Get the Boston Data

This part is basically taken directly from the bigdataexaminer (http://bigdataexaminer.com/uncategorized/how-to-run-linear-regression-in-python-scikit-learn/) tutorial. All the Imports first

```
In [94]: """Example of DNNRegressor for Housing dataset."""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from sklearn import cross_validation
from sklearn import metrics
from sklearn import preprocessing
import tensorflow as tf
from tensorflow.contrib import import learn
import pandas as pd
```

Get the data...

```
In [95]: from sklearn.datasets import load_boston
boston = load_boston()
print( "type of boston = ", type(boston))
type of boston = <class 'sklearn.datasets.base.Bunch'>
```

```
In [96]: boston.keys()
Out[96]: [u'data', u'feature_names', u'DESCRIPTORS', u'target']
```

```
In [97]: boston.data.shape
Out[97]: (506, 13)
```
In [98]: print(boston.feature_names)

['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
'B' 'LSTAT']
In [99]: `print( boston.DESCR )`
Boston House Prices dataset

Notes
-----

Data Set Characteristics:

: Number of Instances: 506
: Number of Attributes: 13 numeric/categorical predictive
: Median Value (attribute 14) is usually the target

: Attribute Information (in order):
- CRIM      per capita crime rate by town
- ZN        proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS     proportion of non-retail business acres per town
- CHAS      Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX       nitric oxides concentration (parts per 10 million)
- RM        average number of rooms per dwelling
- AGE       proportion of owner-occupied units built prior to 1940
- DIS       weighted distances to five Boston employment centres
- RAD       index of accessibility to radial highways
- TAX       full-value property-tax rate per $10,000
- PTRATIO   pupil-teacher ratio by town
- B         1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT     % lower status of the population
- MEDV      Median value of owner-occupied homes in $1000's

: Missing Attribute Values: None

: Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.
The Boston house-price data has been used in many machine learning papers that address regression problems.

**References**
In [100]:
print( "target = ",
', '.join( str(k) for k in boston.target[0:5] ),
'...',
', '.join( str(k) for k in boston.target[-5:] ) )

target = 24.0, 21.6, 34.7, 33.4, 36.2 ... 22.4, 20.6, 23.9, 22.0, 11.9

Convert the boston data into a panda data-frame

In [101]:
bostonDF = pd.DataFrame( boston.data )
bostonDF.head()

Out[101]:
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00632</td>
<td>18.0</td>
<td>2.31</td>
<td>0.538</td>
<td>6.575</td>
<td>65.2</td>
<td>4.0900</td>
<td>1 296</td>
<td>15.3</td>
<td>396.90</td>
<td>4.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.02731</td>
<td>0.07</td>
<td>0.469</td>
<td>6.421</td>
<td>78.9</td>
<td>4.9671</td>
<td>2 242</td>
<td>17.8</td>
<td>396.90</td>
<td>9.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.02729</td>
<td>0.07</td>
<td>0.469</td>
<td>7.185</td>
<td>61.1</td>
<td>4.9671</td>
<td>2 242</td>
<td>17.8</td>
<td>392.83</td>
<td>4.03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.03237</td>
<td>0.18</td>
<td>0.458</td>
<td>6.998</td>
<td>45.8</td>
<td>6.0622</td>
<td>3 222</td>
<td>18.7</td>
<td>394.63</td>
<td>2.94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.06905</td>
<td>0.18</td>
<td>0.458</td>
<td>7.147</td>
<td>54.2</td>
<td>6.0622</td>
<td>3 222</td>
<td>18.7</td>
<td>396.90</td>
<td>5.33</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add column names

In [102]:
bostonDF.columns = boston.feature_names
bostonDF.head()

Out[102]:
<table>
<thead>
<tr>
<th>CRIM</th>
<th>ZN</th>
<th>INDUS</th>
<th>CHAS</th>
<th>NOX</th>
<th>RM</th>
<th>AGE</th>
<th>DIS</th>
<th>RAD</th>
<th>TAX</th>
<th>PTRATIO</th>
<th>B</th>
<th>LSTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00632</td>
<td>18</td>
<td>2.31</td>
<td>0.538</td>
<td>6.575</td>
<td>65.2</td>
<td>4.0900</td>
<td>1 296</td>
<td>15.3</td>
<td>396.90</td>
<td>4.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02731</td>
<td>0</td>
<td>7.07</td>
<td>0.469</td>
<td>6.421</td>
<td>78.9</td>
<td>4.9671</td>
<td>2 242</td>
<td>17.8</td>
<td>396.90</td>
<td>9.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02729</td>
<td>0</td>
<td>7.07</td>
<td>0.469</td>
<td>7.185</td>
<td>61.1</td>
<td>4.9671</td>
<td>2 242</td>
<td>17.8</td>
<td>392.83</td>
<td>4.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.03237</td>
<td>0</td>
<td>2.18</td>
<td>0.458</td>
<td>6.998</td>
<td>45.8</td>
<td>6.0622</td>
<td>3 222</td>
<td>18.7</td>
<td>394.63</td>
<td>2.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.06905</td>
<td>0</td>
<td>2.18</td>
<td>0.458</td>
<td>7.147</td>
<td>54.2</td>
<td>6.0622</td>
<td>3 222</td>
<td>18.7</td>
<td>396.90</td>
<td>5.33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adding the target to the data frame...
So now we have a pandas data frame holding the data.

**Predicting Housing Prices with Linear Regression**

```python
In [103]:
   bostonDF['PRICE'] = boston.target
   bostonDF.head()
```

<table>
<thead>
<tr>
<th>CRIM</th>
<th>ZN</th>
<th>INDUS</th>
<th>CHAS</th>
<th>NOX</th>
<th>RM</th>
<th>AGE</th>
<th>DIS</th>
<th>RAD</th>
<th>TAX</th>
<th>PTRATIO</th>
<th>B</th>
<th>LSTAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00632</td>
<td>18</td>
<td>2.31</td>
<td>0</td>
<td>0.538</td>
<td>6.575</td>
<td>65.2</td>
<td>4.0900</td>
<td>1</td>
<td>296</td>
<td>15.3</td>
<td>396.90</td>
<td>4.98</td>
</tr>
<tr>
<td>0.02731</td>
<td>0</td>
<td>7.07</td>
<td>0</td>
<td>0.469</td>
<td>6.421</td>
<td>78.9</td>
<td>4.9671</td>
<td>2</td>
<td>242</td>
<td>17.8</td>
<td>396.90</td>
<td>9.14</td>
</tr>
<tr>
<td>0.02729</td>
<td>0</td>
<td>7.07</td>
<td>0</td>
<td>0.469</td>
<td>7.185</td>
<td>61.1</td>
<td>4.9671</td>
<td>2</td>
<td>242</td>
<td>17.8</td>
<td>392.83</td>
<td>4.03</td>
</tr>
<tr>
<td>0.03237</td>
<td>0</td>
<td>2.18</td>
<td>0</td>
<td>0.458</td>
<td>6.998</td>
<td>45.8</td>
<td>6.0622</td>
<td>3</td>
<td>222</td>
<td>18.7</td>
<td>394.63</td>
<td>2.94</td>
</tr>
<tr>
<td>0.06905</td>
<td>0</td>
<td>2.18</td>
<td>0</td>
<td>0.458</td>
<td>7.147</td>
<td>54.2</td>
<td>6.0622</td>
<td>3</td>
<td>222</td>
<td>18.7</td>
<td>396.90</td>
<td>5.33</td>
</tr>
</tbody>
</table>

```python
Out[103]:
   LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

The `LinearRegression` objects supports several methods:

- `fit()`: fits a linear model
- `predict()`: predicts Y using the linear model's estimated coeff
- `score()`: returns the coef of determination R^2
- `get_params()`:  
- `mro()`:  
- `register()`:  
- `set_params()`:  

**Fitting the Model**

We are going to use all 13 parameters to fit a linear regression model
In [110]:

    lm.fit( X, y)

Out[110]:

    LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [114]:

    print( "Estimated intercept coeff: ", lm.intercept_
    
    print( "Number of coeffs: ", len(lm.coef_)
    
    print( "Coeffs = ", lm.coef_

    Estimated intercept coeff: 36.4911032804
    Number of coeffs: 13
    Coeffs = [ -1.07170557e-01  4.63952195e-02  2.08602395e-02  2.68856140e+00
                     -1.77957587e+01  3.80475246e+00  7.51061703e-04 -1.47575880e+00
                     3.05655038e-01 -1.23293463e-02 -9.53463555e-01  9.39251272e-03
                     -5.25466633e-01]

Create a dataframe with the coeffs

In [115]:

    pd.DataFrame( zip(X.columns, lm.coef_),
                     columns=['features', 'estimatedCoeffs'])

Out[115]:

<table>
<thead>
<tr>
<th>features</th>
<th>estimatedCoeffs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CRIM</td>
<td>-0.107171</td>
</tr>
<tr>
<td>1 ZN</td>
<td>0.046395</td>
</tr>
<tr>
<td>2 INDUS</td>
<td>0.020860</td>
</tr>
<tr>
<td>3 CHAS</td>
<td>2.688561</td>
</tr>
<tr>
<td>4 NOX</td>
<td>-17.795759</td>
</tr>
<tr>
<td>5 RM</td>
<td>3.804752</td>
</tr>
<tr>
<td>6 AGE</td>
<td>0.000751</td>
</tr>
<tr>
<td>7 DIS</td>
<td>-1.475759</td>
</tr>
<tr>
<td>8 RAD</td>
<td>0.305655</td>
</tr>
<tr>
<td>9 TAX</td>
<td>-0.012329</td>
</tr>
<tr>
<td>10 PTRATIO</td>
<td>-0.953464</td>
</tr>
<tr>
<td>11 B</td>
<td>0.009393</td>
</tr>
<tr>
<td>12 LSTAT</td>
<td>-0.525467</td>
</tr>
</tbody>
</table>

Generate a plot of Price versus RM (Avg # of Rooms per dwelling)
In [138]:
import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter( bostonDF.RM, bostonDF.PRICE, s=5 )
plt.xlabel( "Avg. # Rooms" )
plt.ylabel( "Housing Price (in $10,000)" )
plt.title( "Price vs. # Rooms" )

Out[138]: <matplotlib.text.Text at 0x1170f1f10>

Out[138]:

```
In [131]:
lm.predict( X)[0:10]
```

Out[131]:
```
array([ 30.00821269,  25.0298606 ,  30.5702317 ,  28.60814055,  
       27.94288232,  25.25940048,  23.00433994,  19.5347558 ,  
       11.51696539,  18.91981483])
```

Plot prediction against real values

---

**Predicting Prices**
Let's compute the mean squared error:

```python
In [139]: mse = np.mean((bostonDF.PRICE - lm.predict(X))**2)
print("Mean squared error = ", mse)
Mean squared error = 21.897792177
```

## Training and Validating

```python
In [142]: X_train, X_test, y_train, y_test = \
   cross_validation.train_test_split( X, 
   bostonDF.PRICE, 
   test_size=0.33, 
   random_state=5 )
print( X_train.shape, X_test.shape, y_train.shape, y_test.shape )
(339, 13) (167, 13) (339,) (167,)
```

Building a linear regression model using only the train data:
In [170]:
    
    lm = LinearRegression()
    lm.fit( X_train, y_train )

--
TypeError Traceback (most recent call last)
<ipython-input-170-a70c6f3ff2f8> in <module>()
     1 lm = LinearRegression()
----> 2 lm.fit( X_train, y_train, logdir='/tmp/SKLearnLinReg/ ' )

TypeError: fit() got an unexpected keyword argument 'logdir'

In [146]:
    
    pred_train = lm.predict( X_train )
    pred_test = lm.predict( X_test )
    print( "mse_train = ", np.mean( ( y_train-lm.predict(X_train))**2) )
    print( "mse_test = ", np.mean( ( y_test-lm.predict(X_test))**2) )

mse_train =  19.5467584735
mse_test =  28.5413672756

**Plotting the Residuals**

In [164]:
    
    plt.scatter( lm.predict(X_train), lm.predict(X_train) - y_train, c = 'b', s=30, alpha=0.4 )
    plt.scatter( lm.predict(X_test), lm.predict(X_test) - y_test, c = 'g', s=30 )
    plt.hlines( y=0, xmin=-5, xmax=55)
    plt.title( "Residuals" )
    plt.ylabel( "Residuals" )

Out[164]: <matplotlib.text.Text at 0x118987b90>