EDA Exploratory Data Analysis and Feature Engineering

1. Importing required libraries 2. Loading the data into the data frame 3. Total number of rows and columns 4. Checking the types of data & null values 5. Finding & Dropping the duplicate rows 6. To find out the unique value of the selected column use unique() function 7. To analysis the outlier whether the row will be removed or only 33 value will be replaced 8. Add more Features 9. Now we have to change the feature from Int to Categorical Features using pandas Categorical() function 10. Statistical information describe() 11. Find out Outliers and deleting outliers 12. To Analyze Continuous Variables Column get the outlier count 13. Data Visualizations 14. Categorical variable analysis 15. Bi-Variate Analysis

1. Importing required libraries

In [1]:
```
# Importing required libraries
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib as mpl
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

2. Loading the data into the data frame

In [2]:
```
dfCity=pd.read_csv("innercity.csv")
dfCity.head()
```

Out[2]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	basement	yr_built	yr_
0	3034200666	20141107T000000	808100	4	3.25	3020	13457	1.0	0	0	...	0	1956	
1	8731981640	20141204T000000	277500	4	2.50	2550	7500	1.0	0	0	...	800	1976	
2	5104530220	20150420T000000	404000	3	2.50	2370	4324	2.0	0	0	...	0	2006	
3	6145600285	20140529T000000	300000	2	1.00	820	3844	1.0	0	0	...	0	1916	
4	8924100111	20150424T000000	699000	2	1.50	1400	4050	1.0	0	0	...	0	1954	

5 rows × 23 columns

3. Total number of rows and columns

In [3]:
```
dfCity.shape
```

Out[3]:
```
(21613, 23)
```

4. Checking the types of data & null values

In [4]:
```
dfCity.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          21613 non-null  int64
 4   room_bath         21613 non-null  float64
 5   living_measure    21613 non-null  int64
 6   lot_measure       21613 non-null  int64
 7   ceil              21613 non-null  float64
 8   coast             21613 non-null  int64
 9   sight             21613 non-null  int64
 10  condition         21613 non-null  int64
 11  quality           21613 non-null  int64
 12  ceil_measure      21613 non-null  int64
 13  basement          21613 non-null  int64
 14  yr_built          21613 non-null  int64
 15  yr_renovated      21613 non-null  int64
 16  zipcode           21613 non-null  int64
 17  lat               21613 non-null  float64
 18  long              21613 non-null  float64
 19  living_measure15  21613 non-null  int64
 20  lot_measure15     21613 non-null  int64
 21  furnished         21613 non-null  int64
 22  total_area        21613 non-null  int64
dtypes: float64(4), int64(18), object(1)
memory usage: 3.8+ MB
```

In [5]:
```
dfCity.isnull().sum()
```

Out[5]:
```
cid                 0
dayhours            0
price               0
room_bed            0
room_bath           0
living_measure      0
lot_measure         0
ceil                0
coast               0
sight               0
condition           0
quality             0
ceil_measure        0
basement            0
yr_built            0
yr_renovated        0
zipcode             0
lat                 0
long                0
living_measure15    0
lot_measure15       0
furnished           0
total_area          0
dtype: int64
```

5. Checking and Dropping the duplicate rows

In [6]:
```
dfCity.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
dup_rows = dfCity[dfCity.duplicated()]
print("Duplicated Rows is",dup_rows.shape)
```

```
Duplicated Rows is (0, 23)
```

6. To find out the unique value of the selected column use unique() function

In [8]:
```
print(*list(dfCity.room_bed.unique()))
```

```
4 3 2 5 6 1 8 33 7 0 9 10 11
```

In [9]:
```
# find out the unique value to make categorical variable
print('Bed Rooms')
print(*list(dfCity.room_bed.unique()))
print('Bath Rooms')
print(*list(dfCity.room_bath.unique()))
print('Coast')
print(*list(dfCity.coast.unique()))
print('sight')
print(*list(dfCity.sight.unique()))
print('condition')
print(*list(dfCity.condition.unique()))
print('quality')
print(*list(dfCity.quality.unique()))
print('basemnet')
print(*list(dfCity.basement.unique()))
print('furnished')
print(*list(dfCity.furnished.unique()))
```

```
Bed Rooms
4 3 2 5 6 1 8 33 7 0 9 10 11
Bath Rooms
3.25 2.5 1.0 1.5 1.75 2.0 2.75 2.25 3.0 4.0 4.5 3.5 5.25 4.75 4.25 5.0 7.75 3.75 0.75 5.5 6.75 1.25 6.25 0.0 5.
75 6.0 0.5 6.5 7.5 8.0
Coast
0 1
sight
0 2 4 3 1
condition
5 3 4 2 1
quality
9 8 6 7 10 11 5 13 12 4 3 1
basemnet
0 800 880 1200 620 1720 540 500 720 390 1800 810 830 700 470 300 960 1450 1570 1600 770 270 160 710 1590 750 89
0 350 570 920 430 1100 550 940 690 840 590 190 760 900 260 100 630 2120 580 740 400 380 530 1000 435 520 290 10
60 490 1070 150 480 120 460 1150 980 140 600 440 660 1030 1050 560 1540 1220 1430 1750 650 200 780 1180 1080 13
50 1290 670 850 340 1460 60 280 330 1260 240 250 360 1950 310 1420 790 1440 210 1250 180 1010 640 1210 730 680
1140 1510 990 170 320 80 1390 2010 910 870 1380 130 860 1120 930 1090 1410 1400 1520 4820 420 1110 1170 820 133
0 1340 2850 1020 2220 1790 1280 220 1270 1230 2030 90 230 450 1490 1300 1370 2550 1310 1500 1760 370 950 145 10
40 1610 510 1160 1320 1130 1830 2060 1190 970 1580 610 1780 2490 1480 70 602 410 1700 1940 1960 143 1240 1900 1
481 1620 1360 1548 110 1840 2310 1710 2070 1852 1690 556 1650 2810 50 1530 40 414 704 2040 1850 1284 1660 1816
1740 1550 2020 1670 2620 1560 2130 10 1810 1860 1890 2390 2090 515 1640 1470 1820 2720 1870 1680 1910 475 2160
2600 1930 225 3260 172 1525 946 784 2330 1630 2050 2200 935 65 906 2000 2240 2590 2080 2170 2180 915 2580 2150
1135 295 2500 1798 2110 1248 1990 265 1024 2730 3500 792 2250 1008 415 588 1281 276 2610 506 2100 768 1730 1245
1920 248 374 1913 283 417 875 3480 235 518 652 2196 516 894 862 1880 2300 1770 2360 243 508 20 266 2190 207 257
0 4130 3000 666 1275 861 274 2400 176 2350
furnished
1 0
```

7. To analysis the outlier whether the row will be removed or only 33 value will be replaced

In [10]:
```
dfCity[dfCity.room_bed==33] # to analysis the outlier whether the row will be removed or only 33 value will be
```

Out[10]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	basement	yr_built	y
750	2402100895	20140625T000000	640000	33	1.75	1620	6000	1.0	0	0	...	580	1947	

1 rows × 23 columns

8. Add more Features

In [11]:
```
# to take the years_sold from dayhours colums
dfCity['yr_sold']=dfCity['dayhours'].apply(lambda x:x[:4]).astype(int)
dfCity.head()
```

Out[11]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	yr_built	yr_renovated
0	3034200666	20141107T000000	808100	4	3.25	3020	13457	1.0	0	0	...	1956	0
1	8731981640	20141204T000000	277500	4	2.50	2550	7500	1.0	0	0	...	1976	0
2	5104530220	20150420T000000	404000	3	2.50	2370	4324	2.0	0	0	...	2006	0
3	6145600285	20140529T000000	300000	2	1.00	820	3844	1.0	0	0	...	1916	0
4	8924100111	20150424T000000	699000	2	1.50	1400	4050	1.0	0	0	...	1954	0

5 rows × 24 columns

9. Now we have to change the feature from Int to Categorical Features using pandas Categorical() function

In [12]:
```
##we have certain features that are displayed as integer, but we know that we need to fix them into categories
dfCity.coast=pd.Categorical(dfCity.coast)
dfCity.condition=pd.Categorical(dfCity.condition)
dfCity.quality=pd.Categorical(dfCity.quality)
dfCity.furnished=pd.Categorical(dfCity.furnished)
dfCity.sight=pd.Categorical(dfCity.sight)
```

In [13]:
```
dfCity.head()
```

Out[13]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	yr_built	yr_renovated
0	3034200666	20141107T000000	808100	4	3.25	3020	13457	1.0	0	0	...	1956	0
1	8731981640	20141204T000000	277500	4	2.50	2550	7500	1.0	0	0	...	1976	0
2	5104530220	20150420T000000	404000	3	2.50	2370	4324	2.0	0	0	...	2006	0
3	6145600285	20140529T000000	300000	2	1.00	820	3844	1.0	0	0	...	1916	0
4	8924100111	20150424T000000	699000	2	1.50	1400	4050	1.0	0	0	...	1954	0

5 rows × 24 columns

In [14]:
```
dfCity.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 24 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         21613 non-null  int64
 4   room_bath        21613 non-null  float64
 5   living_measure   21613 non-null  int64
 6   lot_measure      21613 non-null  int64
 7   ceil             21613 non-null  float64
 8   coast            21613 non-null  category
 9   sight            21613 non-null  category
 10  condition        21613 non-null  category
 11  quality          21613 non-null  category
 12  ceil_measure     21613 non-null  int64
 13  basement         21613 non-null  int64
 14  yr_built         21613 non-null  int64
 15  yr_renovated     21613 non-null  int64
 16  zipcode          21613 non-null  int64
 17  lat              21613 non-null  float64
 18  long             21613 non-null  float64
 19  living_measure15 21613 non-null  int64
 20  lot_measure15    21613 non-null  int64
 21  furnished        21613 non-null  category
 22  total_area       21613 non-null  int64
 23  yr_sold          21613 non-null  int32
dtypes: category(5), float64(4), int32(1), int64(13), object(1)
memory usage: 3.2+ MB
```

10. Statistical information describe()

In [15]:
```
dfCity.describe()
```

Out[15]:

	cid	price	room_bed	room_bath	living_measure	lot_measure	ceil	ceil_measure	basement
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000
mean	4.580302e+09	5.401822e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	1788.390691	291.509045
std	2.876566e+09	3.673622e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	828.090978	442.575043
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	290.000000	0.000000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	1190.000000	0.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	1560.000000	0.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	2210.000000	560.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	9410.000000	4820.000000

In [16]:
```
dfCity.describe(include='all')  #include non-numeric cloumn also
```

Out[16]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	s
count	2.161300e+04	21613	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.0	216
unique	NaN	372	NaN	NaN	NaN	NaN	NaN	NaN	2.0	
top	NaN	20140623T000000	NaN	NaN	NaN	NaN	NaN	NaN	0.0	
freq	NaN	142	NaN	NaN	NaN	NaN	NaN	NaN	21450.0	194
mean	4.580302e+09	NaN	5.401822e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	NaN	
std	2.876566e+09	NaN	3.673622e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	NaN	
min	1.000102e+06	NaN	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	NaN	
25%	2.123049e+09	NaN	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	NaN	
50%	3.904930e+09	NaN	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	NaN	
75%	7.308900e+09	NaN	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	NaN	
max	9.900000e+09	NaN	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	NaN	

11 rows × 24 columns

In [17]:
```
dfCity.describe(include='category')
```

Out[17]:

	coast	sight	condition	quality	furnished
count	21613	21613	21613	21613	21613
unique	2	5	5	12	2
top	0	0	3	7	0
freq	21450	19489	14031	8981	17362

In [18]:
```
dfCity.describe(include='object')
```

Out[18]:

	dayhours
count	21613
unique	372
top	20140623T000000
freq	142

11. Find out Outliers

In [19]:
```
#we know Q3 AND Q1 AND IQR=Q3-Q1, any data point which is less than Q1-1.5IQR or Q3+1.5IQR are consider as outl
# Analysis on Room_Bed feature
Q1=dfCity.room_bed.quantile(.25)
Q3=dfCity.room_bed.quantile(.75)
IQR=Q3-Q1
lower_limit=Q1-(1.5*IQR)
upper_limit=Q3+(1.5*IQR)
print("Min Value",dfCity.room_bed.min())
print("Max Value ",dfCity.room_bed.max())
print("Q1 ",Q1)
print("Q3 ",Q3)
print("IQR ",IQR)
print('lower_limit',lower_limit)
print('upper_limit',upper_limit)
```

```
Min Value 0
Max Value  33
Q1  3.0
Q3  4.0
IQR  1.0
lower_limit 1.5
upper_limit 5.5
```

In [20]:
```
## Analysis of continous variables
def findoutliers(column):
    outliers=[]
    Q1=column.quantile(.25)
    Q3=column.quantile(.75)
    IQR=Q3-Q1
    lower_limit=Q1-(1.5*IQR)
    upper_limit=Q3+(1.5*IQR)
    for out1 in column:
        if out1>upper_limit or out1 <lower_limit:
            outliers.append(out1)

    return np.array(outliers)
```

In [21]:
```
findoutliers(dfCity.room_bed)
```

Out[21]:

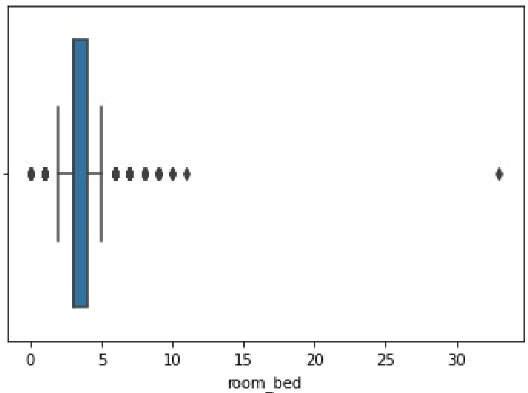
```
array([ 6,  6,  6,  6,  1,  1,  6,  6,  8,  1,  1,  1,  6,  1, 33,  6,  1,
        1,  6,  1,  6,  6,  1,  1,  1,  1,  1,  7,  1,  1,  1,  1,  8,  6,
        1,  1,  6,  1,  1,  6,  0,  6,  8,  9,  6,  7,  6,  6,  6,  0,  6,
        1,  6,  1,  1,  1,  7,  6,  6,  6,  8,  1,  6,  8,  6,  1,  6,  1,
        6,  1,  6,  6,  7,  6,  1,  1,  6,  6,  6,  1,  6,  1,  1,  6,  6,
        6,  6,  7,  8,  1,  6,  1,  6,  6,  6,  6,  6,  6,  6,  6,  1,  1,
        6,  6,  6,  1,  6,  6,  6,  1,  1,  6,  1,  6,  1,  6,  6,  6,  6,
        8,  1,  6,  6,  6,  6,  1,  7,  1,  6,  1,  6,  6,  6,  6,  1, 10,
        6,  6,  1,  6,  6,  6,  1,  6,  6,  6,  1,  6,  6,  1,  7,  6,  1,
        1,  6,  7,  6,  6,  1,  6,  1,  1,  1,  6,  6,  6,  6,  6,  6,  6,
        1,  6,  7,  6,  6,  1,  7,  6,  7,  1,  6,  7,  1,  1,  1,  0,  6,
        9,  7,  6,  6,  6,  6,  8,  1,  0,  6,  6,  1,  1,  6,  1,  1,  1,
        6,  6,  1,  6,  1,  1,  7,  6,  1,  1,  6,  6,  1,  6,  1,  6,  6,
        7,  6,  6,  6,  6,  6,  6,  6,  1,  1,  0,  1,  6,  1,  6,  7,  1,
        1,  1,  1,  6,  1,  6,  6,  1,  6,  6,  6,  6,  1,  1,  6,  1,  6,
        6,  6,  6,  7,  1,  6,  6,  6,  6,  6,  6,  6,  6,  1,  1,  1,  6,
        6,  0,  1,  6,  1,  7,  1,  1,  1,  6,  1,  6,  1,  6,  6,  6,  6,
        9,  6,  1,  6,  6,  1,  1,  8,  6,  1,  6,  7,  1,  6,  6,  6,  6,
        6,  1,  6,  1,  6,  1,  6,  1, 10,  1,  7,  1,  6,  6,  1,  6,  6,
        1,  1,  0,  6,  6,  6,  6,  6,  6,  6,  8,  7,  6,  6,  6,  1,  6,
        1,  1,  1,  6,  6,  1,  6,  1,  1,  7,  1,  1,  1,  6,  8,  6,  1,
        1,  1,  9,  6,  1,  6,  0,  6,  6,  6, 11,  1,  6,  6,  6,  6,  1,
        6,  1,  6,  0,  6,  6,  7,  7,  1,  6,  1,  6,  6,  1,  6,  7,  0,
        7,  0,  1,  1,  1,  6,  6,  7,  6,  6,  1,  1,  1,  6,  1,  6,  1,
        6,  1,  7,  6,  6,  1,  7,  1,  1,  6,  6,  6,  6,  1,  1,  6,  6,
        8,  6,  6,  1, 10,  1,  1,  7,  1,  6,  1,  6,  1,  6,  6,  6,  1,
        1,  1,  1,  6,  6,  1,  6,  1,  6,  6,  6,  6,  6,  6,  7,  6,  6,
        6,  1,  1,  1,  1,  6,  6,  1,  6,  1,  1,  1,  6,  1,  6,  1,  6,
        6,  7,  1,  6,  1,  0,  1,  1,  6,  7,  8,  6,  1,  7,  1,  1,  1,
        6,  6,  6,  6,  6,  1,  6,  6,  7,  6,  6,  6,  1,  1,  7,  6,  6,
        1,  6,  6,  1,  1,  1,  6,  1,  6,  7,  9,  6,  1,  1,  6,  6,  1,
        6,  9,  0,  1,  6,  6,  7,  6,  1,  1,  1,  1,  1,  6,  6,  1,  6,
        1,  6])
```

In [22]:

```python
sn.boxplot(dfCity.room_bed)
```

Out[22]:

```
<AxesSubplot:xlabel='room_bed'>
```



12. To Analyze Continuous Variables Column get the outlier count

In [23]:

```python
print(len(findoutliers(dfCity.room_bed))) #no of rows having outlier
```

546

In [24]:

```python
print(len(findoutliers(dfCity.room_bath)))
```

571

In [25]:

```python
print(len(findoutliers(dfCity.living_measure)))
```

572

13. Data Visualizations:

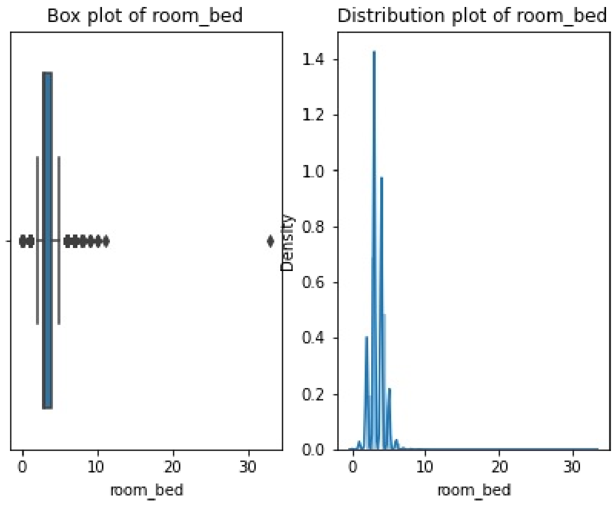
In [26]:

```python
def plotchart(col):
    fix, (ax1,ax2) =plt.subplots(1,2,figsize=(7,5))
    sn.boxplot(col, orient='v',ax=ax1)
    ax1.set_ylabel=col.name
    ax1.set_title('Box plot of {}'.format(col.name))
    sn.distplot(col,ax=ax2)
    ax2.set_title('Distribution plot of {}'.format(col.name))


def analysis_column(col):
    print('count of outlier ', len(findoutliers(col)))
    print('Mean ',format(col.mean()))
    print('Median ',format(col.median()))
    print('Missing values',format(col.isnull().sum()))
    print('% of Missing values',format(round(100*(col.isnull().sum()/len(col)),2)))
```

```
    plotchart(col)
```

In [27]:
```
analysis_column(dfCity.room_bed)
```

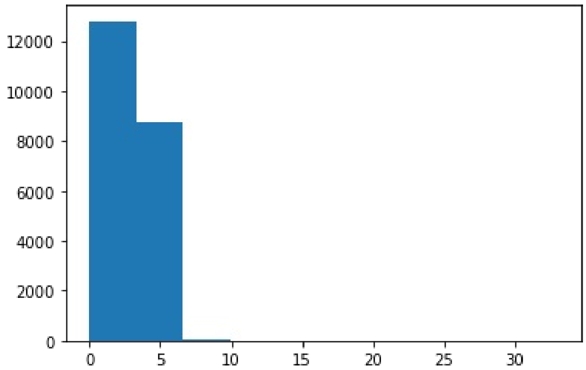
```
count of outlier  546
Mean  3.37084162309721
Median  3.0
Missing values 0
% of Missing values 0.0
```



Analyze individual column:

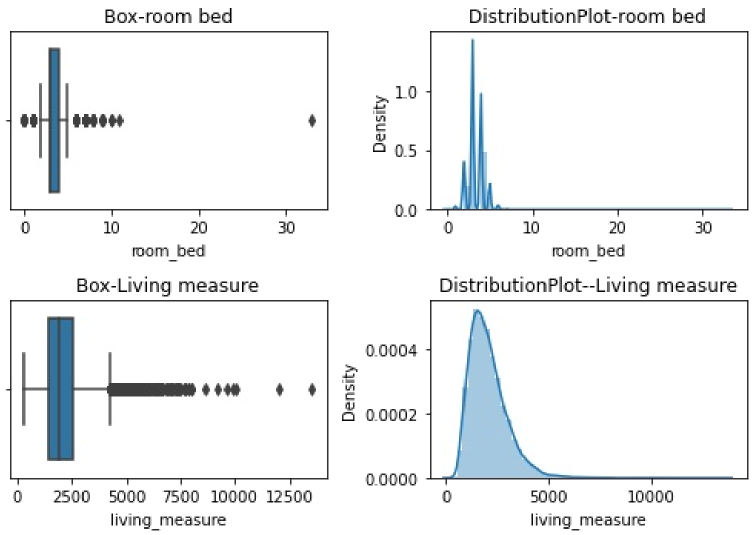
In [28]:
```
import matplotlib.pyplot as plt
plt.hist(dfCity.room_bed)
#dfCity.room_bed.hist()
```

Out[28]:
```
(array([1.2796e+04, 8.7550e+03, 5.7000e+01, 4.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00]),
 array([ 0. ,  3.3,  6.6,  9.9, 13.2, 16.5, 19.8, 23.1, 26.4, 29.7, 33. ]),
 <BarContainer object of 10 artists>)
```



analyze 2 columns in a figure:

In [29]:
```
fig, axes = plt.subplots(nrows=2, ncols=2,figsize=(7,5))
axes[0,0].set_title('Box-room bed')
axes[0,1].set_title('DistributionPlot-room bed')
axes[1,0].set_title('Box-Living measure')
axes[1,1].set_title('DistributionPlot--Living measure')

sn.boxplot(dfCity.room_bed, orient='v',ax=axes[0,0])
sn.distplot(dfCity.room_bed,ax=axes[0,1])
sn.boxplot(dfCity.living_measure, orient='v',ax=axes[1,0])
sn.distplot(dfCity.living_measure,ax=axes[1,1])

fig.tight_layout();  # this reduces the space in between the subplots
```



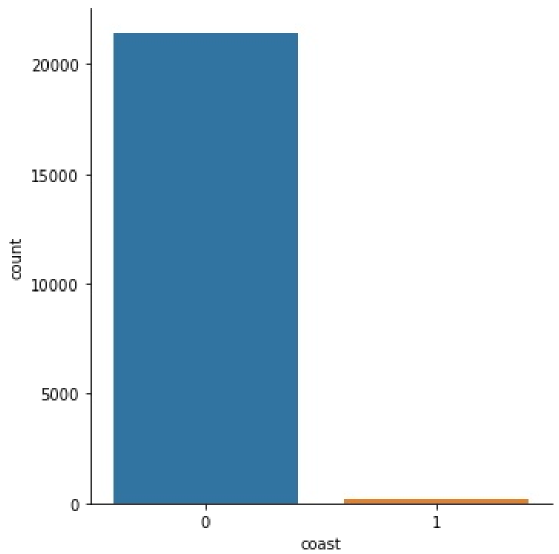
14. Caegorical Variable Analysis

In [30]:
```
dfCity.coast.value_counts()
```

Out[30]:
```
0    21450
1      163
Name: coast, dtype: int64
```

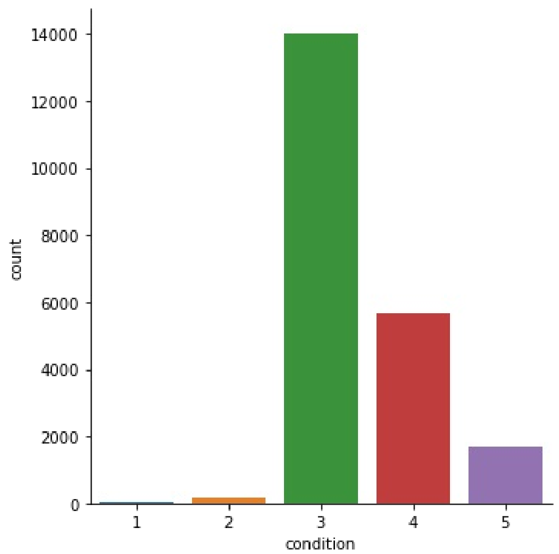
In [31]:
```
sn.factorplot('coast',data=dfCity,kind='count')
```

Out[31]:
```
<seaborn.axisgrid.FacetGrid at 0x2133f9b82b0>
```



In [32]:
```
sn.factorplot('condition',data=dfCity,kind='count')
```

Out[32]:
```
<seaborn.axisgrid.FacetGrid at 0x2133f848fd0>
```



In [33]:
```
dfCity.condition.value_counts()
```

```
# since condition 1&2 the count is less then we can merge these 2 into 1 column same 4 & 5 is also combined
#that way we can reduced the level of condition
```

Out[33]:

```
3    14031
4     5679
5     1701
2      172
1       30
Name: condition, dtype: int64
```

In [35]:

```
dfCity.quality.value_counts()
```

Out[35]:

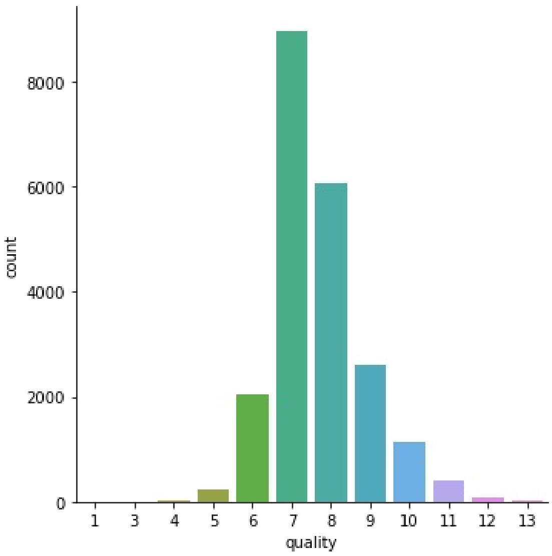
```
7     8981
8     6068
9     2615
6     2038
10    1134
11     399
5      242
12      90
4       29
13      13
3        3
1        1
Name: quality, dtype: int64
```

In [34]:

```
sn.factorplot('quality',data=dfCity,kind='count')
# so here 0-5 merged into a level, and 10-13 also merged into another level
```

Out[34]:

```
<seaborn.axisgrid.FacetGrid at 0x213402ece50>
```

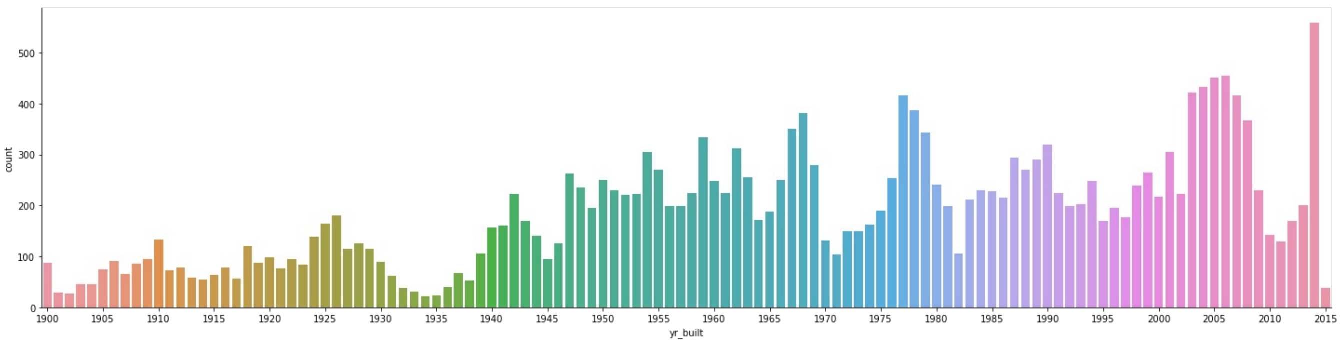


In [36]:

```
pl = sn.factorplot('yr_built',data=dfCity, aspect=4,kind='count')
pl.set_xticklabels(step=5)
```

Out[36]:

```
<seaborn.axisgrid.FacetGrid at 0x2133fa43790>
```

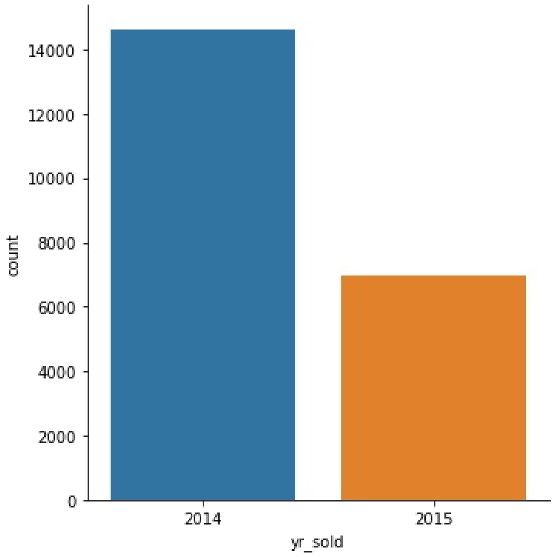


In [37]:

```
sn.factorplot('yr_sold',data=dfCity,kind='count')
```

Out[37]:

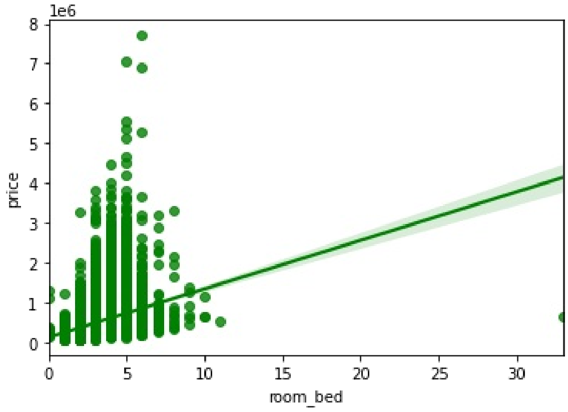
```
<seaborn.axisgrid.FacetGrid at 0x2133f3faac0>
```



15. Bivariate Analysis

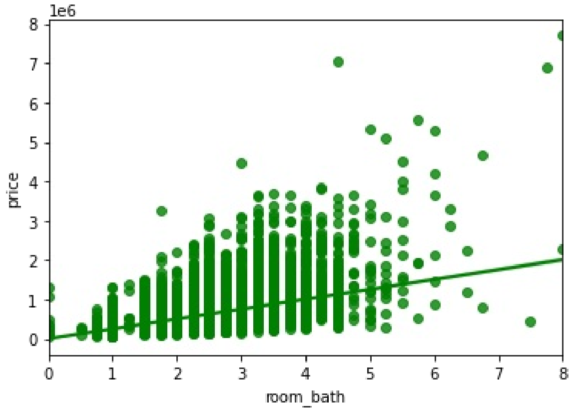
In [38]:
```
# plots between independent variables and price that is target
sn.regplot(x=dfCity.room_bed, y=dfCity.price, color='g')
```

Out[38]: <AxesSubplot:xlabel='room_bed', ylabel='price'>



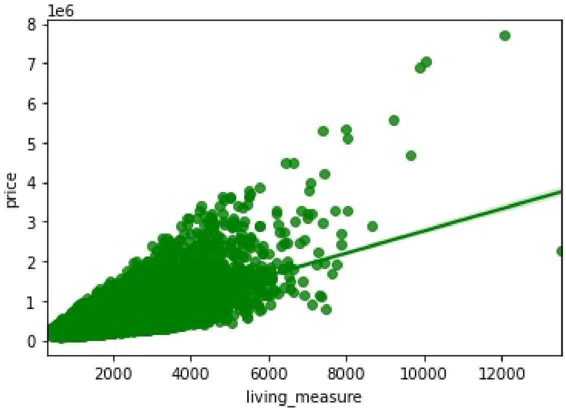
In [39]:
```
sn.regplot(x=dfCity.room_bath, y=dfCity.price, color='g')
```

Out[39]: <AxesSubplot:xlabel='room_bath', ylabel='price'>



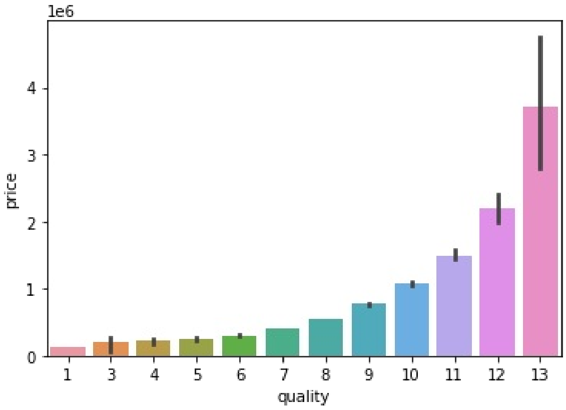
In [40]:
```
from scipy.stats import spearmanr
sn.regplot(x=dfCity.living_measure, y=dfCity.price, color='g')
print(spearmanr(dfCity.living_measure,dfCity.price))  # find the co-relation between living measure and price
# p-value means
```

```
SpearmanrResult(correlation=0.6441923326759279, pvalue=0.0)
```



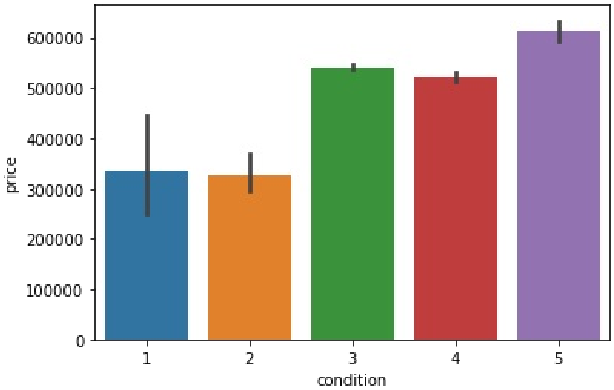
In [41]:
```
#bivariate analysis for independent variable being a category and dependent variable being a number
sn.barplot(x=dfCity.quality,y=dfCity.price)
# mean value for each quality
```

Out[41]:
```
<AxesSubplot:xlabel='quality', ylabel='price'>
```



In [42]:
```
sn.barplot(x=dfCity.condition,y=dfCity.price)
# mean value for each condition value
```

Out[42]:
```
<AxesSubplot:xlabel='condition', ylabel='price'>
```



Feature Selection

1. Univariate Selection

In [43]:
```
dfCity.head()
```

Out[43]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	yr_built	yr_renovated
0	3034200666	20141107T000000	808100	4	3.25	3020	13457	1.0	0	0	...	1956	0
1	8731981640	20141204T000000	277500	4	2.50	2550	7500	1.0	0	0	...	1976	0
2	5104530220	20150420T000000	404000	3	2.50	2370	4324	2.0	0	0	...	2006	0
3	6145600285	20140529T000000	300000	2	1.00	820	3844	1.0	0	0	...	1916	0
4	8924100111	20150424T000000	699000	2	1.50	1400	4050	1.0	0	0	...	1954	0

5 rows × 24 columns

In [44]:
```
dfCity.shape
```

Out[44]: (21613, 24)

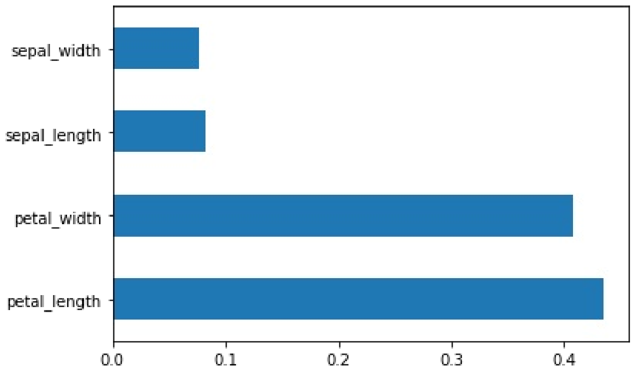
In [45]:
```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

X = dfCity.iloc[:,[0,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,19,20,21,22,23]]  #independent columns
y = dfCity.iloc[:,2]    #target column i.e price range
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=5)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(5,'Score'))  #print 10 best features
```

```
            Specs         Score
0             cid  6.902021e+12
4     lot_measure  3.119557e+08
19     total_area  2.831815e+08
17  lot_measure15  1.579147e+08
13   yr_renovated  7.053617e+06
```

2. Feature Importance using ExtraTrees Classifier

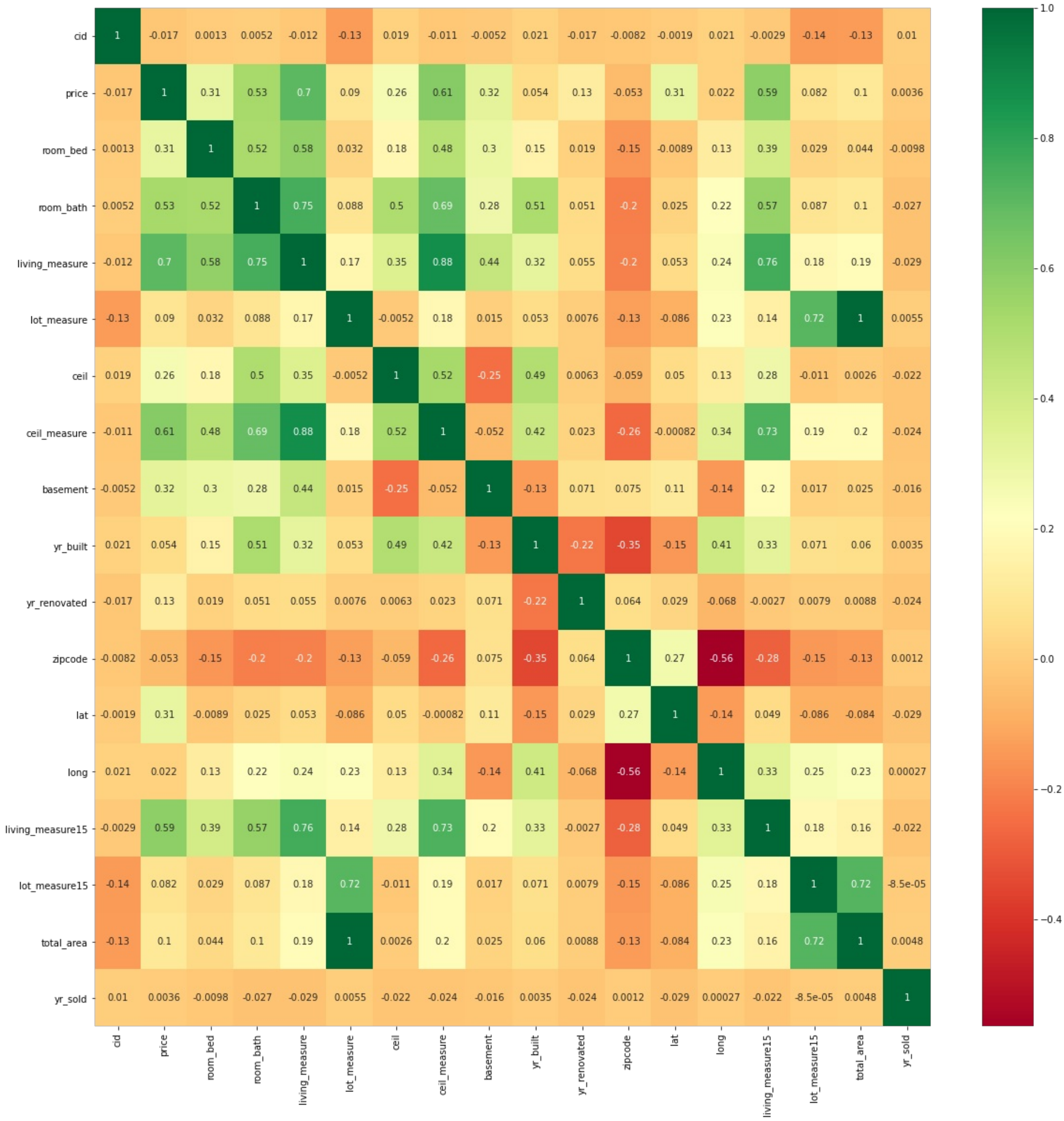
In [47]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('iris(1).csv')
X = data.iloc[:,:-1]
y = data.iloc[:,4:5]

from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_)
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(4).plot(kind='barh')
plt.show()
```

```
[0.08176402 0.07704715 0.43432276 0.40686607]
```



3.Correlation Matrix with Heatmap

In [48]:
```
import pandas as pd
import numpy as np
import seaborn as sns
X = dfCity.iloc[:,[0,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,19,20,21,22,23]]  #independent columns
y = dfCity.iloc[:,2]
#get correlations of each features in dataset
corrmat = dfCity.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(dfCity[top_corr_features].corr(),annot=True,cmap="RdYlGn")
#g=sns.heatmap(dfCity[top_corr_features].corrwith(dfCity['price']),annot=True,cmap="RdYlGn")
```



In [49]:
```
dfCity[top_corr_features].corrwith(dfCity.price)
```

Out[49]:
```
cid                 -0.016797
price                1.000000
room_bed             0.308338
room_bath            0.525134
living_measure       0.702044
lot_measure          0.089655
ceil                 0.256786
ceil_measure         0.605566
basement             0.323837
yr_built             0.053982
yr_renovated         0.126442
zipcode             -0.053168
lat                  0.306919
long                 0.021571
living_measure15     0.585374
lot_measure15        0.082456
total_area           0.104796
yr_sold              0.003554
dtype: float64
```

In [ ]: