Final Project Submission

Please fill out:

- Student name: Jane Njuguna
- Student pace:Full time
- Scheduled project review date/time:30/09/2022
- Instructor name: Lucille Kaleha

Business understanding

Business overview

- Cherie real estate agency is a housing stakeholder that gives advice to homeowners so that they can buy and/or sell home.

- The agency want to help homeowners to be able to predict the current and future prices of their houses depending on different fearures.

Objectives

- To build a linear regression model that will help the stakeholder to predict prices based on features like Square footage of living space in the home, Square footage of the lot, Number of floors (levels) in house, codition of the house, presence of a waterfront,number of bedrooms and bathrooms,grade of the house and the year built.
- Also the model will help us know which featurs increases or decreases the price of the houses.

Importing dependencies

In [537]:

```
# importing libraries
#For visualizations
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
# For numerical operations
import numpy as np
#For data loading, data analysis and data cleaning
import pandas as pd
# For creating linear regression models
from sklearn.linear_model import LinearRegression
# For splitting data to testing and training data
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squ
import scipy.stats as stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
```

Data loading

The data used is from kc_house_data.csv

In [538]:

```
# loading the data
houses_df = pd.read_csv("data/kc_house_data.csv")
```

Data understanding

- Checking the shape of the dataframe
- Generate an overview of the dataframe

In [539]:

```
# to check the shape of the dataframe
houses_df.shape
```

Out[539]:

```
(21597, 21)
```

In [540]:

```
# to check  information about the DataFrame
houses_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  float6
4
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float6
4
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float6
4
 8   waterfront     19221 non-null  object
 9   view           21534 non-null  object
 10  condition      21597 non-null  object
 11  grade          21597 non-null  object
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  object
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float6
4
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float6
4
 18  long           21597 non-null  float6
4
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

In [541]:

```
# checking the type of data types in the dataset
houses_df.dtypes.value_counts()
```

Out[541]:

```
int64      9
object     6
float64    6
dtype: int64
```

In [542]:

```
#to check the columns in the dataset
houses_df.columns
```

Out[542]:

```
Index(['id', 'date', 'price', 'bedrooms',
'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront',
'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_
built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sq
ft_lot15'],
      dtype='object')
```

In [543]:

```
# the overview of the dataframe
houses_df.head()
```

Out[543]:

	id	date	price	bedrooms	batl
0	7129300520	10/13/2014	221900.0	3	
1	6414100192	12/9/2014	538000.0	3	
2	5631500400	2/25/2015	180000.0	2	
3	2487200875	12/9/2014	604000.0	4	
4	1954400510	2/18/2015	510000.0	3	

5 rows × 21 columns

In [544]:

```
# the overview of the dataframe
houses_df.tail()
```

Out[544]:

	id	date	price	bedrooms
21592	263000018	5/21/2014	360000.0	3
21593	6600060120	2/23/2015	400000.0	4
21594	1523300141	6/23/2014	402101.0	2
21595	291310100	1/16/2015	400000.0	3
21596	1523300157	10/15/2014	325000.0	2

5 rows × 21 columns

In [545]:

```
# descriptive statistics of the dataset
houses_df.describe()
```

Out[545]:

	id	price	bedrooms	
count	2.159700e+04	2.159700e+04	21597.000000	21
mean	4.580474e+09	5.402966e+05	3.373200	
std	2.876736e+09	3.673681e+05	0.926299	
min	1.000102e+06	7.800000e+04	1.000000	
25%	2.123049e+09	3.220000e+05	3.000000	
50%	3.904930e+09	4.500000e+05	3.000000	
75%	7.308900e+09	6.450000e+05	4.000000	
max	9.900000e+09	7.700000e+06	33.000000	

In [546]:

```
#checking the unique values per column
houses_df.nunique()
```

Out[546]:

```
id               21420
date               372
price             3622
bedrooms            12
bathrooms           29
sqft_living       1034
sqft_lot          9776
floors               6
waterfront           2
view                 5
condition            5
grade               11
sqft_above         942
sqft_basement      304
yr_built           116
yr_renovated        70
zipcode             70
lat               5033
long               751
sqft_living15      777
sqft_lot15        8682
dtype: int64
```

Observations

1. The data used is in the form of a csv file.
2. The dataframe has 21597 rows and 21 columns.
3. The dataframe has 6 float64,9 int64 and 6 object data types.
4. The data contains the target and predictor variables needed.

Data preparation

a) Univariant analysis

- Data cleaning:Checking missing values and duplicates
- Dropping columns not used
- checking for outliers

In [547]:

```
# To check the the null values in the dataset
missing_data_check = houses_df.isnull().sum().sort_values
missing_data_check
```

Out[547]:

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

In [548]:

```
#percentage of missing values
total_percent = (houses_df.isnull().sum()/houses_df.isnul
missing_data = pd.concat([missing_data_check,total_percer
                          axis = 1,keys =["total missing v
missing_data
```

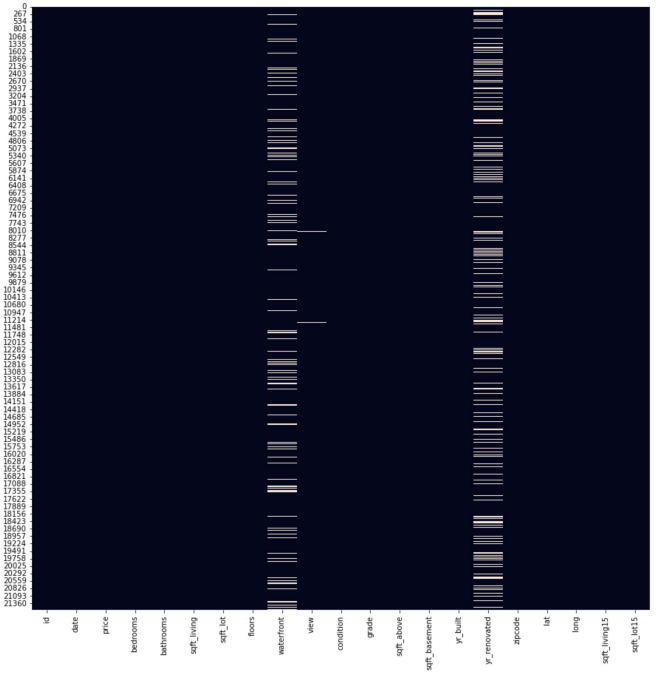
Out[548]:

	total missing values	missing values percentage
yr_renovated	3842	0.177895
waterfront	2376	0.110015
view	63	0.002917
id	0	0.000000
sqft_above	0	0.000000
sqft_living15	0	0.000000
long	0	0.000000
lat	0	0.000000
zipcode	0	0.000000
yr_built	0	0.000000
sqft_basement	0	0.000000
condition	0	0.000000
grade	0	0.000000
date	0	0.000000
floors	0	0.000000
sqft_lot	0	0.000000
sqft_living	0	0.000000
bathrooms	0	0.000000
bedrooms	0	0.000000

	total missing values	missing values percentage
price	0	0.000000
sqft_lot15	0	0.000000

In [549]:

```
# Visualizing the null values
plt.figure(figsize=(15, 15))
sns.heatmap(houses_df.isnull(), cbar=False);
```



observations

- "waterfront", "view" and "yr_renovated" contains null values. The percentage of missing value in this columns are 0.110015,0.002917 and 0.177895 respectively.

Data cleaning

Since the percentage of the null per column in our dataset is low i opted to drop all the null values in the dataset.

1.Dropping columns that will not be used
2.Dealing with the null values

In [550]:

```
# filling view column with "missing"
houses_df['view'].fillna('Missing', inplace=True)
```

In [551]:

```
# Dropping null values in the datase
houses_df.dropna(inplace = True)
```

In [552]:

```
# # Dropping columns that will not be used in the study
houses_df.drop(["yr_renovated","sqft_above","sqft_basemer
               "zipcode","sqft_lot15","sqft_living15","da
               axis=1,inplace = True)
```

In [553]:

```
#Checking for duplicate value
row,col = houses_df.shape
houses_df.drop_duplicates(inplace=True)

if houses_df.shape==(row,col):
    print('The dataset doesn\'t have any duplicates')
else:
    print('The dataset have duplicates')
```

```
The dataset have duplicates
```

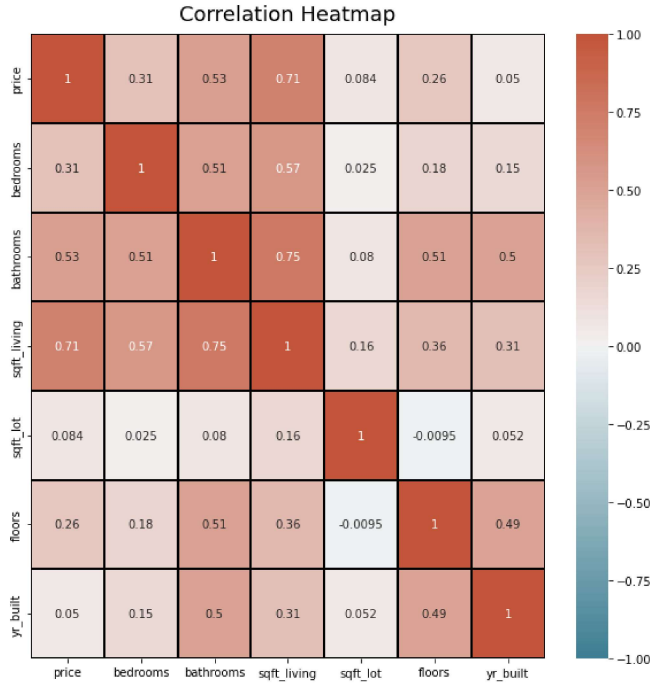
Observation

- The dataset has no duplicated values

b) Bivariate

In [554]:

```
# plotting a heatmap
# It shows the correlation between different column pairs
plt.figure(figsize=(10, 10))
cmap = sns.diverging_palette(220,20,n=200)
heatmap = sns.heatmap(
    houses_df.corr(),vmin=-1, vmax=1,center = 0,
    annot=True,cmap=cmap,linewidths=2, linecolor='black')
heatmap.set_title('Correlation Heatmap', fontdict={'fonts
```



Exploring 'price' column

1. Generate a heatmap to explore the price column

2. Compute the correlation coefficient for price column with other columns
3. Since 'price' column is our dependent variable i will analyse to see it relates with the independent variables

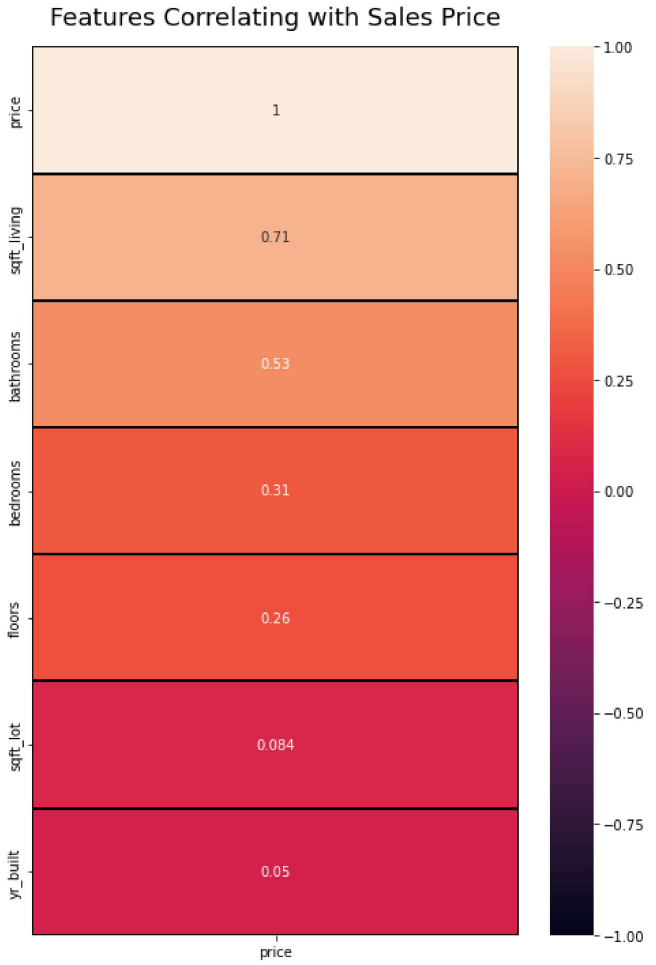
In [555]:

