1. For each of the data sets, do the following: Create a Jupyter notebook to read the data set

In [2]:
```
pip install pyppeteer
```

```
Requirement already satisfied: pyppeteer in /Users/yuxuanzhang/opt/an
aconda3/lib/python3.9/site-packages (1.0.2)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in /Users/yuxuanzh
ang/opt/anaconda3/lib/python3.9/site-packages (from pyppeteer) (4.64.
1)
Requirement already satisfied: certifi>=2021 in /Users/yuxuanzhang/op
t/anaconda3/lib/python3.9/site-packages (from pyppeteer) (2022.9.24)
Requirement already satisfied: websockets<11.0,>=10.0 in /Users/yuxua
nzhang/opt/anaconda3/lib/python3.9/site-packages (from pyppeteer) (10
.4)
Requirement already satisfied: importlib-metadata>=1.4 in /Users/yuxu
anzhang/opt/anaconda3/lib/python3.9/site-packages (from pyppeteer) (4
.11.3)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in /Users/yuxua
nzhang/opt/anaconda3/lib/python3.9/site-packages (from pyppeteer) (1.
26.11)
Requirement already satisfied: pyee<9.0.0,>=8.1.0 in /Users/yuxuanzha
ng/opt/anaconda3/lib/python3.9/site-packages (from pyppeteer) (8.2.2)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in /Users/yuxuan
zhang/opt/anaconda3/lib/python3.9/site-packages (from pyppeteer) (1.4
.4)
Requirement already satisfied: zipp>=0.5 in /Users/yuxuanzhang/opt/an
aconda3/lib/python3.9/site-packages (from importlib-metadata>=1.4->py
ppeteer) (3.8.0)
Note: you may need to restart the kernel to use updated packages.
```

In [19]:
```
!export PATH=/Library/TeX/texbin:$PATH
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
king_county= pd.read_csv('kc_house_data.csv')
diamond = pd.read_csv('diamonds.csv')
king_county
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	flo
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	
...	...	...	...	...	...	...	...	
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	

21613 rows × 21 columns

In [3]:
```
king_county.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_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  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [4]:
```
king_county.isnull().sum()
```

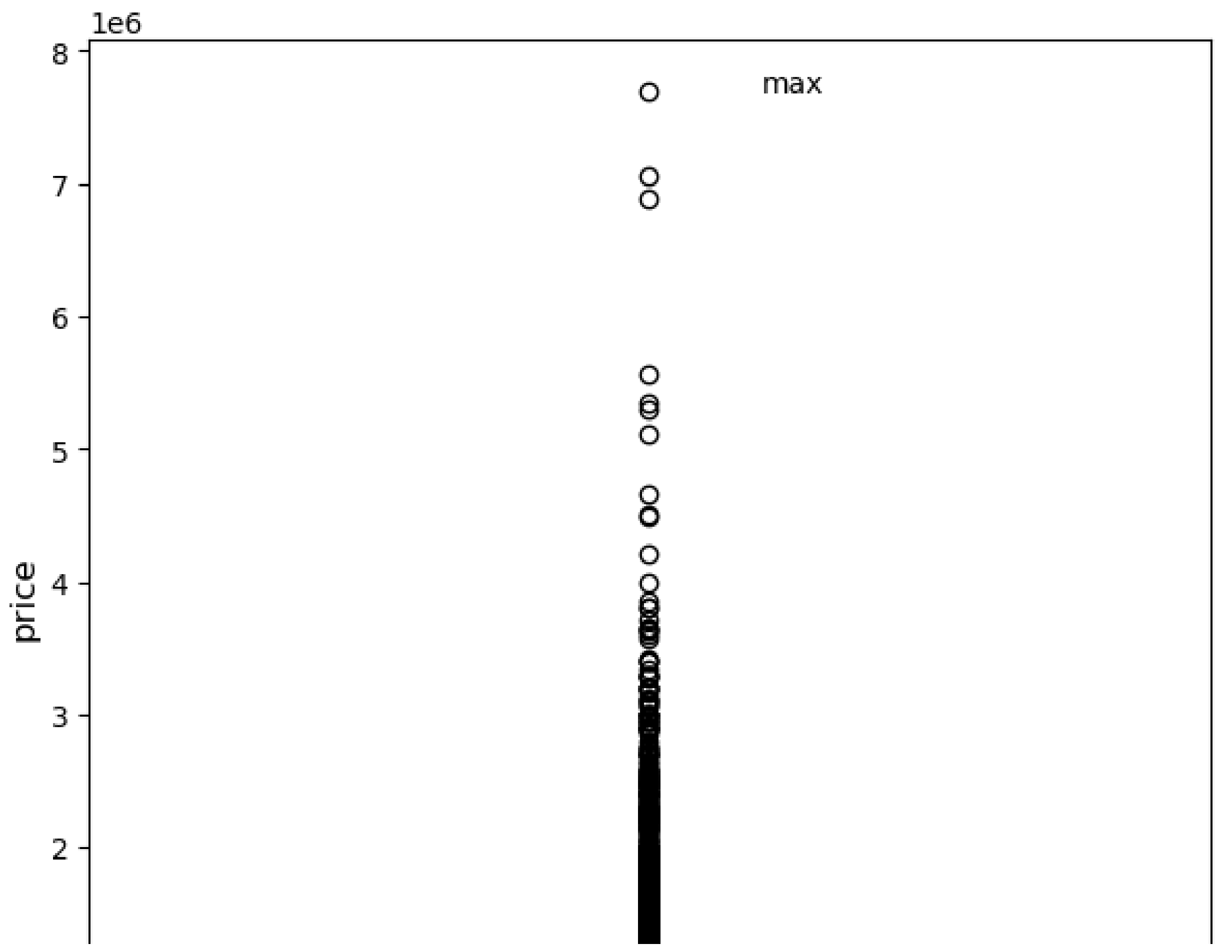
Out[4]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

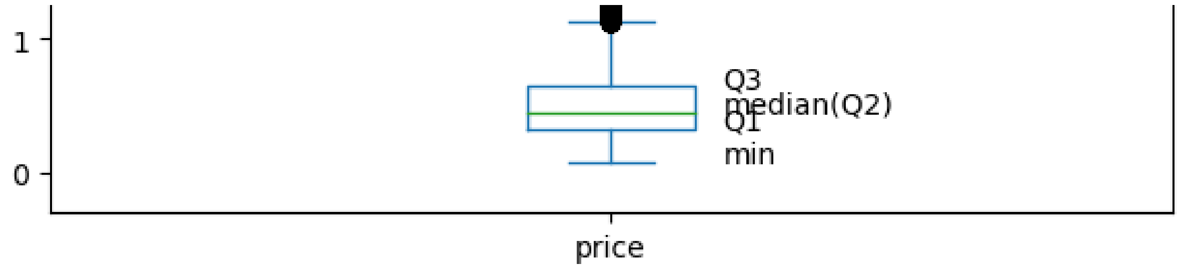
Lucky enough, this dataset doesn't have the null values therefore does not require much cleaning or filling with made up values

Exercise 1:

In [5]:

```
#I want to check the overall price of the house of 14 and 15, looking

price_analysis= king_county['price']

price_analysis.plot(kind='box', subplots=True, figsize=(7,7),layout=(1
plt.ylabel('price', fontsize=12)


plt.text(x = 1.1, y=king_county['price'].min(), s='min')
plt.text(x = 1.1, y= king_county['price'].max(), s='max')

plt.text(x = 1.1, y=king_county['price'].quantile(0.25), s='Q1')
plt.text(x = 1.1, y=king_county['price'].median(), s='median(Q2)')
plt.text(x = 1.1, y=king_county['price'].quantile(0.75), s='Q3')

plt.show()


print('The minimum value of house price from 14-15:',(price_analysis.m
print('The mean value of house price from 14-15:',(price_analysis.mear
print('The maximum value of house price from 14-15:', (price_analysis.
```






```
The minimum value of house price from 14-15: 75000.0
The mean value of house price from 14-15: 540088.1417665294
The maximum value of house price from 14-15: 7700000.0
```

In [6]:

```
# There are huge price discrepancy between prices reflected by the box
```

I want to assess whether the sqft feet has something to do with price, as one potential drives for prices. therefore I made a new columns of total sqft of each house sold (not just inside the house, but also areas such as lot...) in 2014 to make a connection with the price

In [7]:

```
king_county['total_sqft']=king_county['sqft_living']+king_county['sqft
king_county.head()
```

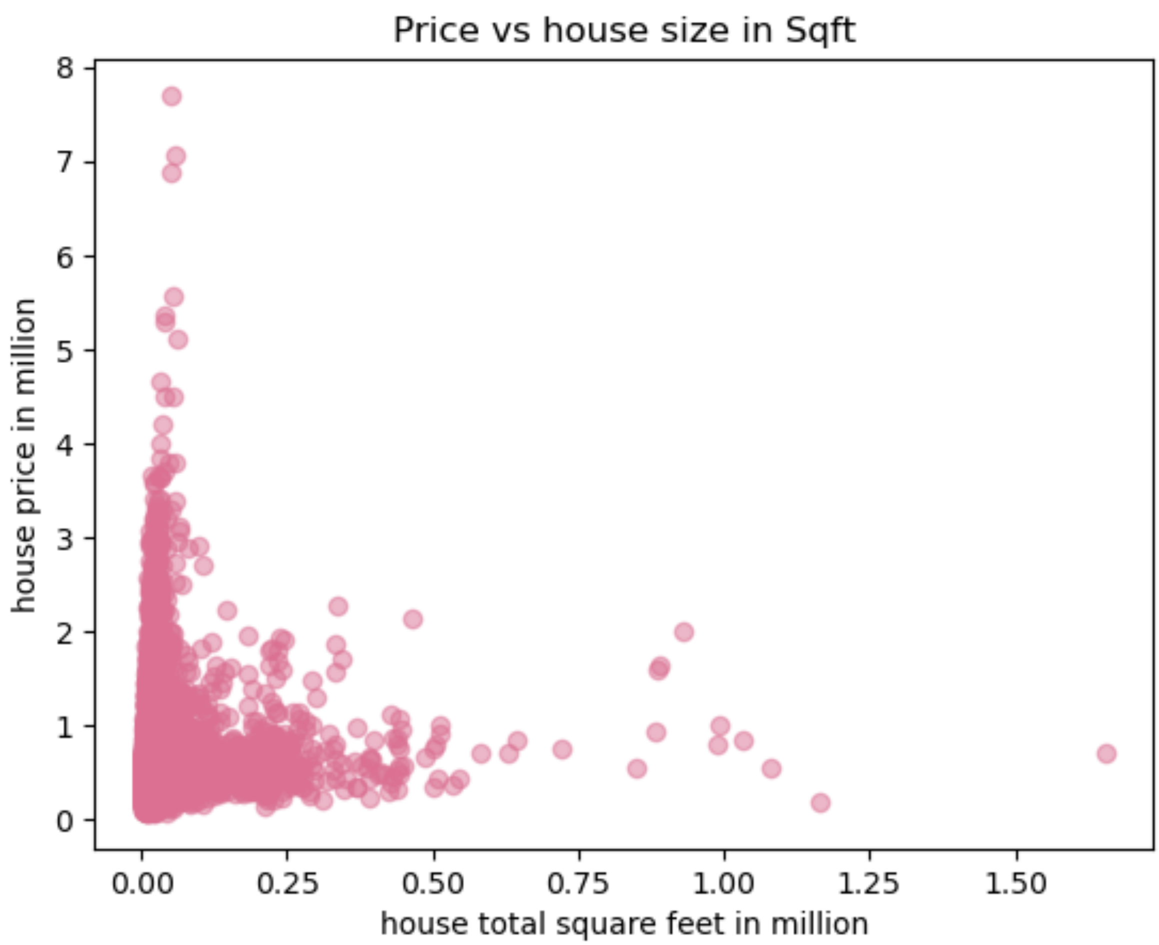
Out[7]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0

5 rows × 22 columns

In [8]:
```
plt.subplot(111)

plt.scatter(x=king_county['total_sqft']/1000000,y=king_county['price']
plt.xlabel('house total square feet in million')
plt.ylabel('house price in million')
plt.title('Price vs house size in Sqft')
```

Out[8]: Text(0.5, 1.0, 'Price vs house size in Sqft')

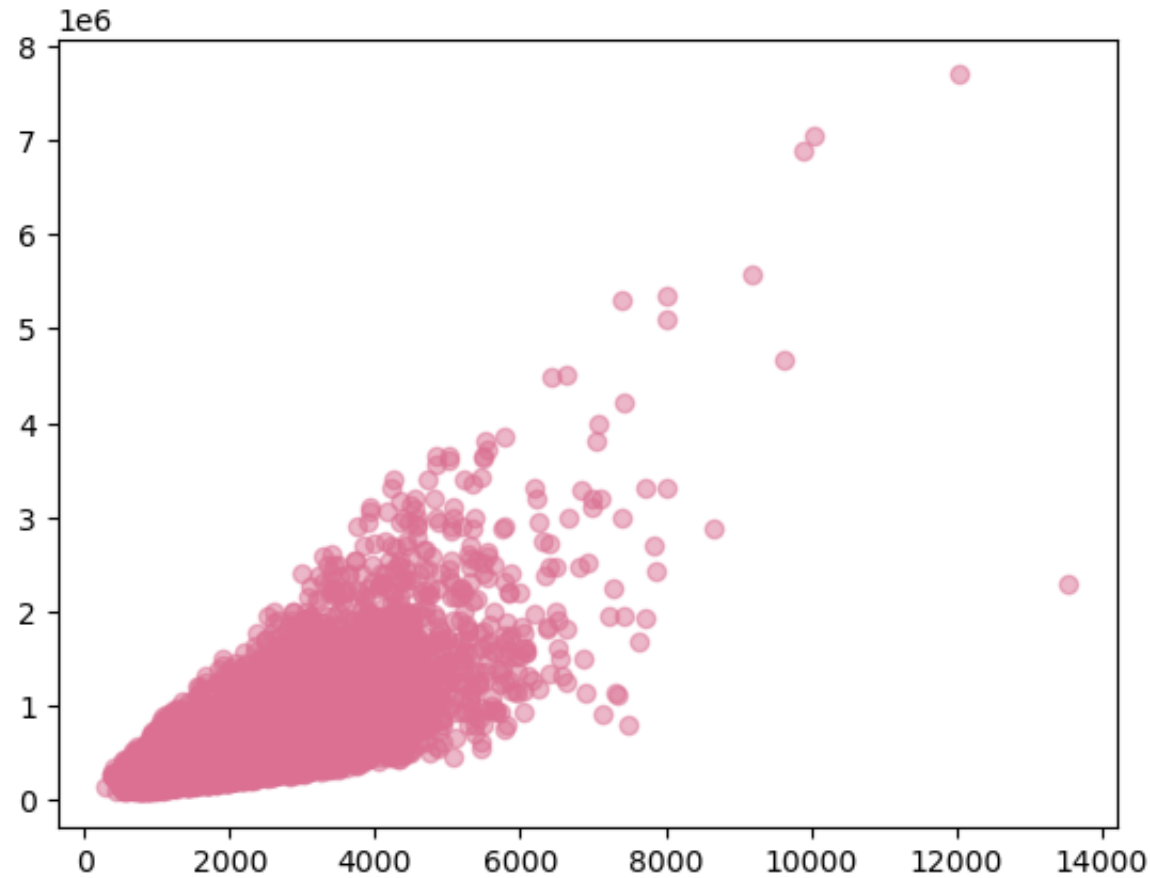


As the scatterplot indicated, house price dramatically decrease between 0 to 0.25 million Square root.Therefore size is a big factor determining price, However it is important to consider most of people buy smaller houses. Therefore some small houses has higher price compared to (for example) the ones at 1.25 million. This happenes due to other factors like location as well. Therefore this graph alone cannot explain why some bigger size houses has lower prices

I Wanted to see which types of apartment were popular

In [9]:
```
plt.subplot(111)

plt.scatter(x=king_county['sqft_living'],y=king_county['price'], alpha
```

Out[9]: <matplotlib.collections.PathCollection at 0x7fda113dec70>

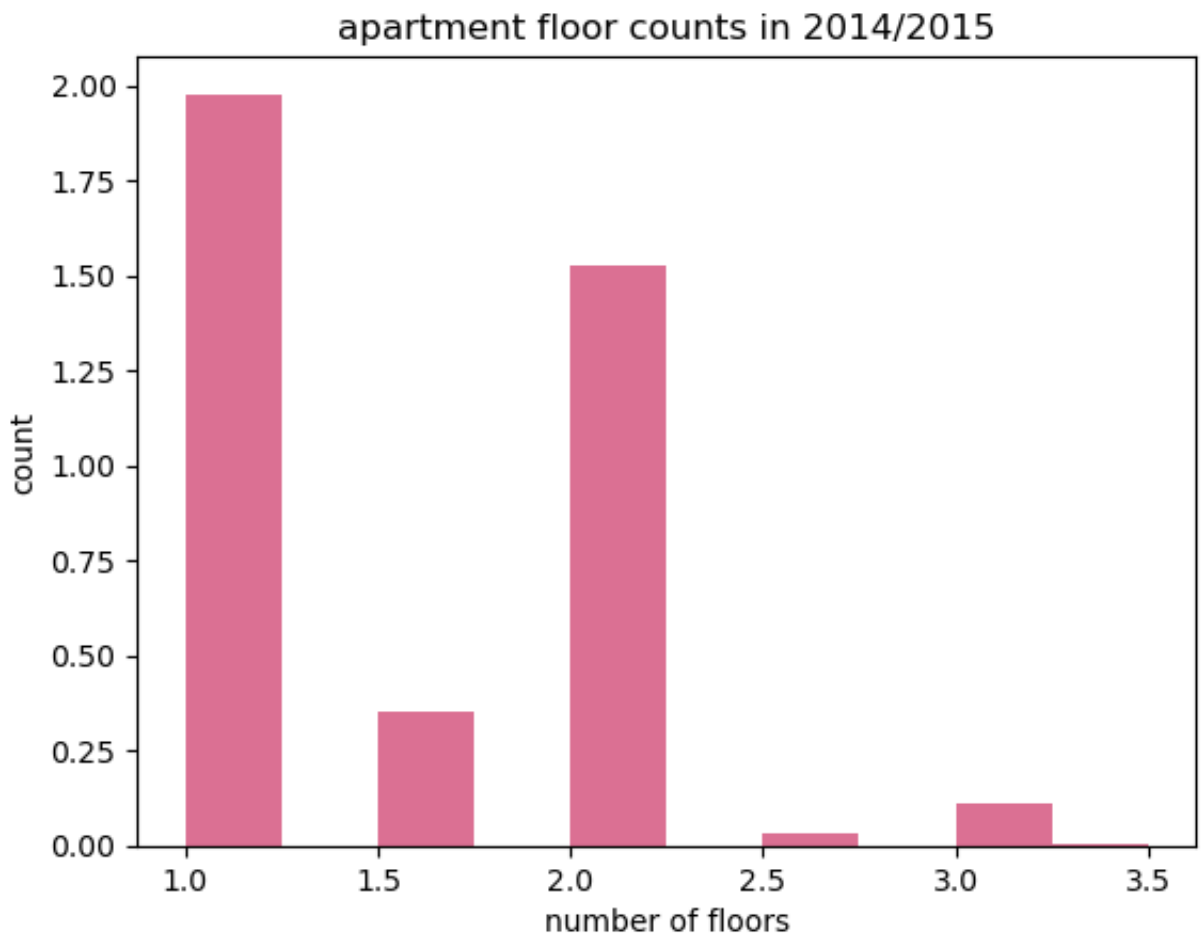


In [10]:

```
plt.hist(king_county['floors'],bins=10, density = True, color='palevic

plt.xlabel('number of floors')
plt.ylabel('count')
plt.title('apartment floor counts in 2014/2015')

# Able to see that most people prepfer one floor apartment
```

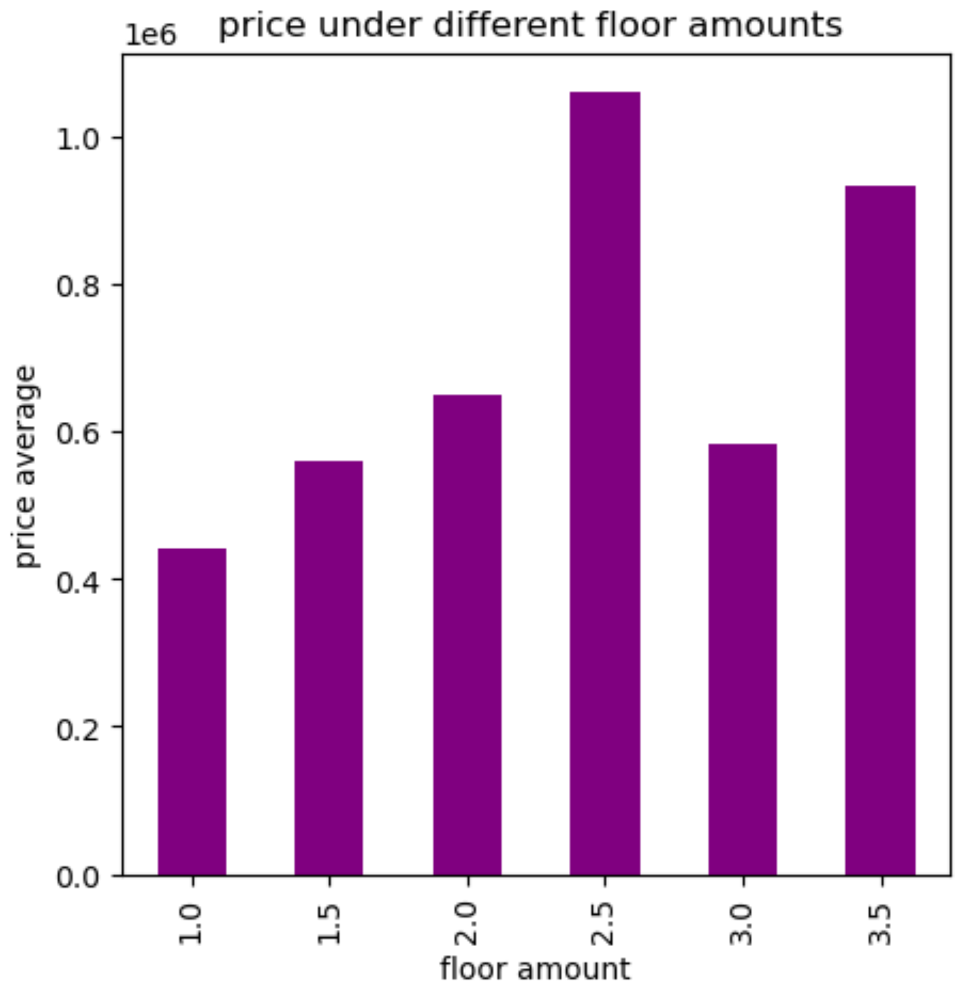
Out[10]:

```
Text(0.5, 1.0, 'apartment floor counts in 2014/2015')
```



In [11]:
```
# Since most of the people go to the single floor apartment type, I as

floor_price=king_county.copy().groupby(['floors'])['price'].agg('mean'
floor_price.plot(kind='bar', figsize=(5,5),width=0.5,color='purple',xl
                title='price under different floor amounts ')
```

Out[11]:
```
<AxesSubplot:title={'center':'price under different floor amounts '},
xlabel='floor amount', ylabel='price average'>
```

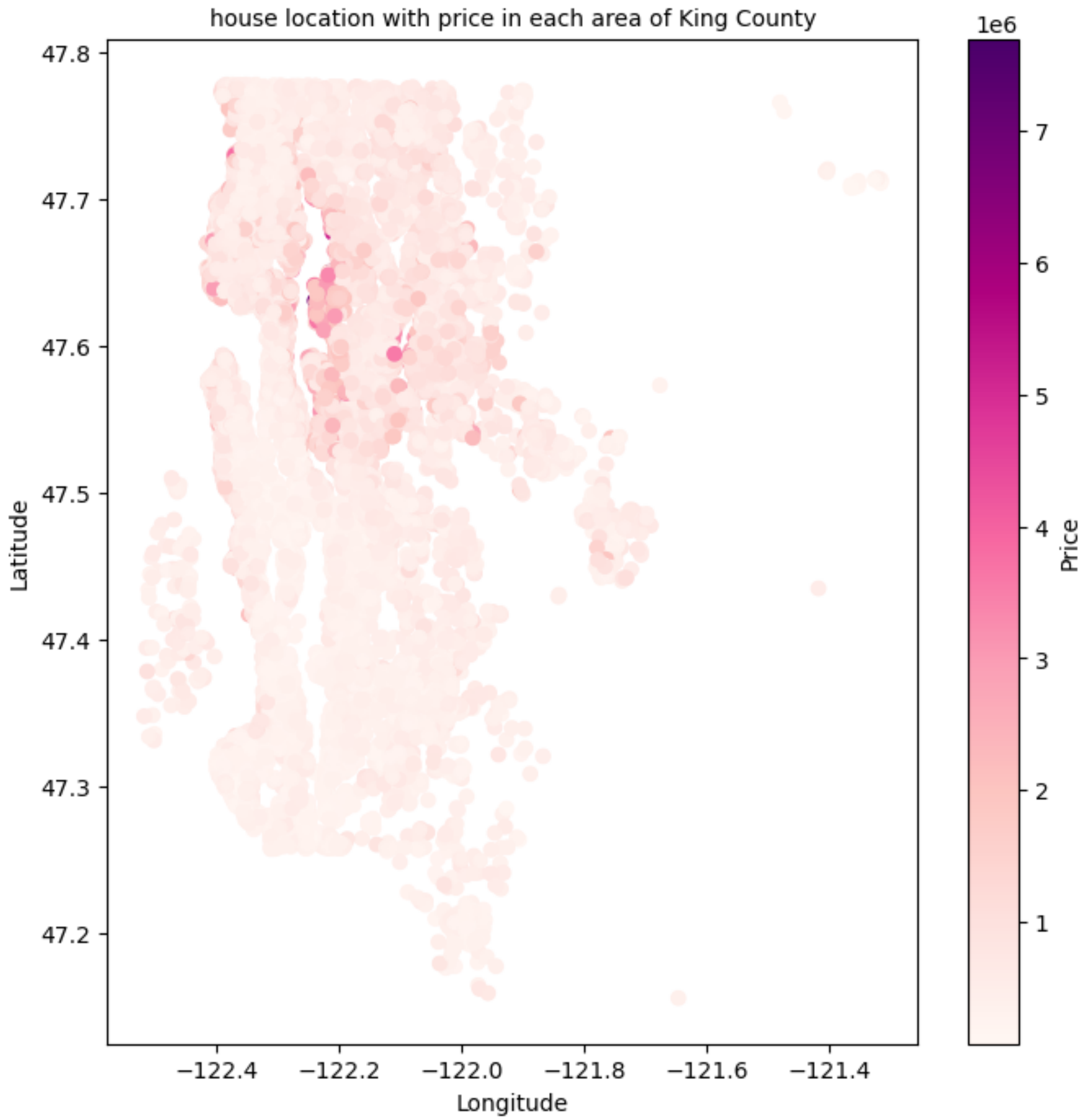


As I expected, single floor is listed with least price, However It is interesting to see floor with 2.5 has the highest average price, followed by 3.5 floors. It is hard to define a half floow however as courtsey to original dataset. The second popular (2 floor types houses) is ranked 3rd as lowest price

I also think location is a driver for price

In [12]:

```
plt.figure(figsize = (8,8))
plt.scatter(king_county['long'], king_county['lat'],c=king_county['pri
plt.colorbar().set_label('Price')

plt.xlabel('Longitude', fontsize=10)
plt.ylabel('Latitude', fontsize=10)
plt.title('house location with price in each area of King County', for
plt.show()
```



As colorbar showed, most houses across the lattitude has price lower than 2 million. From lattitude from 47.6 to 47.7 has a higher price slightest between 4-6 million. With higher lattitude can mean a warmer weather which can be a factor influencing popularity of the houses, therefore the location is a driving force

Exercise 2

In [15]:
```
diamond = pd.read_csv('diamonds.csv')
```

In [16]:
```
diamond.info()
#check the basic datatype of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   53940 non-null  int64
 1   carat        53940 non-null  float64
 2   cut          53940 non-null  object
 3   color        53940 non-null  object
 4   clarity      53940 non-null  object
 5   depth        53940 non-null  float64
 6   table        53940 non-null  float64
 7   price        53940 non-null  int64
 8   x            53940 non-null  float64
 9   y            53940 non-null  float64
 10  z            53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

In [17]:
```
# check whether there
x_zero=diamond[diamond['x']==0].index
y_zero=diamond[diamond['y']==0].index
z_zero=diamond[diamond['z']==0].index
diamond=diamond.drop(x_zero)
diamond=diamond.drop(y_zero)
diamond=diamond.drop(z_zero)

diamond.info()    #dropped all the 0 values in x,y,z. meaning that the
diamond.shape     # reserved 53920 vs beginning 53940
```

```
---------------------------------------------------------------------
------
KeyError                                  Traceback (most recent call
last)
```

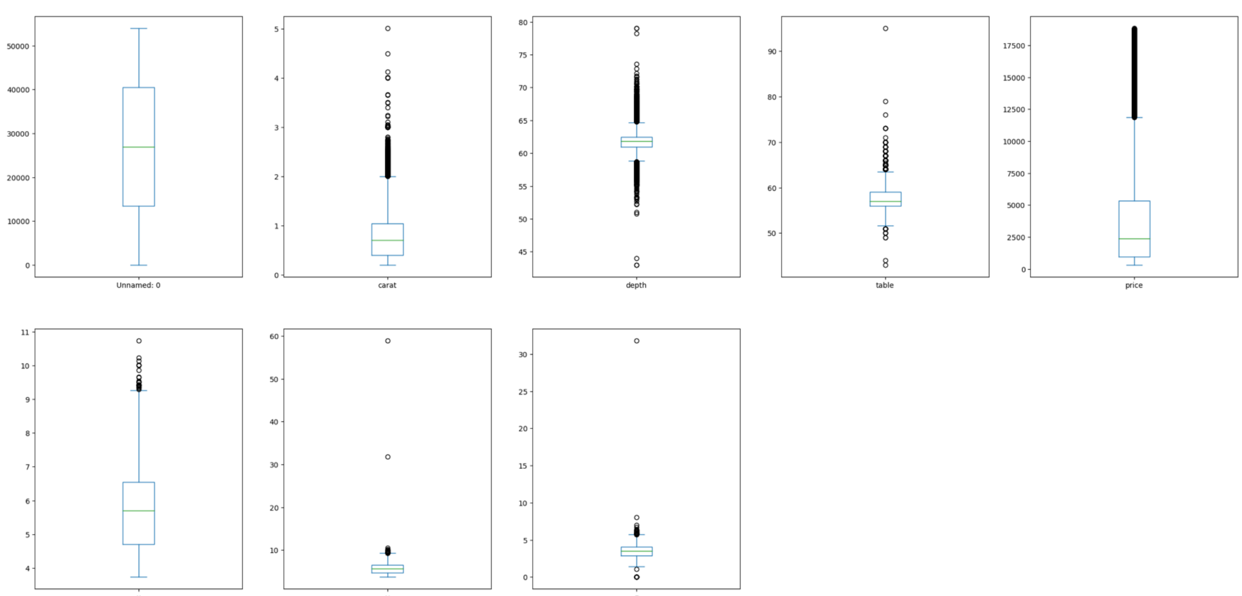
```
/var/folders/2z/v33d68yn3h9c2zf6z1r1mb4m0000gn/T/ipykernel_17242/1118
426028.py in <module>
      4 z_zero=diamond[diamond['z']==0].index
      5 diamond=diamond.drop(x_zero)
----> 6 diamond=diamond.drop(y_zero)
      7 diamond=diamond.drop(z_zero)
      8

~/opt/anaconda3/lib/python3.9/site-packages/pandas/util/_decorators.p
y in wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/frame.py in d
rop(self, labels, axis, index, columns, level, inplace, errors)
   4955                 weight  1.0     0.8
   4956         """
-> 4957         return super().drop(
   4958             labels=labels,
   4959             axis=axis,

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py in
drop(self, labels, axis, index, columns, level, inplace, errors)
   4265         for axis, labels in axes.items():
   4266             if labels is not None:
-> 4267                 obj = obj._drop_axis(labels, axis, level=leve
l, errors=errors)
   4268
   4269         if inplace:

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py in
_drop_axis(self, labels, axis, level, errors, consolidate, only_slice
)
   4309                 new_axis = axis.drop(labels, level=level,
errors=errors)
   4310             else:
-> 4311                 new_axis = axis.drop(labels, errors=errors)
   4312             indexer = axis.get_indexer(new_axis)
   4313

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.
py in drop(self, labels, errors)
   6659         if mask.any():
   6660             if errors != "ignore":
-> 6661                 raise KeyError(f"{list(labels[mask])} not fou
nd in axis")
   6662             indexer = indexer[~mask]
```

```
   6663         return self.delete(indexer)

KeyError: '[11963, 15951, 24520, 26243, 27429, 49556, 49557] not foun
d in axis'
```

In [18]:
```
diamond.head(20)
```

5	6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
6	7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
7	8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
8	9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
9	10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39
10	11	0.30	Good	J	SI1	64.0	55.0	339	4.25	4.28	2.73
11	12	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
12	13	0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
13	14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
14	15	0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27
15	16	0.32	Premium	E	I1	60.9	58.0	345	4.38	4.42	2.68
16	17	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68

In [19]:
```
diamond.plot(kind='box', subplots=True, figsize=(30,30),layout=(4,5),
plt.show()
```

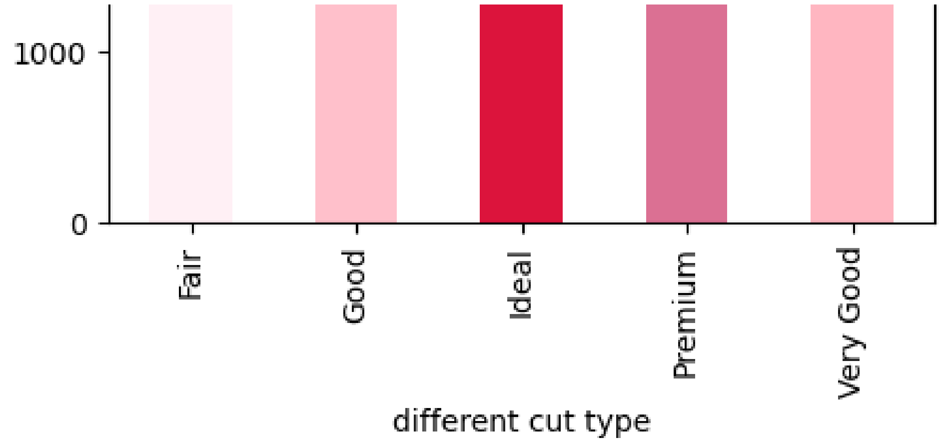


This plot contains the data distribution among different catergories, such as carat, price, depth etc. I want to see how salient is the outliers for each catergories

I want to go ahead and discover the relationship between cut type with price. therefore I decided to make a bar graph and see how different cut type price varies

In [20]:
```
# discover the correlation between cut type with price.


c = ['lavenderblush', 'pink', 'crimson', 'palevioletred','lightpink']
cut_price=diamond.copy().groupby(['cut'])['price'].agg('mean')

cut_price.plot(kind='bar', figsize=(5,5),width=0.5,color=c,xlabel='dif
              title='price under different cut type ')
ax.legend(['cut'])
```

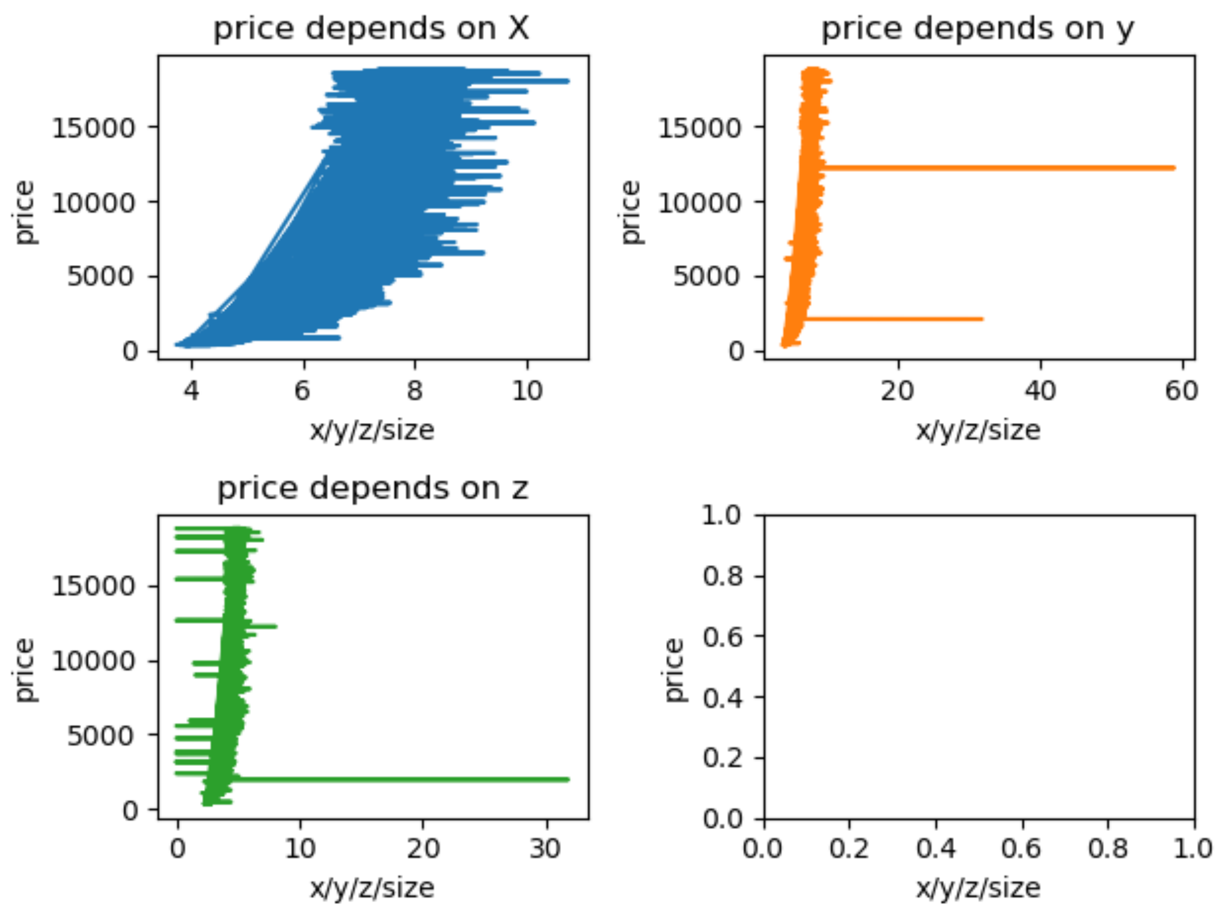
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/2z/v33d68yn3h9c2zf6z1r1mb4m0000gn/T/ipykernel_17242/2509859389.py in <module>
      7 cut_price.plot(kind='bar', figsize=(5,5),width=0.5,color=c,xlabel='different cut type',ylabel='price sum',
      8               title='price under different cut type ')
----> 9 ax.legend(['cut'])

NameError: name 'ax' is not defined
```



The relationship between cut type and price: surprisingly, ideal cut has the lowest average price out of all 5 types. Which is counterintuitive, as better cut of diamond demands more efforts and expertise and therefore more market value. It is understable the premium has the highest, whilest fair has the second highest price just doesn't make sense

Secondly, I want to see how x,y,z which are the 3D dimension of diamond, I assume that X would be the most influential factor as a lot of people care about the width not necessarily the deepth. That's why people getting married look at the size of the ring not its deepth? I assume. I then made three subplots to see its relationsbip with price

In [21]:
```
#
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(diamond['x'], diamond['price'], 'tab:blue')
axs[0, 0].set_title('price depends on X')
axs[0, 1].plot(diamond['y'], diamond['price'], 'tab:orange')
axs[0, 1].set_title('price depends on y')
axs[1, 0].plot(diamond['z'], diamond['price'], 'tab:green')
axs[1, 0].set_title('price depends on z')



for ax in axs.flat:
    ax.set(xlabel='x/y/z/size', ylabel='price')
fig.tight_layout()
```

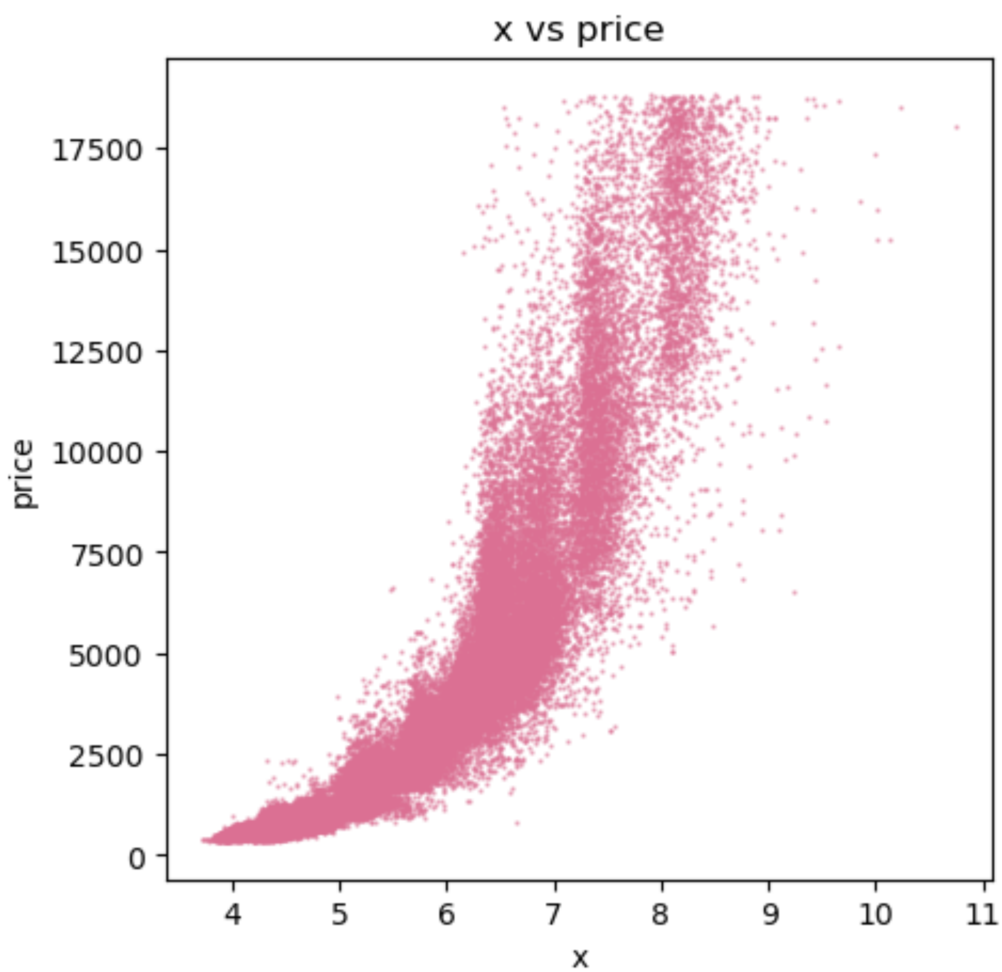


In [22]:

```
diamond.plot(kind='scatter', x='x',y='price' ,figsize=(5,5), s=0.5, al
```

Out[22]:

```
<AxesSubplot:title={'center':'x vs price'}, xlabel='x', ylabel='price
'>
```

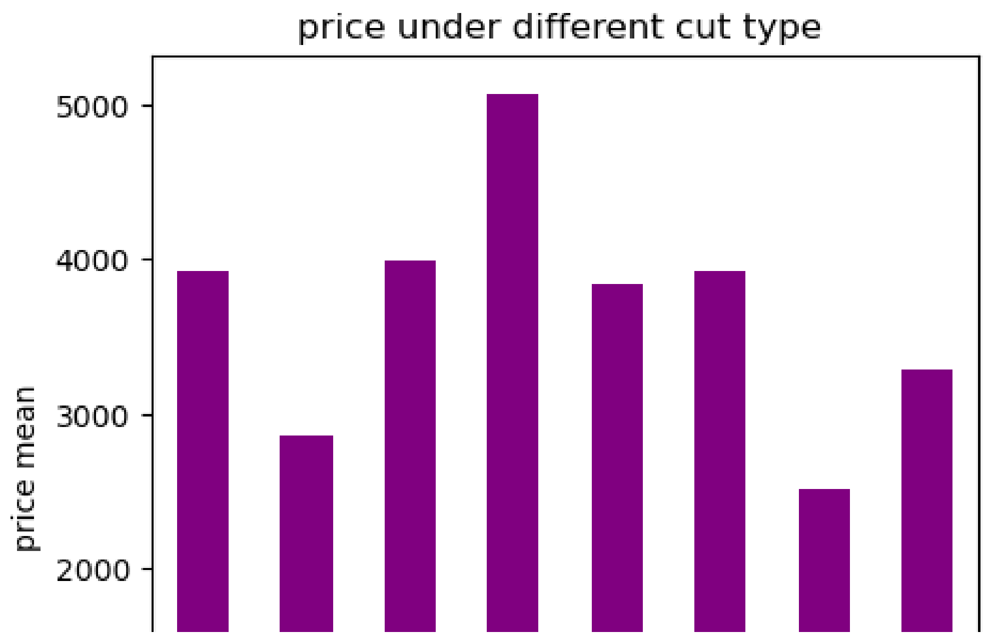


As you can see, X does made the price vary more than y and z. The data on the first graph is more scattered than the 2nd and 3nd. Meaning X can be a driving force to price, with bigger X value, higher price. You can see a rough positive correlation reflected from the pink scatterplot

I also assume the clarity/colorness of the diamond affect prices and I made another bar chat to analyze it

In [23]:
```
cut_price=diamond.copy().groupby(['clarity'])['price'].agg('mean')
cut_price.plot(kind='bar', figsize=(5,5),width=0.5,color='purple',xlab
              title='price under different cut type ')

ax.legend(['cut'])
```

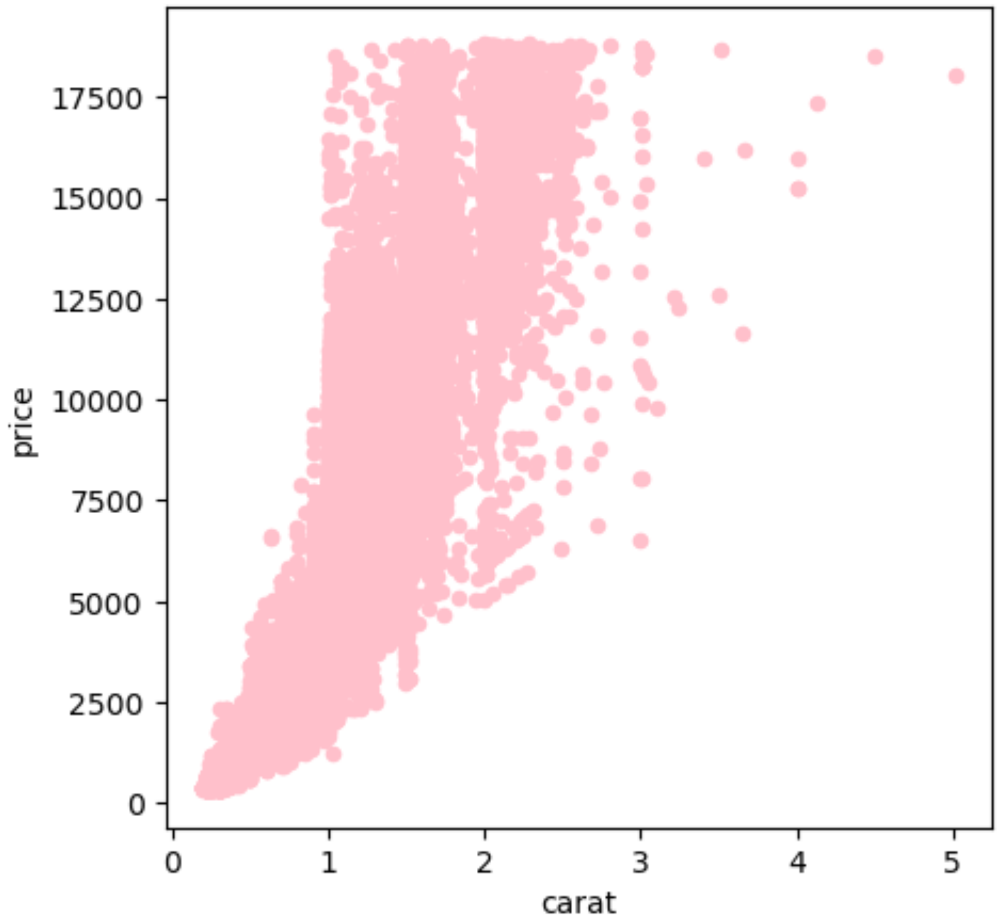
Out[23]: <matplotlib.legend.Legend at 0x7fd9f3174850>



There are differentiation among clarity, with SI2 having the highest average price which is counterintuitive as SI2 has worse clarity than IF(internally flawless), which deserve a higher price. It is important to combine different factors rather than just clarity to see the influence on price

In [24]:
```
diamond.plot(kind='scatter', x='carat',y='price' ,figsize=(5,5), color
```

Out[24]: <AxesSubplot:xlabel='carat', ylabel='price'>



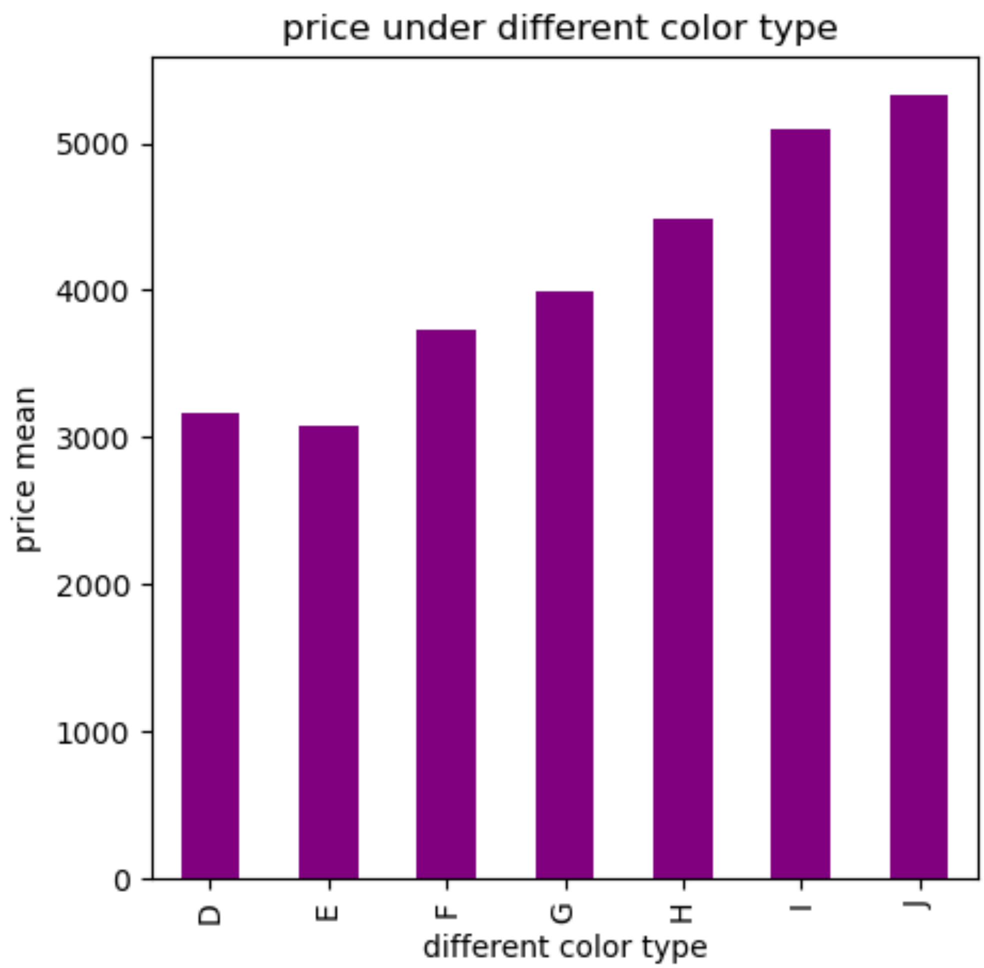
This graph shows that as carat increases, prices increases which is adherent with my assumption.

In [25]:

```
#I also want to see how color influence price.

cut_price=diamond.copy().groupby(['color'])['price'].agg('mean')
cut_price.plot(kind='bar', figsize=(5,5),width=0.5,color='purple',xlab
              title='price under different color type ')
```

Out[25]:

```
<AxesSubplot:title={'center':'price under different color type '}, xl
abel='different color type', ylabel='price mean'>
```



In [ ]: