Project 2 Submission - Student Details

- Student name: Yessy Rayner
- Student pace: Part-time
- Scheduled project review date/time: Saturday 20 August 2022
- Instructor name: JP Hwang

Business Analysis on Property Investment in Kings County, CA

Business Overview

KC Financial Investment would like to expand their business portfolio to include property buy and sell investment. They would like to invest in the property buy and sell market in the Kings County, California. According to Realtor.com homes for sale in Kings County spend an average of 50 days on the market, so it is quite a lucrative market in term of the short sales turn around. Also the outgoing fees for buying and selling in Kings County are relatively low (under 1%) as long as they don't engage the real estate agent services due to high commission.

Their goal is to maximise profits through the buy and sell, and would like the Business Analyst (myself) to build a model that predict the house prices based on the sales database in the past two years.

Once the prediction model is built, they will then analyse all of the available houses that are currently for sale in the market and purchase the properties that are under predicted asking price. They will resell after a few weeks.

Data Cleaning and Preparation

In this section, I will be reviewing, checking and cleaning the datas, in order to:

- Ensure the datas are in correct data types
- Evaluate if there are any missing value and fill it with approriate value

Once datas are cleaned, I will separate them into 3 variables or features as follow for testing and validation in order to get the best model:

- Continuous variables (A continuous variable can be of any value in a range, for example: sqft living and lot sizes)
- Discrete variables (certain number of particular values that can be counted, for example: number of bedrooms)
- Categorical variables (descriptive categories instead of numerical or measured categories, for example: zipcodes)

In [1]:

```
# Import standard packages
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt

import statsmodels.api as sm
from statsmodels.stats import diagnostic as diag
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

%matplotlib inline
```

In [ ]:

```
# %load data/column_names.md
# Column Names and descriptions for Kings County Data Set
* **id** - unique identified for a house
* **dateDate** - house was sold
* **pricePrice** -  is prediction target
* **bedroomsNumber** -  of Bedrooms/House
* **bathroomsNumber** -  of bathrooms/bedrooms
* **sqft_livingsquare** -  footage of the home
* **sqft_lotsquare** -  footage of the lot
* **floorsTotal** -  floors (levels) in house
* **waterfront** - House which has a view to a waterfront
* **view** - Has been viewed
* **condition** - How good the condition is ( Overall )
* **grade** - overall grade given to the housing unit, based on King County grading system
* **sqft_above** - square footage of house apart from basement
* **sqft_basement** - square footage of the basement
* **yr_built** - Built Year
* **yr_renovated** - Year when house was renovated
* **zipcode** - zip
* **lat** - Latitude coordinate
* **long** - Longitude coordinate
* **sqft_living15** - The square footage of interior housing living space for the nearest 1
* **sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors
```

In [3]:

```
#Now let's import the sales database for the past 2 years

kc_data = pd.read_csv('data/kc_house_data.csv')
```

In [4]:

```
kc_data.head()
```

Out[4]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterf
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [5]:

```
# A quick look on the data types and check if there are any missing value
kc_data.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  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 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(8), int64(11), object(2)
memory usage: 3.5+ MB
```

In [6]:

```
#Convert sqft_basement from object to float first as there are numbers value attached to th
kc_data['sqft_basement'] = pd.to_numeric(kc_data['sqft_basement'], errors='coerce')
```

In [7]:

```
kc_data['sqft_basement'].value_counts(dropna=False)
#454 NaN value, I will convert them to 0 at later stage due to majority value of 0 (0 indic
```

Out[7]:

```
0.0       12826
NaN         454
600.0       217
500.0       209
700.0       208
           ...
1920.0        1
3480.0        1
2730.0        1
2720.0        1
248.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

In [8]:

```
#Convert all float to int for consistency with the rest of the data types
m = kc_data.select_dtypes(np.number)
kc_data[m.columns]= m.round().astype('Int64')
```

In [9]:

```
kc_data.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  Int64
 3   bedrooms       21597 non-null  Int64
 4   bathrooms      21597 non-null  Int64
 5   sqft_living    21597 non-null  Int64
 6   sqft_lot       21597 non-null  Int64
 7   floors         21597 non-null  Int64
 8   waterfront     19221 non-null  Int64
 9   view           21534 non-null  Int64
 10  condition      21597 non-null  Int64
 11  grade          21597 non-null  Int64
 12  sqft_above     21597 non-null  Int64
 13  sqft_basement  21143 non-null  Int64
 14  yr_built       21597 non-null  Int64
 15  yr_renovated   17755 non-null  Int64
 16  zipcode        21597 non-null  Int64
 17  lat            21597 non-null  Int64
 18  long           21597 non-null  Int64
 19  sqft_living15  21597 non-null  Int64
 20  sqft_lot15     21597 non-null  Int64
dtypes: Int64(20), object(1)
memory usage: 3.9+ MB
```

In [10]:

```
#Looking at the above data yr_renovated, view, sqft_basement and waterfront have some missi
kc_data['yr_renovated'].value_counts(dropna=False)
```

Out[10]:

```
0       17011
<NA>     3842
2014       73
2013       31
2003       31
        ...
1953        1
1946        1
1976        1
1948        1
1950        1
Name: yr_renovated, Length: 71, dtype: Int64
```

In [11]:

```
kc_data['waterfront'].value_counts(dropna=False)
```

Out[11]:

```
0       19075
<NA>     2376
1         146
Name: waterfront, dtype: Int64
```

In [12]:

```
kc_data['view'].value_counts(dropna=False)
```

Out[12]:

```
0       19422
2         957
3         508
1         330
4         317
<NA>       63
Name: view, dtype: Int64
```

In [13]:

```
kc_data['sqft_basement'].value_counts(dropna=False)
```

Out[13]:

```
0       12826
<NA>      454
600       217
500       209
700       208
         ...
2310        1
2120        1
295         1
207         1
1281        1
Name: sqft_basement, Length: 304, dtype: Int64
```

In [14]:

```
#Due to majority of the counts are 0, I will replace the NaN value to 0. And these 0s indic
#NOT waterfront property, NO basement, NOT being viewed, NOT renovated
kc_data['waterfront'].fillna(0, inplace=True)
kc_data['sqft_basement'].fillna(0, inplace=True)
kc_data['view'].fillna(0, inplace=True)
kc_data['yr_renovated'].fillna(0, inplace=True)
```

In [15]:

```
# Double check the cleaned dataframe, majority of them have value now  and in correct type
# I didn't clean the 'date' column, as I won't be using them as feature
kc_data.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  Int64
 3   bedrooms       21597 non-null  Int64
 4   bathrooms      21597 non-null  Int64
 5   sqft_living    21597 non-null  Int64
 6   sqft_lot       21597 non-null  Int64
 7   floors         21597 non-null  Int64
 8   waterfront     21597 non-null  Int64
 9   view           21597 non-null  Int64
 10  condition      21597 non-null  Int64
 11  grade          21597 non-null  Int64
 12  sqft_above     21597 non-null  Int64
 13  sqft_basement  21597 non-null  Int64
 14  yr_built       21597 non-null  Int64
 15  yr_renovated   21597 non-null  Int64
 16  zipcode        21597 non-null  Int64
 17  lat            21597 non-null  Int64
 18  long           21597 non-null  Int64
 19  sqft_living15  21597 non-null  Int64
 20  sqft_lot15     21597 non-null  Int64
dtypes: Int64(20), object(1)
memory usage: 3.9+ MB
```

In [16]:

```
#Lets check the sales datas that we have
kc_data.describe()
```

Out[16]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	2
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	
mean	4.580474e+09	5.402966e+05	3.373200	2.059777	2080.321850	1.509941e+04	
std	2.876736e+09	3.673681e+05	0.926299	0.754435	918.106125	4.141264e+04	
min	1.000102e+06	7.800000e+04	1.000000	0.000000	370.000000	5.200000e+02	
25%	2.123049e+09	3.220000e+05	3.000000	2.000000	1430.000000	5.040000e+03	
50%	3.904930e+09	4.500000e+05	3.000000	2.000000	1910.000000	7.618000e+03	
75%	7.308900e+09	6.450000e+05	4.000000	2.000000	2550.000000	1.068500e+04	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	

In [17]:

```
kc_data.describe(include=np.number)
```

Out[17]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	2
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	
mean	4.580474e+09	5.402966e+05	3.373200	2.059777	2080.321850	1.509941e+04	
std	2.876736e+09	3.673681e+05	0.926299	0.754435	918.106125	4.141264e+04	
min	1.000102e+06	7.800000e+04	1.000000	0.000000	370.000000	5.200000e+02	
25%	2.123049e+09	3.220000e+05	3.000000	2.000000	1430.000000	5.040000e+03	
50%	3.904930e+09	4.500000e+05	3.000000	2.000000	1910.000000	7.618000e+03	
75%	7.308900e+09	6.450000e+05	4.000000	2.000000	2550.000000	1.068500e+04	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	

In [18]:

```
num_vars = kc_data.describe(include=np.number).columns.tolist()
num_vars = [c for c in num_vars if c != 'price' ]
print(num_vars)
```

```
['id', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterf
ront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_buil
t', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15']
```

Modelling

Upon reviewing the above datas and their value, here are my breakdown on the features or variables. So we can model it approriately on the next step using linear regression model to predict property prices based on below features:

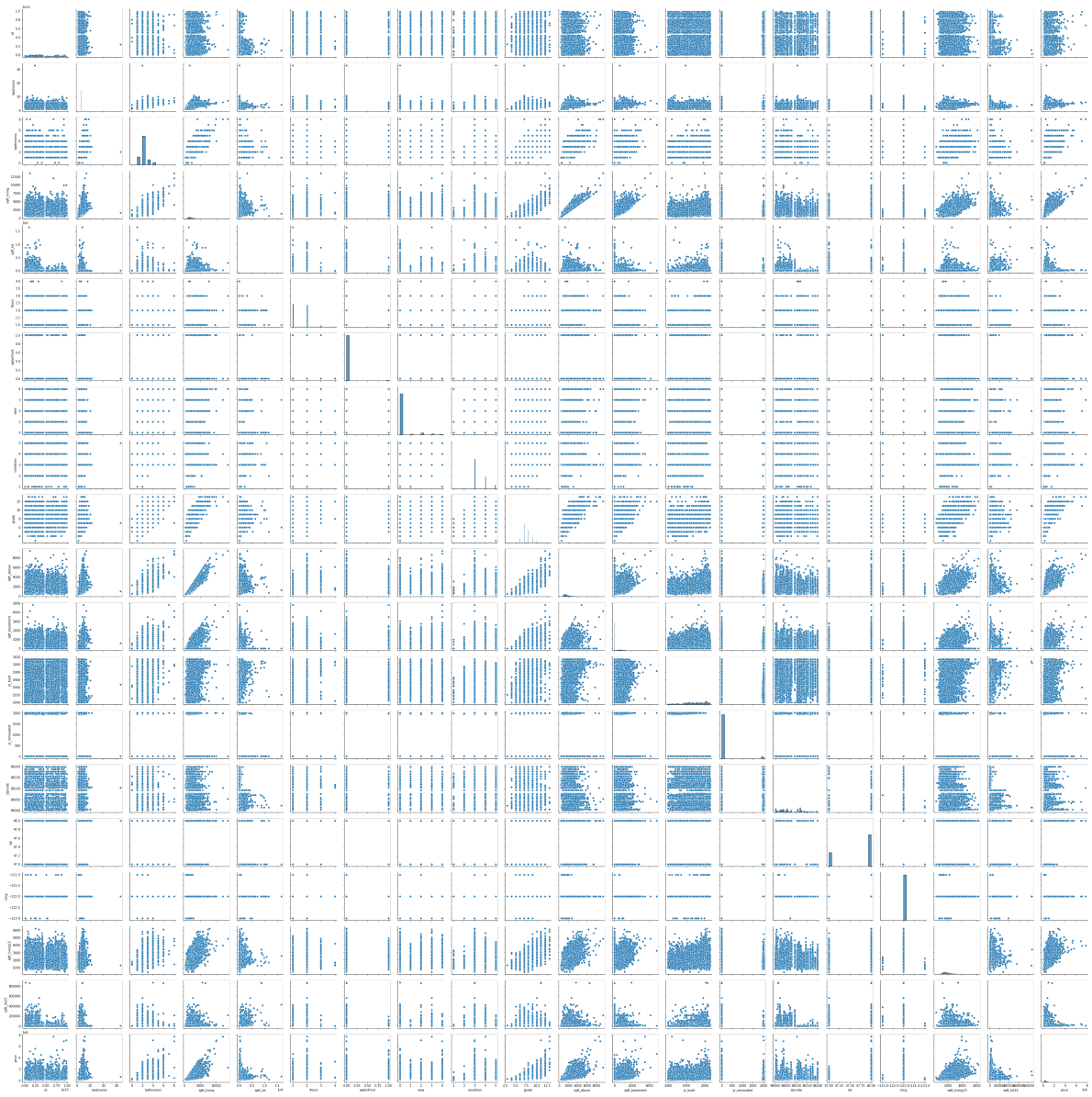
- Categorical: Waterfront, zipcode
- Discrete: condition, grade, bedrooms, bathrooms, floors, view
- Continuous: yr_build, yr_renovated, lat, long, ALL of sqft_

In [19]:

```
#Let's have a quick look to see if there are any relationship or inconsistency in the datas

sns.pairplot(kc_data[num_vars + ['price']])
```

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x2aad5b62820>
```

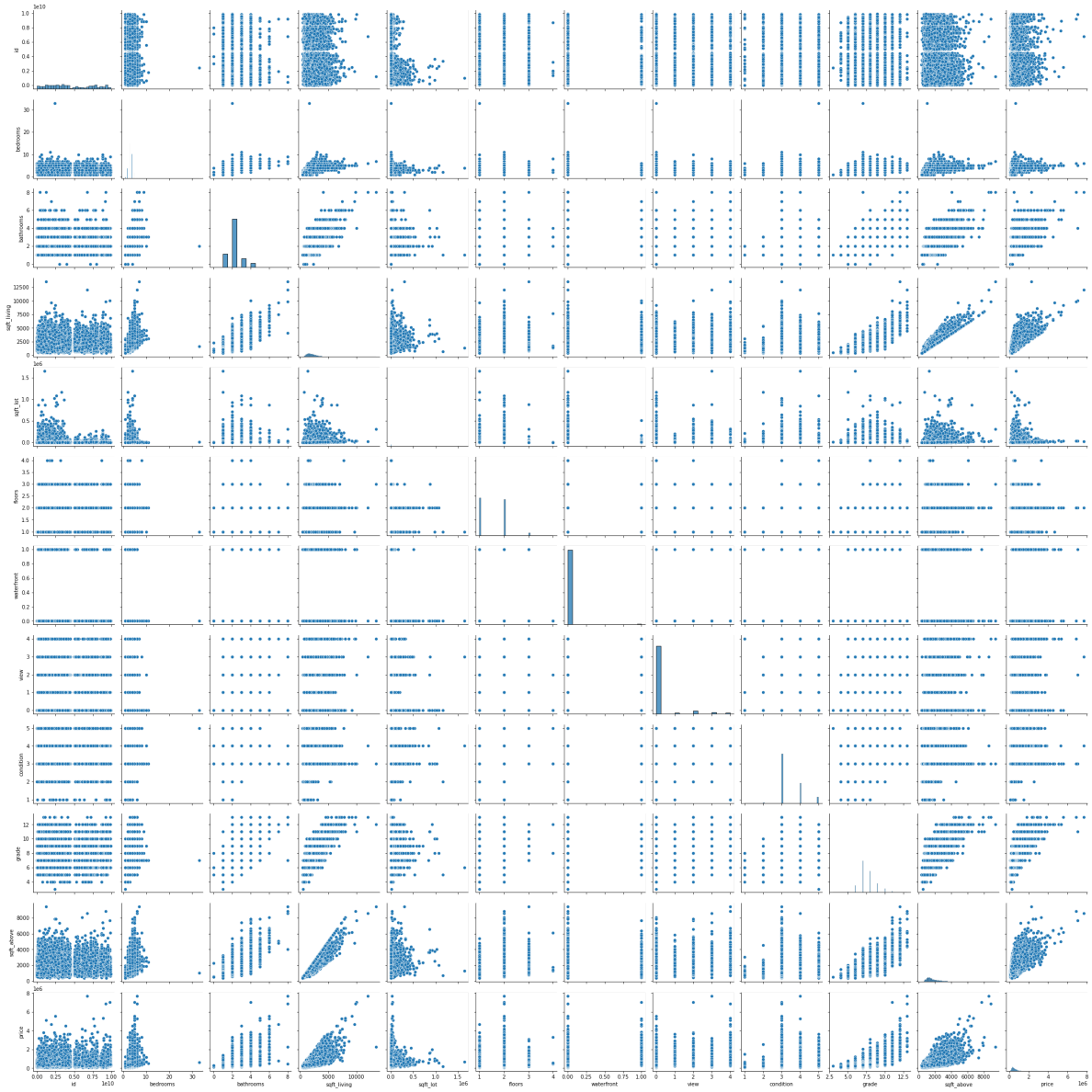


In [20]:

```
#Looking at the graphs above, grade, sqft_living & sqft_above have strong linear regression
#Bathroom & sqft_living15 = somewhat weak linear relationship
#Zoom in to look at it closely
sns.pairplot(kc_data[num_vars [:11] + ['price']])
```

Out[20]:

```
<seaborn.axisgrid.PairGrid at 0x2aaf31407f0>
```



In [21]:

```
#Checking the linearcorrelation and use the highest score to build base model
corr_df = kc_data[num_vars + ['price']].corr()['price'].reset_index()
corr_df[corr_df['price'] > 0.5]
```

Out[21]:

	index	price
2	bathrooms	0.519628
3	sqft_living	0.701917
9	grade	0.667951
10	sqft_above	0.605368
17	sqft_living15	0.585241
19	price	1.000000

In [22]:

```
#base model
base_vars = ['sqft_living', 'grade', 'sqft_above']
```

In [23]:

```
df_sm = kc_data[base_vars + ['price']]
df_sm.head()
```

Out[23]:

	sqft_living	grade	sqft_above	price
0	1180	7	1180	221900
1	2570	7	2170	538000
2	770	6	770	180000
3	1960	7	1050	604000
4	1680	8	1680	510000

In [24]:

```
df_sm.info()
```

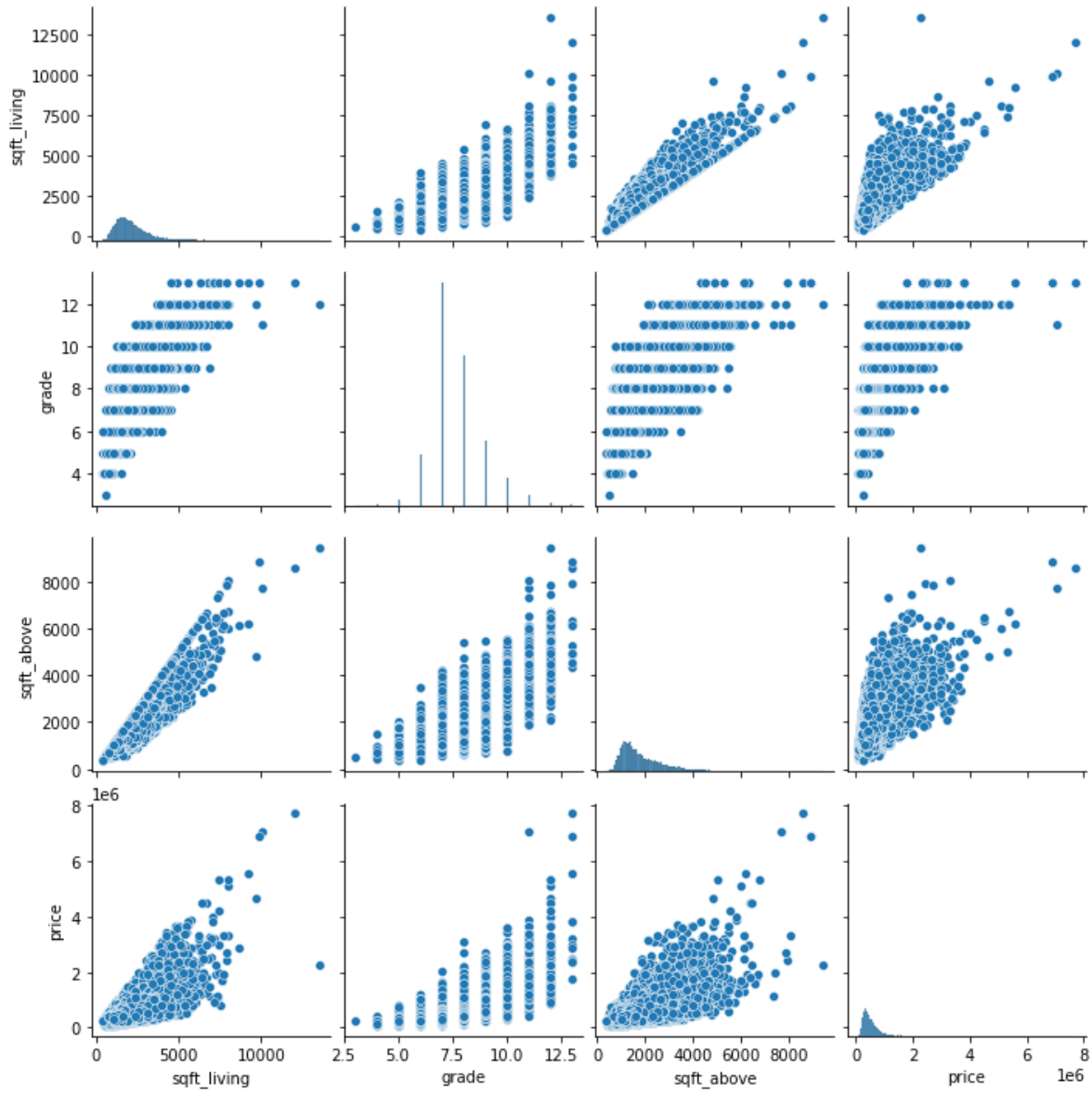
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sqft_living  21597 non-null  Int64
 1   grade        21597 non-null  Int64
 2   sqft_above   21597 non-null  Int64
 3   price        21597 non-null  Int64
dtypes: Int64(4)
memory usage: 759.4 KB
```

In [25]:

```
# checking the graphs to see if there are any anomaly, looks okay so far
sns.pairplot(df_sm)
```

Out[25]:

```
<seaborn.axisgrid.PairGrid at 0x2aa88440fa0>
```



In [26]:

```
#Build a baseline model with just a few variables

X = df_sm.drop('price', axis = 1)
y = df_sm['price']
```

In [27]:

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, rando
```

In [28]:

```
X_train.head()
```

Out[28]:

	sqft_living	grade	sqft_above
6405	1880	8	1880
937	2020	7	1310
19076	4720	9	3960
15201	1430	7	1430
13083	2270	8	1740

In [29]:

```
y_train.head()
```

Out[29]:

```
6405     529000
937      253000
19076    745000
15201    545000
13083    390000
Name: price, dtype: Int64
```

In [30]:

```
#Check np array shape
print(X_train.shape, X_val.shape)
print(y_train.shape, y_val.shape)
```

```
(16197, 3) (5400, 3)
(16197,) (5400,)
```

In [31]:

```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
```

In [32]:

```
linreg.fit(X_train, y_train)
```

Out[32]:

```
LinearRegression()
```

In [33]:

```
#Produce Prediction
y_train_preds = linreg.predict(X_train)
```

In [34]:

```
y_val_preds = linreg.predict(X_val)
```

In [35]:

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

r2_train = r2_score(y_train, y_train_preds)
r2_val = r2_score(y_val, y_val_preds)

mse_train = mean_squared_error(y_train, y_train_preds)
mse_val = mean_squared_error(y_val, y_val_preds)

print(f'R2 train: {r2_train} R2 validation: {r2_val}')
print(f'MSE train: {mse_train:.3e} R2 validation: {mse_val:.3e}')
```

```
R2 train: 0.5440277996976239 R2 validation: 0.5327398199189005
MSE train: 6.119e+10 R2 validation: 6.411e+10

C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:
95: FutureWarning: Arrays of bytes/strings is being converted to decimal num
bers if dtype='numeric'. This behavior is deprecated in 0.24 and will be rem
oved in 1.1 (renaming of 0.26). Please convert your data to numeric values e
xplicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:
95: FutureWarning: Arrays of bytes/strings is being converted to decimal num
bers if dtype='numeric'. This behavior is deprecated in 0.24 and will be rem
oved in 1.1 (renaming of 0.26). Please convert your data to numeric values e
xplicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:
95: FutureWarning: Arrays of bytes/strings is being converted to decimal num
bers if dtype='numeric'. This behavior is deprecated in 0.24 and will be rem
oved in 1.1 (renaming of 0.26). Please convert your data to numeric values e
xplicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:
95: FutureWarning: Arrays of bytes/strings is being converted to decimal num
bers if dtype='numeric'. This behavior is deprecated in 0.24 and will be rem
oved in 1.1 (renaming of 0.26). Please convert your data to numeric values e
xplicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
```

In [36]:

```
#Adjusted R2
def adj_r2_score(r2, n_samples, n_regressors):
    factor = (n_samples-1)/ (n_samples - n_regressors - 1)
    return 1 - ((1-r2) * factor)
```

In [37]:

```
adj_r2_train = adj_r2_score(r2_train, len(X_train), len(X_train.columns.tolist()))
adj_r2_val = adj_r2_score(r2_val, len(X_val), len(X_train.columns.tolist()))

print(f'Adj. R2 train: {adj_r2_train} Adj. R2 validation: {adj_r2_val}')
#Not much improvement
```

```
Adj. R2 train: 0.5439433238993835 Adj. R2 validation: 0.5324800384992854
```

In [38]:

```
sns.scatterplot(x=y_train_preds, y=y_train)
sns.lineplot(x=y_train_preds, y=y_train_preds, color='red')
plt.title('Actual vs. Predicted Sale Price (Training Set)')
plt.xlabel('Predicted ($)')
plt.ylabel('Actual ($)')
```

Out[38]:

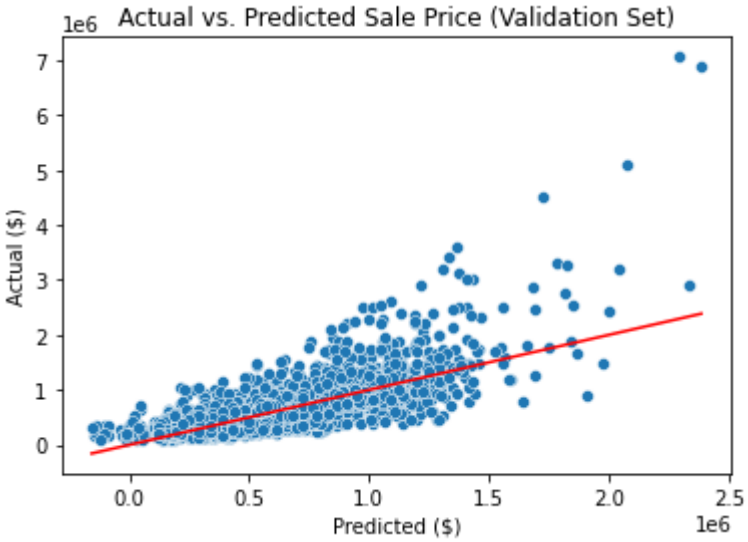
```
Text(0, 0.5, 'Actual ($)')
```

In [39]:

```
y_val_preds = linreg.predict(X_val)
sns.scatterplot(x=y_val_preds, y=y_val)
sns.lineplot(x=y_val_preds, y=y_val_preds, color='red')
plt.title('Actual vs. Predicted Sale Price (Validation Set)')
plt.xlabel('Predicted ($)')
plt.ylabel('Actual ($)')
```

Out[39]:

```
Text(0, 0.5, 'Actual ($)')
```

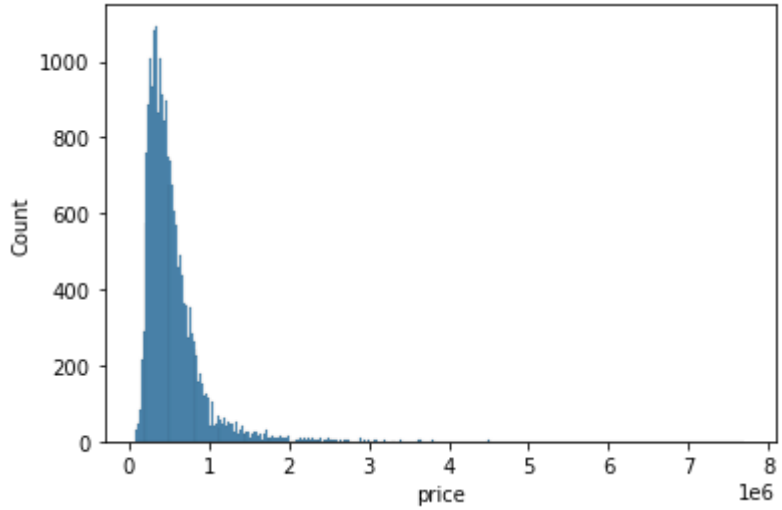


In [40]:

```
sns.histplot(y)
```

Out[40]:

```
<AxesSubplot:xlabel='price', ylabel='Count'>
```

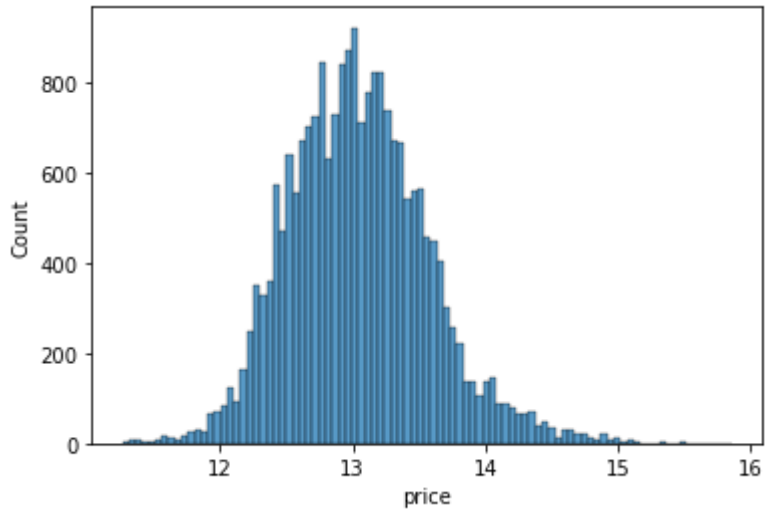


In [41]:

```
#Log transform due to some outliers
y = np.log1p(y) #log 1 + number
sns.histplot(y)
```

Out[41]:

```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



In [42]:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.2, random

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_train_preds = linreg.predict(X_train)
y_val_preds = linreg.predict(X_val)
```

In [43]:

```
#Checking if the train set improved
sns.scatterplot(x=y_train_preds, y=y_train)
sns.lineplot(x=y_train_preds, y=y_train_preds, color='red')
plt.title('Actual vs. Predicted Sale Price (Training Set)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
#Much better
```

Out[43]:

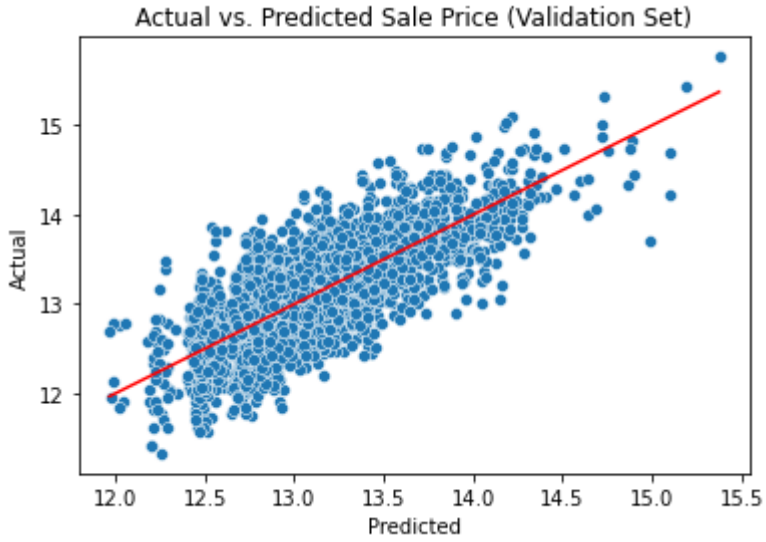
```
Text(0, 0.5, 'Actual')
```

In [44]:

```
#Checking if the test/validation set improved
y_val_preds = linreg.predict(X_val)
sns.scatterplot(x=y_val_preds, y=y_val)
sns.lineplot(x=y_val_preds, y=y_val_preds, color='red')
plt.title('Actual vs. Predicted Sale Price (Validation Set)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

Out[44]:

```
Text(0, 0.5, 'Actual')
```



In [45]:

```
#Cross Validation CV Test
from sklearn.model_selection import cross_validate
scores = cross_validate(linreg, X.astype(float), y.astype(float), cv=5,
                        scoring=['r2', 'neg_mean_squared_error'],
                        return_train_score=True)
scores
```

Out[45]:

```
{'fit_time': array([0.        , 0.        , 0.        , 0.01562548, 0.
]),
 'score_time': array([0., 0., 0., 0., 0.]),
 'test_r2': array([0.5562967 , 0.54959361, 0.53890325, 0.56291139, 0.6076517
9]),
 'train_r2': array([0.56633812, 0.56773869, 0.57012636, 0.56462174, 0.552503
09]),
 'test_neg_mean_squared_error': array([-0.12782403, -0.1243565 , -0.1228835
1, -0.12618492, -0.10308573]),
 'train_neg_mean_squared_error': array([-0.11904068, -0.11992144, -0.1202776
, -0.11945443, -0.12524392])}
```

In [46]:

```
#CV average score
avg_scores = {k:np.mean(v) for k, v in scores.items()}
avg_scores
```

Out[46]:

```
{'fit_time': 0.0031250953674316407,
 'score_time': 0.0,
 'test_r2': 0.5630713505419729,
 'train_r2': 0.5642655993438318,
 'test_neg_mean_squared_error': -0.12086693845134573,
 'train_neg_mean_squared_error': -0.12078761410254946}
```

In [47]:

```
#Create evaluation list for all of validation test

eval_list = list()
avg_scores['dataset'] = 'base'
avg_scores['n_features'] = len(X.columns)
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
```

Out[47]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.0	0.563071	0.564266	-0.120867	

In [48]:

```
#Define the function as we will be using the above metrics over and over using different va

def get_avg_cv_scores(X, y, regmodel, setname, scoring=None):
    if scoring is None:
        scoring = ['r2', 'neg_mean_squared_error']
    scores = cross_validate(regmodel, X, y, cv=5,
                            scoring=scoring,
                            return_train_score=True)
    avg_scores = {k:np.mean(v) for k, v in scores.items()}
    avg_scores['dataset'] = setname
    avg_scores['n_features'] = len(X.columns)
    return avg_scores
```

In [49]:

```
# Introduce new continuous features = all of the sqft, yr_build, yr_renovated, lat, long
# Then analyse to see which one to include and exclude

cont_vars = ['sqft_lot', 'sqft_above', 'yr_built', 'yr_renovated', 'lat', 'long']
```

In [50]:

```
df_cont = kc_data[cont_vars + ['price']]
df_cont.head()
```

Out[50]:

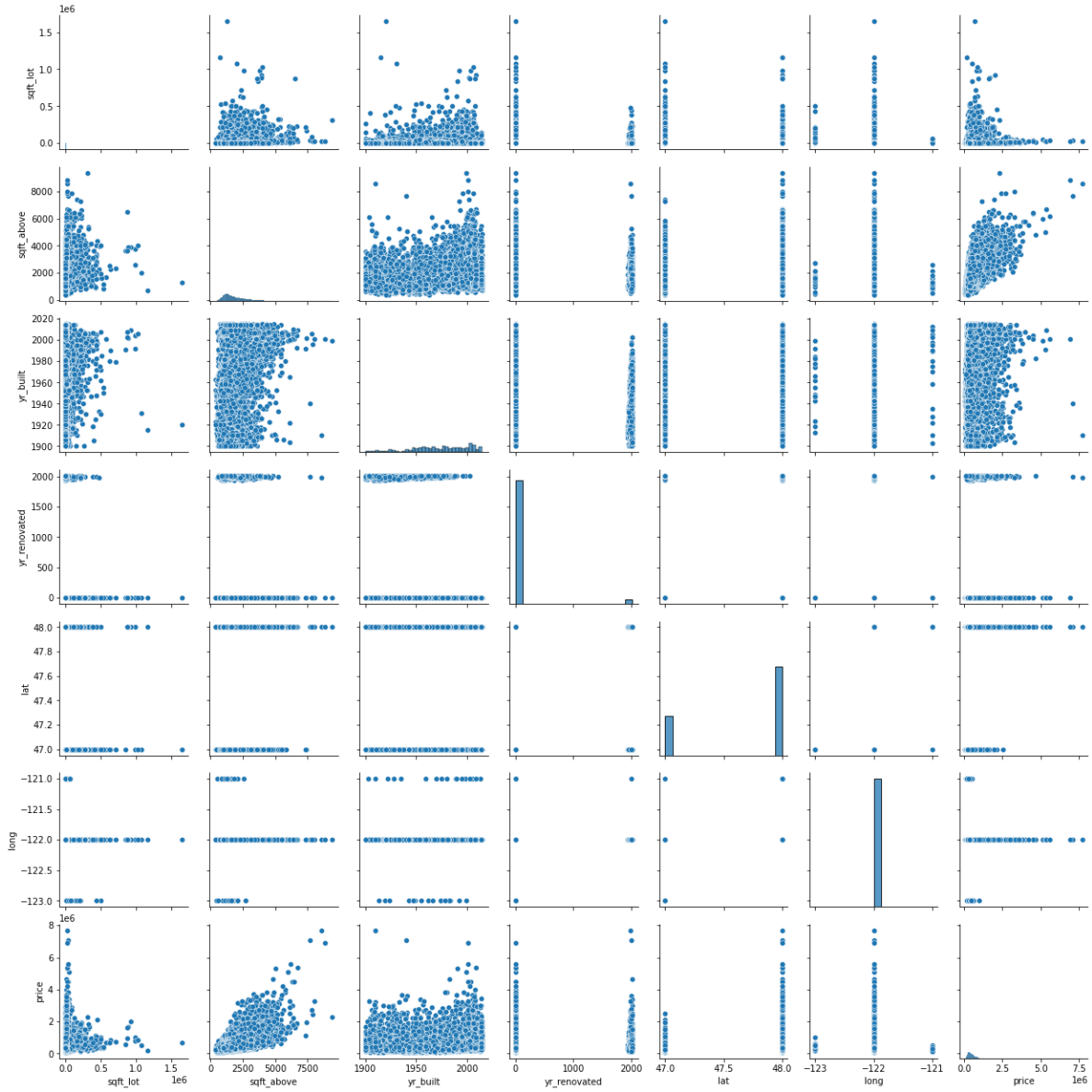
	sqft_lot	sqft_above	yr_built	yr_renovated	lat	long	price
0	5650	1180	1955	0	48	-122	221900
1	7242	2170	1951	1991	48	-122	538000
2	10000	770	1933	0	48	-122	180000
3	5000	1050	1965	0	48	-122	604000
4	8080	1680	1987	0	48	-122	510000

In [51]:

```
#Lets check the plot
sns.pairplot(df_cont)
```

Out[51]:

```
<seaborn.axisgrid.PairGrid at 0x2aa8cd42f10>
```



In [52]:

```
kc_data[cont_vars].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sqft_lot      21597 non-null  Int64
 1   sqft_above    21597 non-null  Int64
 2   yr_built      21597 non-null  Int64
 3   yr_renovated  21597 non-null  Int64
 4   lat           21597 non-null  Int64
 5   long          21597 non-null  Int64
dtypes: Int64(6)
memory usage: 1.1 MB
```

In [53]:

```
kc_data[cont_vars].describe()
```

Out[53]:

	sqft_lot	sqft_above	yr_built	yr_renovated	lat	long
count	2.159700e+04	21597.000000	21597.000000	21597.000000	21597.000000	21597.000000
mean	1.509941e+04	1788.596842	1970.999676	68.758207	47.694356	-122.000046
std	4.141264e+04	827.759761	29.375234	364.037499	0.460690	0.040258
min	5.200000e+02	370.000000	1900.000000	0.000000	47.000000	-123.000000
25%	5.040000e+03	1190.000000	1951.000000	0.000000	47.000000	-122.000000
50%	7.618000e+03	1560.000000	1975.000000	0.000000	48.000000	-122.000000
75%	1.068500e+04	2210.000000	1997.000000	0.000000	48.000000	-122.000000
max	1.651359e+06	9410.000000	2015.000000	2015.000000	48.000000	-121.000000

In [54]:

```
#Check for multicolinearity

kc_data[cont_vars].corr()[abs(kc_data[cont_vars].corr()) > 0.7]

#sqft_living and sqft_above have high multicolinearity, I will be removing one of them
```

Out[54]:

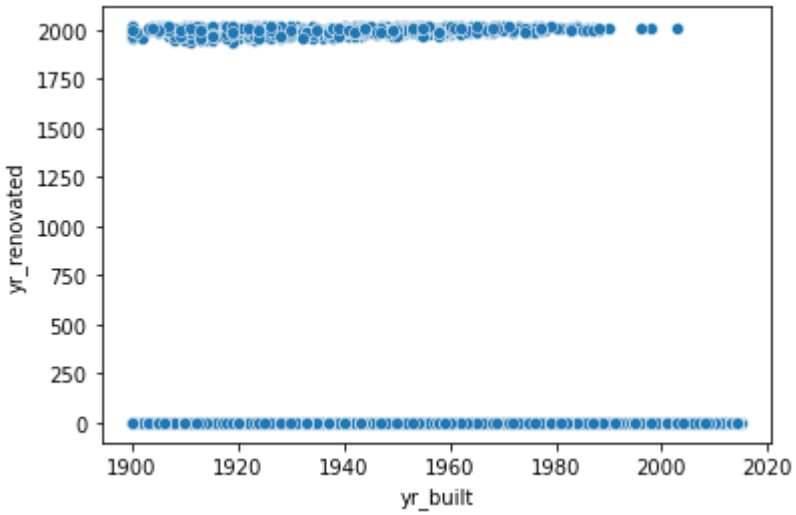
	sqft_lot	sqft_above	yr_built	yr_renovated	lat	long
sqft_lot	1.0	NaN	NaN	NaN	NaN	NaN
sqft_above	NaN	1.0	NaN	NaN	NaN	NaN
yr_built	NaN	NaN	1.0	NaN	NaN	NaN
yr_renovated	NaN	NaN	NaN	1.0	NaN	NaN
lat	NaN	NaN	NaN	NaN	1.0	NaN
long	NaN	NaN	NaN	NaN	NaN	1.0

In [55]:

```
sns.scatterplot(x=kc_data['yr_built'], y=kc_data['yr_renovated'])
```

Out[55]:

```
<AxesSubplot:xlabel='yr_built', ylabel='yr_renovated'>
```



In [56]:

```
#Lets add base+cont_vars to our linreg test and remove sqft_above due to high colinearity w
cont_vars.remove('sqft_above')
X = kc_data[base_vars + cont_vars]
X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, rando
```

In [57]:

```
#Check np array shape
print(X_train.shape, X_val.shape)
print(y_train.shape, y_val.shape)
```

```
(16197, 8) (5400, 8)
(16197,) (5400,)
```

In [58]:

```
linreg = LinearRegression()
avg_scores = get_avg_cv_scores(X.astype(float), y.astype(float), linreg, 'w/cont. feats')
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
#Pretty good improvement for R2 from 0.56 to 0.75
```

Out[58]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.000000	0.563071	0.564266	-0.120867	
1	0.007073	0.003825	0.754208	0.756035	-0.067907	

In [59]:

```
#Running a quick info, so I can grab the name of features on the next step
kc_data.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  Int64
 3   bedrooms       21597 non-null  Int64
 4   bathrooms      21597 non-null  Int64
 5   sqft_living    21597 non-null  Int64
 6   sqft_lot       21597 non-null  Int64
 7   floors         21597 non-null  Int64
 8   waterfront     21597 non-null  Int64
 9   view           21597 non-null  Int64
 10  condition      21597 non-null  Int64
 11  grade          21597 non-null  Int64
 12  sqft_above     21597 non-null  Int64
 13  sqft_basement  21597 non-null  Int64
 14  yr_built       21597 non-null  Int64
 15  yr_renovated   21597 non-null  Int64
 16  zipcode        21597 non-null  Int64
 17  lat            21597 non-null  Int64
 18  long           21597 non-null  Int64
 19  sqft_living15  21597 non-null  Int64
 20  sqft_lot15     21597 non-null  Int64
dtypes: Int64(20), object(1)
memory usage: 3.9+ MB
```

In [60]:

```
# Now I am creating discrete variables, exclude 'grade' as it is already part of base_vars

disc_vars = ['condition', 'bedrooms', 'bathrooms', 'floors', 'view']
kc_data[disc_vars].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   condition  21597 non-null  Int64
 1   bedrooms   21597 non-null  Int64
 2   bathrooms  21597 non-null  Int64
 3   floors     21597 non-null  Int64
 4   view       21597 non-null  Int64
dtypes: Int64(5)
memory usage: 949.2 KB
```

In [61]:

```
kc_data[disc_vars].describe()
```

Out[61]:

	condition	bedrooms	bathrooms	floors	view
count	21597.000000	21597.000000	21597.000000	21597.000000	21597.000000
mean	3.409825	3.373200	2.059777	1.534750	0.233181
std	0.650546	0.926299	0.754435	0.554376	0.764673
min	1.000000	1.000000	0.000000	1.000000	0.000000
25%	3.000000	3.000000	2.000000	1.000000	0.000000
50%	3.000000	3.000000	2.000000	2.000000	0.000000
75%	4.000000	4.000000	2.000000	2.000000	0.000000
max	5.000000	33.000000	8.000000	4.000000	4.000000

In [62]:

```
df_disc = kc_data[disc_vars + ['price']]
df_disc.head()
```

Out[62]:

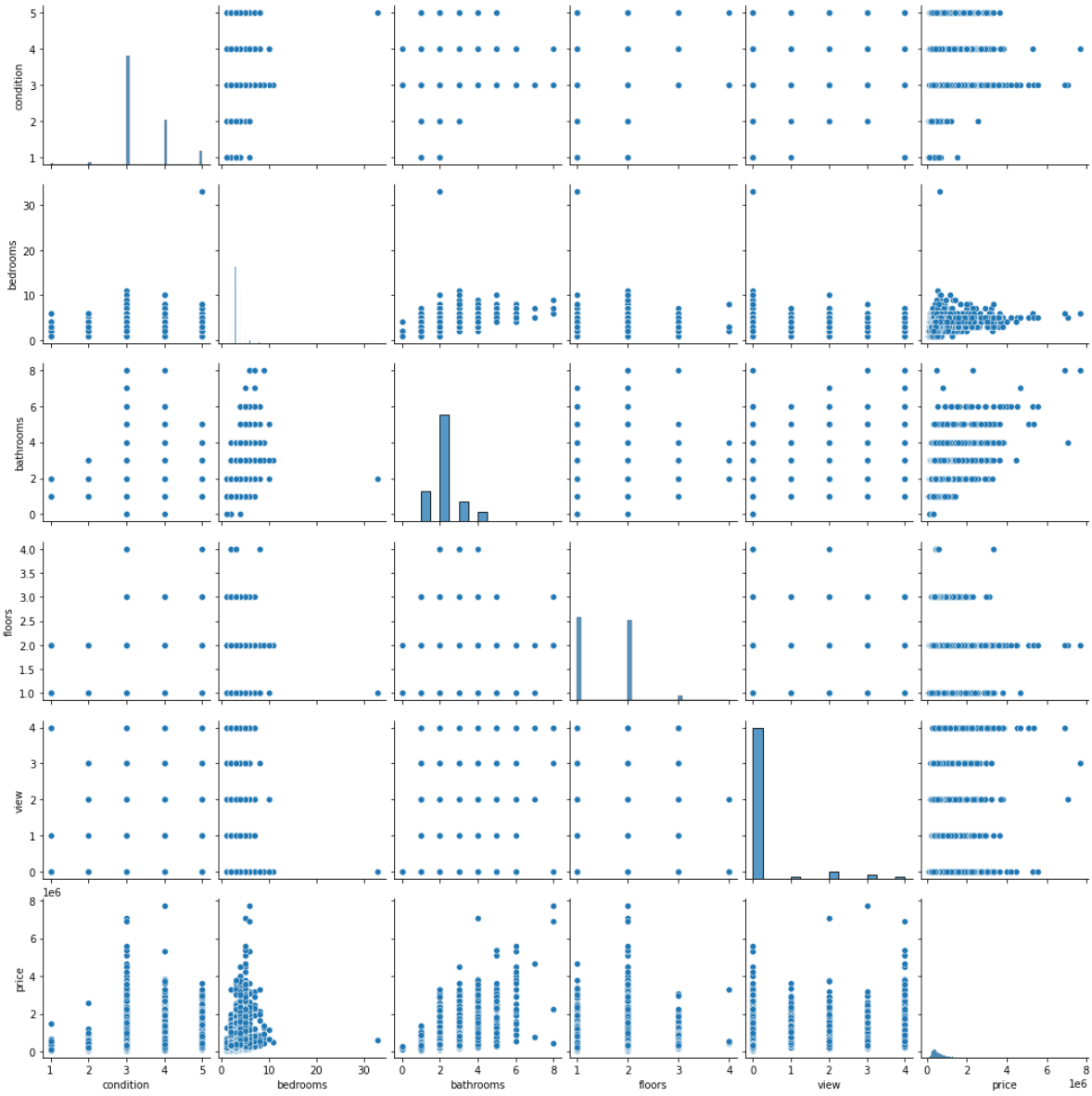
	condition	bedrooms	bathrooms	floors	view	price
0	3	3	1	1	0	221900
1	3	3	2	2	0	538000
2	3	2	1	1	0	180000
3	5	4	3	1	0	604000
4	3	3	2	1	0	510000

In [63]:

```
sns.pairplot(df_disc)
```

Out[63]:

```
<seaborn.axisgrid.PairGrid at 0x2aa92dfa2e0>
```



In [64]:

```
X = kc_data[base_vars + cont_vars + disc_vars]
X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, rando
```

In [65]:

```
print(X_train.shape, X_val.shape)
print(y_train.shape, y_val.shape)
```

```
(16197, 13) (5400, 13)
(16197,) (5400,)
```

In [66]:

```
linreg = LinearRegression()
avg_scores = get_avg_cv_scores(X.astype(float), y.astype(float), linreg, 'base+cont+disc. f
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
#R2 score just slightly better from 0.756 to 0.775 with 13 features
```

Out[66]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.000000	0.563071	0.564266	-0.120867	
1	0.007073	0.003825	0.754208	0.756035	-0.067907	
2	0.015622	0.000000	0.773267	0.775057	-0.062616	

In [67]:

```
#Next let's try RFE Recursive Feature Elimination with 8 features
from sklearn.feature_selection import RFE

X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, rando
n_feats = 8
selector = RFE(linreg, n_features_to_select=n_feats, step=1)
selector = selector.fit(X_train, y_train)
```

In [68]:

```
selector.ranking_
```

Out[68]:

```
array([3, 1, 5, 6, 2, 4, 1, 1, 1, 1, 1, 1, 1])
```

In [69]:

```
selector.support_
```

Out[69]:

```
array([False,  True, False, False, False, False,  True,  True,  True,
        True,  True,  True,  True])
```

In [70]:

```
top_feats = X.columns[selector.support_]
top_feats
```

Out[70]:

```
Index(['grade', 'lat', 'long', 'condition', 'bedrooms', 'bathrooms', 'floor
s',
       'view'],
      dtype='object')
```

In [71]:

```
X = kc_data[cont_vars + base_vars + disc_vars]
X = X[top_feats]
X.head()
```

Out[71]:

	grade	lat	long	condition	bedrooms	bathrooms	floors	view
0	7	48	-122	3	3	1	1	0
1	7	48	-122	3	3	2	2	0
2	6	48	-122	3	2	1	1	0
3	7	48	-122	5	4	3	1	0
4	8	48	-122	3	3	2	1	0

In [72]:

```
linreg = LinearRegression()
avg_scores = get_avg_cv_scores(X.astype(float), y.astype(float), linreg, 'RFE8')
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
#actually worse from 0.775 to 0.596
```

Out[72]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.000000	0.563071	0.564266	-0.120867	
1	0.007073	0.003825	0.754208	0.756035	-0.067907	
2	0.015622	0.000000	0.773267	0.775057	-0.062616	
3	0.006255	0.009627	0.721305	0.724734	-0.076916	

In [73]:

```
#Testing on RFE to see how many features would produce the best scores

X = kc_data[cont_vars + base_vars + disc_vars]

train_r2_scores = list()
train_adj_r2_scores = list()
train_mse_values = list()

test_r2_scores = list()
test_adj_r2_scores = list()
test_mse_values = list()

n_feat_values = list()

for n_feats in range(3, len(X.columns)):
    X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, r
    selector = RFE(linreg, n_features_to_select=n_feats, step=1)
    selector = selector.fit(X_train, y_train)

    top_feats = X.columns[selector.support_]

    X_sm = X[top_feats]
    X_train, X_val, y_train, y_val = train_test_split(X_sm, y, shuffle=True, test_size=0.25

    sm_linreg = LinearRegression()
    sm_linreg.fit(X_train, y_train)

    y_train_preds = sm_linreg.predict(X_train)
    y_val_preds = sm_linreg.predict(X_val)

    train_r2_scores.append(r2_score(y_train, y_train_preds))
    train_mse_values.append(mean_squared_error(y_train, y_train_preds))

    test_r2_scores.append(r2_score(y_val, y_val_preds))
    test_mse_values.append(mean_squared_error(y_val, y_val_preds))

    n_feat_values.append(n_feats)
```

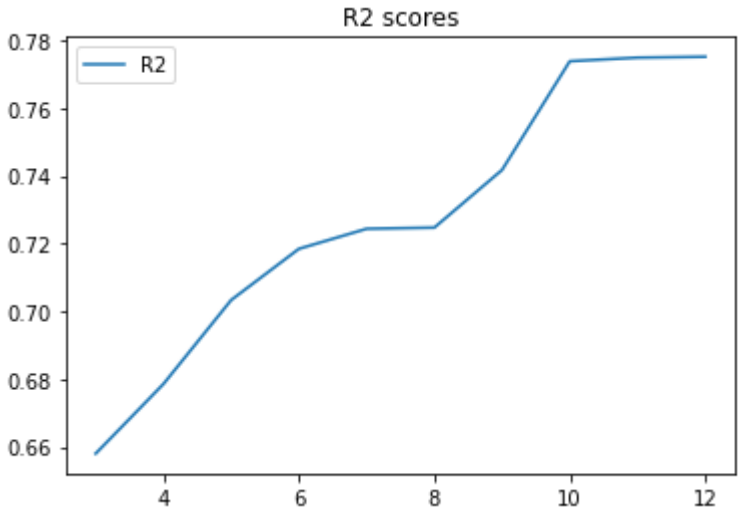
```
al numbers if dtype='numeric'. This behavior is deprecated in 0.24 and wi
ll be removed in 1.1 (renaming of 0.26). Please convert your data to nume
ric values explicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.
py:95: FutureWarning: Arrays of bytes/strings is being converted to decim
al numbers if dtype='numeric'. This behavior is deprecated in 0.24 and wi
ll be removed in 1.1 (renaming of 0.26). Please convert your data to nume
ric values explicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.
py:95: FutureWarning: Arrays of bytes/strings is being converted to decim
al numbers if dtype='numeric'. This behavior is deprecated in 0.24 and wi
ll be removed in 1.1 (renaming of 0.26). Please convert your data to nume
ric values explicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.
py:95: FutureWarning: Arrays of bytes/strings is being converted to decim
al numbers if dtype='numeric'. This behavior is deprecated in 0.24 and wi
```

In [74]:

```
sns.lineplot(x=n_feat_values, y=train_r2_scores, label='R2')
plt.title('R2 scores')
#Looks like 10 features would produce the best score, lets put it to a test next
```

Out[74]:

```
Text(0.5, 1.0, 'R2 scores')
```



In [75]:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, rando
n_feats = 10
selector = RFE(linreg, n_features_to_select=n_feats, step=1)
selector = selector.fit(X_train, y_train)
```

In [76]:

```
selector.ranking_
```

Out[76]:

```
array([4, 1, 2, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1])
```

In [77]:

```
selector.support_
```

Out[77]:

```
array([False,  True, False,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True])
```

In [78]:

```
top_feats = X.columns[selector.support_]
top_feats
```

Out[78]:

```
Index(['yr_built', 'lat', 'long', 'sqft_living', 'grade', 'condition',
       'bedrooms', 'bathrooms', 'floors', 'view'],
      dtype='object')
```

In [79]:

```
X = kc_data[cont_vars + base_vars + disc_vars]
X = X[top_feats]

linreg = LinearRegression()
avg_scores = get_avg_cv_scores(X.astype(float), y.astype(float), linreg, 'RFE10')
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
#Much better just slightly less than 13 features
```

Out[79]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.000000	0.563071	0.564266	-0.120867	
1	0.007073	0.003825	0.754208	0.756035	-0.067907	
2	0.015622	0.000000	0.773267	0.775057	-0.062616	
3	0.006255	0.009627	0.721305	0.724734	-0.076916	
4	0.009075	0.003728	0.771318	0.772831	-0.063166	

In [80]:

```
# Now lets add categorical features, only 2 features left that we haven't use 'waterfront'
# I am not using ID & Date at this stage

cat_vars = ['waterfront', 'zipcode']
kc_data[cat_vars].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   waterfront  21597 non-null  Int64
 1   zipcode     21597 non-null  Int64
dtypes: Int64(2)
memory usage: 379.8 KB
```

In [81]:

```
#I have cleaned the data at the very beginning, but will double-check if there's any missin
kc_data[cat_vars].isnull().sum()
#No missing value
```

Out[81]:

```
waterfront    0
zipcode       0
dtype: int64
```

In [82]:

```
kc_data[cat_vars].describe()
```

Out[82]:

	waterfront	zipcode
count	21597.000000	21597.000000
mean	0.006760	98077.951845
std	0.081944	53.513072
min	0.000000	98001.000000
25%	0.000000	98033.000000
50%	0.000000	98065.000000
75%	0.000000	98118.000000
max	1.000000	98199.000000

In [83]:

```
#Now I am using OneHotEncoder to help transform categorical datas/features

from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(drop='first')
enc.fit(kc_data[cat_vars])
ohe_cols = enc.transform(kc_data[cat_vars])
ohe_cols.toarray().shape
```

Out[83]:

```
(21597, 70)
```

In [84]:

```
ohe_df = pd.DataFrame(ohe_cols.toarray(), columns=enc.get_feature_names_out(cat_vars))
ohe_df.head()
```

Out[84]:

	waterfront_1	zipcode_98002	zipcode_98003	zipcode_98004	zipcode_98005	zipcode_98006	z
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 70 columns

In [85]:

```
#Combined OHE with the last dataframe that I used before RFE
X = kc_data[base_vars + cont_vars + disc_vars].join(ohe_df)
X.head()
```

Out[85]:

	sqft_living	grade	sqft_above	sqft_lot	yr_built	yr_renovated	lat	long	condition	bedroon
0	1180	7	1180	5650	1955	0	48	-122	3	
1	2570	7	2170	7242	1951	1991	48	-122	3	
2	770	6	770	10000	1933	0	48	-122	3	
3	1960	7	1050	5000	1965	0	48	-122	5	
4	1680	8	1680	8080	1987	0	48	-122	3	

5 rows × 83 columns

In [86]:

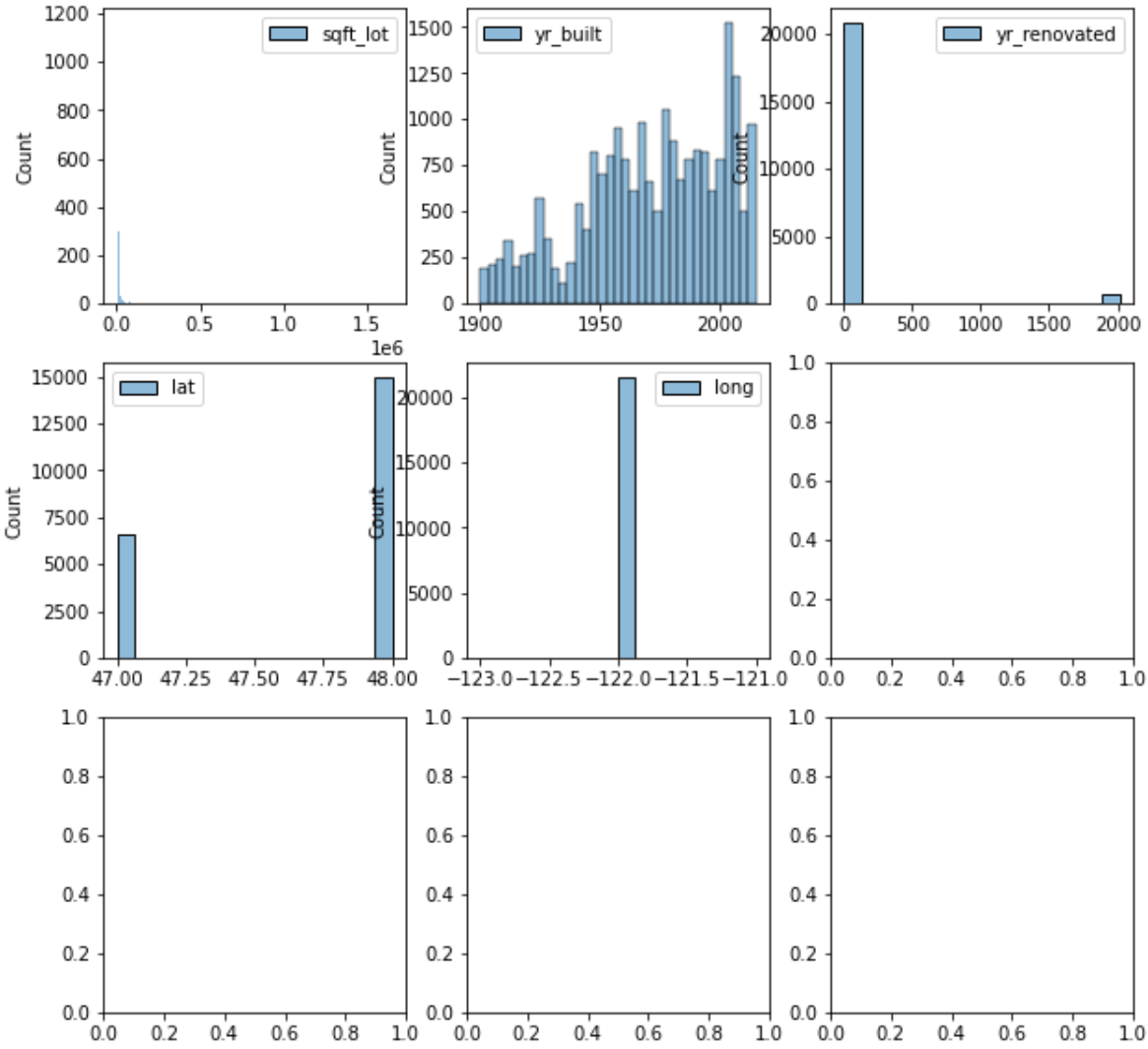
```
linreg = LinearRegression()
avg_scores = get_avg_cv_scores(X.astype(float), y.astype(float), linreg, 'all variables')
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
#Great result with R2 from 0.77 to 0.87, proven that categorial variables such as zipcode a
```

Out[86]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.000000	0.563071	0.564266	-0.120867	
1	0.007073	0.003825	0.754208	0.756035	-0.067907	
2	0.015622	0.000000	0.773267	0.775057	-0.062616	
3	0.006255	0.009627	0.721305	0.724734	-0.076916	
4	0.009075	0.003728	0.771318	0.772831	-0.063166	
5	0.065622	0.003116	0.870394	0.873057	-0.035793	

In [87]:

```
#Now we will transform our input variables, also a quick check on continuous variable to se
#similar pattern that we can get rid off in order to produce better result

fig, ax = plt.subplots(3, 3, figsize=(10, 10))
for i in range(len(cont_vars)):
    sns.histplot(kc_data[[cont_vars[i]]], ax=ax[i // 3, i % 3])
```



In [88]:

```
#Pattern seems okay, however sqft_lot graph doesn't look quite right, will check the data q

kc_data['sqft_lot'].value_counts(dropna=False)
#value below look normal, maybe just the graph tend to clump all the data together due to t
```

Out[88]:

```
5000     358
6000     290
4000     251
7200     220
4800     119
        ...
20672      1
3434       1
914        1
5855       1
<NA>       0
Name: sqft_lot, Length: 9777, dtype: Int64
```

In [89]:

```
#Next I am going to log transform all of the columns due to some skewness in the data, as w
#so to speak to keep the score/value consistents, not using 'long' as log doesn't work on n
long_vars = ['long']
log_vars = [c for c in cont_vars if c not in long_vars]
X = np.log1p(kc_data[log_vars]).join(kc_data[long_vars + base_vars + disc_vars]).join(ohe_d

X.head()
```

Out[89]:

	sqft_lot	yr_built	yr_renovated	lat	long	sqft_living	grade	sqft_above	condition	l
0	8.639588	7.578657	0.0	3.89182	-122	1180	7	1180	3	
1	8.887791	7.57661	7.596894	3.89182	-122	2570	7	2170	3	
2	9.21044	7.567346	0.0	3.89182	-122	770	6	770	3	
3	8.517393	7.583756	0.0	3.89182	-122	1960	7	1050	5	
4	8.997271	7.594884	0.0	3.89182	-122	1680	8	1680	3	

5 rows × 83 columns

In [90]:

```
linreg = LinearRegression()
avg_scores = get_avg_cv_scores(X.astype(float), y.astype(float), linreg, 'all variables log
eval_list.append(avg_scores)
pd.DataFrame(eval_list)
#slightly better after the log, I am happy with the outcome of R2 at 0.876
```

Out[90]:

	fit_time	score_time	test_r2	train_r2	test_neg_mean_squared_error	train_neg_mean_squ
0	0.003125	0.000000	0.563071	0.564266	-0.120867	
1	0.007073	0.003825	0.754208	0.756035	-0.067907	
2	0.015622	0.000000	0.773267	0.775057	-0.062616	
3	0.006255	0.009627	0.721305	0.724734	-0.076916	
4	0.009075	0.003728	0.771318	0.772831	-0.063166	
5	0.065622	0.003116	0.870394	0.873057	-0.035793	
6	0.061071	0.007663	0.873578	0.876233	-0.034917	

In [91]:

```
#Lets do last check on our model
X_train, X_val, y_train, y_val = train_test_split(X, y, shuffle=True, test_size=0.25, rando
linreg.fit(X_train, y_train)
```

Out[91]:

```
LinearRegression()
```

In [92]:

```
#Train set
sns.scatterplot(x=y_train_preds, y=y_train)
sns.lineplot(x=y_train_preds, y=y_train_preds, color='red')
plt.title('Actual vs. Predicted Sale Price (Training Set)')
plt.xlabel('Predicted ($)')
plt.ylabel('Actual ($)')
```

Out[92]:

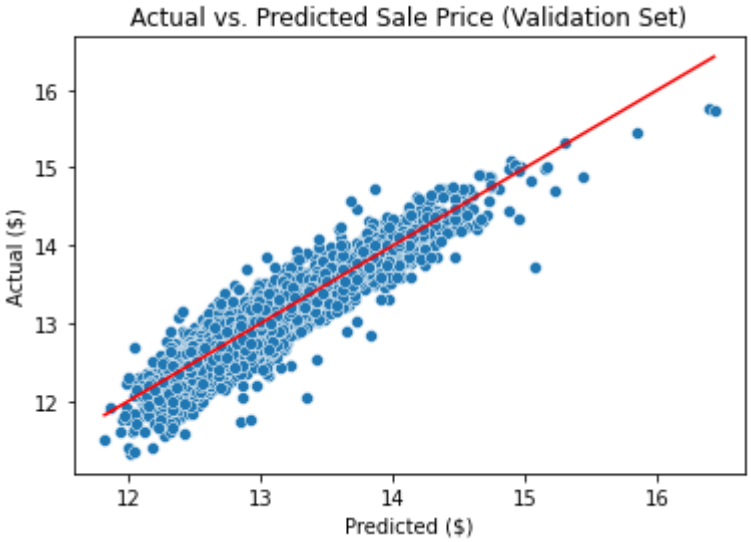
```
Text(0, 0.5, 'Actual ($)')
```

In [93]:

```
#Validation set
y_val_preds = linreg.predict(X_val)
sns.scatterplot(x=y_val_preds, y=y_val)
sns.lineplot(x=y_val_preds, y=y_val_preds, color='red')
plt.title('Actual vs. Predicted Sale Price (Validation Set)')
plt.xlabel('Predicted ($)')
plt.ylabel('Actual ($)')
```

Out[93]:

```
Text(0, 0.5, 'Actual ($)')
```



In [94]:

```
#Next we will run the linear model in 'Price' as the target variable in statsmodel
#to see if the results are similar to our previous R2 & MSE analysis
import statsmodels.api as sm
from statsmodels.formula.api import ols

X_int = sm.add_constant(X)
model = sm.OLS(y.astype(float), X_int.astype(float)).fit()
model.summary()
```

Out[94]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.876
Model:	OLS	**Adj. R-squared:**	0.876
Method:	Least Squares	**F-statistic:**	1832.
Date:	Fri, 19 Aug 2022	**Prob (F-statistic):**	0.00
Time:	12:56:07	**Log-Likelihood:**	5753.7
No. Observations:	21597	**AIC:**	-1.134e+04
Df Residuals:	21513	**BIC:**	-1.067e+04
Df Model:	83		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**

In [95]:

```
# Another check on coefficients and intercept are the same as those from Statsmodels
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
linreg = LinearRegression()
linreg.fit(X, y)
y_pred = linreg.predict(X)
print(linreg.intercept_)
print(linreg.coef_)
print(r2_score(y, y_pred))
#Same results/scores = model is working
```

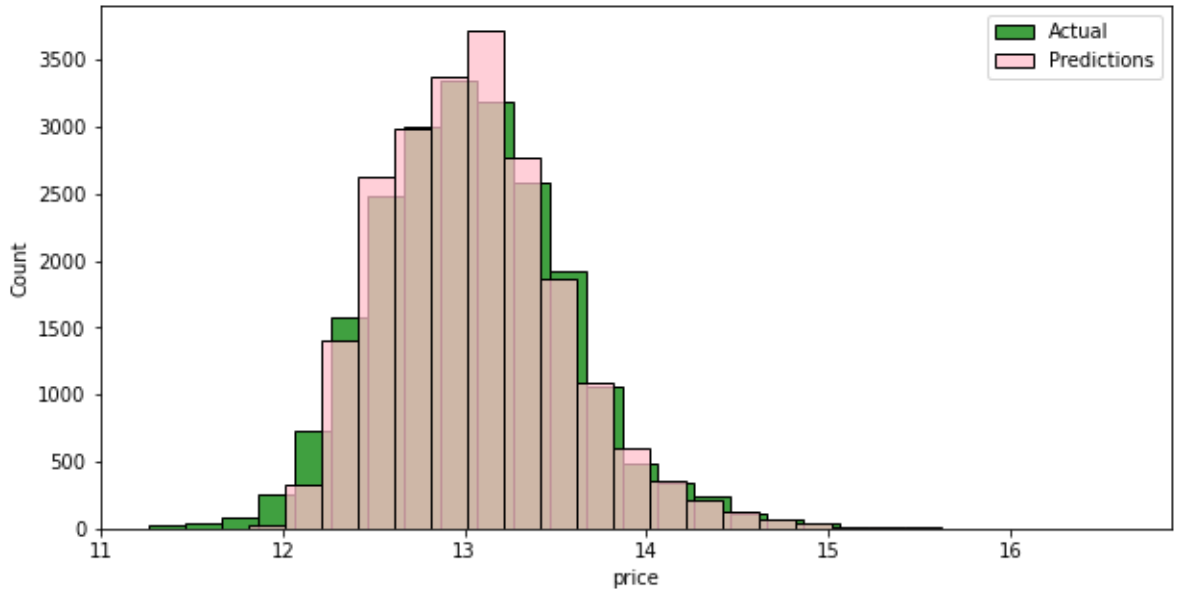
```
-29.316667304767506
[ 6.37053628e-02  1.45495064e-01  1.05034787e-02  6.80488016e+00
 -1.01412299e-01  1.53173404e-04  1.02719427e-01  6.17809613e-05
  5.62384062e-02  3.52723419e-03  1.81392871e-02  6.86746538e-03
  6.72532178e-02  4.32434859e-01 -2.66455728e-02  1.97668414e-02
  9.86580606e-01  6.10689755e-01  5.11590991e-01  5.23803399e-01
  5.22780521e-01  2.17184192e-01  3.35809211e-01  1.41774629e-01
  1.92558233e-01  2.90358295e-02 -2.01451898e-02  2.50739634e-01
  4.19016235e-01  2.96815799e-01  4.92836937e-01  6.78206266e-02
  8.61588237e-02 -3.78688712e-02  6.54531787e-01  4.13289039e-01
  1.91806946e-01  1.11156119e+00  7.46332251e-01  6.59442577e-02
  2.94573364e-01  5.19896683e-01  4.51751679e-01  1.55386297e-01
  2.31943705e-01  1.69503528e-01  3.13917899e-01  2.99625711e-01
  2.53516444e-01  3.48021235e-01  4.33371427e-01  4.41849968e-01
  2.88287936e-01  2.95053558e-02  8.55920476e-01  7.02465502e-01
  8.45185242e-01  1.99157961e-01  7.25142739e-01  2.36938049e-01
  8.96308710e-01  9.50997336e-01  6.99289898e-01  6.40571881e-01
  6.87145934e-01  3.35607639e-01  8.79594710e-01  7.02190111e-01
  4.31237964e-01  4.07062737e-01  3.19601426e-01  5.56299681e-01
  5.57656154e-01  1.80771954e-01  1.55880280e-01  2.84424353e-01
  3.02035430e-01  2.52148104e-02  4.66442356e-01  8.26929735e-02
  8.75519889e-02  5.92953541e-02  7.51275819e-01]
0.8760468797926251

C:\Users\student\anaconda3\lib\site-packages\sklearn\metrics\_regression.py:
95: FutureWarning: Arrays of bytes/strings is being converted to decimal num
bers if dtype='numeric'. This behavior is deprecated in 0.24 and will be rem
oved in 1.1 (renaming of 0.26). Please convert your data to numeric values e
xplicitly instead.
  y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
```

In [96]:

```
# As a final check, compare ranges of predictions to the actual values
fig, ax = plt.subplots(figsize=(10, 5))
sns.histplot(y, label='Actual', color='green', binwidth=0.2)
sns.histplot(y_pred, label='Predictions', color='pink', binwidth=0.2)
plt.legend()
#Results below are pretty close, great :)
```

Out[96]:

```
<matplotlib.legend.Legend at 0x2aa9c285c70>
```



Regression Results / Summary Output

Here are some of the important findings based on the results from above testing and modelling on different features and methods:

- Initially only 3 features stands out that have strong linear regression, which we used to build our base model (base_vars = 'sqft_living', 'grade', 'sqft_above')
- However the R2 result wasn't that great (at 0.56) when using the base variables only
- We know that there are other factors/parameters that will affect the overall house price

- So we used the RFE (Recursive Feature Elimination) method to calculate the most important variables. The results are 10 variables as follow:'yr_built', 'lat', 'long', 'sqft_living', 'grade', 'condition', 'bedrooms', 'bathrooms', 'floors', 'view'
- The R2 score from RFE method increased from 0.56 to 0.77 (increased of 19%, pretty impressive just by adding 7 more features)
- The final step is to add the categorical variables (zipcode and waterfront features) using OneHotEncoder, the result jumped from 0.77 to 0.87
- Great result, as we hear it all the time 'LOCATION, LOCATION, LOCATION' in real estate, and the modelling above proven by adding the location (zipcode) to the parameter/feature make a huge different on the R2 score and to minimise the MSE (Mean Square Error

Next Steps and Recommendation

- The linear regression model is now build, and we can implement the linear regression calculation into client's preferred interface, so the team at KC Financial Investment can drop in these features/parameters then it will calculate the predicted house price automatically
- The simple interface and calculation can be build into Microsoft Excel, Power BI, or Intranet (Internal site) for ease of use by the staff members. This is a great tool if you only have a handful of houses to analyse
- However as a starting point, we can analyse all of the properties available for sale in Kings County by scraping the data from multiple websites (such as realtor.com, homes.com etc) currently over 500 houses available in the market.
- Then we will present list of houses that are under the predicted price for KC Financial Investment to consider purchasing to start maximising their profit earnings.

In [ ]: