Fire up Turi Create

In [2]:
```
import turicreate
```

Load some house value vs. crime rate data

Dataset is from Philadelphia, PA and includes average house sales price in a number of neighborhoods. The attributes of each neighborhood we have include the crime rate ('CrimeRate'), miles from Center City ('MilesPhila'), town name ('Name'), and county name ('County').

In [4]:
```
sales = turicreate.SFrame('https://courses.cs.washington.edu/courses/cse416/
```

```
Finished parsing file https://courses.cs.washington.edu/courses/cse416/18
sp/notebooks/Philadelphia_Crime_Rate_noNA.csv

Parsing completed. Parsed 99 lines in 0.037553 secs.

------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,float,float,float,float,str,str]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------

Finished parsing file https://courses.cs.washington.edu/courses/cse416/18
sp/notebooks/Philadelphia_Crime_Rate_noNA.csv

Parsing completed. Parsed 99 lines in 0.007906 secs.
```

In [3]:
```
sales
```

Out[3]:

HousePrice	HsPrc ($10,000)	CrimeRate	MilesPhila	PopChg	Name	County
140463	14.0463	29.7	10.0	-1.0	Abington	Montgome
113033	11.3033	24.1	18.0	4.0	Ambler	Montgome
124186	12.4186	19.5	25.0	8.0	Aston	Delaware
110490	11.049	49.4	25.0	2.7	Bensalem	Bucks
79124	7.9124	54.1	19.0	3.9	Bristol B.	Bucks
92634	9.2634	48.6	20.0	0.6	Bristol T.	Bucks
89246	8.9246	30.8	15.0	-2.6	Brookhaven	Delaware
195145	19.5145	10.8	20.0	-3.5	Bryn Athyn	Montgome
297342	29.7342	20.2	14.0	0.6	Bryn Mawr	Montgome
264298	26.4298	20.4	26.0	6.0	Buckingham	Bucks

[99 rows x 7 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.

Exploring the data

The house price in a town is correlated with the crime rate of that town. Low crime towns tend to be associated with higher house prices and vice versa.

In [4]:
```
turicreate.show(sales["CrimeRate"], sales["HousePrice"])
```

...

Fit the regression model using crime as the feature

In [5]:
```
crime_model = turicreate.linear_regression.create(
    sales,
    target="HousePrice",
    features=['CrimeRate'],
    validation_set=None,
    verbose=False
)
```

Let's see what our fit looks like

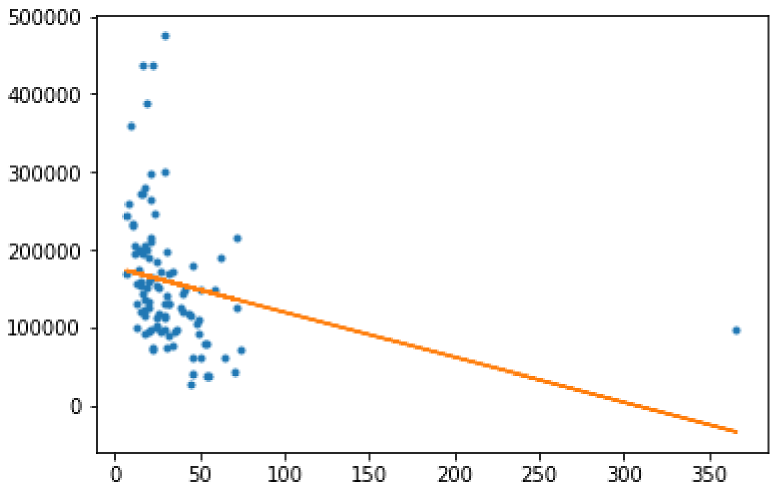
Matplotlib is a Python plotting library that is also useful for plotting. You can install it with:

'pip install matplotlib'

In [6]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [7]:
```
plt.plot(sales['CrimeRate'],sales['HousePrice'],'.',
        sales['CrimeRate'],crime_model.predict(sales),'-')
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x12700ab10>,
 <matplotlib.lines.Line2D at 0x12700ac50>]
```



Above: blue dots are original data, green line is the fit from the simple regression.

Remove Center City and redo the analysis

Center City is the one observation with an extremely high crime rate, yet house prices are not very low. This point does not follow the trend of the rest of the data very well. A question is how much including Center City is influencing our fit on the other datapoints. Let's remove this datapoint and see what happens.

In [8]:
```
sales_noCC = sales[sales['MilesPhila'] != 0.0]
```

In [9]:
```
turicreate.show(sales_noCC['CrimeRate'], sales_noCC['HousePrice'])
```

...

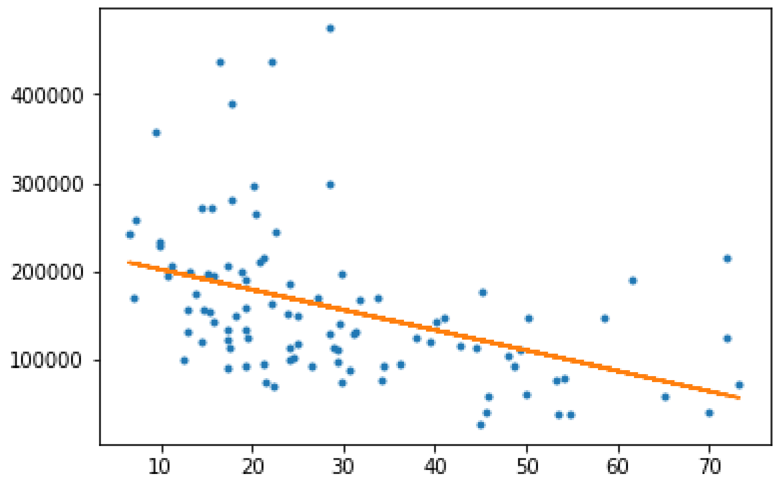
Refit our simple regression model on this modified dataset:

In [10]:
```
crime_model_noCC = turicreate.linear_regression.create(
    sales_noCC,
    target='HousePrice',
    features=['CrimeRate'],
    validation_set = None,
    verbose = False
)
```

Look at the fit:

In [11]:
```
plt.plot(sales_noCC['CrimeRate'], sales_noCC['HousePrice'], '.',
         sales_noCC['CrimeRate'], crime_model_noCC.predict(sales_noCC), '-')
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x12437ee90>,
 <matplotlib.lines.Line2D at 0x10cb593d0>]
```



Compare coefficients for full-data fit versus no-Center-City fit

Visually, the fit seems different, but let's quantify this by examining the estimated coefficients of our original fit and that of the modified dataset with Center City removed.

In [12]:
```
crime_model.coefficients
```

Out[12]:

name	index	value	stderr
(intercept)	None	176626.046881	11245.5882187
CrimeRate	None	-576.804949058	226.902259495

[2 rows x 4 columns]

In [13]:
```
crime_model_noCC.coefficients
```

Out[13]:

name	index	value	stderr
(intercept)	None	225204.604303	16404.0247483
CrimeRate	None	-2287.69717443	491.537478029

[2 rows x 4 columns]

Above: We see that for the "no Center City" version, per unit increase in crime, the predicted decrease in house prices is 2,287. In contrast, for the original dataset, the drop is only 576 per unit increase in crime. This is significantly different!

High leverage points:

Center City is said to be a "high leverage" point because it is at an extreme x value where there are not other observations. As a result, recalling the closed-form solution for simple regression, this point has the ***potential*** to dramatically change the least squares line since the center of x mass is heavily influenced by this one point and the least squares line will try to fit close to that outlying (in x) point. If a high leverage point follows the trend of the other data, this might not have much effect. On the other hand, if this point somehow differs, it can be strongly influential in the resulting fit.

Influential observations:

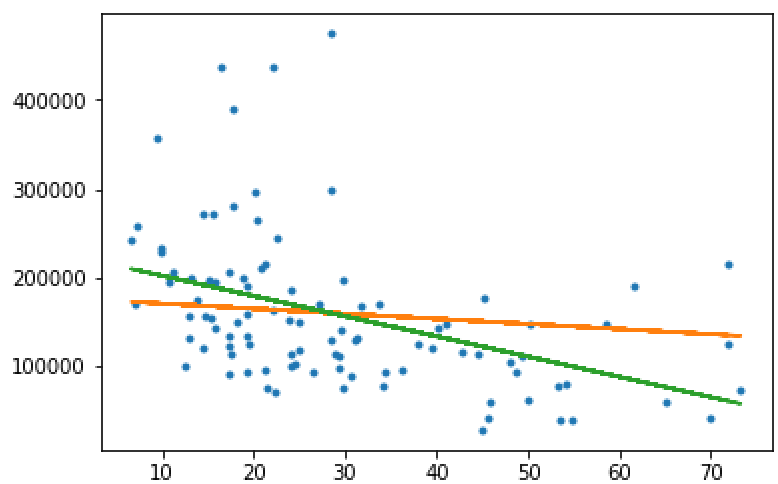
An influential observation is one where the removal of the point significantly changes the fit. As discussed above, high leverage points are good candidates for being influential observations, but need not be. Other observations that are ***not*** leverage points can also be influential observations (e.g., strongly outlying in y even if x is a typical value).

Plotting the two models

Confirm the above calculations by looking at the plots. The orange line is the model trained removing Center City, and the green line is the model trained on all the data. Notice how much steeper the green line is, since the drop in value is much higher according to this model.

In [14]:
```
plt.plot(sales_noCC['CrimeRate'], sales_noCC['HousePrice'], '.',
         sales_noCC['CrimeRate'], crime_model.predict(sales_noCC), '-',
         sales_noCC['CrimeRate'], crime_model_noCC.predict(sales_noCC), '-')
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x12437ec50>,
 <matplotlib.lines.Line2D at 0x10cb8dc10>,
 <matplotlib.lines.Line2D at 0x10cb9a490>]
```



Remove high-value outlier neighborhoods and redo analysis

Based on the discussion above, a question is whether the outlying high-value towns are strongly influencing the fit. Let's remove them and see what happens.

In [15]:
```
sales_nohighend = sales_noCC[sales_noCC['HousePrice'] < 350000]
crime_model_nohighend = turicreate.linear_regression.create(
    sales_nohighend,
    target='HousePrice',
    features=['CrimeRate'],
    validation_set=None,
    verbose=False
)
```

Do the coefficients change much?

In [16]:
```
crime_model_noCC.coefficients
```

Out[16]:

name	index	value	stderr
(intercept)	None	225204.604303	16404.0247483
CrimeRate	None	-2287.69717443	491.537478029

[2 rows x 4 columns]

In [17]:
```
crime_model_nohighend.coefficients
```

Out[17]:

name	index	value	stderr
(intercept)	None	199073.589615	11932.510108
CrimeRate	None	-1837.71280989	351.519609261

[2 rows x 4 columns]

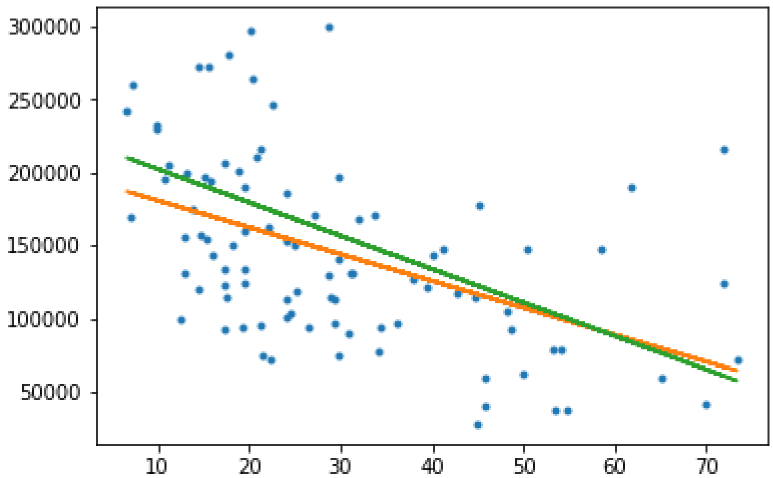
Above: We see that removing the outlying high-value neighborhoods has ***some*** effect on the fit, but not nearly as much as our high-leverage Center City datapoint.

Compare the two models

Confirm the above calculations by looking at the plots. The orange line is the no high-end model, and the green line is the no-city-center model.

In [18]:
```
plt.plot(sales_nohighend['CrimeRate'], sales_nohighend['HousePrice'], '.',
         sales_nohighend['CrimeRate'], crime_model_nohighend.predict(sales_r
         sales_nohighend['CrimeRate'], crime_model_noCC.predict(sales_nohigh
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x10cbb0a10>,
 <matplotlib.lines.Line2D at 0x124533310>,
 <matplotlib.lines.Line2D at 0x124533b50>]
```



In [ ]:

In [ ]:

In [ ]: