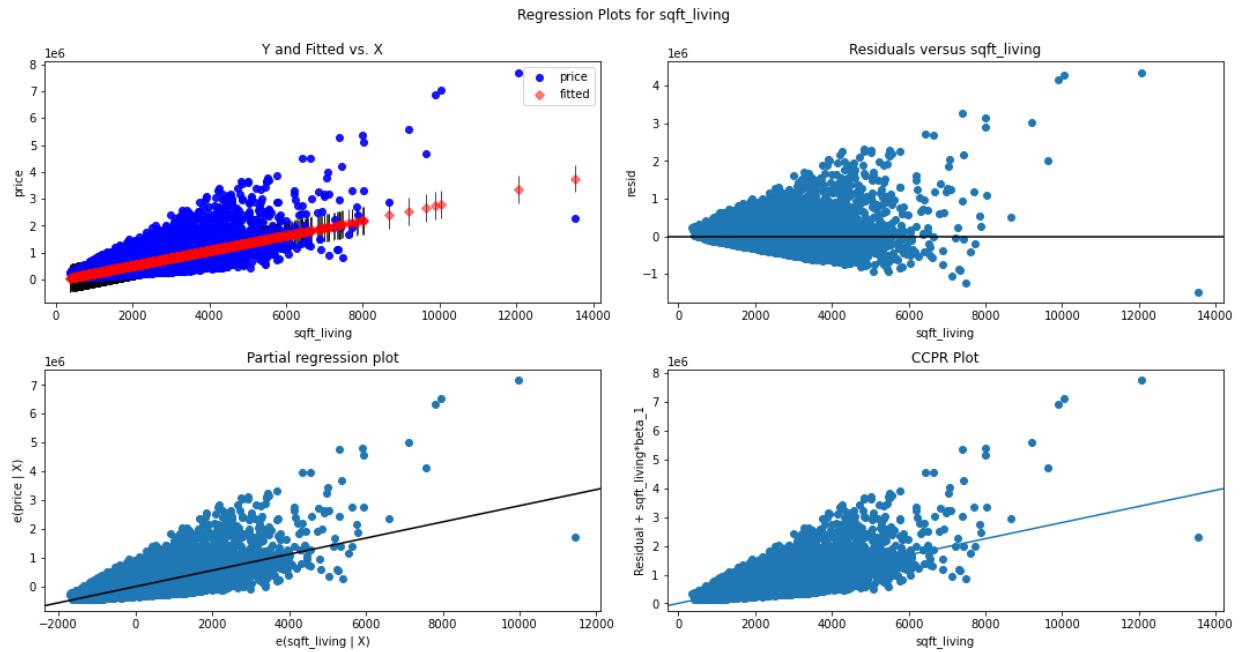
Out[12]: OLS Regression Results

<b>Dep. Variable:</b>	price	<b>R-squared:</b>	0.493			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.493			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	2.097e+04			
<b>Date:</b>	Sun, 21 Nov 2021	<b>Prob (F-statistic):</b>	0.00			
<b>Time:</b>	15:57:45	<b>Log-Likelihood:</b>	-3.0006e+05			
<b>No. Observations:</b>	21597	<b>AIC:</b>	6.001e+05			
<b>Df Residuals:</b>	21595	<b>BIC:</b>	6.001e+05			
<b>Df Model:</b>	1					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	-4.399e+04	4410.023	-9.975	0.000	-5.26e+04	-3.53e+04
<b>sqft_living</b>	280.8630	1.939	144.819	0.000	277.062	284.664
<b>Omnibus:</b>	14801.942	<b>Durbin-Watson:</b>		1.982		
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>		542662.604		
<b>Skew:</b>	2.820	<b>Prob(JB):</b>		0.00		
<b>Kurtosis:</b>	26.901	<b>Cond. No.</b>		5.63e+03		

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 5.63e+03. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [13]: fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, 'sqft_living', fig=fig);
```



Plots show heteroscedasticity.

```
In [14]: f = 'price~distance_from_bellevue'
model = ols(f, df).fit()
model.summary()
```

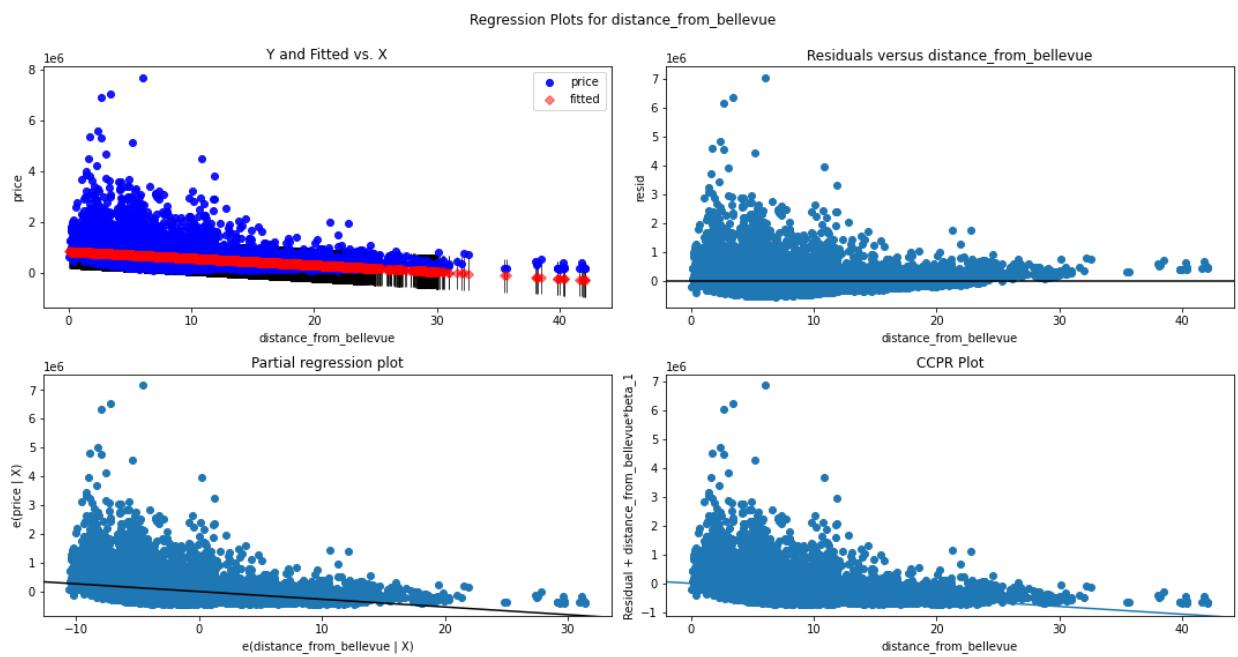
Out[14]: OLS Regression Results

<b>Dep. Variable:</b>	price	<b>R-squared:</b>	0.158			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.158			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	4062.			
<b>Date:</b>	Sun, 21 Nov 2021	<b>Prob (F-statistic):</b>	0.00			
<b>Time:</b>	15:57:47	<b>Log-Likelihood:</b>	-3.0553e+05			
<b>No. Observations:</b>	21597	<b>AIC:</b>	6.111e+05			
<b>Df Residuals:</b>	21595	<b>BIC:</b>	6.111e+05			
<b>Df Model:</b>	1					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	8.254e+05	5026.642	164.200	0.000	8.16e+05	8.35e+05
<b>distance_from_bellevue</b>	-2.677e+04	420.035	-63.734	0.000	-2.76e+04	-2.59e+04
<b>Omnibus:</b>	20006.141	<b>Durbin-Watson:</b>	1.969			
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	1572480.456			
<b>Skew:</b>	4.233	<b>Prob(JB):</b>	0.00			
<b>Kurtosis:</b>	43.936	<b>Cond. No.</b>	26.4			

Notes:

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

```
In [15]: fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "distance_from_bellevue", fig=fig)
plt.show()
```



In [16]: # Normalizing distribution using log transformation

```
df0 = df.copy()
df0['price_log'] = np.log(df0['price'])
df0['sqft_living_log'] = np.log(df0['sqft_living'])
df0 = df0.drop(['price', 'sqft_living'], axis=1)

# OLS between 'price_log' and 'sqft_living_log'

f = 'price_log~sqft_living_log'
model = ols(f, df0).fit()
model.summary()
```

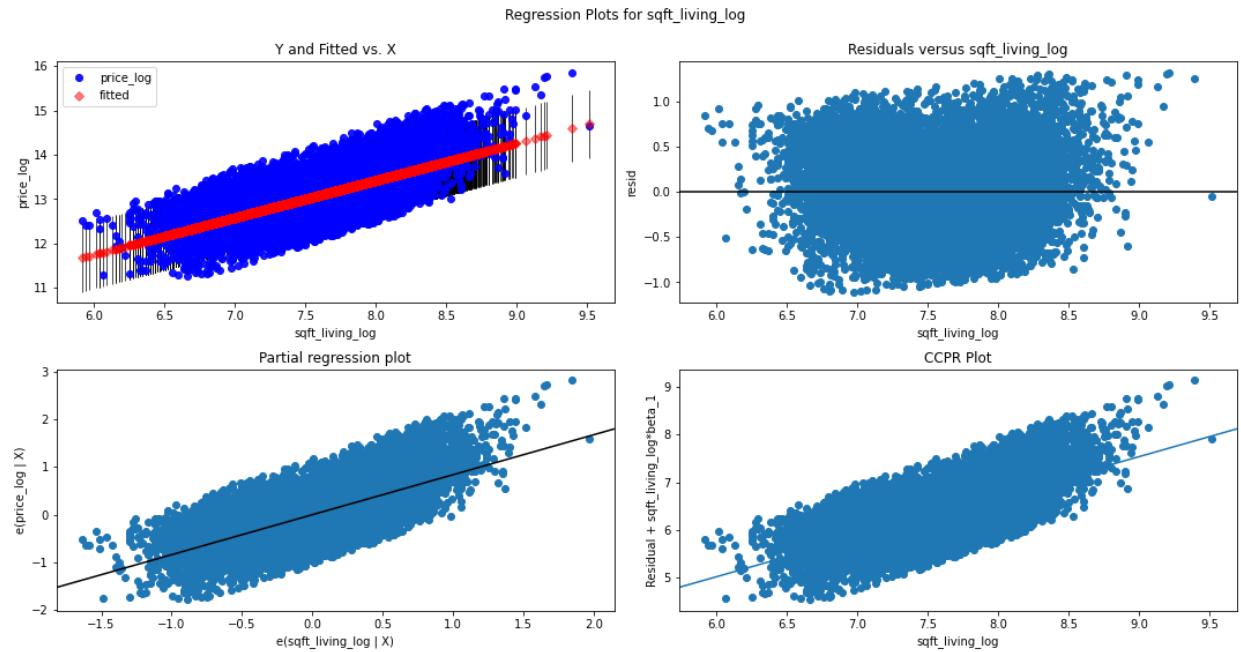
Out[16]: OLS Regression Results

<b>Dep. Variable:</b>	price_log	<b>R-squared:</b>	0.455			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.455			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	1.805e+04			
<b>Date:</b>	Sun, 21 Nov 2021	<b>Prob (F-statistic):</b>	0.00			
<b>Time:</b>	15:57:49	<b>Log-Likelihood:</b>	-10231.			
<b>No. Observations:</b>	21597	<b>AIC:</b>	2.047e+04			
<b>Df Residuals:</b>	21595	<b>BIC:</b>	2.048e+04			
<b>Df Model:</b>	1					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	6.7234	0.047	142.612	0.000	6.631	6.816
<b>sqft_living_log</b>	0.8376	0.006	134.368	0.000	0.825	0.850
<b>Omnibus:</b>	123.577	<b>Durbin-Watson:</b>	1.977			
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	114.096			
<b>Skew:</b>	0.143	<b>Prob(JB):</b>	1.68e-25			
<b>Kurtosis:</b>	2.787	<b>Cond. No.</b>	137.			

Notes:

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

```
In [17]: fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, 'sqft_living_log', fig=fig);
```



When 'price' and 'sqft\_living' undergo log transformation, they are more normally distributed and more homoscedastic, making them better for modeling.

## Predictive Modeling

### Baseline Model & First Simple Linear Regression Model

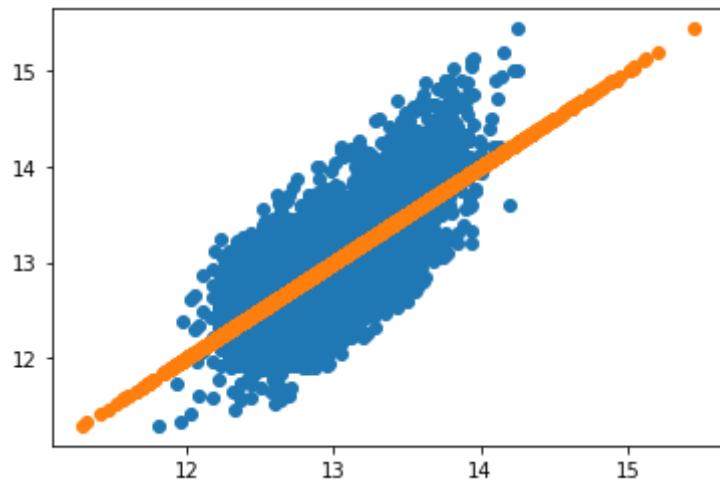
```
In [18]: # df0 has original price & sqft_living removed, has price_log & sqft_living
X = df0[['sqft_living_log']]
y = df0[['price_log']]
X_train, X_test, y_train, y_test = train_test_split(X, y)

# baseline
baseline = DummyRegressor()
baseline.fit(X_train, y_train)
yb_hat_train = baseline.predict(X_train)
yb_hat_test = baseline.predict(X_test)
trainb_rmse = mse(y_train, yb_hat_train, squared=False)
testb_rmse = mse(y_test, yb_hat_test, squared=False)
print('Baseline Train R\u00b2:', baseline.score(X_train, y_train))
print('Baseline Test R\u00b2:', baseline.score(X_test, y_test))
print('Baseline Train RMSE:', trainb_rmse)
print('Baseline Test RMSE:', testb_rmse)
print()

# simple lr
lr = LinearRegression()
lr.fit(X_train, y_train)
y_hat_train = lr.predict(X_train)
y_hat_test = lr.predict(X_test)
train_rmse = mse(y_train, y_hat_train, squared=False)
test_rmse = mse(y_test, y_hat_test, squared=False)
condition = sm.OLS(y_train, sm.add_constant(X_train)).fit().condition_number
print('Simple LR Train R\u00b2:', lr.score(X_train, y_train))
print('Simple LR Test R\u00b2:', lr.score(X_test, y_test))
print('Simple LR Train RMSE:', train_rmse)
print('Simple LR Test RMSE:', test_rmse)
print('Simple LR Condition Number:', condition)
y_test_pred = lr.predict(X_test)
plt.scatter(y_test_pred, y_test)
plt.scatter(y_test, y_test);
```

```
Baseline Train R2: 0.0
Baseline Test R2: -7.387837847616652e-05
Baseline Train RMSE: 0.5265623896745106
Baseline Test RMSE: 0.5264882503793875

Simple LR Train R2: 0.45750692680679916
Simple LR Test R2: 0.44887325162487945
Simple LR Train RMSE: 0.38783494222759424
Simple LR Test RMSE: 0.39083944363519657
Simple LR Condition Number: 136.9967999494335
```

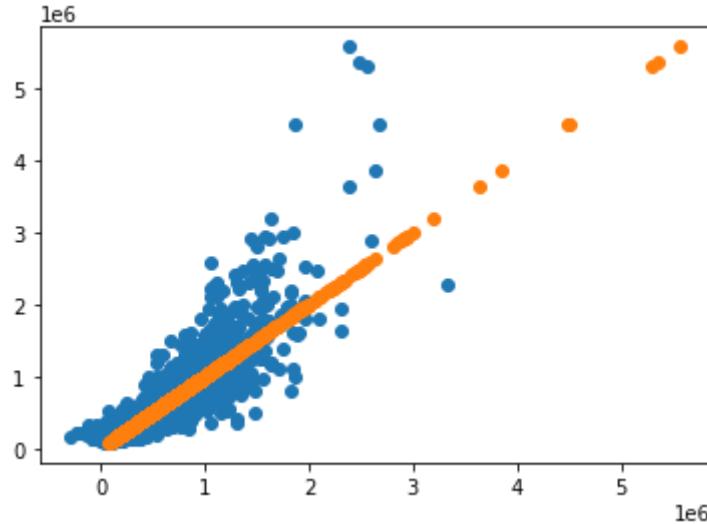


### First Multiple Linear Regression Model

Model with all untouched predictor variables.

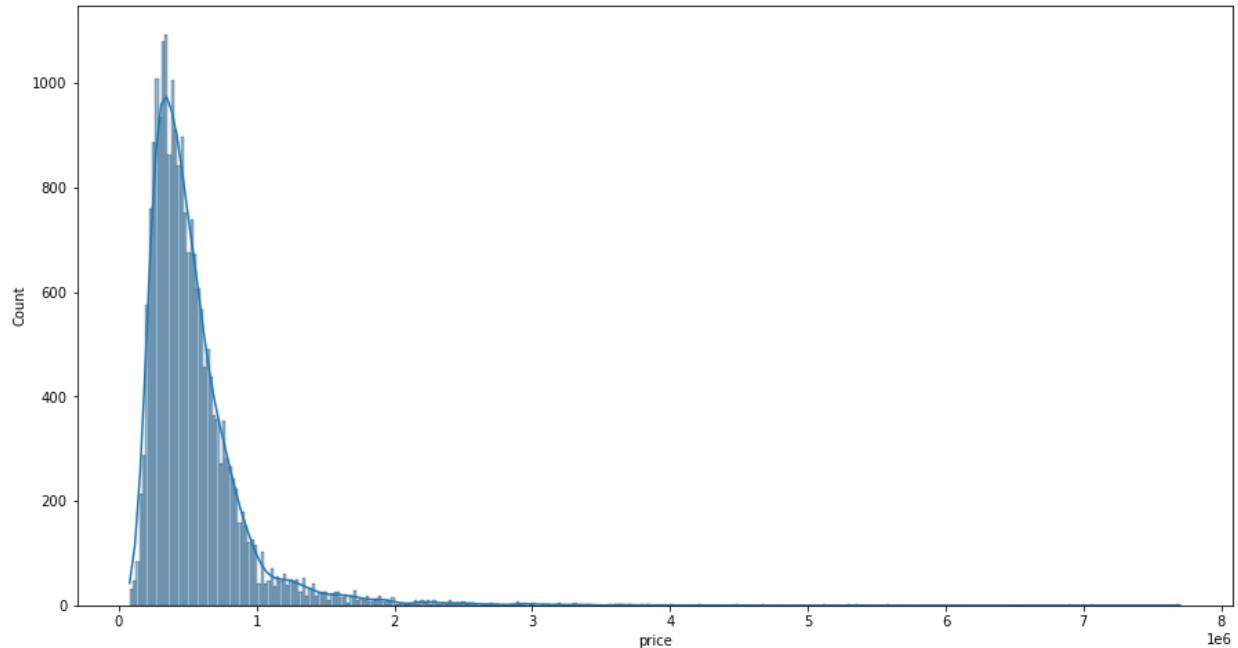
```
In [19]: # df1 = original df
df1 = df.copy()
X1 = df1.drop(['price'], axis=1)
y1 = df1[['price']]
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1)
lr1 = LinearRegression()
lr1.fit(X1_train, y1_train)
y1_hat_train = lr1.predict(X1_train)
y1_hat_test = lr1.predict(X1_test)
train1_rmse = mse(y1_train, y1_hat_train, squared=False)
test1_rmse = mse(y1_test, y1_hat_test, squared=False)
condition1 = sm.OLS(y1_train, sm.add_constant(X1_train)).fit().condition_number
print('LR1 Train R\u00b2:', lr1.score(X1_train, y1_train))
print('LR1 Test R\u00b2:', lr1.score(X1_test, y1_test))
print('LR1 Train RMSE:', train1_rmse)
print('LR1 Test RMSE:', test1_rmse)
print('LR1 Condition Number:', condition1)
y1_test_pred = lr1.predict(X1_test)
plt.scatter(y1_test_pred, y1_test)
plt.scatter(y1_test, y1_test);
```

```
LR1 Train R\u00b2: 0.7285105767313159
LR1 Test R\u00b2: 0.711381272939767
LR1 Train RMSE: 191682.71507876235
LR1 Test RMSE: 196513.74301967124
LR1 Condition Number: 1.006535730857547e+16
```

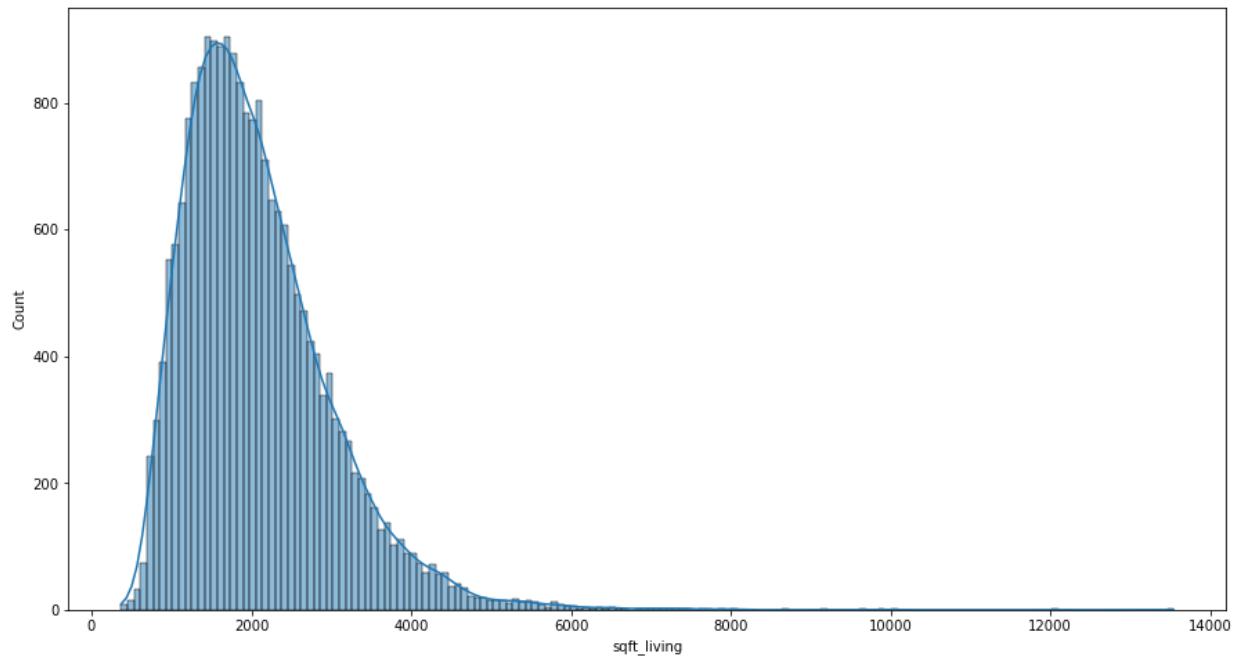


There are outliers in our dataset affecting our model.

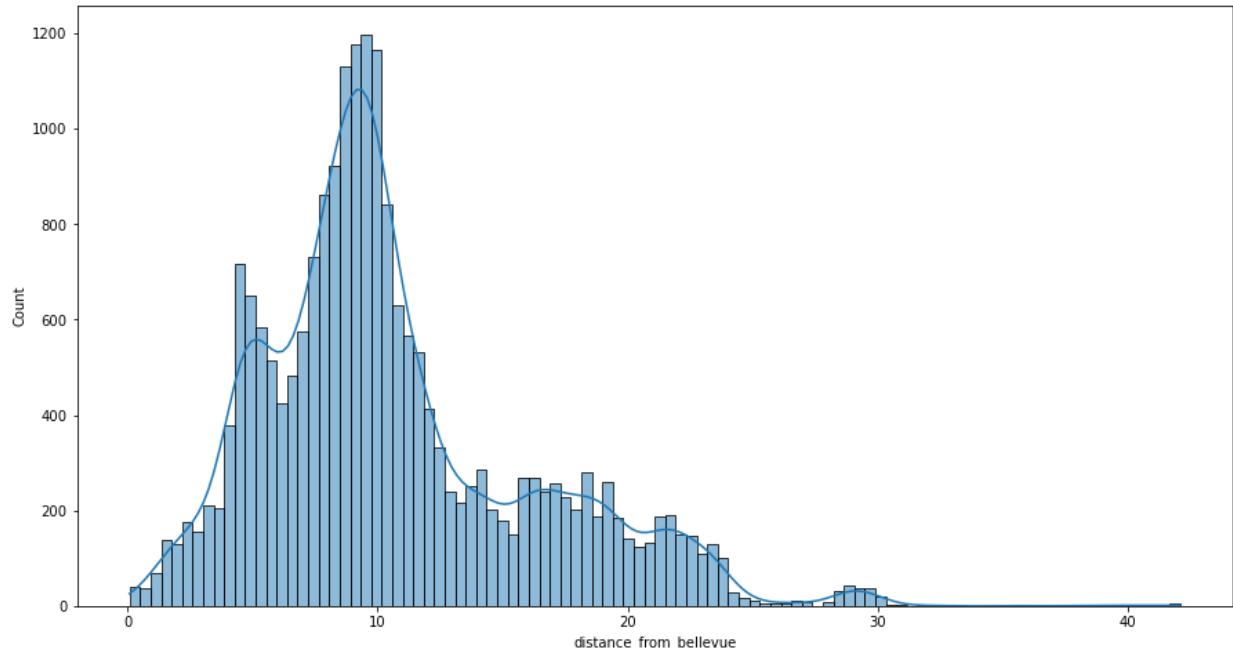
```
In [20]: plt.figure(figsize=(15,8))
sns.histplot(df['price'], kde=True);
```



```
In [21]: plt.figure(figsize=(15,8))
sns.histplot(df['sqft_living'], kde=True);
```



```
In [22]: plt.figure(figsize =(15,8))
sns.histplot(df['distance_from_bellevue'], kde=True);
```



The three histograms above show heavy right skew.

### Second Multiple Linear Regression Model

Model with price, sqft\_living, distance\_from\_bellevue, and other continuous variable outliers removed.

```
In [23]: price_low = df1["price"].quantile(0.023)
price_hi = df1["price"].quantile(0.977)

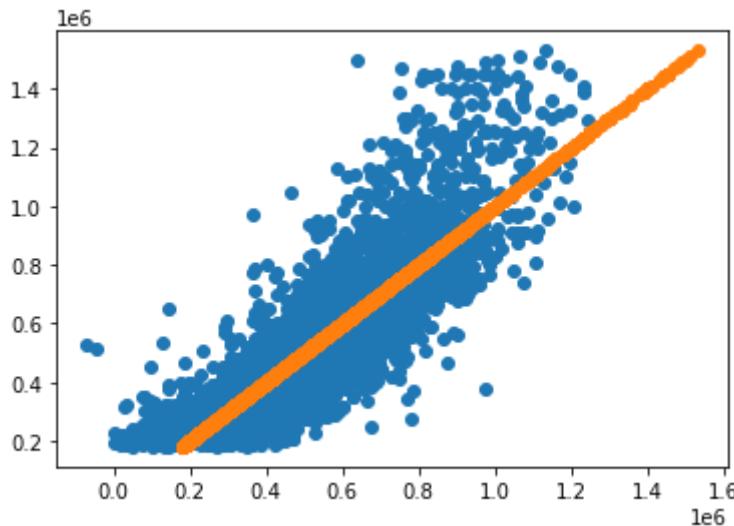
sqft_low = df1['sqft_living'].quantile(0.023)
sqft_hi = df1['sqft_living'].quantile(0.977)

distance_hi = df1['distance_from_bellevue'].quantile(0.99)

df2 = df1.copy()
df2 = df2[(df2["price"] < price_hi) & (df2["price"] > price_low)]
df2 = df2[(df2['sqft_living'] < sqft_hi) & (df2['sqft_living'] > sqft_low)]
df2 = df2[(df2['distance_from_bellevue'] < distance_hi)]
```

```
In [24]: X2 = df2.drop(['price'], axis=1)
y2 = df2[['price']]
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2)
lr2 = LinearRegression()
lr2.fit(X2_train, y2_train)
y2_hat_train = lr2.predict(X2_train)
y2_hat_test = lr2.predict(X2_test)
train2_rmse = mse(y2_train, y2_hat_train, squared=False)
test2_rmse = mse(y2_test, y2_hat_test, squared=False)
condition2 = sm.OLS(y2_train, sm.add_constant(X2_train)).fit().condition_number
print('LR2 Train R\u00b2:', lr2.score(X2_train, y2_train))
print('LR2 Test R\u00b2:', lr2.score(X2_train, y2_train))
print('LR2 Train RMSE:', train2_rmse)
print('LR2 Test RMSE:', test2_rmse)
print('LR2 Condition Number:', condition2)
y2_test_pred = lr2.predict(X2_test)
plt.scatter(y2_test_pred, y2_test)
plt.scatter(y2_test, y2_test);
```

LR2 Train R<sup>2</sup>: 0.722383328302759  
 LR2 Test R<sup>2</sup>: 0.722383328302759  
 LR2 Train RMSE: 125663.22380378452  
 LR2 Test RMSE: 126432.92682749794  
 LR2 Condition Number: 1.0042033407253232e+16



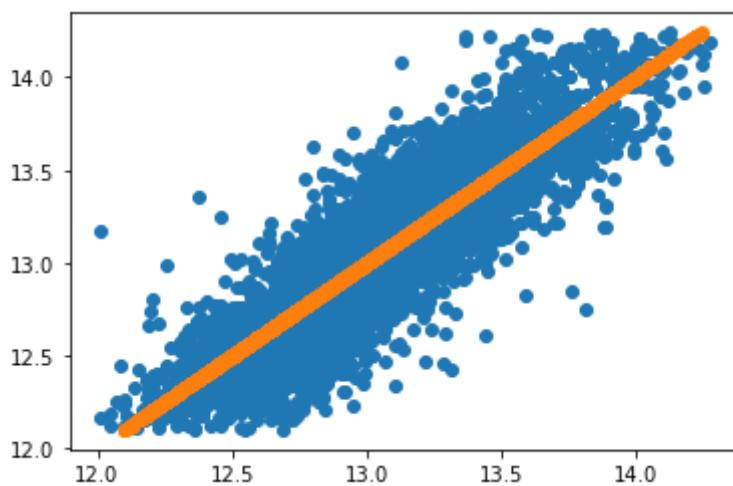
### Third Multiple Linear Regression Model

Second model with log transformed price, sqft\_living, and distance\_from\_bellvue, and other continuous variables.

```
In [25]: df3 = df2.copy()
df3['price_log'] = np.log(df2['price'])
df3['sqft_living_log'] = np.log(df3['sqft_living'])
df3['distance_from_bellevue_log'] = np.log(df3['distance_from_bellevue'])
df3['sqft_lot_log'] = np.log(df2['sqft_lot'])
df3 = df3.drop(['price', 'sqft_living', 'distance_from_bellevue', 'sqft_lot'])
```

```
In [26]: X3 = df3.drop(['price_log'], axis=1)
y3 = df3[['price_log']]
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3)
lr3 = LinearRegression()
lr3.fit(X3_train, y3_train)
y3_hat_train = lr3.predict(X3_train)
y3_hat_test = lr3.predict(X3_test)
train3_rmse = mse(y3_train, y3_hat_train, squared=False)
test3_rmse = mse(y3_test, y3_hat_test, squared=False)
condition3 = sm.OLS(y3_train, sm.add_constant(X3_train)).fit().condition_number
print('LR3 Train R\u00b2:', lr3.score(X3_train, y3_train))
print('LR3 Test R\u00b2:', lr3.score(X3_test, y3_test))
print('LR3 Train RMSE:', train3_rmse)
print('LR3 Test RMSE:', test3_rmse)
print('LR3 Condition Number:', condition3)
y3_test_pred = lr3.predict(X3_test)
plt.scatter(y3_test_pred, y3_test)
plt.scatter(y3_test, y3_test);
```

LR3 Train R<sup>2</sup>: 0.7704109825727867  
 LR3 Test R<sup>2</sup>: 0.7685315730342026  
 LR3 Train RMSE: 0.21084634524208756  
 LR3 Test RMSE: 0.21206417939799654  
 LR3 Condition Number: 1.00129709682521e+16



#### Fourth Multiple Linear Regression Model

Third model with multicollinear variables removed.

```
In [27]: corr = df3.corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr[mask] = np.nan
(corr
    .style
    .background_gradient(cmap='mako', axis=None, vmin=-1, vmax=1)
    .highlight_null(null_color='#f1f1f1') # Color NaNs grey
    .set_precision(2))
```

Out[27]:

	<b>bedrooms</b>	<b>bathrooms</b>	<b>floors</b>	<b>waterfront</b>	<b>grade</b>	<b>sqft_above</b>	<b>sqft_basement</b>
<b>bedrooms</b>	nan	nan	nan	nan	nan	nan	nan
<b>bathrooms</b>	0.47	nan	nan	nan	nan	nan	nan
<b>floors</b>	0.13	0.49	nan	nan	nan	nan	nan
<b>waterfront</b>	-0.03	0.00	-0.00	nan	nan	nan	nan
<b>grade</b>	0.27	0.58	0.44	0.00	nan	nan	nan
<b>sqft_above</b>	0.44	0.61	0.52	-0.01	0.69	nan	nan
<b>sqft_basement</b>	0.26	0.19	-0.33	0.03	0.04	-0.22	nan
<b>yr_built</b>	0.11	0.51	0.49	-0.04	0.45	0.43	-1
<b>yr_renovated</b>	0.01	0.04	-0.00	0.08	-0.00	0.00	1
<b>zipcode</b>	-0.13	-0.19	-0.05	0.05	-0.17	-0.26	1
<b>lat</b>	-0.04	-0.02	0.02	-0.04	0.08	-0.05	1
<b>long</b>	0.13	0.23	0.12	-0.05	0.21	0.38	-1
<b>sqft_living15</b>	0.35	0.50	0.24	0.02	0.67	0.70	1
<b>sqft_lot15</b>	0.01	0.05	-0.04	0.04	0.09	0.17	-1
<b>yr_sold</b>	-0.01	-0.03	-0.02	-0.01	-0.03	-0.03	-1
<b>house_age</b>	-0.11	-0.51	-0.49	0.04	-0.45	-0.43	1
<b>cond_avg</b>	-0.01	0.20	0.33	-0.01	0.21	0.21	-1
<b>cond_fair</b>	-0.02	-0.06	-0.04	-0.00	-0.06	-0.04	-1
<b>cond_good</b>	-0.00	-0.18	-0.27	0.01	-0.15	-0.16	1
<b>cond_poor</b>	-0.02	-0.03	-0.02	0.03	-0.04	-0.02	-1
<b>cond_verygood</b>	0.03	-0.04	-0.13	0.01	-0.10	-0.11	1
<b>view_avg</b>	0.03	0.06	-0.01	0.01	0.10	0.04	1
<b>view_excellent</b>	0.00	0.04	-0.00	0.48	0.06	0.02	1
<b>view_fair</b>	0.01	0.02	-0.03	-0.00	0.04	0.01	1
<b>view_good</b>	0.02	0.07	-0.00	0.04	0.11	0.05	1
<b>view_none</b>	-0.04	-0.10	0.02	-0.19	-0.16	-0.06	-1
<b>price_log</b>	0.26	0.43	0.26	0.08	0.62	0.50	1
<b>sqft_living_log</b>	0.59	0.71	0.32	0.01	0.67	0.81	1

	bedrooms	bathrooms	floors	waterfront	grade	sqft_above	sqft_basem
<b>distance_from_bellevue_log</b>	-0.06	-0.02	0.02	0.03	-0.12	0.03	-0.1
<b>sqft_lot_log</b>	0.17	0.03	-0.29	0.06	0.13	0.29	0.1

```
In [28]: df4 = df3[['price_log', 'sqft_living_log', 'distance_from_bellevue_log', 'sqft_lot_log', 'waterfront', 'yr_renovated', 'house_age', 'view_none']]
```

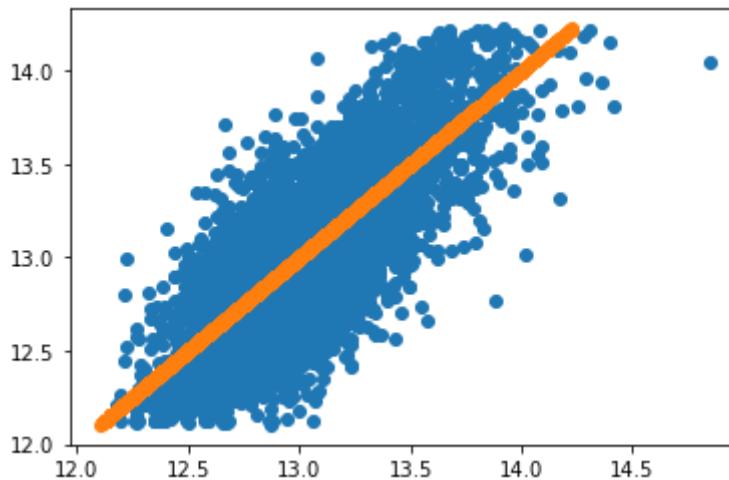
```
In [29]: corr = df4.corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr[mask] = np.nan
(corr
    .style
    .background_gradient(cmap='mako', axis=None, vmin=-1, vmax=1)
    .highlight_null(null_color='#f1f1f1') # Color NaNs grey
    .set_precision(2))
```

Out[29]:

	price_log	sqft_living_log	distance_from_bellevue_log	sqft_lot_log	waterfront
<b>price_log</b>	nan	nan		nan	nan
<b>sqft_living_log</b>	0.59	nan		nan	nan
<b>distance_from_bellevue_log</b>	-0.49	-0.05		nan	nan
<b>sqft_lot_log</b>	0.08	0.30		0.18	nan
<b>waterfront</b>	0.08	0.01		0.03	0.06
<b>yr_renovated</b>	0.10	0.03		-0.05	0.01
<b>house_age</b>	-0.01	-0.31		-0.20	0.04
<b>view_none</b>	-0.27	-0.18		0.05	-0.07

```
In [30]: X4 = df4.drop(['price_log'], axis=1)
y4 = df4[['price_log']]
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y4)
lr4 = LinearRegression()
lr4.fit(X4_train, y4_train)
y4_hat_train = lr4.predict(X4_train)
y4_hat_test = lr4.predict(X4_test)
train4_rmse = mse(y4_train, y4_hat_train, squared=False)
test4_rmse = mse(y4_test, y4_hat_test, squared=False)
condition4 = sm.OLS(y4_train, sm.add_constant(X4_train)).fit().condition_number
print('LR4 Train R\u00b2:', lr4.score(X4_train, y4_train))
print('LR4 Test R\u00b2:', lr4.score(X4_test, y4_test))
print('LR4 Train RMSE:', train4_rmse)
print('LR4 Test RMSE:', test4_rmse)
print('LR4 Condition Number:', condition4)
y4_test_pred = lr4.predict(X4_test)
plt.scatter(y4_test_pred, y4_test)
plt.scatter(y4_test, y4_test);
```

LR4 Train R<sup>2</sup>: 0.5874742076113897  
 LR4 Test R<sup>2</sup>: 0.6002691102718177  
 LR4 Train RMSE: 0.2822019468107538  
 LR4 Test RMSE: 0.28005647401245  
 LR4 Condition Number: 9006.627049169936



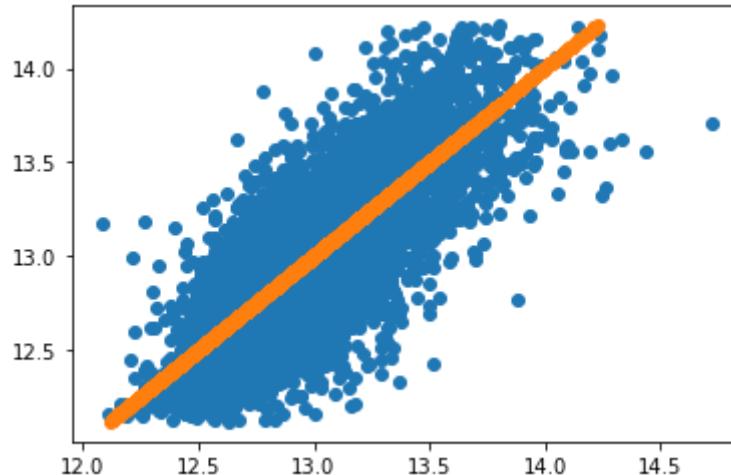
## Fifth Multiple Linear Regression Model

Fourth model with several predictor variables scaled.

```
In [31]: df5 = df4.copy()
col_names = ['sqft_living_log', 'distance_from_bellevue_log', 'sqft_lot_log',
             'yr_renovated', 'house_age']
features = df5[col_names]
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
df5[col_names] = features
```

```
In [32]: X5 = df5.drop(['price_log'], axis=1)
y5 = df5[['price_log']]
X5_train, X5_test, y5_train, y5_test = train_test_split(X5, y5)
lr5 = LinearRegression()
lr5.fit(X5_train, y5_train)
y5_hat_train = lr5.predict(X5_train)
y5_hat_test = lr5.predict(X5_test)
train5_rmse = mse(y5_train, y5_hat_train, squared=False)
test5_rmse = mse(y5_test, y5_hat_test, squared=False)
condition5 = sm.OLS(y5_train, sm.add_constant(X5_train)).fit().condition_number
print('LR5 Train R\u00b2:', lr5.score(X5_train, y5_train))
print('LR5 Test R\u00b2:', lr5.score(X5_test, y5_test))
print('LR5 Train RMSE:', train5_rmse)
print('LR5 Test RMSE:', test5_rmse)
print('LR5 Condition Number:', condition5)
y5_test_pred = lr5.predict(X5_test)
plt.scatter(y5_test_pred, y5_test)
plt.scatter(y5_test, y5_test);
```

```
LR5 Train R2: 0.5922348767146427
LR5 Test R2: 0.585949418367634
LR5 Train RMSE: 0.28285285809494004
LR5 Test RMSE: 0.27807090027595405
LR5 Condition Number: 25.18729505490083
```



```
In [33]: selector = RFE(lr5, n_features_to_select=4)
selector = selector.fit(X5, y5)
print(selector.support_)
display(X5)
```

```
[ True  True False  True False False  True]
```

	sqft_living_log	distance_from_bellvue_log	sqft_lot_log	waterfront	yr_renovated	house_age	
0	-1.315688		-0.597021	-0.361090	0	-0.184258	0.561359
1	0.813688		0.124814	-0.080384	0	5.412837	0.698607
3	0.072443		0.231388	-0.499291	0	-0.184258	0.218238
4	-0.349253		-0.422152	0.043429	0	-0.184258	-0.502317
6	-0.292847		1.503457	-0.148440	0	-0.184258	-0.811126
...	...		...	...	...	...	...
21592	-0.605104		0.055117	-2.180001	0	-0.184258	-1.291496
21593	0.521912		0.091045	-0.328929	0	-0.184258	-1.428745
21594	-1.714298		-1.307050	-1.979851	0	-0.184258	-1.291496
21595	-0.482724		-0.343840	-1.334912	0	-0.184258	-1.085623
21596	-1.714298		-1.306193	-2.236372	0	-0.184258	-1.257184

19733 rows × 7 columns

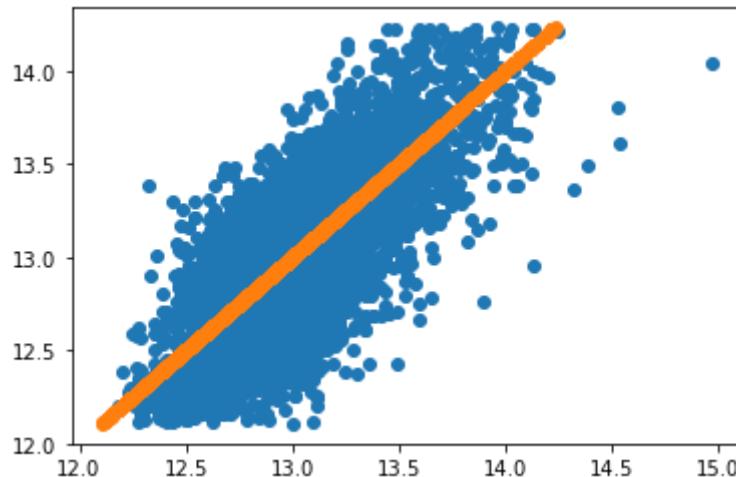
Selector selects sqft\_living\_log, distance\_from\_bellvue\_log, waterfront, & view\_none.

## Sixth Multiple Linear Regression Model

Fifth model using recursive feature elimination.

```
In [34]: df6 = df5.copy()
X6 = df6[['sqft_living_log', 'distance_from_bellevue_log', 'waterfront', 'view']]
y6 = df6[['price_log']]
X6_train, X6_test, y6_train, y6_test = train_test_split(X6, y6)
lr6 = LinearRegression()
lr6.fit(X6_train, y6_train)
y6_hat_train = lr6.predict(X6_train)
y6_hat_test = lr6.predict(X6_test)
train6_rmse = mse(y6_train, y6_hat_train, squared=False)
test6_rmse = mse(y6_test, y6_hat_test, squared=False)
condition6 = sm.OLS(y6_train, sm.add_constant(X6_train)).fit().condition_number
print('LR6 Train R\u00b2:', lr6.score(X6_train, y6_train))
print('LR6 Test R\u00b2:', lr6.score(X6_test, y6_test))
print('LR6 Train RMSE:', train6_rmse)
print('LR6 Test RMSE:', test6_rmse)
print('LR6 Condition Number:', condition6)
y6_test_pred = lr6.predict(X6_test)
plt.scatter(y6_test_pred, y6_test)
plt.scatter(y6_test, y6_test);
```

```
LR6 Train R2: 0.5898812221871109
LR6 Test R2: 0.5716586186734381
LR6 Train RMSE: 0.28307941528112196
LR6 Test RMSE: 0.2846196621614078
LR6 Condition Number: 23.666889119774158
```



## Seventh Multiple Linear Regression Model

Sixth model using stepwise selection to choose significant features.

```
In [35]: def stepwise_selection(X, y,
                           initial_list=[],
                           threshold_in=0.01,
                           threshold_out = 0.05,
                           verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusion
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    See https://en.wikipedia.org/wiki/Stepwise_regression for the details
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]])))
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add {:30} with p-value {:.6}'.format(best_feature, best_pval))
        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included

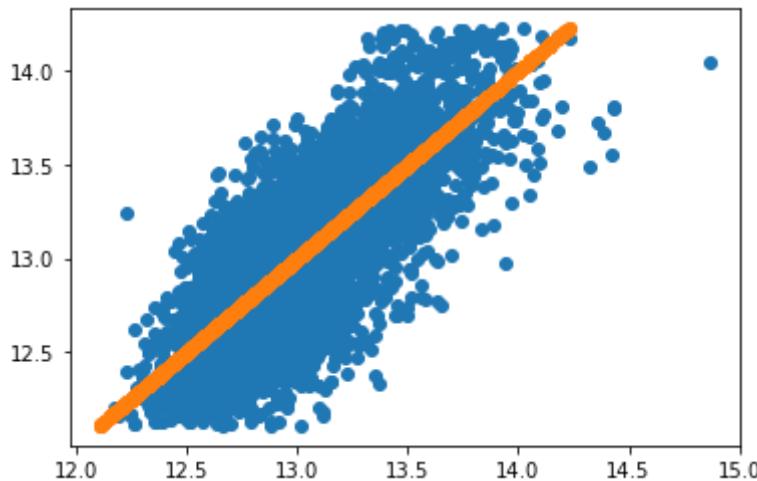
result = stepwise_selection(X5, y5, verbose=True)
print('resulting features:')
print(result)
```

Add view_none	with p-value 0.0
Add sqft_living_log	with p-value 0.0
Add distance_from_bellvue_log	with p-value 0.0

```
Add house_age                         with p-value 1.9158e-40
Add waterfront                        with p-value 8.52867e-30
Add yr_renovated                      with p-value 4.12047e-11
Add sqft_lot_log                      with p-value 6.57435e-09
resulting features:
['view_none', 'sqft_living_log', 'distance_from_bellevue_log', 'house_ag
e', 'waterfront', 'yr_renovated', 'sqft_lot_log']
```

```
In [36]: df7 = df6.copy()
X7 = df7[['distance_from_bellevue_log', 'view_none', 'sqft_living_log',
           'house_age', 'waterfront', 'yr_renovated', 'sqft_lot_log']]
y7 = df7[['price_log']]
X7_train, X7_test, y7_train, y7_test = train_test_split(X7, y7)
lr7 = LinearRegression()
lr7.fit(X7_train, y7_train)
y7_hat_train = lr7.predict(X7_train)
y7_hat_test = lr7.predict(X7_test)
train7_rmse = mse(y7_train, y7_hat_train, squared=False)
test7_rmse = mse(y7_test, y7_hat_test, squared=False)
condition7 = sm.OLS(y7_train, sm.add_constant(X7_train)).fit().condition_nu
print('LR7 Train R\u00b2:', lr7.score(X7_train, y7_train))
print('LR7 Test R\u00b2:', lr7.score(X7_test, y7_test))
print('LR7 Train RMSE:', train7_rmse)
print('LR7 Test RMSE:', test7_rmse)
print('LR7 Condition Number:', condition7)
y7_test_pred = lr7.predict(X7_test)
plt.scatter(y7_test_pred, y7_test)
plt.scatter(y7_test, y7_test);
```

LR7 Train R<sup>2</sup>: 0.5911549647603884  
 LR7 Test R<sup>2</sup>: 0.5895525188149706  
 LR7 Train RMSE: 0.2814919826720074  
 LR7 Test RMSE: 0.28212531959026727  
 LR7 Condition Number: 23.53952434535512



```
In [37]: f = 'price_log~distance_from_bellevue_log+view_none+sqft_living_log+house_a
model = ols(f, df7).fit()
model.summary()'
```

Out[37]: OLS Regression Results

<b>Dep. Variable:</b>	price_log	<b>R-squared:</b>	0.591			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.591			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	4068.			
<b>Date:</b>	Sun, 21 Nov 2021	<b>Prob (F-statistic):</b>	0.00			
<b>Time:</b>	15:57:57	<b>Log-Likelihood:</b>	-2996.1			
<b>No. Observations:</b>	19733	<b>AIC:</b>	6008.			
<b>Df Residuals:</b>	19725	<b>BIC:</b>	6071.			
<b>Df Model:</b>	7					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	13.2288	0.007	1836.990	0.000	13.215	13.243
<b>distance_from_bellevue_log</b>	-0.1925	0.002	-90.546	0.000	-0.197	-0.188
<b>view_none</b>	-0.2054	0.008	-27.270	0.000	-0.220	-0.191
<b>sqft_living_log</b>	0.2519	0.002	107.418	0.000	0.247	0.257
<b>house_age</b>	0.0283	0.002	12.341	0.000	0.024	0.033
<b>waterfront</b>	0.3898	0.035	11.208	0.000	0.322	0.458
<b>yr_renovated</b>	0.0132	0.002	6.403	0.000	0.009	0.017
<b>sqft_lot_log</b>	-0.0127	0.002	-5.804	0.000	-0.017	-0.008
<b>Omnibus:</b>	23.650	<b>Durbin-Watson:</b>	1.977			
<b>Prob(Omnibus):</b>	0.000	<b>Jarque-Bera (JB):</b>	23.806			
<b>Skew:</b>	-0.079	<b>Prob(JB):</b>	6.77e-06			
<b>Kurtosis:</b>	3.064	<b>Cond. No.</b>	23.8			

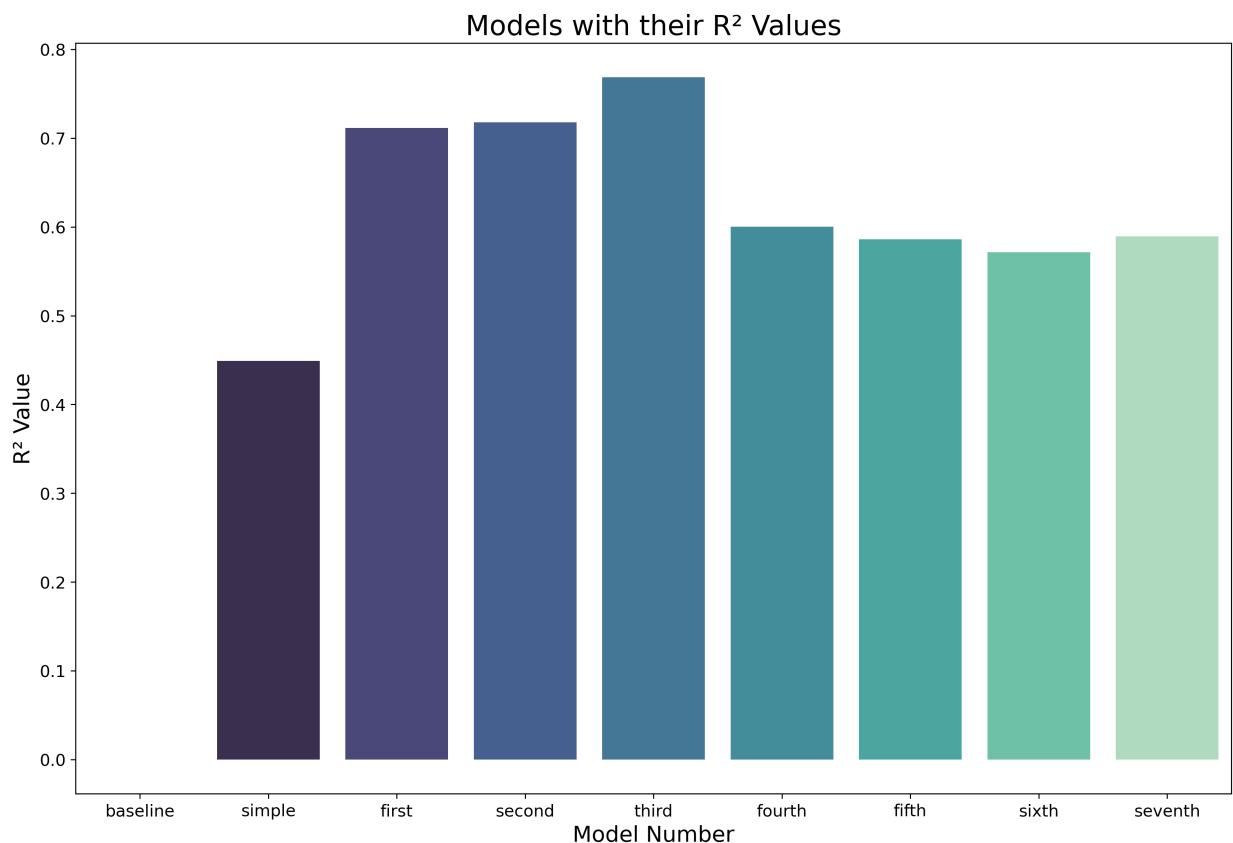
Notes:

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

## Model Visualizations

```
In [38]: baseline = baseline.score(X_test, y_test)
simple = lr.score(X_test, y_test)
first = lr1.score(X1_test, y1_test)
second = lr2.score(X2_test, y2_test)
third = lr3.score(X3_test, y3_test)
fourth = lr4.score(X4_test, y4_test)
fifth = lr5.score(X5_test, y5_test)
sixth = lr6.score(X6_test, y6_test)
seventh = lr7.score(X7_test, y7_test)

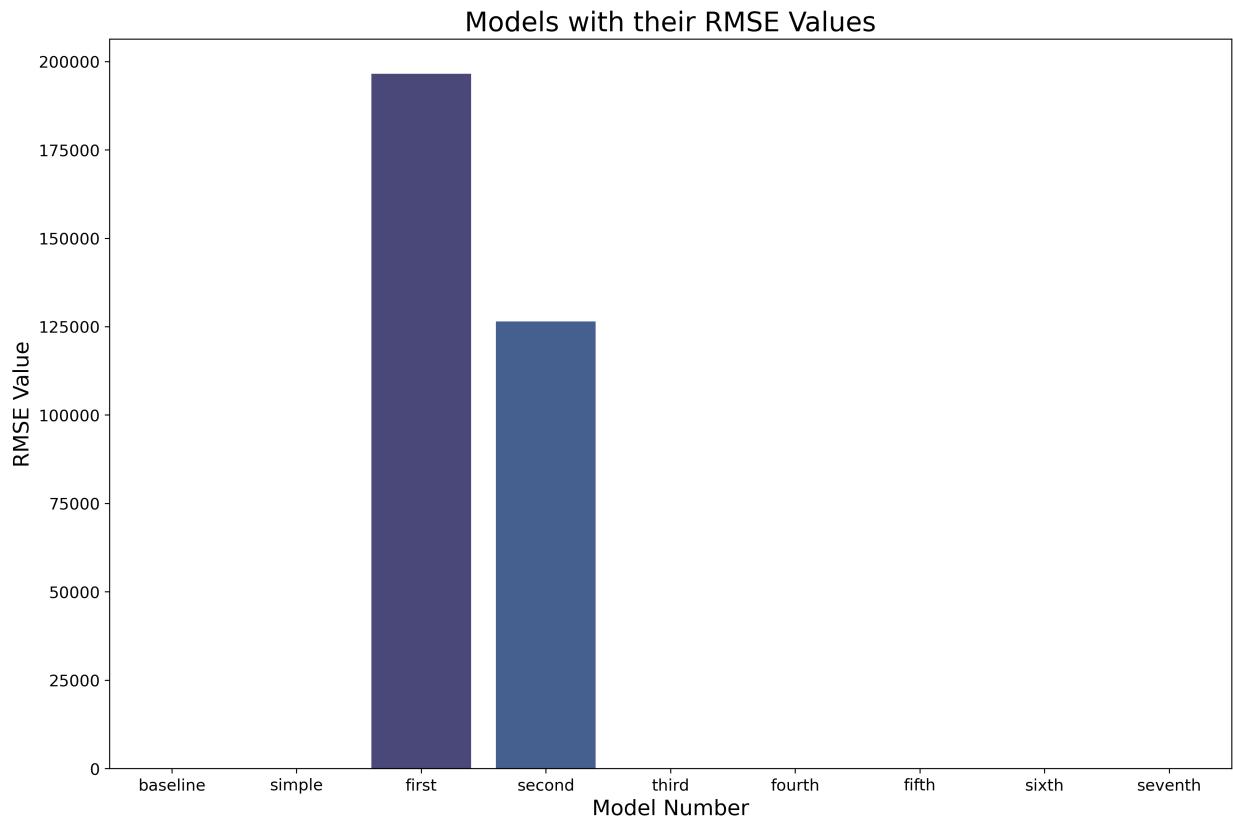
barchart = pd.DataFrame({'Model':['baseline', 'simple', 'first', 'second',
                                  'third', 'fourth', 'fifth', 'sixth', 'seven
                                  'R\u00b2':[baseline, simple, first, second, third,
plt.figure(figsize=(15,10), dpi=300)
ax = sns.barplot(x=barchart['Model'], y=barchart['R\u00b2'], palette="mako")
plt.title("Models with their R\u00b2 Values", fontsize=20)
ax.set_xlabel('Model Number', fontsize=16)
ax.set_ylabel('R\u00b2 Value', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12);
```



Bar chart showing R<sup>2</sup> values of all of our models.

```
In [39]: rmse_b = testb_rmse
rmse_0 = test_rmse
rmse_1 = test1_rmse
rmse_2 = test2_rmse
rmse_3 = test3_rmse
rmse_4 = test4_rmse
rmse_5 = test5_rmse
rmse_6 = test6_rmse
rmse_7 = test7_rmse

barchart = pd.DataFrame({'Model':['baseline', 'simple', 'first', 'second',
                                  'third', 'fourth', 'fifth', 'sixth', 'sev
                                  'RMSE':[rmse_b, rmse_0, rmse_1, rmse_2, rmse_3, rm
plt.figure(figsize=(15,10), dpi=300)
ax = sns.barplot(x=barchart['Model'], y=barchart['RMSE'], palette="mako")
plt.title("Models with their RMSE Values", fontsize=20)
ax.set_xlabel('Model Number', fontsize=16)
ax.set_ylabel('RMSE Value', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12);
```

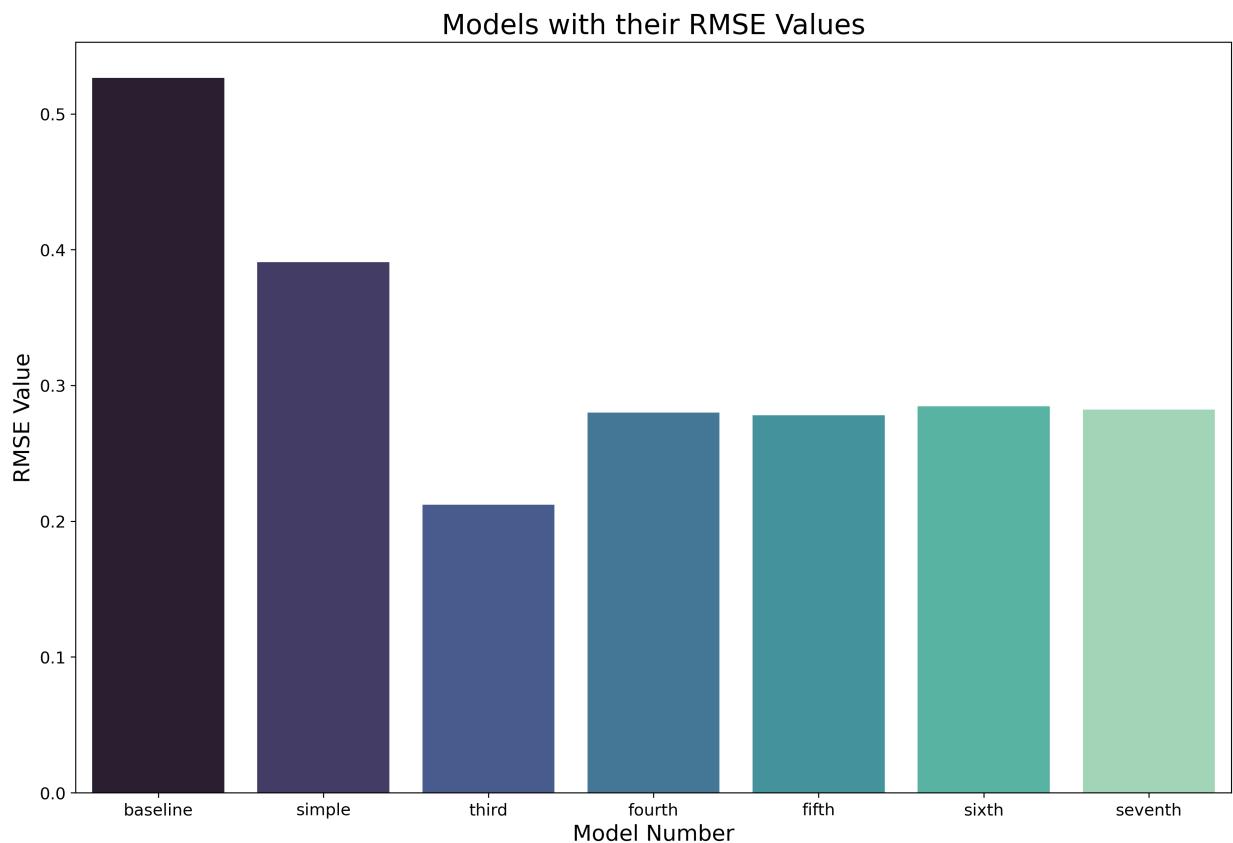


Bar chart showing RMSE values of all of our models.

```
In [40]: rmse_b = testb_rmse
rmse_0 = test_rmse
#rmse_1 = test1_rmse
#rmse_2 = test2_rmse
rmse_3 = test3_rmse
rmse_4 = test4_rmse
rmse_5 = test5_rmse
rmse_6 = test6_rmse
rmse_7 = test7_rmse

barchart = pd.DataFrame({'Model':['baseline', 'simple', 'third', 'fourth',
                                  'RMSE':[rmse_b, rmse_0, rmse_3, rmse_4, rmse_5, rmse_6, rmse_7]})

plt.figure(figsize=(15,10), dpi=300)
ax = sns.barplot(x=barchart['Model'], y=barchart['RMSE'], palette="mako")
plt.title("Models with their RMSE Values", fontsize=20)
ax.set_xlabel('Model Number', fontsize=16)
ax.set_ylabel('RMSE Value', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12);
```



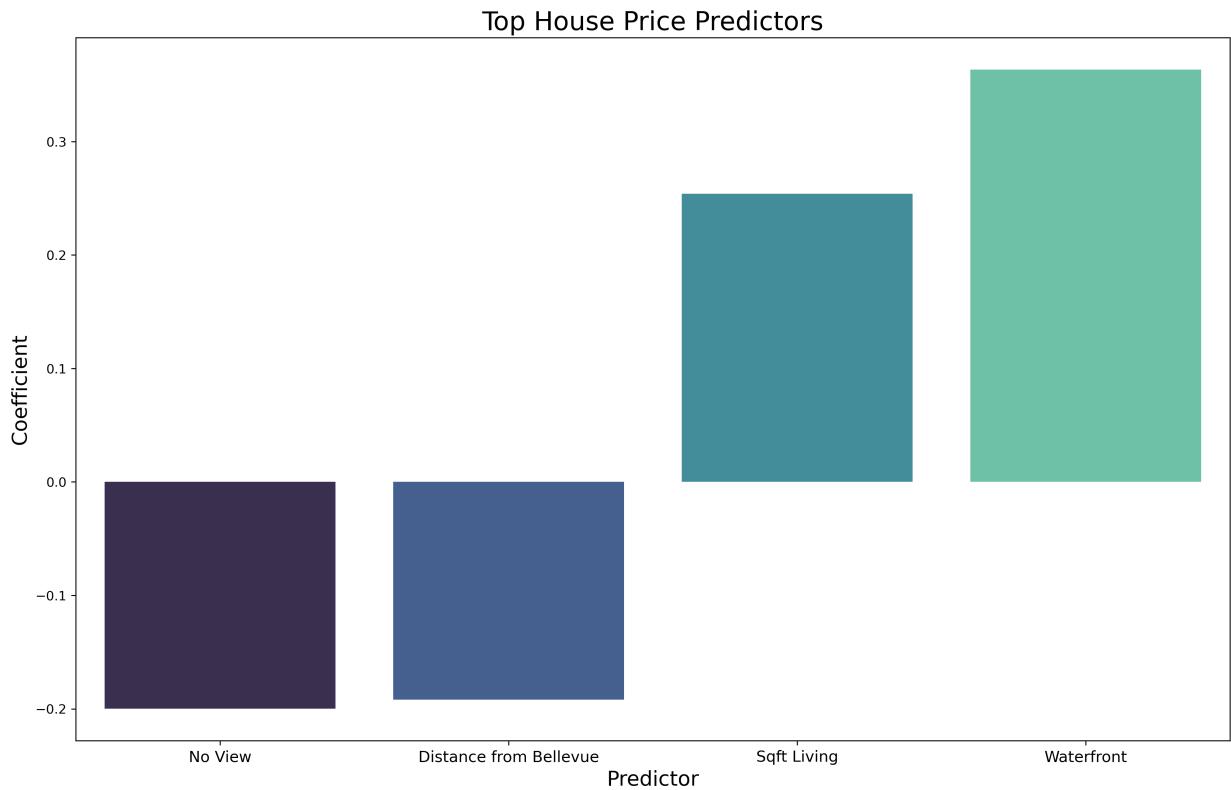
Close up of bar chart showing RMSE values except first and second models.

```
In [41]: lr7.coef_
```

```
Out[41]: array([[-0.19282574, -0.20491918,  0.25154644,  0.02779904,  0.39037694,
   0.01385139, -0.01358739]])
```

```
In [42]: plotdf = pd.DataFrame({'Predictor':['distance_from_bellevue_log', 'view_non',
                                         'Coefficient':[-0.19199832, -0.1999198 , 0.25411448],
                                         plotdf = plotdf.sort_values(by=['Coefficient'])})
```

```
In [43]: fig, ax = plt.subplots(figsize=(16,10), dpi=300)
ax = sns.barplot(x=plotdf['Predictor'], y=plotdf['Coefficient'], palette="magma")
ax.set_title('Top House Price Predictors', fontsize=20)
ax.set_xlabel('Predictor', fontsize=16)
ax.set_ylabel('Coefficient', fontsize=16)
labels = ['No View', 'Distance from Bellevue', 'Sqft Living', 'Waterfront']
ax.set_xticklabels(labels, fontsize=12);
```



Bar chart of best predictor variables.

## Conclusions

After preparing the data, we made seven multiple linear regression models. Our final model was our best performing model with an  $R^2$  value of 0.591, RMSE of 0.282, JB of 23.8, and Condition Number of 23.8. Our strongest predictor variables that will increase house prices are square

footage of the house and whether the home is located on a waterfront. The strongest predictors that will decrease cost are homes with no view and being farther located from the city of Bellevue.

Through multiple iterations of our model, we came to the conclusion that linear regression is not the best method to make a predictive model with this dataset. Linear regression is ill suited for a dataset with many categorical variables, as is the case with this dataset.

Our next steps would include gathering more data such as more recent home sales and expanding beyond single family homes into condos and apartments. We would also explore more complex modeling algorithms.