Washington House Price Prediction -- Capstone Project

Table of Content

1. Introduction

Problem Statement

As a house value is simply more than location and square footage. Like the features that make up a person, an educated party would want to know all aspects that give a house its value. For example, if we want to sell a house and we don't know the price which we can take, as it can't be too low or too high. To find house price we usually try to find similar properties in our neighbourhood and based on collected data we trying to assess our house price.

Data Dictionary

1. cid: a notation for a house (house id)
2. dayhours: Date house was sold (date-month-year of sale)
3. price: Price is prediction target (Target variable)
4. room_bed: Number of Bedrooms/House (number of bedrooms per house)
5. room_bath: Number of bathrooms/bedrooms (number of bathrooms per bedrooms)
6. living_measure: square footage of the home
7. lot_measure: square footage of the lot
8. ceil: Total floors (levels) in house (how many floors, ground, 1st floor, 2nd floor ..)
9. coast: House which has a view to a waterfront (house with or without coastview/seafacing)
10. sight: Has been viewed (sight has been viewed before buying this house)
11. condition: How good the condition is (Overall)
12. quality: grade given to the housing unit, based on grading system
13. ceil_measure: square footage of house apart from basement
14. basement_measure: square footage of the basement
15. yr_built: Built Year (year the house was built)
16. yr_renovated: Year when house was renovated (year the house was renovated)
17. zipcode: zip code of the area
18. lat: Latitude coordinate
19. long: Longitude coordinate
20. living_measure15: Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area
21. lot_measure15: lotSize area in 2015(implies-- some renovations)
22. furnished: Based on the quality of room
23. total_area: Measure of both living and lot

Initial data loading

Import Initial Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
## import important libraries for model building

from scipy.stats import norm
from scipy import stats
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
```

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor

from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from math import sqrt
```

Read Dataset

In [3]:
```
#read dataset
dataset_raw = pd.read_excel("innercity.xlsx")
```

In [4]:
```
#display head of the complete dataset (with all columns)
pd.set_option("display.max_columns", None)
dataset_raw.head()
```

Out[4]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition	quality	ceil_measu
0	3876100940	20150427T000000	600000	4.0	1.75	3050.0	9440.0	1	0	0.0	3	8.0	180
1	3145600250	20150317T000000	190000	2.0	1.00	670.0	3101.0	1	0	0.0	4	6.0	67
2	7129303070	20140820T000000	735000	4.0	2.75	3040.0	2415.0	2	1	4.0	3	8.0	304
3	7338220280	20141010T000000	257000	3.0	2.50	1740.0	3721.0	2	0	0.0	3	8.0	174
4	7950300670	20150218T000000	450000	2.0	1.00	1120.0	4590.0	1	0	0.0	3	7.0	112

In [5]:
```
print('These are the variable columns in the dataset: \n\n', dataset_raw.columns )
```

```
These are the variable columns in the dataset:

 Index(['cid', 'dayhours', 'price', 'room_bed', 'room_bath', 'living_measure',
       'lot_measure', 'ceil', 'coast', 'sight', 'condition', 'quality',
       'ceil_measure', 'basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'living_measure15', 'lot_measure15', 'furnished',
       'total_area'],
      dtype='object')
```

In [6]:
```
# number of columns and rows
print('The number of rows (observations) is',dataset_raw.shape[0],'\n''The number of columns (variables) is',dataset_raw.
```

```
The number of rows (observations) is 21613
The number of columns (variables) is 23
```

In [7]:
```
dataset_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   cid               21613 non-null  int64
 1   dayhours          21613 non-null  object
 2   price             21613 non-null  int64
 3   room_bed          21505 non-null  float64
 4   room_bath         21505 non-null  float64
 5   living_measure    21596 non-null  float64
 6   lot_measure       21571 non-null  float64
 7   ceil              21571 non-null  object
 8   coast             21612 non-null  object
 9   sight             21556 non-null  float64
 10  condition         21556 non-null  object
 11  quality           21612 non-null  float64
 12  ceil_measure      21612 non-null  float64
 13  basement          21612 non-null  float64
 14  yr_built          21612 non-null  object
 15  yr_renovated      21613 non-null  int64
 16  zipcode           21613 non-null  int64
 17  lat               21613 non-null  float64
 18  long              21613 non-null  object
 19  living_measure15  21447 non-null  float64
 20  lot_measure15     21584 non-null  float64
 21  furnished         21584 non-null  float64
 22  total_area        21584 non-null  object
dtypes: float64(12), int64(4), object(7)
memory usage: 3.8+ MB
```

Note:

1. There are 23 variables. There are 12 float datatypes, 4 integer datatypes and 7 object datatypes.

2. There are few time-data columns like dayhours (day at which house was sold),
3. yr_built (year on which house was built), and yr_renovated (year on which house was renovated). These variables are not in time format, which we have to change, "if required".
4. Object datatypes can also be changed to float for seeing their correlation with other variables.

In [8]:
```
## statistical summary of the raw dataset
dataset_raw.describe().round(2).T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
cid	21613.0	4.580302e+09	2.876566e+09	1000102.00	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	21613.0	5.401822e+05	3.673622e+05	75000.00	3.219500e+05	4.500000e+05	6.450000e+05	7.700000e+06
room_bed	21505.0	3.370000e+00	9.300000e-01	0.00	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
room_bath	21505.0	2.120000e+00	7.700000e-01	0.00	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
living_measure	21596.0	2.079860e+03	9.185000e+02	290.00	1.429250e+03	1.910000e+03	2.550000e+03	1.354000e+04
lot_measure	21571.0	1.510458e+04	4.142362e+04	520.00	5.040000e+03	7.618000e+03	1.068450e+04	1.651359e+06
sight	21556.0	2.300000e-01	7.700000e-01	0.00	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
quality	21612.0	7.660000e+00	1.180000e+00	1.00	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
ceil_measure	21612.0	1.788370e+03	8.281000e+02	290.00	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03
basement	21612.0	2.915200e+02	4.425800e+02	0.00	0.000000e+00	0.000000e+00	5.600000e+02	4.820000e+03
yr_renovated	21613.0	8.440000e+01	4.016800e+02	0.00	0.000000e+00	0.000000e+00	0.000000e+00	2.015000e+03
zipcode	21613.0	9.807794e+04	5.351000e+01	98001.00	9.803300e+04	9.806500e+04	9.811800e+04	9.819900e+04
lat	21613.0	4.756000e+01	1.400000e-01	47.16	4.747000e+01	4.757000e+01	4.768000e+01	4.778000e+01
living_measure15	21447.0	1.987070e+03	6.855200e+02	399.00	1.490000e+03	1.840000e+03	2.360000e+03	6.210000e+03
lot_measure15	21584.0	1.276654e+04	2.728699e+04	651.00	5.100000e+03	7.620000e+03	1.008700e+04	8.712000e+05
furnished	21584.0	2.000000e-01	4.000000e-01	0.00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00

Dealing with location data

Using zipcodes to track city, county, state and population of the area.

In [6]:
```
us_zipcodes = pd.read_excel("USA_Zipcodes.xlsx")
us_zipcodes.head(3)
```

Out[6]:

	zip	lat	lng	city	state_id	state_name	population	density	county_name	county_names_all
0	601	18.18005	-66.75218	Adjuntas	PR	Puerto Rico	17113.0	102.7	Adjuntas	Adjuntas|Utuado
1	602	18.36074	-67.17519	Aguada	PR	Puerto Rico	37751.0	476.0	Aguada	Aguada
2	603	18.45440	-67.12201	Aguadilla	PR	Puerto Rico	47081.0	574.9	Aguadilla	Aguadilla

In [7]:
```
dataset_new = pd.merge(dataset_raw,us_zipcodes, how='left',left_on=['zipcode'],right_on=['zip']).dropna()
dataset = dataset_new
dataset.head(3)
```

Out[7]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition	quality	ceil_measu
0	3876100940	20150427T000000	600000	4.0	1.75	3050.0	9440.0	1	0	0.0	3	8.0	180
1	3145600250	20150317T000000	190000	2.0	1.00	670.0	3101.0	1	0	0.0	4	6.0	67
2	7129303070	20140820T000000	735000	4.0	2.75	3040.0	2415.0	2	1	4.0	3	8.0	304

In [8]:
```
cols_unique = ('city', 'state_id', 'state_name', 'county_name', 'county_names_all')
for i in cols_unique:
    print('unique items in', i , ': \n', dataset[i].unique(), '\n')
```

```
unique items in city :
 ['Kirkland' 'Seattle' 'Auburn' 'Kent' 'Bellevue' 'Snoqualmie' 'Renton'
 'Redmond' 'Maple Valley' 'Federal Way' 'Sammamish' 'Bothell' 'Carnation'
 'Woodinville' 'North Bend' 'Issaquah' 'Mercer Island' 'Vashon' 'Kenmore'
 'Duvall' 'Fall City' 'Black Diamond' 'Medina' 'Enumclaw']

unique items in state_id :
 ['WA']

unique items in state_name :
 ['Washington']

unique items in county_name :
 ['King']

unique items in county_names_all :
 ['King' 'King|Pierce' 'King|Snohomish']
```

In [9]:
```
# lets drop the unnecessary columns from us zipcodes
# and also drop repeated columns like lat and long and rename them as latitude and lontitude

dataset.drop(['lat_y', 'lng', 'state_id', 'state_name', 'county_name', 'county_names_all', 'zip'], axis=1, inplace=True)
dataset.rename(columns={'lat_x':'latitude', 'long':'longitude'}, inplace=True)
dataset.head()
```

Out[9]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	condition	quality	ceil_measu
0	3876100940	20150427T000000	600000	4.0	1.75	3050.0	9440.0	1	0	0.0	3	8.0	180
1	3145600250	20150317T000000	190000	2.0	1.00	670.0	3101.0	1	0	0.0	4	6.0	67
2	7129303070	20140820T000000	735000	4.0	2.75	3040.0	2415.0	2	1	4.0	3	8.0	304
3	7338220280	20141010T000000	257000	3.0	2.50	1740.0	3721.0	2	0	0.0	3	8.0	174
4	7950300670	20150218T000000	450000	2.0	1.00	1120.0	4590.0	1	0	0.0	3	7.0	112

Basic Data Cleaning

Convert datatypes for ease

In [10]:
```
## converting dayhours into timestamp
dataset['dayhours'] = pd.to_datetime(dataset['dayhours'], format='%Y-%m-%d')
```

In [11]:
```
## changing datatype of cid (house id) from int to object
dataset['cid']=dataset['cid'].astype(np.object)
```

In [12]:
```
## changing datatype of zipcode from int to integer datatype
dataset = dataset.astype({'zipcode':'int64'})
```

In [13]:
```
## printing total number of unique house in the dataset
print('The list of unique house ids present in the dataset is',dataset['cid'].nunique(), ', \n as compared to total numbe
```

```
The list of unique house ids present in the dataset is 21211 ,
 as compared to total number of 21387 rows of the dataset.
```

This shows that there are many house that are sold more than one times.

In [14]:
```
## list of columns which are float types
floatColumns = dataset.dtypes[dataset.dtypes == np.object]
print(floatColumns)
```

```
cid           object
ceil          object
coast         object
condition     object
yr_built      object
longitude     object
total_area    object
city          object
dtype: object
```

here, total area and lontitude needs to be converted into numeric values

In [15]:
```
## printing unique values of columns which is object datatype but has numeric value'
## we are ignoring total area which is obviously wrongly placed as object

object_unique = ('ceil', 'coast', 'condition')
for i in object_unique:
    print('unique items in', i , ': \n', dataset[i].unique(), '\n')
```

```
unique items in ceil :
 [1 2 3 1.5 2.5 '$' 3.5]

unique items in coast :
 [0 1 '$']

unique items in condition :
 [3 4 5 2 1]
```

In [16]:
```
## list of columns which are numeric types
numericColumns = dataset.dtypes[dataset.dtypes != np.object]
print(numericColumns)
```

```
dayhours           datetime64[ns]
price                       int64
room_bed                  float64
room_bath                 float64
living_measure            float64
lot_measure               float64
sight                     float64
quality                   float64
```

```
ceil_measure              float64
basement                  float64
yr_renovated                int64
zipcode                     int64
latitude                  float64
living_measure15          float64
lot_measure15             float64
furnished                 float64
population                float64
density                   float64
dtype: object
```

In [17]:

```
## printing unique values of columns which is float datatype but might have a object type feel'

## zipcode is an obvious object which is recorded as float
numeric_float_unique = ('room_bed', 'room_bath', 'sight', 'quality' , 'yr_renovated' , 'furnished')
for i in numeric_float_unique:
    print('unique items in', i , ': \n', dataset[i].unique(), '\n')
```

```
unique items in room_bed :
 [ 4.  2.  3.  1.  5.  6.  7. 10.  8.  0.  9. 33. 11.]

unique items in room_bath :
 [1.75 1.   2.75 2.5  1.5  3.5  2.   2.25 3.   4.   3.25 3.75 5.   0.75
 5.5  4.25 4.5  4.75 8.   6.75 5.25 6.   0.   1.25 5.75 7.5  6.5  0.5
 7.75 6.25]

unique items in sight :
 [0. 4. 2. 3. 1.]

unique items in quality :
 [ 8.  6.  7. 10.  9.  5. 11. 13.  4. 12.  1.  3.]

unique items in yr_renovated :
 [   0 1993 2014 1983 1992 2000 2011 1994 2009 1944 1971 2003 1955 1985
 2008 2015 2005 1979 1998 1968 2010 1989 2002 1987 1999 1996 1940 1986
 1988 1969 1995 2004 2007 2013 2001 1990 1958 2012 1967 1991 1970 1984
 2006 1982 1951 1960 1956 1997 1980 1959 1974 1973 1975 1981 1963 1957
 1976 1948 1945 1977 1978 1972 1965 1964 1953 1950 1962 1946 1934 1954]

unique items in furnished :
 [0. 1.]
```

In [18]:

```
## Changing datatype from object to float with error coarse to change unwanted strings into NaN values

change_datatype_cols = dataset[['ceil','coast','condition', 'yr_built' , 'longitude', 'total_area']]

for i in change_datatype_cols:
    dataset[i] = pd.to_numeric(dataset[i],errors='coerce')
```

In [19]:

```
## Converting other date variables from object/int datatype to date datatype
    # dataset['yr_built'] = pd.to_datetime(dataset['yr_built'], errors='coerce', format='%Y')
    # dataset['yr_renovated'] = pd.to_datetime(dataset['yr_renovated'], format='%Y')
```

In [20]:

```
## Remove unwanted string characters
   #dataset = dataset.replace("$","")
```

In [21]:

```
## checking datatypes after conversion
dataset.dtypes
```

Out[21]:

```
cid                        object
dayhours           datetime64[ns]
price                       int64
room_bed                  float64
room_bath                 float64
living_measure            float64
lot_measure               float64
ceil                      float64
coast                     float64
sight                     float64
condition                   int64
quality                   float64
ceil_measure              float64
basement                  float64
yr_built                    int64
yr_renovated                int64
zipcode                     int64
latitude                  float64
longitude                 float64
living_measure15          float64
lot_measure15             float64
furnished                 float64
total_area                float64
city                       object
population                float64
density                   float64
dtype: object
```

Renaming columns for ease to understand

we are renaming the confusing column names for an ease of understanding.

In [22]:
```
# creating reference dataframe of the list of columns which are renamed

cols_renaming = {'original_Column':['cid','dayhours','room_bed','room_bath','ceil', 'coast', 'sight', 'quality','living_m
        'renamed_Column':['house_id','date','bedroom','ratio_bathroom','total_floors', 'seaface', 'sight_viewed', 'qualit

# Create DataFrame
df_renamedCols = pd.DataFrame(cols_renaming)
df_renamedCols
```

Out[22]:

	original_Column	renamed_Column
0	cid	house_id
1	dayhours	date
2	room_bed	bedroom
3	room_bath	ratio_bathroom
4	ceil	total_floors
5	coast	seaface
6	sight	sight_viewed
7	quality	quality_grade
8	living_measure	living_area
9	lot_measure	lot_area
10	ceil_measure	floor_area
11	basement	basement_area
12	lat	latitude
13	long	longitude
14	living_measure15	living_area_2015
15	lot_measure15	lot_area_2015
16	density	population_density

In [23]:
```
## renaming all the confusing column names
dataset.rename(columns={'cid':'house_id','dayhours':'date','room_bed':'bedroom','room_bath':'ratio_bathroom','ceil':'tota

## rechecking head after renaming the columns
dataset.head()
```

Out[23]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	flo
0	3876100940	2015-04-27	600000	4.0	1.75	3050.0	9440.0	1.0	0.0	0.0	3	8.0	
1	3145600250	2015-03-17	190000	2.0	1.00	670.0	3101.0	1.0	0.0	0.0	4	6.0	
2	7129303070	2014-08-20	735000	4.0	2.75	3040.0	2415.0	2.0	1.0	4.0	3	8.0	
3	7338220280	2014-10-10	257000	3.0	2.50	1740.0	3721.0	2.0	0.0	0.0	3	8.0	
4	7950300670	2015-02-18	450000	2.0	1.00	1120.0	4590.0	1.0	0.0	0.0	3	7.0	

In [24]:
```
## sorting dataset based on date
dataset = dataset.sort_values('date')
```

In [25]:
```
## resorted datset
dataset.reset_index(inplace=True)
dataset.drop(['index'], axis=1, inplace=True)
```

In [26]:
```
dataset.head()
```

Out[26]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f
0	4217401055	2014-05-02	1400000	4.0	2.50	2920.0	4000.0	1.5	0.0	0.0	5	8.0	
1	2493200195	2014-05-02	615000	3.0	1.75	2360.0	7291.0	1.0	0.0	0.0	4	8.0	
2	3221059036	2014-05-02	400000	4.0	2.50	3630.0	42884.0	1.5	0.0	0.0	3	9.0	
3	1525079056	2014-05-02	284000	3.0	1.75	1800.0	23103.0	1.0	0.0	0.0	3	7.0	

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f
4	7525100520	2014-05-02	335000	2.0	2.00	1350.0	2560.0	1.0	0.0	0.0	3	8.0	

In [27]:
```
## checking duplicate rows
dataset.duplicated().sum()
```

Out[27]:
```
0
```

In [28]:
```
## checking if there are duplicate houses that sold more than one times
dataset.duplicated(subset=['house_id']).sum()
```

Out[28]:
```
176
```

Note: This shows that few of the house are sold multiples times.

In [29]:
```
## house sold multiple times
id_count = dataset['house_id'].value_counts()
id_count[id_count>1]
```

Out[29]:
```
795000620     3
4202400078    2
1545800290    2
7961500010    2
4139440480    2
             ..
7972000010    2
2044500213    2
1568100300    2
6308000010    2
3303000130    2
Name: house_id, Length: 175, dtype: int64
```

Deriving new columns

1. Adding a new column 'prev_sold'

Note: new dataset showing house resold no of times

1. 0 mean not resold or sold the very first time. House sold before 0 times.
2. 1 means house sold once before current purchase. House sold before 1 times.
3. 2 means house sold twice before current purchase. House sold before 2 times.

The reason behind doing this is because the price might increase or decrease if a property is resold multiple times.

In [30]:
```
## lets generate a new column that counts in order that how many times a house was sold
dataset['prev_sold'] = dataset.groupby('house_id').cumcount()
```

In [31]:
```
## new dataset showing house resold no of times
dataset.head(3)
```

Out[31]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f
0	4217401055	2014-05-02	1400000	4.0	2.50	2920.0	4000.0	1.5	0.0	0.0	5	8.0	
1	2493200195	2014-05-02	615000	3.0	1.75	2360.0	7291.0	1.0	0.0	0.0	4	8.0	
2	3221059036	2014-05-02	400000	4.0	2.50	3630.0	42884.0	1.5	0.0	0.0	3	9.0	

In [32]:
```
dataset[(dataset['prev_sold'] == 2)]
```

Out[32]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade
17225	795000620	2015-03-11	157000	3.0	1.0	1080.0	6250.0	1.0	0.0	0.0	2	5.0

2. from ratio of bathroom per bedroom to number of bathrooms

In [33]:
```
dataset['bathroom']=dataset['ratio_bathroom']*dataset['bedroom']
```

In [34]:
```
## new dataset showing new bathroom column
dataset.head(3)
```

Out[34]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f
0	4217401055	2014-05-02	1400000	4.0	2.50	2920.0	4000.0	1.5	0.0	0.0	5	8.0	
1	2493200195	2014-05-02	615000	3.0	1.75	2360.0	7291.0	1.0	0.0	0.0	4	8.0	
2	3221059036	2014-05-02	400000	4.0	2.50	3630.0	42884.0	1.5	0.0	0.0	3	9.0	

3. from Counting house built-years and renovation-years

We are counting years of house built and renovation years instead of keeping them as a year number. This will help us in further analysis.

In [35]:
```
## deriving house age from the year house is built by taking next year (2016) as refernce year.
## we haven't taken 2015 as ref year orlese we have got 0 for many house built in 2015.

dataset['house_age'] = 2016 - dataset['yr_built']
```

In [36]:
```
## deriving renovation age

dataset['renovation_yrs'] = np.where(dataset['yr_renovated']!= 0, 2016 - dataset['yr_renovated'], 0)
```

In [37]:
```
## deriving renovation status
dataset['renovated_orNot'] = np.where(dataset['renovation_yrs']!= 0, 1, 0)
```

4. Month house was sold

In [38]:
```
## converting dayhours into timestamp

dataset['sold_month'] = pd.to_datetime(dataset['date'], format='%Y-%m')
dataset['sold_month'] = dataset['sold_month'].apply(lambda x: x.strftime('%m'))
dataset['sold_month'].head()
```

Out[38]:
```
0    05
1    05
2    05
3    05
4    05
Name: sold_month, dtype: object
```

In [39]:
```
## changing datatype of zipcode from int to integer datatype
dataset = dataset.astype({'sold_month':'int64'})
```

In [40]:
```
dataset.head(3)
```

Out[40]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f
0	4217401055	2014-05-02	1400000	4.0	2.50	2920.0	4000.0	1.5	0.0	0.0	5	8.0	
1	2493200195	2014-05-02	615000	3.0	1.75	2360.0	7291.0	1.0	0.0	0.0	4	8.0	
2	3221059036	2014-05-02	400000	4.0	2.50	3630.0	42884.0	1.5	0.0	0.0	3	9.0	

5. deal with areas

In [41]:
```
areas=dataset[['living_area','lot_area', 'floor_area', 'basement_area', 'total_area', 'living_area_2015', 'lot_area_2015'
areas.head()
```

Out[41]:

	living_area	lot_area	floor_area	basement_area	total_area	living_area_2015	lot_area_2015	renovated_orNot
0	2920.0	4000.0	1910.0	1010.0	6920.0	2470.0	4000.0	0
1	2360.0	7291.0	1360.0	1000.0	9651.0	1860.0	5499.0	0
2	3630.0	42884.0	2300.0	1330.0	46514.0	2830.0	80148.0	0
3	1800.0	23103.0	1800.0	0.0	24903.0	1410.0	18163.0	0
4	1350.0	2560.0	1350.0	0.0	3910.0	1790.0	2560.0	0

Note:

Here, we can observe that:

1. total_area = living_area + lot_area
2. living area = floor_area + basement_area
3. even though house is not renovated but still living area and lot area in 2015 has been updated with changed values.
4. here we can see that few of the house may not have basement. So we can derive a new columns as basement_orNot.

6. column mentioning Basement availability

In [42]:
```
dataset["basement_orNot"]=dataset["basement_area"].apply( lambda x:1 if x>0 else 0)
dataset.head(3)
```

Out[42]:

	house_id	date	price	bedroom	ratio_bathroom	living_area	lot_area	total_floors	seaface	sight_viewed	condition	quality_grade	f
0	4217401055	2014-05-02	1400000	4.0	2.50	2920.0	4000.0	1.5	0.0	0.0	5	8.0	
1	2493200195	2014-05-02	615000	3.0	1.75	2360.0	7291.0	1.0	0.0	0.0	4	8.0	
2	3221059036	2014-05-02	400000	4.0	2.50	3630.0	42884.0	1.5	0.0	0.0	3	9.0	

Eliminating unnecessary columns

Note: We are removing unnecessary columns like house id, yr_built and yr_renovated. We have already extracted meaningful values and created new columns based on these columns.

In [43]:
```
dataset = dataset.drop(['house_id', 'ratio_bathroom', 'yr_built', 'yr_renovated'], axis = 1)
```

In [44]:
```
dataset = dataset.drop(['date'], axis = 1)
```

Statistical Summary of the new dataset

In [45]:
```
#statistical summary
dataset.describe().round(2).T
```

Out[45]:

	count	mean	std	min	25%	50%	75%	max
price	21387.0	540352.04	368108.93	75000.00	321000.00	450000.00	645000.00	7700000.00
bedroom	21387.0	3.37	0.93	0.00	3.00	3.00	4.00	33.00
living_area	21387.0	2080.47	918.94	290.00	1430.00	1910.00	2550.00	13540.00
lot_area	21387.0	15111.42	41449.08	520.00	5040.00	7620.00	10687.50	1651359.00
total_floors	21357.0	1.49	0.54	1.00	1.00	1.50	2.00	3.50
seaface	21357.0	0.01	0.09	0.00	0.00	0.00	0.00	1.00
sight_viewed	21387.0	0.23	0.77	0.00	0.00	0.00	0.00	4.00
condition	21387.0	3.41	0.65	1.00	3.00	3.00	4.00	5.00
quality_grade	21387.0	7.66	1.18	1.00	7.00	7.00	8.00	13.00
floor_area	21387.0	1789.03	828.58	290.00	1190.00	1560.00	2210.00	9410.00
basement_area	21387.0	291.45	442.68	0.00	0.00	0.00	560.00	4820.00
zipcode	21387.0	98077.89	53.50	98001.00	98033.00	98065.00	98117.00	98199.00
latitude	21387.0	47.56	0.14	47.16	47.47	47.57	47.68	47.78
longitude	21353.0	-122.21	0.14	-122.52	-122.33	-122.23	-122.12	-121.32
living_area_2015	21387.0	1987.04	685.76	399.00	1490.00	1840.00	2360.00	6210.00
lot_area_2015	21387.0	12762.13	27241.16	651.00	5100.00	7620.00	10085.00	871200.00
furnished	21387.0	0.20	0.40	0.00	0.00	0.00	0.00	1.00
total_area	21348.0	17198.85	41653.51	1423.00	7035.00	9580.00	12999.25	1652659.00
population	21387.0	34710.95	12700.56	3267.00	25474.00	35041.00	43471.00	70245.00
population_density	21387.0	1908.81	1400.22	17.60	892.50	1791.00	2649.40	7776.10
prev_sold	21387.0	0.01	0.09	0.00	0.00	0.00	0.00	2.00
bathroom	21387.0	7.50	4.23	0.00	4.50	7.00	10.00	67.50
house_age	21387.0	44.97	29.37	1.00	19.00	41.00	65.00	116.00
renovation_yrs	21387.0	0.85	5.16	0.00	0.00	0.00	0.00	82.00
renovated_orNot	21387.0	0.04	0.20	0.00	0.00	0.00	0.00	1.00
sold_month	21387.0	6.57	3.12	1.00	4.00	6.00	9.00	12.00
basement_orNot	21387.0	0.39	0.49	0.00	0.00	0.00	1.00	1.00

In [46]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21387 entries, 0 to 21386
Data columns (total 28 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   price               21387 non-null  int64
 1   bedroom             21387 non-null  float64
 2   living_area         21387 non-null  float64
 3   lot_area            21387 non-null  float64
 4   total_floors        21357 non-null  float64
 5   seaface             21357 non-null  float64
 6   sight_viewed        21387 non-null  float64
 7   condition           21387 non-null  int64
 8   quality_grade       21387 non-null  float64
 9   floor_area          21387 non-null  float64
 10  basement_area       21387 non-null  float64
 11  zipcode             21387 non-null  int64
 12  latitude            21387 non-null  float64
 13  longitude           21353 non-null  float64
 14  living_area_2015    21387 non-null  float64
 15  lot_area_2015       21387 non-null  float64
 16  furnished           21387 non-null  float64
 17  total_area          21348 non-null  float64
 18  city                21387 non-null  object
 19  population          21387 non-null  float64
 20  population_density  21387 non-null  float64
 21  prev_sold           21387 non-null  int64
 22  bathroom            21387 non-null  float64
 23  house_age           21387 non-null  int64
 24  renovation_yrs      21387 non-null  int64
 25  renovated_orNot     21387 non-null  int32
 26  sold_month          21387 non-null  int64
 27  basement_orNot      21387 non-null  int64
dtypes: float64(18), int32(1), int64(8), object(1)
memory usage: 4.5+ MB
```

Checking missing values in dataset

check whether while changing datatype we have replaced the special unwanted characters with NaN or not?

In [47]:

```
## missing values before datatype change and string manipulation
dataset_raw.isna().sum()
```

Out[47]:

```
cid                   0
dayhours              0
price                 0
room_bed            108
room_bath           108
living_measure       17
lot_measure          42
ceil                 42
coast                 1
sight                57
condition            57
quality               1
ceil_measure          1
basement              1
yr_built              1
yr_renovated          0
zipcode               0
lat                   0
long                  0
living_measure15    166
lot_measure15        29
furnished            29
total_area           29
dtype: int64
```

In [48]:

```
#for reference for renamed columns
df_renamedCols
```

Out[48]:

	original_Column	renamed_Column
0	cid	house_id
1	dayhours	date
2	room_bed	bedroom
3	room_bath	ratio_bathroom
4	ceil	total_floors
5	coast	seaface
6	sight	sight_viewed
7	quality	quality_grade
8	living_measure	living_area
9	lot_measure	lot_area
10	ceil_measure	floor_area
11	basement	basement_area
12	lat	latitude
13	long	longitude
14	living_measure15	living_area_2015

	original_Column	renamed_Column
15	lot_measure15	lot_area_2015
16	density	population_density

In [49]:
```
## missing values after datatype change and string manipulation

dataset.isna().sum()
```

Out[49]:
```
price                 0
bedroom               0
living_area           0
lot_area              0
total_floors         30
seaface              30
sight_viewed          0
condition             0
quality_grade         0
floor_area            0
basement_area         0
zipcode               0
latitude              0
longitude            34
living_area_2015      0
lot_area_2015         0
furnished             0
total_area           39
city                  0
population            0
population_density    0
prev_sold             0
bathroom              0
house_age             0
renovation_yrs        0
renovated_orNot       0
sold_month            0
basement_orNot        0
dtype: int64
```

Note: We have few missing values across the columns, but the number is small enough. Hence, can be imputed or even can be removed. We will impute this before modeling.

2. Exploratory Data Analysis

Let's do some visual data analysis of the features

Uni-variate Analysis

In [53]:
```
dataset.shape
```

Out[53]:
```
(21387, 28)
```

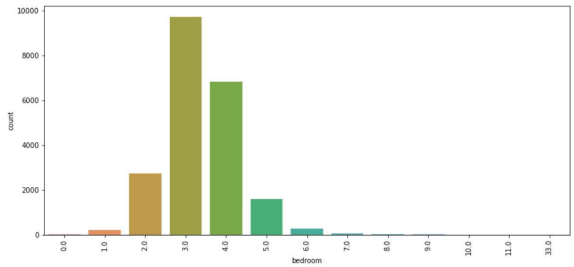
In [54]:
```
dataset.columns
```

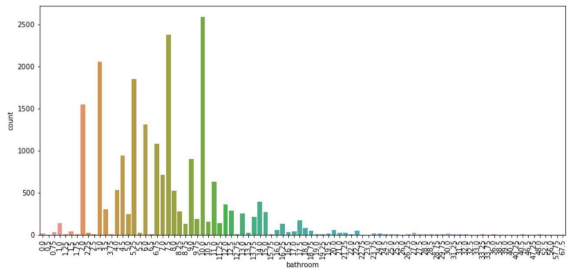
Out[54]:
```
Index(['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'total_area', 'city', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot'],
      dtype='object')
```

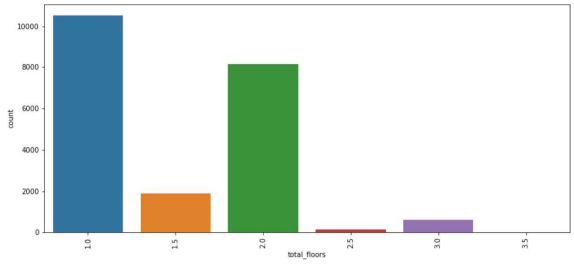
In [55]:
```
features = ['bedroom', 'bathroom', 'total_floors', 'seaface', 'sight_viewed', 'condition', 'quality_grade',
       'furnished', 'prev_sold', 'house_age', 'renovation_yrs', 'city']
list(enumerate(features))
```

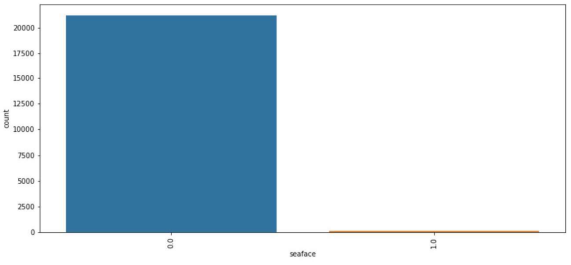
Out[55]:
```
[(0, 'bedroom'),
 (1, 'bathroom'),
 (2, 'total_floors'),
 (3, 'seaface'),
 (4, 'sight_viewed'),
 (5, 'condition'),
 (6, 'quality_grade'),
 (7, 'furnished'),
 (8, 'prev_sold'),
 (9, 'house_age'),
 (10, 'renovation_yrs'),
 (11, 'city')]
```

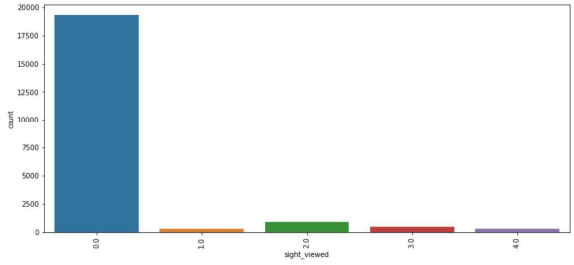
In [56]:
```
#count plot

plt.figure(figsize = (30, 50))
for i in enumerate(features):
    plt.subplot(7, 2,i[0]+1)
    sns.countplot(i[1], data = dataset)
    plt.xticks(rotation = 90)
```

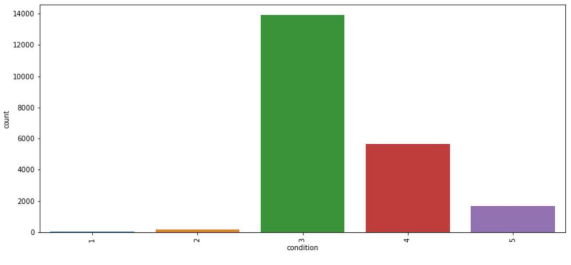


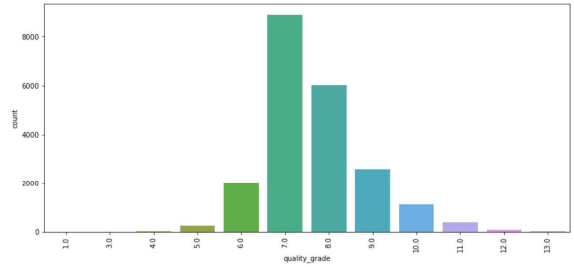


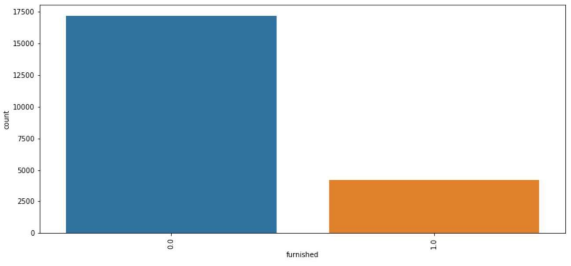


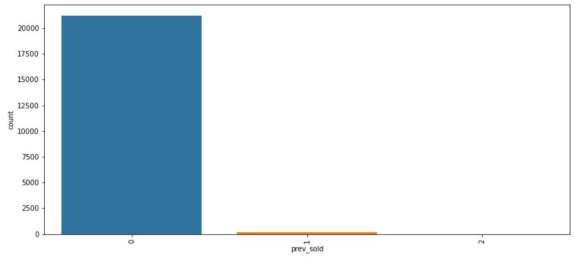


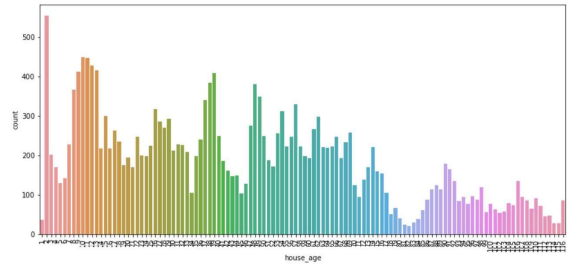


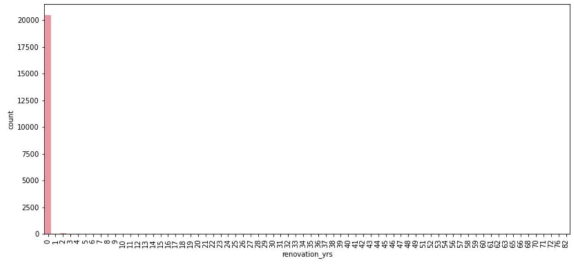


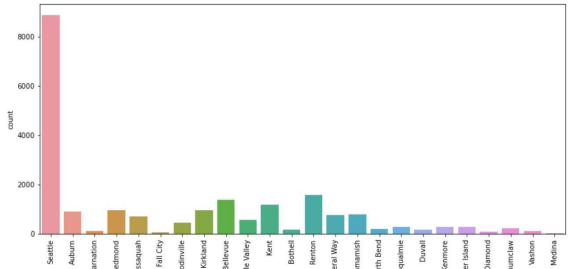






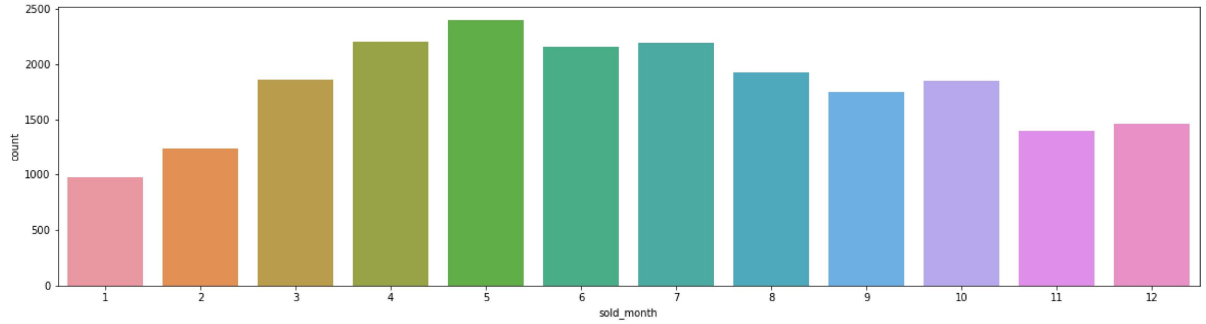






In [57]:

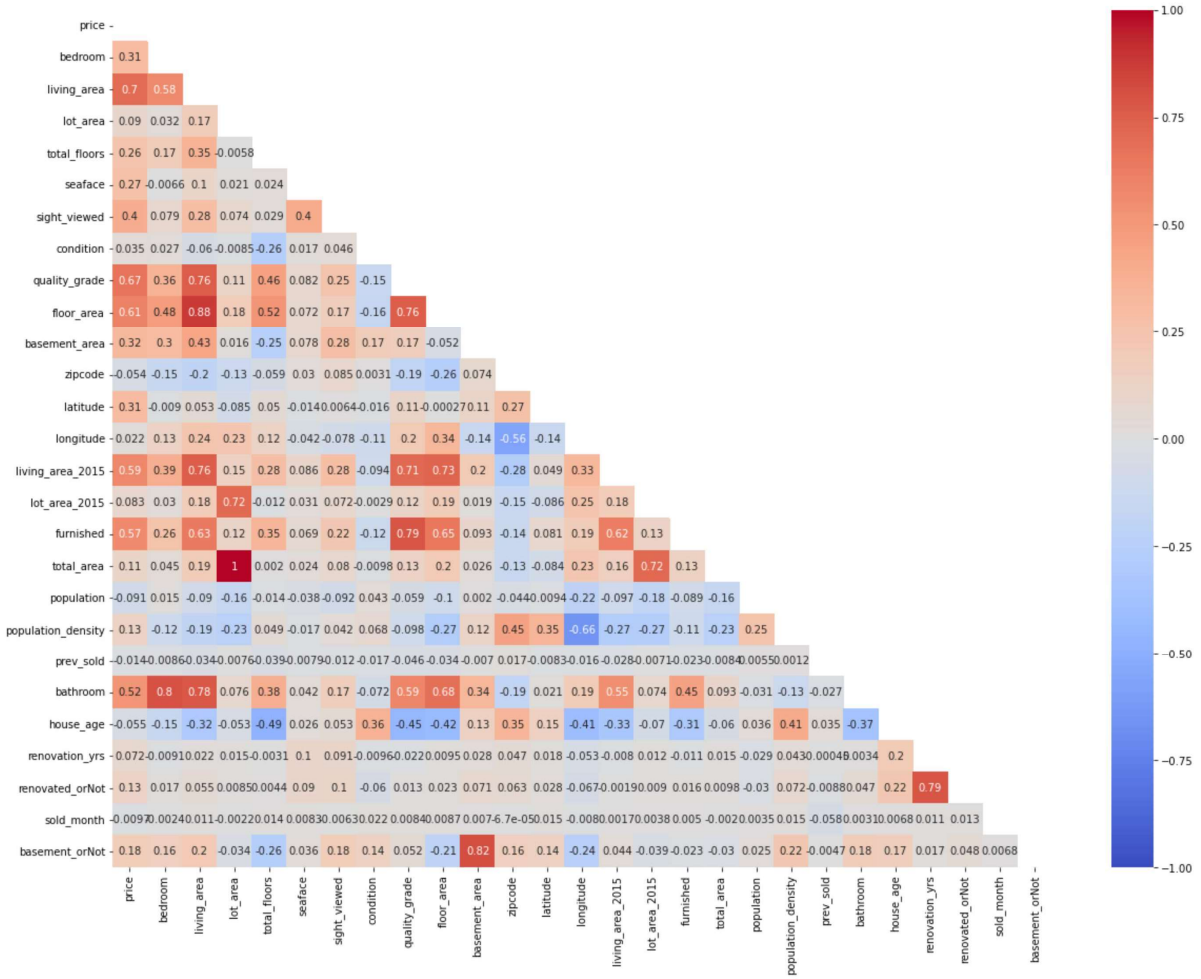
```
plt.figure(figsize = (20, 5))
sns.countplot(data = dataset, x= 'sold_month');
```



Note: The most number of house were sold in the month of April-2015 also in June and July 2014. The least were sold in month of May-2015.

Bi-Variate Analysis

In [59]:
```
#correlation heatmap

plt.figure(figsize = (20,15))
sns.heatmap(dataset.corr(), annot=True,mask=np.triu(dataset.corr()),cmap ='coolwarm', vmin = -1, vmax= 1);
```



Note: The following correlated items can be removed to avoid multi-collinearity.

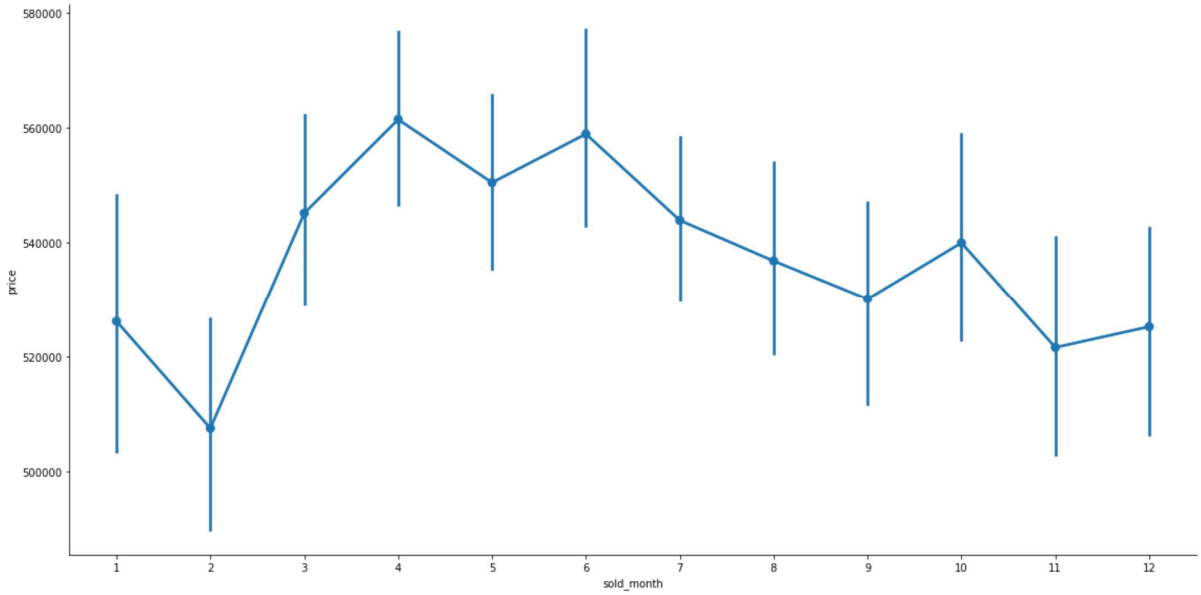
1. total_area is totally correlated with lot_area,
2. floor_area is highly correlated with living_area.
3. quality_grade is correlated with furnished.
4. renovated_orNot is correlated with renovation_yrs.

Analyzing Bivariate for Feature

1. Months vs Price

Variation in Price over the period of Months

In [60]:
```
## At which month price was higher or lesser
sns.factorplot(x='sold_month',y='price',data=dataset, size=8, aspect=2 );
```



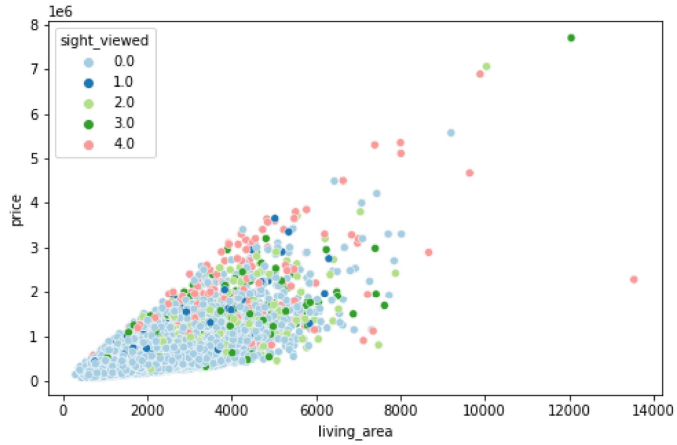
Note: We can see that in the month of Feb-2015, the prices were cheaper, whereas in the month of April. the price goes high.

In [61]:
```
dataset.columns
```

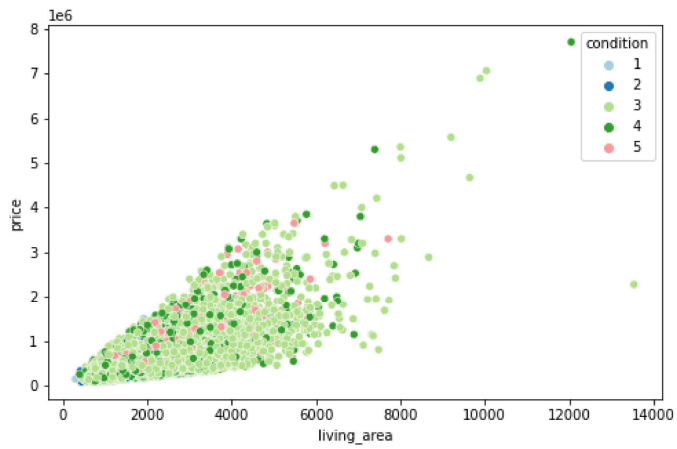
Out[61]:
```
Index(['price', 'bedroom', 'living_area', 'lot_area', 'total_floors',
       'seaface', 'sight_viewed', 'condition', 'quality_grade', 'floor_area',
       'basement_area', 'zipcode', 'latitude', 'longitude', 'living_area_2015',
       'lot_area_2015', 'furnished', 'total_area', 'city', 'population',
       'population_density', 'prev_sold', 'bathroom', 'house_age',
       'renovation_yrs', 'renovated_orNot', 'sold_month', 'basement_orNot'],
      dtype='object')
```

In [62]:
```
plotsizeX=8
plotsizeY=5
```

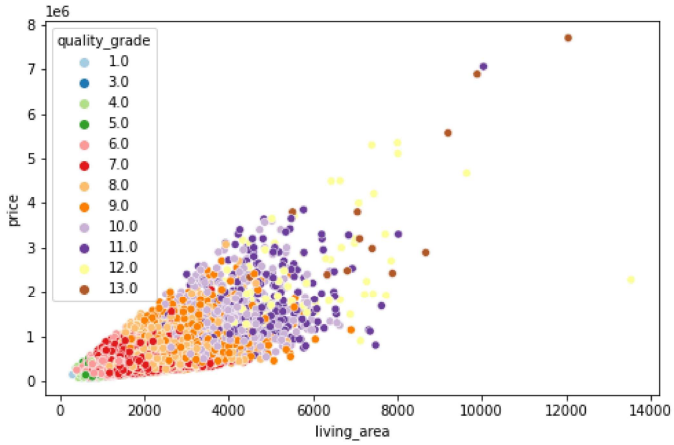
In [63]:
```
plt.figure(figsize=(plotsizeX, plotsizeY))
sns.scatterplot(dataset['living_area'],dataset['price'],hue=dataset['sight_viewed'],palette='Paired',legend='full');
```



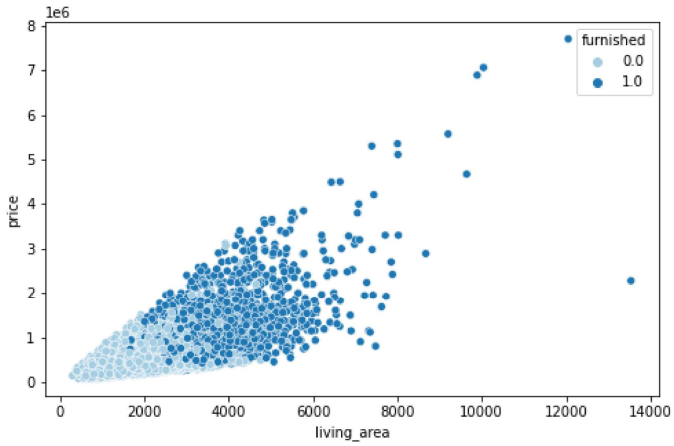
In [64]:
```
plt.figure(figsize=(plotsizeX, plotsizeY))
sns.scatterplot(dataset['living_area'],dataset['price'],hue=dataset['condition'],palette='Paired',legend='full');
```



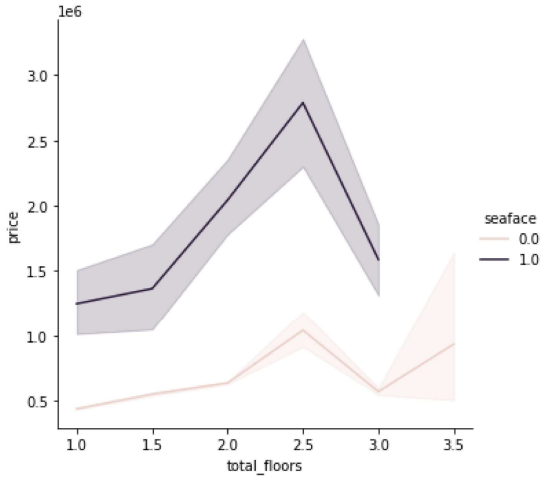
In [65]:
```
plt.figure(figsize=(plotsizeX, plotsizeY))
sns.scatterplot(dataset['living_area'],dataset['price'],hue=dataset['quality_grade'],palette='Paired',legend='full');
```



In [66]:
```
plt.figure(figsize=(plotsizeX, plotsizeY))
sns.scatterplot(dataset['living_area'],dataset['price'],hue=dataset['furnished'],palette='Paired',legend='full');
```

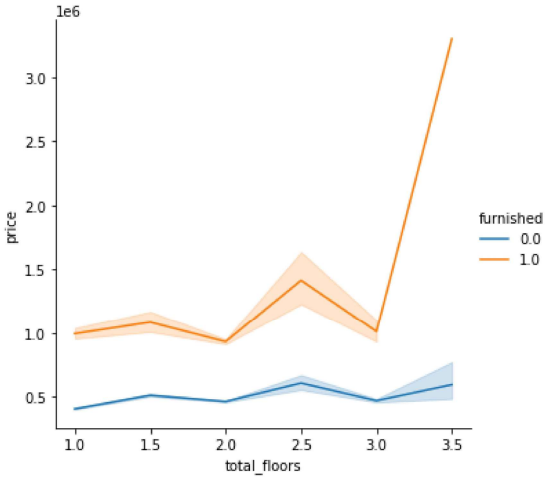


In [67]:
```
sns.relplot(data=dataset, x='total_floors',  y='price', hue='seaface', kind='line');
```



Note: the property with seafacing front are costlier than the house without any seafacing front. But it also shows that how with increase in number of floors the seaface house cost so costlier.

In [68]:
```
sns.relplot(data=dataset, x='total_floors',  y='price', hue='furnished', kind='line');
```



Note: This shows that the furnished house has higher cost and as the number of floors increases beyond 3 then the cost of the house shoots up for furnished house.

In [74]:
```
import missingno as msn
import folium
from folium import plugins
import branca.colormap as cm


m = folium.Map([47 ,-122], zoom_start=5,width="%100",height="%100")
locations = list(zip(dataset.latitude, dataset.longitude))
cluster = plugins.MarkerCluster(locations=locations,popups=dataset["price"].tolist())
m.add_child(cluster)
m
```

Out[74]: Make this Notebook Trusted to load map: File -> Trust Notebook

3. Data Processing

Data processing before Modeling

Imputing the missing values

In [50]:
```
dataset.isnull().sum()
```

Out[50]:
```
price                0
bedroom              0
living_area          0
lot_area             0
total_floors        30
seaface             30
sight_viewed         0
condition            0
quality_grade        0
floor_area           0
basement_area        0
zipcode              0
```