**Regression Week 1: Simple Linear Regression**

In this notebook we will use data on house sales in King County to predict house prices using simple (one input) linear regression. You will:

* Use graphlab SArray and SFrame functions to compute important summary statistics
* Write a function to compute the Simple Linear Regression weights using the closed form solution
* Write a function to make predictions of the output given the input feature
* Turn the regression around to predict the input given the output
* Compare two different models for predicting house prices

In this notebook you will be provided with some already complete code as well as some code that you should complete yourself in order to answer quiz questions. The code we provide to complete is optional and is there to assist you with solving the problems but feel free to ignore the helper code and write your own.

**Fire up graphlab create**

In [30]:

*#import graphlab*

In [31]:

**import** **pandas** **as** **pd**

**from** **pandas** **import** DataFrame

**import** **numpy** **as** **np**

*# visualization*

**import** **seaborn** **as** **sns**

**import** **matplotlib.pyplot** **as** **plt**

*# this allows plots to appear directly in the notebook*

%**matplotlib** inline

sns.set(style='whitegrid', context='notebook')

**Load house sales data**

Dataset is from house sales in King County, the region where the city of Seattle, WA is located.

In [32]:

*#sales = graphlab.SFrame('kc\_house\_data.gl/')*

*# read the housing data*

sales = pd.read\_csv('kc\_house\_data.csv')

sales.head(10)

Out[32]:

id date price bedrooms bathrooms sqft\_living sqft\_lot floors waterfront view ... grade sqft\_above sqft\_basement yr\_built yr\_renovated zipcode lat long sqft\_living15 sqft\_lot15

0 7129300520 20141013T000000 221900 3 1.00 1180 5650 1 0 0 ... 7 1180 0 1955 0 98178 47.5112 -122.257 1340 5650

1 6414100192 20141209T000000 538000 3 2.25 2570 7242 2 0 0 ... 7 2170 400 1951 1991 98125 47.7210 -122.319 1690 7639

2 5631500400 20150225T000000 180000 2 1.00 770 10000 1 0 0 ... 6 770 0 1933 0 98028 47.7379 -122.233 2720 8062

3 2487200875 20141209T000000 604000 4 3.00 1960 5000 1 0 0 ... 7 1050 910 1965 0 98136 47.5208 -122.393 1360 5000

4 1954400510 20150218T000000 510000 3 2.00 1680 8080 1 0 0 ... 8 1680 0 1987 0 98074 47.6168 -122.045 1800 7503

5 7237550310 20140512T000000 1225000 4 4.50 5420 101930 1 0 0 ... 11 3890 1530 2001 0 98053 47.6561 -122.005 4760 101930

6 1321400060 20140627T000000 257500 3 2.25 1715 6819 2 0 0 ... 7 1715 0 1995 0 98003 47.3097 -122.327 2238 6819

7 2008000270 20150115T000000 291850 3 1.50 1060 9711 1 0 0 ... 7 1060 0 1963 0 98198 47.4095 -122.315 1650 9711

8 2414600126 20150415T000000 229500 3 1.00 1780 7470 1 0 0 ... 7 1050 730 1960 0 98146 47.5123 -122.337 1780 8113

9 3793500160 20150312T000000 323000 3 2.50 1890 6560 2 0 0 ... 7 1890 0 2003 0 98038 47.3684 -122.031 2390 7570

10 rows × 21 columns

**Split data into training and testing**

We use seed=0 so that everyone running this notebook gets the same results. In practice, you may set a random seed (or let GraphLab Create pick a random seed for you).

In [33]:

*#train\_data,test\_data = sales.random\_split(.8,seed=0)*

train\_data = pd.read\_csv('kc\_house\_train\_data.csv')

test\_data = pd.read\_csv('kc\_house\_test\_data.csv')

**Useful SFrame summary functions**

In order to make use of the closed form soltion as well as take advantage of graphlab's built in functions we will review some important ones. In particular:

* Computing the sum of an SArray
* Computing the arithmetic average (mean) of an SArray
* multiplying SArrays by constants
* multiplying SArrays by other SArrays

In [34]:

*# Let's compute the mean of the House Prices in King County in 2 different ways.*

prices = sales['price'] *# extract the price column of the sales DataFrame -- this is now an Series*

*# recall that the arithmetic average (the mean) is the sum of the prices divided by the total number of houses:*

sum\_prices = prices.sum()

num\_houses = prices.size *# when prices is an Series .size returns its length*

avg\_price\_1 = sum\_prices/num\_houses

avg\_price\_2 = prices.mean() *# if you just want the average, the .mean() function*

**print**("average price via method 1: ", avg\_price\_1)

**print**("average price via method 2: ", avg\_price\_2)

('average price via method 1: ', 540088.1417665294)

('average price via method 2: ', 540088.1417665294)

As we see we get the same answer both ways

In [35]:

*# if we want to multiply every price by 0.5 it's a simple as:*

half\_prices = 0.5\*prices

*# Let's compute the sum of squares of price. We can multiply two Series of the same length elementwise also with \**

prices\_squared = prices\*prices

sum\_prices\_squared = prices\_squared.sum() *# price\_squared is a Series of the squares and we want to add them up.*

**print**("the sum of price squared is: ", sum\_prices\_squared)

('the sum of price squared is: ', 9217325138472052.0)

Aside: The python notation x.xxe+yy means x.xx \* 10^(yy). e.g 100 = 10^2 = 1\*10^2 = 1e2

**Build a generic simple linear regression function**

Armed with these SArray functions we can use the closed form solution found from lecture to compute the slope and intercept for a simple linear regression on observations stored as SArrays: input\_feature, output.

Complete the following function (or write your own) to compute the simple linear regression slope and intercept:

In [36]:

**def** simple\_linear\_regression(input\_feature, output):

N = output.size

*# compute the mean of input\_feature and output*

*# compute the product of the output and the input\_feature and its mean*

input\_output\_prod = input\_feature \* output

*# compute the squared value of the input\_feature and its mean*

input\_squared = input\_feature \* input\_feature

*# use the formula for the slope*

slope = (input\_output\_prod.sum() - (input\_feature.sum() \* output.sum())/N) / (input\_squared.sum() - (input\_feature.sum() \* input\_feature.sum())/N)

*# use the formula for the intercept*

intercept = output.mean() - slope \* input\_feature.mean()

**return** (intercept, slope)

We can test that our function works by passing it something where we know the answer. In particular we can generate a feature and then put the output exactly on a line: output = 1 + 1\*input\_feature then we know both our slope and intercept should be 1

In [37]:

test\_feature = np.array(range(2))

test\_output = np.array(1 + 1\*test\_feature)

(test\_intercept, test\_slope) = simple\_linear\_regression(test\_feature, test\_output)

**print**("Intercept: ", test\_intercept)

**print**("Slope: ", test\_slope)

('Intercept: ', 1.0)

('Slope: ', 1)

In [38]:

test\_feature

Out[38]:

array([0, 1])

In [39]:

test\_output

Out[39]:

array([1, 2])

Now that we know it works let's build a regression model for predicting price based on sqft\_living. Rembember that we train on train\_data!

In [40]:

sqft\_intercept, sqft\_slope = simple\_linear\_regression(train\_data['sqft\_living'], train\_data['price'])

**print**("Intercept: ", sqft\_intercept)

**print**("Slope: ", sqft\_slope)

('Intercept: ', -47116.07906738971)

('Slope: ', 281.95883962769625)

**Predicting Values**

Now that we have the model parameters: intercept & slope we can make predictions. Using SArrays it's easy to multiply an SArray by a constant and add a constant value. Complete the following function to return the predicted output given the input\_feature, slope and intercept:

In [41]:

**def** get\_regression\_predictions(input\_feature, intercept, slope):

*# calculate the predicted values:*

predicted\_values = intercept + slope \* input\_feature

**return** predicted\_values

Now that we can calculate a prediction given the slope and intercept let's make a prediction. Use (or alter) the following to find out the estimated price for a house with 2650 squarefeet according to the squarefeet model we estiamted above.

**Quiz Question: Using your Slope and Intercept from (4), What is the predicted price for a house with 2650 sqft?**

In [42]:

my\_house\_sqft = 2650

estimated\_price = get\_regression\_predictions(my\_house\_sqft, sqft\_intercept, sqft\_slope)

**print**("The estimated price for a house with **%d** squarefeet is $**%.2f**" % (my\_house\_sqft, estimated\_price))

The estimated price for a house with 2650 squarefeet is $700074.85

**Residual Sum of Squares**

Now that we have a model and can make predictions let's evaluate our model using Residual Sum of Squares (RSS). Recall that RSS is the sum of the squares of the residuals and the residuals is just a fancy word for the difference between the predicted output and the true output.

Complete the following (or write your own) function to compute the RSS of a simple linear regression model given the input\_feature, output, intercept and slope:

In [43]:

**def** get\_residual\_sum\_of\_squares(input\_feature, output, intercept, slope):

*# First get the predictions*

predicted\_values = intercept + slope \* input\_feature

*# then compute the residuals (since we are squaring it doesn't matter which order you subtract)*

residuals = output - predicted\_values

*# square the residuals and add them up*

residuals\_squared = residuals \* residuals

RSS = residuals\_squared.sum()

**return**(RSS)

Let's test our get\_residual\_sum\_of\_squares function by applying it to the test model where the data lie exactly on a line. Since they lie exactly on a line the residual sum of squares should be zero!

In [44]:

test\_rss = get\_residual\_sum\_of\_squares(test\_feature, test\_output, test\_intercept, test\_slope) *# should be 0.0*

test\_rss

*#print(get\_residual\_sum\_of\_squares(test\_feature, test\_output, test\_intercept, test\_slope)) # should be 0.0*

Out[44]:

0.0

Now use your function to calculate the RSS on training data from the squarefeet model calculated above.

**Quiz Question: According to this function and the slope and intercept from the squarefeet model What is the RSS for the simple linear regression using squarefeet to predict prices on TRAINING data?**

In [45]:

rss\_prices\_on\_sqft = get\_residual\_sum\_of\_squares(train\_data['sqft\_living'], train\_data['price'], sqft\_intercept, sqft\_slope)

**print**("The RSS of predicting Prices based on Square Feet is : $**%.6f**" % (rss\_prices\_on\_sqft))

*#print 'The RSS of predicting Prices based on Square Feet is : ' + str(rss\_prices\_on\_sqft)*

The RSS of predicting Prices based on Square Feet is : $1201918354177287.000000

**Predict the squarefeet given price**

What if we want to predict the squarefoot given the price? Since we have an equation y = a + b\*x we can solve the function for x. So that if we have the intercept (a) and the slope (b) and the price (y) we can solve for the estimated squarefeet (x).

Comlplete the following function to compute the inverse regression estimate, i.e. predict the input\_feature given the output!

In [46]:

**def** inverse\_regression\_predictions(output, intercept, slope):

*# solve output = intercept + slope\*input\_feature for input\_feature. Use this equation to compute the inverse predictions:*

estimated\_feature = (output - intercept) / slope

**return** estimated\_feature

Now that we have a function to compute the squarefeet given the price from our simple regression model let's see how big we might expect a house that coses $800,000 to be.

**Quiz Question: According to this function and the regression slope and intercept from (3) what is the estimated square-feet for a house costing $800,000?**

In [47]:

my\_house\_price = 800000

estimated\_squarefeet = inverse\_regression\_predictions(my\_house\_price, sqft\_intercept, sqft\_slope)

**print**("The estimated squarefeet for a house worth $**%.2f** is **%d**" % (my\_house\_price, estimated\_squarefeet))

The estimated squarefeet for a house worth $800000.00 is 3004

**New Model: estimate prices from bedrooms**

We have made one model for predicting house prices using squarefeet, but there are many other features in the sales SFrame. Use your simple linear regression function to estimate the regression parameters from predicting Prices based on number of bedrooms. Use the training data!

In [48]:

*# Estimate the slope and intercept for predicting 'price' based on 'bedrooms'*

bdrm\_intercept, bdrm\_slope = simple\_linear\_regression(train\_data['bedrooms'], train\_data['price'])

**print**("Intercept: ", bdrm\_intercept)

**print**("Slope: ", bdrm\_slope)

('Intercept: ', 109493.56662402855)

('Slope: ', 127582.90164082233)

**Test your Linear Regression Algorithm**

Now we have two models for predicting the price of a house. How do we know which one is better? Calculate the RSS on the TEST data (remember this data wasn't involved in learning the model). Compute the RSS from predicting prices using bedrooms and from predicting prices using squarefeet.

**Quiz Question: Which model (square feet or bedrooms) has lowest RSS on TEST data? Think about why this might be the case.**

In [49]:

*# Compute RSS when using bedrooms on TEST data:*

bdrm\_test\_rss = get\_residual\_sum\_of\_squares(test\_data['bedrooms'], test\_data['price'], bdrm\_intercept, bdrm\_slope)

bdrm\_test\_rss

Out[49]:

493363257575886.8

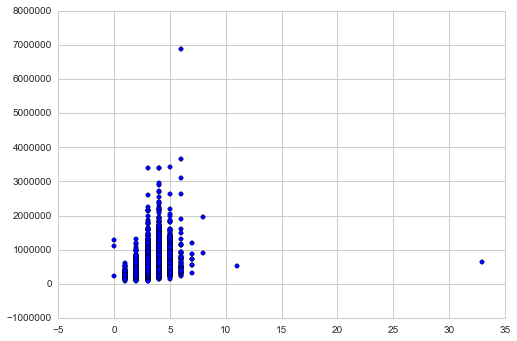
In [50]:

plt.figure()

plt.scatter(x=test\_data[['bedrooms']], y=test\_data['price'], marker='o', c='b')

Out[50]:

<matplotlib.collections.PathCollection at 0x20deb8d0>



In [51]:

*# Compute RSS when using squarfeet on TEST data:*

sqft\_test\_rss = get\_residual\_sum\_of\_squares(test\_data['sqft\_living'], test\_data['price'], sqft\_intercept, sqft\_slope)

sqft\_test\_rss

Out[51]:

275402933617682.8

In [52]:

plt.figure()

plt.scatter(x=test\_data[['sqft\_living']], y=test\_data['price'], marker='o', c='b')

Out[52]:

<matplotlib.collections.PathCollection at 0x20efb630>

