House Price prediction using Linear Regression

- Linear Regression: A Linear Model for understanding the relationship between input and output numerical variables. For e.g. it assumes a linear relationship between the input variables (x) and the single output variable (y). This implies that y can be estimated from a linear combination of x.
- Simple linear regression has single input variable (x).
- Multiple linear regression has more than a single input variable (x).
- Ordinary Least Squares (OLS): common method used to train a linear regression model.

$Y = a + bX + \varepsilon$

where Y = Dependent Variable

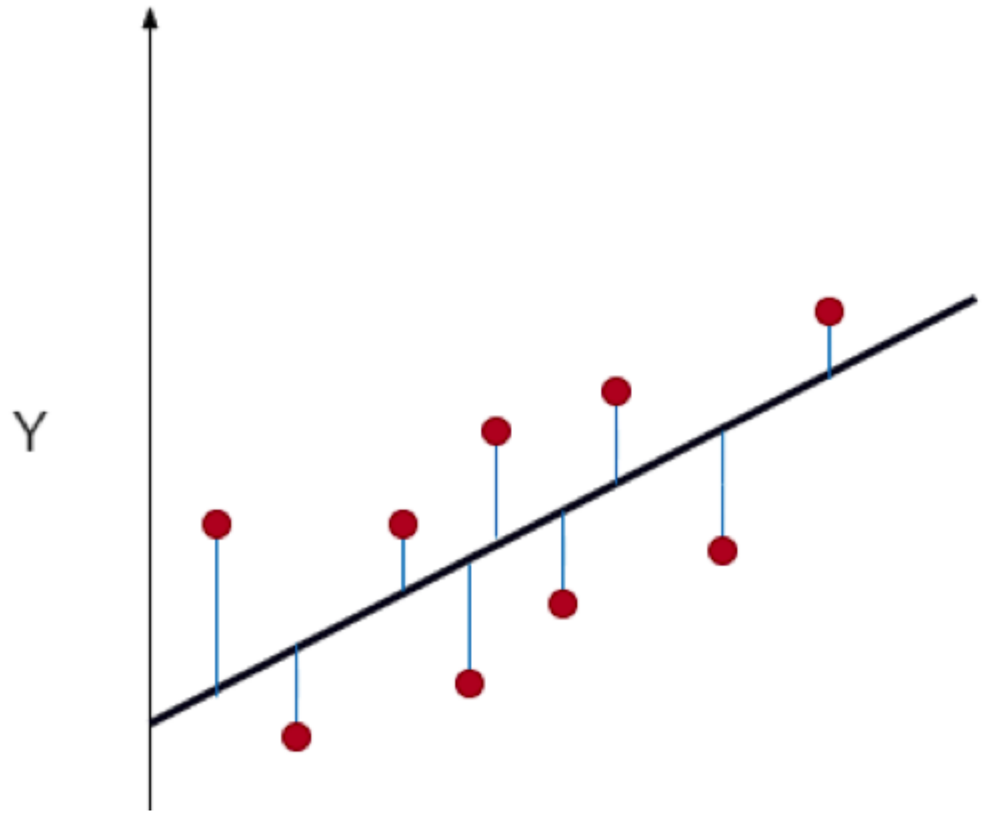
X = Independent Variable

a = Intercept of the line that offers additional DOF or degree of freedom.

b = Linear regression coefficient, which is a scale factor to every input value.

ε = Random error

Linear Regression



Target: develop a model which predicts the price of a house.

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("data/kc_house_data.csv")
data.head()
```

Out[2]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	s
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	s
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

In [3]:
```
data.shape #(rows,columns)
```
Out[3]:
```
(21613, 21)
```

In [4]:
```
data.columns
```
Out[4]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [5]:
```
len(data.columns)
```
Out[5]:
```
21
```

In [6]:
```
data.count() #Number of non-NA values
```
Out[6]:
```
id               21613
date             21613
price            21613
bedrooms         21613
bathrooms        21613
sqft_living      21613
sqft_lot         21613
floors           21613
waterfront       21613
view             21613
condition        21613
grade            21613
sqft_above       21613
sqft_basement    21613
yr_built         21613
yr_renovated     21613
zipcode          21613
```

```
lat              21613
long             21613
sqft_living15    21613
sqft_lot15       21613
dtype: int64
```

In [7]:
```
data.describe()
```

Out[7]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
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	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000

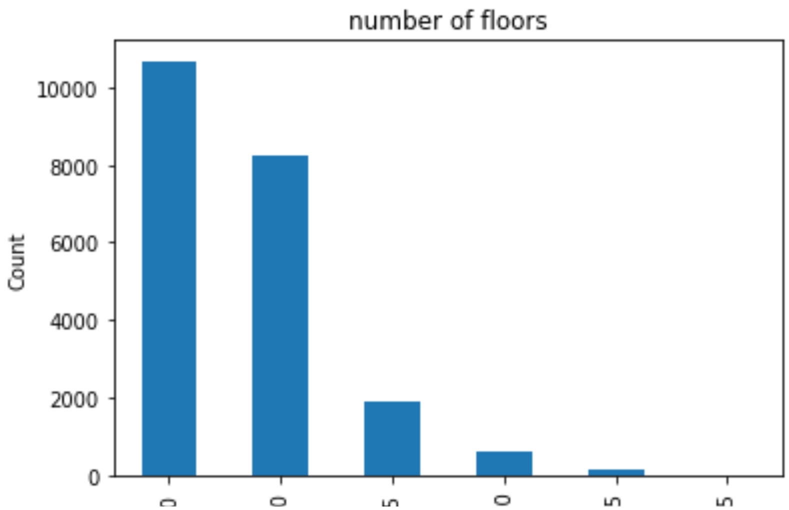
observe the smallest and largest house:

```
- 0 vs. 8 bathrooms!!
- 0 vs. 33 bedrooms !!
- 1 to 3.5 floors !!
```

How many floors do most houses have?

In [8]:
```
data['floors'].value_counts().plot(kind='bar')
plt.title('number of floors')
plt.xlabel('Floors')
plt.ylabel('Count')
```

Out[8]:
```
Text(0, 0.5, 'Count')
```



Most houses have a single floor

What about bedrooms?

In [9]:
```
data['bedrooms'].value_counts().plot(kind='bar')
plt.title('number of Bedroom')
plt.xlabel('Bedrooms')
plt.ylabel('Count')
```

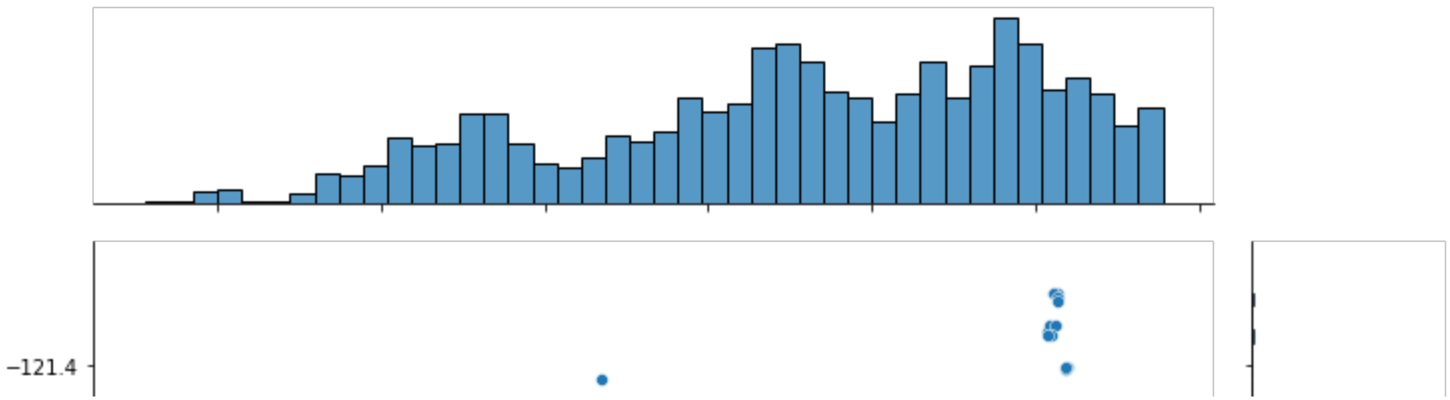
Out[9]:
```
Text(0, 0.5, 'Count')
```

number of Bedroom

Where are the houses located?

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

```
<Figure size 720x720 with 0 Axes>
```



In [ ]:

How are the house priced?

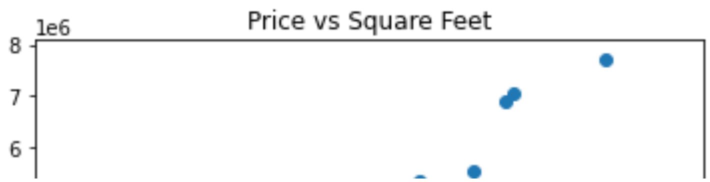
- Any relation between cost and size of the house?
- or cost and location of the house?
- or cost and no. of bathrooms/bedrooms of the house?

Lets explore some scatter plots

In [11]:

```
plt.scatter(data.sqft_living, data.price)
plt.title("Price vs Square Feet")
plt.xlabel("sqft_living")
plt.ylabel("Price")
```

Out[11]: Text(0, 0.5, 'Price')

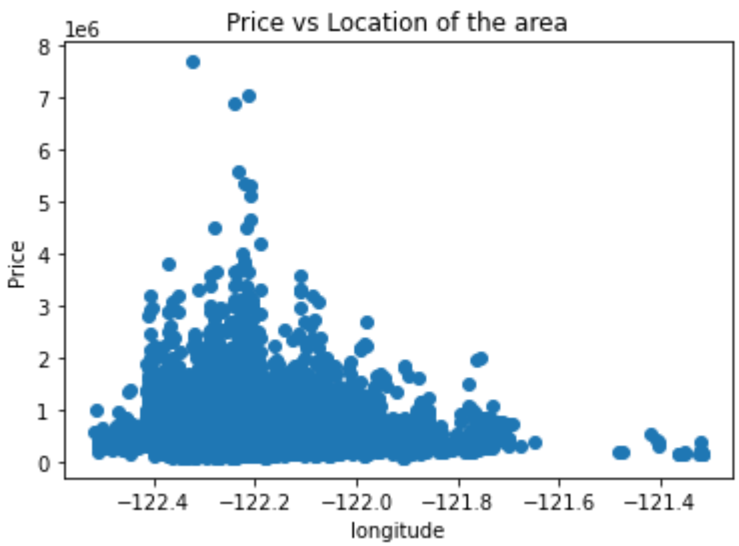


- General trend of data points- somewhat Linear
- But the Largest House (about 14000) did not cost the most
- The most costly house did not have the largest area (about 12000)

So it seems that other factors also affects the house price

In [12]:
```
plt.scatter(data.long, data.price)
plt.title("Price vs Location of the area")
plt.xlabel("longitude")
plt.ylabel("Price")
```

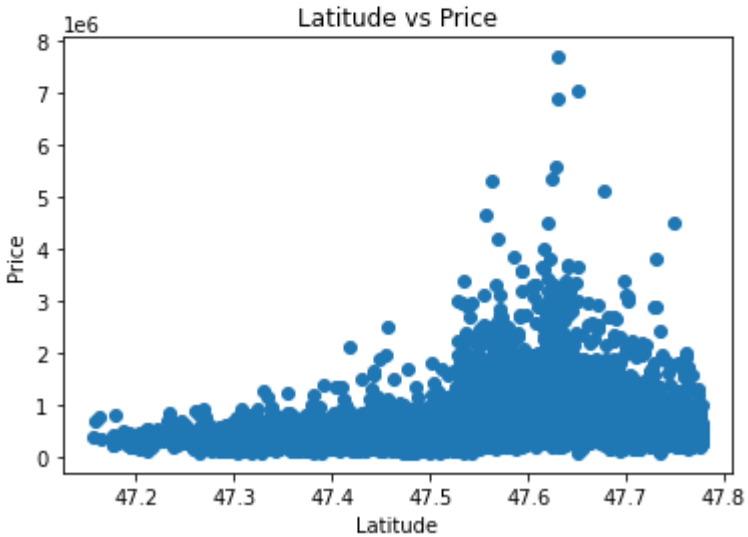
Out[12]: `Text(0, 0.5, 'Price')`



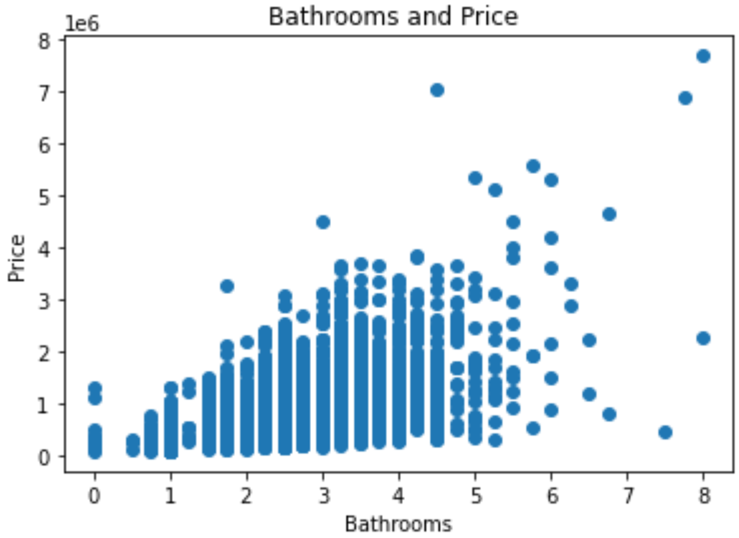
Houses at longitude about -122.3 costs Highest.

In [13]:
```
plt.scatter(data.lat, data.price)
plt.ylabel("Price")
plt.xlabel('Latitude')
plt.title("Latitude vs Price")
```

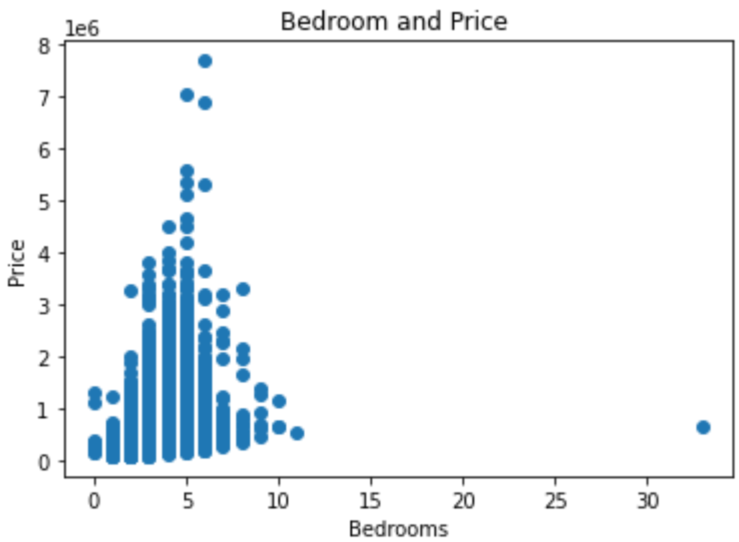
Out[13]:
```
Text(0.5, 1.0, 'Latitude vs Price')
```



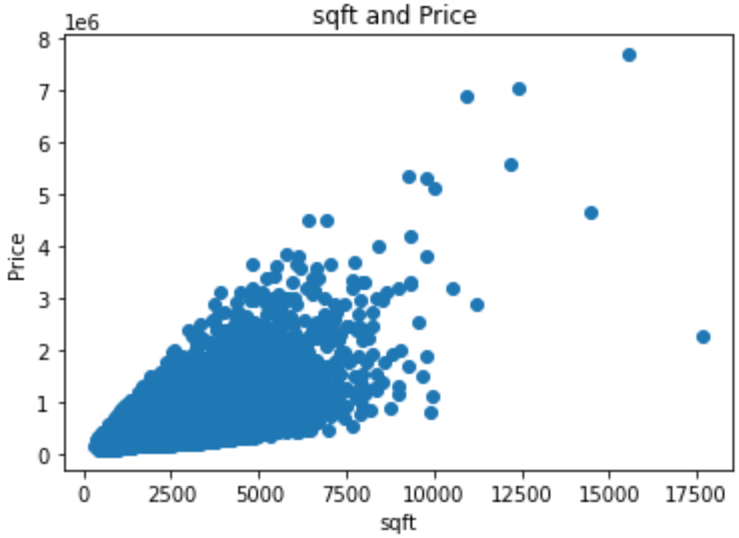
In [14]:
```
plt.scatter(data.bathrooms,data.price)
plt.title("Bathrooms and Price ")
plt.xlabel("Bathrooms")
plt.ylabel("Price")
plt.show()
```



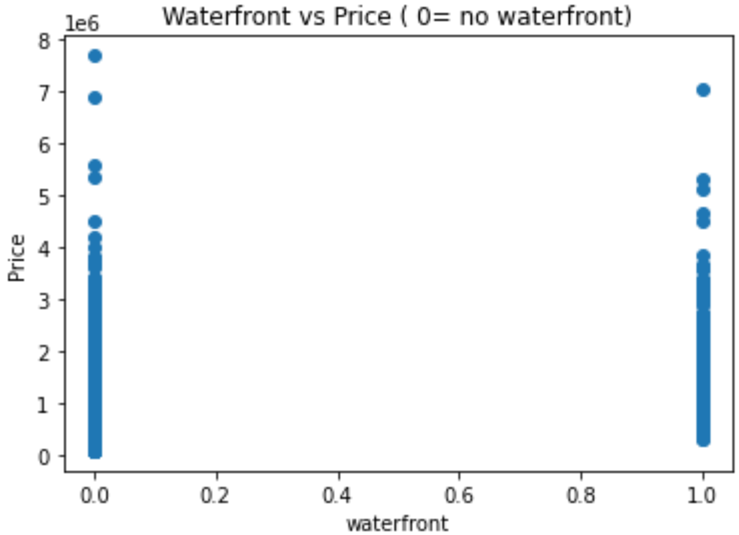
In [15]:
```
plt.scatter(data.bedrooms,data.price)
plt.title("Bedroom and Price ")
plt.xlabel("Bedrooms")
plt.ylabel("Price")
plt.show()
```



In [16]:
```
plt.scatter((data['sqft_living']+data['sqft_basement']),data['price'])
plt.title("sqft and Price ")
plt.xlabel("sqft")
plt.ylabel("Price")
plt.show()
```



In [17]:
```
plt.scatter(data.waterfront,data.price)
plt.title("Waterfront vs Price ( 0= no waterfront)")
plt.xlabel("waterfront")
plt.ylabel("Price")
plt.show()
```

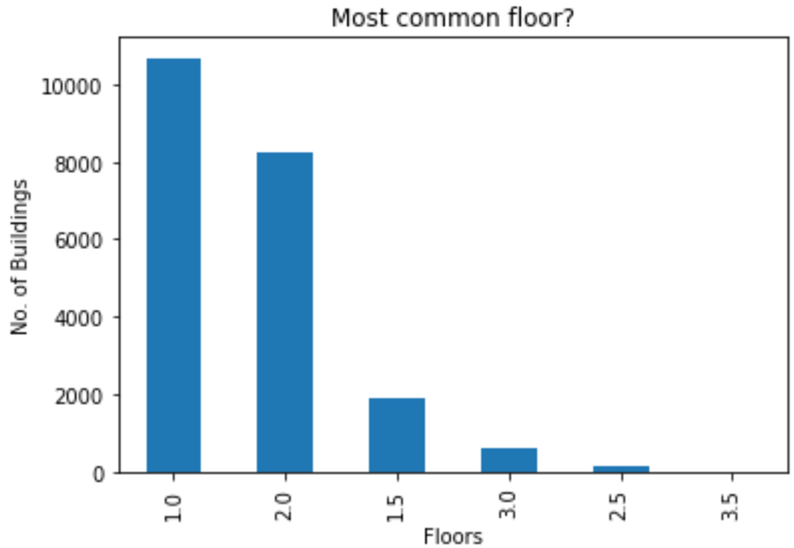


Lets check if floor is an important factor

In [18]:

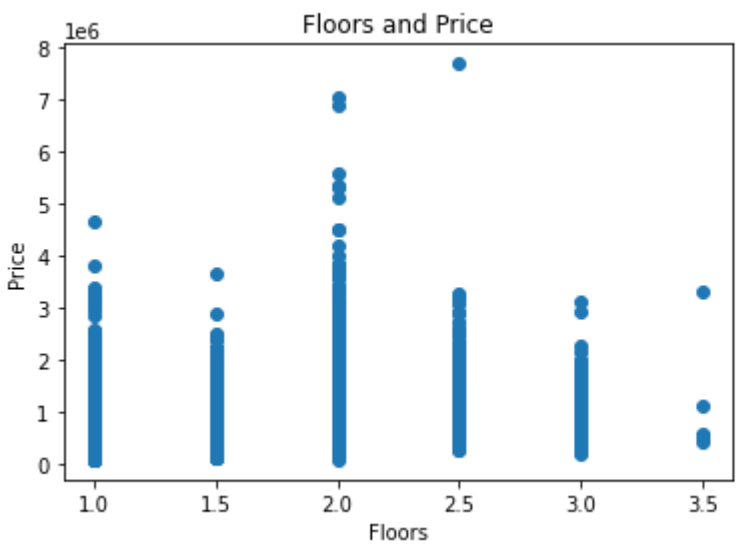
```
data.floors.value_counts().plot(kind='bar')
plt.title("Most common floor?")

plt.xlabel("Floors")
plt.ylabel("No. of Buildings")
plt.show()
```



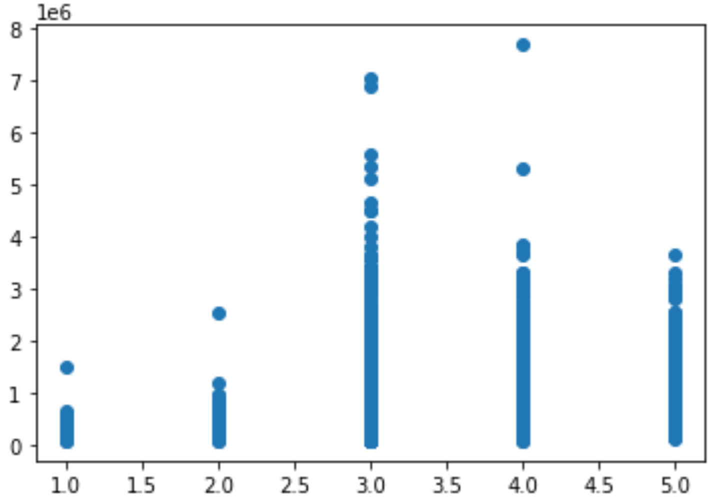
In [19]:
```
plt.scatter(data.floors,data.price)
plt.title("Floors and Price")

plt.xlabel("Floors")
plt.ylabel("Price")
plt.show()
```



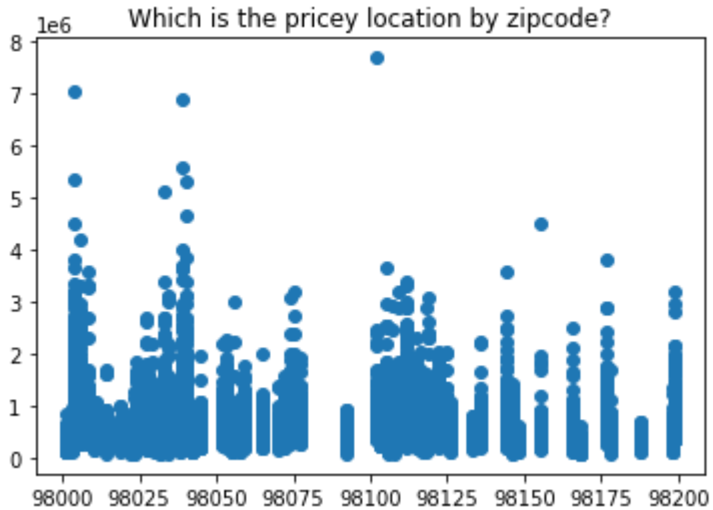
In [20]:
```
plt.scatter(data.condition,data.price)
```

Out[20]: `<matplotlib.collections.PathCollection at 0x7fc27473b438>`



In [21]:
```
plt.scatter(data.zipcode,data.price)
plt.title("Which is the pricey location by zipcode?")
```

Out[21]: `Text(0.5, 1.0, 'Which is the pricey location by zipcode?')`



In [ ]:

Now lets apply the linear regression

In [22]:
```
data.head()
```

Out[22]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	s
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

Regression parameters

- Dependent variable , Y -> price i.e. Output Label
- Independent variable, X -> others like floors, bedrooms, bathrooms, etc.
- Necessary data preprocessing:
  - date should not influence price much BUT new and older houses cost differently
  - So, lets convert dates to 1's and 0's so that it doesn't influence our data much
    - 0 for houses which are new that is built after 2014.
    - 1 for older houses

In [26]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
labels = data['price']
```

In [27]:
```python
# date preprocessing
conv_dates = [1 if values == 2014 else 0 for values in data.date ]
data['date'] = conv_dates

data.head(2)
```

Out[27]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement
0	7129300520	0	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0
1	6414100192	0	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400

2 rows × 21 columns

In [28]:
```python
train1 = data.drop(['id', 'price'],axis=1)

train1.head(2)
```

Out[28]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_re
0	0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1955	
1	0	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1951	

train and test the model

- popular train test ration -> 8:2

In [29]:
```python
from sklearn.model_selection import train_test_split
```

In [43]:
```python
x_train , x_test , y_train , y_test = train_test_split(train1 , labels , test_size = 0.20,random_state =2)
```

Train and Test dataset are ready. Now, lets fit the train and test data into the regression model.

In [49]:
```python
reg.fit(x_train,y_train)
```

Out[49]:
```
LinearRegression()
```

In [50]:
```
reg.score(x_test,y_test)
```

Out[50]:
```
0.7320342760357743
```

try using other test_size and see

In [ ]:
```

```

Can we improve the accuracy??

Testing Gradient boosting regression for building a prediction model

- Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.
- More: http://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

In [46]:
```
from sklearn import ensemble
clf = ensemble.GradientBoostingRegressor(n_estimators = 400, max_depth = 5, min_samples_split = 2,
          learning_rate = 0.1, loss = 'ls')
# n_estimator — The number of boosting stages to perform. We should not set it too high which would overfit
# max_depth — The depth of the tree node.
# learning_rate — Rate of learning the data.
# loss — loss function to be optimized. ‘ls’ refers to least squares regression
# minimum sample split — Number of sample to be split for learning the data
```

In [47]:
```
clf.fit(x_train, y_train)
```

Out[47]:
```
GradientBoostingRegressor(max_depth=5, n_estimators=400)
```

In [48]:
```
clf.score(x_test,y_test)
```

Out[48]:
```
0.9185553248258101
```

Accuracy improved after boosting the classifier!!

References

https://machinelearningmastery.com/linear-regression-for-machine-learning/

https://www.youtube.com/watch?v=8onB7rPG4Pk

https://github.com/llSourcell/math_of_machine_learning/blob/master/housesales.ipynb

https://towardsdatascience.com/create-a-model-to-predict-house-prices-using-python-d34fe8fad88f

https://linuxhint.com/house-price-prediction-linear-regression/

In [ ]:

```
```