Final Project Submission

Please fill out:

- Student name: Peggy Obam, Tobias Ng'ong'a,John Gikonyo, Muthoni Kahura, Wambui Thuku,Phelix Okumu Catherine Gakii
- Student pace: Parttime DSP3
- Scheduled project review date/time:
- Instructor name:Group2
- Blog post URL:https://github.com/GikonyoJohn/dsc-phase-2-project-v2-3-group2 (https://github.com/GikonyoJohn/dsc-phase-2-project-v2-3-group2)

1.0 Introduction

This study employs linear regression analysis to deduce the extent to which specific variables influence housing prices

1.1 Project overview

The project aims to help the real estate agency to assist homeowners in making informed decisions about home renovations by utilizing the King County House Sales dataset. By analyzing the dataset, we can determine the influence of various factors on house prices, ultimately providing valuable insights to homeowners regarding the potential increase in the estimated value of their homes through different renovation choices.

####1.2 Objectives

1. To understand which factors determines the price of a home.
2. To understand how square feet living affect the value of a home.
3. To explore how condition affect the price of a home.
4. To explore which features will decrease and increase value of the house.

###1.3 Research questions

1. Which factors determines the price of a home?
2. How square feet living affect the value of a home?
3. How condition affect the price of a home?
4. Which features will decrease and increase value of the house?

1.4 Source & description

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.

Column Names and descriptions for Kings County Data Set

id -Unique identified for a house

date - Date house was sold

price - Price is prediction target

bedrooms - Number of Bedrooms/House

bathrooms - Number of bathrooms/bedrooms

sqft_living - Square footage of the home

sqft_lot - Square footage of the lot

floors - Total 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 ** **bold text - 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* -* Zipcode

lat - Latitude coordinate

long - Longitude coordinate

sqft_living15 - Square footage of interior housing living space for the nearest 15 neighbors

sqft_lot15 - Square footage of the land lots of the nearest 15 neighbors

##**2.0 Data understanding**

In [90]:

```
#importing different libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

import statsmodels.api as sm
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
from random import gauss
from mpl_toolkits.mplot3d import Axes3D
from scipy import stats as stats
%matplotlib inline
import os
os.makedirs('Images', exist_ok=True)
```

In [91]:

```
# Read the csv file using pandas, storing it in house_df dataframe and previewing the dat
house_df = pd.read_csv("kc_house_data.csv")
house_df
```

Out[91]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
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
...	...	...	...	...	...	...	...	...
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0

21597 rows × 21 columns

In [92]:

```
# check the shape to see the number of columns and row
house_df.shape
```

Out[92]:

```
(21597, 21)
```

The house_df has 21597 rows and 21 columns

In [93]:

```
# checking the general info about column data types
house_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
```

house_df has both numerical and categorical columns. date, waterfront, view, condition, grade and sqft_basement are categorical columns while the rest are numerical columns

In data cleaning, date should be converted to date.time data type. For later regression analysis, the categorical columns will be converted to numerical using OneHotEncoding or dummy

##3.0 Data preparation and cleaning per column

In [94]:

```
# creating a new df to use for purposes of cleaning
house_df2 = house_df
```

Generally cheking for missing values

In [95]:

```
# checking for %tage of null values across all columns
house_df2.isna().sum()/len(house_df2)*100
```

Out[95]:

```
id                0.000000
date              0.000000
price             0.000000
bedrooms          0.000000
bathrooms         0.000000
sqft_living       0.000000
sqft_lot          0.000000
floors            0.000000
waterfront       11.001528
view              0.291707
condition         0.000000
grade             0.000000
sqft_above        0.000000
sqft_basement     0.000000
yr_built          0.000000
yr_renovated     17.789508
zipcode           0.000000
lat               0.000000
long              0.000000
sqft_living15     0.000000
sqft_lot15        0.000000
dtype: float64
```

Waterfront, view and yr_renovated have 11%, 0.3% and 17.8% missing values respectively. Based on above, we will handle NANs in these columns differently in the subsequent cells

#####**View Column preparation**

In [96]:

```
# Checking view column for counts of ratings
house_df2['view'].value_counts()
```

Out[96]:

```
NONE         19422
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

From above, the NONE rating has way more values 19422. With this, it would make sense to drop this column in the regression

In [97]:

```
# dropping view column
house_df2 = house_df2.drop('view', axis=1)
```

In [98]:

```
house_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 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   condition      21597 non-null  object 
 10  grade          21597 non-null  object 
 11  sqft_above     21597 non-null  int64  
 12  sqft_basement  21597 non-null  object 
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   17755 non-null  float64
 15  zipcode        21597 non-null  int64  
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(9), object(5)
memory usage: 3.3+ MB
```

Date Column preparation

Changing the date column into date.time format/datatype

In [99]:

```
# creating function to change date column to datetime datatype
def convert_to_datetime(df, column_name):
    df[column_name] = pd.to_datetime(df[column_name])
    return df
```

In [100]:

```
# calling the fuction to the date column
convert_to_datetime(house_df2, 'date')
```

Out[100]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	conditio
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	NaN	Averag
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	NO	Averag
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	NO	Averag
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	NO	Ve Goo
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	NO	Averag
...	...	...	...	...	...	...	...	...	...	
21592	263000018	2014-	360000.0	3	2.50	1530	1131	3.0	NO	Averag

In [101]:

```
# checking if the date column data type has changed
house_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  datetime64[ns]
 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   condition      21597 non-null  object
 10  grade          21597 non-null  object
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21597 non-null  object
 13  yr_built       21597 non-null  int64
 14  yr_renovated   17755 non-null  float64
 15  zipcode        21597 non-null  int64
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64
 19  sqft_lot15     21597 non-null  int64
dtypes: datetime64[ns](1), float64(6), int64(9), object(4)
memory usage: 3.3+ MB
```

Date column has changed from an object to datetime datatype

In [102]:

```
# previewing the date column fo see head and tail
house_df2['date']
```

Out[102]:

```
0       2014-10-13
1       2014-12-09
2       2015-02-25
3       2014-12-09
4       2015-02-18
           ...
21592   2014-05-21
21593   2015-02-23
21594   2014-06-23
21595   2015-01-16
21596   2014-10-15
Name: date, Length: 21597, dtype: datetime64[ns]
```

Grade Column view and preparation

In [103]:

```
# checking for all the unique entries in the grade column
house_df2['grade'].unique()
```

Out[103]:

```
array(['7 Average', '6 Low Average', '8 Good', '11 Excellent', '9 Better',
       '5 Fair', '10 Very Good', '12 Luxury', '4 Low', '3 Poor',
       '13 Mansion'], dtype=object)
```

grade column is currently categorical. However, there are also numerical grading against every categorical grading

In [104]:

```
# doing value counts to get a view of the occurence of the unique entries
house_df2['grade'].value_counts()
```

Out[104]:

```
7 Average        8974
8 Good           6065
9 Better         2615
6 Low Average    2038
10 Very Good     1134
11 Excellent      399
5 Fair            242
12 Luxury          89
4 Low              27
13 Mansion         13
3 Poor              1
Name: grade, dtype: int64
```

For above grade data, we will change the strings to integer data points by removing the wordings and using already existing integers. Note: oneHotEncoding is avoided because the column already has a mixup of

In [105]:

```
# Extract numbers and drop words after the first space
house_df2['grade'] = house_df2['grade'].str.extract(r'(\d+)')
house_df2['grade'] = house_df2['grade'].astype(int)
```

In [106]:

```
# printing both unique entries of the grade column and general info about the dataframe t
print(house_df2['grade'].unique())
house_df2.info()
```

```
[ 7  6  8 11  9  5 10 12  4  3 13]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  datetime64[ns]
 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   condition      21597 non-null  object
 10  grade          21597 non-null  int32
 11  sqft_above     21597 non-null  int64
 12  sqft_basement  21597 non-null  object
 13  yr_built       21597 non-null  int64
 14  yr_renovated   17755 non-null  float64
 15  zipcode        21597 non-null  int64
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64
 19  sqft_lot15     21597 non-null  int64
dtypes: datetime64[ns](1), float64(6), int32(1), int64(9), object(3)
memory usage: 3.2+ MB
```

Grade column is changed to numerical data type (int64)

Id Column view and preparation

In [107]:

```
# dropping the duplicated id's by keeping the recent id when the house was sold
house_df2=house_df2.sort_values('id', ascending = False).drop_duplicates(subset = 'id', k
```

In [108]:

```
# checking if the duplicated id dropped
house_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21420 entries, 15937 to 2494
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21420 non-null  int64
 1   date           21420 non-null  datetime64[ns]
 2   price          21420 non-null  float64
 3   bedrooms       21420 non-null  int64
 4   bathrooms      21420 non-null  float64
 5   sqft_living    21420 non-null  int64
 6   sqft_lot       21420 non-null  int64
 7   floors         21420 non-null  float64
 8   waterfront     19065 non-null  object
 9   condition      21420 non-null  object
 10  grade          21420 non-null  int32
 11  sqft_above     21420 non-null  int64
 12  sqft_basement  21420 non-null  object
 13  yr_built       21420 non-null  int64
 14  yr_renovated   17610 non-null  float64
 15  zipcode        21420 non-null  int64
 16  lat            21420 non-null  float64
 17  long           21420 non-null  float64
 18  sqft_living15  21420 non-null  int64
 19  sqft_lot15     21420 non-null  int64
dtypes: datetime64[ns](1), float64(6), int32(1), int64(9), object(3)
memory usage: 3.4+ MB
```

Waterfront and Condition Columns

In [109]:

```
# Replacing NANs in waterfront by the mode of the column
waterfront_value = house_df2['waterfront'].mode().iloc[0]
house_df2['waterfront'].fillna(waterfront_value, inplace=True)
```

In [110]:

```
# checking for unique values entries for the waterfront column after replacing NANs with
house_df2['waterfront'].unique()
```

Out[110]:

```
array(['NO', 'YES'], dtype=object)
```

Waterfront only contains NO and YES as unique entries as expected after clean up

In [111]:

```
# checking unique entries for the condition column to understand extent of cleaning neede
house_df2['condition'].unique()
```

Out[111]:

```
array(['Average', 'Good', 'Very Good', 'Fair', 'Poor'], dtype=object)
```

In [112]:

```
house_df2['condition'].value_counts()
```

Out[112]:

```
Average      13900
Good          5643
Very Good     1687
Fair           162
Poor            28
Name: condition, dtype: int64
```

From the above waterfront and condition, it is clear that row enteries are strings. For this columns we will do a Dummy to change this column to numerical

In [113]:

```
# creating a new df called house_df2_dummy to get dummies
house_df2_dummy = house_df2.copy(deep=True)
house_df2_dummy
condition_dummy_df = house_df2_dummy[['condition', 'waterfront']]
```

In [114]:

```
# viewing head and tail of the new condition_dummy_df dataframe created
condition_dummy_df
```

Out[114]:

	condition	waterfront
15937	Average	NO
20963	Average	NO
7614	Good	NO
3257	Very Good	NO
16723	Average	NO
...	...	...
3553	Average	NO
8800	Good	NO
8404	Average	NO
6729	Good	NO
2494	Average	NO

21420 rows × 2 columns

In [115]:

```
#using one-encoding to create dummy column
ohe = OneHotEncoder()
data_enc1= ohe.fit_transform(condition_dummy_df)


#converting the finding into dataframe
data_enc1.todense()


#getting feature names
ohe.get_feature_names_out()
```

Out[115]:

```
array(['condition_Average', 'condition_Fair', 'condition_Good',
       'condition_Poor', 'condition_Very Good', 'waterfront_NO',
       'waterfront_YES'], dtype=object)
```

In [116]:

```
# geting feature names in a dataframe
data3 = pd.DataFrame(data_enc1.todense(), columns=ohe.get_feature_names_out())
data3.head()
```

Out[116]:

	condition_Average	condition_Fair	condition_Good	condition_Poor	condition_Very Good	waterfr
0	1.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	1.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	1.0	
4	1.0	0.0	0.0	0.0	0.0	

In [117]:

```
##checking for encoded categorical variables
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21420 entries, 0 to 21419
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   condition_Average    21420 non-null  float64
 1   condition_Fair       21420 non-null  float64
 2   condition_Good       21420 non-null  float64
 3   condition_Poor       21420 non-null  float64
 4   condition_Very Good  21420 non-null  float64
 5   waterfront_NO        21420 non-null  float64
 6   waterfront_YES       21420 non-null  float64
dtypes: float64(7)
memory usage: 1.1 MB
```

sqft_basement variable view and preparation

In [118]:

```
# cleaning the sqft_basement column by replacing the '?' that is making the column an obj
house_df2['sqft_basement'].unique()
house_df2.sqft_basement.replace('?', np.NaN, inplace=True)

# changing the sqft_basement column datatype to a floating point
house_df2.sqft_basement = house_df2.sqft_basement.astype(float)
```

In [119]:

```
# replacing the np.NaN int he sqft_basement column with the mean
house_df2.sqft_basement.replace(np.NaN, house_df2['sqft_basement'].mean(), inplace=True)
```

In [120]:

```
# General view of house_df2 head to after making changes to the sqft_basement
house_df2.head()
```

Out[120]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
15937	9900000190	2014-10-30	268950.0	3	1.00	1320	8100	1.0	
20963	9895000040	2014-07-03	399900.0	2	1.75	1410	1005	1.5	
7614	9842300540	2014-06-24	339000.0	3	1.00	1100	4128	1.0	
3257	9842300485	2015-03-11	380000.0	2	1.00	1040	7372	1.0	
16723	9842300095	2014-07-25	365000.0	5	2.00	1600	4168	1.5	

In [121]:

```
# doing general checking of head and tail of date column
house_df2['date']
```

Out[121]:

```
15937   2014-10-30
20963   2014-07-03
7614    2014-06-24
3257    2015-03-11
16723   2014-07-25
           ...
3553    2015-03-19
8800    2015-04-01
8404    2014-08-11
6729    2014-05-08
2494    2014-09-16
Name: date, Length: 21420, dtype: datetime64[ns]
```

Creating the year house was built

In order to get year the house was built, we need to extract the year from the date column

In [122]:

```
# creating a year column by extacting year house was build from the date column
house_df2['year'] = house_df2['date'].dt.year
```

In [123]:

```
# previewing the year column to see that we've extracted the year and created a new colum
house_df2['year']
```

Out[123]:

```
15937    2014
20963    2014
7614     2014
3257     2015
16723    2014
         ...
3553     2015
8800     2015
8404     2014
6729     2014
2494     2014
Name: year, Length: 21420, dtype: int64
```

Creating the age of the house column

In [124]:

```
# Creating the age of the house as a difference between sold year column created and year
house_df2['age'] = house_df2['year'] - house_df2['yr_built']
house_df2['age']
```

Out[124]:

```
15937    71
20963     3
7614     72
3257     76
16723    87
         ..
3553     64
8800     85
8404     62
6729     67
2494     23
Name: age, Length: 21420, dtype: int64
```

In [125]:

```
house_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21420 entries, 15937 to 2494
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21420 non-null  int64
 1   date           21420 non-null  datetime64[ns]
 2   price          21420 non-null  float64
 3   bedrooms       21420 non-null  int64
 4   bathrooms      21420 non-null  float64
 5   sqft_living    21420 non-null  int64
 6   sqft_lot       21420 non-null  int64
 7   floors         21420 non-null  float64
 8   waterfront     21420 non-null  object
 9   condition      21420 non-null  object
 10  grade          21420 non-null  int32
 11  sqft_above     21420 non-null  int64
 12  sqft_basement  21420 non-null  float64
 13  yr_built       21420 non-null  int64
 14  yr_renovated   17610 non-null  float64
 15  zipcode        21420 non-null  int64
 16  lat            21420 non-null  float64
 17  long           21420 non-null  float64
 18  sqft_living15  21420 non-null  int64
 19  sqft_lot15     21420 non-null  int64
 20  year           21420 non-null  int64
 21  age            21420 non-null  int64
dtypes: datetime64[ns](1), float64(7), int32(1), int64(11), object(2)
memory usage: 3.7+ MB
```

In [126]:

```
house_df2['yr_renovated'].isna().sum()
```

Out[126]:

```
3810
```

Year renovated colum view and preparation

In [127]:

```
# preview the yr_renovated colums
house_df2['yr_renovated'].head(10)
```

Out[127]:

```
15937    NaN
20963    0.0
7614     NaN
3257     0.0
16723    0.0
11642    0.0
13015    0.0
4817     0.0
4675     NaN
1714     NaN
Name: yr_renovated, dtype: float64
```

In [128]:

```
# cleaning yr_renovated column by replacing nulls with corresponding values in the yr_bui
house_df2.yr_renovated.fillna(house_df2.yr_built, inplace=True)
# replacing yr_renovated 0.0 with corresponding year in the yr_built
house_df2.loc[house_df2.yr_renovated == 0.0, 'yr_renovated'] = house_df2.yr_built
```

Creating age of house since renovation variable

In [129]:

```
# creating age_ronovated= year house was sold -year house was renovated
house_df2['age_renovated'] = house_df2['year'] - house_df2['yr_renovated']
house_df2['age_renovated']
```

Out[129]:

```
15937    71.0
20963     3.0
7614     72.0
3257     76.0
16723    87.0
         ...
3553      2.0
8800     85.0
8404     62.0
6729     67.0
2494     23.0
Name: age_renovated, Length: 21420, dtype: float64
```

Zipcode column view and preparation

In [130]:

```
# preview zipcode variable
house_df2['zipcode']
```

Out[130]:

```
15937    98166
20963    98027
7614     98126
3257     98126
16723    98126
         ...
3553     98144
8800     98168
8404     98166
6729     98166
2494     98002
Name: zipcode, Length: 21420, dtype: int64
```

###3.1 Merging categorical encoded and continous variable

In [131]:

```
data3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21420 entries, 0 to 21419
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   condition_Average    21420 non-null  float64
 1   condition_Fair       21420 non-null  float64
 2   condition_Good       21420 non-null  float64
 3   condition_Poor       21420 non-null  float64
 4   condition_Very Good  21420 non-null  float64
 5   waterfront_NO        21420 non-null  float64
 6   waterfront_YES       21420 non-null  float64
dtypes: float64(7)
memory usage: 1.1 MB
```

In [132]:

```
# merging the converted categorical columns (data3 df) to the main house_df2 datagframe
house_df2_final = pd.merge(house_df2, data3, left_index=True, right_index=True)
```

In [133]:

```
house_df2_final
```

Out[133]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
15937	9900000190	2014-10-30	268950.0	3	1.00	1320	8100	1.0	
20963	9895000040	2014-07-03	399900.0	2	1.75	1410	1005	1.5	
7614	9842300540	2014-06-24	339000.0	3	1.00	1100	4128	1.0	
3257	9842300485	2015-03-11	380000.0	2	1.00	1040	7372	1.0	
16723	9842300095	2014-07-25	365000.0	5	2.00	1600	4168	1.5	
...	...	...	...	...	...	...	...	...	
3553	3600057	2015-03-19	402500.0	4	2.00	1650	3504	1.0	
8800	2800031	2015-04-01	235000.0	3	1.00	1430	7599	1.5	
8404	1200021	2014-08-11	400000.0	3	1.00	1460	43000	1.0	
6729	1200019	2014-05-08	647500.0	4	1.75	2060	26036	1.0	
2494	1000102	2014-09-16	280000.0	6	3.00	2400	9373	2.0	

21244 rows × 30 columns

4.0 Exploratory Data Analysis (EDA)

In [134]:

```
# Dropping date and id columns
house_df2_final = house_df2_final.drop(['date', 'id'], axis=1)
```

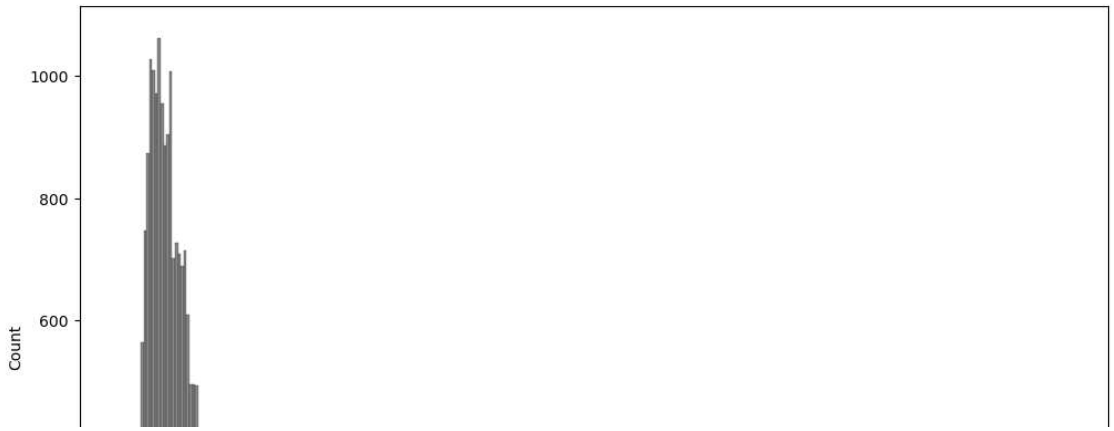
In [135]:

```
# dropping unwanted columns
house_df2_final = house_df2_final.drop([ 'condition', 'waterfront', 'year', 'yr_built'],
```

In [136]:

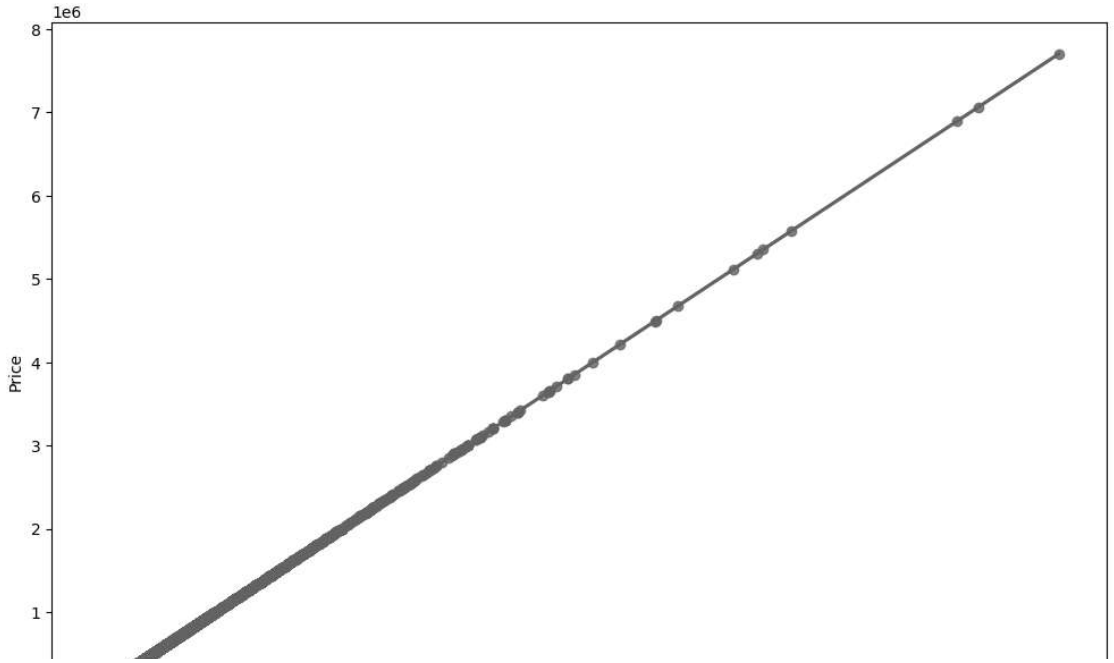
```
#View distribution plots for all columns
for col in house_df2_final.columns:
    plt.subplots(1, 1)
    sns.histplot(house_df2_final[col])
plt.savefig('Images/eda.png')
```

```
C:\Users\CGAKII\AppData\Local\Temp\ipykernel_8740\2052153547.py:3: Runti
meWarning: More than 20 figures have been opened. Figures created throug
h the pyplot interface (`matplotlib.pyplot.figure`) are retained until e
xplicitly closed and may consume too much memory. (To control this warni
ng, see the rcParam `figure.max_open_warning`).
  plt.subplots(1, 1)
```



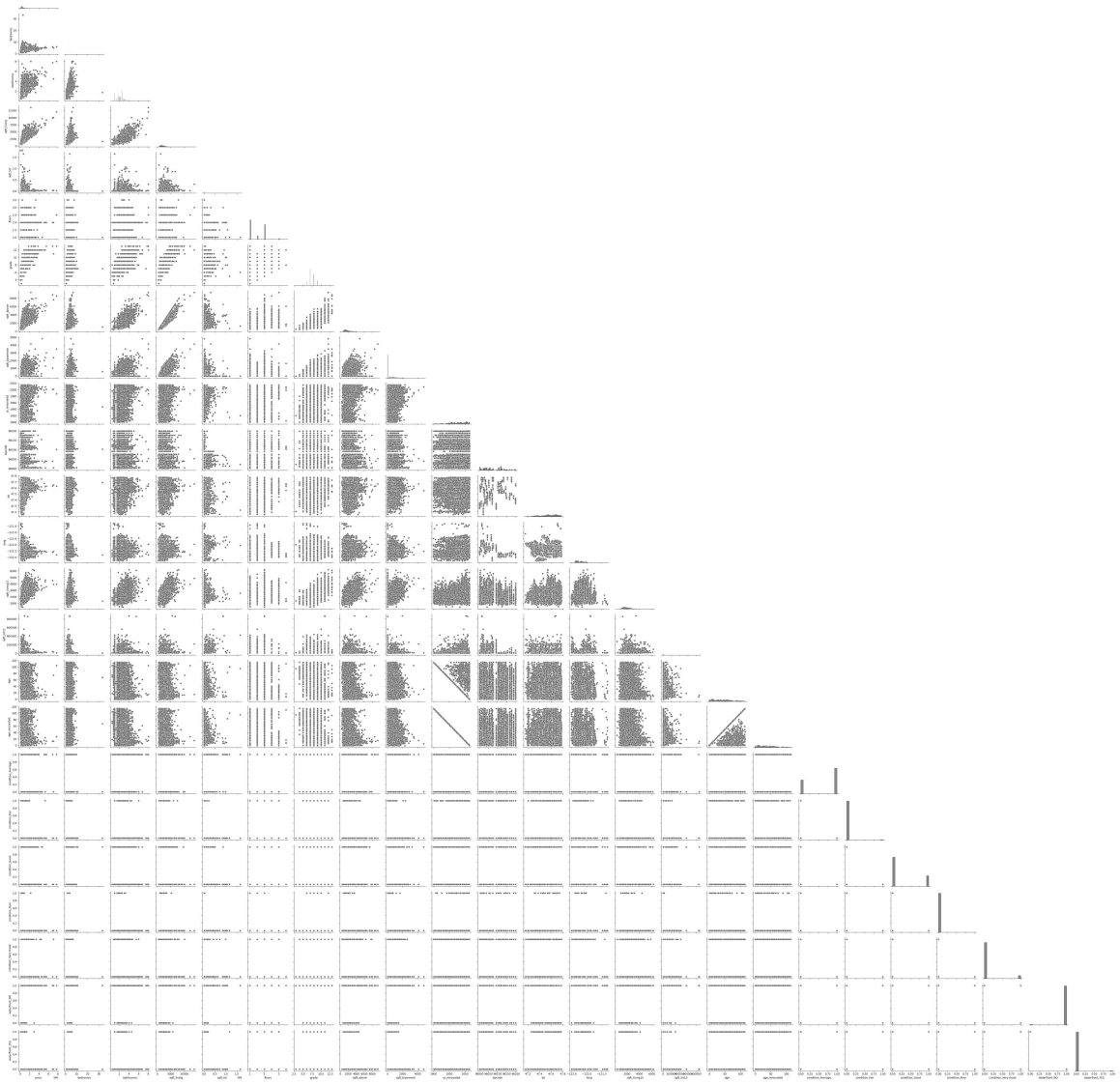
In [137]:

```
### checking for a simple vizual regression
for col in house_df2_final.columns:
    plt.subplots(1, 1)
    sns.regplot(x=col, y='price', data=house_df2_final)
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.show()
```



In [138]:

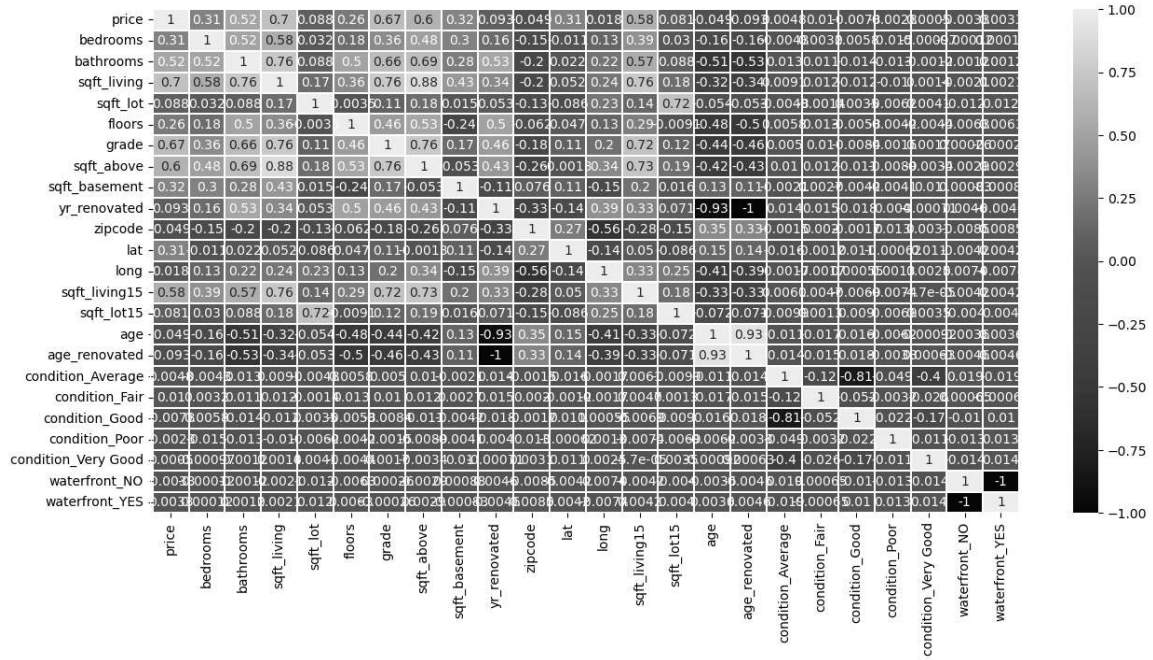
```
#quickly scan for linearity
sns.pairplot(house_df2_final, diag_kind = 'hist', corner = True)
plt.savefig('Images/sns.png')
```



Conclusion: Some variable are not correlated worth exploring more

In [139]:

```
# plotting the correlationmatrix
plt.figure(figsize =(15,7))
sns.heatmap(house_df2_final.corr() , annot =True , linewidth =0.2)
plt.savefig('Images/all_feature_corr.png')
```



Conclusion: Based on the correlation matrix there some variables are highly correlated which should be considered for linear regression. Additionally there is multicollinearity which needs to be explored

##5.0 Modelling

5.1 Baseline Model

In [140]:

```
#our target variable will be price content.
X = house_df2_final.drop("price", axis=1) # predictors  # always avoid data leakage
y = house_df2_final["price"]# target
```

In [141]:

```
# statmodel
# use sm.add_constant(), to add constant term/y-intercept
X_pred = sm.add_constant(X)# we add constant to differentiate predictor from other featur

#building the model
model =  sm.OLS(y,X_pred) .fit()

#getting the model summary
model.summary()
```

Out[141]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.660
Model:	OLS	**Adj. R-squared:**	0.660
Method:	Least Squares	**F-statistic:**	1964.
Date:	Fri, 02 Jun 2023	**Prob (F-statistic):**	0.00
Time:	22:59:57	**Log-Likelihood:**	-2.9085e+05
No. Observations:	21244	**AIC:**	5.818e+05
Df Residuals:	21222	**BIC:**	5.819e+05
Df Model:	21		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-3.371e+07	4.13e+06	-8.159	0.000	-4.18e+07	-2.56e+07
bedrooms	-4.55e+04	2024.056	-22.479	0.000	-4.95e+04	-4.15e+04
bathrooms	4.745e+04	3497.152	13.569	0.000	4.06e+04	5.43e+04
sqft_living	109.1784	22.653	4.820	0.000	64.777	153.580
sqft_lot	0.1422	0.051	2.792	0.005	0.042	0.242
floors	9096.6499	3867.668	2.352	0.019	1515.728	1.67e+04
grade	1.033e+05	2309.367	44.711	0.000	9.87e+04	1.08e+05
sqft_above	76.7000	22.666	3.384	0.001	32.272	121.128
sqft_basement	74.6117	22.657	3.293	0.001	30.202	119.021
yr_renovated	2.549e+04	3143.506	8.108	0.000	1.93e+04	3.17e+04
zipcode	-522.2306	34.950	-14.942	0.000	-590.736	-453.725
lat	5.477e+05	1.14e+04	48.031	0.000	5.25e+05	5.7e+05
long	-2.482e+05	1.41e+04	-17.635	0.000	-2.76e+05	-2.21e+05
sqft_living15	38.0384	3.657	10.401	0.000	30.870	45.207
sqft_lot15	-0.3115	0.078	-3.992	0.000	-0.464	-0.159
age	3854.6952	135.390	28.471	0.000	3589.321	4120.069
age_renovated	2.487e+04	3144.132	7.909	0.000	1.87e+04	3.1e+04
condition_Average	-6.755e+06	8.26e+05	-8.176	0.000	-8.37e+06	-5.14e+06
condition_Fair	-6.735e+06	8.26e+05	-8.150	0.000	-8.35e+06	-5.11e+06
condition_Good	-6.757e+06	8.26e+05	-8.179	0.000	-8.38e+06	-5.14e+06
condition_Poor	-6.703e+06	8.27e+05	-8.102	0.000	-8.32e+06	-5.08e+06
condition_Very Good	-6.758e+06	8.26e+05	-8.180	0.000	-8.38e+06	-5.14e+06
waterfront_NO	-1.685e+07	2.07e+06	-8.159	0.000	-2.09e+07	-1.28e+07
waterfront_YES	-1.685e+07	2.07e+06	-8.159	0.000	-2.09e+07	-1.28e+07

Omnibus:	19043.086	**Durbin-Watson:**	1.309
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1873733.100

Skew:	3.914	**Prob(JB):**	0.00
Kurtosis:	48.338	**Cond. No.**	6.88e+20

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.55e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Conclusion: Our first model has an adjusted r-squared of .660 All features with p_values that are significant, let's check our residuals.

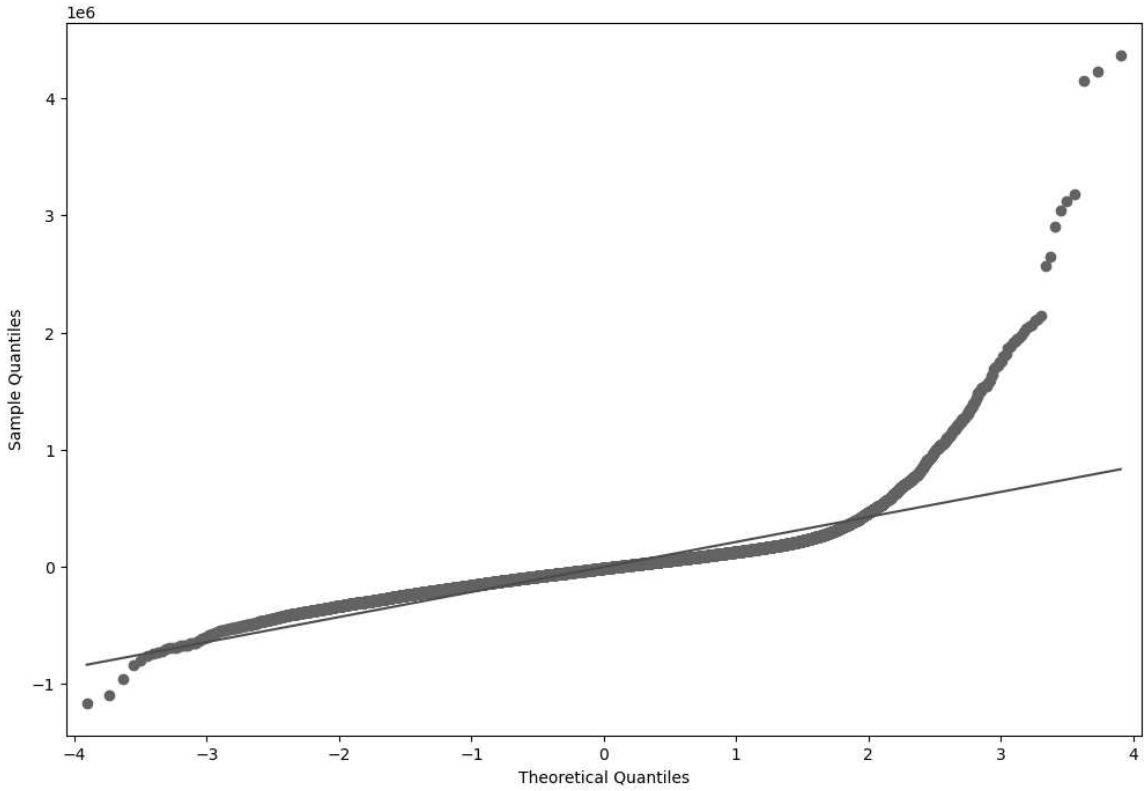
In [142]:

```
#Save model summary as image
plt.rc('figure', figsize=(12, 8))
plt.text(0.01, 0.05, str(model.summary()), {'fontsize': 10}, fontproperties = 'monospace'
plt.axis('off')
plt.tight_layout()
plt.savefig('Images/baselineModel.png')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.660
Model:                            OLS   Adj. R-squared:                  0.660
Method:                 Least Squares   F-statistic:                     1964.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        22:59:57   Log-Likelihood:            -2.9085e+05
No. Observations:               21244   AIC:                         5.818e+05
Df Residuals:                   21222   BIC:                         5.819e+05
Df Model:                          21
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const               -3.371e+07   4.13e+06     -8.159      0.000   -4.18e+07   -2.56e+07
bedrooms             -4.55e+04   2024.056    -22.479      0.000   -4.95e+04   -4.15e+04
bathrooms            4.745e+04   3497.152     13.569      0.000    4.06e+04    5.43e+04
sqft_living            109.1784     22.653      4.820      0.000      64.777     153.580
sqft_lot                 0.1422      0.051      2.792      0.005       0.042       0.242
floors                9096.6499   3867.668      2.352      0.019    1515.728    1.67e+04
grade                 1.033e+05   2309.367     44.711      0.000    9.87e+04    1.08e+05
sqft_above              76.7000     22.666      3.384      0.001      32.272     121.128
sqft_basement           74.6117     22.657      3.293      0.001      30.202     119.021
yr_renovated          2.549e+04   3143.506      8.108      0.000    1.93e+04    3.17e+04
zipcode                -522.2306     34.950    -14.942      0.000    -590.736    -453.725
lat                   5.477e+05   1.14e+04     48.031      0.000    5.25e+05     5.7e+05
long                 -2.482e+05   1.41e+04    -17.635      0.000   -2.76e+05   -2.21e+05
sqft_living15           38.0384      3.657     10.401      0.000      30.870      45.207
sqft_lot15              -0.3115      0.078     -3.992      0.000      -0.464      -0.159
age                   3854.6952    135.390     28.471      0.000    3589.321    4120.069
age_renovated         2.487e+04   3144.132      7.909      0.000    1.87e+04     3.1e+04
condition_Average    -6.755e+06   8.26e+05     -8.176      0.000   -8.37e+06   -5.14e+06
condition_Fair       -6.735e+06   8.26e+05     -8.150      0.000   -8.35e+06   -5.11e+06
condition_Good       -6.757e+06   8.26e+05     -8.179      0.000   -8.38e+06   -5.14e+06
condition_Poor       -6.703e+06   8.27e+05     -8.102      0.000   -8.32e+06   -5.08e+06
condition_Very Good  -6.758e+06   8.26e+05     -8.180      0.000   -8.38e+06   -5.14e+06
waterfront_NO        -1.685e+07   2.07e+06     -8.159      0.000   -2.09e+07   -1.28e+07
waterfront_YES       -1.685e+07   2.07e+06     -8.159      0.000   -2.09e+07   -1.28e+07
==============================================================================
Omnibus:                    19043.086   Durbin-Watson:                   1.309
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1873733.100
Skew:                           3.914   Prob(JB):                         0.00
Kurtosis:                      48.338   Cond. No.                     6.88e+20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.55e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [143]:

```
import statsmodels.graphics.gofplots as smg
# Obtain the residuals
residuals = model.resid

# Create the QQ plot
smg.qqplot(residuals, line='s')

# Show the plot
plt.show()
plt.savefig('Images/qq_baseline.png')
```



<Figure size 1200x800 with 0 Axes>

Conclusion.Residuals are not normal, which violates are assumption of normality we will use log transform

##5.2 Model 2 with log transformed y variables

In [144]:

```
#our target variable will be price content.
X = house_df2_final.drop("price", axis=1) # predictors  # always avoid data leakage
y = house_df2_final["price"]# target
y_log = np.log(y)
```

In [169]:

```
# statmodel
# use sm.add_constant(), to add constant term/y-intercept
X_pred = sm.add_constant(X)# we add constant to differentiate predictor from other featur

#building the model
model =  sm.OLS(y_log,X_pred) .fit()

#getting the model summary
model.summary()
```

Out[169]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.740
Model:	OLS	**Adj. R-squared:**	0.740
Method:	Least Squares	**F-statistic:**	4654.
Date:	Fri, 02 Jun 2023	**Prob (F-statistic):**	0.00
Time:	23:02:01	**Log-Likelihood:**	-2145.4
No. Observations:	21244	**AIC:**	4319.
Df Residuals:	21230	**BIC:**	4430.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-19.1124	2.512	-7.609	0.000	-24.036	-14.189
bedrooms	-0.0198	0.003	-7.819	0.000	-0.025	-0.015
bathrooms	0.0725	0.004	16.586	0.000	0.064	0.081
sqft_living	0.0002	5.77e-06	35.087	0.000	0.000	0.000
sqft_lot	4.287e-07	4.63e-08	9.256	0.000	3.38e-07	5.19e-07
floors	0.0657	0.005	13.593	0.000	0.056	0.075
grade	0.1609	0.003	55.785	0.000	0.155	0.167
sqft_above	-5.477e-05	5.75e-06	-9.523	0.000	-6.6e-05	-4.35e-05
zipcode	-0.0006	4.37e-05	-12.865	0.000	-0.001	-0.000
lat	1.3575	0.014	95.255	0.000	1.330	1.385
long	-0.2469	0.017	-14.131	0.000	-0.281	-0.213
sqft_living15	0.0001	4.57e-06	24.031	0.000	0.000	0.000
age_renovated	0.0037	8.93e-05	41.553	0.000	0.004	0.004
waterfront_NO	-9.5608	1.256	-7.614	0.000	-12.022	-7.099
waterfront_YES	-9.5516	1.256	-7.605	0.000	-12.013	-7.090

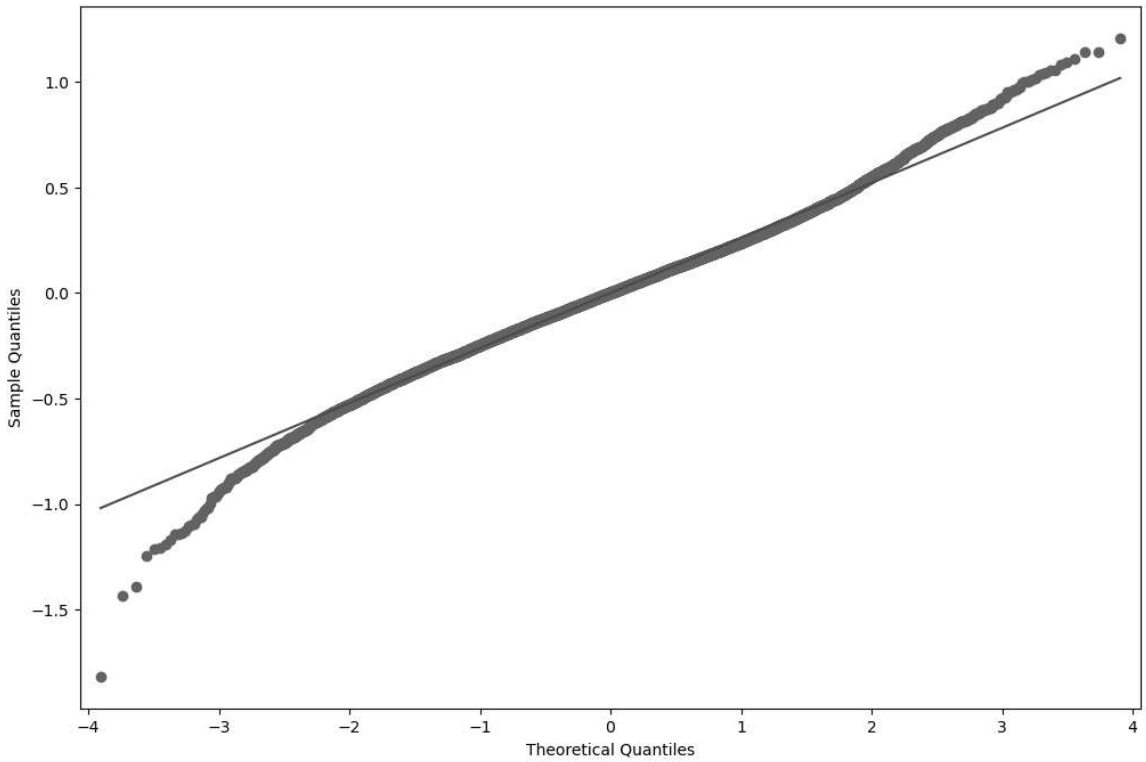
Omnibus:	426.976	**Durbin-Watson:**	1.143
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	873.336
Skew:	0.089	**Prob(JB):**	2.28e-190
Kurtosis:	3.977	**Cond. No.**	9.00e+20

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.6e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Conclusion: After log transformation of the dependent variable Our second model has an adjusted r-squared of .753 sqft_lot15 has p_values that is not significant, but before we drop these feature from the data, let's check our residuals.

In [146]:

```
#Plot QQ plot for the second model

# Obtain the residuals
residuals = model.resid

# Create the QQ plot
smg.qqplot(residuals, line='s')

# Show the plot
plt.show()
plt.savefig('Images/QQ2.png')
```



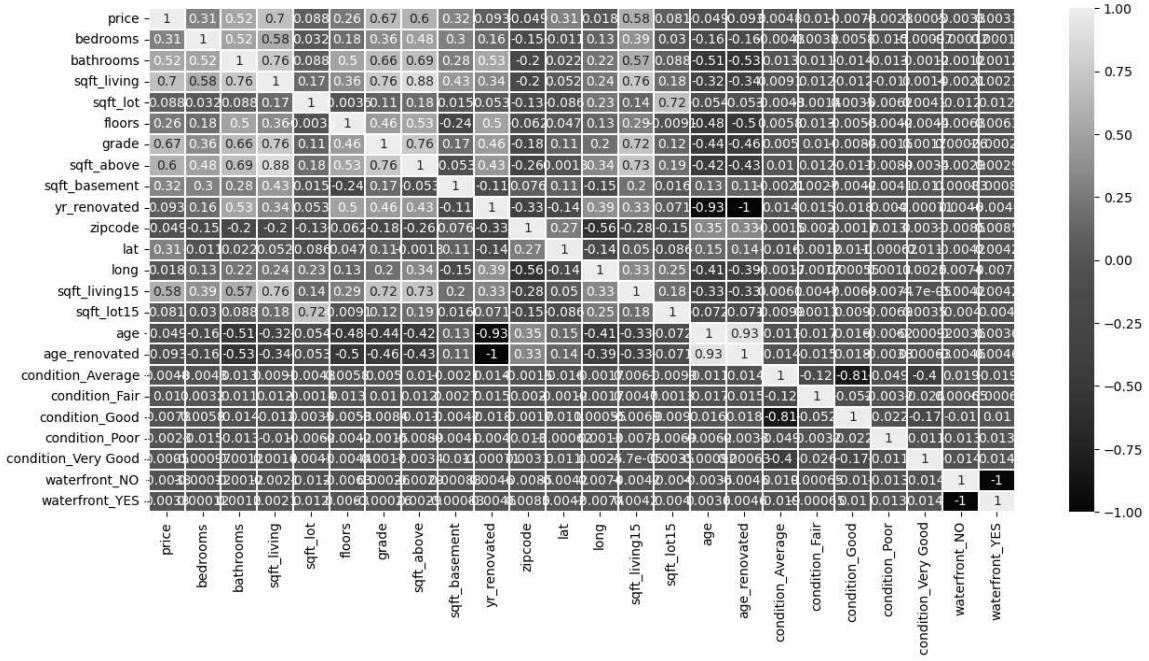
<Figure size 1200x800 with 0 Axes>

Conclusion: After log transformation of the dependent variables our residuals are much closer to a normal distribution.

###5.3 Model 3: Dealing with Multicollinearity

In [147]:

```
# plotting the correlationmatrix
plt.figure(figsize =(15,7))
sns.heatmap(house_df2_final.corr() , annot =True , linewidth =0.2)
plt.savefig('Images/CORR2.png')
```



In [148]:

```
#### checking variable with correlation of 0.7  and obove
abs(X.corr()) >= .7
```

Out[148]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above	sqf
bedrooms	True	False	False	False	False	False	False	
bathrooms	False	True	True	False	False	False	False	
sqft_living	False	True	True	False	False	True	True	
sqft_lot	False	False	False	True	False	False	False	
floors	False	False	False	False	True	False	False	
grade	False	False	True	False	False	True	True	
sqft_above	False	False	True	False	False	True	True	
sqft_basement	False	False	False	False	False	False	False	
yr_renovated	False	False	False	False	False	False	False	
zipcode	False	False	False	False	False	False	False	
lat	False	False	False	False	False	False	False	
long	False	False	False	False	False	False	False	
sqft_living15	False	False	True	False	False	True	True	
sqft_lot15	False	False	False	True	False	False	False	
age	False	False	False	False	False	False	False	
age_renovated	False	False	False	False	False	False	False	
condition_Average	False	False	False	False	False	False	False	
condition_Fair	False	False	False	False	False	False	False	
condition_Good	False	False	False	False	False	False	False	
condition_Poor	False	False	False	False	False	False	False	
condition_Very Good	False	False	False	False	False	False	False	
waterfront_NO	False	False	False	False	False	False	False	
waterfront_YES	False	False	False	False	False	False	False	

23 rows × 23 columns

Conclusion: There are several features with high correlation worth exploring more

In [149]:

```
corr_df = X.corr().abs().stack().reset_index().sort_values(0, ascending=False)

# zip the variable name columns (Which were only named level_0 and level_1 by default) in
corr_df['pairs'] = list(zip(corr_df.level_0, corr_df.level_1))

# set index to pairs
corr_df.set_index(['pairs'], inplace = True)

#drop level columns
corr_df.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
corr_df.columns = ['cc']

# drop duplicates
corr_df.drop_duplicates(inplace=True)
```

In [150]:

```
#Filter and correlations above .7 and below 1
corr_df[(corr_df.cc>.70) & (corr_df.cc <1)]
```

Out[150]:

	cc
pairs	
(age_renovated, yr_renovated)	0.999869
(age_renovated, age)	0.925105
(yr_renovated, age)	0.925082
(sqft_above, sqft_living)	0.875406
(condition_Good, condition_Average)	0.813025
(sqft_living, grade)	0.762370
(sqft_living15, sqft_living)	0.756164
(sqft_above, grade)	0.755953
(bathrooms, sqft_living)	0.755948
(sqft_above, sqft_living15)	0.731731
(sqft_lot15, sqft_lot)	0.717135
(sqft_living15, grade)	0.715253

Conclusion: There are several features that seem to have multicollinearity. Rather than just dropping some of these features, let's first look at the variance inflation factor to understand the severity of the multicollinearity.

In [151]:

```
#Create dictionary of features and their variance inflation factor
from statsmodels.stats.outliers_influence import variance_inflation_factor
X = sm.add_constant(X)
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif_dict = dict(zip(X.columns, vif))
vif_dict
```

```
C:\Users\CGAKII\anaconda3\lib\site-packages\statsmodels\regression\linear_
model.py:1736: RuntimeWarning: divide by zero encountered in double_scalar
s
  return 1 - self.ssr/self.centered_tss
C:\Users\CGAKII\anaconda3\lib\site-packages\statsmodels\stats\outliers_inf
luence.py:195: RuntimeWarning: divide by zero encountered in double_scalar
s
  vif = 1. / (1. - r_squared_i)
```

Out[151]:

```
{'const': 0.0,
 'bedrooms': 1.6307403800532274,
 'bathrooms': 3.3464498430982763,
 'sqft_living': 200.22111344390765,
 'sqft_lot': 2.0913700461728606,
 'floors': 2.0062541247447085,
 'grade': 3.39366327069723,
 'sqft_above': 162.81164790650422,
 'sqft_basement': 45.91100868249492,
 'yr_renovated': 3831.2944979604404,
 'zipcode': 1.6267951592089793,
 'lat': 1.1641384915092674,
 'long': 1.823385301487551,
 'sqft_living15': 2.905613803711736,
 'sqft_lot15': 2.1258454130348237,
 'age': 7.315762391477903,
 'age_renovated': 3834.145897152863,
 'condition_Average': inf,
 'condition_Fair': inf,
 'condition_Good': inf,
 'condition_Poor': inf,
 'condition_Very Good': inf,
 'waterfront_NO': inf,
 'waterfront_YES': inf}
```

In [152]:

```
#Create a list of columns to drop with a vif cutoff of
new_dict = {}
for (key, value) in vif_dict.items():
    if value >= 6:
        new_dict[key] = value
columns_to_drop = list(new_dict.keys())
columns_to_drop = columns_to_drop[1:]
columns_to_drop
```

Out[152]:

```
['sqft_above',
 'sqft_basement',
 'yr_renovated',
 'age',
 'age_renovated',
 'condition_Average',
 'condition_Fair',
 'condition_Good',
 'condition_Poor',
 'condition_Very Good',
 'waterfront_NO',
 'waterfront_YES']
```

conclusion all have VIF OF 6 and above 'sqft_above', 'sqft_basement', 'yr_renovated', 'age', 'age_renovated', 'condition_Average', 'condition_Fair', 'condition_Good', 'condition_Poor', 'condition_Very Good', 'waterfront_NO', 'waterfront_YES']

In [153]:

```
## dropping 'sqft_basement', 'yr_renovated', 'age', 'condition_Poor','condition_Fair' to
house_df3_final = house_df2_final.drop(['sqft_basement', 'yr_renovated', 'age', 'conditic
```

In [154]:

```
#our target variable will be price content.
X = house_df3_final.drop("price", axis=1) # predictors  # always avoid data leakage
y = house_df3_final["price"]# target
y_log = np.log(y)
```

In [170]:

```
# statmodel
# use sm.add_constant(), to add constant term/y-intercept
X_pred = sm.add_constant(X)# we add constant to differentiate predictor from other featur

#building the model
model =  sm.OLS(y_log,X_pred) .fit()

#getting the model summary
model.summary()
```

Out[170]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.740
Model:	OLS	**Adj. R-squared:**	0.740
Method:	Least Squares	**F-statistic:**	4654.
Date:	Fri, 02 Jun 2023	**Prob (F-statistic):**	0.00
Time:	23:02:16	**Log-Likelihood:**	-2145.4
No. Observations:	21244	**AIC:**	4319.
Df Residuals:	21230	**BIC:**	4430.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-19.1124	2.512	-7.609	0.000	-24.036	-14.189
bedrooms	-0.0198	0.003	-7.819	0.000	-0.025	-0.015
bathrooms	0.0725	0.004	16.586	0.000	0.064	0.081
sqft_living	0.0002	5.77e-06	35.087	0.000	0.000	0.000
sqft_lot	4.287e-07	4.63e-08	9.256	0.000	3.38e-07	5.19e-07
floors	0.0657	0.005	13.593	0.000	0.056	0.075
grade	0.1609	0.003	55.785	0.000	0.155	0.167
sqft_above	-5.477e-05	5.75e-06	-9.523	0.000	-6.6e-05	-4.35e-05
zipcode	-0.0006	4.37e-05	-12.865	0.000	-0.001	-0.000
lat	1.3575	0.014	95.255	0.000	1.330	1.385
long	-0.2469	0.017	-14.131	0.000	-0.281	-0.213
sqft_living15	0.0001	4.57e-06	24.031	0.000	0.000	0.000
age_renovated	0.0037	8.93e-05	41.553	0.000	0.004	0.004
waterfront_NO	-9.5608	1.256	-7.614	0.000	-12.022	-7.099
waterfront_YES	-9.5516	1.256	-7.605	0.000	-12.013	-7.090

Omnibus:	426.976	**Durbin-Watson:**	1.143
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	873.336
Skew:	0.089	**Prob(JB):**	2.28e-190
Kurtosis:	3.977	**Cond. No.**	9.00e+20

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.6e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Conclusion: Our adjusted R squared stays the same at .740, but sqft_lot15,condition_Average,condition_Good,condition_Very Good is still showing as insignificant. We will remove this feature and rebuild our model.

5.4 Model 4: Dropping Insignificant Features

In [156]:

```
## dropping 'sqft_basement', 'yr_renovated', 'age', 'condition_Poor','condition_Fair' to
house_df4_final = house_df3_final.drop(['sqft_lot15', 'condition_Average', 'condition_Goo
```

In [157]:

```
#our target variable will be price content.
X = house_df4_final.drop("price", axis=1) # predictors  # always avoid data leakage
y = house_df4_final["price"]# target
y_log = np.log(y)
```

In [158]:

```
# statmodel
# use sm.add_constant(), to add constant term/y-intercept
X_pred = sm.add_constant(X)# we add constant to differentiate predictor from other featur

#building the model
model =  sm.OLS(y_log,X_pred) .fit()

#getting the model summary
model.summary()
```

Out[158]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.740
Model:	OLS	**Adj. R-squared:**	0.740
Method:	Least Squares	**F-statistic:**	4654.
Date:	Fri, 02 Jun 2023	**Prob (F-statistic):**	0.00
Time:	23:00:03	**Log-Likelihood:**	-2145.4
No. Observations:	21244	**AIC:**	4319.
Df Residuals:	21230	**BIC:**	4430.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-19.1124	2.512	-7.609	0.000	-24.036	-14.189
bedrooms	-0.0198	0.003	-7.819	0.000	-0.025	-0.015
bathrooms	0.0725	0.004	16.586	0.000	0.064	0.081
sqft_living	0.0002	5.77e-06	35.087	0.000	0.000	0.000
sqft_lot	4.287e-07	4.63e-08	9.256	0.000	3.38e-07	5.19e-07
floors	0.0657	0.005	13.593	0.000	0.056	0.075
grade	0.1609	0.003	55.785	0.000	0.155	0.167
sqft_above	-5.477e-05	5.75e-06	-9.523	0.000	-6.6e-05	-4.35e-05
zipcode	-0.0006	4.37e-05	-12.865	0.000	-0.001	-0.000
lat	1.3575	0.014	95.255	0.000	1.330	1.385
long	-0.2469	0.017	-14.131	0.000	-0.281	-0.213
sqft_living15	0.0001	4.57e-06	24.031	0.000	0.000	0.000
age_renovated	0.0037	8.93e-05	41.553	0.000	0.004	0.004
waterfront_NO	-9.5608	1.256	-7.614	0.000	-12.022	-7.099
waterfront_YES	-9.5516	1.256	-7.605	0.000	-12.013	-7.090

Omnibus:	426.976	**Durbin-Watson:**	1.143
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	873.336
Skew:	0.089	**Prob(JB):**	2.28e-190
Kurtosis:	3.977	**Cond. No.**	9.00e+20

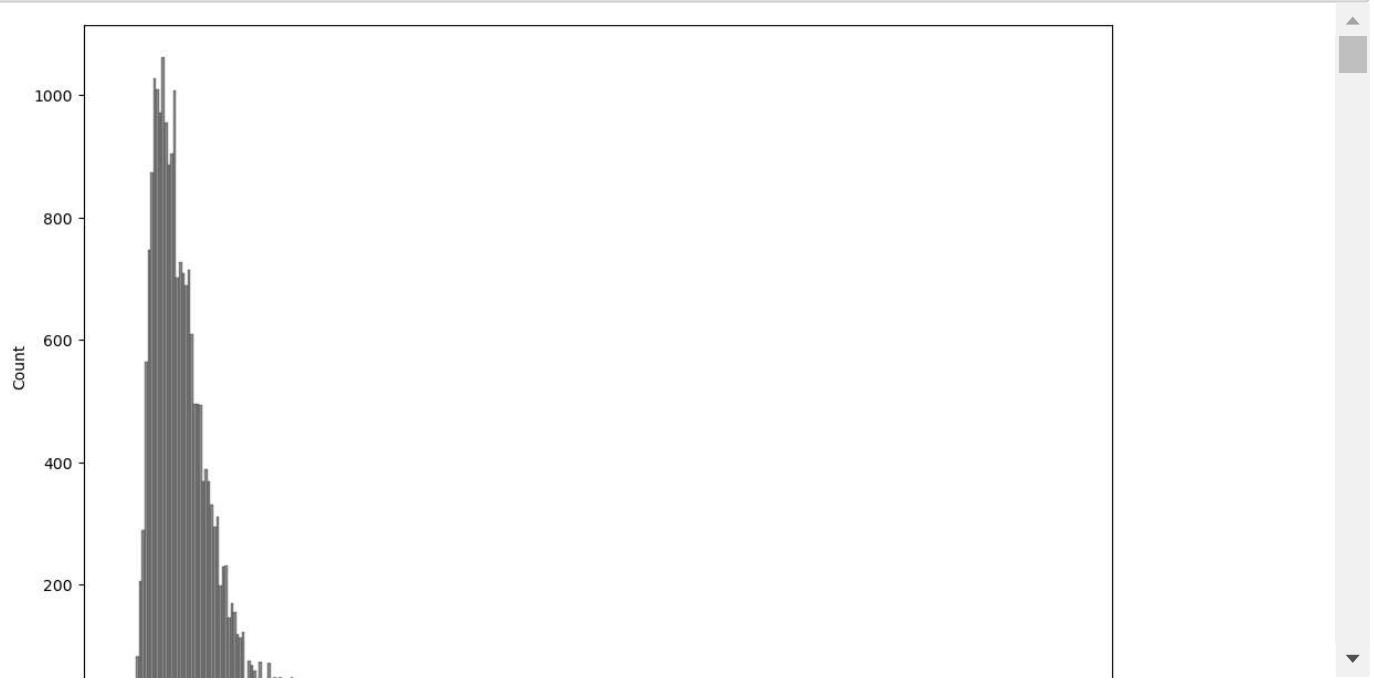
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.6e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Conclusion: Our adjusted R squared still stays the same at .740 and all features are significant. Next, we will further refine our data by removing additional, potential outliers

5.5 Model 5: Standardizing Features

In [159]:

```
#View distribution plots for all columns
for col in house_df4_final.columns:
    plt.subplots(1, 1)
    sns.histplot(house_df4_final[col])
```



Conclusion: Several features have heavy tails, so we will reduce these features by removing outliers.Bedrooms and sqft_lot have extreme outliers. Sqft_living also has outliers, so we will remove them from each of these columns.

In [160]:

```
house_df5_final = house_df4_final.copy()  # Create a copy of house_df4_final

house_df5_final.loc[house_df5_final['bedrooms'] >= 11]   # Set bedroom values to NaN if >
house_df5_final.loc[house_df5_final['sqft_lot'] > 15000]  # Set sqft_lot values to NaN if
house_df5_final.loc[house_df5_final['sqft_living'] > 5000 ] #  # Set sqft_living values t
```

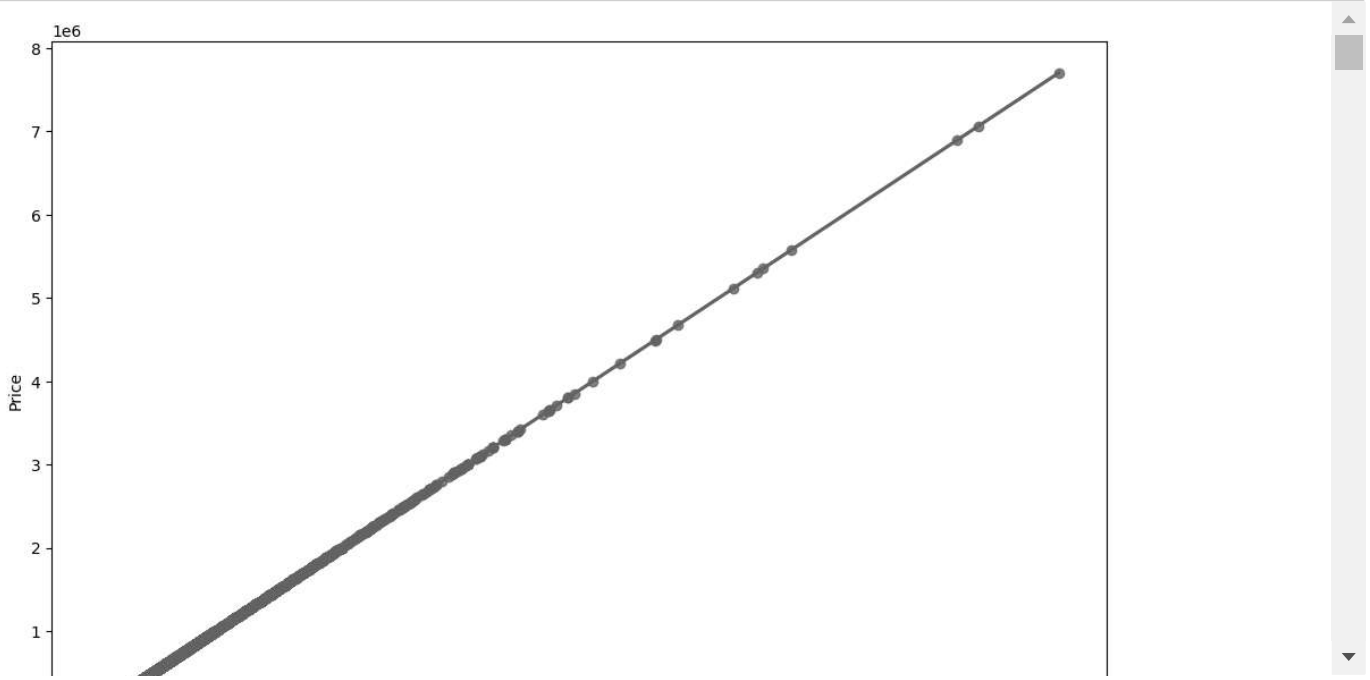
Out[160]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above	zipc
4807	2480000.0	5	3.75	6810	7500	2.5	13	6110	98
3910	7060000.0	5	4.50	10040	37325	2.0	11	7680	98
20444	3350000.0	5	3.75	5350	15360	1.0	11	3040	98
2897	3000000.0	5	3.25	5370	14091	2.0	10	3850	98
1620	610000.0	4	3.25	5450	37058	1.5	9	5450	98
...	...	...	...	...	...	...	...	...	
9705	937500.0	4	4.00	5545	871200	2.0	11	3605	98
6978	1210000.0	3	3.75	5400	24740	2.0	11	5400	98
8436	2140000.0	4	3.75	5150	453895	2.0	11	4360	98
10272	1030000.0	5	3.50	5050	16500	2.0	11	5050	98
1394	1060000.0	4	4.00	5320	20041	2.0	11	5320	98

206 rows × 15 columns

In [161]:

```
### conduct simple regression
for col in house_df5_final.columns:
    plt.subplots(1, 1)
    sns.regplot(x=col, y='price', data=house_df5_final)
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.show()
```



In [162]:

```
#our target variable will be price content.
X = house_df5_final.drop("price", axis=1) # predictors  # always avoid data leakage
y = house_df5_final["price"]# target
y_log = np.log(y)
```

In [163]:

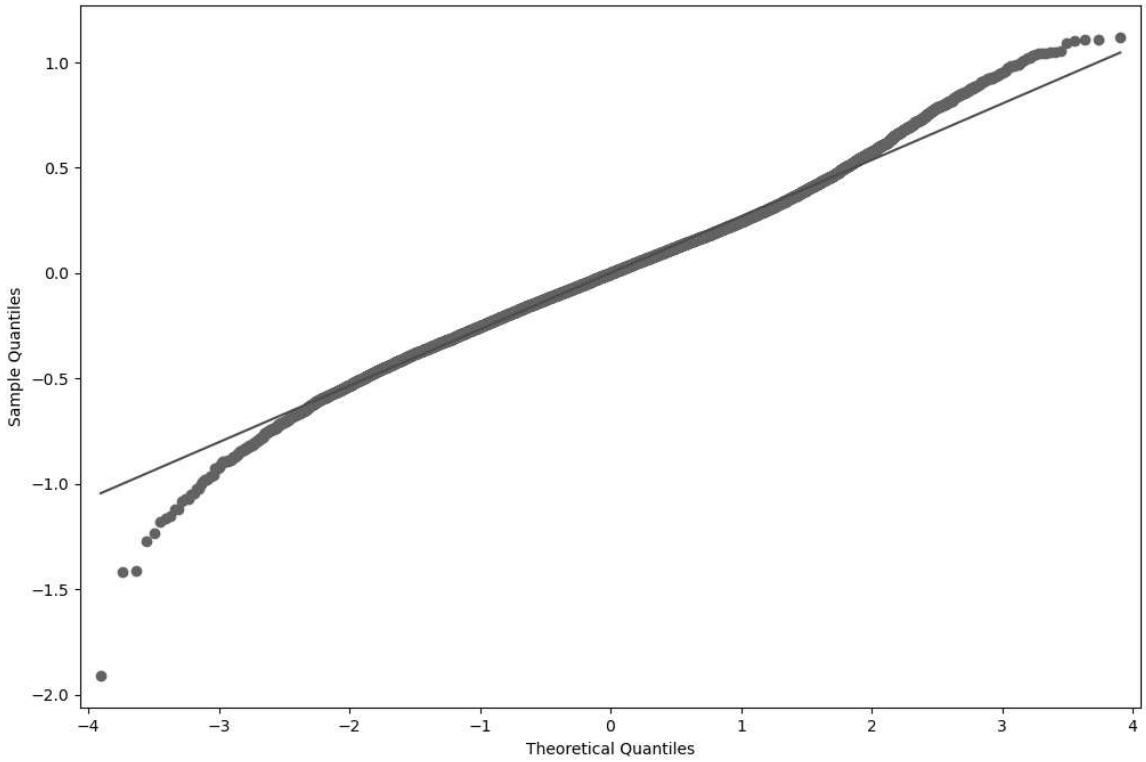
```
# statmodel
# use sm.add_constant(), to add constant term/y-intercept
X_pred = sm.add_constant(X)# we add constant to differentiate predictor from other featur

#building the model
model =  sm.OLS(y_log,X_pred) .fit()

#getting the model summary
model.summary()
plt.savefig('Images/model5.png')
```

```
<Figure size 1200x800 with 0 Axes>
```

Conclusion: Model 5 has r2 of 0.740 after stadardizing the feaures

In [164]:

```
#Plot QQ plot for the second model

# Obtain the residuals
residuals = model.resid

# Create the QQ plot
smg.qqplot(residuals, line='s')

# Show the plot
plt.show()
```



Conclusion: The QQ plot shows normal distribution.

In [165]:

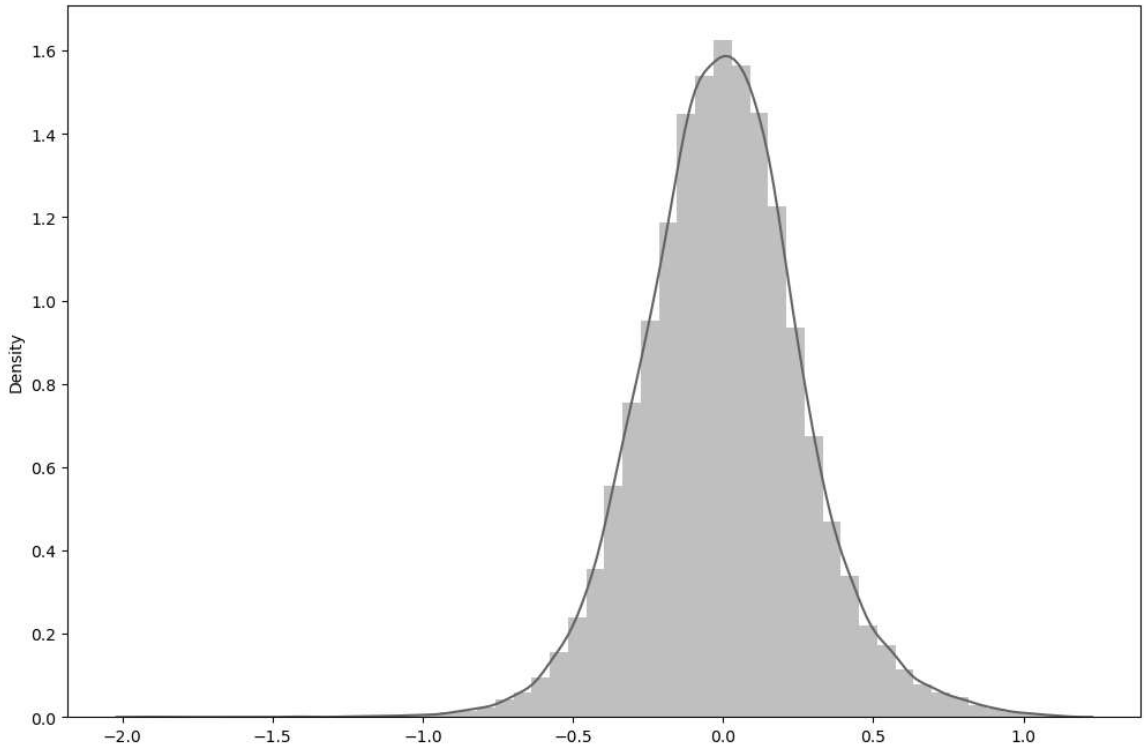
```
import os
os.makedirs('Images', exist_ok=True)
#Save model summary as image
plt.rc('figure', figsize=(12, 8))
plt.text(0.01, 0.05, str(model.summary()), {'fontsize': 10}, fontproperties = 'monospace'
plt.axis('off')
plt.tight_layout()
plt.savefig('Images/FinalModel.png')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.740
Model:                            OLS   Adj. R-squared:                  0.740
Method:                 Least Squares   F-statistic:                     4654.
Date:                Fri, 02 Jun 2023   Prob (F-statistic):               0.00
Time:                        23:00:29   Log-Likelihood:                -2145.4
No. Observations:               21244   AIC:                             4319.
Df Residuals:                   21230   BIC:                             4430.
Df Model:                          13
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          -19.1124      2.512     -7.609      0.000     -24.036     -14.189
bedrooms        -0.0198      0.003     -7.819      0.000      -0.025      -0.015
bathrooms        0.0725      0.004     16.586      0.000       0.064       0.081
sqft_living      0.0002   5.77e-06     35.087      0.000       0.000       0.000
sqft_lot      4.287e-07   4.63e-08      9.256      0.000    3.38e-07    5.19e-07
floors           0.0657      0.005     13.593      0.000       0.056       0.075
grade            0.1609      0.003     55.785      0.000       0.155       0.167
sqft_above    -5.477e-05   5.75e-06     -9.523      0.000    -6.6e-05   -4.35e-05
zipcode          -0.0006   4.37e-05    -12.865      0.000      -0.001      -0.000
lat               1.3575      0.014     95.255      0.000       1.330       1.385
long             -0.2469      0.017    -14.131      0.000      -0.281      -0.213
sqft_living15     0.0001   4.57e-06     24.031      0.000       0.000       0.000
age_renovated     0.0037   8.93e-05     41.553      0.000       0.004       0.004
waterfront_NO    -9.5608      1.256     -7.614      0.000     -12.022      -7.099
waterfront_YES   -9.5516      1.256     -7.605      0.000     -12.013      -7.090
==============================================================================
Omnibus:                      426.976   Durbin-Watson:                   1.143
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              873.336
Skew:                           0.089   Prob(JB):                    2.28e-190
Kurtosis:                       3.977   Cond. No.                     9.00e+20
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.6e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [166]:

```
###ploting residual to check normal distribution
sns.distplot(model.resid)
plt.savefig('Images/Finalresidual.png')
```

```
C:\Users\CGAKII\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```



In [167]:

```
import statsmodels.stats.api as sms

name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(model.resid, model.model.exog)
list(zip(name, test))
```

Out[167]:

```
[('F statistic', 1.0561782485627713), ('p-value', 0.002442965463354118)]
```

Conclusion: Since the p-value (0.0024) is less than the significance level (e.g., 0.05), we can reject the null hypothesis. This suggests that there is significant evidence of heteroscedasticity in the data. It implies that the variance of the errors is not constant across the range of the predictors.

In summary,F statistic and p-value, there is evidence of heteroscedasticity in the data, indicating that the assumption of constant error variance may not hold.

In [168]:

```
print("Final Regression Formula\n")
print(model.params[0], '+')
print(' + '.join([f'{model.params[i]} * {X.columns[i-1]}' for i in range(1, len(model.par
```

```
Final Regression Formula

-19.11239653245633 +
-0.019796846106053673 * bedrooms + 0.07253655844197224 * bathrooms + 0.000
2023838897631484 * sqft_living + 4.286622915049802e-07 * sqft_lot + 0.0657
0339219768592 * floors + 0.16091514868200807 * grade + -5.477357160189785e
-05 * sqft_above + -0.0005621859739730626 * zipcode + 1.357540159536652 *
lat + -0.24691616270697103 * long + 0.00010989530962926673 * sqft_living15
+ 0.0037120137528561916 * age_renovated + -9.560829464063175 * waterfront_
NO + -9.55156706839307 * waterfront_YES
```

##6.0 Conclusion Multiple regression analysis was used to test if certain variables significantly predicted the sale price of homes in King County.

The results of the regression indicated that 11 features can be 74% be explaned by the price.All the features are statistically significant with a p-vaue of <0.05.

Analyzing the models the following conclusion can be made.

1. An increase with one bedroom decreases the house sale by $ 0.002.
2. An increase with one bathroom increases the house price by $ 0.0725
3. An increase in Square footage of the home by one square foot increases the price pof the house by $ 0.0002
4. An increase in Square footage of the by one square feet decreases the house price by $ 4.287e-07
5. An increase in floors by one increases price by $0.0657
6. An increase in grade rating by one increases the price by $ 0.1609
7. An increase in one square foot from basement decrease price by $ -5.477e-05
8. An increase in Square footage of interior housing living space for the nearest 15 neighbors by one foot increase prices by $ 0.0001
9. There is no significant increase/decrease in the house price with the condition of the house
10. Presence of waterfront deccreases the house price by $ -9.5516

##7.0 Recommendation

1.Bedrooms: Consider the impact of the number of bedrooms on the **2. house price.** If possible, evaluate the demand and preferences of potential buyers in the target market. If the majority of buyers are looking for houses with fewer bedrooms, it may be beneficial to focus on properties with a lower bedroom count.

3 Bathrooms: Increasing the number of bathrooms tends to have a positive impact on the house price. If feasible, consider adding or upgrading bathrooms in properties to attract buyers who value this feature.

4.Square Footage: Pay attention to the square footage of the living space and lot size. Increasing the living space generally increases the house price, while larger lot sizes may have a slight negative impact. Consider the preferences of the target market and the local real estate trends to find the right balance.

5.Floors: Houses with multiple floors tend to have higher prices. If feasible, explore opportunities to add or emphasize multiple floors in properties to increase their perceived value.

6. Grade Rating: The grade rating of a house positively affects its price. Consider improving the grade rating through renovations, upgrades, or focusing on properties with higher grade ratings.

7. Basement: The square footage of the basement has a small negative impact on the house price. Evaluate the demand for basements in the target market and consider their condition and potential for improvement.

8.Neighborhood: Consider the impact of the square footage of interior living space for the nearest 15 neighbors on the house price. Properties with larger nearby living spaces may attract higher prices. Evaluate the desirability and market demand for properties in different neighborhoods.

9. Condition and Waterfront: The condition of the house does not have a significant impact on the price, while the presence of a waterfront has a negative impact. Consider these factors in pricing strategies and marketing efforts.

##8.0 Next steps

1. Explore if creating new features or transforming existing ones can improve the model.
2. Identify the most important features to include in the model.
3. Evaluate the model's performance using cross-validation techniques.
4. Compare the model with other algorithms to see if there are better options.