index

July 17, 2024

1 1. Introduction

In the King County housing market, multiple dynamics play a major role in shaping the prosperity of Real Estate companies. Real estate development companies face significant challenges in navigating many complexities when it comes to deciding on the most important features to include in homes.

The project aims to help a Real Estate Development Company looking to enter the King County housing market. This will be done by providing a comprehensive analysis that focuses on the key factors that influence housing prices. By leveraging data-driven approaches, we aim to uncover insights that will empower the stakeholder to make informed decisions.

With the insights gained from the data from the King County House Sales dataset, we hope to advise the Real Estate Development Company on the most important characteristics/features to focus on when constructing homes, in order to generate more income and be as profitable as possible.

2 2. Problem Statement- Business Problem

The Real Estate Development Company is venturing into the King County Housing market and would like to develop a sales strategy that makes it profitable by targeting houses that have the highest prices in the market.

3 3. Objectives

1. Explore the Relationship Between Number of bathrooms and its price

- Investigate the relationship between the number of bathrooms (`bathrooms`) in a house and its price (`price`). Determine if houses with more bathrooms are priced higher.

2. Explore the relationship between the square footage of the home and its price

- Investigate the relationship between the square footage of the home (`sqft_living`) and its price (`price`). Determine if houses with more square footage are priced higher. #### 3. Explore the impact of the King County housing grading system on the pricing of homes in the market.
- Investigate the relationship between the grades awarded to a house (`grade`) and its price. Determine if houses with higher grades are priced higher. #### 4. Explore the impact of a home's neighborhood on its price

- Investigate the relationship between square footage of interior housing living space for the nearest 15 neighbors of a house and its price. Determine if the size of homes within a neighborhood affects the price of a house.

5. Develop a Linear Regression Model to Predict Housing Prices

- Build and evaluate a linear regression model using features `bathrooms`, `sqft_living`, and `grade`to predict house prices (`price`). Provide the stakeholder with a predictive tool for estimating housing prices and supporting strategic decision-making in housing development.

4 4. Data Understanding:

This project uses the King County House Sales dataset whose size, descripritive statistics for all the features used in the analysis, and justification of the inclusion of features based on their properties and relevance for the project have been given in the Exploratory Data Analysis part.

The data contains all the relevant features that are needed to meet our objectives as described above.

However, to explain the relationship fully, additional features are needed.

4.0.1 Limitations of the data that have implications for the project

- The data is not normally distributed, which may impact the reliability of the model especially in the extreme of the price distribution. The data was scaled to deal with non-normality.
- There are some missing values for some columns, which will have to be dealt with.

4.1 Imports and Data Loading

[254]:
```
#imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from statsmodels.formula.api import ols
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
```

[255]:
```
data = pd.read_csv('./Data/kc_house_data.csv')
data.head(4)
```

[255]:

	id	date	price	bedrooms	bathrooms	sqft_living	\
0	7129300520	10/13/2014	221900.0	3	1.00	1180	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	
2	5631500400	2/25/2015	180000.0	2	1.00	770	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	

```
   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0      5650     1.0         NaN   0.0  …      7        1180            0.0
1      7242     2.0         0.0   0.0  …      7        2170          400.0
2     10000     1.0         0.0   0.0  …      6         770            0.0
3      5000     1.0         0.0   0.0  …      7        1050          910.0

  yr_built  yr_renovated  zipcode      lat     long  sqft_living15  sqft_lot15
0     1955           0.0    98178  47.5112 -122.257           1340        5650
1     1951        1991.0    98125  47.7210 -122.319           1690        7639
2     1933           NaN    98028  47.7379 -122.233           2720        8062
3     1965           0.0    98136  47.5208 -122.393           1360        5000

[4 rows x 21 columns]
```

4.2 Exploratory data analysis

[256]:
```
data.shape
```

[256]:
```
(21597, 21)
```

There are 21 columns and 25,597 rows

[257]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21597 non-null  int64
 1   date           21597 non-null  object
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64
 6   sqft_lot       21597 non-null  int64
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  int64
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  object
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
```

```
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
```

[258]:
```
data.columns
```

[258]:
```
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', 'sqft_lot15'],
      dtype='object')
```

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

A data description of the numerical columns

[259]:
```
data.describe()
```

[259]:

	id	price	bedrooms	bathrooms	sqft_living	\
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	

	sqft_lot	floors	waterfront	view	condition	\
count	2.159700e+04	21597.000000	19221.000000	21534.000000	21597.000000	
mean	1.509941e+04	1.494096	0.007596	0.233863	3.409825	
std	4.141264e+04	0.539683	0.086825	0.765686	0.650546	
min	5.200000e+02	1.000000	0.000000	0.000000	1.000000	
25%	5.040000e+03	1.000000	0.000000	0.000000	3.000000	
50%	7.618000e+03	1.500000	0.000000	0.000000	3.000000	
75%	1.068500e+04	2.000000	0.000000	0.000000	4.000000	

```
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above      yr_built  yr_renovated       zipcode  \
count  21597.000000  21597.000000  21597.000000  17755.000000  21597.000000
mean       7.657915   1788.596842   1970.999676     83.636778  98077.951845
std        1.173200    827.759761     29.375234    399.946414     53.513072
min        3.000000    370.000000   1900.000000      0.000000  98001.000000
25%        7.000000   1190.000000   1951.000000      0.000000  98033.000000
50%        7.000000   1560.000000   1975.000000      0.000000  98065.000000
75%        8.000000   2210.000000   1997.000000      0.000000  98118.000000
max       13.000000   9410.000000   2015.000000   2015.000000  98199.000000

                lat          long  sqft_living15     sqft_lot15
count  21597.000000  21597.000000   21597.000000   21597.000000
mean      47.560093   -122.213982    1986.620318   12758.283512
std        0.138552      0.140724     685.230472   27274.441950
min       47.155900   -122.519000     399.000000     651.000000
25%       47.471100   -122.328000    1490.000000    5100.000000
50%       47.571800   -122.231000    1840.000000    7620.000000
75%       47.678000   -122.125000    2360.000000   10083.000000
max       47.777600   -121.315000    6210.000000  871200.000000
```

5 5. Data Cleaning

5.0.1 Checking for duplicates

[260]:
```
data.duplicated().sum()
```

[260]:
```
0
```

There are no duplicates

5.0.2 Looking for null values

[261]:
```
data.isna().sum()
```

[261]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront     2376
view             63
condition         0
```

```
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated    3842
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

The waterfront, view, and year renovated columns have null values

- waterfront, view, yr_renovated: 2376, 63, 3842 missing values respectively
- is,date, price, bedrooms, sqft_living, sqft_lot, floors, condition, grade, sqft_above, sqft_basement, yr_built, zipcode, lat, long, sqft_living15, sq_lot15: No missing values.

[262]:
```
# We will fill null values in the waterfront and view columns with the mode␣
 ↪since these are categorical,
# We will also convert them to the string datatype, which works for categorical␣
 ↪data

# and for the year renovated, fill with the median since there are possibly␣
 ↪outliers
data['view'] = data['view'].astype(str)
data['waterfront'] = data['waterfront'].astype(str)

wf_mode = data.waterfront.mode()
view_mode = data.view.mode()
median_year = data.yr_renovated.median()

data.waterfront.fillna(wf_mode, inplace = True)
data.view.fillna(view_mode, inplace = True)
data.yr_renovated.fillna(median_year, inplace = True)
```

[263]:
```
data.isna().sum()
```

[263]:
```
id             0
date           0
price          0
bedrooms       0
bathrooms      0
sqft_living    0
sqft_lot       0
floors         0
waterfront     0
view           0
```

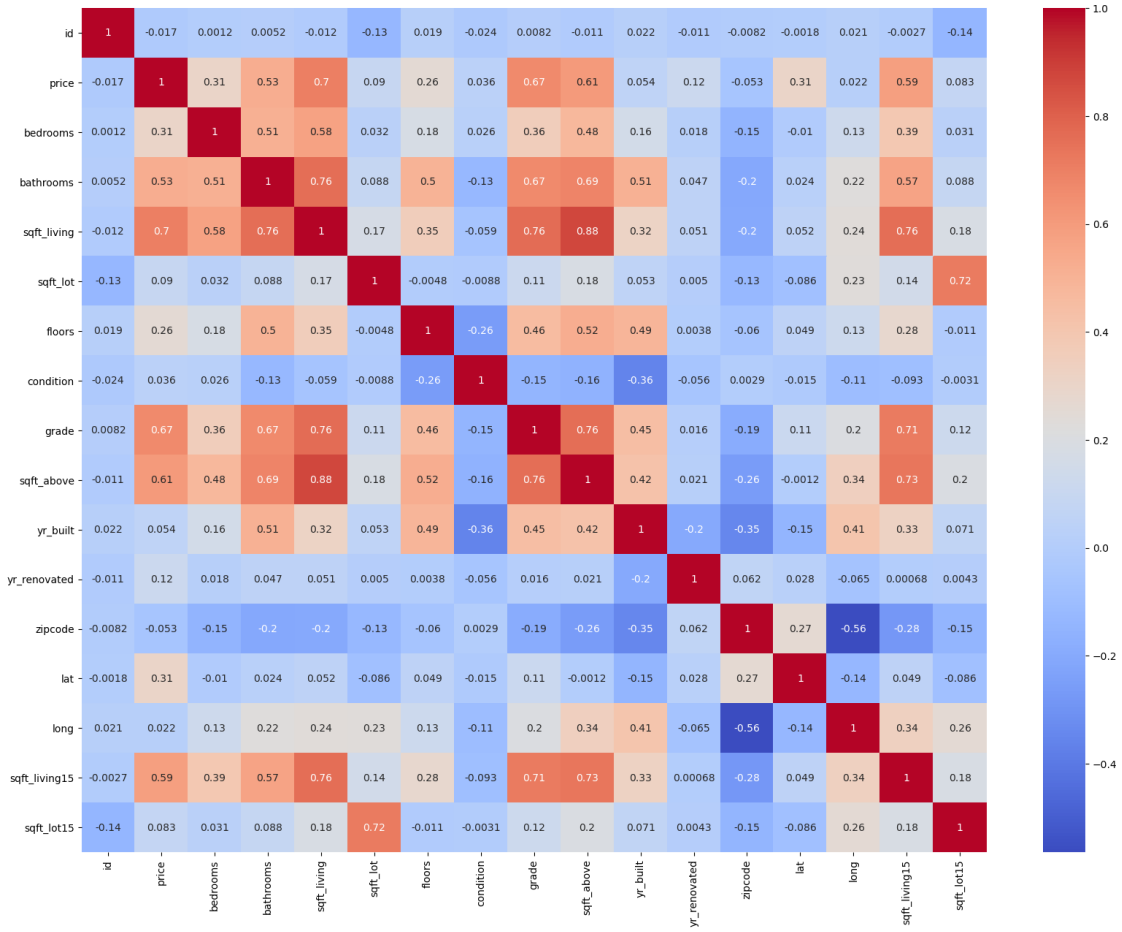
```
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

There are now no null values

6 6. Data Preparation

Correlation Heatmap A correlation heatmap to obtain numbers for an easier reading of the correlation of the relationship between columns

[264]:
```
plt.subplots(figsize=(20,15))
sns.heatmap(data.corr(),cmap="coolwarm",annot=True);
```



We have already determined that our predicted column is price, and will, therefore, be looking at the correlation of the rest of the columns with price Choose the features that have the highest collinearity with the price. These are: - Bathrooms - sqft_living - grade - sqft_above - sqft_living15

Based on the correlation coefficients with price from the KC Housing Data dataset, the most important features are:

```
Bathrooms- Number of bathrooms
    Justification: Bathrooms RM has a strong positive correlation with price (0.53). This indi


sqft_living - Footage of the home
    Justification: sqft_living has a very strong positive correlation with price (0.7). This i

grade - overall grade given to the housing unit, based on King County grading system
    Justification: Grade has a strong positive correlation with price (0.67). This indicates t

sqft_above - square footage of house apart from basement
    Justification: sqft_above has a strong correlation with price (0.61). This indicates that
```

```
sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbo
    Justification: sqft_living 15 has a strong correlation with price (0.59). This indicates t
```

6.0.1 Checking the correlations for the selected important features

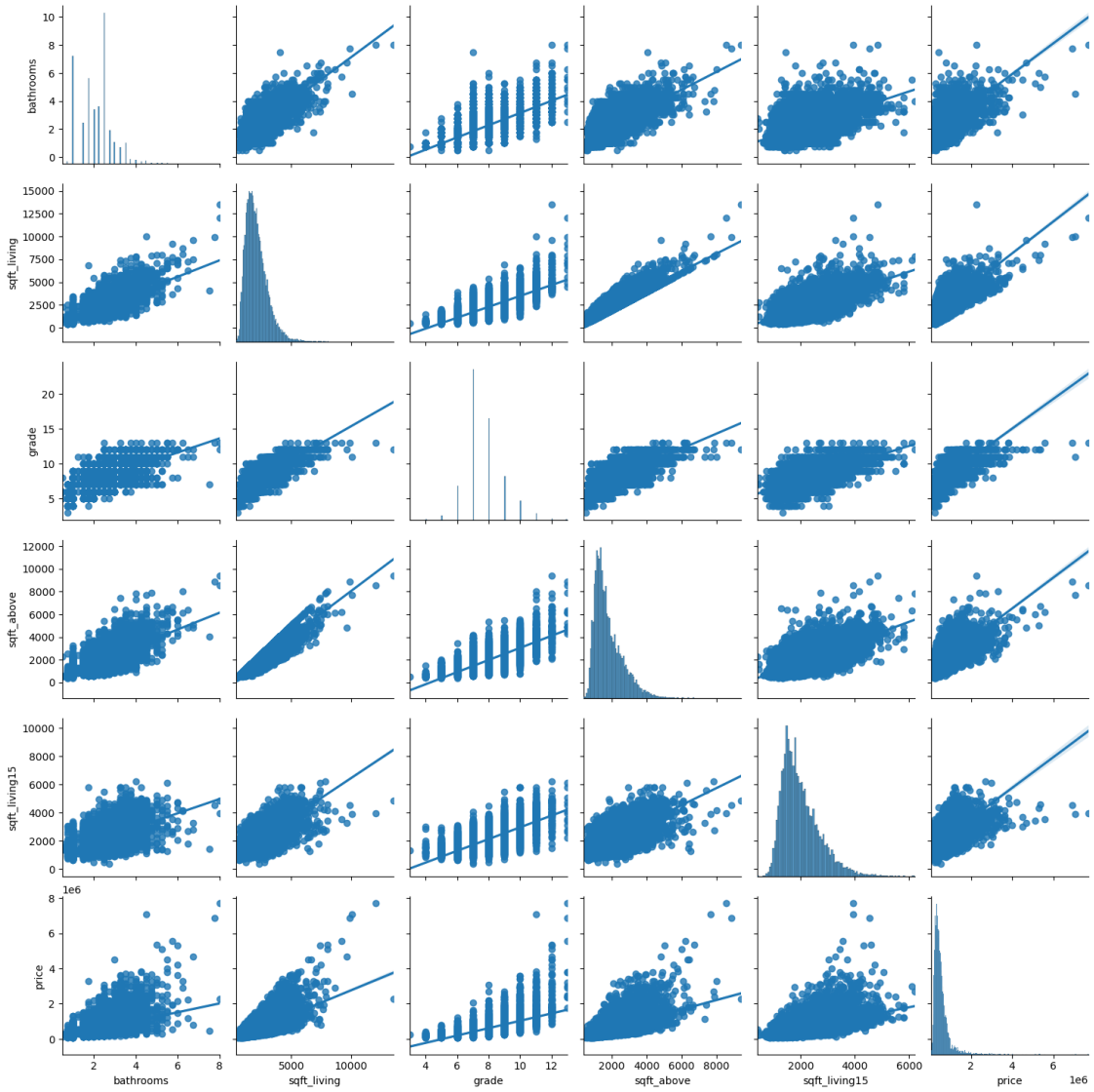
Create a pairplot with a regression line to show the best fit line

[265]:
```
data2 = data[['bathrooms','sqft_living','grade','sqft_above','sqft_living15',␣
 ↪'price']]
data2.head()
```

[265]:

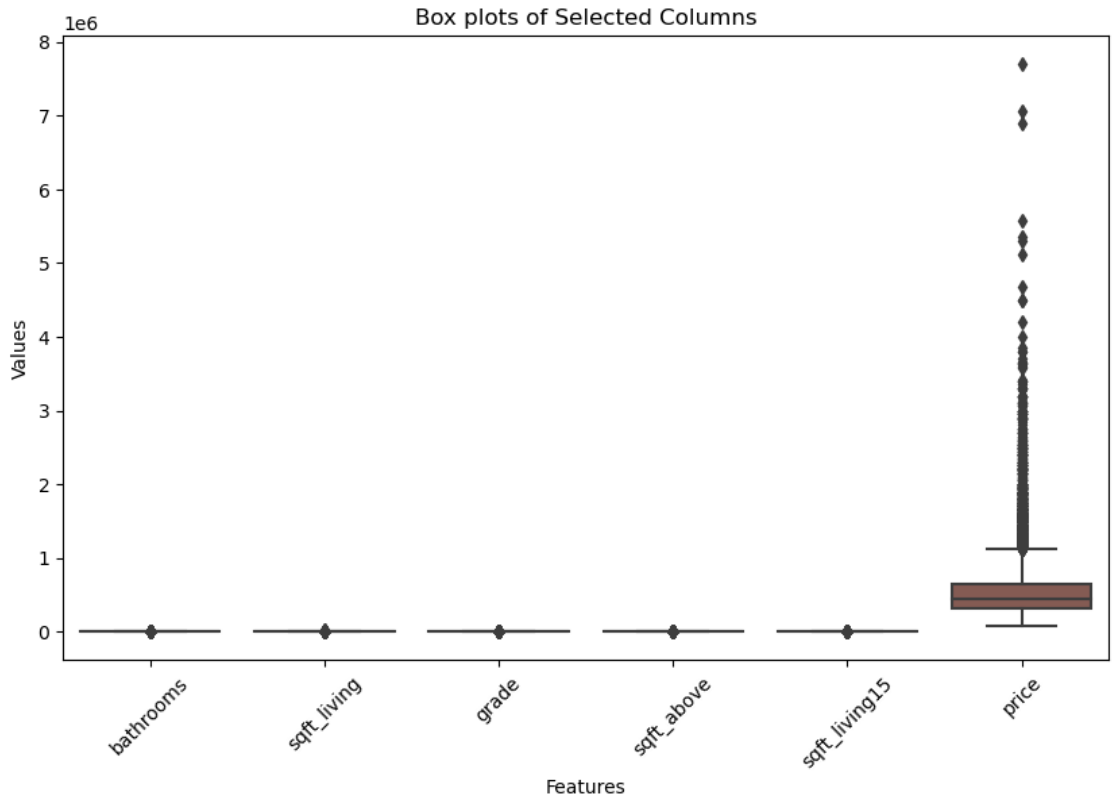
	bathrooms	sqft_living	grade	sqft_above	sqft_living15	price
0	1.00	1180	7	1180	1340	221900.0
1	2.25	2570	7	2170	1690	538000.0
2	1.00	770	6	770	2720	180000.0
3	3.00	1960	7	1050	1360	604000.0
4	2.00	1680	8	1680	1800	510000.0

[266]:
```
sns.pairplot(data2, kind = "reg");
```

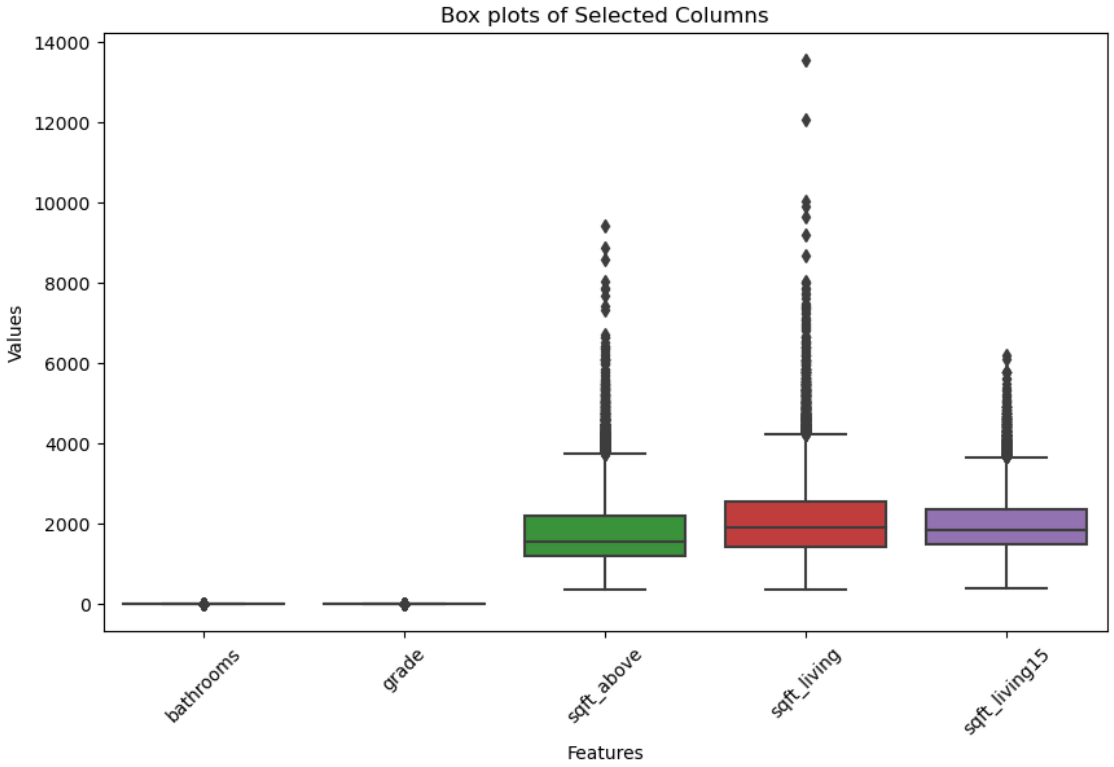


Check for outliers

[267]:
```
# Plotting box plots for each column
plt.figure(figsize=(10, 6))
sns.boxplot(data = data2)
plt.title('Box plots of Selected Columns')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.show();
```



[268]:
```
# Plotting box plots for each column
plt.figure(figsize=(10, 6))
sns.boxplot(data = data2[['bathrooms', 'grade', 'sqft_above', 'sqft_living',␣
 ↪'sqft_living15']])
plt.title('Box plots of Selected Columns')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.show();
```

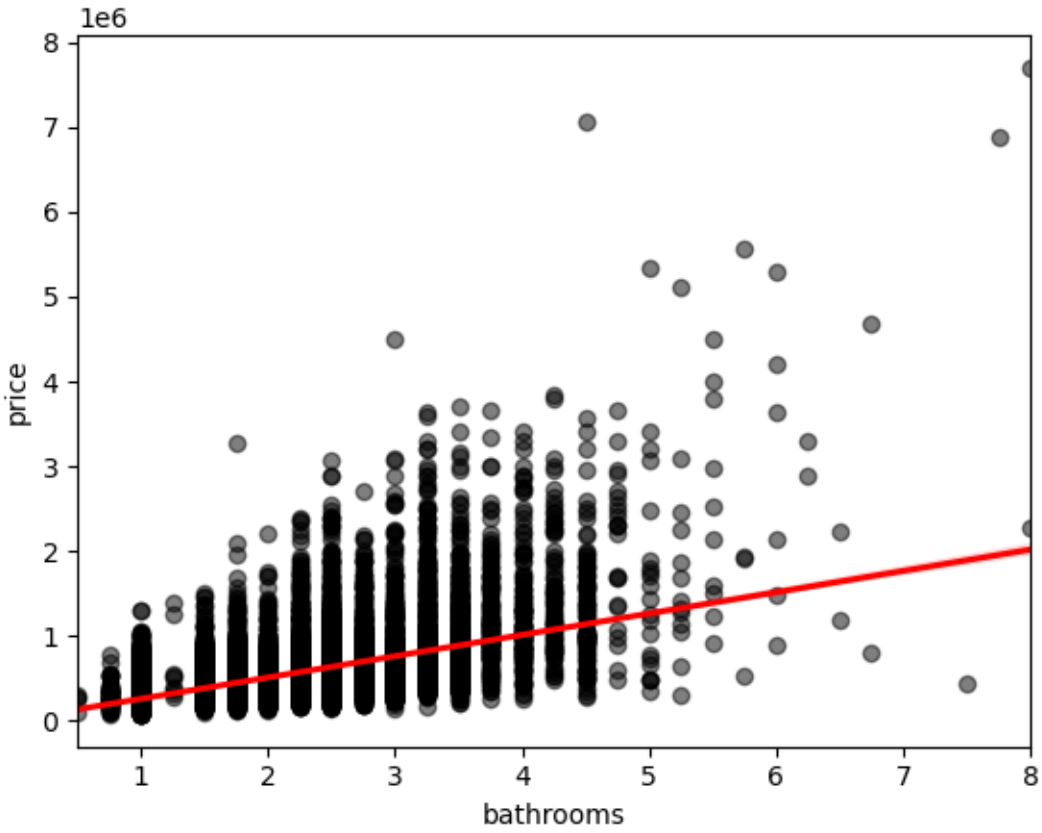


In this case, outliers are part of the distribution and removing them presents a false representation of the problem Based on the box plots, the price is what seems to have a lot of outliers, which is good since it gives a proper representation of the market.

7 7. Data Analysis

7.0.1 Analysis 1: What is the relationship between bathrooms and price?

[269]:
```
sns.regplot(x = data2['bathrooms'], y = data2['price'],
            scatter_kws = {"color": "black", "alpha": 0.5},
            line_kws = {"color": "red"});
```



Trend Observation:

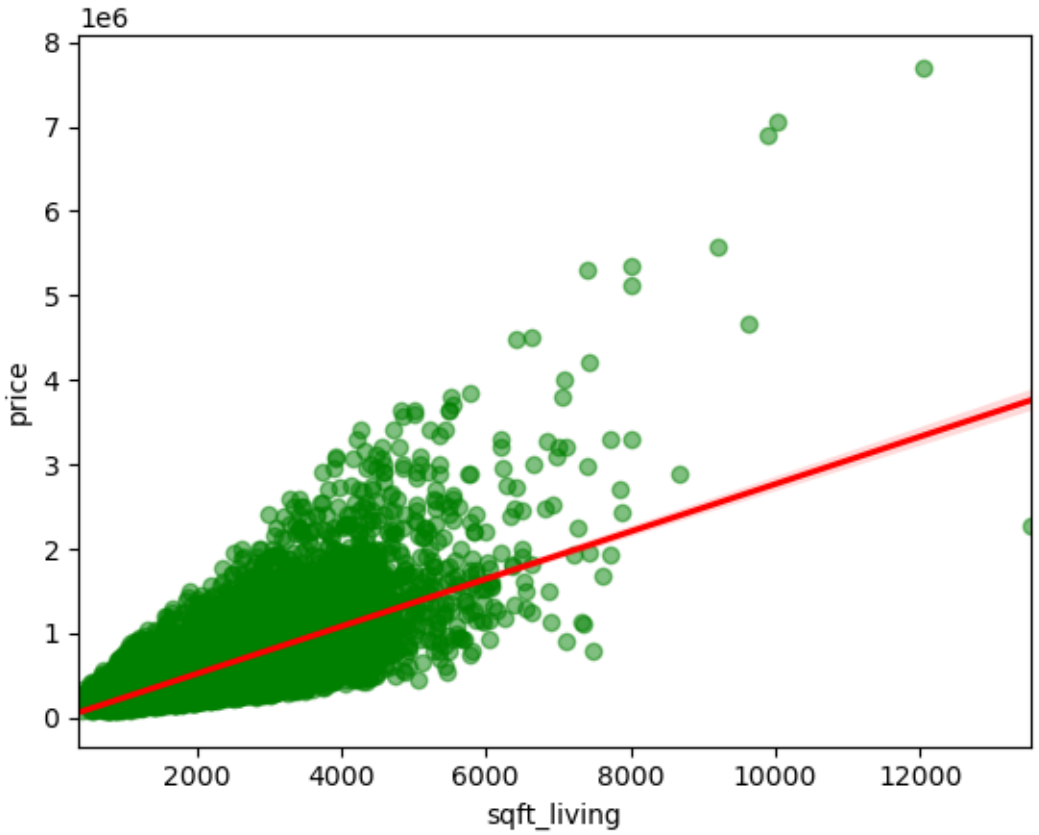
In the scatter plot with a regression line of bathrooms (number of bathrooms in house) against price, you can observe a positive trend. As the average number of bathrooms increases, the price of the house tends to increase as well.

Implication:

This suggests that houses with more bathrooms generally have higher prices in the King County housing market. They are priced higher.

7.0.2 Analysis 2: What is the Relationship between sqft_living and price?

[270]:
```
sns.regplot(x = data2['sqft_living'], y = data2['price'],
            scatter_kws = {"color": "green", "alpha": 0.5},
            line_kws = {"color": "red"});
```



Trend Observation:

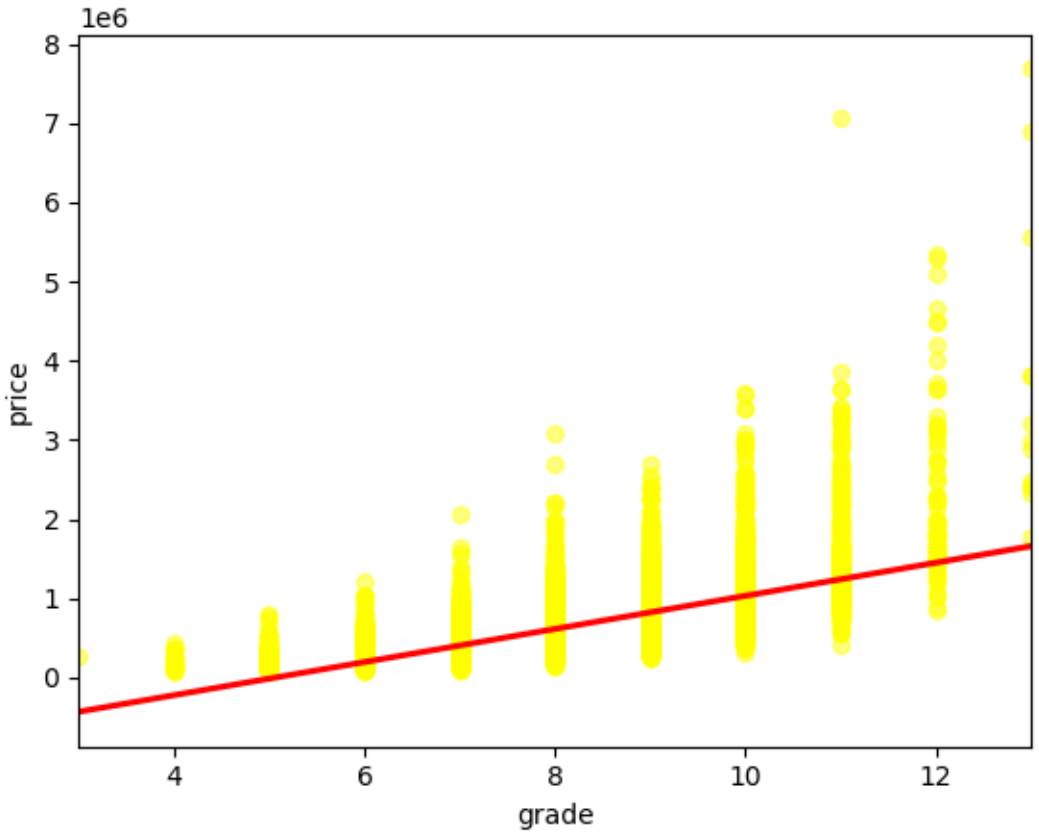
In the scatter plot with a regression line of square footage of the home (sqft_living) against price, you can observe a positive trend. As the average square footage of the home increases, the price of the house tends to increase as well.

