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

In [2]:
```
data=pd.read_csv("C:/Users/91740/OneDrive/Desktop/SLASH MARKS INTERNSHIP/house-price-prediction-master/house-pr
data.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_ab
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1

5 rows × 21 columns

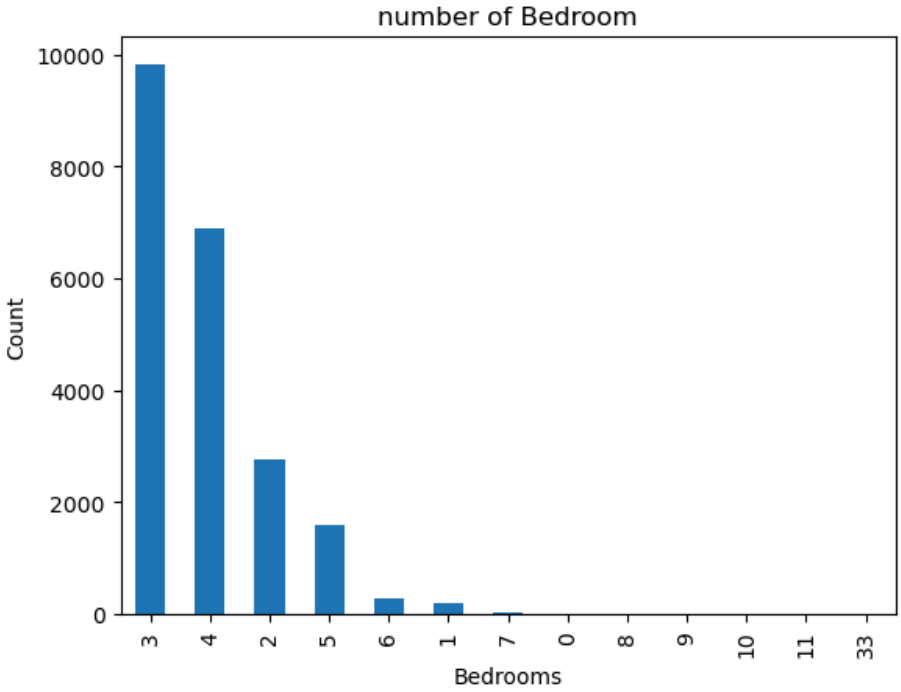
In [3]:
```
data.describe()
```

Out[3]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vi
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.0000
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.2343
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.7663
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.0000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.0000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.0000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.0000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.0000

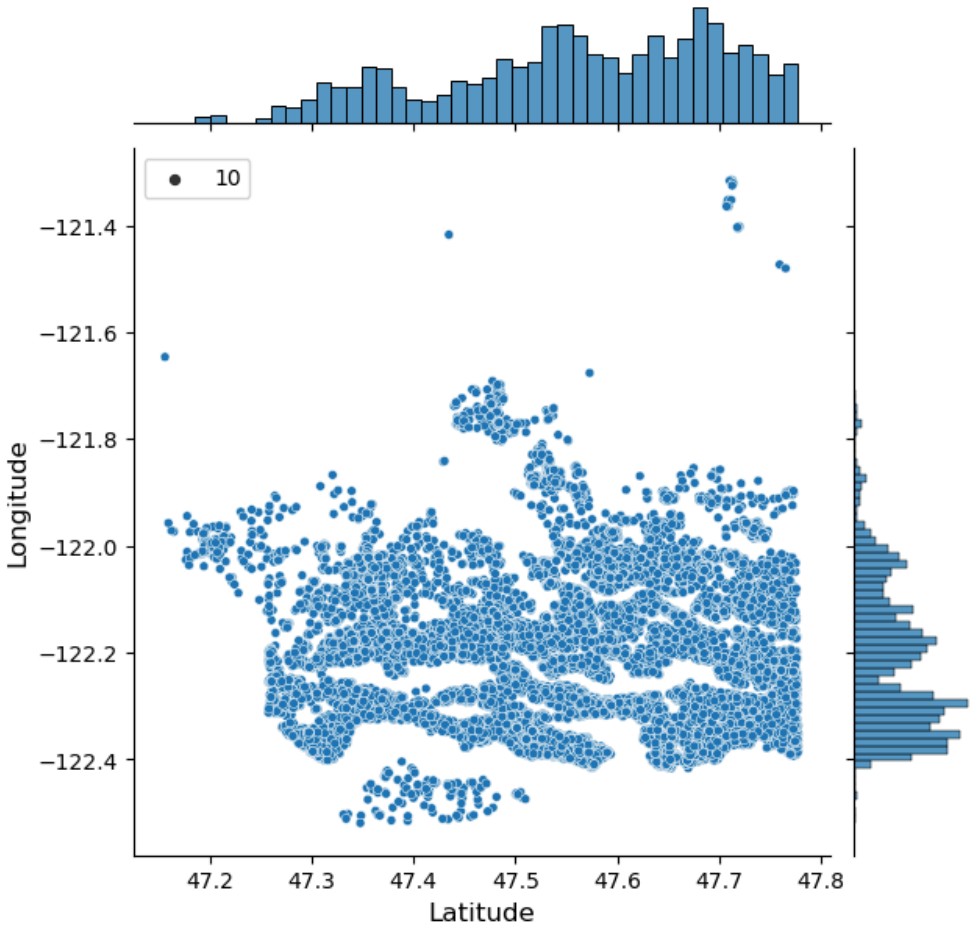
In [4]:
```
data['bedrooms'].value_counts().plot(kind='bar')
plt.title('number of Bedroom')
plt.xlabel('Bedrooms')
plt.ylabel('Count')
sns.despine
```

Out[4]:
```
<function seaborn.utils.despine(fig=None, ax=None, top=True, right=True, left=False, bottom=False, offset=None,
trim=False)>
```



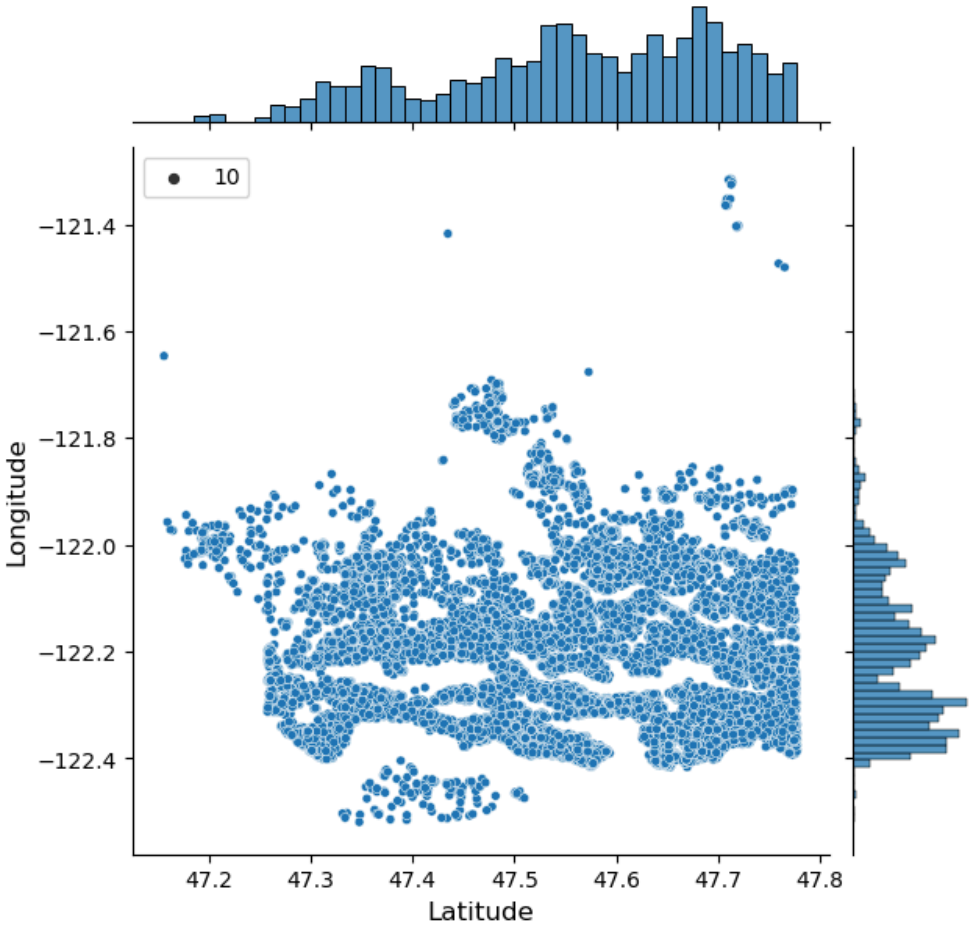
In [5]:
```
plt.figure(figsize=(10,10))
sns.jointplot(x=data.lat.values, y=data.long.values, size=10)
plt.ylabel('Longitude', fontsize=12)
plt.xlabel('Latitude', fontsize=12)
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



In [6]:
```
plt.figure(figsize=(10,10))
sns.jointplot(x=data.lat.values, y=data.long.values, size=10)
plt.ylabel('Longitude', fontsize=12)
plt.xlabel('Latitude', fontsize=12)
plt.show()
plt1 = plt()
sns.despine
```

```
<Figure size 1000x1000 with 0 Axes>
```

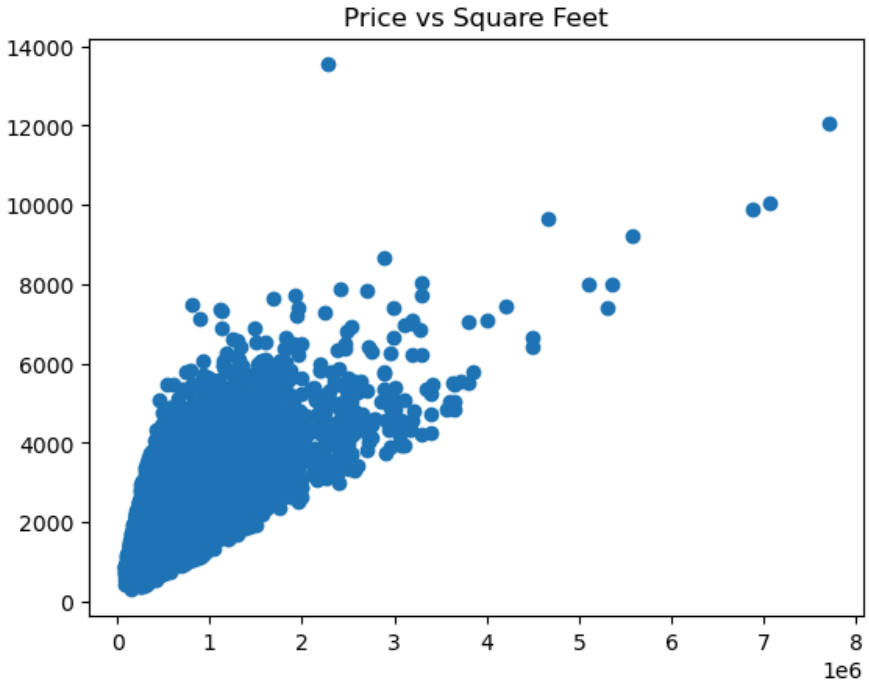



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 6
      4 plt.xlabel('Latitude', fontsize=12)
      5 plt.show()
----> 6 plt1 = plt()
      7 sns.despine

TypeError: 'module' object is not callable
```

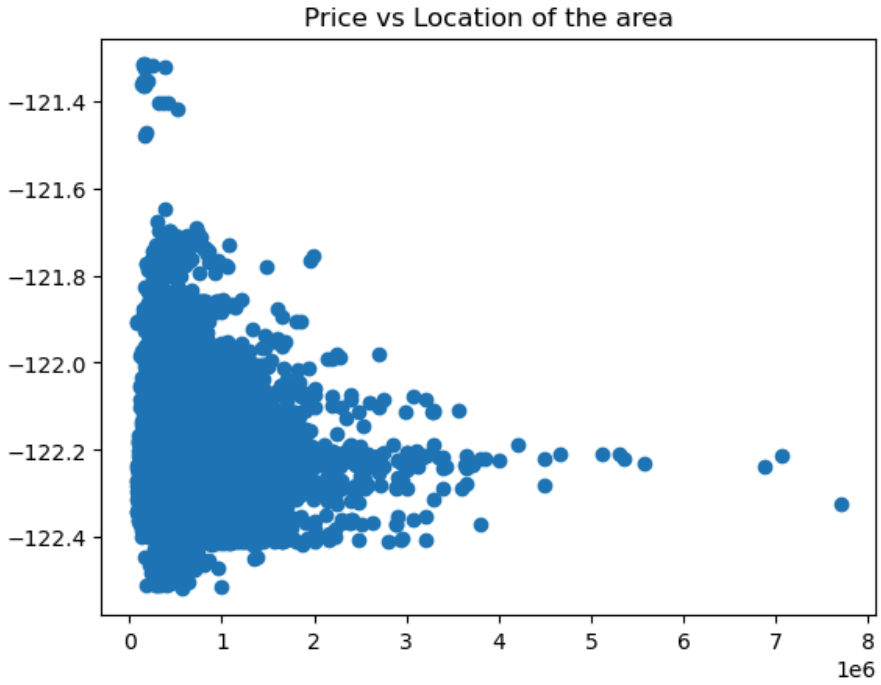
In [7]:
```
plt.scatter(data.price,data.sqft_living)
plt.title("Price vs Square Feet")
```

Out[7]:
```
Text(0.5, 1.0, 'Price vs Square Feet')
```



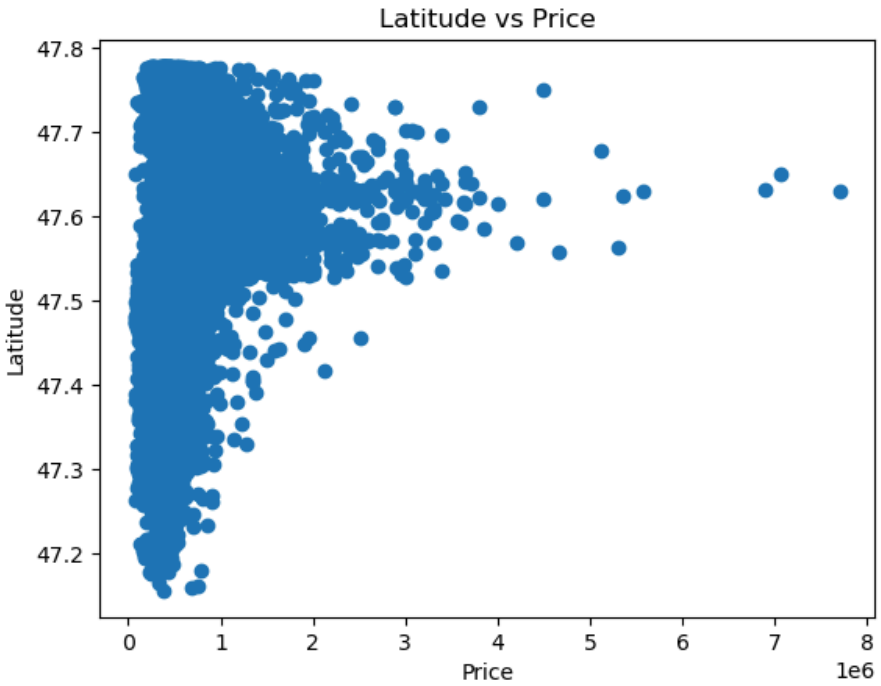
In [8]:
```
plt.scatter(data.price,data.long)
plt.title("Price vs Location of the area")
```

Out[8]:
```
Text(0.5, 1.0, 'Price vs Location of the area')
```

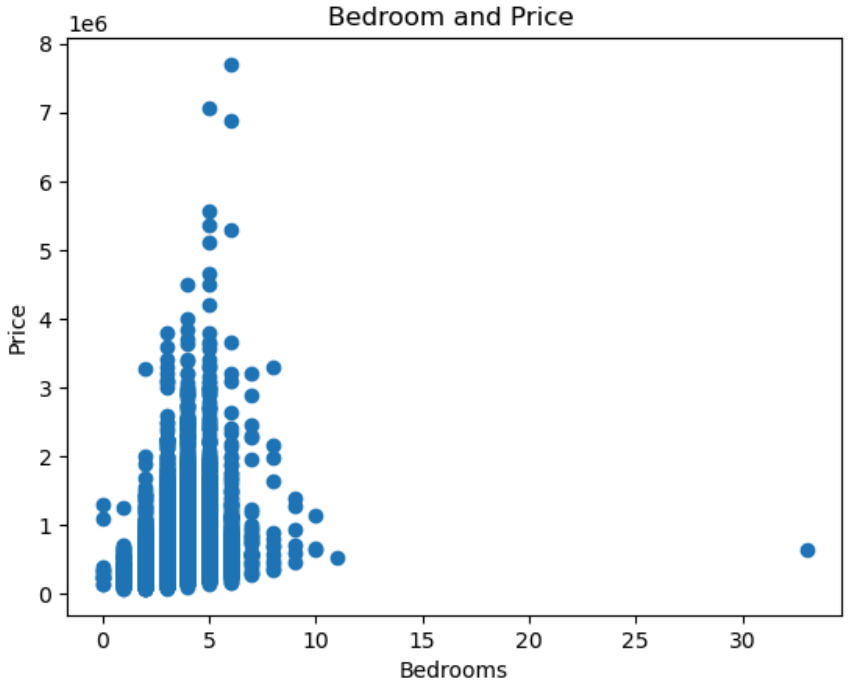


In [9]:
```
plt.scatter(data.price,data.lat)
plt.xlabel("Price")
plt.ylabel('Latitude')
plt.title("Latitude vs Price")
```

Out[9]:
```
Text(0.5, 1.0, 'Latitude vs Price')
```



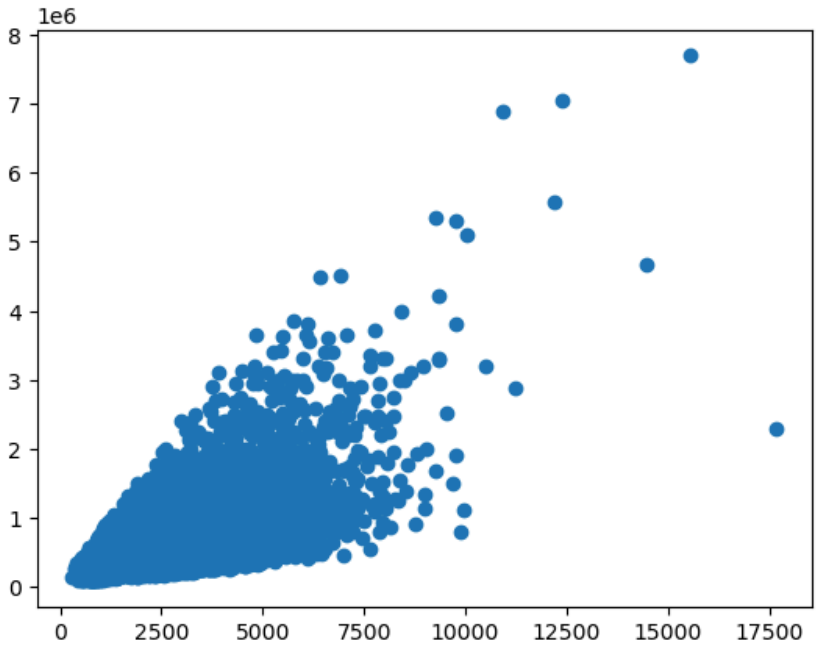
In [10]:
```
plt.scatter(data.bedrooms,data.price)
plt.title("Bedroom and Price ")
plt.xlabel("Bedrooms")
plt.ylabel("Price")
plt.show()
sns.despine
```



Out[10]:
```
<function seaborn.utils.despine(fig=None, ax=None, top=True, right=True, left=False, bottom=False, offset=None,
trim=False)>
```

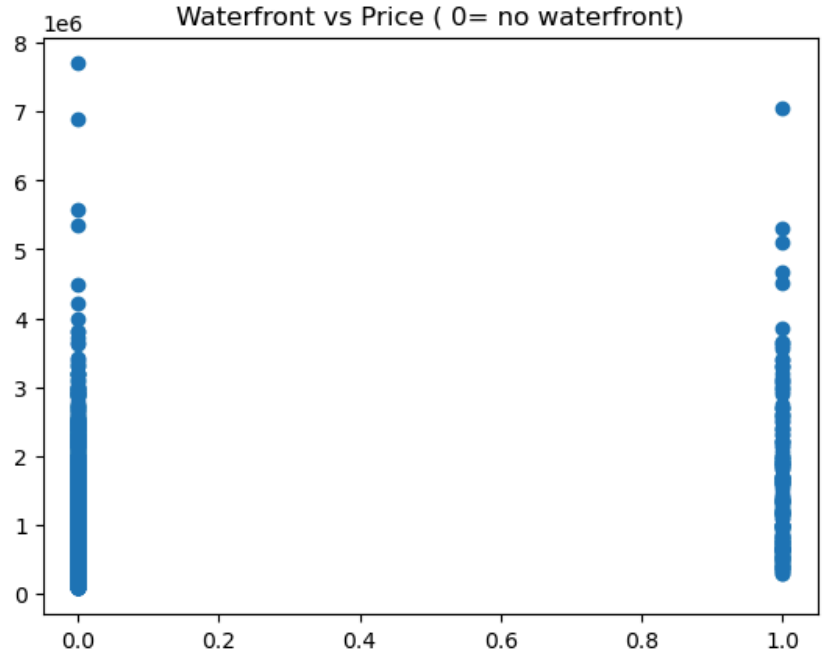
In [11]:
```
plt.scatter((data['sqft_living']+data['sqft_basement']),data['price'])
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x20a434fd210>
```



In [12]:
```
plt.scatter(data.waterfront,data.price)
plt.title("Waterfront vs Price ( 0= no waterfront)")
```

Out[12]:
```
Text(0.5, 1.0, 'Waterfront vs Price ( 0= no waterfront)')
```



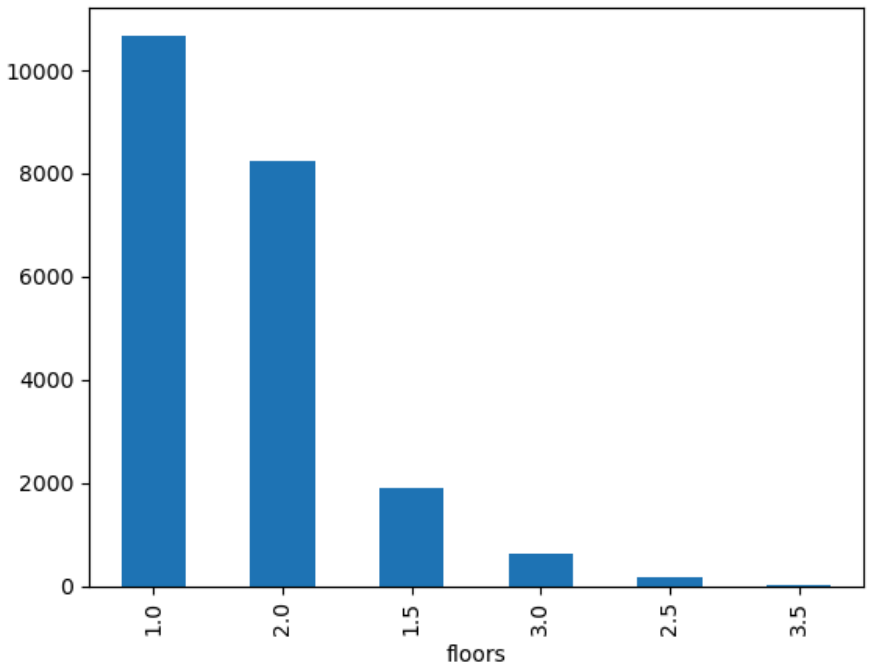
In [13]:
```
train1=data.drop(['id','price'],axis=1)
train1.head()
```

Out[13]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basen
0	20141013T000000	3	1.00	1180	5650	1.0	0	0	3	7	1180	
1	20141209T000000	3	2.25	2570	7242	2.0	0	0	3	7	2170	
2	20150225T000000	2	1.00	770	10000	1.0	0	0	3	6	770	
3	20141209T000000	4	3.00	1960	5000	1.0	0	0	5	7	1050	
4	20150218T000000	3	2.00	1680	8080	1.0	0	0	3	8	1680	

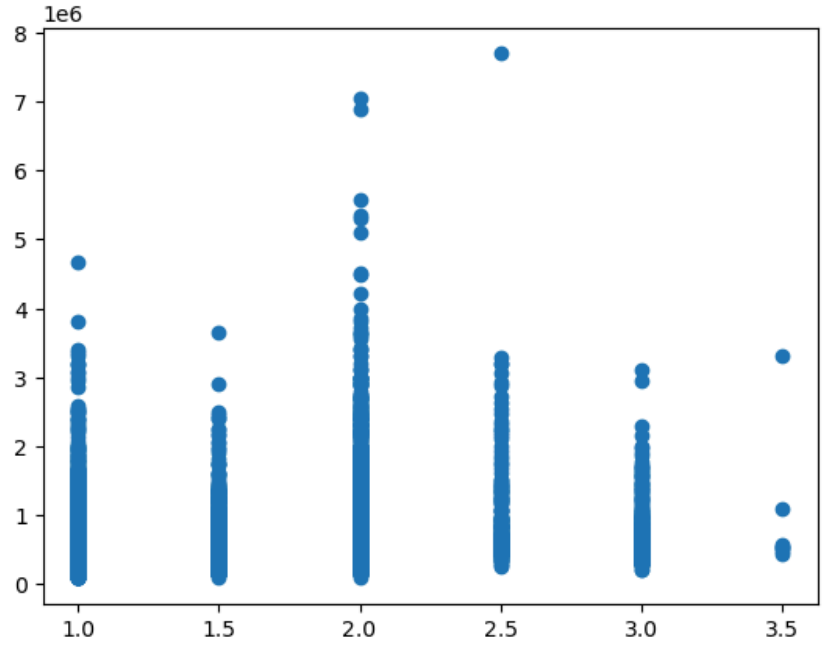
In [14]:
```
data.floors.value_counts().plot(kind='bar')
```

Out[14]:
```
<Axes: xlabel='floors'>
```



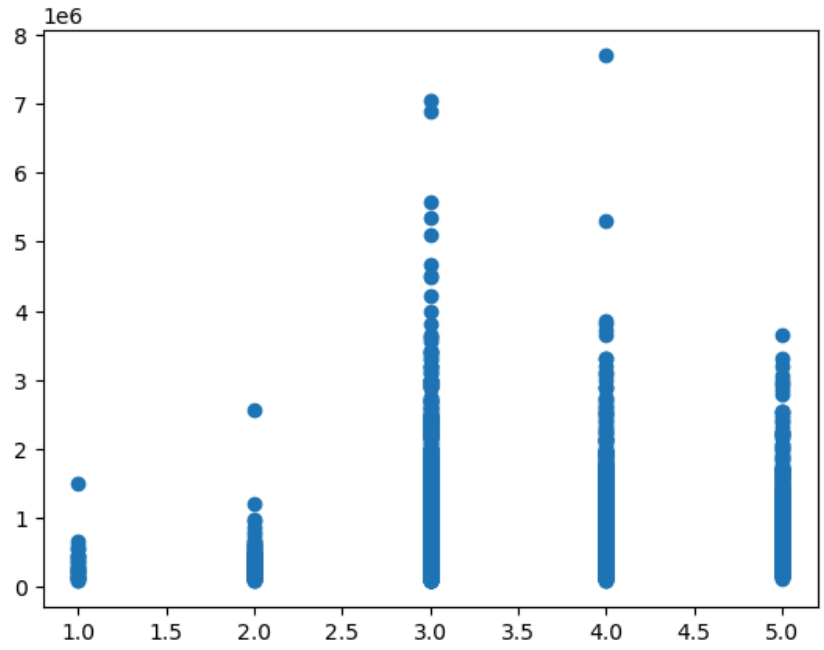
In [15]:
```
plt.scatter(data.floors,data.price)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x20a429563d0>
```



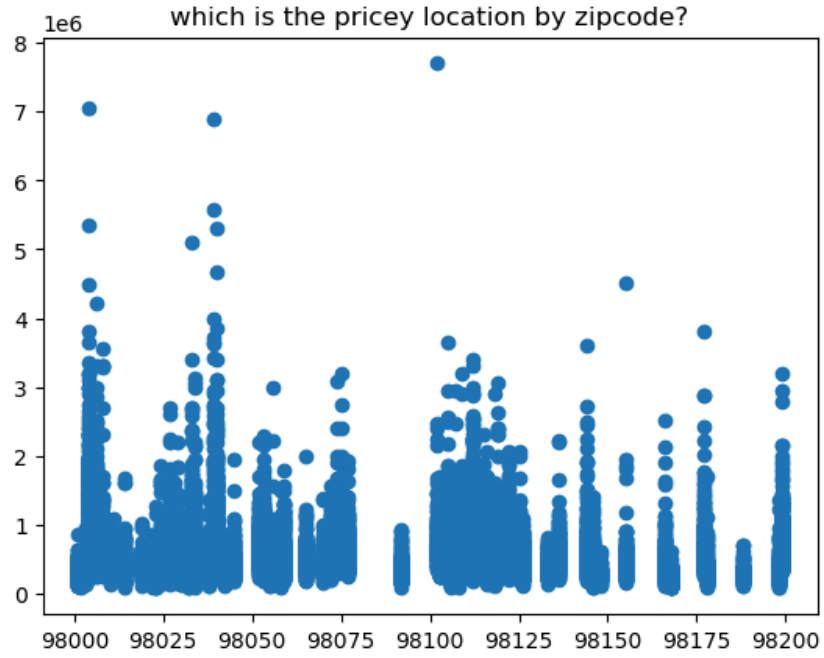
In [16]:
```
plt.scatter(data.condition,data.price)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x20a45d48890>
```



In [17]:
```
plt.scatter(data.zipcode,data.price)
plt.title("which is the pricey location by zipcode?")
```

Out[17]:
```
Text(0.5, 1.0, 'which is the pricey location by zipcode?')
```



In [18]:
```
from sklearn.linear_model import LinearRegression
reg=LinearRegression
```

In [19]:
```
labels = data['price']
conv_dates = [1 if values == 2014 else 0 for values in data.date ]
data['date'] = conv_dates
train1 = data.drop(['id', 'price'],axis=1)
```

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
X_train , X_test , y_train , y_test = train_test_split(train1 , labels , test_size = 0.10,random_state =2)
```

In [22]:
```
from sklearn.linear_model import LinearRegression

# Assuming you have your training and testing data split
X_train, X_test, y_train, y_test = train_test_split(train1 , labels , test_size = 0.10,random_state =2)

# Create the regressor object
reg = LinearRegression()

# Train the regressor
reg.fit(X_train, y_train)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
from sklearn.linear_model import LinearRegression

# Assuming you have your training and testing data split
X_train, X_test, y_train, y_test = train_test_split(train1 , labels , test_size = 0.10,random_state =2)

# Create the regressor object
reg = LinearRegression()

# Train the regressor
reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = reg.predict(X_test)

# Evaluate the performance using score (higher is better for regression)
score = reg.score(X_test, y_test)
print("R-squared score:", score)
```

```
R-squared score: 0.7320342760357297
```

In [24]:
```
from sklearn import ensemble
clf = ensemble.GradientBoostingRegressor(n_estimators = 400, max_depth = 5, min_samples_split = 2,
          learning_rate = 0.1, loss = 'huber')
```

In [25]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Assuming you have your training data X_train and y_train
clf = GradientBoostingRegressor(loss="huber")  # Set the correct loss function

clf.fit(X_train, y_train)
```

Out[25]:
```
GradientBoostingRegressor
GradientBoostingRegressor(loss='huber')
```

In [26]:
```
clf.score(X_test,y_test)
```

Out[26]:
```
0.8807478418833407
```

In [27]:
```
# Define params dictionary (assuming you have values for these parameters)
params = {
    'n_estimators': 100,  # Number of boosting stages
    # Other parameters for GradientBoostingRegressor
}

t_sc = np.zeros((params['n_estimators']), dtype=np.float64)
```

In [28]:
```
t_sc
```

Out[28]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [29]:
```
y_pred = reg.predict(X_test)
```

In [30]:
```
y_pred
```

Out[30]:
```
array([ 708968.57473738, 1454115.56763583,  422619.27778294, ...,
        191420.54893185,  238056.6859847 ,  451235.02573487])
```

In [32]:
```
from sklearn.metrics import mean_squared_error  # Assuming squared error loss
for i, y_pred in enumerate(clf.staged_predict(X_test)):
    t_sc[i] = mean_squared_error(y_test, y_pred)  # Calculate MSE for each stage
```

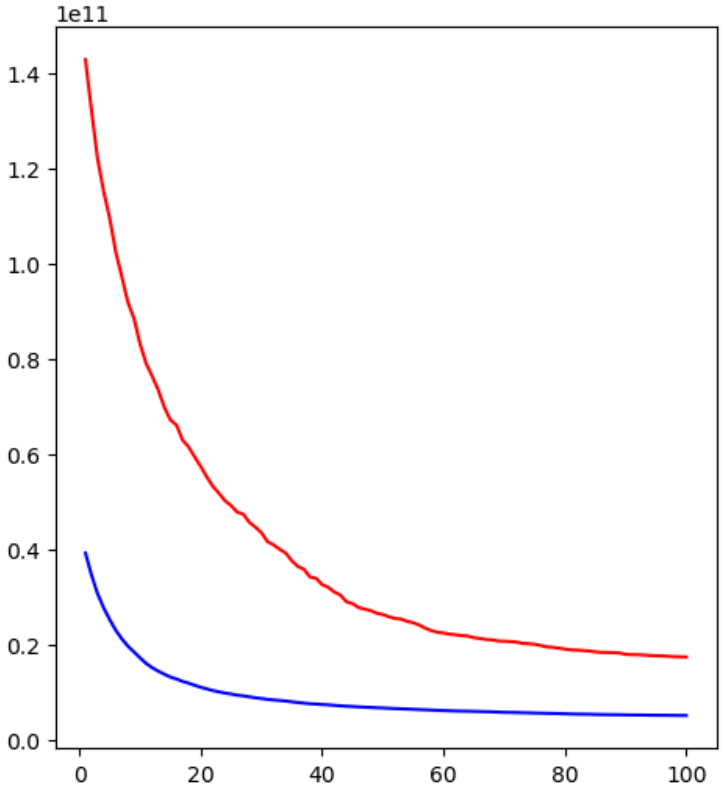
In [33]:
```
testsc = np.arange((params['n_estimators']))+1
```

In [34]:
```
testsc
```

Out[34]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
        92,  93,  94,  95,  96,  97,  98,  99, 100])
```

In [35]:
```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(testsc,clf.train_score_,'b-',label= 'Set dev train')
plt.plot(testsc,t_sc,'r-',label = 'set dev test')
```

Out[35]:
```
[<matplotlib.lines.Line2D at 0x20a47ea4190>]
```



In [36]:
```
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
```

In [37]:
```
pca = PCA()
```

In [38]:
```
pca
```

Out[38]:
```
PCA()
```

In [39]:
```
pca.fit_transform(scale(train1))
```

Out[39]:
```
array([[-2.64785461e+00, -4.54699955e-02, -3.16665762e-01, ...,
        -7.94687728e-02,  2.59674388e-17,  0.00000000e+00],
       [-2.34485164e-01,  1.68297114e+00, -7.61521725e-01, ...,
         9.81487761e-01,  2.80892751e-14, -0.00000000e+00],
       [-2.57007792e+00, -6.14344122e-01,  3.49292423e-01, ...,
        -1.38570764e-01,  3.09103597e-14, -0.00000000e+00],
       ...,
       [-2.41985641e+00, -1.10027662e+00, -1.46293798e+00, ...,
         9.66785881e-01,  1.09776500e-16, -0.00000000e+00],
       [ 3.32183025e-01, -1.88043103e+00, -1.04412760e+00, ...,
        -3.97449542e-01,  2.89754352e-17,  0.00000000e+00],
       [-2.43180432e+00, -1.08505981e+00, -1.47248379e+00, ...,
         9.53674385e-01,  2.73549743e-17, -0.00000000e+00]])
```

In [ ]: