jupyter RegressionAnalysisHouseValue Last Checkpoint: Last Friday at 10:19 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Python [conda env:gl-env]

Code CellToolbar

Predicting house value using machine learning regression analysis

author: bhavesh patel

credit: GraphLab

some definitions.

rmse = root mean squared error. This is used to identify errors and compare different models. rss = residual sum of squares is an error metric for regression.

These are two common measures of error regression, and RMSE is simply the square root of the mean RSS:

rmse = square root of (rss/n) where n=number of data points.

Predicting house value with regression analysis.

In [2]:
```
import graphlab
```

In [3]:
```
# Limit number of worker processes. This preserves system memory, which prevents hosted notebooks from crashing.
graphlab.set_runtime_config('GRAPHLAB_DEFAULT_NUM_PYLAMBDA_WORKERS', 4)
```

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: /tmp/graphlab_server_1479486745.log
```

```
This non-commercial license of GraphLab Create for academic use is assigned to bhaveshhk8@gmail.com and will expire o
n October 17, 2017.
```

In [4]:
```
# load data.
homesales = graphlab.SFrame('home_data.gl')
```

In [5]:
```
homesales
```

Out[5]:

6414100192	2014-12-09 00:00:00+00:00	538000	3	2.25	2570	7242	2	0
5631500400	2015-02-25 00:00:00+00:00	180000	2	1	770	10000	1	0
2487200875	2014-12-09 00:00:00+00:00	604000	4	3	1960	5000	1	0
1954400510	2015-02-18 00:00:00+00:00	510000	3	2	1680	8080	1	0
7237550310	2014-05-12 00:00:00+00:00	1225000	4	4.5	5420	101930	1	0
1321400060	2014-06-27 00:00:00+00:00	257500	3	2.25	1715	6819	2	0
2008000270	2015-01-15 00:00:00+00:00	291850	3	1.5	1060	9711	1	0
2414600126	2015-04-15 00:00:00+00:00	229500	3	1	1780	7470	1	0
3793500160	2015-03-12 00:00:00+00:00	323000	3	2.5	1890	6560	2	0

view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
0	3	7	1180	0	1955	0	98178	47.51123398
0	3	7	2170	400	1951	1991	98125	47.72102274
0	3	6	770	0	1933	0	98028	47.73792661
0	5	7	1050	910	1965	0	98136	47.52082
0	3	8	1680	0	1987	0	98074	47.61681228
0	3	11	3890	1530	2001	0	98053	47.65611835
0	3	7	1715	0	1995	0	98003	47.30972002
0	3	7	1060	0	1963	0	98198	47.40949984
0	3	7	1050	730	1960	0	98146	47.51229381
0	3	7	1890	0	2003	0	98038	47.36840673

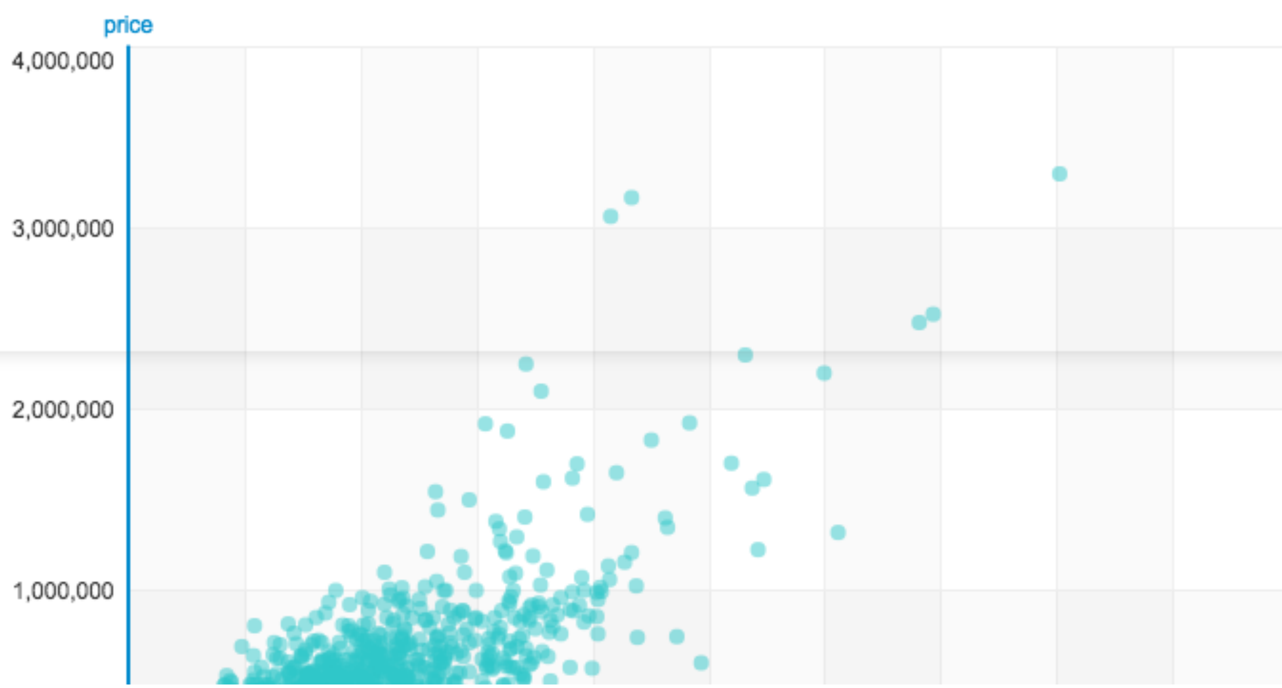
long	sqft_living15	sqft_lot15
-122.25677536	1340.0	5650.0
-122.3188624	1690.0	7639.0
-122.23319601	2720.0	8062.0

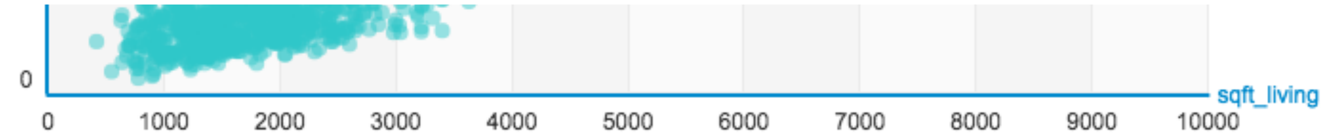
-122.39318505	1360.0	5000.0
-122.04490059	1800.0	7503.0
-122.00528655	4760.0	101930.0
-122.32704857	2238.0	6819.0
-122.31457273	1650.0	9711.0
-122.33659507	1780.0	8113.0
-122.0308176	2390.0	7570.0

[21613 rows x 21 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

In [6]:
```
# show graphis here, not a pop up window.
graphlab.canvas.set_target('ipynb')
```

In [7]:
```
# there are total 21K+ rows.  Let's view the data.
homesales.show(view="Scatter Plot", x="sqft_living", y="price")
```





Let's create regression model for prediction.

In [8]:
```
# first get training data set and test data set.  80% training data, 20% test data.
train_data, test_data = homesales.random_split(0.8, seed=0)
```

In [9]:
```
# build regression model with one variable for sq ft and store the results.
sqft_model = graphlab.linear_regression.create(train_data, target='price', features=['sqft_living'], validation_set=Non
```

```
Linear regression:

--------------------------------------------------------

Number of examples          : 17384

Number of features          : 1

Number of unpacked features : 1

Number of coefficients    : 2

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+--------------------+---------------+

| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |

+-----------+----------+--------------+--------------------+---------------+

| 1         | 2        | 1.014617     | 4349521.926170     | 262943.613754 |

+-----------+----------+--------------+--------------------+---------------+
```

In [10]:
```
# let's perform some functions.
print test_data['price'].mean()
```

```
SUCCESS: Optimal solution found.
```

```
543054.042563
```

In [11]:
```
print sqft_model.evaluate(test_data)
```

```
{'max_error': 4143550.8825285938, 'rmse': 255191.02870527358}
```

In [12]:
```
print train_data['price'].mean()
```

```
539366.628221
```

In [13]:
```
print sqft_model.evaluate(train_data)
```

```
{'max_error': 4349521.9261700595, 'rmse': 262943.6137536495}
```

In [14]:
```
sqft_model.get('coefficients')
```

Out[14]:

name	index	value	stderr
(intercept)	None	-47114.0206702	4923.34437753
sqft_living	None	281.957850166	2.16405465323

[2 rows x 4 columns]

In [15]:
```
sqft_model.show()
```

Schema	
Number of coefficients	2
Number of examples	17384
Number of feature columns	1
Number of unpacked features	1
Hyperparameters	
L1 penalty	0
L2 penalty	0.01
Training Summary	

Training Summary	
Solver	newton
Solver iterations	1
Solver status	SUCCESS: Optimal solution found.
Training time (sec)	1.0201

Settings	
Residual sum of squares	1201918356336392
Training RMSE	262943.6138

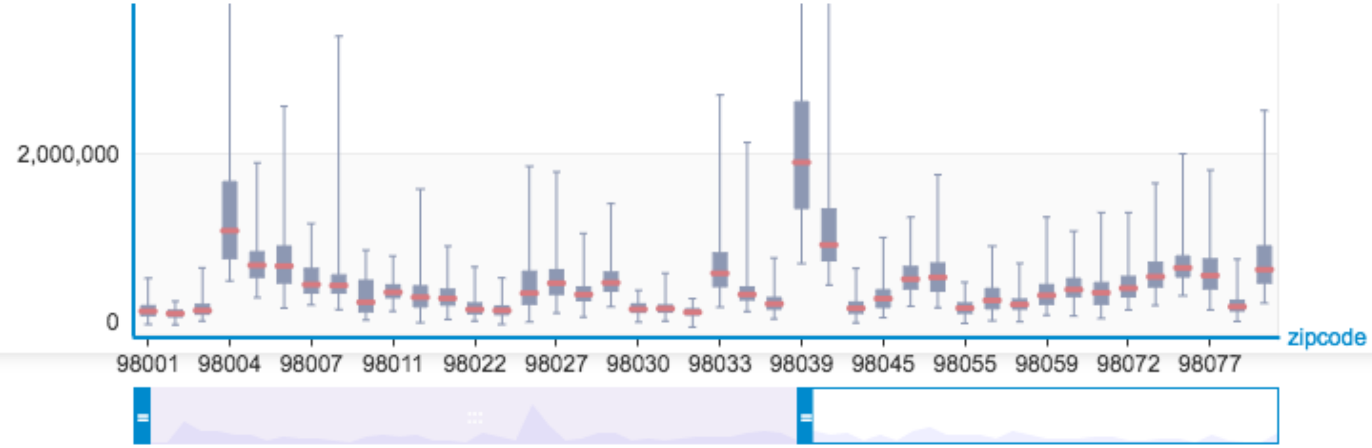
Highest Positive Coefficients

sqft_living	281.958

Lowest Negative Coefficients

(intercept)	-47,114.021

In [16]:
```
homesales.show(view='BoxWhisker Plot', x='zipcode', y='price')
```



Explore more attributes in linear regression.

zip code 98039 has most expensive houses.

```
In [17]: my_features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'zipcode']
```

```
In [18]: homesales[my_features].show()
```

bedrooms

dtype:	str
num_unique (est.):	13
num_undefined:	0

frequent items:

3
4
2
5
6
1

bathrooms

dtype:	str
num_unique (est.):	30
num_undefined:	0

frequent items:

2.5
1
1.75
2.25
2
1.5

sqft_living

dtype:	int
num_unique (est.):	1,036
num_undefined:	0
min:	290
max:	13,540
median:	1,910
mean:	2,079.9
std:	918.42

distribution of values:

sqft_lot

dtype:	int
num_unique (est.):	9,747
num_undefined:	0
min:	520
max:	1,651,359
median:	7,617
mean:	15,106.968
std:	41,419.553

distribution of values:

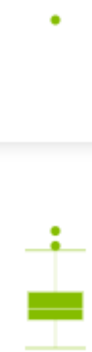
floors

dtype:	s
num_unique (est.):	6
num_undefined:	0

frequent items:

1
2
1.5
3
2.5
3.5

7	2.75
0	3
8	3.5
9	3.25
10	3.75
11	4



now let's build another regression model with more attributes.

In [19]:
```
more_attribute_model=graphlab.linear_regression.create(train_data, target='price', features=my_features,validation_set=N
```

```
Linear regression:
--------------------------------------------------------
Number of examples          : 17384
Number of features          : 6
Number of unpacked features : 6
Number of coefficients    : 115
Starting Newton Method
--------------------------------------------------------
+-----------+----------+--------------+--------------------+---------------+
| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |
+-----------+----------+--------------+--------------------+---------------+
| 1         | 2        | 0.041900     | 3763208.270523     | 181908.848367 |
+-----------+----------+--------------+--------------------+---------------+
SUCCESS: Optimal solution found.
```

In [20]:
```
# now let's compare the two model.
print sqft_model.evaluate(test_data)
print more_attribute_model.evaluate(test_data)
```

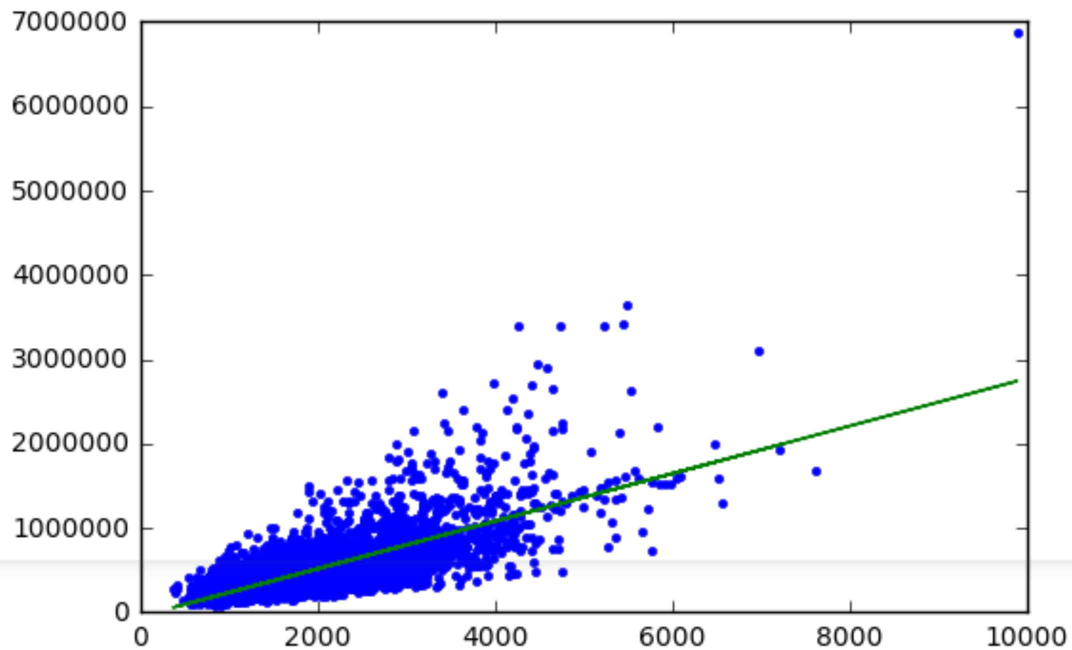
```
{'max_error': 4143550.8825285938, 'rmse': 255191.02870527358}
{'max_error': 3486584.509381705, 'rmse': 179542.4333126903}
```

In [21]:
```
# with more attributes, we can see that error have reduced and RMSE has reduced too.
# That's good and bad.  We will see some overfitting in predicting house value.
```

In [22]:
```
# let's get matplotlib to plot the regression analysis.
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(test_data['sqft_living'],test_data['price'],'.',
        test_data['sqft_living'],sqft_model.predict(test_data),'-')
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x11c822750>,
 <matplotlib.lines.Line2D at 0x11c822810>]
```

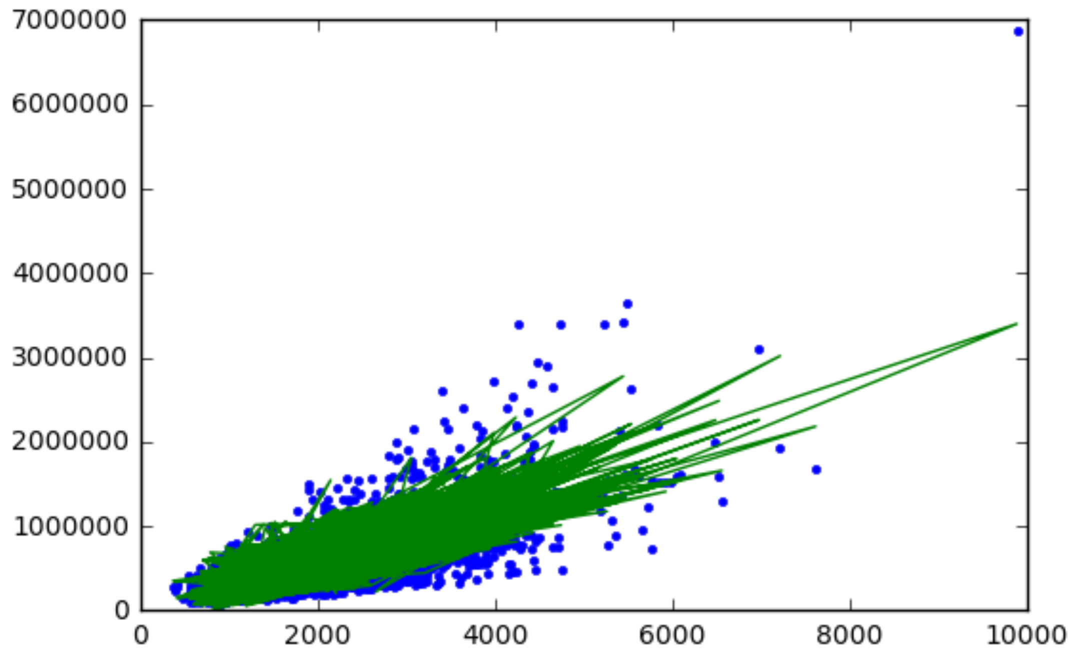


In [23]:
```
#Ok now let's build the model with second regression we built with more attributes.

plt.plot(test_data['sqft_living'],test_data['price'],'.',
```

```
        test_data['sqft_living'],more_attribute_model.predict(test_data),'-')
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x11cb522d0>,
 <matplotlib.lines.Line2D at 0x11cb52390>]
```



In [30]:
```
# now let's get one of the house id and try to predict the value of that house using both the regression models.
# we needt to get the object for one of the house.  Let's pick the first house in the data, which has id
# of 7129300520.

house1=homesales[homesales['id']=='7129300520']
```

In [31]:
```
house1
```

Out[31]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
7129300520	2014-10-13 00:00:00+00:00	221900	3	1	1180	5650	1	0

view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
0	3	7	1180	0	1955	0	98178	47.51123398

long	sqft_living15	sqft_lot15
[illegible]	[illegible]	[illegible]

-122.25677536	1340.0	5650.0

[? rows x 21 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.

In [32]:
```
# now let's use this object and call our regression analysis model to predict house price.

print sqft_model.predict(house1)
print more_attribute_model.predict(house1)
```

```
[285596.24252564885]
[211196.3344448047]
```

In [ ]:
```
# ok.  So actual house value is $221,900.  Our SQFT model predicted house value of $285,596 while our more attribute
# model predicted house value of $211,196.  So we can say that more attribute value is showing us correct results.
# Well, its art with science.  It is not always true.  See below example for other way round.
```

In [33]:
```
# now let's pick another house:  5309101200

house2 = homesales[homesales['id']=='5309101200']
house2
```

Out[33]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
5309101200	2014-06-05 00:00:00+00:00	620000	4	2.25	2400	5350	1.5	0

view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
0	4	7	1460	940	1929	0	98117	47.67632376

long	sqft_living15	sqft_lot15
-122.37010126	1250.0	4880.0

[? rows x 21 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.

In [34]:
```
# now let's predict values for this house.

print sqft_model.predict(house2)
print more_attribute_model.predict(house2)
```

```
[629584.8197281545]
```

```
[721918.9333272863]
```

In [35]:
```
# In this case, actual house value is: $620,000.  SQFT model predicted $629,584 while additional attribute model
# predicted $721,918.  Really bad!
```

In [38]:
```
# now let's select the house with zip code

housesHighValueZip=homesales[homesales['zipcode']=='98039']
housesHighValueZip
```

Out[38]:

2540700110	2015-02-12 00:00:00+00:00	1905000	4	3.5	4210	18564	2	0
3262300940	2014-11-07 00:00:00+00:00	875000	3	1	1220	8119	1	0
3262300940	2015-02-10 00:00:00+00:00	940000	3	1	1220	8119	1	0
6447300265	2014-10-14 00:00:00+00:00	4000000	4	5.5	7080	16573	2	0
2470100110	2014-08-04 00:00:00+00:00	5570000	5	5.75	9200	35069	2	0
2210500019	2015-03-24 00:00:00+00:00	937500	3	1	1320	8500	1	0
6447300345	2015-04-06 00:00:00+00:00	1160000	4	3	2680	15438	2	0
6447300225	2014-11-06 00:00:00+00:00	1880000	3	2.75	2620	17919	1	0
2525049148	2014-10-07 00:00:00+00:00	3418800	5	5	5450	20412	2	0

view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
0	3	12	4860	0	1996	0	98039	47.61717049
0	3	11	4210	0	2001	0	98039	47.62060082
0	4	7	1220	0	1955	0	98039	47.63281908
0	4	7	1220	0	1955	0	98039	47.63281908
0	3	12	5760	1320	2008	0	98039	47.61512031
0	3	13	6200	3000	2001	0	98039	47.62888314
0	4	7	1320	0	1954	0	98039	47.61872888
2	3	8	2680	0	1902	1956	98039	47.61089438
1	4	9	2620	0	1949	0	98039	47.61435052
0	3	11	5450	0	2014	0	98039	47.62087993

long	sqft_living15	sqft_lot15
-122.23040939	3580.0	16054.0
-122.2245047	3520.0	18564.0
-122.23554392	1910.0	8119.0
-122.23554392	1910.0	8119.0
-122.22420058	3140.0	15996.0
-122.23346379	3560.0	24345.0
-122.22643371	2790.0	10800.0
-122.22582388	4480.0	14406.0
-122.22772057	3400.0	14400.0
-122.23726918	3160.0	17825.0

[? rows x 21 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.

In [43]:

```
# what's average price for the zip?

print housesHighValueZip['price'].mean()
```

```
2160606.6
```

In [47]:

```
# SFrame filtering.
#select the houses that have ‘sqft_living’ higher than 2000 sqft but no larger than 4000 sqft.

houseFilter1 = homesales[(homesales['sqft_living'] > 2000) & (homesales['sqft_living'] <= 4000)]
houseFilter1
```

Out[47]:

1736800520	2015-04-03 00:00:00+00:00	662500	3	2.5	3560	9796	1	0
9297300055	2015-01-24 00:00:00+00:00	650000	4	3	2950	5000	2	0
2524049179	2014-08-26 00:00:00+00:00	2000000	3	2.75	3050	44867	1	0
7137970340	2014-07-03 00:00:00+00:00	285000	5	2.5	2270	6300	2	0

3814700200	2014-11-20 00:00:00+00:00	329000	3	2.25	2450	6500	2	0
1794500383	2014-06-26 00:00:00+00:00	937000	3	1.75	2450	2691	2	0
1873100390	2015-03-02 00:00:00+00:00	719000	4	2.5	2570	7173	2	0
8562750320	2014-11-10 00:00:00+00:00	580500	3	2.5	2320	3980	2	0
0461000390	2014-06-24 00:00:00+00:00	687500	4	1.75	2330	5000	1.5	0

view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
0	3	7	2170	400	1951	1991	98125	47.72102274
0	3	8	1860	1700	1965	0	98007	47.60065993
3	3	9	1980	970	1979	0	98126	47.57136955
4	3	9	2330	720	1968	0	98040	47.53164379
0	3	8	2270	0	1995	0	98092	47.32658071
0	4	8	2450	0	1985	0	98030	47.37386303
0	3	8	1750	700	1915	0	98119	47.63855772
0	3	8	2570	0	2005	0	98052	47.70732168
0	3	8	2320	0	2003	0	98027	47.5391103
0	4	7	1510	820	1929	0	98117	47.68228235

long	sqft_living15	sqft_lot15
-122.3188624	1690.0	7639.0
-122.14529566	2210.0	8925.0
-122.37541218	2140.0	4000.0
-122.23345881	4110.0	20336.0
-122.16892624	2240.0	7005.0
-122.17228981	2200.0	6865.0
-122.35985573	1760.0	3573.0
-122.11029785	2630.0	6026.0
-122.06971484	2580.0	3980.0

-122.36760203	1460.0	5000.0

[? rows x 21 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use sf.materialize() to force materialization.

In [48]:
```
houseFilter1.show()
```

id

dtype:	str
num_unique (est.):	9,071
num_undefined:	0

frequent items:

5332200530
9250900104
1254200015
1522059120
1561930020
1568100300
1630700361
1823059205
1825069031
1954420170
1974300020
2143700830

date

dtype:	datetime
num_unique (est.):	353
num_undefined:	0

frequent items:

2014-06-26 00:00:00+...
2014-07-08 00:00:00+...
2014-07-29 00:00:00+...
2014-06-20 00:00:00+...
2015-04-21 00:00:00+...
2015-03-25 00:00:00+...
2014-07-01 00:00:00+...
2014-08-27 00:00:00+...
2014-06-23 00:00:00+...
2015-04-23 00:00:00+...
2015-04-27 00:00:00+...
2014-08-26 00:00:00+...

price

dtype:	int
num_unique (est.):	2,415
num_undefined:	0
min:	194,250
max:	3,100,000
median:	598,500
mean:	653,299.737
std:	312,808.223

distribution of values:

bedrooms

dtype:	str
num_unique (est.):	12
num_undefined:	0

frequent items:

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

bathrooms

dtype:	
num_unique (est.):	
num_undefined:	

frequent items:

2.5
2.25
2.75
1.75
3
2
3.5
3.25
1.5
1
3.75
4

In [52]:
```
# alright, now to regression model 3, where we use lot more features.

even_more_features = [
```

```
'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'zipcode',
'condition', # condition of house
'grade', # measure of quality of construction
'waterfront', # waterfront property
'view', # type of view
'sqft_above', # square feet above ground
'sqft_basement', # square feet in basement
'yr_built', # the year built
'yr_renovated', # the year renovated
'lat', 'long', # the lat-long of the parcel
'sqft_living15', # average sq.ft. of 15 nearest neighbors
'sqft_lot15', # average lot size of 15 nearest neighbors
]
```

In [53]:

```
# build regression model using these features.
# more_attribute_model=graphlab.linear_regression.create(train_data, target='price', features=my_features,validation_se

even_more_att_model=graphlab.linear_regression.create(train_data, target='price', features=even_more_features,validatio
```

```
Linear regression:

--------------------------------------------------------

Number of examples          : 17384

Number of features          : 18

Number of unpacked features : 18

Number of coefficients    : 127

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+--------------------+---------------+

| Iteration | Passes   | Elapsed Time | Training-max_error | Training-rmse |

+-----------+----------+--------------+--------------------+---------------+

| 1         | 2        | 0.071567     | 3469012.450686     | 154580.940736 |

+-----------+----------+--------------+--------------------+---------------+

SUCCESS: Optimal solution found.
```

In [60]:
```
# now let's get RMSE value for all three models we built.

print sqft_model.evaluate(test_data)
print more_attribute_model.evaluate(test_data)
print even_more_att_model.evaluate(test_data)
```

```
{'max_error': 4143550.8825285938, 'rmse': 255191.02870527358}
{'max_error': 3486584.509381705, 'rmse': 179542.4333126903}
{'max_error': 3556849.413858208, 'rmse': 156831.1168021901}
```