Implication:

This suggests that houses with a higher square footage have higher prices in the King County housing market. They are priced higher.

7.0.3 Analysis 3: What is the relationship between Grade and Price

[271]:
```
sns.regplot(x = data2['grade'], y = data2['price'],
            scatter_kws = {"color": "yellow", "alpha": 0.5},
            line_kws = {"color": "red"});
```



Trend Observation:

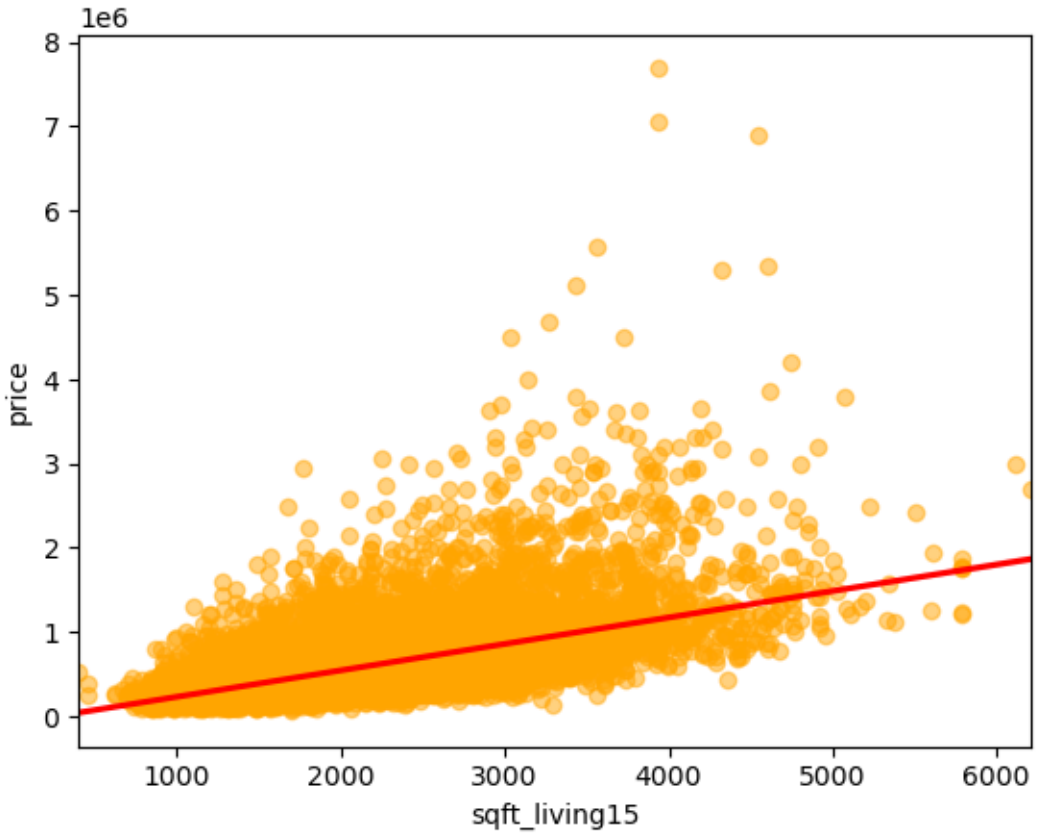
In the scatter plot with a regression line of grade (overall grade given to the housing unit, based on King County grading system) against price, you can observe a positive trend. As the grade given to the home increases, the price of the house tends to increase as well.

Implication:

This suggests that houses with a higher grade have higher prices in the King County housing market. They are priced higher.

7.0.4 Analysis 4: What is the relationship between sqft_living15 and Price?

[272]:
```
sns.regplot(x = data2['sqft_living15'], y = data2['price'],
            scatter_kws = {"color": "orange", "alpha": 0.5},
            line_kws = {"color": "red"});
```



Trend Observation:

In the scatter plot with a regression line of sqft_living15 (The square footage of interior housing living space for the nearest 15 neighbors) against price, you can observe a positive trend. As the average square footage of the homes within the neighborhood increases, the price of the house tends to increase as well.

Implication:

This suggests that houses within a neighborhood that has homes with a higher square footage have higher prices in the King County housing market. They are priced higher.

7.0.5 Analysis 5: Modeling

Feature Selection First create the first iteration of our linear model with all the selected features

[273]:
```
# Create a model to test using all the selected features
from statsmodels.formula.api import ols


formula = 'price ~ bathrooms + sqft_living + grade + sqft_above + sqft_living15'
model = ols(formula, data2).fit()
```

```
model_summary = model.summary()

model_summary
```

[273]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.544
Model:                            OLS   Adj. R-squared:                  0.544
Method:                 Least Squares   F-statistic:                     5161.
Date:                Tue, 16 Jul 2024   Prob (F-statistic):               0.00
Time:                        22:15:04   Log-Likelihood:            -2.9890e+05
No. Observations:               21597   AIC:                         5.978e+05
Df Residuals:                   21591   BIC:                         5.979e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept      -6.52e+05   1.36e+04    -48.079      0.000   -6.79e+05
-6.25e+05
bathrooms     -3.567e+04   3440.363    -10.369      0.000   -4.24e+04
-2.89e+04
sqft_living     245.5631      4.528     54.238      0.000     236.689
254.437
grade          1.119e+05   2470.776     45.293      0.000    1.07e+05
1.17e+05
sqft_above      -80.6961      4.458    -18.101      0.000     -89.434
-71.958
sqft_living15    22.2598      4.032      5.520      0.000      14.356
30.164
==============================================================================
Omnibus:                    17251.912   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1128620.760
Skew:                           3.361   Prob(JB):                         0.00
Kurtosis:                      37.771   Cond. No.                     2.96e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.96e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
"""
```

[274]:
```
# Check for multicollinearity
predictors_data2 = data2[['bathrooms', 'sqft_living', 'grade', 'sqft_above',␣
 ↪'sqft_living15']]
predictors_data2.corr()
```

[274]:

	bathrooms	sqft_living	grade	sqft_above	sqft_living15
bathrooms	1.000000	0.755758	0.665838	0.686668	0.569884
sqft_living	0.755758	1.000000	0.762779	0.876448	0.756402
grade	0.665838	0.762779	1.000000	0.756073	0.713867
sqft_above	0.686668	0.876448	0.756073	1.000000	0.731767
sqft_living15	0.569884	0.756402	0.713867	0.731767	1.000000

[275]:
```
abs(predictors_data2.corr()) > 0.75
```

[275]:

	bathrooms	sqft_living	grade	sqft_above	sqft_living15
bathrooms	True	True	False	False	False
sqft_living	True	True	True	True	True
grade	False	True	True	True	False
sqft_above	False	True	True	True	False
sqft_living15	False	True	False	False	True

Sqft_living seems to have very strong correlation with all the other predictors, so we will drop it and see the impact on our model

[276]:
```
# Create a second model after dropping sqft_living from the
from statsmodels.formula.api import ols


formula2_without_sqft_living = 'price ~ bathrooms + grade + sqft_above +␣
 ↪sqft_living15'
model2_without_sqft_living = ols(formula2_without_sqft_living, data2 ).fit()
model2_summary_without_sqft_living = model2_without_sqft_living.summary()

model2_summary_without_sqft_living
```

[276]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.482
Model:                            OLS   Adj. R-squared:                  0.482
Method:                 Least Squares   F-statistic:                     5030.
Date:                Tue, 16 Jul 2024   Prob (F-statistic):               0.00
Time:                        22:15:04   Log-Likelihood:            -3.0028e+05
No. Observations:               21597   AIC:                         6.006e+05
```

```
Df Residuals:                    21592   BIC:                          6.006e+05
Df Model:                            4
Covariance Type:            nonrobust
==============================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept     -7.819e+05   1.42e+04    -54.962      0.000    -8.1e+05
-7.54e+05
bathrooms      3.627e+04   3383.637     10.720      0.000    2.96e+04
4.29e+04
grade          1.274e+05   2615.981     48.705      0.000    1.22e+05
1.33e+05
sqft_above       58.8061      3.881     15.151      0.000      51.199
66.414
sqft_living15    82.8561      4.130     20.062      0.000      74.761
90.951
==============================================================================
Omnibus:                    19141.980   Durbin-Watson:                   1.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1752289.555
Skew:                           3.866   Prob(JB):                         0.00
Kurtosis:                      46.445   Cond. No.                     2.29e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.29e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

This actually lowers the R Squared value (0.544 versus 0.482), so dropping the sqft_living column does not improve the model

Let's drop the sqft_living15 column from the original model since it has a strong collinearity with multiple columns and see the impact that it has on the model

[277]:
```
# Create a second model after dropping sqft_living15 from the
from statsmodels.formula.api import ols


formula2_without_sqft_living15 = 'price ~ bathrooms + sqft_living + grade +␣
 ↪sqft_above'
model2_without_sqft_living15 = ols(formula2_without_sqft_living15, data2).fit()
model2_summary_without_sqft_living15 = model2_without_sqft_living15.summary()
```

```
model2_summary_without_sqft_living15
```

[277]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.544
Model:                            OLS   Adj. R-squared:                  0.544
Method:                 Least Squares   F-statistic:                     6434.
Date:                Tue, 16 Jul 2024   Prob (F-statistic):               0.00
Time:                        22:15:04   Log-Likelihood:            -2.9892e+05
No. Observations:               21597   AIC:                         5.978e+05
Df Residuals:                   21592   BIC:                         5.979e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -6.554e+05   1.36e+04    -48.353      0.000   -6.82e+05   -6.29e+05
bathrooms  -3.723e+04   3431.175    -10.849      0.000    -4.4e+04   -3.05e+04
sqft_living   252.4879      4.353     58.000      0.000     243.955     261.021
grade        1.159e+05   2366.154     48.968      0.000    1.11e+05    1.21e+05
sqft_above    -77.1922      4.416    -17.482      0.000     -85.847     -68.537
==============================================================================
Omnibus:                    17067.412   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1065708.366
Skew:                           3.319   Prob(JB):                         0.00
Kurtosis:                      36.767   Cond. No.                     2.43e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.43e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

The R squared value is the same as the first model, meaning that sqft_living15 did not actually contribute to the model. We can leave it as is. Next, let's try removing sqft_above from the model since it also has relationships with multiple columns

[278]:
```
# Create a second model after dropping sqft_living15 from the
from statsmodels.formula.api import ols


formula2_without_sqft_above = 'price ~ bathrooms + sqft_living + grade'
model2_without_sqft_above = ols(formula2_without_sqft_above, data2).fit()
```

```
model2_summary_without_sqft_above = model2_without_sqft_above.summary()

model2_summary_without_sqft_above
```

[278]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.537
Model:                            OLS   Adj. R-squared:                  0.537
Method:                 Least Squares   F-statistic:                     8359.
Date:                Tue, 16 Jul 2024   Prob (F-statistic):               0.00
Time:                        22:15:04   Log-Likelihood:            -2.9907e+05
No. Observations:               21597   AIC:                         5.981e+05
Df Residuals:                   21593   BIC:                         5.982e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -6.024e+05   1.33e+04    -45.279      0.000   -6.28e+05   -5.76e+05
bathrooms  -3.836e+04   3454.677    -11.103      0.000   -4.51e+04   -3.16e+04
sqft_living   203.1650      3.339     60.855      0.000     196.621     209.709
grade       1.046e+05   2293.086     45.626      0.000       1e+05    1.09e+05
==============================================================================
Omnibus:                    16926.041   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1008526.554
Skew:                           3.291   Prob(JB):                         0.00
Kurtosis:                      35.824   Cond. No.                     1.80e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.8e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

This also does not have a major impact on our relationship (0.544 versus 0.537 R squared values), so we can omit.

7.0.6 Feature Ranking With Recursive Elimination to validate the selected features for improving our model

[279]:
```
#from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
predictors_for_feature_ranking = data2[['bathrooms', 'sqft_living', 'grade',␣
 ↪'sqft_above', 'sqft_living15']]
selector = RFE(linreg, n_features_to_select=3)
selector = selector.fit(predictors_for_feature_ranking, data2['price'])

# Calling the .support_ attribute tells you which variables are selected

selector.support_

# Calling .ranking_ shows the ranking of the features, selected features are␣
 ↪assigned rank 1

selector.ranking_
```

[279]:
```
array([1, 1, 1, 2, 3])
```

This validates our selection of bathrooms, sft_living, and grade as the most important features since they were ranked as best with the ranking of 1. sqft_above and sqft_living have rankings of 2 and 3 respectively. So, this validates the fact that we discarded them.

7.0.7 Final Model- First Iteration

The selected features are: Bathrooms, sqft_living, and grade

[280]:
```
data2 = data2.drop(columns = ['sqft_above', 'sqft_living15'], axis = 1)
data2.head()
```

[280]:

	bathrooms	sqft_living	grade	price
0	1.00	1180	7	221900.0
1	2.25	2570	7	538000.0
2	1.00	770	6	180000.0
3	3.00	1960	7	604000.0
4	2.00	1680	8	510000.0

Let's see whether the selected columns are categorical and deal with them

[281]:
```
print("Number of unique values in bathrooms column:", data2['bathrooms'].
 ↪nunique())
print("Number of unique values in sqft_living column:", data2['sqft_living'].
 ↪nunique())
```

```
print("Number of unique values in grade column:", data2['grade'].nunique())
```

```
Number of unique values in bathrooms column: 29
Number of unique values in sqft_living column: 1034
Number of unique values in grade column: 11
```

Let's change the column data dtypes to string, to prevent them from being deemed as non-categorical, simply because they have numerical data types

[282]:
```
data['bathrooms'] = data['bathrooms'].astype(str)
data['grade'] = data['grade'].astype(str)
data['sqft_living15'] = data['sqft_living15'].astype(str)
```

[283]:
```
print(data2['bathrooms']is pd.CategoricalDtype)
print(data2['sqft_living']is pd.CategoricalDtype)
print(data2['grade']is pd.CategoricalDtype)
```

```
False
False
False
```

[284]:
```
# Get the list of all categorical columns
data2.select_dtypes(include=['object']).columns.tolist()
```

[284]:
```
[]
```

There are no categorical columns

[285]:
```
data2.select_dtypes(exclude=['object']).columns.tolist()
```

[285]:
```
['bathrooms', 'sqft_living', 'grade', 'price']
```

All the columns are numerical

[286]:
```
data2._get_numeric_data().head()
```

[286]:

	bathrooms	sqft_living	grade	price
0	1.00	1180	7	221900.0
1	2.25	2570	7	538000.0
2	1.00	770	6	180000.0
3	3.00	1960	7	604000.0
4	2.00	1680	8	510000.0

All the columns are numeric

Using all three methods above, we can see that none of the predictors are categorical, and we, therefore, do not have to deal with them

Let's convert them back to their original data types

[287]:
```
data['bathrooms'] = data['bathrooms'].astype(float)
data['grade'] = data['grade'].astype(int)
data['sqft_living15'] = data['sqft_living15'].astype('int')
```

[288]:
```
x = data2[['bathrooms', 'sqft_living', 'grade']]
x.head()
```

[288]:

	bathrooms	sqft_living	grade
0	1.00	1180	7
1	2.25	2570	7
2	1.00	770	6
3	3.00	1960	7
4	2.00	1680	8

[289]:
```
y = data2['price']
y.head()
```

[289]:
```
0    221900.0
1    538000.0
2    180000.0
3    604000.0
4    510000.0
Name: price, dtype: float64
```

7.0.8 Test Train Split

[290]:
```
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.
 ↪2,random_state=42)

# Scale the data
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)


# Modeling
model = LinearRegression()
# Training the model
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
data_after_test_training = pd.DataFrame({"true":y_test,"pred":y_pred})
data_after_test_training.head()
```

[290]:

	true	pred
3686	132500.0	165305.860046
10247	415000.0	289304.819885
4037	494000.0	407057.879160

```
3437   355000.0  303327.701085
19291  606000.0  394192.983226
```

7.0.9 Validation

[291]:
```
mse = mean_squared_error(y_test,y_pred)
mae = mean_absolute_error(y_test,y_pred)
r2  = r2_score(y_test,y_pred)

print("mse",mse)
print("mae",mae)
print("R2" ,r2)


y_hat_train = model.predict(x_train_scaled)
y_hat_test = model.predict(x_test_scaled)
train_residuals = y_hat_train - y_train
test_residuals = y_hat_test - y_test
train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error:', test_mse)
print('Difference between Train and Test MSE is:', (train_mse/test_mse-1).
 ↪round(3)*100,'%)')
```

```
mse 61048667496.09867
mae 161628.345898995
R2 0.5311748466249215
Train Mean Squared Error: 62794624746.37335
Test Mean Squared Error: 61048667496.09867
Difference between Train and Test MSE is: 2.9000000000000004 %)
```

The R squared value from the first iteration of our model shows that our model is accurate 53.11% of the time A mean absolute error of 161628 shows the predicted value is 161628 points away from the actual value. Our training and test MSE exhibit a very small difference (2.9%). This is a sign that the model generalizes well to future cases.

These are not ideal conditions for the model we want to eventually have, let's make some iterations.

7.0.10 Final Model- Second Iteration

Let us change the ratio of our test-train data (from 80-20 to 70-30) and see whether this improves our model

[292]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪30,random_state=42)

# Scale the data
scaler = StandardScaler()
```

```
x_train_scaled = scaler.fit_transform(x_train)
#IMPORTANT NOTES!!!!!!!!!
# train data with fit transform
# test data with test data
# We only scale our predictors, we don't scale the predicted value
x_test_scaled = scaler.transform(x_test)
# Modeling
model = LinearRegression()
# Training the model
model.fit(x_train_scaled,y_train)
y_pred = model.predict(x_test_scaled)
data_after_test_training = pd.DataFrame({"true":y_test,"pred":y_pred})
data_after_test_training.head()
```

[292]:

	true	pred
3686	132500.0	164746.383818
10247	415000.0	286457.299637
4037	494000.0	407895.728169
3437	355000.0	304803.668107
19291	606000.0	392380.599492

7.0.11 Validation

[293]:
```
mse = mean_squared_error(y_test,y_pred)
mae = mean_absolute_error(y_test,y_pred)
r2  = r2_score(y_test,y_pred)

print("mse",mse)
print("mae",mae)
print("R2" ,r2)

y_hat_train = model.predict(x_train_scaled)
y_hat_test = model.predict(x_test_scaled)

train_residuals = y_hat_train - y_train
test_residuals = y_hat_test - y_test

train_mse = mean_squared_error(y_train, y_hat_train)
test_mse = mean_squared_error(y_test, y_hat_test)
print('Train Mean Squared Error:', train_mse)
print('Test Mean Squared Error:', test_mse)
print('Difference between Train and Test MSE is:', (train_mse/test_mse-1).
 ↪round(3)*100,'%)')
```

```
mse 62009541655.03987
mae 161609.34538598044
R2 0.5323509874463058
```

```
Train Mean Squared Error: 62642802802.33004
Test Mean Squared Error: 62009541655.03987
Difference between Train and Test MSE is: 1.0 %)
```

The R squared value from the second iteration of our model shows that our model is accurate 53.23% of the time based on the test data. A mean absolute error of 161609 shows the predicted value is 161609 points away from the actual value, based on the test data. However, the difference between the train and test mean squared error is now very little (1%). This indicates that we are not overfitting. This is a large improvement on our model and we will take it.

Now let's get our model's summary to get values like the p-values of the predictors, f-statistic and its probability, R squared, and coefficients..

7.0.12 Model Summary

[294]:
```
import statsmodels.api as sm

# Add a constant to the model (for the intercept)
x_train_sm = sm.add_constant(x_train)

# Fit the model using statsmodels
model_sm = sm.OLS(y_train, x_train_sm).fit()

# Print the model summary
print(model_sm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.539
Model:                            OLS   Adj. R-squared:                  0.539
Method:                 Least Squares   F-statistic:                     5895.
Date:                Tue, 16 Jul 2024   Prob (F-statistic):               0.00
Time:                        22:15:05   Log-Likelihood:            -2.0936e+05
No. Observations:               15117   AIC:                         4.187e+05
Df Residuals:                   15113   BIC:                         4.188e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -6.251e+05    1.6e+04    -39.181      0.000   -6.56e+05   -5.94e+05
bathrooms   -3.33e+04   4166.666     -7.991      0.000   -4.15e+04   -2.51e+04
sqft_living  197.0231      3.997     49.295      0.000     189.189     204.857
grade       1.079e+05   2746.353     39.286      0.000    1.03e+05    1.13e+05
==============================================================================
Omnibus:                    11024.571   Durbin-Watson:                   1.987
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           469683.221
Skew:                           3.044   Prob(JB):                         0.00
Kurtosis:                      29.620   Cond. No.                     1.81e+04
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.81e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Model Summary Interpretation R-squared: 0.539 This means that approximately 53.9% of the variability in the house prices can be explained by the model.

Adjusted R-squared: We will create another model less one identified predictor to see how this impacts the model and whether it performs better. If the model performs better, we need to drop this variable or decide that it is not one of the most important ones.

F-statistic and its p-value: F-statistic: 5895, Prob (F-statistic): 0.00 The very low p-value suggests that the overall model is statistically significant, meaning that at least one of the predictors is significantly related to the dependent variable (house prices).

Coefficients:

Each row represents a predictor in the model with its coefficient, standard error, t-value, and p-value. For instance, the coefficient for sqft_living is 197.0231, meaning that for each additional square foot of living space, the house price increases by approximately $197.0231, holding other variables constant. The p-values for all predictors are less than 0.05, indicating that they are statistically significant.

Let's use feature ranking with recursive elimination to select only two variables for our model.

[295]:
```
#from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
predictors_for_feature_ranking2 = data2[['bathrooms', 'sqft_living', 'grade']]
selector2 = RFE(linreg, n_features_to_select=2)
selector2 = selector.fit(predictors_for_feature_ranking2, data2['price'])

# Calling the .support_ attribute tells you which variables are selected

selector.support_

# Calling .ranking_ shows the ranking of the features, selected features are␣
 ↪assigned rank 1

selector.ranking_
```

[295]:
```
array([1, 1, 1])
```

All the predictors still have the same ranking of 1

[296]:
```
import pandas as pd
import statsmodels.api as sm

def stepwise_selection(x, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out=0.05,
                       verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
        x - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    See https://en.wikipedia.org/wiki/Stepwise_regression for the details
    """
    included = list(initial_list)
    while True:
        changed = False
        # forward step
        excluded = list(set(x.columns) - set(included))
        new_pval = pd.Series(index=excluded, dtype=float)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(x[included +␣
↪[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed = True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature,␣
↪best_pval))

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(x[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max()  # null if pvalues is empty
        if worst_pval > threshold_out:
            changed = True
```

```
            worst_feature = pvalues.idxmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature,␣
 ↪worst_pval))
        if not changed:
            break
    return included
    return included

result = stepwise_selection(x, y, verbose=True);
print('resulting features:')
print(result)
```

```
Add  bathrooms                  with p-value 0.0
Add  grade                      with p-value 0.0
Add  sqft_living                with p-value 0.0
resulting features:
['bathrooms', 'grade', 'sqft_living']
```

All the selected predictors have been retained

8 8. Regression Results:

The features with strong relationships to sale prices are: – Number of bathrooms (`bathrooms`) – Sqft_living (`sqft_living`) – Grade (`grade`)

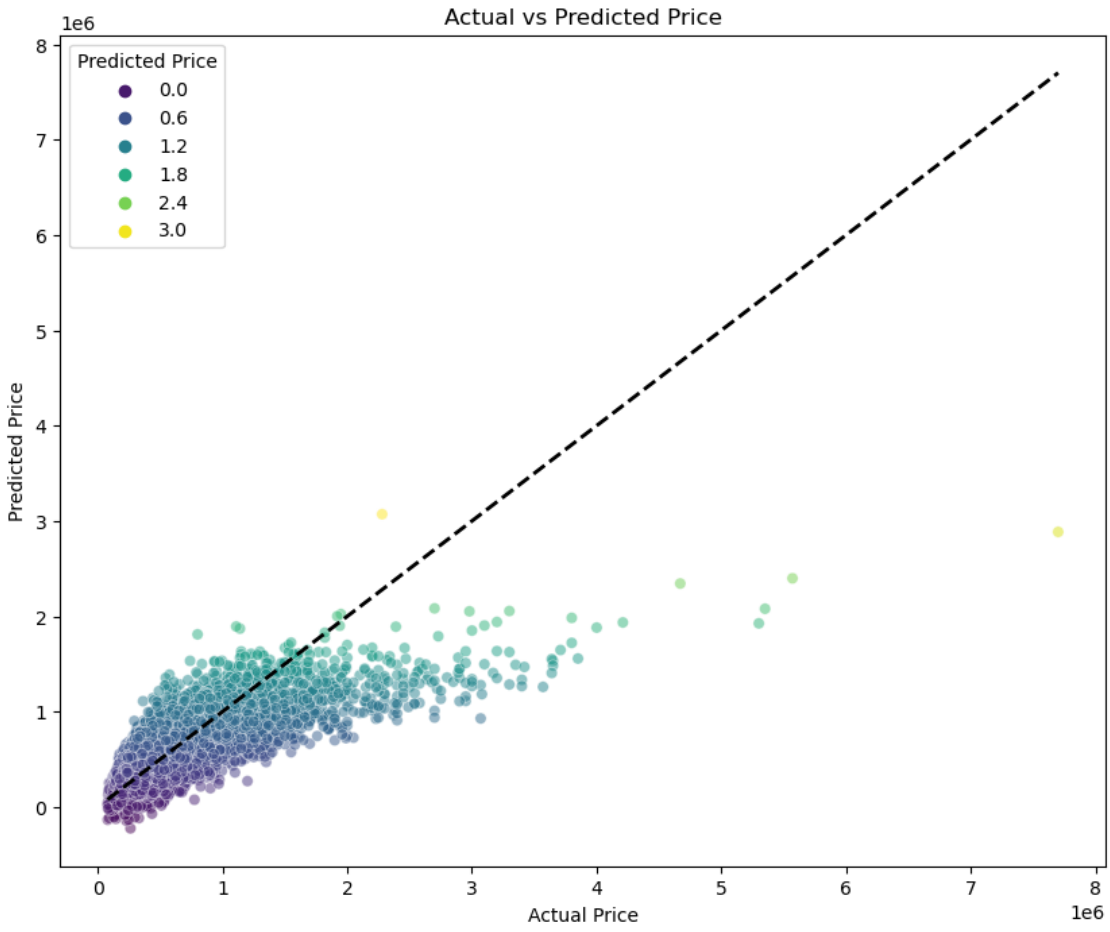
Their coeficients are: – `bathrooms`: -33,304. For each additional bathroom, the house price decreases by approximately $33,304, holding other variables constant.

– `sqft_living`: $197,023. For each additional square foot of living space, the house price increases by approximately $197.0231, holding other variables constant.

– `grade`: $107,900. For each increase in grade, the house price increases by approximately $107,900, holding other variables constant.

[297]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 8))

# Create the scatter plot
scatter = sns.scatterplot(x=y_train, y=y_hat_train, alpha=0.5, hue=y_hat_train,␣
 ↪palette='viridis')

# Add the line for y=x
plt.plot([data2['price'].min(), data2['price'].max()],
         [data2['price'].min(), data2['price'].max()], 'k--', lw=2)
```

```
# Add labels and title
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Price')

# Add legend
plt.legend(title='Predicted Price', loc='upper left')

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



9 9. Conclusion

9.1 Significant Predictors of House Prices:

9.1.1 Objective 1: Explore the Relationship Between Number of bathrooms and its price

Number of Bathrooms (bathrooms): Contrary to initial expectations, the coefficient suggests that each additional bathroom is associated with a decrease in house prices by $-33,300 , holding other variables constant. This unexpected finding may warrant further investigation into potential factors influencing this relationship. ### Objective 2: Explore the relationship between the square footage of the home and its price

Square Footage of Living Space (sqft_living): The coefficient suggests that for each additional square foot of living space, house prices are estimated to increase by $197.0231, holding other variables constant. This indicates that larger houses tend to command higher prices. ### Objective 3: Explore the impact of the King County housing grading system on the pricing of homes in the market.

Grade of the House (grade): A higher grade is associated with higher house prices, with each unit increase in grade estimated to increase house prices by $107,900, holding other variables constant. Grades typically reflect the quality and features of the house, influencing buyer preferences and willingness to pay.

9.1.2 Objective 4: Explore the impact of a home's neighborhood on its price

sq_ft_living15(The square footage of interior housing living space for the nearest 15 neighbors): A higher sqft_living15 is associated with higher house prices. With a strong linear relationship of 0.59 with price, this feature can be important for the Real Estate Development Company when picking the best neighborhoods for their houses. It is, however, worth noting that it was not selected for the model since through feature ranking with recursive elimination, it received a ranking of 3.

9.1.3 Objective 5: Develop a Linear Regression Model to Predict Housing Prices

Model Fit and Statistical Significance:

The overall model has an R-squared value of 0.539, indicating that approximately 53.9% of the variance in house prices can be explained by the predictors (sqft_living, grade, bathrooms) included in the model. This suggests that while these predictors are significant, there are other factors not captured by the model that also influence house prices.

The F-statistic is very high (5895) with a low p-value (0.00), indicating that the model is statistically significant. This implies that at least one of the predictors is significantly related to house prices, reinforcing the reliability of the model's predictions.

10 10. Recommendations

Business Strategy Implications Given these insights, the Real Estate Development Company should consider the following strategies:

10.0.1 Target Larger Houses:

- Since the model indicates that house price increases with the square footage of living space, the company should focus on acquiring or developing larger houses to maximize potential sales prices.

10.0.2 Focus on High-Grade Houses:

- Houses with higher grades are significantly associated with higher prices. The company should aim to develop or invest in properties with higher quality finishes, better construction, and more desirable features.

10.0.3 Investigate Bathroom Anomaly:

- The negative coefficient for bathrooms is unusual and suggests a deeper investigation is needed. The company should look into whether this result is due to multicollinearity or other factors not captured in the model. It may be beneficial to re-examine the quality or location of properties with many bathrooms.

10.0.4 Consider Additional Predictors:

- Since the R-squared value is 0.537, there are other factors influencing house prices that are not included in this model. The company should consider incorporating additional relevant variables such as location, lot size, view, and age of the house to improve the model's predictive power.

10.0.5 Use Predictive Analytics for Targeted Marketing:

- By using the model to predict high-priced houses, the company can create targeted marketing strategies to attract high-end buyers. This can involve highlighting the features that are most strongly associated with higher prices, such as larger living spaces and higher grades.

10.0.6 Price Optimization:

- The company can use the model's predictions to set competitive yet profitable prices for their properties, ensuring that they are priced appropriately based on their characteristics.