```
#descriptive statistics summary
summary_statistics = houses_df.price.describe()
print("count:",summary_statistics[0])
print("Mean of the price:",summary_statistics[1])
print("Standard deviation of mean:",summary_statistics[2]
print("Minimum value for price:",summary_statistics[3])
print("Lower quantile for mprice:",summary_statistics[4])
print("median for price:",summary_statistics[5])
print("Upper quantile for the price:",summary_statistics[
print("Maximum value for the price:",summary_statistics[7
```

```
count: 15804.0
Mean of the price: 541573.8255504936
Standard deviation of mean: 373976.8063981
49
Minimum value for price: 82000.0
Lower quantile for mprice: 321000.0
median for price: 450000.0
Upper quantile for the price: 645000.0
Maximum value for the price: 7700000.0
```

In [556]:

```
# plotting a heat map
# to show the correlation of price column with the indepe
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(
    houses_df.corr()[['price']].sort_values(by='price',as
    vmin=-1, vmax=1, annot=True,linewidths=2, linecolor=
heatmap.set_title('Features Correlating with Sales Price
```



Observations

There is a high correlation between sqft_living and bathroom with the price column

In [557]:

```
# function for checking for outliers
def outlier(data):
    q1=data.quantile(0.25)
    q3=data.quantile(0.75)
    IQR=q3-q1
    outliers = data[((data<(q1-1.5*IQR)) | (data>(q3+1.5*
    return outliers
# Checking the number of outliers in the 'price' column
print("Count of outliers:",outlier(houses_df.price).count
```

```
Count of outliers: 835
```

In [558]:

```
#skewness and kurtosis of price column
print("Skewness:",round(houses_df['price'].skew(),3))
print("Kurtosis:",round(houses_df['price'].kurt(),3))
```

```
Skewness: 4.287
Kurtosis: 38.865
```

Linear regression assumptions

a) Linearity

Relationship of 'price' column with numerical variables

In [559]:

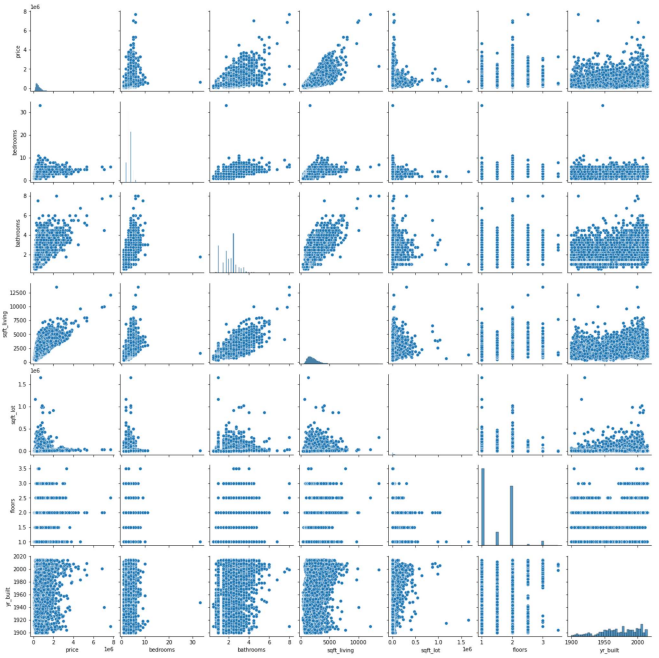
```
# to show the most correlated columns with price columns
PriceCorr = houses_df.corr()['price']
filteredcorr = PriceCorr[((PriceCorr >= .5) | (PriceCorr
filteredcorr
```

Out[559]:

```
sqft_living    0.706537
bathrooms      0.527319
Name: price, dtype: float64
```

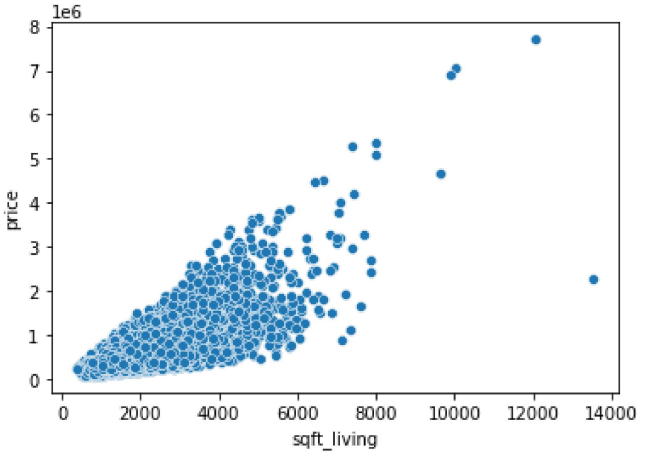
In [560]:

