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/li
b/python3.8/site-packages (2.2.0)
Requirement already satisfied: geographiclib<2,>=1.49 in /opt/anaconda3/e
nvs/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
```

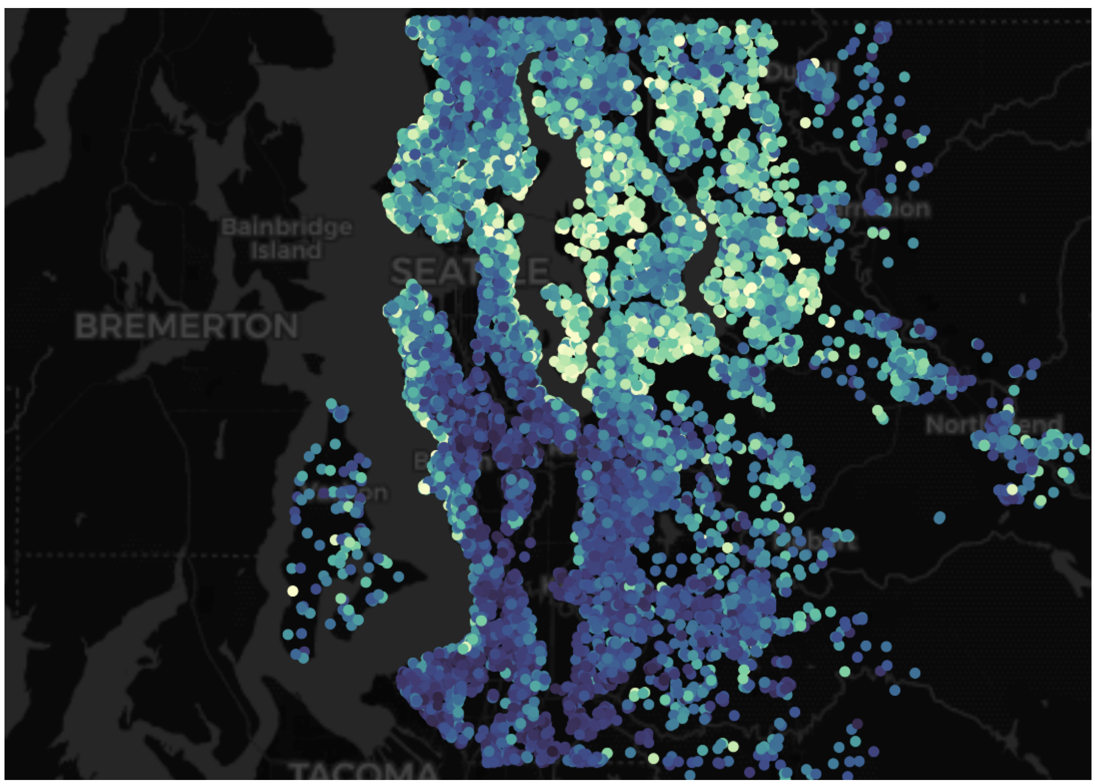
```
None
```

Cleaning & 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]:

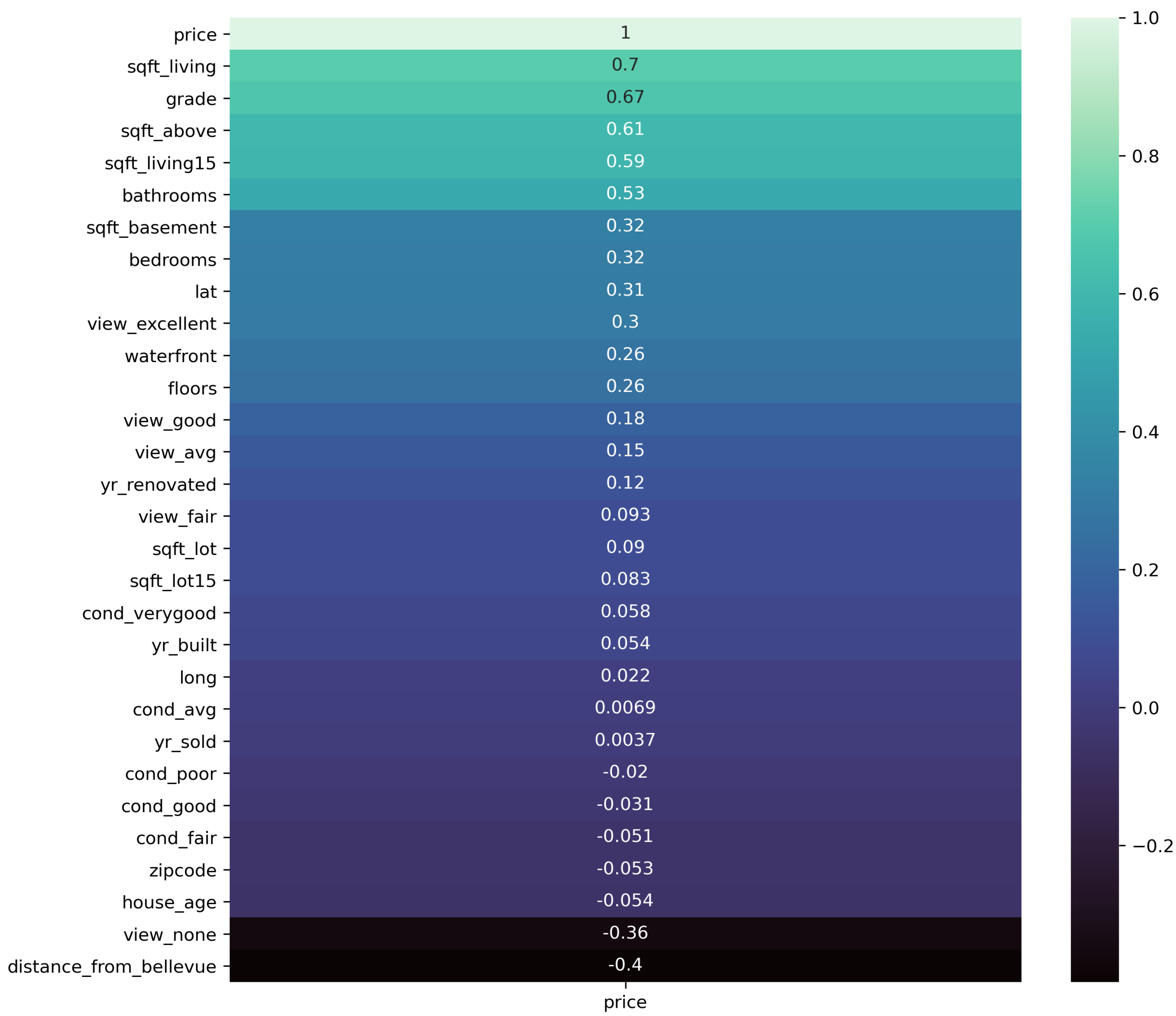
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	grade
price	nan	nan	nan	nan	nan	nan	nan	nan
bedrooms	0.32	nan	nan	nan	nan	nan	nan	nan
bathrooms	0.53	0.53	nan	nan	nan	nan	nan	nan
sqft_living	0.70	0.59	0.76	nan	nan	nan	nan	nan
sqft_lot	0.09	0.03	0.09	0.17	nan	nan	nan	nan
floors	0.26	0.18	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
view_fair	0.09	0.02	0.04	0.07	-0.01	-0.02	-0.01	0.05
view_good	0.18	0.05	0.11	0.16	0.07	0.02	0.04	0.14
view_none	-0.36	-0.08	-0.18	-0.27	-0.07	-0.02	-0.25	-0.24
distance_from_bellevue	-0.40	-0.06	-0.06	-0.11	0.18	-0.03	-0.01	-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);
```



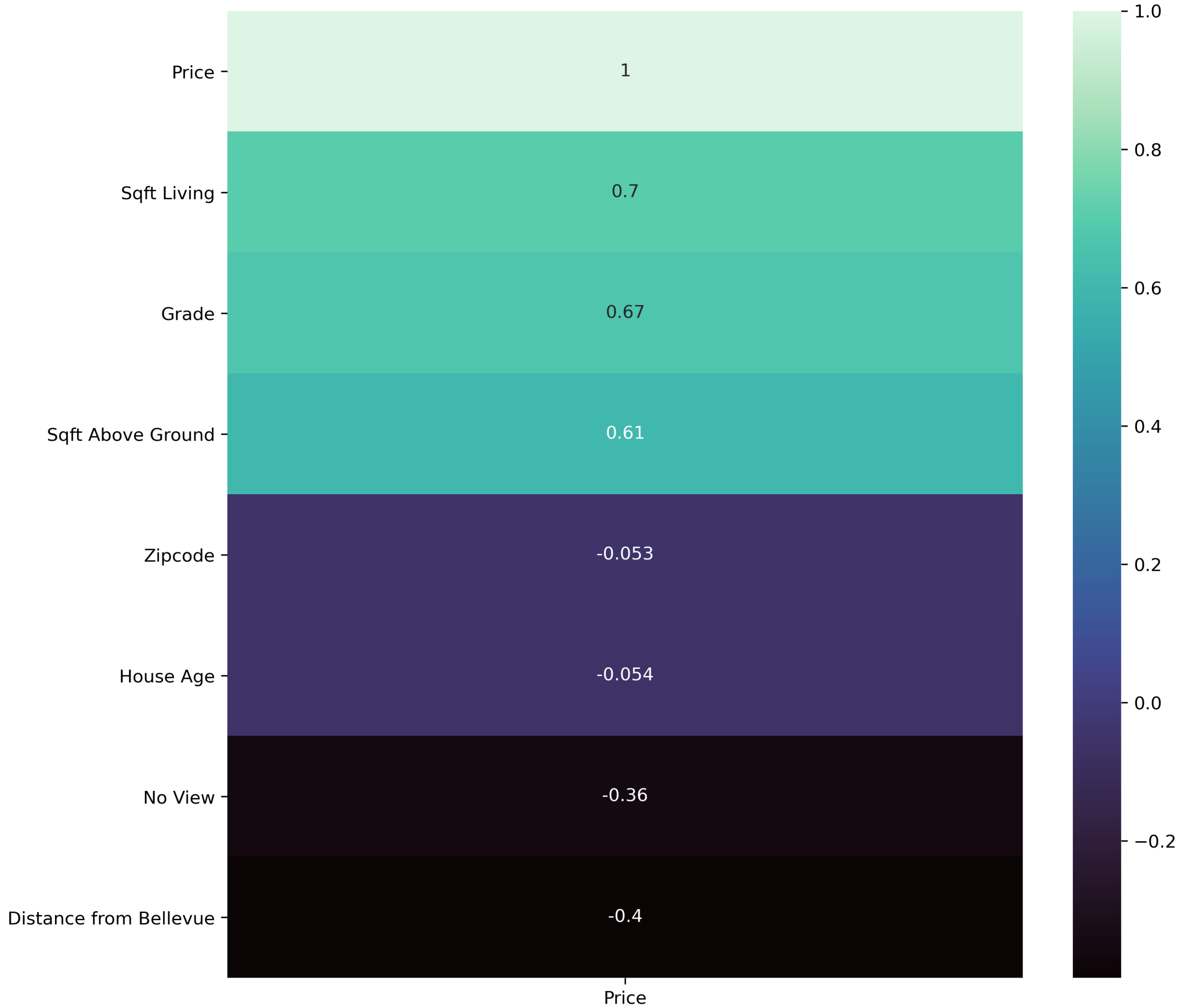
In [8]:
```
heatmap = df[['price','sqft_living','grade','sqft_above','distance_from_bel
heatmap
```

Out[8]:

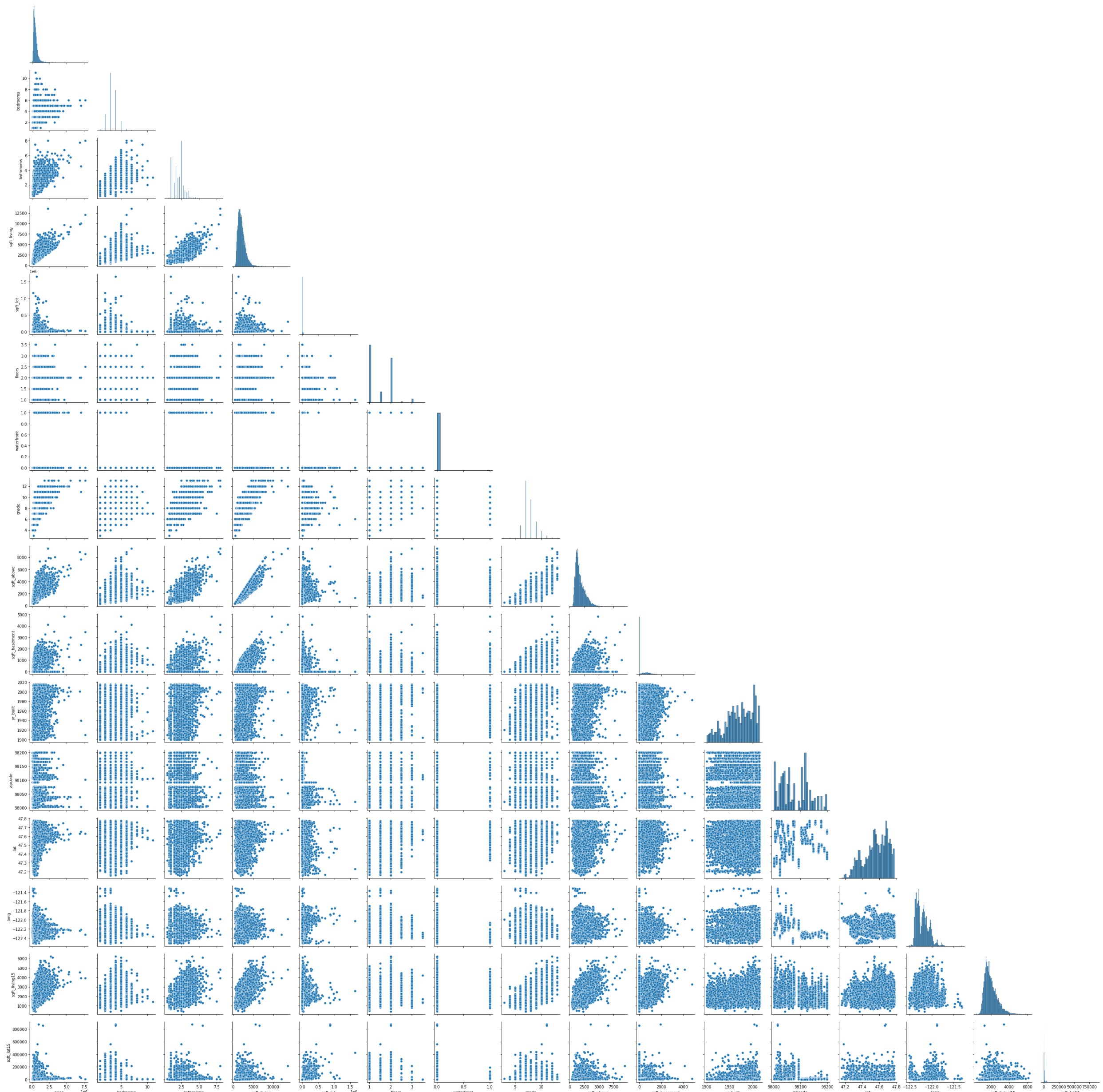
	price	sqft_living	grade	sqft_above	distance_from_bellevue	view_none	house_age	zipc
0	221900.0	1180	7	1180	6.724913	1.0	59	9
1	538000.0	2570	7	2170	9.940080	1.0	63	9
2	180000.0	770	6	770	9.571487	1.0	82	9
3	604000.0	1960	7	1050	10.530416	1.0	49	9
4	510000.0	1680	8	1680	7.392620	1.0	28	9
...	...	...	...	...	...	...	...	
21592	360000.0	1530	8	1530	9.572031	1.0	5	9
21593	400000.0	2310	8	2310	9.760026	1.0	1	9
21594	402101.0	1020	7	1020	4.578875	1.0	5	9
21595	400000.0	1600	8	1600	7.712755	1.0	11	9
21596	325000.0	1020	7	1020	4.580999	1.0	6	9

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 Groun
                         'Zipcode', 'House Age', 'No View', 'Distance from
            cmap="mako",
            annot=True)
plt.yticks(rotation=0);
```



In [10]:
```
# Scatter matrix

scatter_columns = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_l
                   'floors', 'waterfront', 'grade', 'sqft_above', 'sqft_bas
                   'yr_built', 'zipcode', 'lat', 'long', 'sqft_living15','s

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

Dep. Variable:	price	**R-squared:**	0.725
Model:	OLS	**Adj. R-squared:**	0.724
Method:	Least Squares	**F-statistic:**	2270.
Date:	Fri, 19 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:15:12	**Log-Likelihood:**	-2.9347e+05
No. Observations:	21597	**AIC:**	5.870e+05
Df Residuals:	21571	**BIC:**	5.872e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-2.869e+07	4.53e+06	-6.328	0.000	-3.76e+07	-1.98e+07
bedrooms	-4.093e+04	1899.880	-21.544	0.000	-4.47e+04	-3.72e+04
bathrooms	4.038e+04	3139.238	12.864	0.000	3.42e+04	4.65e+04
sqft_living	111.5017	2.194	50.811	0.000	107.200	115.803
sqft_lot	0.2136	0.046	4.639	0.000	0.123	0.304
floors	2372.2405	3463.560	0.685	0.493	-4416.592	9161.073
waterfront	5.355e+05	1.96e+04	27.378	0.000	4.97e+05	5.74e+05
grade	8.844e+04	2087.403	42.369	0.000	8.43e+04	9.25e+04
sqft_above	78.3929	2.171	36.108	0.000	74.137	82.648
sqft_basement	33.1190	2.548	12.999	0.000	28.125	38.113
yr_built	8639.5763	939.648	9.194	0.000	6797.797	1.05e+04
yr_renovated	24.9298	3.824	6.520	0.000	17.435	32.425
zipcode	-507.6069	31.836	-15.944	0.000	-570.008	-445.206
lat	3.072e+05	1.26e+04	24.379	0.000	2.83e+05	3.32e+05
long	-1.492e+05	1.27e+04	-11.719	0.000	-1.74e+05	-1.24e+05
sqft_living15	8.8410	3.328	2.656	0.008	2.318	15.364
sqft_lot15	-0.1242	0.071	-1.759	0.079	-0.263	0.014
yr_sold	1.954e+04	1877.232	10.410	0.000	1.59e+04	2.32e+04

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

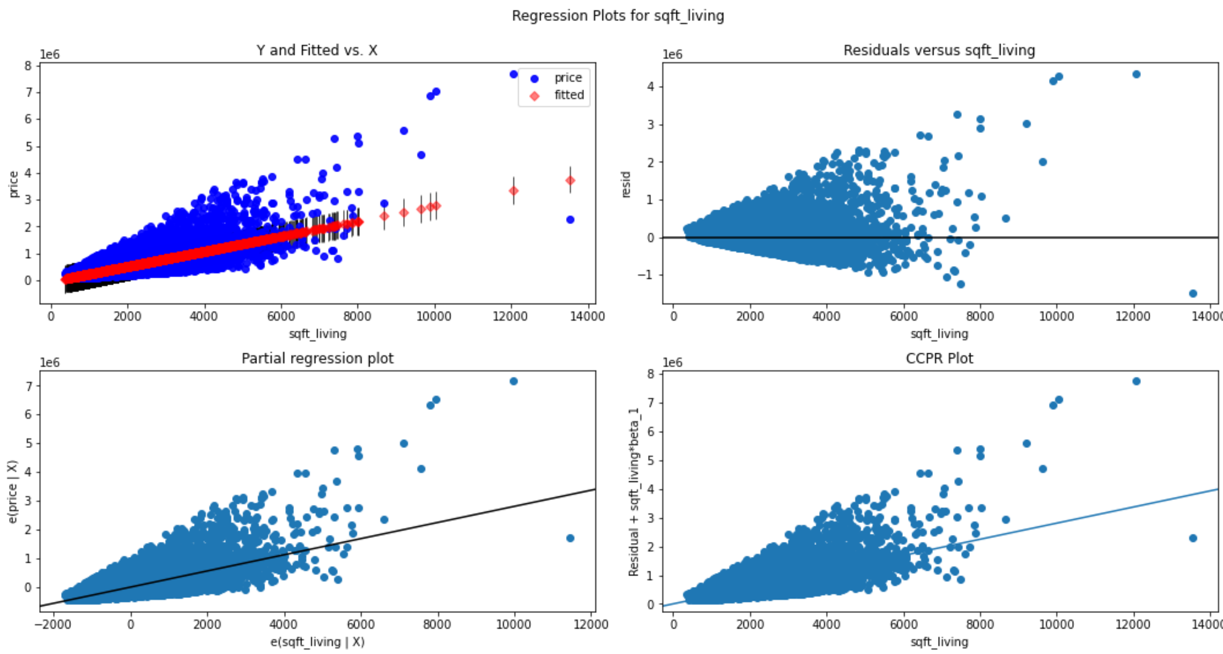
Dep. Variable:	price	**R-squared:**	0.493
Model:	OLS	**Adj. R-squared:**	0.493
Method:	Least Squares	**F-statistic:**	2.097e+04
Date:	Fri, 19 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:15:12	**Log-Likelihood:**	-3.0006e+05
No. Observations:	21597	**AIC:**	6.001e+05
Df Residuals:	21595	**BIC:**	6.001e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-4.399e+04	4410.023	-9.975	0.000	-5.26e+04	-3.53e+04
sqft_living	280.8630	1.939	144.819	0.000	277.062	284.664

Omnibus:	14801.942	**Durbin-Watson:**	1.982
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	542662.604
Skew:	2.820	**Prob(JB):**	0.00
Kurtosis:	26.901	**Cond. No.**	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

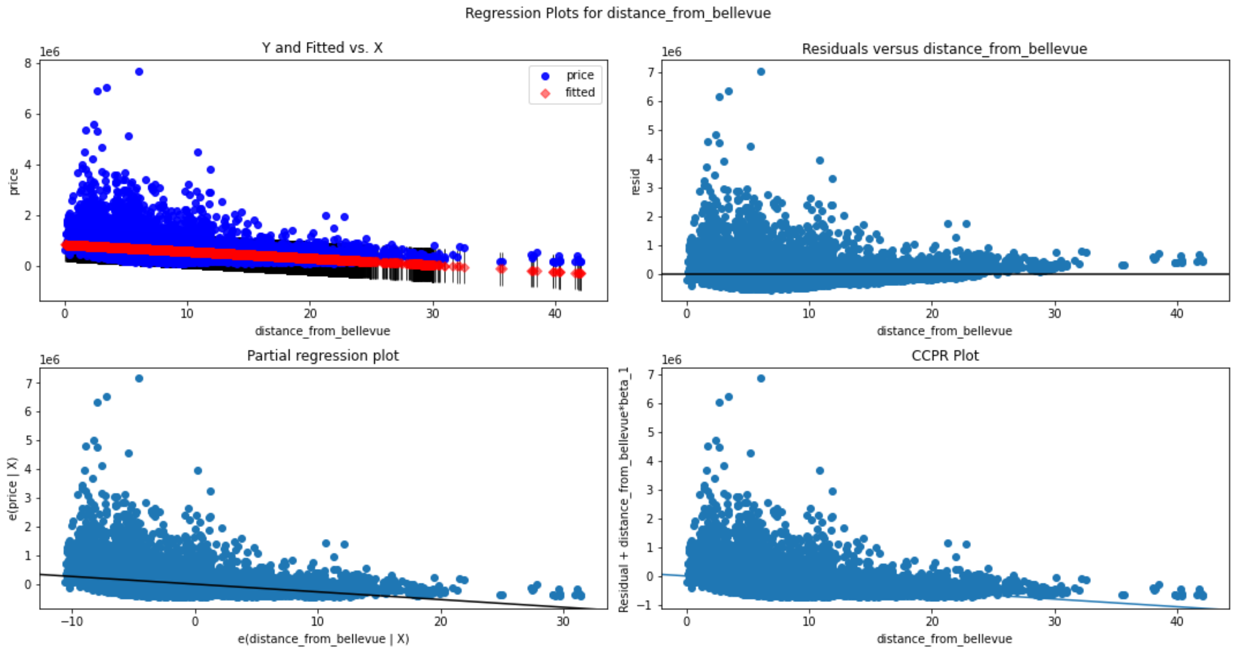
Dep. Variable:	price	**R-squared:**	0.158
Model:	OLS	**Adj. R-squared:**	0.158
Method:	Least Squares	**F-statistic:**	4062.
Date:	Fri, 19 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:15:17	**Log-Likelihood:**	-3.0553e+05
No. Observations:	21597	**AIC:**	6.111e+05
Df Residuals:	21595	**BIC:**	6.111e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	8.254e+05	5026.642	164.200	0.000	8.16e+05	8.35e+05
distance_from_bellevue	-2.677e+04	420.035	-63.734	0.000	-2.76e+04	-2.59e+04

Omnibus:	20006.141	**Durbin-Watson:**	1.969
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1572480.456
Skew:	4.233	**Prob(JB):**	0.00
Kurtosis:	43.936	**Cond. No.**	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=fi
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

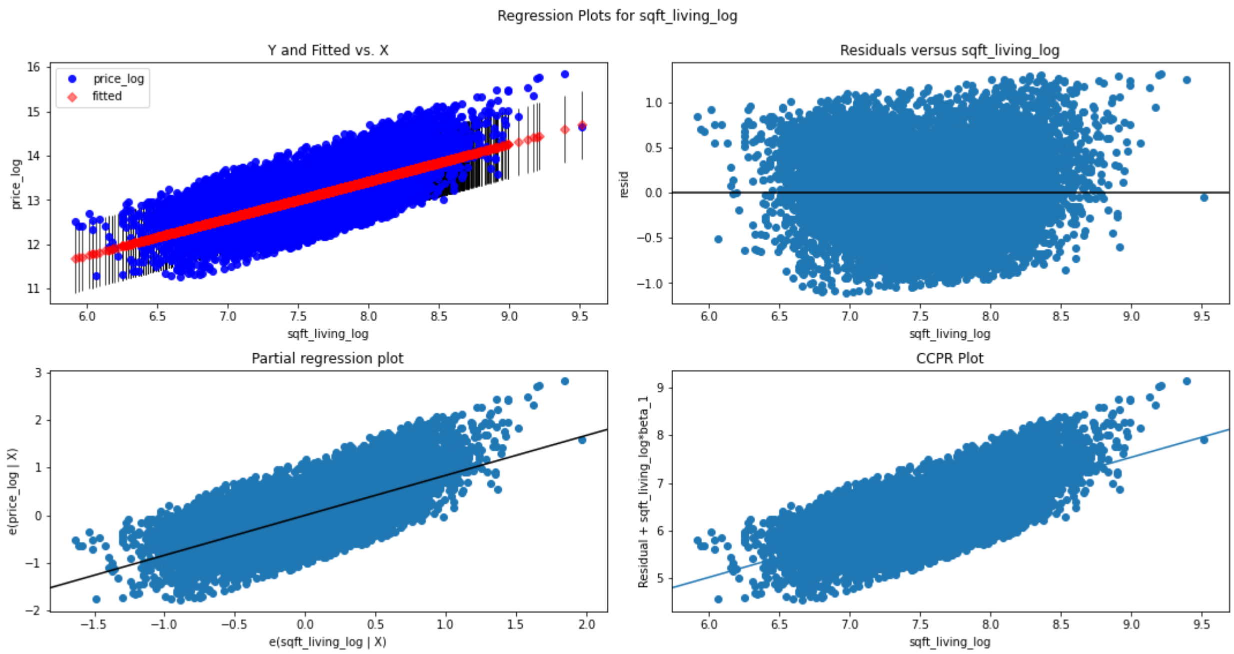
Dep. Variable:	price_log	**R-squared:**	0.455
Model:	OLS	**Adj. R-squared:**	0.455
Method:	Least Squares	**F-statistic:**	1.805e+04
Date:	Fri, 19 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:15:20	**Log-Likelihood:**	-10231.
No. Observations:	21597	**AIC:**	2.047e+04
Df Residuals:	21595	**BIC:**	2.048e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.7234	0.047	142.612	0.000	6.631	6.816
sqft_living_log	0.8376	0.006	134.368	0.000	0.825	0.850

Omnibus:	123.577	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	114.096
Skew:	0.143	**Prob(JB):**	1.68e-25
Kurtosis:	2.787	**Cond. No.**	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);
```



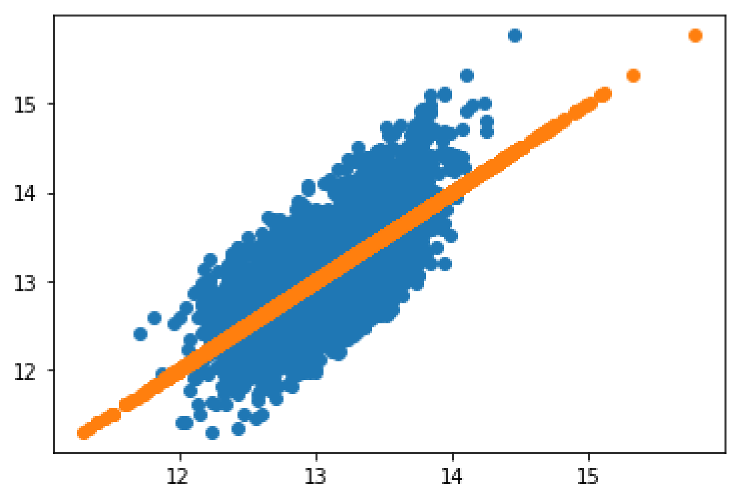
> When 'price' and 'sqft_living' undergo log transformation, they are more normally distributed and more homoscedastic, making them better for modeling.

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)
print('Baseline Train R\u00b2:', baseline.score(X_train, y_train))
print('Baseline Test R\u00b2:', baseline.score(X_test, y_test))
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)
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)
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: -5.5961729294118356e-05

Simple LR Train R²: 0.4603990854859664
Simple LR Test R²: 0.43954158799876863
Simple LR Train RMSE: 0.3883567577431248
Simple LR Test RMSE: 0.38933804308730074
```

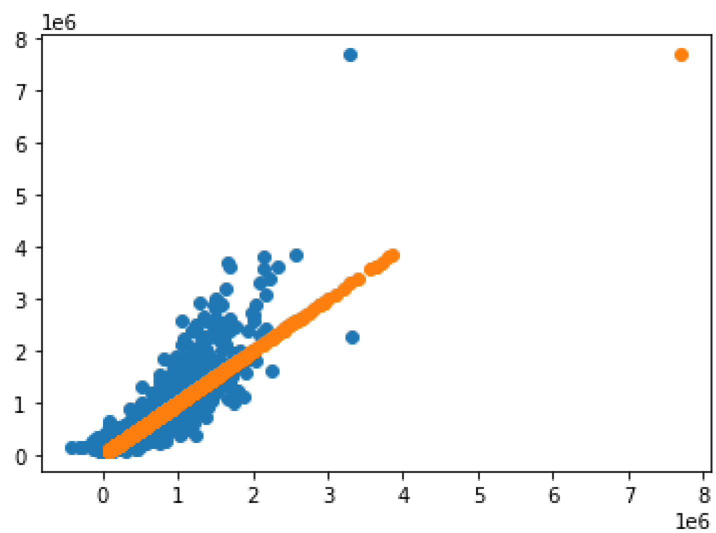


First Multiple Linear Regression Model

Model with all untouched predictor variables.

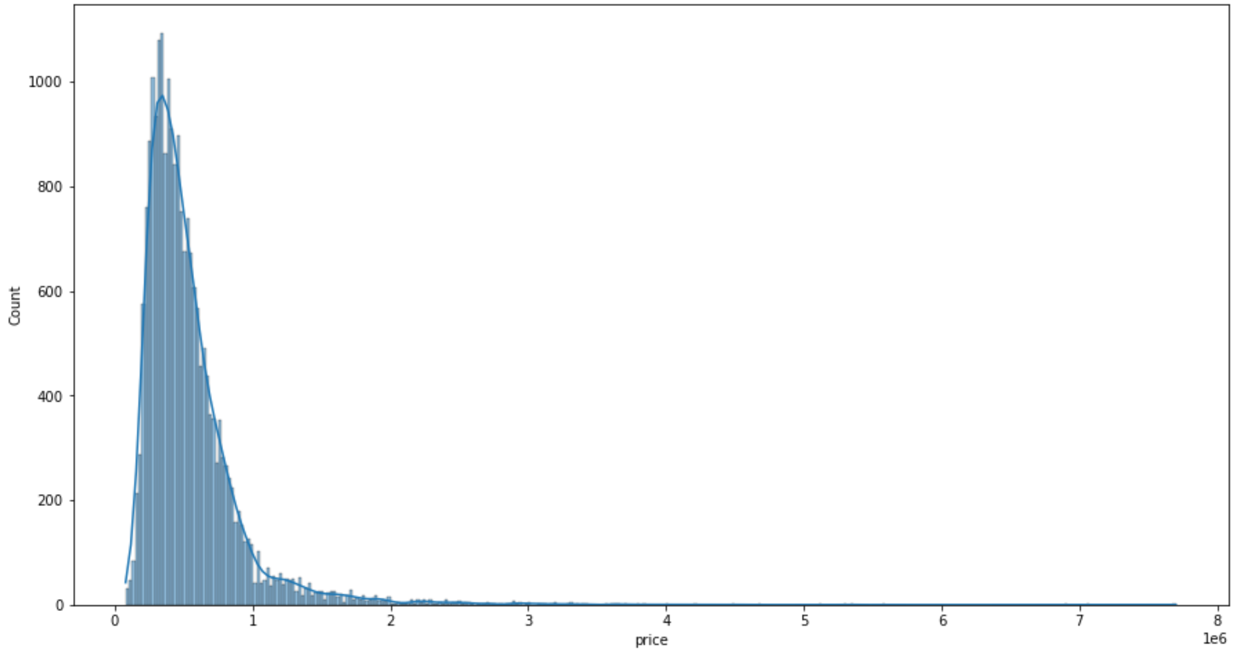
In [19]:
```
# df1 = original df
df1 = df.copy()
X1 = df1.drop(['price'], axis=1)
y1 = df1[['price']]
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1)
lr1 = LinearRegression()
lr1.fit(X1_train, y1_train)
y1_hat_train = lr1.predict(X1_train)
y1_hat_test = lr1.predict(X1_test)
train1_rmse = mse(y1_train, y1_hat_train, squared=False)
test1_rmse = mse(y1_test, y1_hat_test, squared=False)
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)
y1_test_pred = lr1.predict(X1_test)
plt.scatter(y1_test_pred, y1_test)
plt.scatter(y1_test, y1_test);
```

```
LR1 Train R²: 0.7252891338790399
LR1 Test R²: 0.7215939504683346
LR1 Train RMSE: 191506.27076791142
LR1 Test RMSE: 196932.5740261702
```

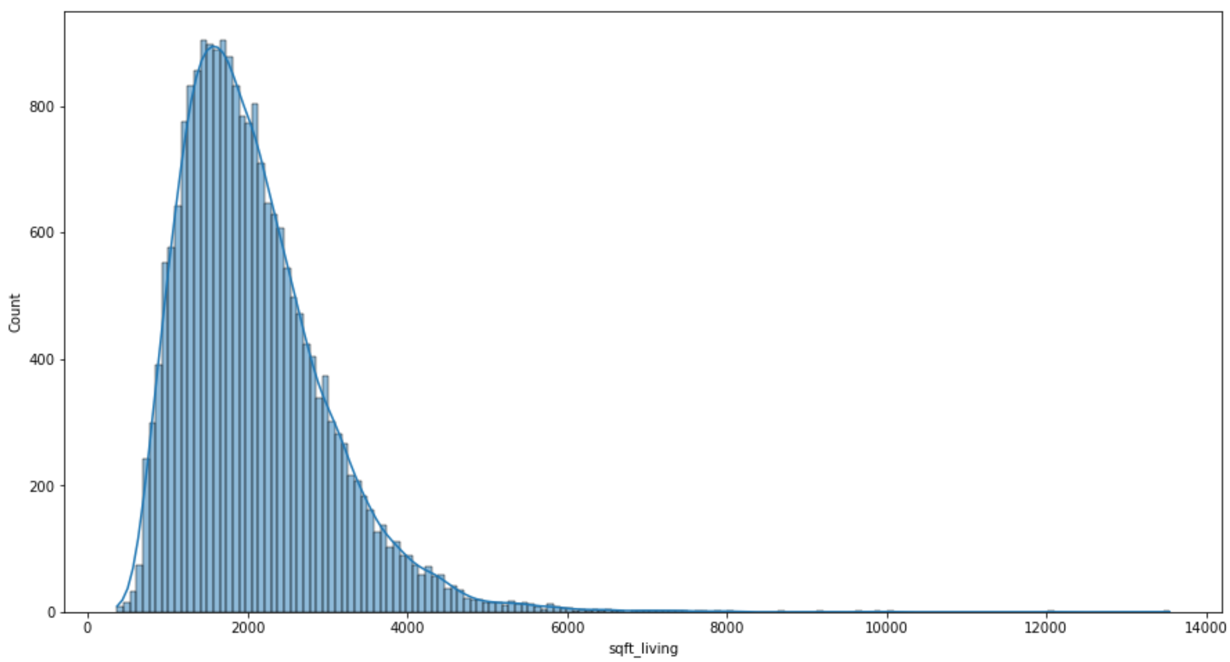


> There are outliers in our dataset affecting our model.

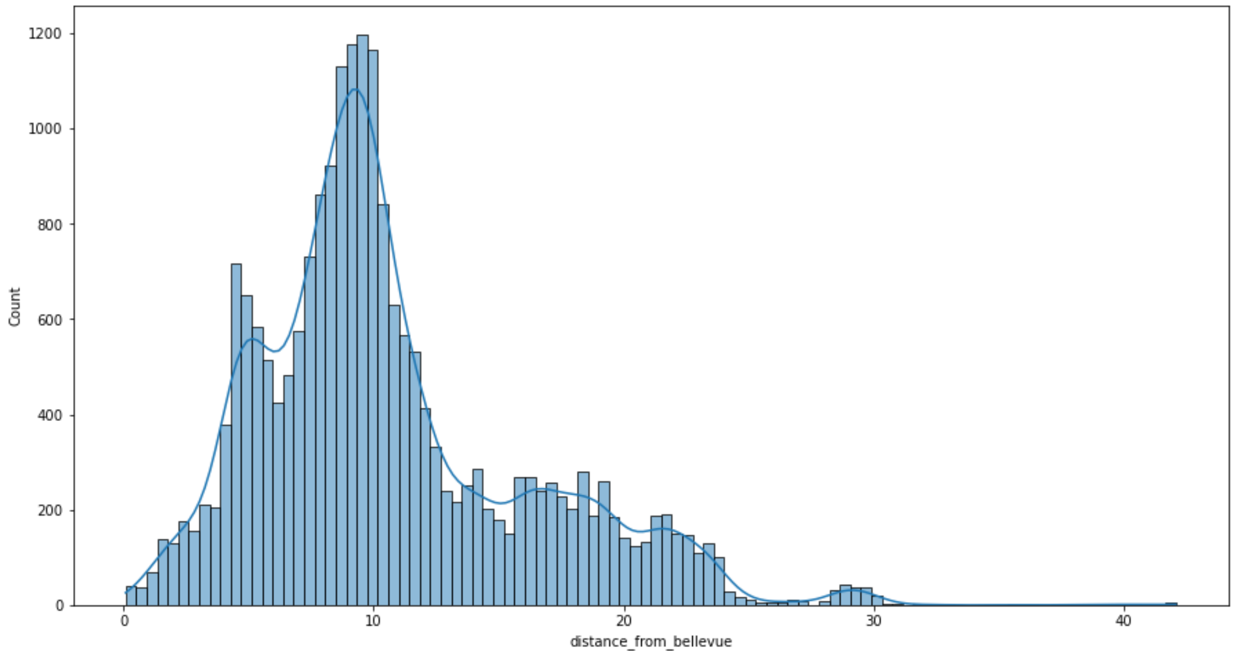
In [20]:
```
plt.figure(figsize=(15,8))
sns.histplot(df['price'], kde=True);
```



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.

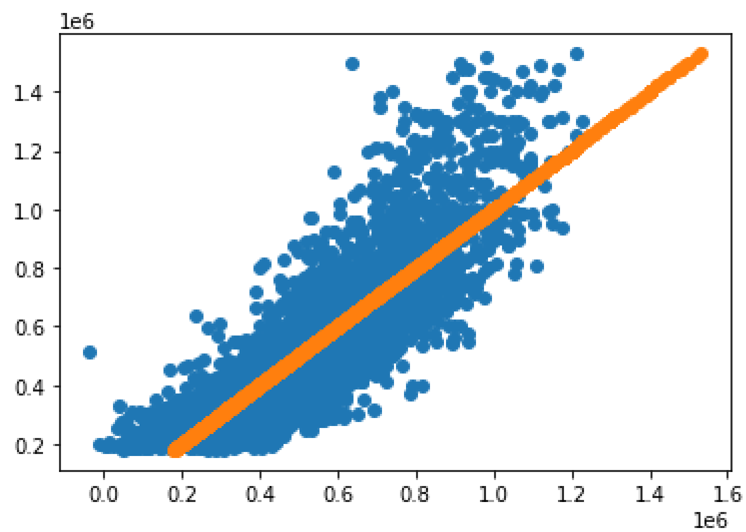
Second Multiple Linear Regression Model

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)
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)
y2_test_pred = lr2.predict(X2_test)
plt.scatter(y2_test_pred, y2_test)
plt.scatter(y2_test, y2_test);
```

```
LR2 Train R²: 0.7198195176766033
LR2 Test R²: 0.7198195176766033
LR2 Train RMSE: 126331.89233509479
LR2 Test RMSE: 124371.53184994315
```



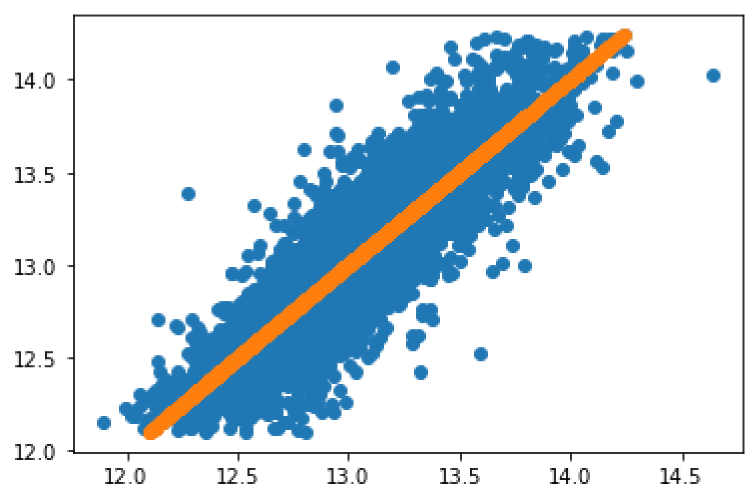
Third Multiple Linear Regression Model

Second model with log transformed price, sqft_living, and distance_from_bellevue, 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)
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)
y3_test_pred = lr3.predict(X3_test)
plt.scatter(y3_test_pred, y3_test)
plt.scatter(y3_test, y3_test);
```

```
LR3 Train R²: 0.7694868743494628
LR3 Test R²: 0.7714920558138721
LR3 Train RMSE: 0.21237417722684118
LR3 Test RMSE: 0.20747599993248741
```



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]:

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

	bedrooms	bathrooms	floors	waterfront	grade	sqft_above	sqft_basen
distance_from_bellevue_log	-0.06	-0.02	0.02	0.03	-0.12	0.03	-
sqft_lot_log	0.17	0.03	-0.29	0.06	0.13	0.29	(

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

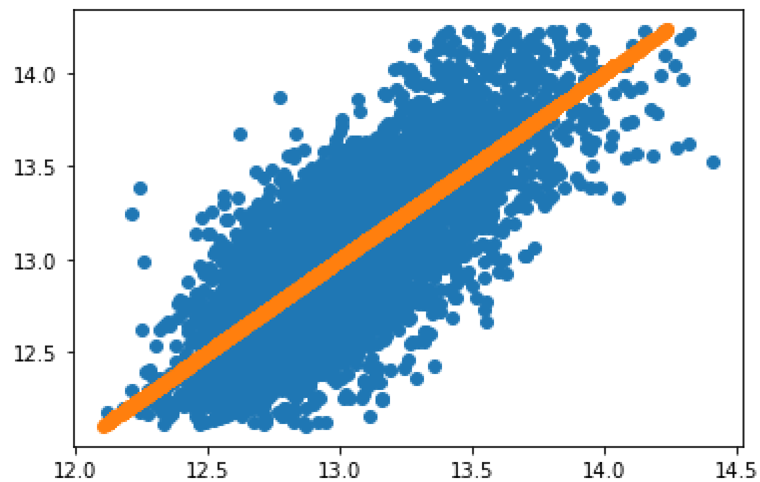
In [29]:
```
corr = df4.corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr[mask] = np.nan
(corr
 .style
 .background_gradient(cmap='mako', axis=None, vmin=-1, vmax=1)
 .highlight_null(null_color='#f1f1f1')  # Color NaNs grey
 .set_precision(2))
```

Out[29]:

	price_log	sqft_living_log	distance_from_bellevue_log	sqft_lot_log	waterf
price_log	nan	nan	nan	nan	
sqft_living_log	0.59	nan	nan	nan	
distance_from_bellevue_log	-0.49	-0.05	nan	nan	
sqft_lot_log	0.08	0.30	0.18	nan	
waterfront	0.08	0.01	0.03	0.06	
yr_renovated	0.10	0.03	-0.05	0.01	
house_age	-0.01	-0.31	-0.20	0.04	
view_none	-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)
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)
y4_test_pred = lr4.predict(X4_test)
plt.scatter(y4_test_pred, y4_test)
plt.scatter(y4_test, y4_test);
```

```
LR4 Train R²: 0.5953521487730509
LR4 Test R²: 0.5761276051781581
LR4 Train RMSE: 0.28135365640413407
LR4 Test RMSE: 0.28263608097796394
```



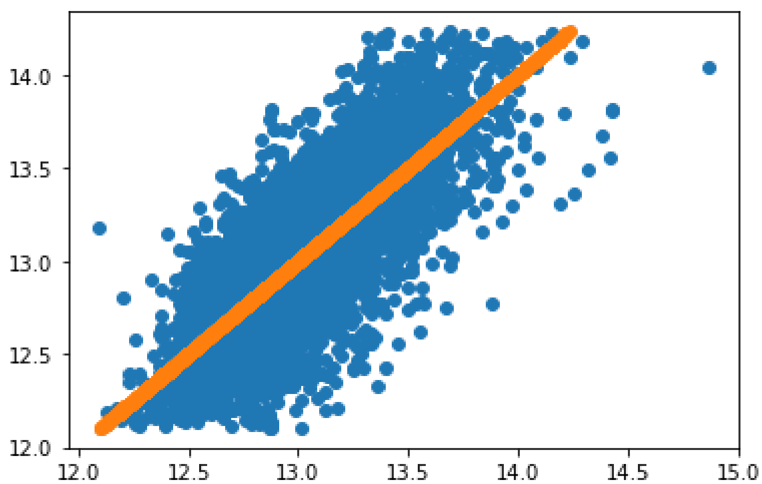
Fifth Multiple Linear Regression Model

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)
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)
y5_test_pred = lr5.predict(X5_test)
plt.scatter(y5_test_pred, y5_test)
plt.scatter(y5_test, y5_test);
```

```
LR5 Train R²: 0.5928583773386145
LR5 Test R²: 0.5837221735060115
LR5 Train RMSE: 0.2814435882651718
LR5 Test RMSE: 0.2824652820120836
```



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_bellevue_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

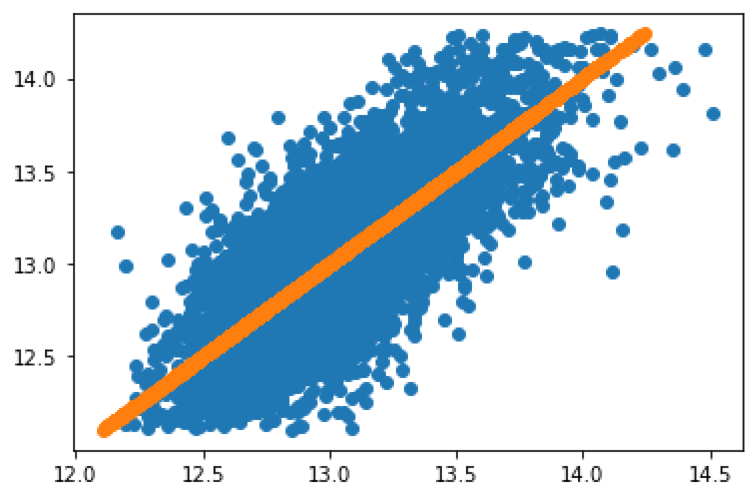
19733 rows × 7 columns

> Selector selects sqft_living_log, distance_from_bellevue_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', 'v
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)
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)
y6_test_pred = lr6.predict(X6_test)
plt.scatter(y6_test_pred, y6_test)
plt.scatter(y6_test, y6_test);
```

```
LR6 Train R²: 0.5850359670236334
LR6 Test R²: 0.5870529875220447
LR6 Train RMSE: 0.28342757017777953
LR6 Test RMSE: 0.28345492090205016
```



Seventh Multiple Linear Regression Model

Sixth model using stepwise selection to choose significant features.

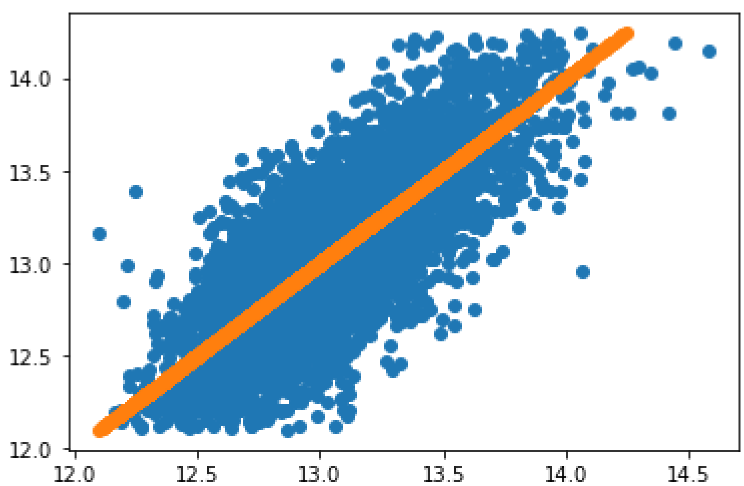
In [35]:
```
def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    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_
            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, b

        # 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,
        if not changed:
            break
    return included

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

```
Add sqft_living_log                with p-value 0.0
Add distance_from_bellevue_log     with p-value 0.0
Add view_none                      with p-value 7.06227e-222
```

```
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:
['sqft_living_log', 'distance_from_bellevue_log', 'view_none', 'house_ag
e', 'waterfront', 'yr_renovated', 'sqft_lot_log']
```

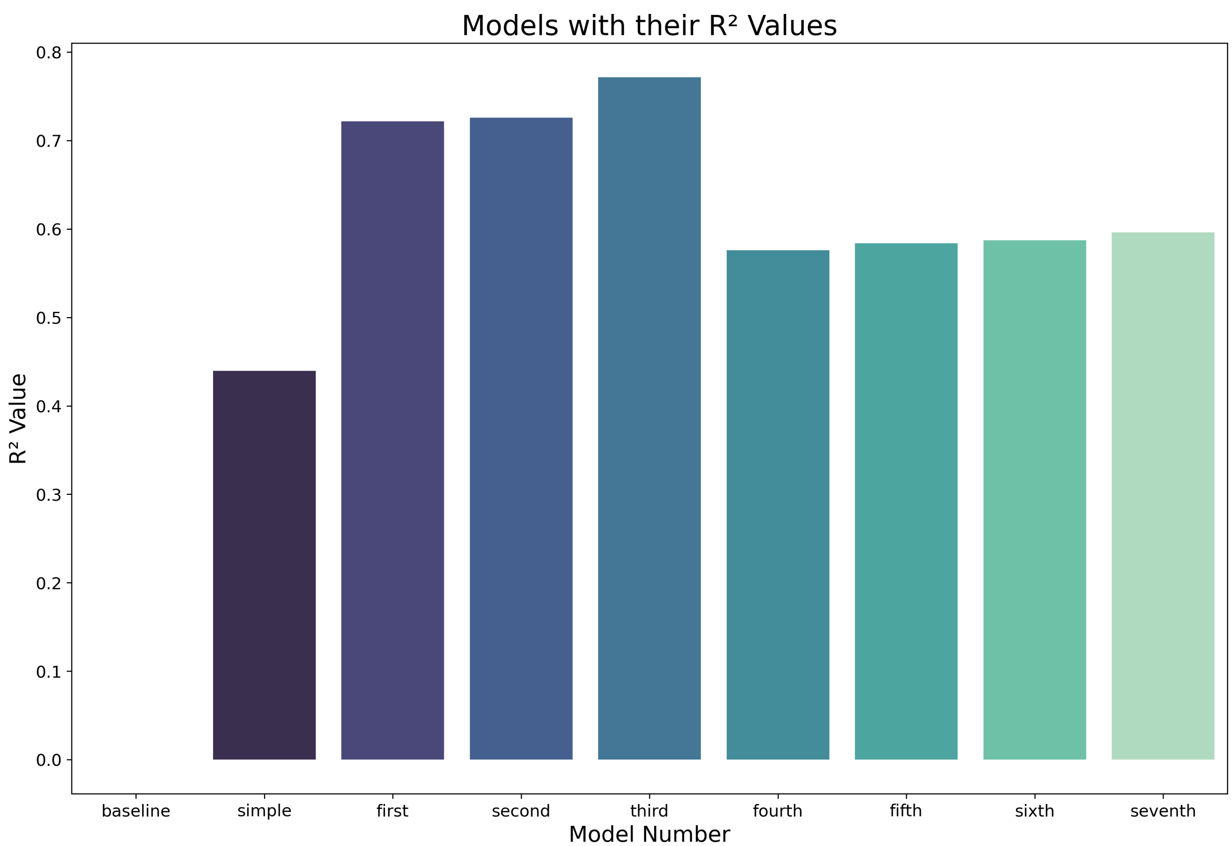
In [36]:
```
df7 = df6.copy()
X7 = df7[['distance_from_bellevue_log', 'view_none', 'sqft_living_log', 'ho
y7 = df7[['price_log']]
X7_train, X7_test, y7_train, y7_test = train_test_split(X7, y7)
lr7 = LinearRegression()
lr7.fit(X7_train, y7_train)
y7_hat_train = lr7.predict(X7_train)
y7_hat_test = lr7.predict(X7_test)
train7_rmse = mse(y7_train, y7_hat_train, squared=False)
test7_rmse = mse(y7_test, y7_hat_test, squared=False)
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)
y7_test_pred = lr7.predict(X7_test)
plt.scatter(y7_test_pred, y7_test)
plt.scatter(y7_test, y7_test);
```

```
LR7 Train R²: 0.5889029655704165
LR7 Test R²: 0.5962276283602786
LR7 Train RMSE: 0.2829314101404769
LR7 Test RMSE: 0.2778441592085275
```



Model Visualizations

In [37]:
```
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', 'sev
                         '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);
```



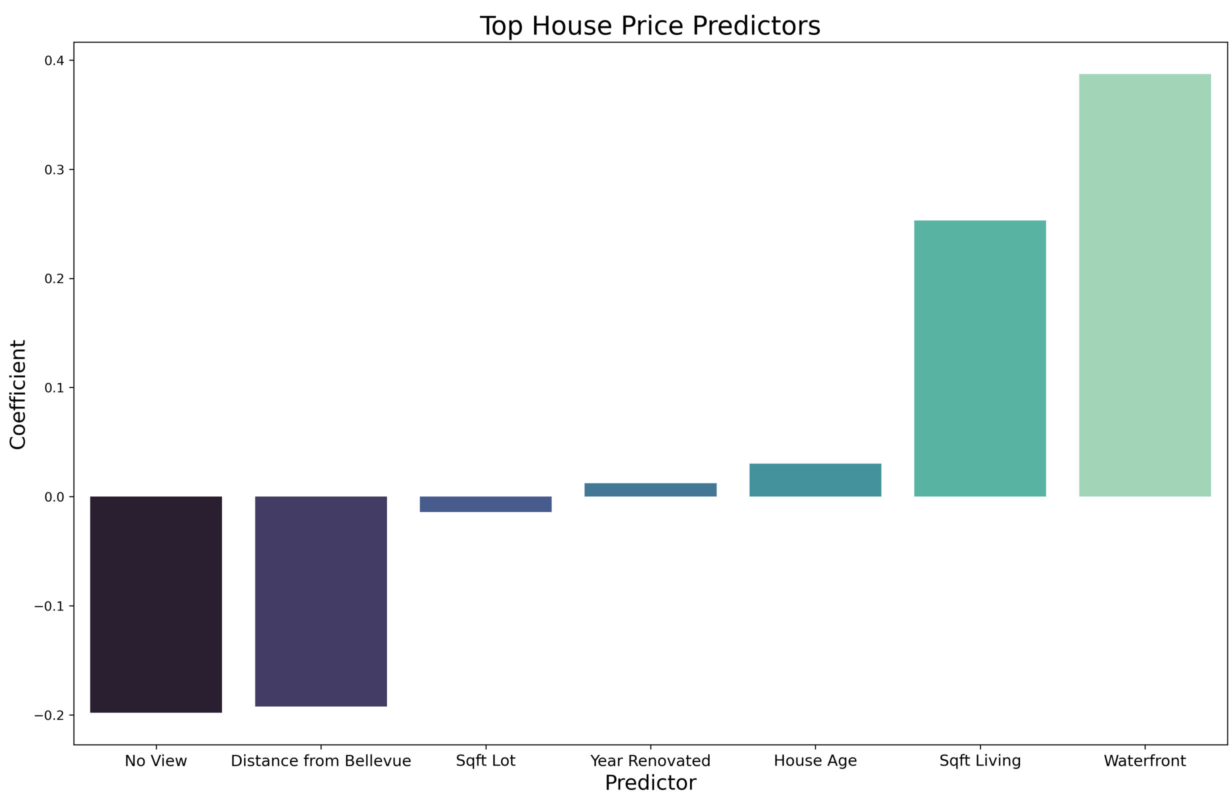
Bar chart showing R^2 values of all of our models.

In [38]:
```
lr7.coef_
```

Out[38]:
```
array([[-0.19132927, -0.2134266 ,  0.25331088,  0.02843826,  0.36880234,
         0.01449829, -0.01372193]])
```

In [39]:
```
plotdf = pd.DataFrame({'Predictor':['distance_from_bellevue_log', 'view_non
                                    'house_age', 'waterfront', 'yr_renovate
                       'Coefficient':[-0.19245633, -0.19812453, 0.25314856,
                                     0.38718511, 0.01240198, -0.01416162]}
plotdf = plotdf.sort_values(by=['Coefficient'])
```

In [40]:
```
fig, ax = plt.subplots(figsize=(16,10), dpi=300)
ax = sns.barplot(x=plotdf['Predictor'], y=plotdf['Coefficient'], palette="m
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 Lot', 'Year Renovated'
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 R2 value of 0.592, RMSE of 0.282, and Condition Number of 23.73. 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.