kc_house_sales

October 4, 2022

0.1 Final Project Submission

Please fill out: * Student name: Emmanuel Kiplimo * Student pace: Full Time * Scheduled project review date/time: 3/10/2022 * Instructor name: William Okomba

0.2 Overview

A real estate agency in King County, Washington State is looking to explore sales and details on houses in the region to identify various features and factors that significantly contribute to the price of houses. As a Data Scientist, I'll analyze and model the data provided to draw insights and predictions on house prices.

The following queries will shed some light on house sales and features of an ideal house and a serve as build up for a predictive model:

1. Notable features that contribute to house prices.
2. Is there an observable trend of house sales across the year

0.2.1 Data understanding

[1]:
```
# Import standard packages
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

```
/home/kiplimo/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:7: FutureWarning: pandas.Int64Index
is deprecated and will be removed from pandas in a future version. Use
pandas.Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
/home/kiplimo/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:7: FutureWarning: pandas.Float64Index
```

```
is deprecated and will be removed from pandas in a future version. Use
pandas.Index with the appropriate dtype instead.
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

[2]:
```python
# Loading data onto a dataframe
df = pd.read_csv('data/kc_house_data.csv')
df.head()
```

[2]:
```
           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
4  1954400510   2/18/2015  510000.0         3       2.00         1680

   sqft_lot  floors waterfront  view  …          grade sqft_above  \
0      5650     1.0        NaN  NONE  …      7 Average       1180
1      7242     2.0         NO  NONE  …      7 Average       2170
2     10000     1.0         NO  NONE  …  6 Low Average        770
3      5000     1.0         NO  NONE  …      7 Average       1050
4      8080     1.0         NO  NONE  …         8 Good       1680

   sqft_basement yr_built  yr_renovated  zipcode      lat     long  \
0            0.0     1955           0.0    98178  47.5112 -122.257
1          400.0     1951        1991.0    98125  47.7210 -122.319
2            0.0     1933           NaN    98028  47.7379 -122.233
3          910.0     1965           0.0    98136  47.5208 -122.393
4            0.0     1987           0.0    98074  47.6168 -122.045

   sqft_living15  sqft_lot15
0           1340        5650
1           1690        7639
2           2720        8062
3           1360        5000
4           1800        7503

[5 rows x 21 columns]
```

[3]:
```python
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
 14  yr_built       21597 non-null  int64
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64
 20  sqft_lot15     21597 non-null  int64
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

From the above summary: * The data set has 21597 entries * object type columns have to be explored further * date
* waterfront * view
* condition * grade * sqft_basement

0.3 Data preparation

[4]:
```
df['date']
```

[4]:
```
0        10/13/2014
1         12/9/2014
2         2/25/2015
3         12/9/2014
4         2/18/2015
            …
21592     5/21/2014
21593     2/23/2015
21594     6/23/2014
21595     1/16/2015
21596    10/15/2014
Name: date, Length: 21597, dtype: object
```

[5]:
```
# Change date column from object to the appropriate data type (datetime)
# We may later need to re-engineer the date column to derive seasons
df['date'] = pd.to_datetime(df['date'])
```

[6]:
```python
print(df['waterfront'].unique())
df['waterfront'].describe()
```

```
[nan 'NO' 'YES']
```

[6]:
```
count     19221
unique        2
top          NO
freq      19075
Name: waterfront, dtype: object
```

Waterfront column seems to be a binary column. Having 1 and 0 to represent 'YES' and 'NO' respectively will make analysis much simpler. It's also safe to replace the missing values with the mode; The house not having a view of the waterfront

[7]:
```python
def waterfront(df):
    df['waterfront'] = df['waterfront'].map(lambda x: 1 if x == 'YES' else 0)
    return df

df = waterfront(df)
df['waterfront'].unique()
```

[7]:
```
array([0, 1])
```

[8]:
```python
print(df['view'].unique())
```

```
['NONE' nan 'GOOD' 'EXCELLENT' 'AVERAGE' 'FAIR']
```

[9]:
```python
# The null values could be filled with 'NONE'
print(df['view'].isna().value_counts())
```

```
False    21534
True        63
Name: view, dtype: int64
```

[10]:
```python
print(df['view'].value_counts())
```

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

Using a quantitative variable to represent quality of view from the house would be more effective in EDA

[11]:
```python
#Function to handle null values and change the view column into int based column
def view(df):
```

```
    df['view'].fillna(1,inplace = True)
    old = ['NONE','AVERAGE','FAIR','GOOD','EXCELLENT']
    new = [1,2,3,4,5]

    df['view'].replace(old,new,inplace=True)
    return df

df = view(df)
df['view'].unique()
```

[11]:
```
array([1, 4, 5, 2, 3])
```

[12]:
```
df['grade'].unique()
```

[12]:
```
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)
```

[13]:
```
def grade(df):
    old = ['3 Poor','4 Low','5 Fair','6 Low Average','7 Average','8 Good','9␣
 ↪Better','10 Very Good','11 Excellent','12 Luxury','13 Mansion']
    new = [3,4,5,6,7,8,9,10,11,12,13]

    df['grade'].replace(old,new,inplace=True)
    return df
df = grade(df)
df['grade'].unique()
```

[13]:
```
array([ 7,  6,  8, 11,  9,  5, 10, 12,  4,  3, 13])
```

[14]:
```
df['bathrooms'].unique()
```

[14]:
```
array([1.  , 2.25, 3.  , 2.  , 4.5 , 1.5 , 2.5 , 1.75, 2.75, 3.25, 4.  ,
       3.5 , 0.75, 4.75, 5.  , 4.25, 3.75, 1.25, 5.25, 6.  , 0.5 , 5.5 ,
       6.75, 5.75, 8.  , 7.5 , 7.75, 6.25, 6.5 ])
```

- The floating point values in the bathroom column could be indicating the amenities in the bathroom. A whole number could be representing a bathroom with all features, i.e `shower`, `toilet` and a `sink`
- Working with integer values would be more suitable.

[15]:
```
# A function to have bathroom columns as int
def bathrooms(df):
    df['bathrooms'] = df['bathrooms'].map(lambda x: int(round(x,0)))
    return df
df = bathrooms(df)
df['bathrooms'].unique()
```

[15]: array([1, 2, 3, 4, 5, 6, 0, 7, 8])

[16]:
```
df['sqft_basement'].unique()
```

[16]:
```
array(['0.0', '400.0', '910.0', '1530.0', '?', '730.0', '1700.0', '300.0',
       '970.0', '760.0', '720.0', '700.0', '820.0', '780.0', '790.0',
       '330.0', '1620.0', '360.0', '588.0', '1510.0', '410.0', '990.0',
       '600.0', '560.0', '550.0', '1000.0', '1600.0', '500.0', '1040.0',
       '880.0', '1010.0', '240.0', '265.0', '290.0', '800.0', '540.0',
       '710.0', '840.0', '380.0', '770.0', '480.0', '570.0', '1490.0',
       '620.0', '1250.0', '1270.0', '120.0', '650.0', '180.0', '1130.0',
       '450.0', '1640.0', '1460.0', '1020.0', '1030.0', '750.0', '640.0',
       '1070.0', '490.0', '1310.0', '630.0', '2000.0', '390.0', '430.0',
       '850.0', '210.0', '1430.0', '1950.0', '440.0', '220.0', '1160.0',
       '860.0', '580.0', '2060.0', '1820.0', '1180.0', '200.0', '1150.0',
       '1200.0', '680.0', '530.0', '1450.0', '1170.0', '1080.0', '960.0',
       '280.0', '870.0', '1100.0', '460.0', '1400.0', '660.0', '1220.0',
       '900.0', '420.0', '1580.0', '1380.0', '475.0', '690.0', '270.0',
       '350.0', '935.0', '1370.0', '980.0', '1470.0', '160.0', '950.0',
       '50.0', '740.0', '1780.0', '1900.0', '340.0', '470.0', '370.0',
       '140.0', '1760.0', '130.0', '520.0', '890.0', '1110.0', '150.0',
       '1720.0', '810.0', '190.0', '1290.0', '670.0', '1800.0', '1120.0',
       '1810.0', '60.0', '1050.0', '940.0', '310.0', '930.0', '1390.0',
       '610.0', '1830.0', '1300.0', '510.0', '1330.0', '1590.0', '920.0',
       '1320.0', '1420.0', '1240.0', '1960.0', '1560.0', '2020.0',
       '1190.0', '2110.0', '1280.0', '250.0', '2390.0', '1230.0', '170.0',
       '830.0', '1260.0', '1410.0', '1340.0', '590.0', '1500.0', '1140.0',
       '260.0', '100.0', '320.0', '1480.0', '1060.0', '1284.0', '1670.0',
       '1350.0', '2570.0', '1090.0', '110.0', '2500.0', '90.0', '1940.0',
       '1550.0', '2350.0', '2490.0', '1481.0', '1360.0', '1135.0',
       '1520.0', '1850.0', '1660.0', '2130.0', '2600.0', '1690.0',
       '243.0', '1210.0', '1024.0', '1798.0', '1610.0', '1440.0',
       '1570.0', '1650.0', '704.0', '1910.0', '1630.0', '2360.0',
       '1852.0', '2090.0', '2400.0', '1790.0', '2150.0', '230.0', '70.0',
       '1680.0', '2100.0', '3000.0', '1870.0', '1710.0', '2030.0',
       '875.0', '1540.0', '2850.0', '2170.0', '506.0', '906.0', '145.0',
       '2040.0', '784.0', '1750.0', '374.0', '518.0', '2720.0', '2730.0',
       '1840.0', '3480.0', '2160.0', '1920.0', '2330.0', '1860.0',
       '2050.0', '4820.0', '1913.0', '80.0', '2010.0', '3260.0', '2200.0',
       '415.0', '1730.0', '652.0', '2196.0', '1930.0', '515.0', '40.0',
       '2080.0', '2580.0', '1548.0', '1740.0', '235.0', '861.0', '1890.0',
       '2220.0', '792.0', '2070.0', '4130.0', '2250.0', '2240.0',
       '1990.0', '768.0', '2550.0', '435.0', '1008.0', '2300.0', '2610.0',
       '666.0', '3500.0', '172.0', '1816.0', '2190.0', '1245.0', '1525.0',
       '1880.0', '862.0', '946.0', '1281.0', '414.0', '2180.0', '276.0',
       '1248.0', '602.0', '516.0', '176.0', '225.0', '1275.0', '266.0',
       '283.0', '65.0', '2310.0', '10.0', '1770.0', '2120.0', '295.0',
```

```
       '207.0', '915.0', '556.0', '417.0', '143.0', '508.0', '2810.0',
       '20.0', '274.0', '248.0'], dtype=object)
```

[17]:
```
df['sqft_basement'].value_counts()
```

[17]:
```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
          …
1920.0        1
3480.0        1
2730.0        1
2720.0        1
248.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

[18]:
```
df['sqft_basement'].describe()
```

[18]:
```
count     21597
unique      304
top         0.0
freq      12826
Name: sqft_basement, dtype: object
```

- From the summary statistics it seems many houses don't have a basement hence 0 sqft. The '?' is a place holder for 454 houses in which the basement status is not indicated
- Having the basement status as a binary option could provide better insights as compared to the size.

[19]:
```
def basement(df):
    df['sqft_basement'] = df['sqft_basement'].map(lambda x : float(x.replace('?
 ↪', '0')))
    df['basement'] = df['sqft_basement'].map(lambda x: 1 if x > 0 else 0)
    df['sqft_basement'] = df['basement']

    return df

df = basement(df)
df['basement'].unique()
```

[19]:
```
array([0, 1])
```

[20]:
```
# Let's check for duplicates and remove them if any
df[df.duplicated()==True].shape
```

[20]: (0, 22)

[21]:
```
# Determinig missing values
df.isnull().sum()
```

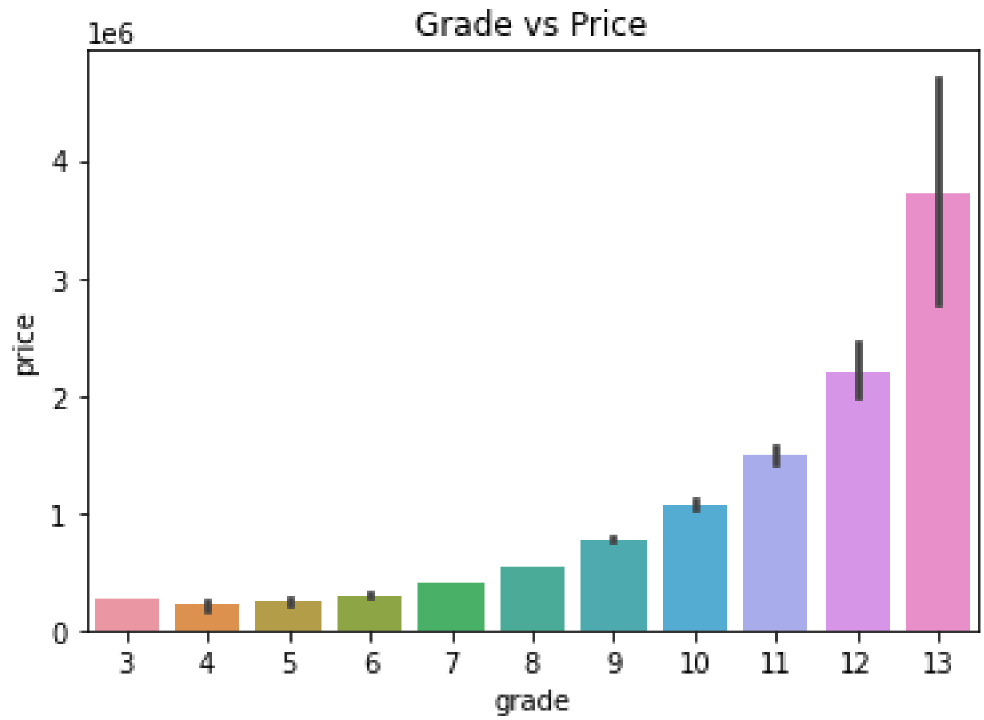
[21]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront          0
view                0
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
basement            0
dtype: int64
```

- There seems to be no duplicated rows across the 22 colums
- Most of the null values were handled in the initial feature engineering apart from `yr_renovated`

[22]:
```
df['yr_renovated'].dropna(inplace=True)
```

0.3.1 Data Analysis

[23]:
```
sns.barplot(x='grade',y='price',data=df)
plt.title("Grade vs Price")
plt.show()
```



- The grade of the house refers to the general quality of the house with 3 being poor quality and 13 being more than luxurious.
- Renovations are likely to improve the grade and genreal condition of the house hence contributing to the overall price.

[24]:
```
df['bedrooms'].value_counts()
```

[24]:
```
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

[25]:
```
df[df['bedrooms'] > 10 ]
```

[25]:

```
                id        date     price  bedrooms  bathrooms  sqft_living  \
8748   1773100755 2014-08-21  520000.0        11          3         3000
15856  2402100895 2014-06-25  640000.0        33          2         1620

       sqft_lot  floors  waterfront  view  … sqft_above  sqft_basement  \
8748       4960     2.0           0     1  …       2400              1
15856      6000     1.0           0     1  …       1040              1

       yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
8748       1918        1999.0    98106  47.5560 -122.363           1420
15856      1947           0.0    98103  47.6878 -122.331           1330

       sqft_lot15  basement
8748         4960         1
15856        4700         1

[2 rows x 22 columns]
```

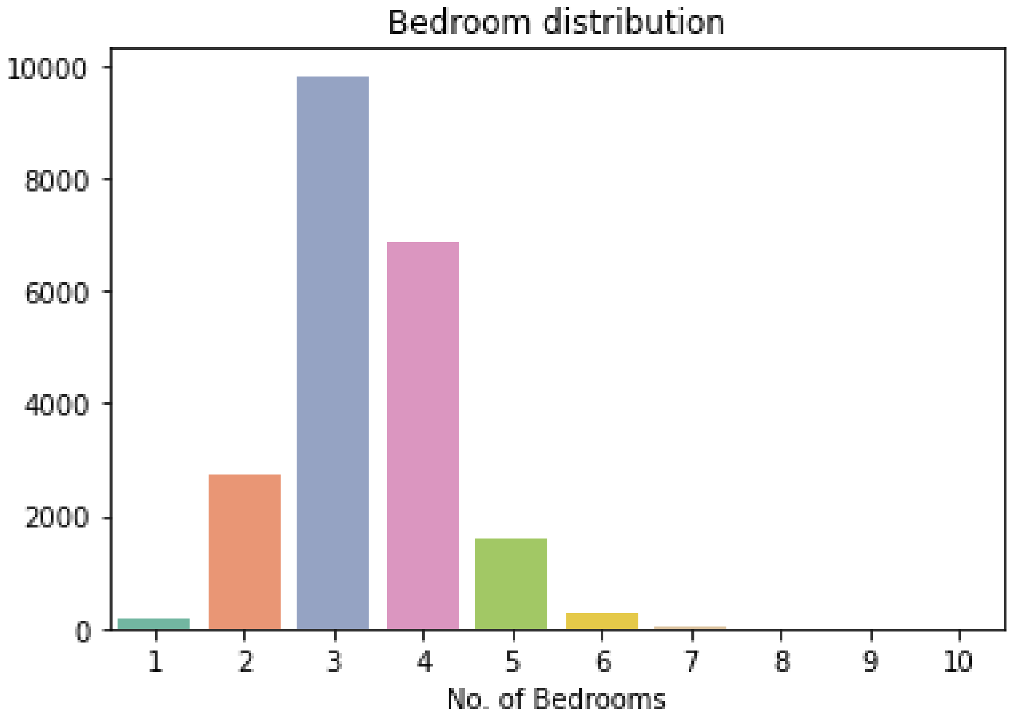
The two houses with 11 and 33 bedrooms have a significantly lower ratio of bathrooms. It's highly likely to be an anomaly and should therefore be dropped.

[26]:

```python
outliers = df[df['bedrooms'] > 10 ].index
df.drop(outliers, inplace=True)
```

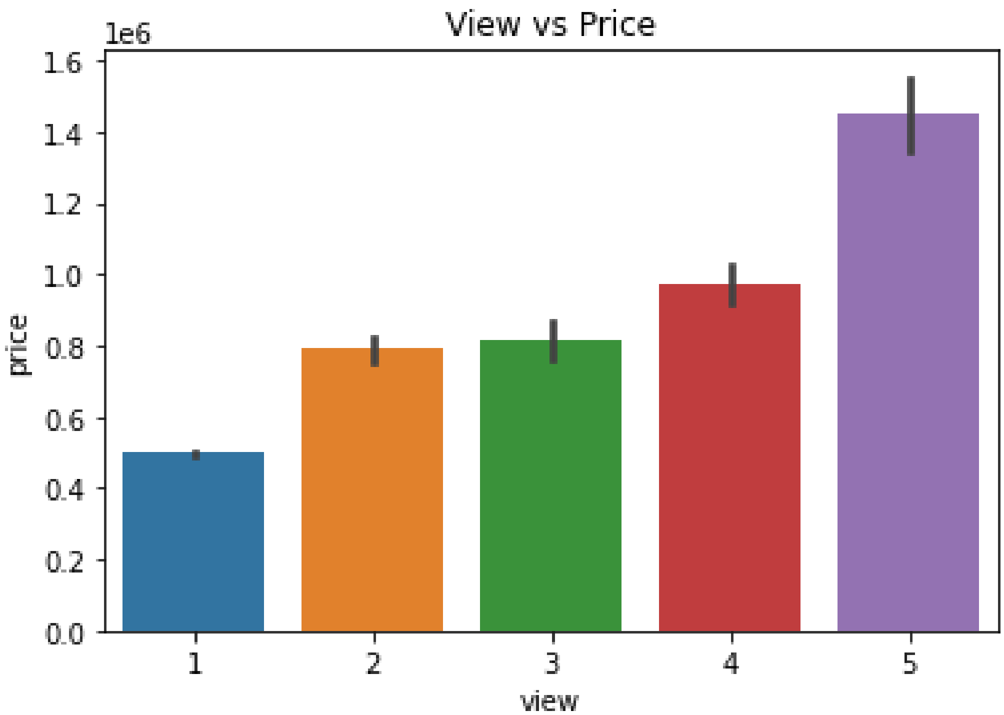
[27]:

```python
# Bedroom distribution
counts = df["bedrooms"].value_counts()
sns.barplot(x=counts.index,y=counts.values,palette=("Set2"))
plt.xlabel("No. of Bedrooms")
plt.title("Bedroom distribution")
plt.show()
```

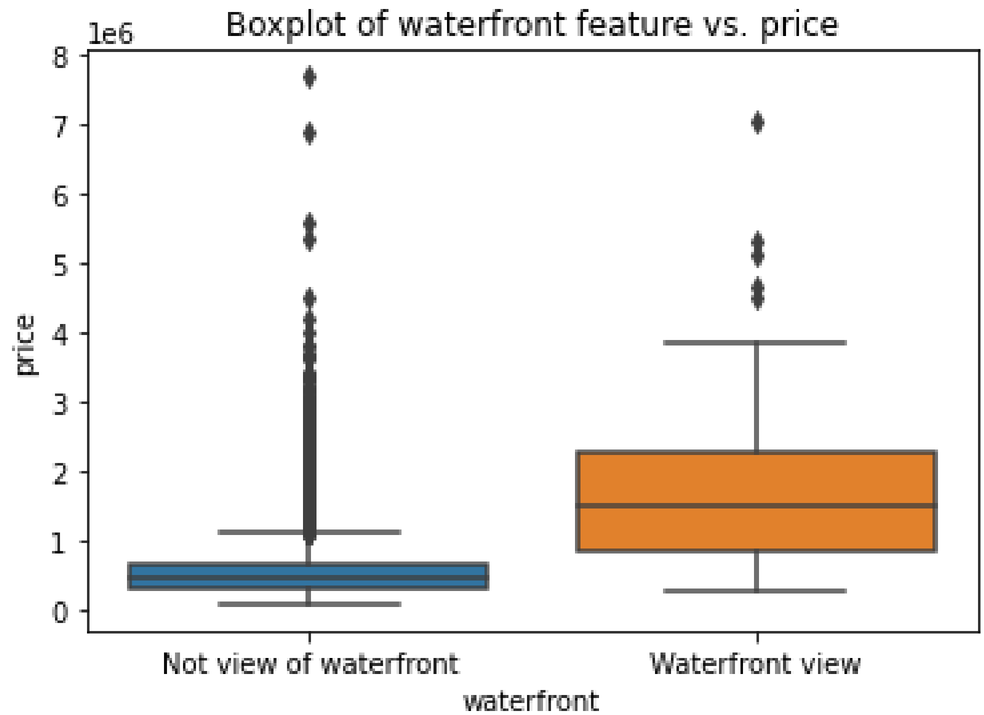


A typical household has 3 bedrooms followed by 4 bedroom and 3 bedroom houses.

[28]:
```
sns.barplot(x='view',y='price',data=df)
plt.title('View vs Price');
```



[29]:
```
sns.boxplot(x = df['waterfront'], y = df['price'])
plt.xticks(np.arange(2), ('Not view of waterfront', 'Waterfront view'))
plt.title("Boxplot of waterfront feature vs. price")
plt.show()
```



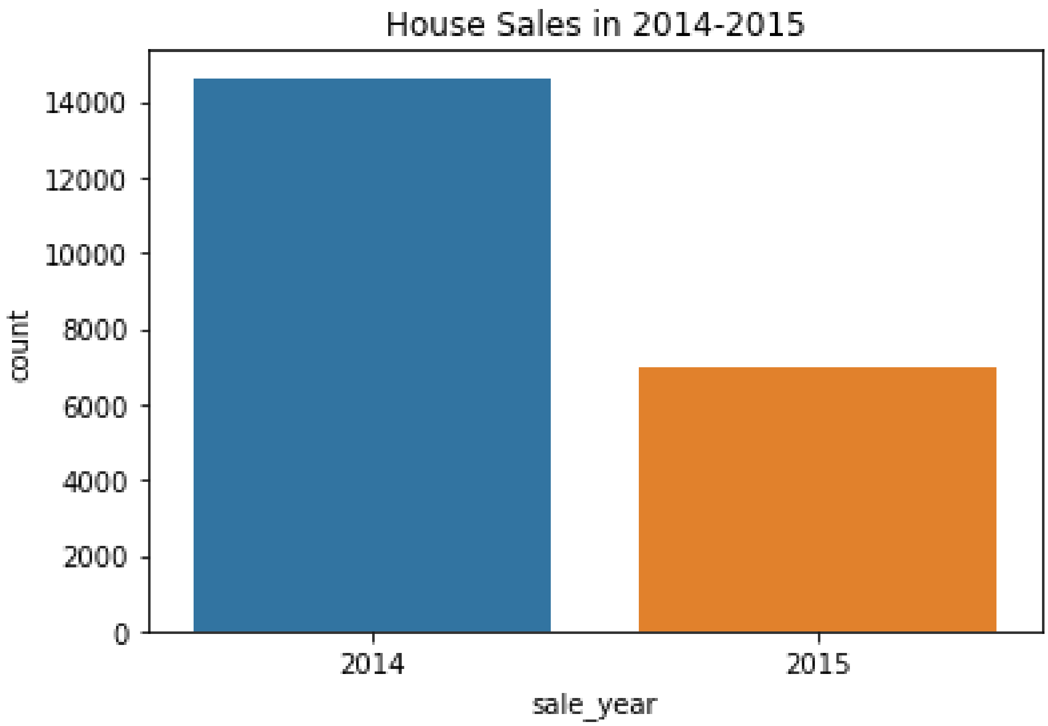
[30]:
```
waterfront = df[df['waterfront'] == 1]['price'].mean()
non_waterfront = df[df['waterfront'] == 0]['price'].mean()
print(f"A house with a waterfront has an average price of USD␣
↪{round(waterfront,2)}")
print(f"Houses without a waterfront have an average price of USD␣
↪{round(non_waterfront,2)}")
```

```
A house with a waterfront has an average price of USD 1717214.73
Houses without a waterfront have an average price of USD 532281.77
```

[31]:
```
df['sale_year'] = df.apply(lambda x: x.date.year, axis=1)
df['sale_year'].value_counts()
```

[31]:
```
2014    14620
2015     6975
Name: sale_year, dtype: int64
```

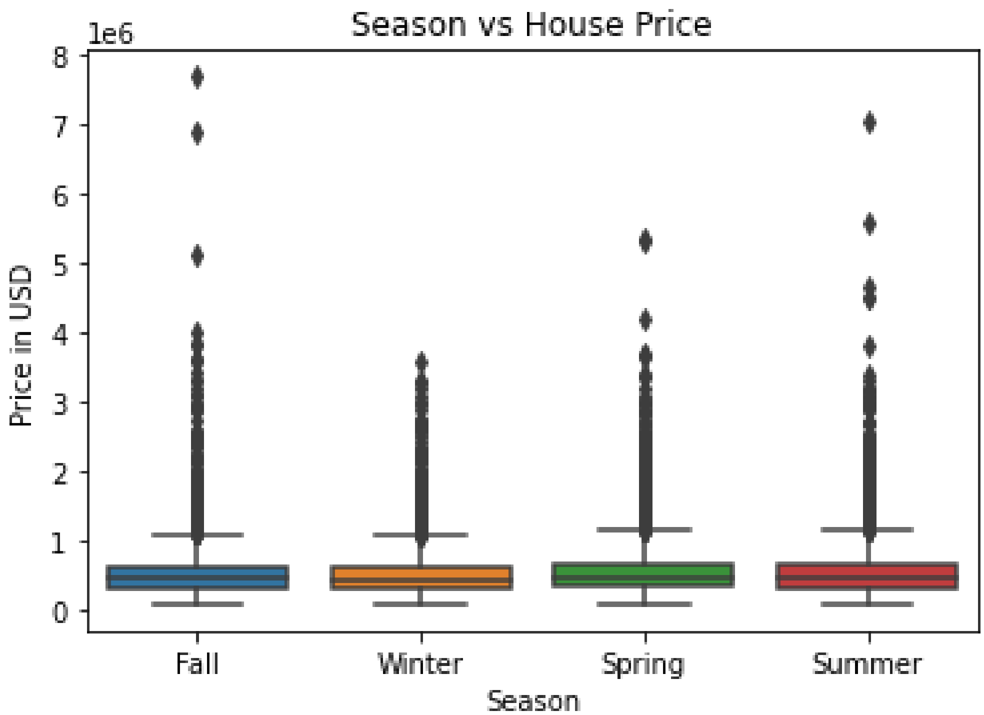
[32]:
```
sns.countplot(x = df['sale_year'])
plt.title('House Sales in 2014-2015')
plt.show()
```



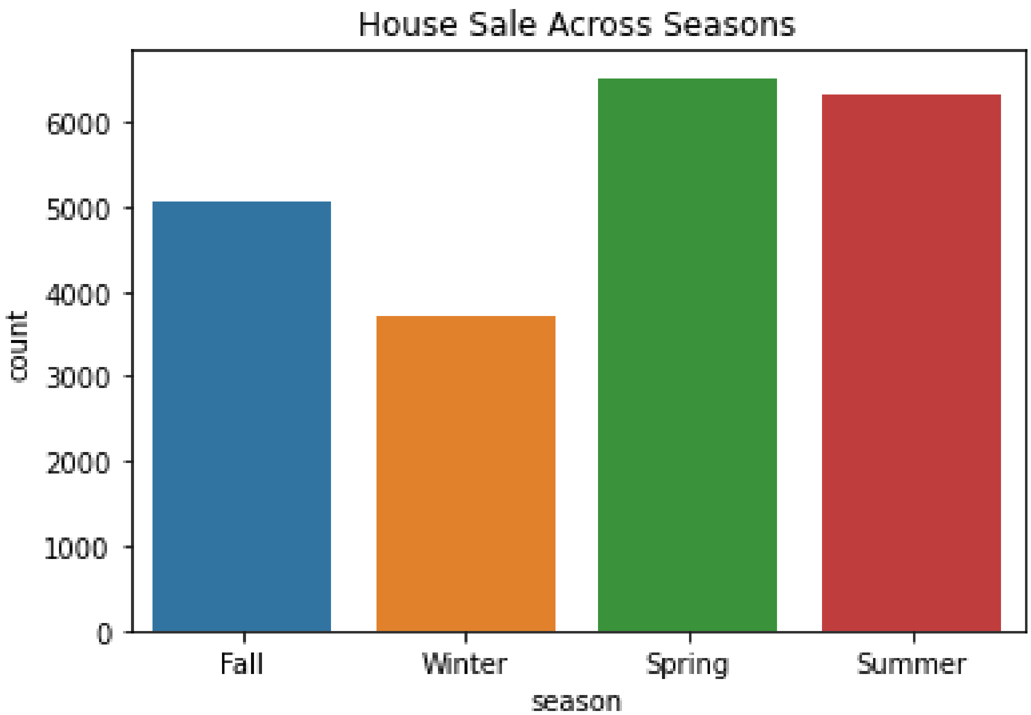
[33]:
```
# A function to determine season from the date column
def monthToSeason(df):
    df['sale_month'] = df.apply(lambda x: x.date.month, axis=1)
    seasons={1:'Winter',2:'Winter',3:'Spring',4:'Spring',5:'Spring',6:'Summer',
             7:'Summer',8:'Summer',9:'Fall',10:'Fall',11:'Fall',12:'Winter'}
    df['season']=df.sale_month.map(seasons)
    return df

df = monthToSeason(df)
df['season'].unique()
```

[33]:
```
array(['Fall', 'Winter', 'Spring', 'Summer'], dtype=object)
```

[34]:
```
sns.boxplot(x = df['season'], y = df['price'], data = df)
plt.xlabel('Season')
plt.ylabel('Price in USD')
plt.title('Season vs House Price')
plt.show()
```



[35]:
```
sns.countplot(x=df['season'])
plt.title('House Sale Across Seasons')
plt.show()
```



- There's an interesting observation in house sales across the years, almost twice as many houses were sold in 2014 as compared to 2015.
- The season has little to no impact on the overall price of the house.
- Many houses are sold/bought in Spring and Summer.
- Gearing our market campaings during Fall and Winter could get the ball rolling and complete purchase/sale in Spring and Winter

0.3.2 Modelling

[36]:
```
df.sort_values("price", ascending = False).head()
```

[36]:
```
              id        date      price  bedrooms  bathrooms  sqft_living  \
7245  6762700020  2014-10-13  7700000.0         6          8        12050
3910  9808700762  2014-06-11  7060000.0         5          4        10040
9245  9208900037  2014-09-19  6890000.0         6          8         9890
4407  2470100110  2014-08-04  5570000.0         5          6         9200
1446  8907500070  2015-04-13  5350000.0         5          5         8000

      sqft_lot  floors  waterfront  view  …  yr_renovated  zipcode      lat  \
7245     27600     2.5           0     4  …        1987.0    98102  47.6298
3910     37325     2.0           1     2  …        2001.0    98004  47.6500
9245     31374     2.0           0     5  …           0.0    98039  47.6305
4407     35069     2.0           0     1  …           NaN    98039  47.6289
1446     23985     2.0           0     5  …           0.0    98004  47.6232
```

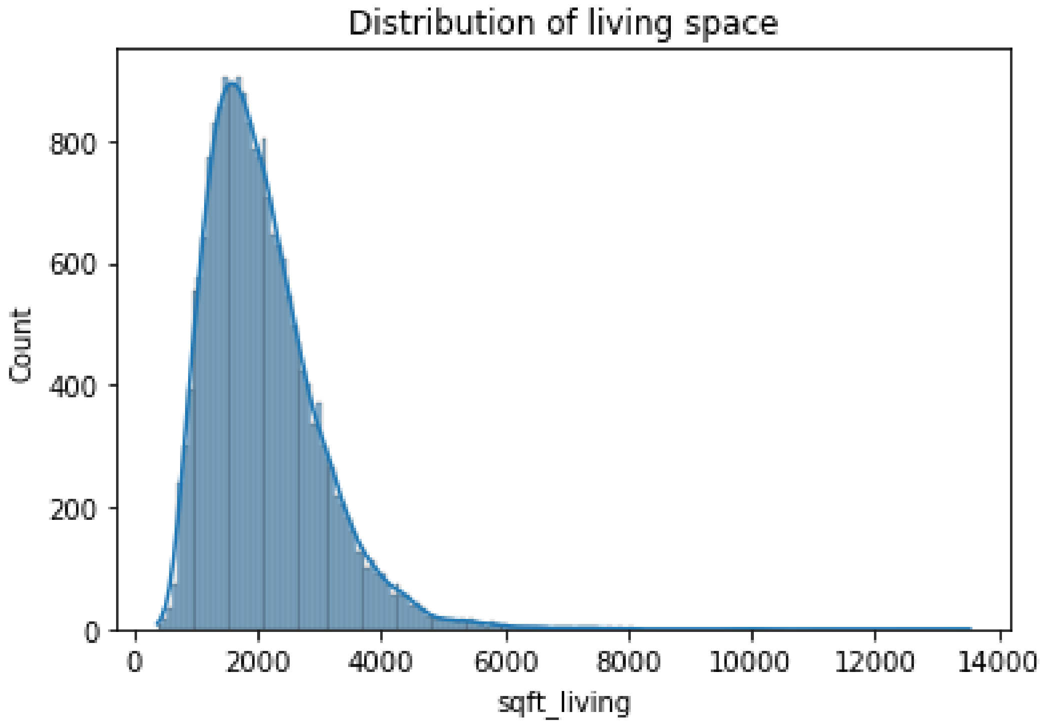
```
         long  sqft_living15  sqft_lot15  basement  sale_year  sale_month  \
7245 -122.323           3940        8800         1       2014          10
3910 -122.214           3930       25449         1       2014           6
9245 -122.240           4540       42730         1       2014           9
4407 -122.233           3560       24345         1       2014           8
1446 -122.220           4600       21750         1       2015           4

      season
7245    Fall
3910  Summer
9245    Fall
4407  Summer
1446  Spring

[5 rows x 25 columns]
```

[37]:
```
#Explore the relationshp between price and other columns
df.corr('pearson')['price']
```

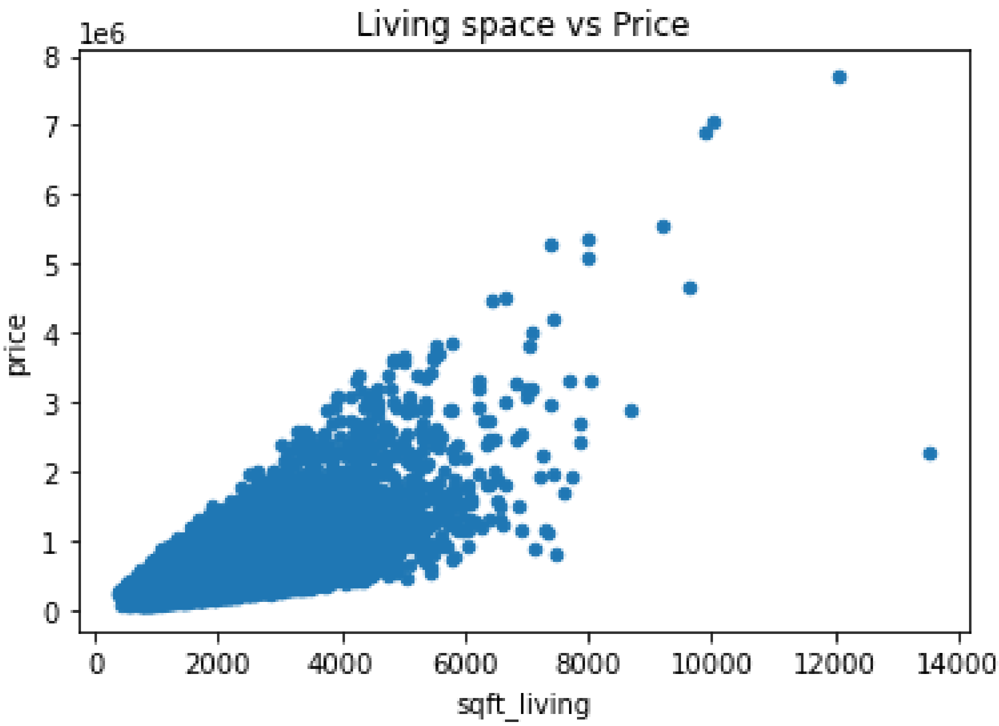
[37]:
```
id               -0.016765
price             1.000000
bedrooms          0.316504
bathrooms         0.519652
sqft_living       0.701948
sqft_lot          0.089879
floors            0.256828
waterfront        0.264308
view              0.394713
grade             0.667967
sqft_above        0.605401
sqft_basement     0.178264
yr_built          0.053964
yr_renovated      0.129702
zipcode          -0.053408
lat               0.306687
long              0.022045
sqft_living15     0.585274
sqft_lot15        0.082848
basement          0.178264
sale_year         0.003734
sale_month       -0.009925
Name: price, dtype: float64
```

[38]:
```
# living space Distribution
sns.histplot(df['sqft_living'], kde=True);
plt.title("Distribution of living space")
```

```
plt.show()
```



[39]:
```
# The 'sqft_living' column has the highest relationship with price
df.plot(x='sqft_living',y='price', kind='scatter')
plt.title("Living space vs Price")
plt.show()
```



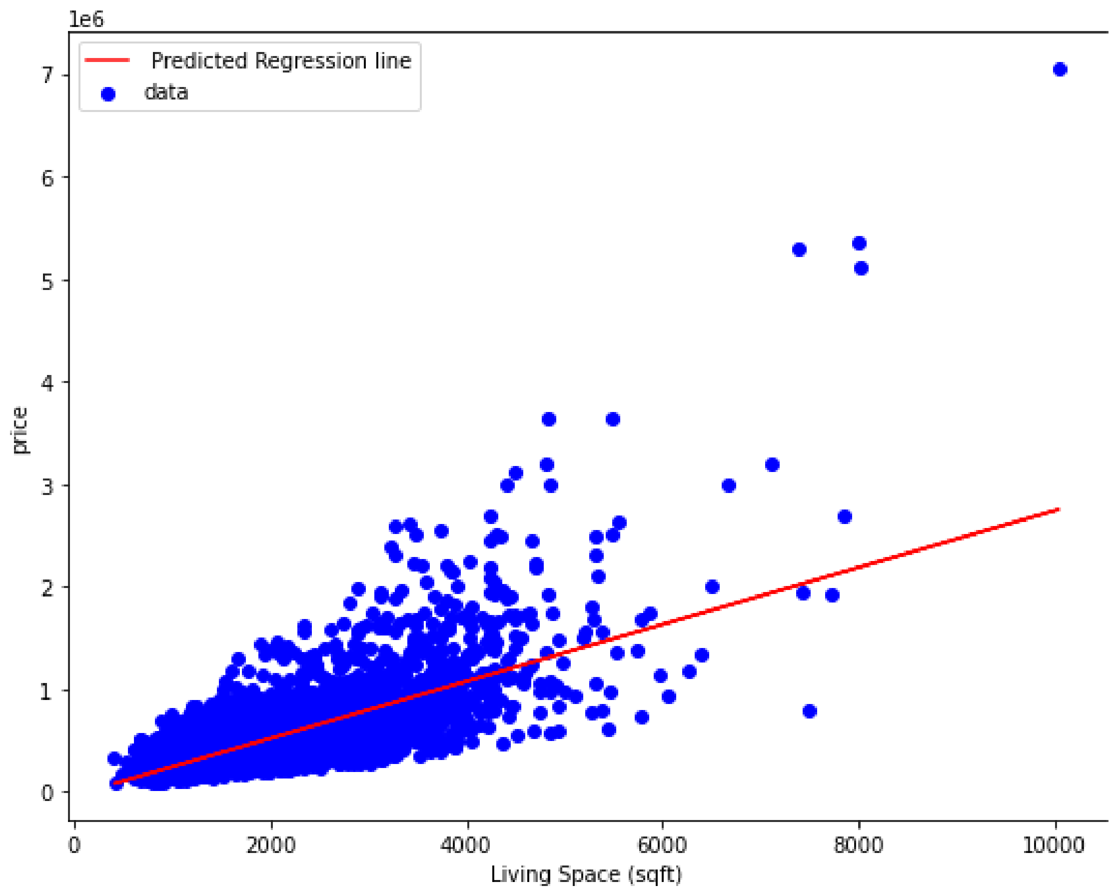
- From the two distributions, most of the houses have a living space within 6000 sqft and tend to cost a figure below $4,000,000

[40]:
```
train_data,test_data=train_test_split(df,train_size=0.8,random_state=3)
reg=LinearRegression()
x_train=np.array(train_data['sqft_living']).reshape(-1,1)
y_train=np.array(train_data['price']).reshape(-1,1)
reg.fit(x_train,y_train)

x_test=np.array(test_data['sqft_living']).reshape(-1,1)
y_test=np.array(test_data['price']).reshape(-1,1)
pred=reg.predict(x_test)
```

[41]:
```
plt.subplots(figsize = (9, 7))
plt.scatter(x_test, y_test, color= 'blue', label = 'data')
plt.plot(x_test, reg.predict(x_test), color ='red', label = ' Predicted␣
 ↪Regression line')
plt.xlabel('Living Space (sqft)')
plt.ylabel('price')
plt.legend()
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

#Model summary
```

```
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('Sqaured mean error', round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(x_train,y_train),3))
print('R sqaured testing',round(reg.score(x_test,y_test),3) )
print('intercept',reg.intercept_)
print('coefficient',reg.coef_)
```

```
Sqaured mean error 273789.66
R squared training 0.494
R sqaured testing 0.488
intercept [-38017.20149875]
coefficient [[277.90934404]]
```



- After living space, grade of the house is the next variable to have a strong correlation with price.

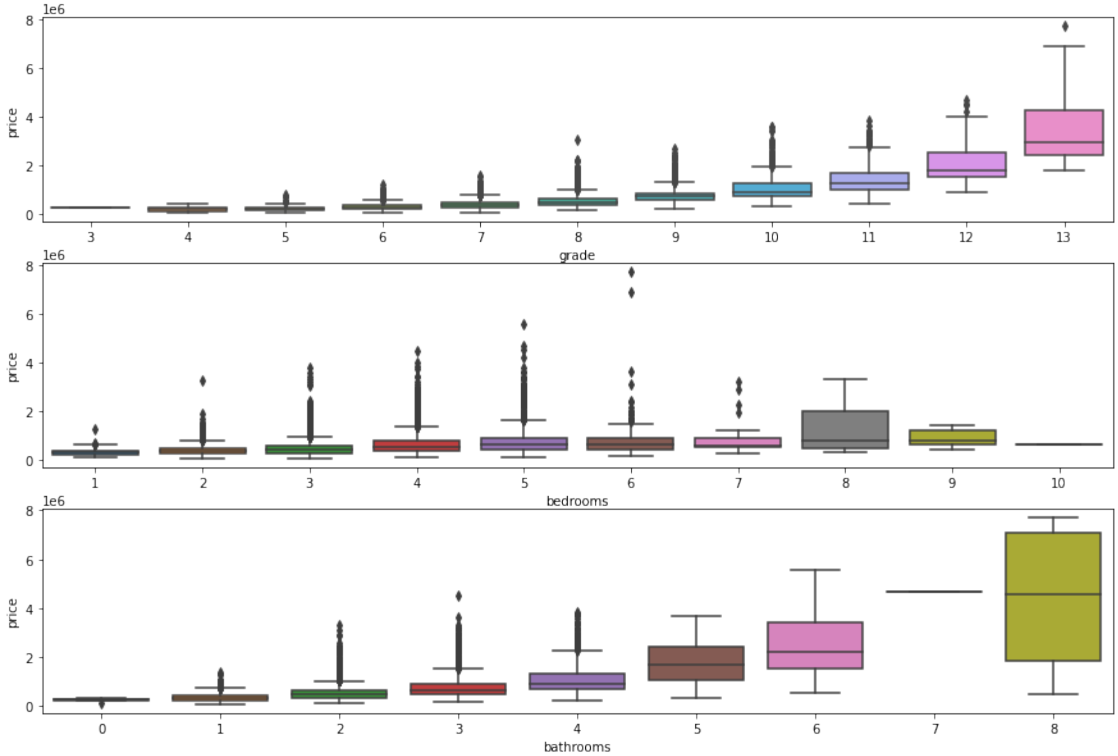
[42]:
```
train_data,test_data = train_test_split(df,train_size=0.8,random_state=3)
reg = LinearRegression()
x_train = np.array(train_data['grade']).reshape(-1,1)
```

```
y_train = np.array(train_data['price']).reshape(-1,1)
reg.fit(x_train,y_train)

x_test = np.array(test_data['grade']).reshape(-1,1)
y_test = np.array(test_data['price']).reshape(-1,1)
pred = reg.predict(x_test)

print("Model 1")
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('squared mean error',round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.score(x_train,y_train),3))
print('R squared testing',round(reg.score(x_test,y_test),3))
print('intercept',reg.intercept_)
print('coeeficient',reg.coef_)
```

```
Model 1
squared mean error 288645.26
R squared training 0.45
R squared testing 0.431
intercept [-1050417.80366832]
coeeficient [[207678.58717913]]
```

0.3.3 Multiple regression

[43]:
```
fig,ax=plt.subplots(3,1,figsize=(15,10))
sns.boxplot(x=train_data['grade'],y=train_data['price'],ax=ax[0])
sns.boxplot(x=train_data['bedrooms'],y=train_data['price'],ax=ax[1])
sns.boxplot(x=train_data['bathrooms'],y=train_data['price'],ax=ax[2])
plt.show()
```



[44]:
```
model_2 = ['bedrooms','grade','sqft_living','sqft_above']
reg=LinearRegression()
reg.fit(train_data[model_2],train_data['price'])
pred=reg.predict(test_data[model_2])
print('Model 2')
mean_squared_error=metrics.mean_squared_error(y_test,pred)
print('mean squared error(MSE)', round(np.sqrt(mean_squared_error),2))
print('R squared training',round(reg.
 ↪score(train_data[model_2],train_data['price']),3))
print('R squared testing', round(reg.
 ↪score(test_data[model_2],test_data['price']),3))
print('Intercept: ', reg.intercept_)
print('Coefficient:', reg.coef_)
```

```
Model 2
mean squared error(MSE) 260437.56
R squared training 0.554
R squared testing 0.537
Intercept:  -513600.3368260097
Coefficient: [-4.73070108e+04  1.04713234e+05  2.68444581e+02 -8.22739427e+01]
```

[45]:
```
model_3 =␣
↪['basement','bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view','g
reg = LinearRegression()
reg.fit(train_data[model_3],train_data['price'])
pred = reg.predict(test_data[model_3])

print('Model 3')
mean_squared_error = metrics.mean_squared_error(y_test, pred)
print('Mean Squared Error (MSE) ', round(np.sqrt(mean_squared_error), 2))
print('R-squared (training) ', round(reg.score(train_data[model_3],␣
↪train_data['price']), 3))
print('R-squared (testing) ', round(reg.score(test_data[model_3],␣
↪test_data['price']), 3))
print('Intercept: ', reg.intercept_)
print('Coefficient:', reg.coef_)
```

```
Model 3
Mean Squared Error (MSE)  222830.8
R-squared (training)  0.659
R-squared (testing)  0.661
Intercept:  -31881653.514646027
Coefficient: [ 2.13948976e+03 -3.24404951e+04  2.20573040e+04  1.77897381e+02
 -1.94735194e-01 -3.82042849e+04  5.25017963e+05  7.83235763e+04
  8.10187710e+04  1.27264594e+01  6.60923385e+05]
```

0.3.4 Model summary

- Our third model better as compared to our baseline model.
- It was a better predictor of price as it had a lower error value and a higher R-Squared value

0.3.5 Conclusions

- An increase in the number of `bedrooms` and `bathrooms` increases the price of the house. Houses with a good view also tend to fetch high prices
- Grade is a good indicator of price. Renovations will therefore contribute to price as it improves the house condition and grade to some extent.
- Houses with a waterfront cost thrice as smuch as houses without. This are high end properties that should have customised marketing campaign.
- The season or time of the year was a prospect in determining price but it turns out there's no significant contribution. However, the distribution gives some insight on timing of our marketing campaigns.