```
# Pairplot to show linearity
sns.pairplot(houses_df);
```



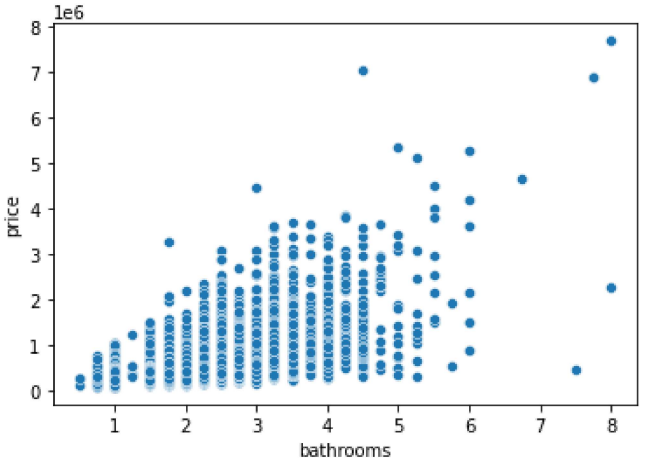
In [561]:

```
# A scatter plot between prie and sqft_living
living = houses_df["sqft_living"]
price = houses_df["price"]
sns.scatterplot(data = houses_df,x = living,y = price);
```



In [562]:

```
# A scatter plot between prie and bathrooms
bathrooms = houses_df["bathrooms"]
price = houses_df["price"]
sns.scatterplot(data = houses_df,x = bathrooms,y = price)
```



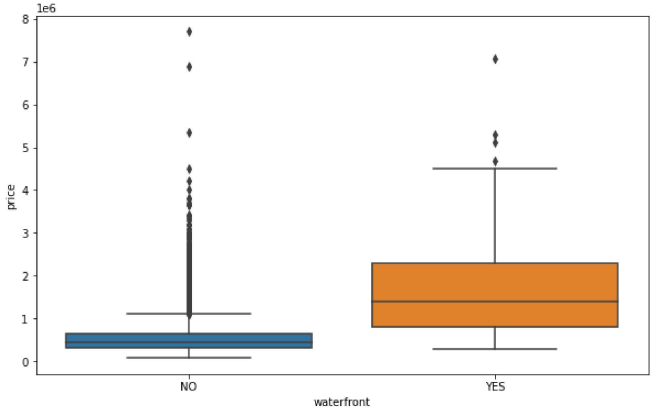
Observations

- 'sqft_living' and 'bathrooms' seem to have a linear relatationship with 'price' column. This relationships are positive which means that as one variable increases the other increases.

Relationship of 'price' column with categorical variables

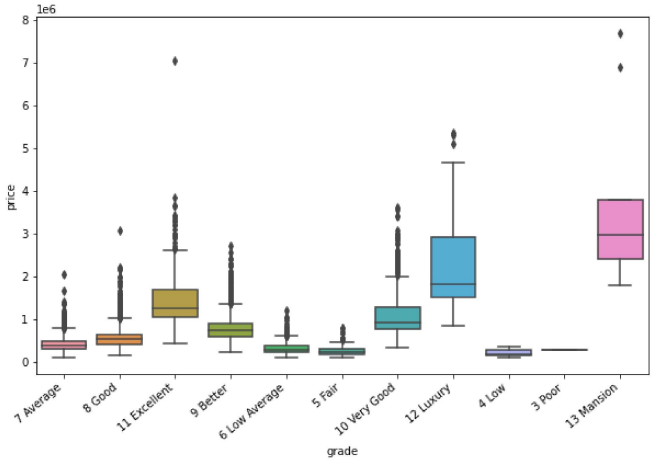
In [563]:

```
# Boxplot to show relationship between waterfront and pri
waterfront = houses_df.waterfront
data = pd.concat([houses_df['price'], houses_df["waterfro
fig, ax = plt.subplots(figsize=(10,6))
fig = sns.boxplot(x=waterfront,y=price,data=data);
```



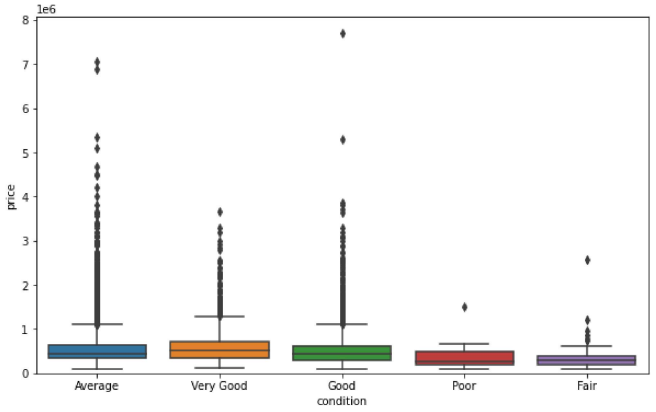
In [564]:

```
# Boxplot to show relationship between grade and prices
grade = houses_df.grade
data = pd.concat([houses_df['price'], houses_df["grade"]]
fig, ax = plt.subplots(figsize=(10,6))
fig = sns.boxplot(x=grade,y=price,data=data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha=
```



In [565]:

```
# Boxplot to show relationship between condition and pric
condition = houses_df.condition
data = pd.concat([houses_df['price'], houses_df["conditic
fig, ax = plt.subplots(figsize=(10,6))
fig = sns.boxplot(x=condition,y=price,data=data)
fig.axis(ymin=0);
#ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha
```



Observations

- 'waterfront' and "price are not highly related because the houses that are not near a waterfront have higher prices.
- The relationship seems to be stronger in the case of 'condition' where the box plot shows how prices decrease if the condition of the house is bad
- 'grade' and 'price' are not related.

b) Normality

This is to show if the data is normally distributed.

Density plots will help to visualize how the data is distributed.
Density plots to show the distributions of values in different columns.

checking the distribution of price

In [566]:

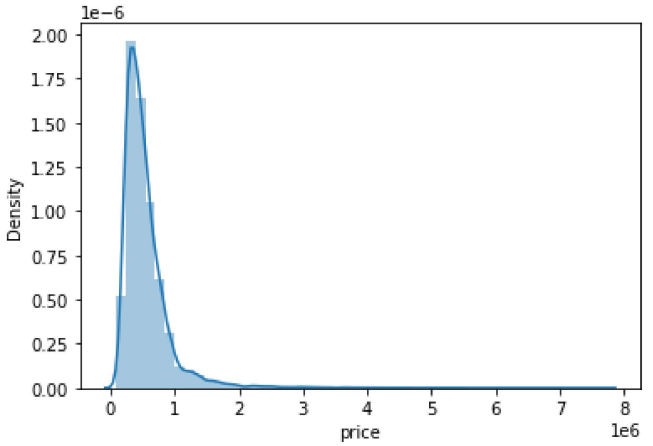
```
#skewness and kurtosis of price column
print("Skewness:",round(houses_df['price'].skew(),3))
print("Kurtosis:",round(houses_df['price'].kurt(),3))

# Distribution of prices

# plotting a distribution plot for price

sns.distplot(houses_df['price']);
```

```
Skewness: 4.287
Kurtosis: 38.865
```



Observations Of The Distribution Plot Of 'price' Column

- The plot shows that The data is not normally distributed.
- The plot shows that the price column displaces a Leptokurtic kurtosis
- The plot shows that the price column is Highly positively skewed

checking the distribution of sqft_living

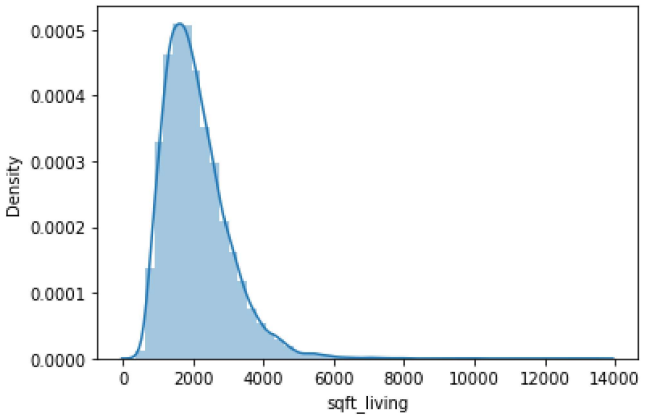
In [567]:

```
#skewness and kurtosis of sqft_living
print("Skewness:",round(houses_df['sqft_living'].skew(),3
print("Kurtosis:",round(houses_df['sqft_living'].kurt(),3

# Distribution of sqft_living

# plotting a distribution plot for sqft_living
sns.distplot(houses_df['sqft_living']);
```

```
Skewness: 1.505
Kurtosis: 5.764
```



Observations Of The Distribution Plot Of "sqft_living Column

- The plot shows that the data is not normally distributed.

- The plot shows that the sqft_living column displaces a Leptokurtic kurtosis having a kurtosis of 5.
- The plot shows that the sqft_living column have a positive skewness.

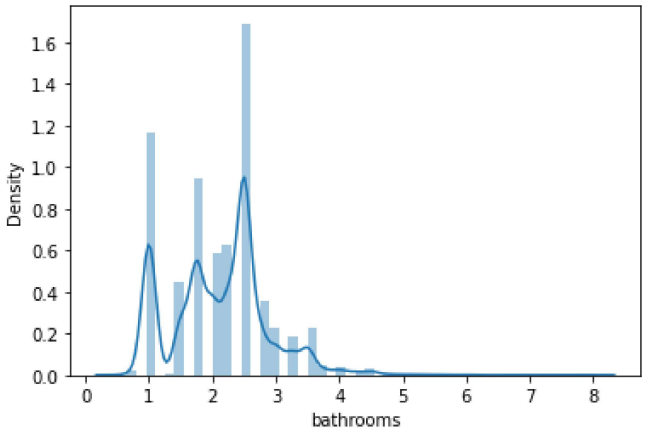
checking the distribution of bathrooms

In [568]:

```
#skewness and kurtosis of bathrooms column
print("Skewness:",round(houses_df['bathrooms'].skew(),3))
print("Kurtosis:",round(houses_df['bathrooms'].kurt(),3))

# Distribution of bathrooms
sns.distplot(houses_df['bathrooms']); # Density plot
```

```
Skewness: 0.524
Kurtosis: 1.367
```



Observations Of The Distribution Plot Of "bathrooms" Column

- The plot shows that the data is normally distributed.
- The plot shows that the bathroom column displaces a platykurtic kurtosis having a kurtosis of 1.

- The plot shows that the bathroom column have normal skewness.

Simple linear regression Model

For this model i will use sqft_living as the independent variable that will help us to predict the price of the houses.

Determining the independent variable (x) and independent variable (y)

In [569]:

```
# independent variable

X_baseline = houses_df['sqft_living']

# dependent variable

y = houses_df['price']
```

Creating the model

In [570]:

```
# Creating a simple linear regression model

model = sm.OLS(endog=y, exog=sm.add_constant(X_baseline))

results = model.fit()   # Fitting the model
```

In [571]:

```
# model summary
results.summary()
```

Out[571]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.499
Model:	OLS	**Adj. R-squared:**	0.499
Method:	Least Squares	**F-statistic:**	1.575e+04
Date:	Thu, 29 Sep 2022	**Prob (F-statistic):**	0.00
Time:	23:44:47	**Log-Likelihood:**	-2.1976e+05
No. Observations:	15804	**AIC:**	4.395e+05
Df Residuals:	15802	**BIC:**	4.395e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|
const	-5.754e+04	5217.330	-11.030	0.000
sqft_living	287.3399	2.289	125.504	0.000

Omnibus:	11314.260	**Durbin-Watson:**	1.971
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	481039.886
Skew:	2.958	**Prob(JB):**	0.00
Kurtosis:	29.372	**Cond. No.**	5.65e+03

```
Notes:
[1] Standard Errors assume that the covariance
matrix of the errors is correctly specified.
[2] The condition number is large, 5.65e+03. This
might indicate that there are
strong multicollinearity or other numerical
problems.
```

In [572]:

```
results.params
```

Out[572]:

```
const          -57544.943355
sqft_living      287.339859
dtype: float64
```

Interpreting the model

Our model smmary tells us that:

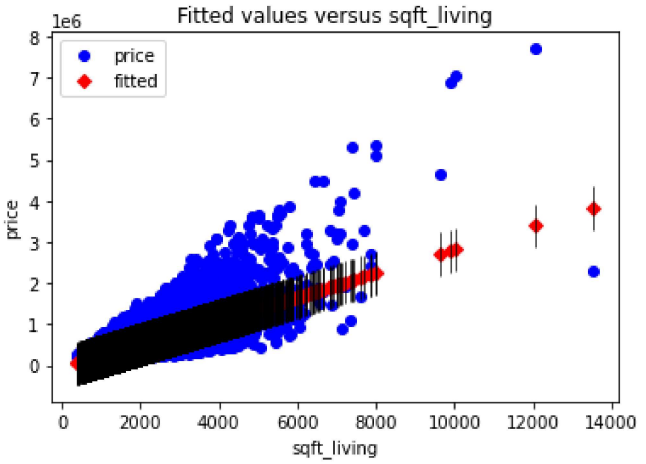
1. The model is statistically significant having F-statistic p-value below the alpha,given that the alpha is 0.05.
2. The model coefficients (`const` and `sqft_living`) are both statistically significant having t-statistic p-values below the alpha,given that the alpha is 0.05.
3. The model explains about 50% of the variance in price.
4. If a house's sqft_living is at 0 , we would expect a decrese in the price by about about 57544.943355 .
5. An increase of 1 in sqft_living is associated with an increase of about 287.339859 in price.

   Our regression line is:

$$\hat{price} = -57544.943355 + 287.339859\ sqft\_living$$

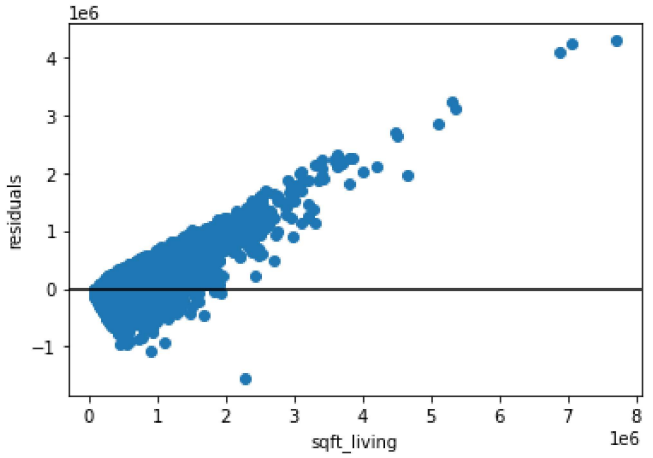
In [573]:

```
# Visualizing our model
sm.graphics.plot_fit(results, "sqft_living")
plt.show()
```



In [574]:

```
# visualizing the distribution of the residuals
fig, ax = plt.subplots()
ax.scatter(houses_df["price"], results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_living")
ax.set_ylabel("residuals");
```



Multiple linear regression

First multiple linear regression model

Determining the dependent and independent variables

For this model i am going to use sqft_living and bathrooms as the independent variables

In [575]:

```
# independent variable
fx = houses_df[[ 'sqft_living', "bathrooms"]]
# dependent variable
y = houses_df['price']

# fx_var = pd.get_dummies(fx,drop_first=True)
```

In [576]:

```
lr1 = sm.OLS(endog = y,exog = sm.add_constant(fx)).fit()
lr1.summary()
```

Out[576]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.499
Model:	OLS	**Adj. R-squared:**	0.499
Method:	Least Squares	**F-statistic:**	7877.
Date:	Thu, 29 Sep 2022	**Prob (F-statistic):**	0.00
Time:	23:44:48	**Log-Likelihood:**	-2.1975e+05
No. Observations:	15804	**AIC:**	4.395e+05
Df Residuals:	15801	**BIC:**	4.395e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	
const	-5.233e+04	6215.544	-8.419	0.000	-
sqft_living	291.4014	3.488	83.548	0.000	
bathrooms	-6452.3221	4180.153	-1.544	0.123	-

Omnibus:	11297.646	**Durbin-Watson:**	1.971
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	478782.248
Skew:	2.952	**Prob(JB):**	0.00

Kurtosis:	29.310	**Cond. No.**	7.32e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.32e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [577]:

```
lr1.params
```

Out[577]:

```
const          -52329.890631
sqft_living      291.401373
bathrooms      -6452.322117
dtype: float64
```

Interpreting the model

Our model summary tells us that:

1. The model is statistically significant having F-statistic p-value below the alpha,given that the alpha is 0.05.
2. The model coefficients ('const','bathrooms' and `sqft_living`) are both statistically significant having t-statistic p-values below the alpha,given that the alpha is 0.05.
3. The model explains about 50% of the variance in price.
4. If a house's sqft_living and bathrooms is at 0 , we would expect a decrese in the price by about about 52329.890631 dollars .
5. An increase of 1 in sqft_living is associated with an increase of about 291.401373 dollars.

6. An increase of 1 bathroom is associated with a decrease of about 6452.322117 dollars.

Evaluate the performance

- Mean absolute error
- Mean Square Error
- Root mean Squared Error

In [578]:

```
y_pred = lr1.predict(sm.add_constant(fx))
print(f"Mean absolute error: {np.round(mean_absolute_erro
print(f"Root mean squared error: {np.round(np.sqrt(mean_s
print(f"Adjusted r_squarde: {lr1.rsquared_adj}")
```

```
Mean absolute error: 175013.7849
Root mean squared error: 264626.0732
Adjusted r_squarde: 0.49920685528060715
```

Interpreting the Mean absolute error and root mean squared error

It means that our model is off by about 175013.7849 USD in a given prediction.

Interpreting the root mean squared error

It means that our model is off by about 264626.0732 USD in a given prediction.

Interpreting the Adjusted r_squarded

The model explains about 50% of the variance in price

The mean absolute error and the root mean squared error for this model is very high meaning its accuracy is low, therefore i will create

another model including more data to increase the accuracy of the model.

Second multiple linear regression model

Determining the dependent and independent variables

For this model i am going to use all the numerical variables as the independent variables

In [579]:

```
# independent variable
sx = houses_df[[ 'sqft_living', "bathrooms","bedrooms","s
# dependent variable
y = houses_df['price']
```

In [580]:

```
lr2 = sm.OLS(endog = y,exog = sm.add_constant(sx)).fit()
lr2.summary()
```

Out[580]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.564
Model:	OLS	**Adj. R-squared:**	0.564
Method:	Least Squares	**F-statistic:**	3406.
Date:	Thu, 29 Sep 2022	**Prob (F-statistic):**	0.00
Time:	23:44:49	**Log-Likelihood:**	-2.1866e+05
No. Observations:	15804	**AIC:**	4.373e+05
Df Residuals:	15797	**BIC:**	4.374e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|
const	6.709e+06	1.59e+05	42.203	0.000
sqft_living	310.6223	3.518	88.289	0.000
bathrooms	6.945e+04	4541.934	15.292	0.000
bedrooms	-7.047e+04	2623.302	-26.864	0.000
sqft_lot	-0.3048	0.048	-6.357	0.000
yr_built	-3448.5415	82.514	-41.793	0.000
floors	5.234e+04	4465.384	11.721	0.000

Omnibus:	10921.847	**Durbin-Watson:**	1.975
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	455848.303
Skew:	2.806	**Prob(JB):**	0.00
Kurtosis:	28.705	**Cond. No.**	3.61e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.61e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [581]:

```
lr2.params
```

Out[581]:

```
const          6.708607e+06
sqft_living    3.106223e+02
bathrooms      6.945494e+04
bedrooms      -7.047140e+04
sqft_lot      -3.047603e-01
yr_built      -3.448542e+03
floors         5.233927e+04
dtype: float64
```

Interpreting the model

Our model summary tells us that:

- The model is statistically significant having F-statistic p-value below the alpha,given that the alpha is 0.05.
- The model coefficients ('const','bathrooms','sqft_living','bedrooms,'sqft_lot,'yr_built' and 'floors') are both statistically significant having t-statistic p-values below the alpha,given that the alpha is 0.05.
- The model explains about 56% of the variance in price.
- If all this house's features are at 0 , we would expect a decrese in the price by about about 6.708607e+06 dollars .
- An increase of 1 in sqft_living is associated with an increase of about 6.945494e+04 dollars.
- An increase of 1 bathroom is associated with a increase of about 6.945494e+04 dollars.
- An increase of 1 bedroom is associated with a decrease of about 7.047140e+04 dollars.
- An increase of 1 sqft_lot is associated with a decrease of about 3.047603e-01 dollars.
- An increase of 1 yr_built is associated with a decrease of about 3.448542e+03 dollars.
- An increase of 1 floor is associated with a increase of about 5.233927e+04 dollars.

Evaluate the performance

- Mean absolute error
- Mean Square Error
- Root mean Squared Error

In [582]:

```
sy_pred = lr2.predict(sm.add_constant(sx))
print(f"Mean absolute error: {np.round(mean_absolute_erro
print(f"Root mean squared error: {np.round(np.sqrt(mean_s
print(f"Adjusted r_squarde: {lr2.rsquared_adj}")
```

```
Mean absolute error: 160823.3452
Root mean squared error: 246918.0566
Adjusted r_squarde: 0.5638772257935419
```

Interpreting the Mean absolute error and root mean squared error

It means that our model is off by about160823.3452 USD in a given prediction.

Interpreting the root mean squared error

It means that our model is off by about 246918.0566 USD in a given prediction.

Interpreting the Adjusted r_squarded

The model explains about 56% of the variance in price

conclusion

The mean absolute error for this model has dropped from 175013.7849 to 160823.3452 from the first model.

The root mean squared error for this model has also dropped for 264626.0732 to 246918.0566 from the first model .

This means that this model is better that our first model.

However,the accuracy of the model is still low. I'll add more data to the model and see if it's accuracy increases.

Third multiple linear regression model

Determining the dependent and independent variables

For this model i am going to use all the target varibles as the independent variables

In [583]:

```
# independent variable
tx = houses_df[[ 'sqft_living', "bathrooms","bedrooms","s
                "condition","grade"]]
# dependent variable
y = houses_df['price']

tx_var = pd.get_dummies(tx,drop_first=True)
```

In [584]:

```
lr3 = sm.OLS(endog = y,exog = sm.add_constant(tx_var)).fi
lr3.summary()
```

Out[584]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.677
Model:	OLS	**Adj. R-squared:**	0.677
Method:	Least Squares	**F-statistic:**	1577.
Date:	Thu, 29 Sep 2022	**Prob (F-statistic):**	0.00
Time:	23:44:49	**Log-Likelihood:**	-2.1629e+05
No. Observations:	15804	**AIC:**	4.326e+05
Df Residuals:	15782	**BIC:**	4.328e+05
Df Model:	21		
Covariance Type:	nonrobust		

	coef	std err	t	P:
const	7.443e+06	1.52e+05	48.954	0.0
sqft_living	151.8199	3.898	38.947	0.0
bathrooms	5.813e+04	3967.334	14.651	0.0
bedrooms	-2.956e+04	2349.145	-12.584	0.0
sqft_lot	-0.2680	0.041	-6.470	0.0
yr_built	-3532.0176	77.356	-45.659	0.0
floors	2.819e+04	3991.131	7.063	0.0
waterfront_YES	7.491e+05	1.97e+04	38.028	0.0

condition_Fair	-3.001e+04	1.89e+04	-1.585	0.1
condition_Good	1.457e+04	4209.342	3.461	0.0
condition_Poor	-1.174e+04	4.92e+04	-0.239	0.8
condition_Very Good	5.39e+04	6757.913	7.976	0.0
grade_11 Excellent	2.341e+05	1.47e+04	15.875	0.0
grade_12 Luxury	7.946e+05	2.78e+04	28.551	0.0
grade_13 Mansion	1.859e+06	6.57e+04	28.312	0.0
grade_3 Poor	-5.711e+05	2.13e+05	-2.681	0.0
grade_4 Low	-5.298e+05	5.46e+04	-9.699	0.0
grade_5 Fair	-5.493e+05	2e+04	-27.477	0.0
grade_6 Low Average	-4.953e+05	1.21e+04	-41.008	0.0
grade_7 Average	-4.17e+05	9890.121	-42.166	0.0
grade_8 Good	-3.257e+05	9068.825	-35.917	0.0
grade_9 Better	-1.83e+05	9063.725	-20.189	0.0

Omnibus:	9302.173	**Durbin-Watson:**	1.972
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	355182.803
Skew:	2.223	**Prob(JB):**	0.00
Kurtosis:	25.795	**Cond. No.**	5.62e+06

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

[2] The condition number is large, 5.62e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [585]:

```
lr3.params
```

Out[585]:

```
const                  7.443015e+06
sqft_living            1.518199e+02
bathrooms              5.812563e+04
bedrooms              -2.956081e+04
sqft_lot              -2.680492e-01
yr_built              -3.532018e+03
floors                 2.818848e+04
waterfront_YES         7.490893e+05
condition_Fair        -3.001413e+04
condition_Good         1.456809e+04
condition_Poor        -1.174129e+04
condition_Very Good    5.389874e+04
grade_11 Excellent     2.340605e+05
grade_12 Luxury        7.945561e+05
grade_13 Mansion       1.859227e+06
grade_3 Poor          -5.711283e+05
grade_4 Low           -5.297917e+05
grade_5 Fair          -5.493480e+05
grade_6 Low Average   -4.953342e+05
grade_7 Average       -4.170269e+05
grade_8 Good          -3.257280e+05
grade_9 Better        -1.829845e+05
dtype: float64
```

Interpreting the model

Our model summary tells us that:

- The model is statistically significant having F-statistic p-value below the alpha,given that the alpha is 0.05.

- The model coefficients ('const','bathrooms','sqft_living','bedrooms,'sqft_lot,'yr_built' and 'floors') are both statistically significant having t-statistic p-values below the alpha,given that the alpha is 0.05.
- The model explains about 68% of the variance in price.
- If all this house's features are at 0 , we would expect a decrese in the price by about about 7.443015e+06 dollars .
- An increase of 1 in sqft_living is associated with an increase of about 1.518199e+02 dollars.
- An increase of 1 bathroom is associated with a increase of about 5.812563e+04 dollars.
- An increase of 1 bedroom is associated with a decrease of about 2.956081e+04 dollars.
- An increase of 1 sqft_lot is associated with a decrease of about 2.680492e-01 dollars.
- An increase of 1 yr_built is associated with a decrease of about 3.448542e+03 dollars.
- An increase of 1 floor is associated with a increase of about 3.532018e+03 dollars.

Evaluate the performance

- Mean absolute error
- Mean Square Error
- Root mean Squared Error

In [586]:

```
ty_pred = lr3.predict(sm.add_constant(tx_var))
print(f"Mean absolute error: {np.round(mean_absolute_erro
print(f"Root mean squared error: {np.round(np.sqrt(mean_s
print(f"Adjusted r_squarde: {lr3.rsquared_adj}")
```

```
Mean absolute error: 138476.1124
Root mean squared error: 212468.1853
Adjusted r_squarde: 0.6767761029968034
```

Interpreting the Mean absolute error and root mean squared error

It means that our model is off by about 138476.1124 USD in a given prediction.

Interpreting the root mean squared error

It means that our model is off by about 212468.1853 USD in a given prediction.

Interpreting the Adjusted r_squarded

The model explains about 68% of the variance in pric

conclusion

The mean absolute error for this model has dropped from 160823.3452 to 138476.1124 from our second model.

The root mean squared error for this model has also dropped from 246918.0566 to 212468.1853 from our second model.

This means that this model is better than the second model.

MODEL RESULTS

- The third model where all the predictor variables were added explains about 68% of variance in price unlike the baseline model, first model and second model that explains about 50%, 50% and 56% of variance respectively.
- All the models were statistically significant having a p_value of 0.00 given an alpha of 0.05.
- The r_squared of our model increased as the number of predictor variables increased.
- The third model had the lowest mean absolute error and root mean squared error among the three models. This means that it has a higher accuracy compared to the other models.
- The third model is off by about 138476.1124 USD in a given prediction.
- This means that cherie housing agency can use this model but for every prediction using the model, the prediction made would be off by 138476.1124 USD.
- However if more predictor variables were added to our model, the accuracy would increase.

LIMITATIONS OF THE MODEL

- One of the limitation of the model is that it assumes that there is a linear relationship between the dependent and the independent variables
- If there is any non-linear relationship between any independent variable and the independent variable, the model will perform poorly.

RECOMMENDATION

- The third model will be useful to cherie housing agency since they will be able to give good advice to homeowners by using it to predict the price of the houses.
- Some predictor variables such as condition of the house affects the price of the house.
- If the condition of the house is 'very good',there will be an increase in the price of the house.
- Also if the number of bedrooms,larger square foot of the living room and number of floors would lead to increase in the price of the house.
- I would recommend cherie housing agency to use the third model i making prediction of the price.