Regression Week 3: Assessing Fit (polynomial regression)

In this notebook you will compare different regression models in order to assess which model fits best. We will be using polynomial regression as a means to examine this topic. In particular you will:

- Write a function to take an SArray and a degree and return an SFrame where each column is the SArray to a polynomial value up to the total degree e.g. degree = 3 then column 1 is the SArray column 2 is the SArray squared and column 3 is the SArray cubed
- Use matplotlib to visualize polynomial regressions
- Use matplotlib to visualize the same polynomial degree on different subsets of the data
- Use a validation set to select a polynomial degree
- Assess the final fit using test data

We will continue to use the House data from previous notebooks.

Fire up graphlab create

In [6]:

```
import graphlab
```

Next we're going to write a polynomial function that takes an SArray and a maximal degree and returns an SFrame with columns containing the SArray to all the powers up to the maximal degree.
The easiest way to apply a power to an SArray is to use the .apply() and lambda x: functions. For example to take the example array and compute the third power we can do as follows: (note running this cell the first time may take longer than expected since it loads graphlab)

In [7]:

```
tmp = graphlab.SArray([1., 2., 3.])
tmp_cubed = tmp.apply(lambda x: x**3)
print tmp
print tmp_cubed
```

```
[1.0, 2.0, 3.0]
[1.0, 8.0, 27.0]
```

We can create an empty SFrame using graphlab.SFrame() and then add any columns to it with ex_sframe['column_name'] = value. For example we create an empty SFrame and make the column 'power_1' to be the first power of tmp (i.e. tmp itself).

In [8]:

```
ex_sframe = graphlab.SFrame()
ex_sframe['power_1'] = tmp
print ex_sframe
```

```
+---------+
| power_1 |
+---------+
|   1.0   |
|   2.0   |
|   3.0   |
+---------+
[3 rows x 1 columns]
```

Polynomial_sframe function

Using the hints above complete the following function to create an SFrame consisting of the powers of an SArray up to a specific degree:

In [9]:
```
def polynomial_sframe(feature, degree):
    # assume that degree >= 1
    # initialize the SFrame:
    poly_sframe = graphlab.SFrame()
    # and set poly_sframe['power_1'] equal to the passed feature
    poly_sframe['power_1'] = feature
    # first check if degree > 1
    if degree > 1:
        # then loop over the remaining degrees:
        # range usually starts at 0 and stops at the endpoint-1. We want
it to start at 2 and stop at degree
        for power in range(2, degree+1):
            # first we'll give the column a name:
            name = 'power_' + str(power)
            # then assign poly_sframe[name] to the appropriate power of
feature
            poly_sframe[name] = feature.apply(lambda x: x**power)
    return poly_sframe
```

To test your function consider the smaller tmp variable and what you would expect the outcome of the following call:

In [10]:
```
print polynomial_sframe(tmp, 3)
```

```
+---------+---------+---------+
| power_1 | power_2 | power_3 |
+---------+---------+---------+
|   1.0   |   1.0   |   1.0   |
|   2.0   |   4.0   |   8.0   |
|   3.0   |   9.0   |   27.0  |
+---------+---------+---------+
[3 rows x 3 columns]
```

Visualizing polynomial regression

Let's use matplotlib to visualize what a polynomial regression looks like on some real data.

In [11]:
```
sales = graphlab.SFrame('kc_house_data.gl/')
```

As in Week 3, we will use the sqft_living variable. For plotting purposes (connecting the dots), you'll need to sort by the values of sqft_living. For houses with identical square footage, we break the tie by their prices.

In [12]:
```
sales = sales.sort(['sqft_living', 'price'])
```

Let's start with a degree 1 polynomial using 'sqft_living' (i.e. a line) to predict 'price' and plot what it looks like.

In [13]:
```
poly1_data = polynomial_sframe(sales['sqft_living'], 1)
poly1_data['price'] = sales['price'] # add price to the data since it's
the target
```

NOTE: for all the models in this notebook use validation_set = None to ensure that all results are consistent across users.

In [14]:
```
model1 = graphlab.linear_regression.create(poly1_data, target = 'price',
features = ['power_1'], validation_set = None)
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 21613
PROGRESS: Number of features          : 1
PROGRESS: Number of unpacked features : 1
PROGRESS: Number of coefficients    : 2
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | 1         | 2        | 1.097063     | 4362074.696077     |
261440.790724 |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
```

In [15]:
```
#let's take a look at the weights before we plot
model1.get("coefficients")
```

Out[15]:

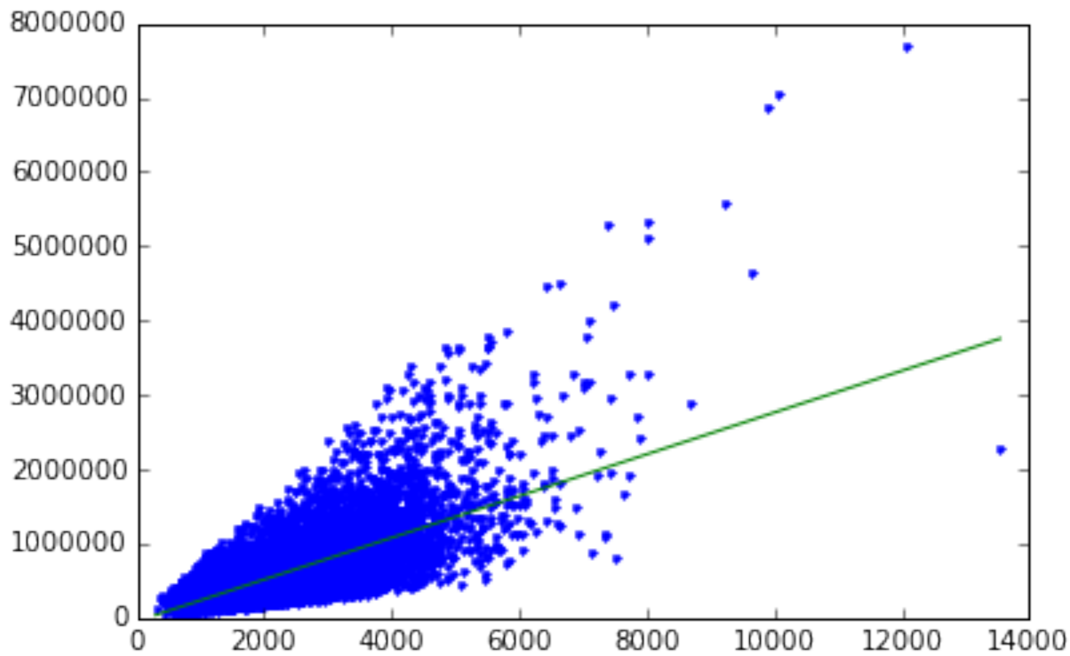
name	index	value
(intercept)	None	-43579.0852515
power_1	None	280.622770886

[2 rows x 3 columns]

In [16]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [17]:
```
plt.plot(poly1_data['power_1'],poly1_data['price'],'.',
        poly1_data['power_1'], model1.predict(poly1_data),'-')
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x21b6a6d8>,
 <matplotlib.lines.Line2D at 0x21cd10f0>]
```



Let's unpack that plt.plot() command. The first pair of SArrays we passed are the 1st power of sqft and the actual price we then ask it to print these as dots '.'. The next pair we pass is the 1st power of sqft and the predicted values from the linear model. We ask these to be plotted as a line '-'. We can see, not surprisingly, that the predicted values all fall on a line, specifically the one with slope 280 and intercept -43579. What if we wanted to plot a second degree polynomial?

In [18]:

```
poly2_data = polynomial_sframe(sales['sqft_living'], 2)
square_features = poly2_data.column_names() # get the name of the features
poly2_data['price'] = sales['price'] # add price to the data since it's
the target
model2 = graphlab.linear_regression.create(poly2_data, target = 'price',
features = square_features, validation_set = None)
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 21613
PROGRESS: Number of features          : 2
PROGRESS: Number of unpacked features : 2
PROGRESS: Number of coefficients    : 3
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | 1         | 2        | 0.009001     | 5913020.984255     |
250948.368758 |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: SUCCESS: Optimal solution found.
```

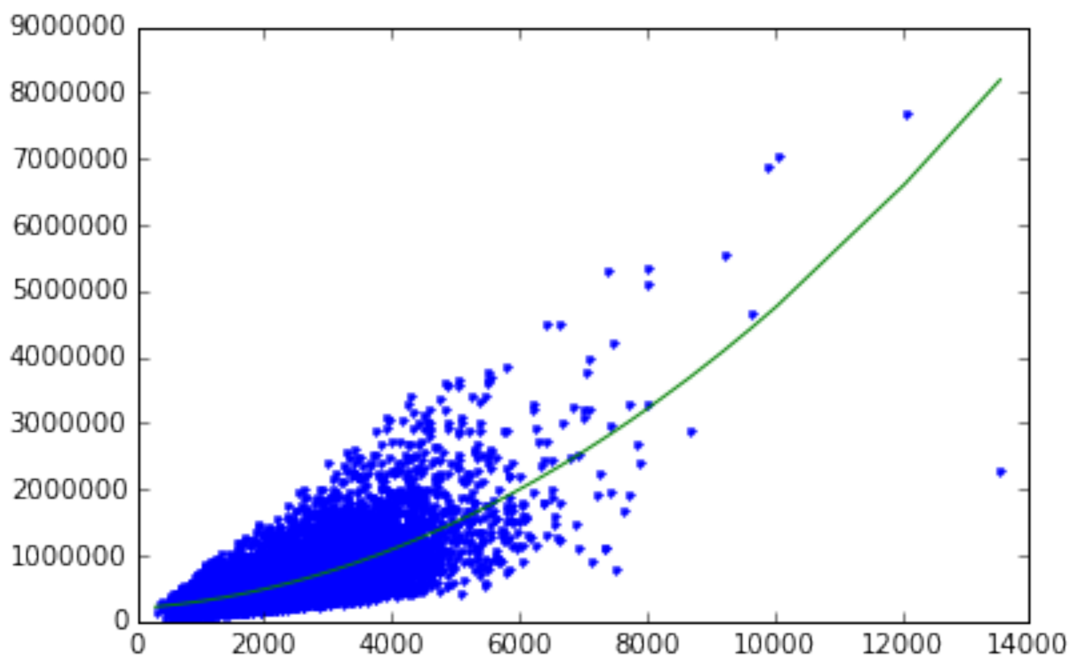
```
PROGRESS:
```

In [19]:
```
model2.get("coefficients")
```

Out[19]:

name	index	value
(intercept)	None	199222.496445
power_1	None	67.9940640677
power_2	None	0.0385812312789
[3 rows x 3 columns]		

In [20]:
```
plt.plot(poly2_data['power_1'],poly2_data['price'],'.',
        poly2_data['power_1'], model2.predict(poly2_data),'-')
```
Out[20]:
```
[<matplotlib.lines.Line2D at 0x22ae4978>,
 <matplotlib.lines.Line2D at 0x22ae4ba8>]
```



The resulting model looks like half a parabola. Try on your own to see what the cubic looks like:

In [21]:
```
poly3_data = polynomial_sframe(sales['sqft_living'], 3)
cube_features = poly3_data.column_names() # get the name of the features
poly3_data['price'] = sales['price'] # add price to the data since it's
the target
model3 = graphlab.linear_regression.create(poly3_data, target = 'price',
features = cube_features, validation_set = None)
```

```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 21613
PROGRESS: Number of features          : 3
PROGRESS: Number of unpacked features : 3
PROGRESS: Number of coefficients    : 4
```

```
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error |
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | 1         | 2        | 0.012001     | 3261066.736007     |
249261.286346 |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
```

In [25]:

```
model3.get("coefficients")
```

Out[25]:

name	index	value
(intercept)	None	336788.117952
power_1	None	-90.1476236119
power_2	None	0.087036715081
power_3	None	-3.8398521196e-06

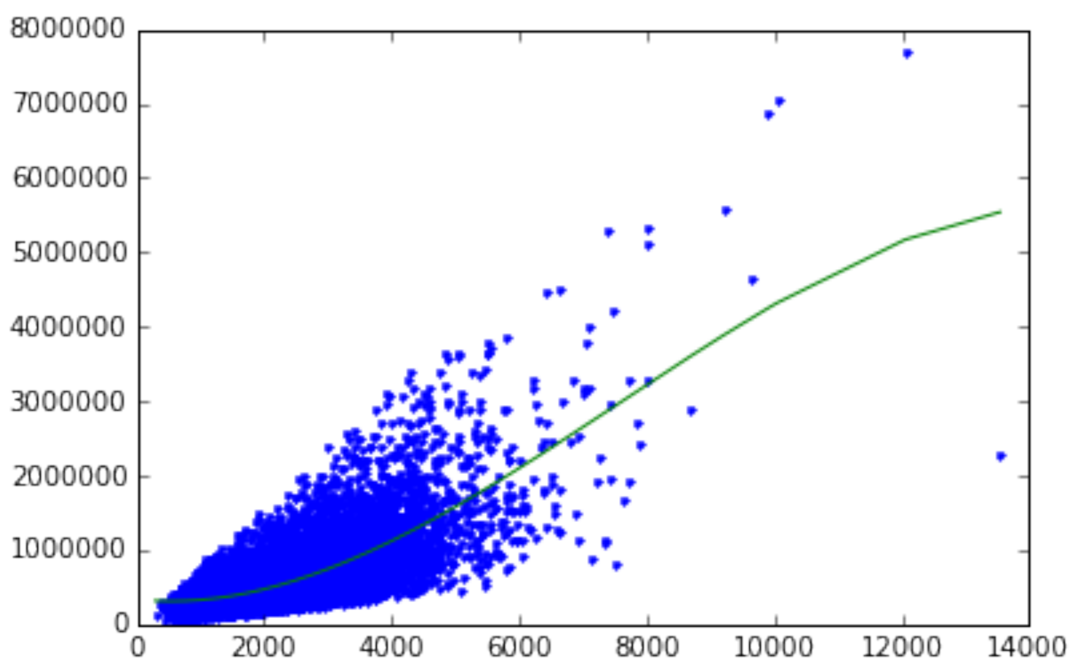
[4 rows x 3 columns]

In [22]:

```
plt.plot(poly3_data['power_1'],poly3_data['price'],'.',
        poly3_data['power_1'], model3.predict(poly3_data),'-')
```

Out[22]:

```
[<matplotlib.lines.Line2D at 0x2315da20>,
 <matplotlib.lines.Line2D at 0x2315dc18>]
```



Now try a 15th degree polynomial:

In [23]:

```
poly15_data = polynomial_sframe(sales['sqft_living'], 15)
fifteen_features = poly15_data.column_names() # get the name of the 
features
poly15_data['price'] = sales['price'] # add price to the data since it's 
the target
model15 = graphlab.linear_regression.create(poly15_data, target = 'price', 
features = fifteen_features, validation_set = None)
```

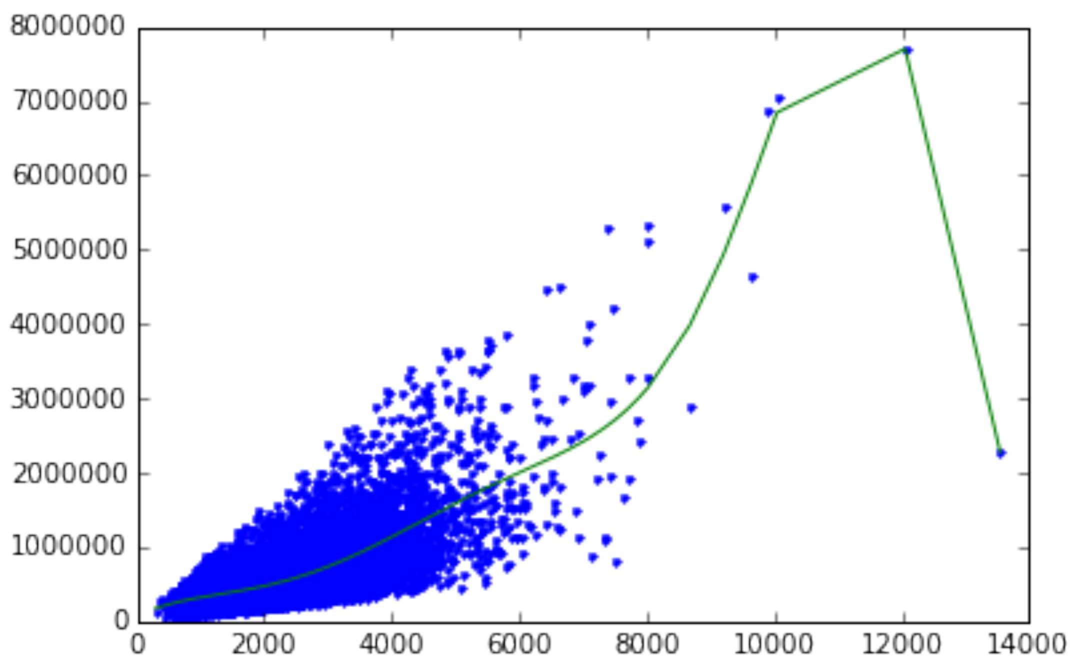
```
PROGRESS: Linear regression:
PROGRESS: --------------------------------------------------------
PROGRESS: Number of examples          : 21613
PROGRESS: Number of features          : 15
PROGRESS: Number of unpacked features : 15
PROGRESS: Number of coefficients    : 16
PROGRESS: Starting Newton Method
PROGRESS: --------------------------------------------------------
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | Iteration | Passes   | Elapsed Time | Training-max_error | 
Training-rmse |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: | 1         | 2        | 0.026002     | 2662308.584338     | 
245690.511190 |
PROGRESS: +-----------+----------+--------------+--------------------+----
-----------+
PROGRESS: SUCCESS: Optimal solution found.
PROGRESS:
```

In [27]:

```
model15.get("coefficients").print_rows(num_rows=16)
```

```
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   73619.7521135    |
|   power_1   |  None |   410.287462533    |
|   power_2   |  None |  -0.230450714427   |
|   power_3   |  None |  7.5884054245e-05  |
|   power_4   |  None | -5.65701802663e-09 |
|   power_5   |  None | -4.5702813057e-13  |
|   power_6   |  None |  2.6636020643e-17  |
|   power_7   |  None | 3.38584769284e-21  |
|   power_8   |  None | 1.14723104081e-25  |
|   power_9   |  None | -4.65293586088e-30 |
|   power_10  |  None | -8.68796202665e-34 |
|   power_11  |  None | -6.30994294704e-38 |
|   power_12  |  None | -2.70390384071e-42 |
|   power_13  |  None | -1.2124198128e-47  |
|   power_14  |  None | 1.11397452722e-50  |
|   power_15  |  None | 1.39881690857e-54  |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

In [24]:
```
plt.plot(poly15_data['power_1'],poly15_data['price'],'.',
        poly15_data['power_1'], model15.predict(poly15_data),'-')
```
Out[24]:
```
[<matplotlib.lines.Line2D at 0x23387c88>,
 <matplotlib.lines.Line2D at 0x23387e80>]
```



What do you think of the 15th degree polynomial? Do you think this is appropriate? If we were to change the data do you think you'd get pretty much the same curve? Let's take a look.

Changing the data and re-learning¶

We're going to split the sales data into four subsets of roughly equal size. Then you will estimate a 15th degree polynomial model on all four subsets of the data. Print the coefficients (you should use .print_rows(num_rows = 16) to view all of them) and plot the resulting fit (as we did above). The quiz will ask you some questions about these results.

To split the sales data into four subsets, we perform the following steps:

- First split sales into 2 subsets with `.random_split(0.5, seed=0)`.
- Next split the resulting subsets into 2 more subsets each. Use `.random_split(0.5, seed=0)`.

We set `seed=0` in these steps so that different users get consistent results. You should end up with 4 subsets (`set_1`, `set_2`, `set_3`, `set_4`) of approximately equal size.

In [28]:
```
set_1,set_2 = sales.random_split(.5,seed=0)
set_1,set_3 = set_1.random_split(.5,seed=0)
set_2,set_4 = set_2.random_split(.5,seed=0)
```

Fit a 15th degree polynomial on set_1, set_2, set_3, and set_4 using sqft_living to predict prices. Print the coefficients and make a plot of the resulting model.

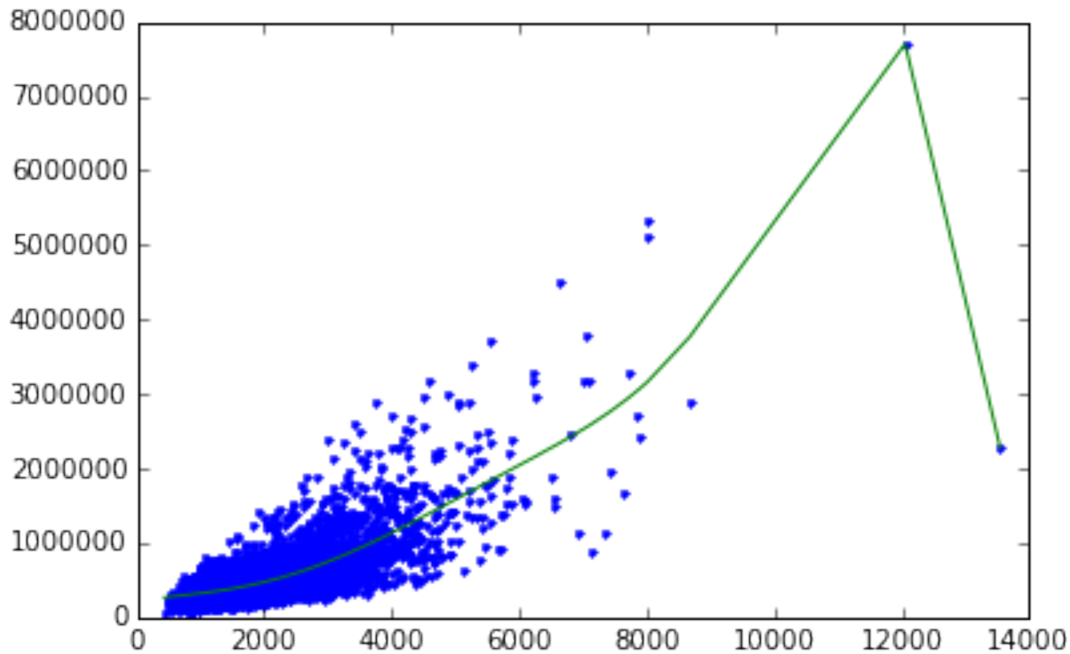
In [35]:
```
def fit15_deg_poly(data):
```

```
    poly15_data = polynomial_sframe(data['sqft_living'], 15)
    fifteen_features = poly15_data.column_names() # get the name of the
features
    poly15_data['price'] = data['price'] # add price to the data since
it's the target
    model15 = graphlab.linear_regression.create(poly15_data, target =
'price', features = fifteen_features, validation_set = None,
verbose=False)

    model15.get("coefficients").print_rows(num_rows=16)

    plt.plot(poly15_data['power_1'],poly15_data['price'],'.',
            poly15_data['power_1'], model15.predict(poly15_data),'-')
```

In [36]:

```
fit15_deg_poly(set_1)
```

```
+-------------+-------+--------------------+
|     name    | index |       value        |
+-------------+-------+--------------------+
| (intercept) |  None |   223312.750249    |
|   power_1   |  None |   118.086127587    |
|   power_2   |  None |  -0.0473482011345  |
|   power_3   |  None |  3.2531034247e-05  |
|   power_4   |  None | -3.32372152563e-09 |
|   power_5   |  None | -9.75830457749e-14 |
|   power_6   |  None | 1.15440303427e-17  |
|   power_7   |  None | 1.05145869404e-21  |
|   power_8   |  None | 3.46049616534e-26  |
|   power_9   |  None | -1.0965445417e-30  |
|   power_10  |  None | -2.42031812013e-34 |
|   power_11  |  None | -1.99601206822e-38 |
|   power_12  |  None | -1.0770990383e-42  |
|   power_13  |  None | -2.72862818005e-47 |
|   power_14  |  None | 2.44782693056e-51  |
|   power_15  |  None | 5.01975232909e-55  |
+-------------+-------+--------------------+
[16 rows x 3 columns]
```

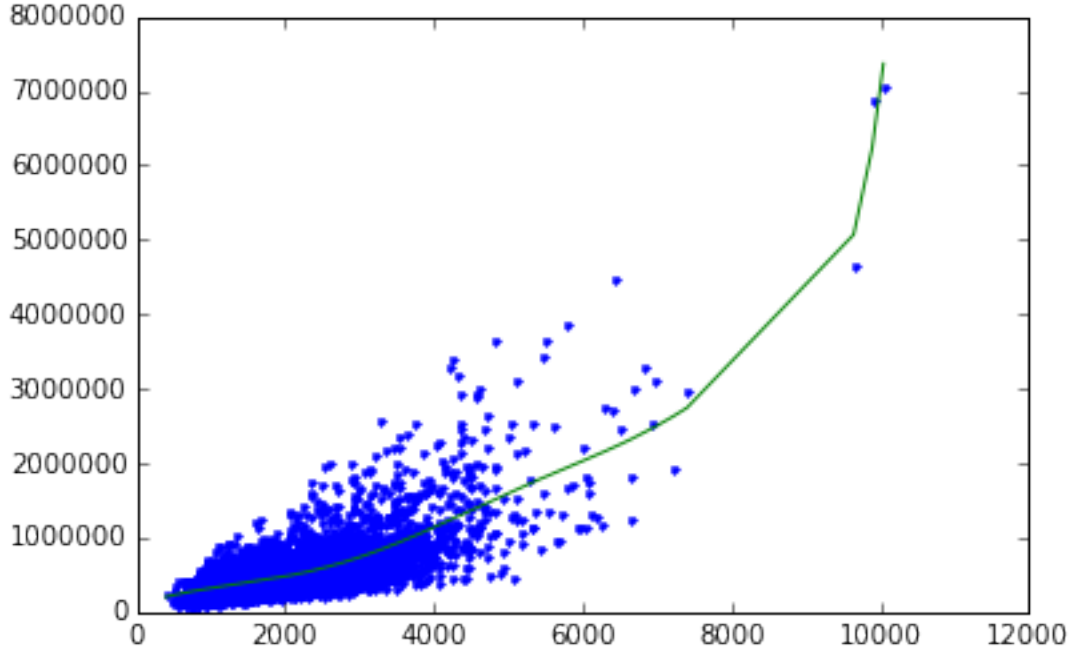


In [37]:

```
fit15_deg_poly(set_2)
```

name	index	value
(intercept)	None	87317.9795547
power_1	None	356.304911045
power_2	None	-0.164817442809
power_3	None	4.40424992697e-05
power_4	None	6.48234876179e-10
power_5	None	-6.75253226587e-13
power_6	None	-3.36842592661e-17
power_7	None	3.60999704242e-21
power_8	None	6.46999725625e-25
power_9	None	4.23639388865e-29
power_10	None	-3.62149427043e-34
power_11	None	-4.27119527274e-37
power_12	None	-5.61445971705e-41
power_13	None	-3.87452772861e-45
power_14	None	4.69430359483e-50
power_15	None	6.39045885992e-53

[16 rows x 3 columns]

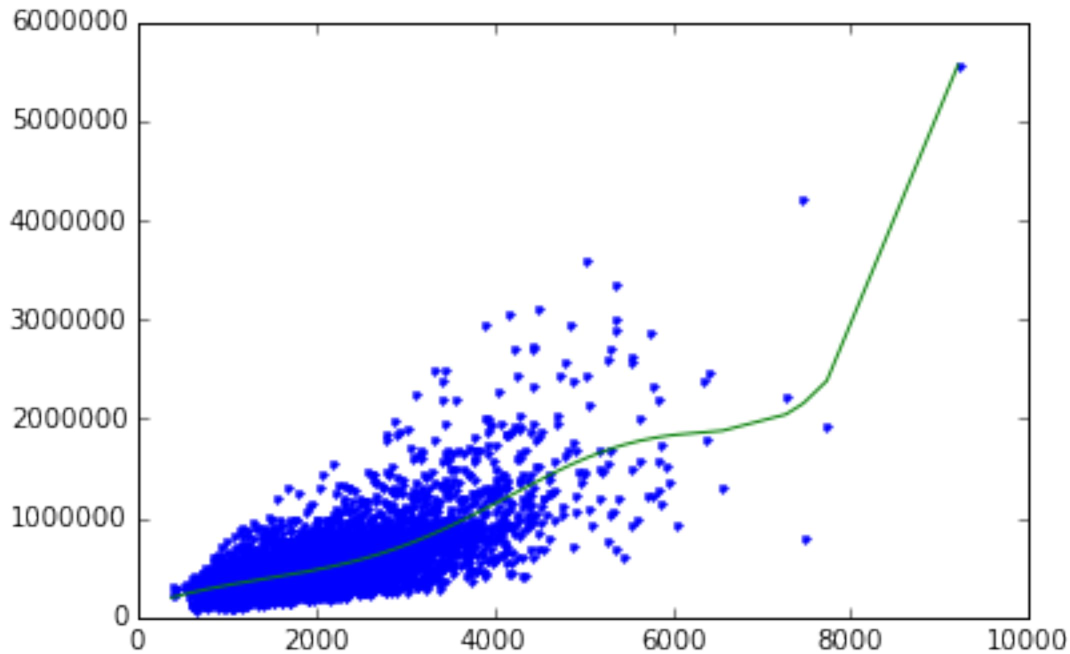


In [38]:

```
fit15_deg_poly(set_3)
```

name	index	value
(intercept)	None	89836.5077336
power_1	None	319.806946762
power_2	None	-0.103315397041
power_3	None	1.06682476068e-05
power_4	None	5.75577097709e-09
power_5	None	-2.54663464754e-13
power_6	None	-1.09641345055e-16
power_7	None	-6.36458441789e-21
power_8	None	5.52560416916e-25
power_9	None	1.35082038973e-28
power_10	None	1.18408188259e-32
power_11	None	1.98348000471e-37
power_12	None	-9.92533590368e-41
power_13	None	-1.60834847057e-44
power_14	None	-9.12006024271e-49
power_15	None	1.68636658332e-52

[16 rows x 3 columns]

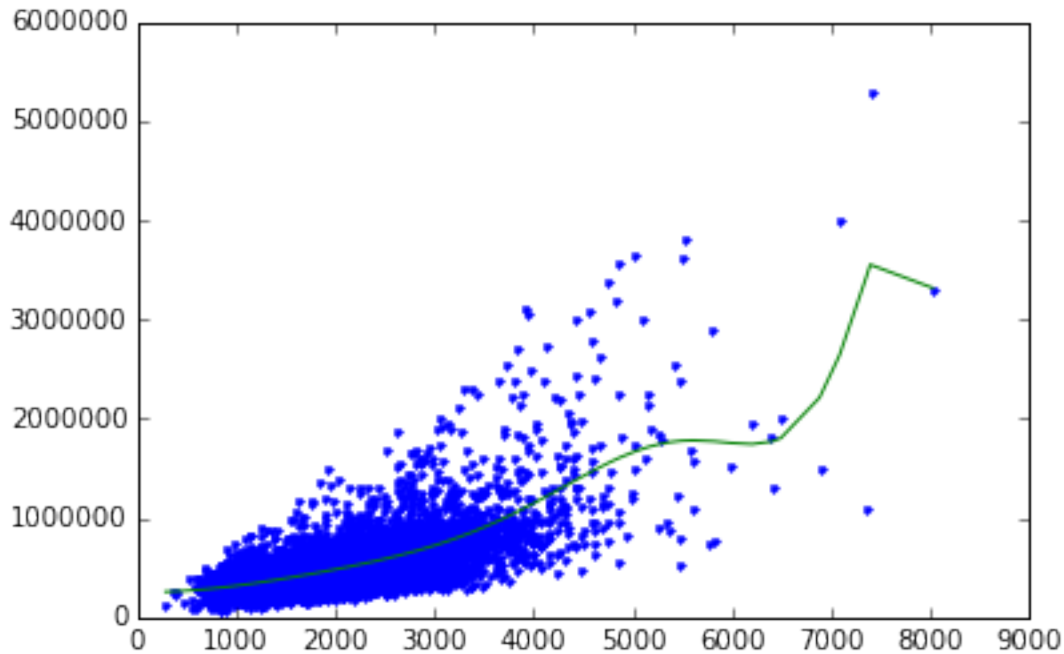


In [39]:

```
fit15_deg_poly(set_4)
```

name	index	value
(intercept)	None	259020.879455
power_1	None	-31.7277162089
power_2	None	0.10970276962
power_3	None	-1.58383847342e-05
power_4	None	-4.4766062378e-09
power_5	None	1.13976573483e-12
power_6	None	1.97669120543e-16
power_7	None	-6.15783678625e-21
power_8	None	-4.88012304078e-24
power_9	None	-6.62186781367e-28
power_10	None	-2.70631583096e-32
power_11	None	6.7237041138e-36
power_12	None	1.74115646277e-39
power_13	None	2.09188375718e-43
power_14	None	4.78015566127e-48
power_15	None	-4.74535333103e-51

[16 rows x 3 columns]



Some questions you will be asked on your quiz:
Quiz Question: Is the sign (positive or negative) for power_15 the same in all four models?
Quiz Question: (True/False) the plotted fitted lines look the same in all four plots

Selecting a Polynomial Degree

Whenever we have a "magic" parameter like the degree of the polynomial there is one well-known way to select these parameters: validation set. (We will explore another approach in week 4).
We split the sales dataset 3-way into training set, test set, and validation set as follows:

- Split our sales data into 2 sets: `training_and_validation` and `testing`. Use `random_split(0.9, seed=1)`.
- Further split our training data into two sets: `training` and `validation`. Use `random_split(0.5, seed=1)`.

Again, we set `seed=1` to obtain consistent results for different users.

In [40]:
```
training_and_validation,testing = sales.random_split(.9,seed=1)
training,validation = training_and_validation.random_split(.5,seed=1)
```

Next you should write a loop that does the following:

- For degree in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] (to get this in python type range(1, 15+1))
  - Build an SFrame of polynomial data of train_data['sqft_living'] at the current degree
  - hint: my_features = poly_data.column_names() gives you a list e.g. ['power_1', 'power_2', 'power_3'] which you might find useful for graphlab.linear_regression.create(features = my_features)
  - Add train_data['price'] to the polynomial SFrame
  - Learn a polynomial regression model to sqft vs price with that degree on TRAIN data
  - Compute the RSS on VALIDATION data (here you will want to use .predict()) for that degree and you will need to make a polynmial SFrame using validation data.
- Report which degree had the lowest RSS on validation data (remember python indexes from 0)

(Note you can turn off the print out of linear_regression.create() with verbose = False)

In [41]:
```
for degree in range(1, 16):
    train_data = polynomial_sframe(training['sqft_living'], degree)
    train_features = train_data.column_names() # get the name of the features
    train_data['price'] = training['price'] # add price to the data since it's the target
    model = graphlab.linear_regression.create(train_data, target = 'price', features = train_features, validation_set = None, verbose=False)

    validation_data = polynomial_sframe(validation['sqft_living'], degree)
    validation_data['price'] = validation['price'] # add price to the data since it's the target

    # First get the predictions
    predictions = model.predict(validation_data)
    # then compute the residuals (since we are squaring it doesn't matter which order you subtract)
    residuals = validation_data['price'] - predictions
    # square the residuals and add them up
    residuals_squared = residuals * residuals
    RSS = residuals_squared.sum()
    print("Degree: %s, RSS: $%.6f" % (degree, RSS))
```

```
Degree: 1, RSS: $676709775198048.250000
Degree: 2, RSS: $607090530698013.500000
Degree: 3, RSS: $616714574532759.375000
Degree: 4, RSS: $609129230654382.625000
Degree: 5, RSS: $599177138583682.000000
Degree: 6, RSS: $589182477809203.625000
Degree: 7, RSS: $591717038417878.250000
Degree: 8, RSS: $601558237776796.125000
Degree: 9, RSS: $612563853988437.000000
Degree: 10, RSS: $621744288936065.000000
Degree: 11, RSS: $627012012703947.625000
Degree: 12, RSS: $627757914772014.250000
Degree: 13, RSS: $624738503262080.375000
Degree: 14, RSS: $619369705904740.500000
Degree: 15, RSS: $613089202413658.875000
```

Quiz Question: Which degree (1, 2, …, 15) had the lowest RSS on Validation data?

Now that you have chosen the degree of your polynomial using validation data, compute the RSS of this model on TEST data. Report the RSS on your quiz.

In [42]:
```
train_data = polynomial_sframe(training['sqft_living'], 6)
train_features = train_data.column_names() # get the name of the features
train_data['price'] = training['price'] # add price to the data since it's the target
model = graphlab.linear_regression.create(train_data, target = 'price', features = train_features, validation_set = None, verbose=False)
```

```
test_data = polynomial_sframe(testing['sqft_living'], 6)
test_data['price'] = testing['price'] # add price to the data since it's
the target

# First get the predictions
predictions = model.predict(test_data)
# then compute the residuals (since we are squaring it doesn't matter
which order you subtract)
residuals = test_data['price'] - predictions
# square the residuals and add them up
residuals_squared = residuals * residuals
RSS = residuals_squared.sum()
print("Degree: %s, Test Data RSS: $%.6f" % (degree, RSS))
```

```
Degree: 15, Test Data RSS: $125529337847968.734375
```

Quiz Question: what is the RSS on TEST data for the model with the degree selected from Validation data? (Make sure you got the correct degree from the previous question)