Regression in Python

This is a very quick run-through of some basic statistical concepts, adapted from Lab 4 in Harvard's CS109 (https://github.com/cs109/2015lab4) course. Please feel free to try the original lab if you're feeling ambitious :-) The CS109 git repository also has the solutions if you're stuck.

- Linear Regression Models
- Prediction using linear regression
- Some re-sampling methods
    - Train-Test splits
    - Cross Validation

Linear regression is used to model and predict continuous outcomes while logistic regression is used to model binary outcomes. We'll see some examples of linear regression as well as Train-test splits.

The packages we'll cover are: `statsmodels`, `seaborn`, and `scikit-learn`. While we don't explicitly teach `statsmodels` and `seaborn` in the Springboard workshop, those are great libraries to know.

 (https://imgs.xkcd.com/comics/sustainable.png)

In [111]:

```
# special IPython command to prepare the notebook for matplotlib and other lib
raries
%pylab inline

import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt

#import sklearn
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn import svm

import seaborn as sns

# special matplotlib argument for improved plots
from matplotlib import rcParams
sns.set_style("whitegrid")
sns.set_context("poster")
```

```
Populating the interactive namespace from numpy and matplotlib
```

Part 1: Linear Regression

Purpose of linear regression

Given a dataset X and Y, linear regression can be used to:

- Build a **predictive model** to predict future values of X_i without a Y value.
- Model the **strength of the relationship** between each dependent variable X_i and Y
  - Sometimes not all X_i will have a relationship with Y
  - Need to figure out which X_i contributes most information to determine Y
- Linear regression is used in so many applications that I won't warrant this with examples. It is in many cases, the first pass prediction algorithm for continuous outcomes.

A brief recap (feel free to skip if you don't care about the math)

Linear Regression (http://en.wikipedia.org/wiki/Linear_regression) is a method to model the relationship between a set of independent variables X (also knowns as explanatory variables, features, predictors) and a dependent variable Y. This method assumes the relationship between each predictor X is linearly related to the dependent variable Y.

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where ϵ is considered as an unobservable random variable that adds noise to the linear relationship. This is the simplest form of linear regression (one variable), we'll call this the simple model.

- β_0 is the intercept of the linear model
- Multiple linear regression is when you have more than one independent variable
  - $X_1, X_2, X_3, \ldots$

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon$$

- Back to the simple model. The model in linear regression is the *conditional mean* of Y given the values in X is expressed a linear function.

$$y = f(x) = E(Y|X = x)$$

conditional mean

http://www.learner.org/courses/againstallodds/about/glossary.html (http://www.learner.org/courses/againstallodds/about/glossary.html)

- The goal is to estimate the coefficients (e.g. β_0 and β_1). We represent the estimates of the coefficients with a "hat" on top of the letter.

$$\hat{\beta}_0, \hat{\beta}_1$$

- Once you estimate the coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$, you can use these to predict new values of Y

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1$$

- How do you estimate the coefficients?
  - There are many ways to fit a linear regression model
  - The method called **least squares** is one of the most common methods
  - We will discuss least squares today

Estimating $\hat{\beta}$: Least squares

Least squares (http://en.wikipedia.org/wiki/Least_squares) is a method that can estimate the coefficients of a linear model by minimizing the difference between the following:

$$S = \sum_{i=1}^{N} r_i = \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_i))^2$$

where N is the number of observations.

- We will not go into the mathematical details, but the least squares estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ minimize the sum of the squared residuals $r_i = y_i - (\beta_0 + \beta_1 x_i)$ in the model (i.e. makes the difference between the observed y_i and linear model $\beta_0 + \beta_1 x_i$ as small as possible).

The solution can be written in compact matrix notation as

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

We wanted to show you this in case you remember linear algebra, in order for this solution to exist we need $X^T X$ to be invertible. Of course this requires a few extra assumptions, X must be full rank so that $X^T X$ is invertible, etc. **This is important for us because this means that having redundant features in our regression models will lead to poorly fitting (and unstable) models.** We'll see an implementation of this in the extra linear regression example.

Note: The "hat" means it is an estimate of the coefficient.

Part 2: Boston Housing Data Set

The Boston Housing data set (https://archive.ics.uci.edu/ml/datasets/Housing) contains information about the housing values in suburbs of Boston. This dataset was originally taken from the StatLib library which is maintained at Carnegie Mellon University and is now available on the UCI Machine Learning Repository.

Load the Boston Housing data set from `sklearn`

This data set is available in the sklearn (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston) python module which is how we will access it today.

In [2]:
```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [3]:
```
boston.keys()
```

Out[3]:
```
dict_keys(['feature_names', 'data', 'target', 'DESCR'])
```

In [4]:
```
boston.data.shape
```

Out[4]:
```
(506, 13)
```

In [6]:
```
# Print column names
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [11]:
```
# Print description of Boston housing data set
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.
http://archive.ics.uci.edu/ml/datasets/Housing


This dataset was taken from the StatLib library which is maintained at Carneg
ie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers tha
t address regression
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
 Data and Sources of Collinearity', Wiley, 1980. 244-261.
```

```
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
```

Now let's explore the data set itself.

In [13]:
```
bos = pd.DataFrame(boston.data)
bos.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

There are no column names in the DataFrame. Let's add those.

In [14]:
```
# Show the first 5 records of the data set
bos.columns = boston.feature_names
bos.head()
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90

Now we have a pandas DataFrame called `bos` containing all the data we want to use to predict Boston Housing prices. Let's create a variable called `PRICE` which will contain the prices. This information is contained in the `target` data.

In [15]:
```
# Show the shape of the PRICE variable being imported from target data (506 ro
ws/instances/records)
print(boston.target.shape)
```

```
(506,)
```

In [16]:
```
# Add a price predictor column to the data frame
bos['PRICE'] = boston.target
bos.head()
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90

EDA and Summary Statistics

Let's explore this data set. First we use `describe()` to get basic summary statistics for each of the columns.

In [17]:
```
# basic summary statistics for each of the columns
bos.describe()
```

Out[17]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.5749
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.1488
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.90000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.0250
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.5000
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.0750
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.0000

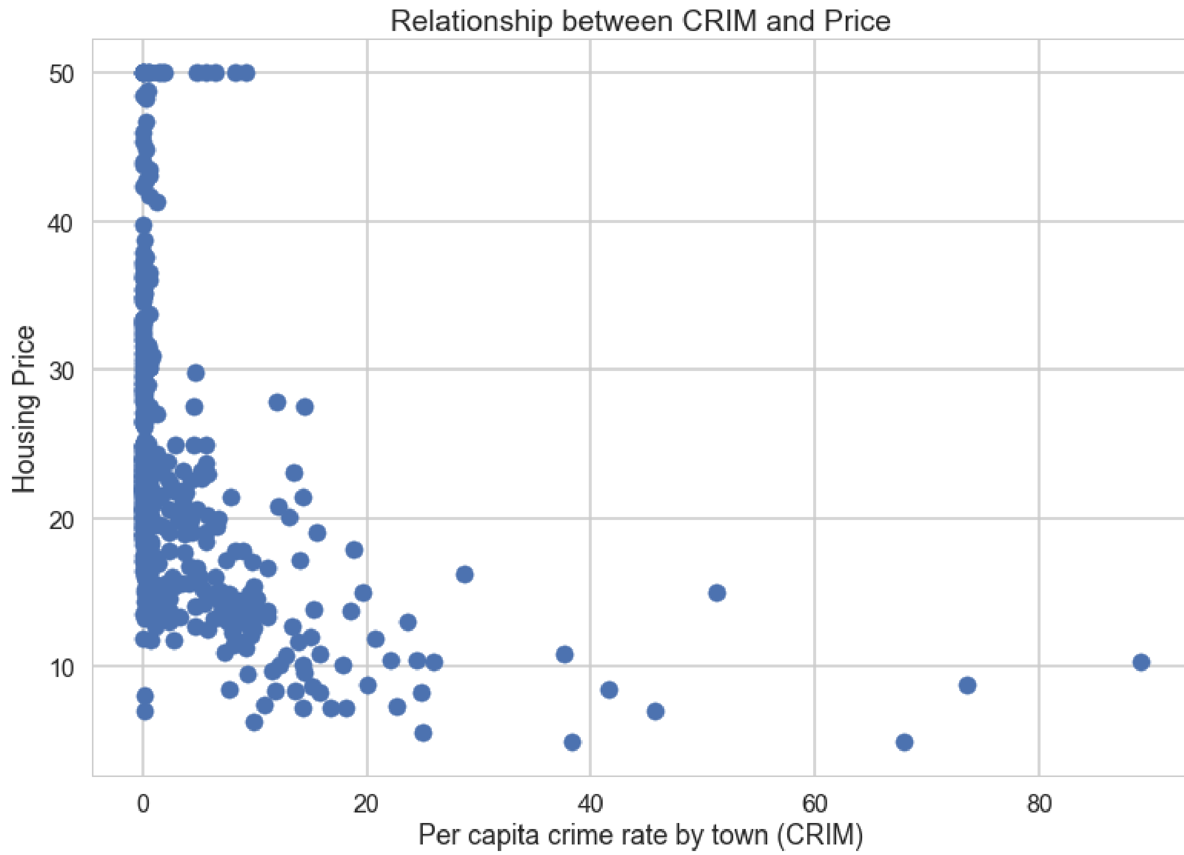
Scatter plots

Let's look at some scatter plots for three variables: 'CRIM', 'RM' and 'PTRATIO'.

What kind of relationship do you see? e.g. positive, negative? linear? non-linear?

In [23]:
```
plt.scatter(bos.CRIM, bos.PRICE)
plt.xlabel("Per capita crime rate by town (CRIM)")
plt.ylabel("Housing Price")
plt.title("Relationship between CRIM and Price")
```

Out[23]: <matplotlib.text.Text at 0x264aee712e8>

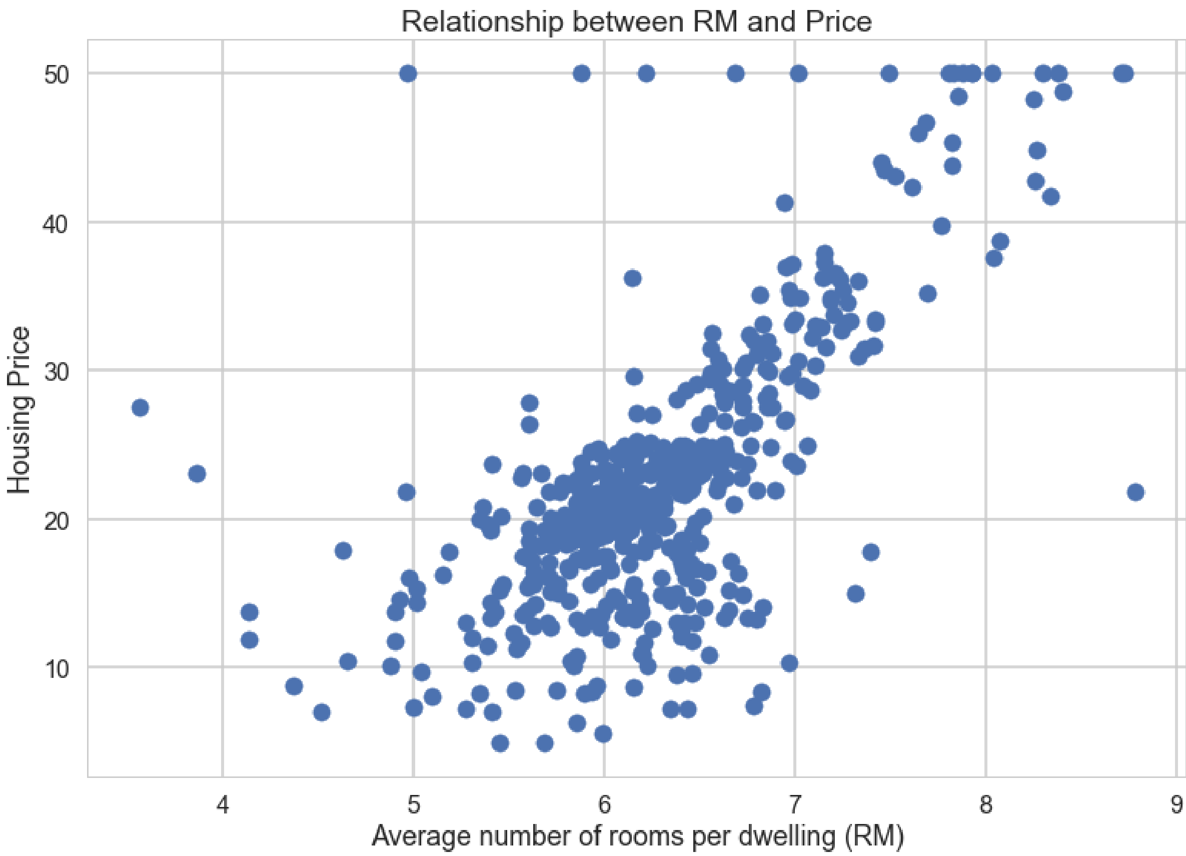


Your turn: Create scatter plots between *RM* and *PRICE*, and *PTRATIO* and *PRICE*. What do you notice?

There appears to be a non-linear relationship (NO RELATIONSHIP) between the two variables "per capita crime rate by town (CRIM)" and price (PRICE)

In [24]:
```
#your turn: scatter plot between *RM* and *PRICE*
plt.scatter(bos.RM, bos.PRICE)
plt.xlabel("Average number of rooms per dwelling (RM)")
plt.ylabel("Housing Price")
plt.title("Relationship between RM and Price")
```

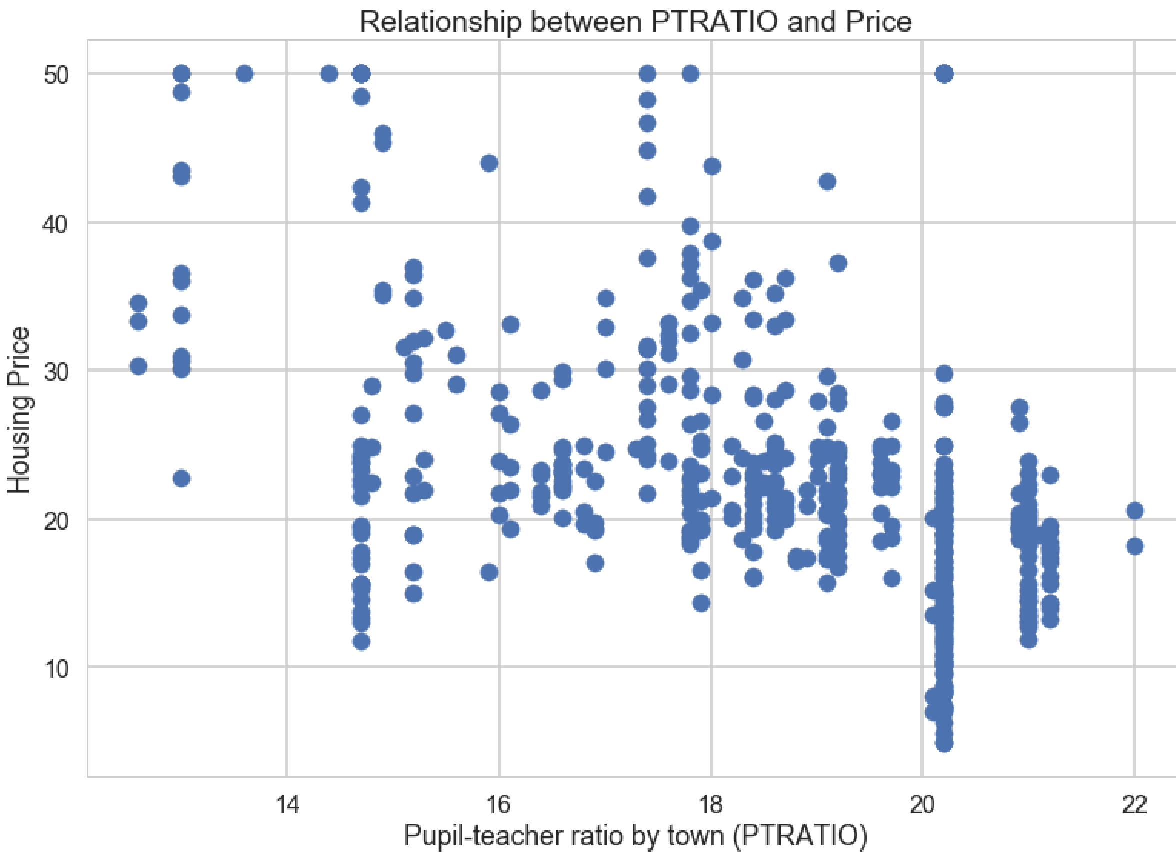
Out[24]: <matplotlib.text.Text at 0x264aeed50b8>



There appears to be a positive relationship between the two variables "average number of rooms per dwelling (RM)" and price (PRICE). As the "average number of rooms per dwelling (RM)" increases so does the "housing prices (PRICE)"

In [20]:
```
#your turn: scatter plot between *PTRATIO* and *PRICE*
plt.scatter(bos.PTRATIO, bos.PRICE)
plt.xlabel("Pupil-teacher ratio by town (PTRATIO)")
plt.ylabel("Housing Price")
plt.title("Relationship between PTRATIO and Price")
```

Out[20]: <matplotlib.text.Text at 0x264aea37ac8>



There appears to be a negative relationship between the two variables "pupil-teacher ratio by town (PTRATIO)" and price (PRICE). As the "pupil-teacher ratio by town (PTRATIO)" increases "housing prices (PRICE)" decreases

Your turn: What are some other numeric variables of interest? Plot scatter plots with these variables and *PRICE*.

```
In [26]: #your turn: create some other scatter plots
         plt.scatter(bos.TAX, bos.PRICE)
         plt.xlabel("full-value property-tax rate per $10,000 (TAX)")
         plt.ylabel("Housing Price")
         plt.title("Relationship between TAX and Price")
```

```
Out[26]: <matplotlib.text.Text at 0x264af2df3c8>
```

There appears to be a positive relationship between the two variables "full-value property-tax rate per $10,000(TAX)\text{"} \ and price(PRICE). As the \text{"} \ full-value property-tax rate per$ **10,000 (TAX)" increases so does the "housing prices (PRICE)"**

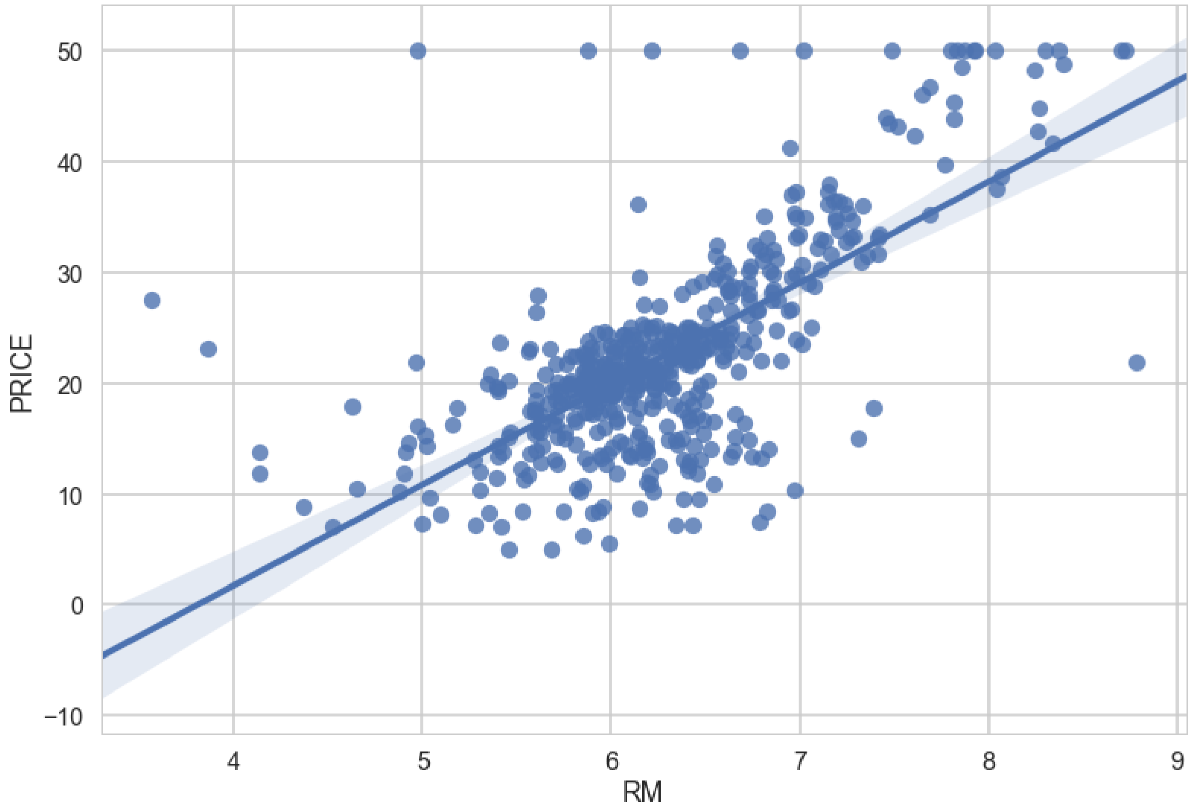
Scatter Plots using Seaborn

[Seaborn (https://stanford.edu/~mwaskom/software/seaborn/)](https://stanford.edu/~mwaskom/software/seaborn/) is a cool Python plotting library built on top of matplotlib. It provides convenient syntax and shortcuts for many common types of plots, along with better-looking defaults.

We can also use [seaborn regplot (https://stanford.edu/~mwaskom/software/seaborn/tutorial/regression.html#functions-to-draw-linear-regression-models)](https://stanford.edu/~mwaskom/software/seaborn/tutorial/regression.html#functions-to-draw-linear-regression-models) for the scatterplot above. This provides automatic linear regression fits (useful for data exploration later on). Here's one example below.

In [74]:
```
sns.regplot(y="PRICE", x="RM", data=bos, fit_reg = True)
```

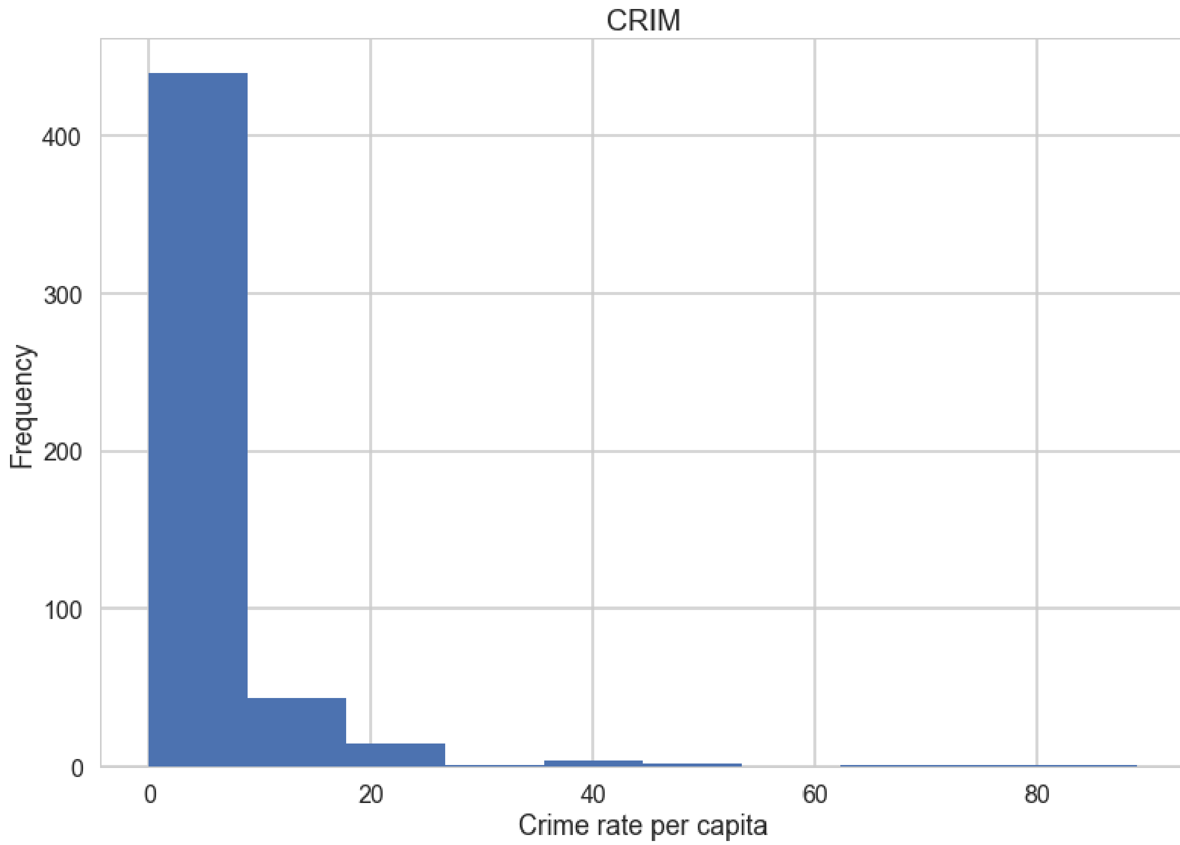
Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x264b1b71b00>



Histograms

Histograms are a useful way to visually summarize the statistical properties of numeric variables. They can give you an idea of the mean and the spread of the variables as well as outliers.

In [27]:
```
plt.hist(bos.CRIM)
plt.title("CRIM")
plt.xlabel("Crime rate per capita")
plt.ylabel("Frequency")
plt.show()
```

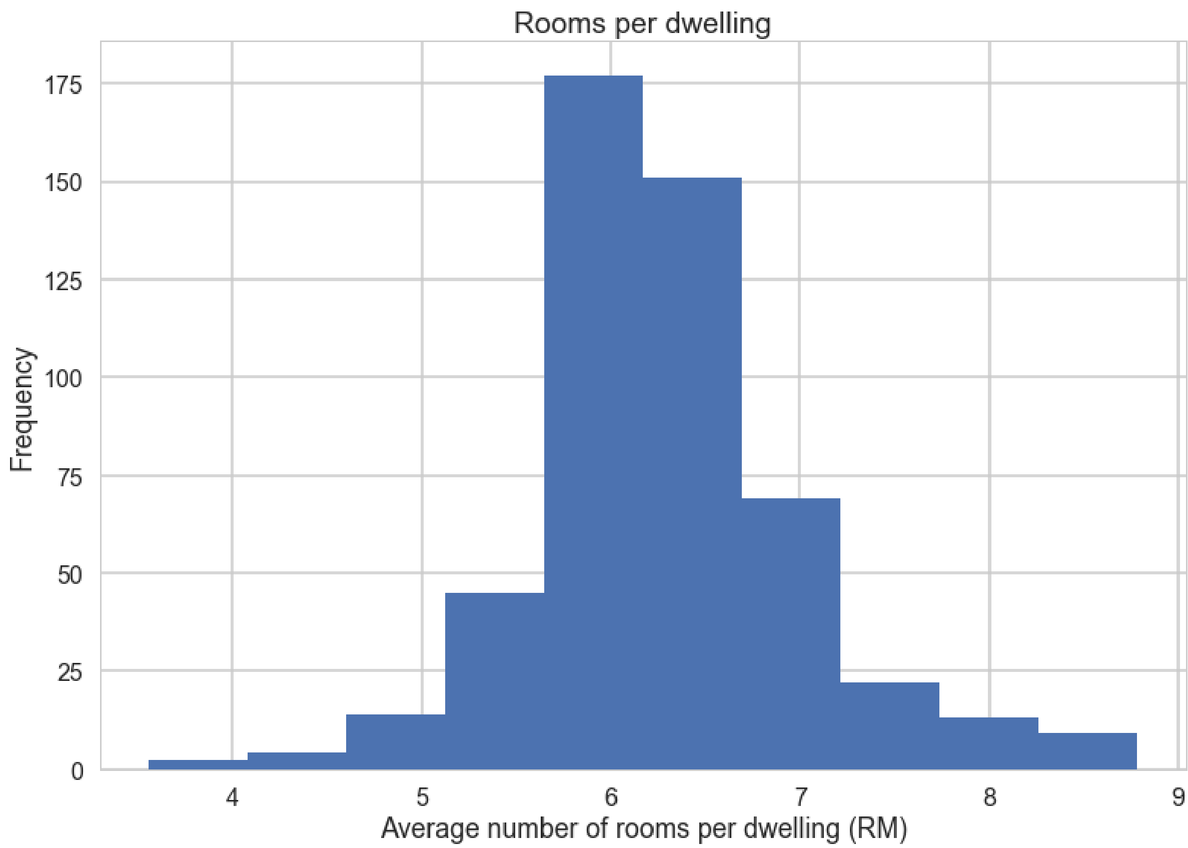


Your turn: Plot separate histograms and one for *RM*, one for *PTRATIO*. Any interesting observations?

It appears that the crime rate in Boston is higher where there is high population densities.

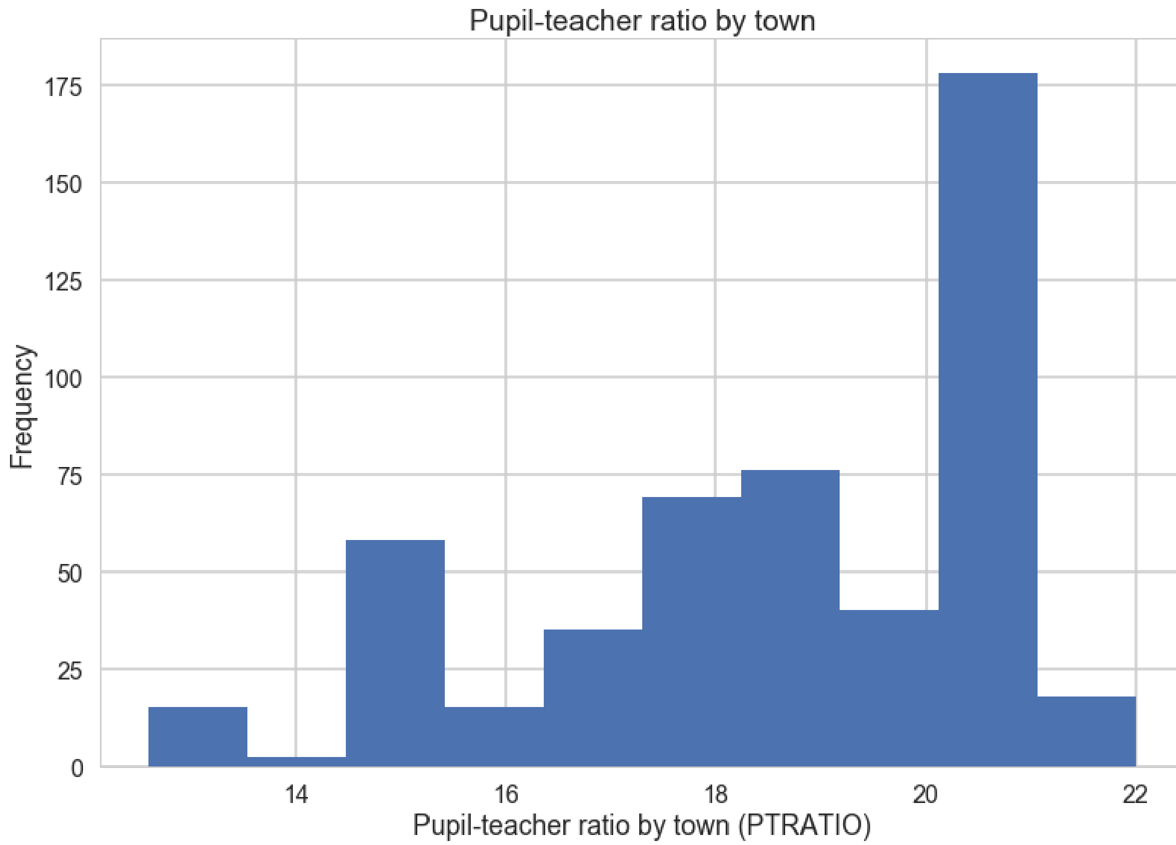
In [32]:

```
#your turn
plt.hist(bos.RM)
plt.title("Rooms per dwelling")
plt.xlabel("Average number of rooms per dwelling (RM)")
plt.ylabel("Frequency")
plt.show()
```



It appears that units with 6 and 7 rooms represent the majority of dwellings in the city of Boston

In [31]:
```
plt.hist(bos.PTRATIO)
plt.title("Pupil-teacher ratio by town")
plt.xlabel("Pupil-teacher ratio by town (PTRATIO)")
plt.ylabel("Frequency")
plt.show()
```



There appears to be no relationship between pupil-teacher ratios by town

Linear regression with Boston housing data example

Here,

Y = boston housing prices (also called "target" data in python)

and

X = all the other features (or independent variables)

which we will use to fit a linear regression model and predict Boston housing prices. We will use the least squares method as the way to estimate the coefficients.

We'll use two ways of fitting a linear regression. We recommend the first but the second is also powerful in its features.

Fitting Linear Regression using `statsmodels`

Statsmodels (http://statsmodels.sourceforge.net/) is a great Python library for a lot of basic and inferential statistics. It also provides basic regression functions using an R-like syntax, so it's commonly used by statisticians. While we don't cover statsmodels officially in the Data Science Intensive, it's a good library to have in your toolbox. Here's a quick example of what you could do with it.

In [33]:
```
# Import regression modules
# ols - stands for Ordinary least squares, we'll use this
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [35]:
```
# statsmodels works nicely with pandas dataframes
# The thing inside the "quotes" is called a formula, a bit on that below
m = ols('PRICE ~ RM',bos).fit()
print(m.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  PRICE   R-squared:                       0.48
4
Model:                            OLS   Adj. R-squared:                  0.48
3
Method:                 Least Squares   F-statistic:                     471.
8
Date:                Fri, 12 May 2017   Prob (F-statistic):           2.49e-7
4
Time:                        20:00:44   Log-Likelihood:                -1673.
1
No. Observations:                 506   AIC:                              335
0.
Df Residuals:                     504   BIC:                              335
9.
Df Model:                           1

Covariance Type:            nonrobust

==============================================================================
=
                 coef    std err          t      P>|t|      [95.0% Conf. In
t.]
------------------------------------------------------------------------------
-
Intercept    -34.6706      2.650    -13.084      0.000       -39.877   -29.46
5
RM             9.1021      0.419     21.722      0.000         8.279     9.92
5
==============================================================================
=
Omnibus:                      102.585   Durbin-Watson:                   0.68
4
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              612.44
9
Skew:                           0.726   Prob(JB):                    1.02e-13
3
Kurtosis:                       8.190   Cond. No.                         58.
4
==============================================================================
=

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
```

Interpreting coefficients

There is a ton of information in this output. But we'll concentrate on the coefficient table (middle table). We can interpret the `RM` coefficient (9.1021) by first noticing that the p-value (under `P>|t|`) is so small, basically zero. We can interpret the coefficient as, if we compare two groups of towns, one where the average number of rooms is say 5 and the other group is the same except that they all have 6 rooms. For these two groups the average difference in house prices is about 9.1 (in thousands) so about $\$9,100$ difference. The confidence interval fives us a range of plausible values for this difference, about ($\$8,279, \$9,925$), deffinitely not chump change.

`statsmodels` formulas

This formula notation will seem familiar to `R` users, but will take some getting used to for people coming from other languages or are new to statistics.

The formula gives instruction for a general structure for a regression call. For `statsmodels` (`ols` or `logit`) calls you need to have a Pandas dataframe with column names that you will add to your formula. In the below example you need a pandas data frame that includes the columns named (`Outcome`, `X1`,`X2`, ...), bbut you don't need to build a new dataframe for every regression. Use the same dataframe with all these things in it. The structure is very simple:

`Outcome ~ X1`

But of course we want to to be able to handle more complex models, for example multiple regression is doone like this:

`Outcome ~ X1 + X2 + X3`

This is the very basic structure but it should be enough to get you through the homework. Things can get much more complex, for a quick run-down of further uses see the `statsmodels` help page (http://statsmodels.sourceforge.net/devel/example_formulas.html).

Let's see how our model actually fit our data. We can see below that there is a ceiling effect, we should probably look into that. Also, for large values of Y we get underpredictions, most predictions are below the 45-degree gridlines.

Your turn: Create a scatterpot between the predicted prices, available in `m.fittedvalues` and the original prices. How does the plot look?

Fitting Linear Regression using `sklearn`

In [44]:
```
# Create a scatterplot between predicted prices and original prices.
plt.scatter(m.fittedvalues, bos.PRICE)
plt.xlabel("Predicted prices")
plt.ylabel("Original Price")
plt.title("Relationship between Predicted and Original Price")
```

Out[44]: <matplotlib.text.Text at 0x264b01af550>

There appears to be a positive relationship between the predicted price and original price

In [48]:
```
from sklearn.linear_model import LinearRegression
# Remove the PRICE column from the data frame
X = bos.drop('PRICE', axis = 1)

# This creates a LinearRegression object
lm = LinearRegression()
lm
```

Out[48]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [73]:
```
X
```

Out[73]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386
10	0.22489	12.5	7.87	0.0	0.524	6.377	94.3	6.3467	5.0	311.0	15.2	392
11	0.11747	12.5	7.87	0.0	0.524	6.009	82.9	6.2267	5.0	311.0	15.2	396
12	0.09378	12.5	7.87	0.0	0.524	5.889	39.0	5.4509	5.0	311.0	15.2	390
13	0.62976	0.0	8.14	0.0	0.538	5.949	61.8	4.7075	4.0	307.0	21.0	396
14	0.63796	0.0	8.14	0.0	0.538	6.096	84.5	4.4619	4.0	307.0	21.0	380
15	0.62739	0.0	8.14	0.0	0.538	5.834	56.5	4.4986	4.0	307.0	21.0	395
16	1.05393	0.0	8.14	0.0	0.538	5.935	29.3	4.4986	4.0	307.0	21.0	386
17	0.78420	0.0	8.14	0.0	0.538	5.990	81.7	4.2579	4.0	307.0	21.0	386
18	0.80271	0.0	8.14	0.0	0.538	5.456	36.6	3.7965	4.0	307.0	21.0	288
19	0.72580	0.0	8.14	0.0	0.538	5.727	69.5	3.7965	4.0	307.0	21.0	390
20	1.25179	0.0	8.14	0.0	0.538	5.570	98.1	3.7979	4.0	307.0	21.0	376
21	0.85204	0.0	8.14	0.0	0.538	5.965	89.2	4.0123	4.0	307.0	21.0	392
22	1.23247	0.0	8.14	0.0	0.538	6.142	91.7	3.9769	4.0	307.0	21.0	396
23	0.98843	0.0	8.14	0.0	0.538	5.813	100.0	4.0952	4.0	307.0	21.0	394
24	0.75026	0.0	8.14	0.0	0.538	5.924	94.1	4.3996	4.0	307.0	21.0	394
25	0.84054	0.0	8.14	0.0	0.538	5.599	85.7	4.4546	4.0	307.0	21.0	303
26	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376
27	0.95577	0.0	8.14	0.0	0.538	6.047	88.8	4.4534	4.0	307.0	21.0	306
28	0.77299	0.0	8.14	0.0	0.538	6.495	94.4	4.4547	4.0	307.0	21.0	387
29	1.00245	0.0	8.14	0.0	0.538	6.674	87.3	4.2390	4.0	307.0	21.0	380
...	...	...	...	...	...	...	...	...	...	...	...	...
476	4.87141	0.0	18.10	0.0	0.614	6.484	93.6	2.3053	24.0	666.0	20.2	396

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
477	15.02340	0.0	18.10	0.0	0.614	5.304	97.3	2.1007	24.0	666.0	20.2	349
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0	20.2	379
479	14.33370	0.0	18.10	0.0	0.614	6.229	88.0	1.9512	24.0	666.0	20.2	383
480	5.82401	0.0	18.10	0.0	0.532	6.242	64.7	3.4242	24.0	666.0	20.2	390
481	5.70818	0.0	18.10	0.0	0.532	6.750	74.9	3.3317	24.0	666.0	20.2	393
482	5.73116	0.0	18.10	0.0	0.532	7.061	77.0	3.4106	24.0	666.0	20.2	395
483	2.81838	0.0	18.10	0.0	0.532	5.762	40.3	4.0983	24.0	666.0	20.2	392
484	2.37857	0.0	18.10	0.0	0.583	5.871	41.9	3.7240	24.0	666.0	20.2	370
485	3.67367	0.0	18.10	0.0	0.583	6.312	51.9	3.9917	24.0	666.0	20.2	388
486	5.69175	0.0	18.10	0.0	0.583	6.114	79.8	3.5459	24.0	666.0	20.2	392
487	4.83567	0.0	18.10	0.0	0.583	5.905	53.2	3.1523	24.0	666.0	20.2	388
488	0.15086	0.0	27.74	0.0	0.609	5.454	92.7	1.8209	4.0	711.0	20.1	395
489	0.18337	0.0	27.74	0.0	0.609	5.414	98.3	1.7554	4.0	711.0	20.1	344
490	0.20746	0.0	27.74	0.0	0.609	5.093	98.0	1.8226	4.0	711.0	20.1	318
491	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0	20.1	390
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0	20.1	390
493	0.17331	0.0	9.69	0.0	0.585	5.707	54.0	2.3817	6.0	391.0	19.2	390
494	0.27957	0.0	9.69	0.0	0.585	5.926	42.6	2.3817	6.0	391.0	19.2	390
495	0.17899	0.0	9.69	0.0	0.585	5.670	28.8	2.7986	6.0	391.0	19.2	393
496	0.28960	0.0	9.69	0.0	0.585	5.390	72.9	2.7986	6.0	391.0	19.2	390
497	0.26838	0.0	9.69	0.0	0.585	5.794	70.6	2.8927	6.0	391.0	19.2	390
498	0.23912	0.0	9.69	0.0	0.585	6.019	65.3	2.4091	6.0	391.0	19.2	390
499	0.17783	0.0	9.69	0.0	0.585	5.569	73.5	2.3999	6.0	391.0	19.2	395
500	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6.0	391.0	19.2	390
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	390
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	390
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	390

506 rows × 13 columns

What can you do with a LinearRegression object?

Check out the scikit-learn docs here (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html). We have listed the main functions here.

Main functions	**Description**
`lm.fit()`	Fit a linear model
`lm.predit()`	Predict Y using the linear model with estimated coefficients
`lm.score()`	Returns the coefficient of determination (R^2). *A measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model*

What output can you get?

In [50]:
```
# Look inside lm object
# lm.<tab>
# Output from a linear model
# lm.coef_ - Estimated coefficients
# lm.intercept_ - Estimated intercep
lm
```

Out[50]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Output	**Description**
`lm.coef_`	Estimated coefficients
`lm.intercept_`	Estimated intercept

Fit a linear model

The `lm.fit()` function estimates the coefficients the linear regression using least squares.

In [55]:
```
# Use all 13 predictors to fit linear regression model
lm.fit(X, bos.PRICE)
```

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

Your turn: How would you change the model to not fit an intercept term? Would you recommend not having an intercept?

Estimated intercept and coefficients

Let's look at the estimated coefficients from the linear model using `lm.intercept_` and `lm.coef_`.

After we have fit our linear regression model using the least squares method, we want to see what are the estimates of our coefficients β_0, β_1, ..., β_{13}:

$$\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_{13}$$

In [57]:
```
print('Estimated intercept coefficient:', lm.intercept_)
```

```
Estimated intercept coefficient: 36.4911032804
```

In [58]:
```
print('Number of coefficients:', len(lm.coef_))
```

```
Number of coefficients: 13
```

In [70]:
```
# list df columns
X.columns
```

Out[70]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
X',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

In [71]:
```
# list related coefficients
lm.coef_
```

Out[71]:
```
array([ -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])
```

In [69]:
```
# Align columns and coefficients into dataframe
pd.DataFrame(list(zip(X.columns, lm.coef_)), columns = ['features', 'estimated
Coefficients'])
```

Out[69]:

	features	estimatedCoefficients
0	CRIM	-0.107171
1	ZN	0.046395
2	INDUS	0.020860
3	CHAS	2.688561
4	NOX	-17.795759
5	RM	3.804752
6	AGE	0.000751
7	DIS	-1.475759
8	RAD	0.305655
9	TAX	-0.012329
10	PTRATIO	-0.953464
11	B	0.009393
12	LSTAT	-0.525467

Predict Prices

We can calculate the predicted prices ($\hat{Y}_i$) using `lm.predict`.

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \ldots \hat{\beta}_{13} X_{13}$$

In [82]:
```
# first five predicted prices
bos_pre = lm.predict(X)[0:5]
all_bos_pre = lm.predict(X)[0:505]
lm.predict(X)[0:5]
```

Out[82]:
```
array([ 30.00821269,  25.0298606 ,  30.5702317 ,  28.60814055,  27.94288232])
```

In [78]:
```
bos_orig = bos.PRICE.head(6)
bos.PRICE.head(6)
```

Out[78]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
5    28.7
Name: PRICE, dtype: float64
```

In [79]:
```
# Compare predicted with original prices
pd.DataFrame(list(zip(bos_pre, bos_orig)), columns = ['predicted',
'original'])
```

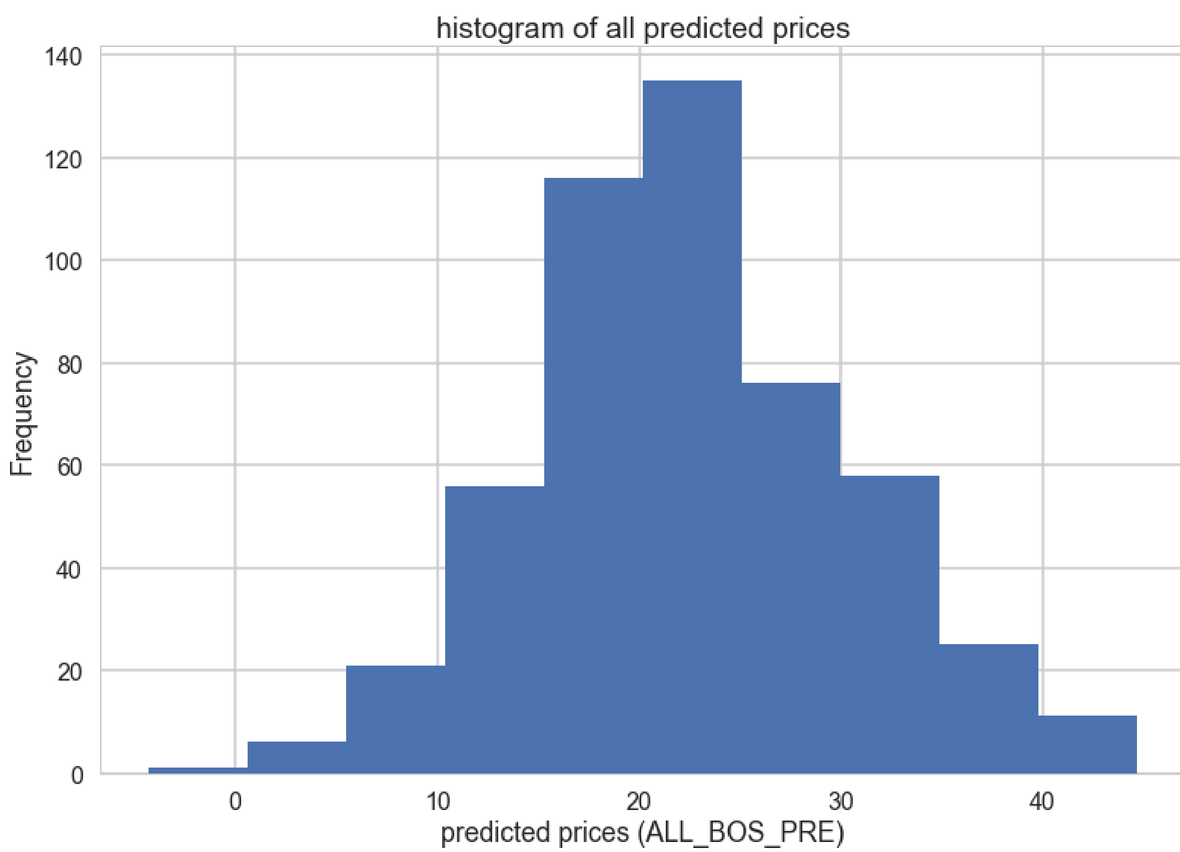
Out[79]:

	predicted	original
0	30.008213	24.0
1	25.029861	21.6
2	30.570232	34.7
3	28.608141	33.4
4	27.942882	36.2

Your turn:

- Histogram: Plot a histogram of all the predicted prices
- Scatter Plot: Let's plot the true prices compared to the predicted prices to see they disagree (we did this with `statsmodels` before).

In [83]:
```
# your turn
plt.hist(all_bos_pre)
plt.title("histogram of all predicted prices")
plt.xlabel("predicted prices (ALL_BOS_PRE)")
plt.ylabel("Frequency")
plt.show()
```



Residual sum of squares

Let's calculate the residual sum of squares

$$S = \sum_{i=1}^{N} r_i = \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_i))^2$$

In [85]:
```
# calculate the residual sum of squares
print(np.sum((bos.PRICE - lm.predict(X)) ** 2))
```

```
11080.276284149868
```

Mean squared error

This is simply the mean of the residual sum of squares.

Your turn: Calculate the mean squared error and print it.

In [98]:
```
# Calculate the mean squared error
# mse = ((A - B) ** 2).mean(axis=ax)
# with ax=0 the average is performed along the row, for each column, returning
 an array
# with ax=1 the average is performed along the column, for each row, returning
 an array
# with ax=None the average is performed element-wise along the array, returnin
g a single value
mse = ((bos.PRICE - lm.predict(X))** 2).mean(axis=0)
mse
```

Out[98]:
```
21.897779217687486
```

In [99]:
```
# your turn
# Calculate the mean squared error
mse = np.mean((bos.PRICE - lm.predict(X))**2)
print('The mean squared error is', mse)
```

```
The mean squared error is 21.897779217687486
```

Relationship between `PTRATIO` and housing price

Try fitting a linear regression model using only the 'PTRATIO' (pupil-teacher ratio by town)

Calculate the mean squared error.

In [100]:
```
lm = LinearRegression()
lm.fit(X[['PTRATIO']], bos.PRICE)
```

Out[100]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

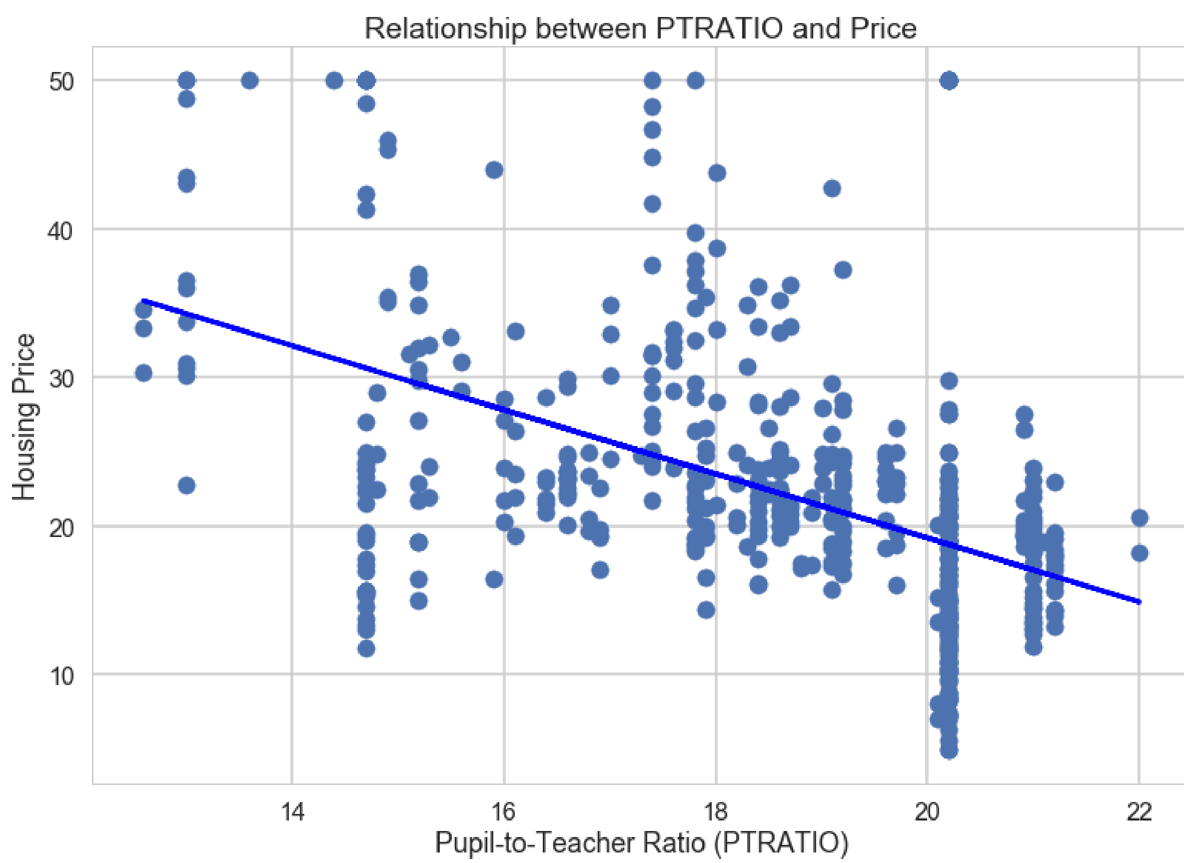
In [101]:
```
msePTRATIO = np.mean((bos.PRICE - lm.predict(X[['PTRATIO']])) ** 2)
print(msePTRATIO)
```

```
62.65220001376927
```

We can also plot the fitted linear regression line.

In [102]:
```
plt.scatter(bos.PTRATIO, bos.PRICE)
plt.xlabel("Pupil-to-Teacher Ratio (PTRATIO)")
plt.ylabel("Housing Price")
plt.title("Relationship between PTRATIO and Price")

plt.plot(bos.PTRATIO, lm.predict(X[['PTRATIO']]), color='blue', linewidth=3)
plt.show()
```



Your turn

Try fitting a linear regression model using three independent variables

1. 'CRIM' (per capita crime rate by town)
2. 'RM' (average number of rooms per dwelling)
3. 'PTRATIO' (pupil-teacher ratio by town)

Calculate the mean squared error.

In [105]:
```
# your turn
# Create linear regression object
lm = LinearRegression()
lm.fit(X[['CRIM','RM','PTRATIO']],bos.PRICE)
```

Out[105]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [107]:
```
mseCRIM_RM_PTRATIO = np.mean((bos.PRICE -
lm.predict(X[['CRIM','RM','PTRATIO']])) ** 2)
print('The mean squared error is',mseCRIM_RM_PTRATIO)
```

```
The mean squared error is 34.32379656468118
```

Other important things to think about when fitting a linear regression model

- **Linearity**. The dependent variable Y is a linear combination of the regression coefficients and the independent variables X.
- **Constant standard deviation**. The SD of the dependent variable Y should be constant for different values of X.
    - e.g. PTRATIO
- **Normal distribution for errors**. The ϵ term we discussed at the beginning are assumed to be normally distributed.
$$\epsilon_i \sim N(0, \sigma^2)$$
Sometimes the distributions of responses Y may not be normally distributed at any given value of X. e.g. skewed positively or negatively.
- **Independent errors**. The observations are assumed to be obtained independently.
    - e.g. Observations across time may be correlated

Part 3: Training and Test Data sets

Purpose of splitting data into Training/testing sets

Let's stick to the linear regression example:

- We built our model with the requirement that the model fit the data well.
- As a side-effect, the model will fit **THIS** dataset well. What about new data?
  - We wanted the model for predictions, right?
- One simple solution, leave out some data (for **testing**) and **train** the model on the rest
- This also leads directly to the idea of cross-validation, next section.

One way of doing this is you can create training and testing data sets manually.

In [109]:
```
X_train = X[:-50]
X_test = X[-50:]
Y_train = bos.PRICE[:-50]
Y_test = bos.PRICE[-50:]
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(456, 13)
(50, 13)
(456,)
(50,)
```

Another way, is to split the data into random train and test subsets using the function `train_test_split` in `sklearn.cross_validation`. Here's the documentation (http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html).

In [113]:
```
#X_train, X_test, Y_train, Y_test = sklearn.cross_validation.train_test_split(
#    X, bos.PRICE, test_size=0.33, random_state = 5)

# correct command syntax for python version 3
X_train, X_test, Y_train, Y_test = train_test_split(X, bos.PRICE, test_size=0.
33, random_state = 5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339,)
(167,)
```

Your turn: Let's build a linear regression model using our new training data sets.

- Fit a linear regression model to the training set
- Predict the output on the test set

In [135]:
```
# your turn
# Fit a linear regression model to the training set
# Create linear regression object
lm_train = LinearRegression()
lm_train.fit(X_train,Y_train)
```

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

In [129]:
```
# your turn
# Fit a linear regression model to the test set
# Create linear regression object
lm_test = LinearRegression()
lm_test.fit(X_test,Y_test)
```

Out[129]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Your turn:

Calculate the mean squared error

- using just the test data
- using just the training data

Are they pretty similar or very different? What does that mean?

In [138]:
```
# your turn
# mean squared error using just the test data
mse_test = np.mean((Y_test - lm_test.predict(X_test)) ** 2)
print('The mean squared error is',mse_test)
```

```
The mean squared error is 24.048796968678072
```

In [139]:
```
# your turn
# mean squared error using just the training data
mse_train = np.mean((Y_train - lm_test.predict(X_train)) ** 2)
print('The mean squared error is',mse_train)
```

```
The mean squared error is 22.71882539268712
```

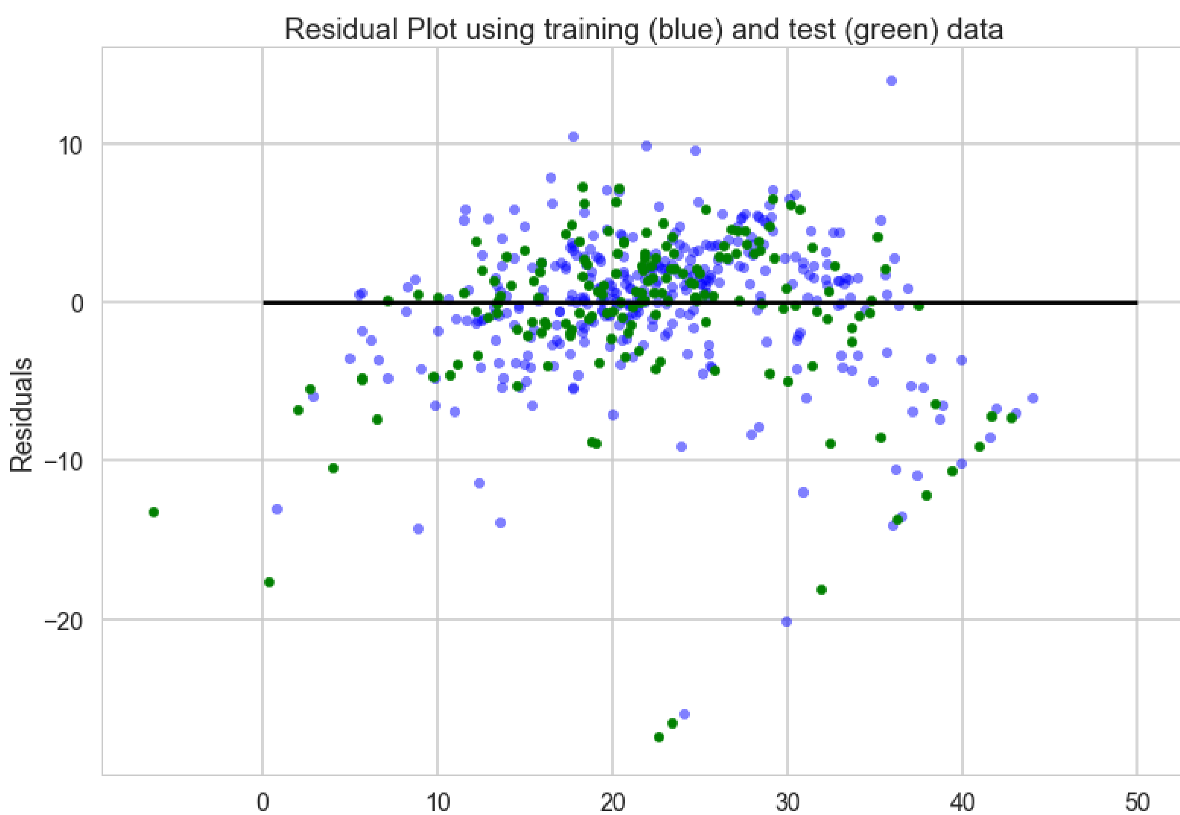
The mean squared errors for the test and training data sets are pretty similar.

This means that the predictive performance of the "test" data set is consistent with that of the "train" data set. This can be interpreted to mean that when this linear model is used on new, unseen data it should work well at prediction.

Residual plots

In [140]:
```
plt.scatter(lm.predict(X_train), lm.predict(X_train) - Y_train, c='b', s=40, a
lpha=0.5)
plt.scatter(lm.predict(X_test), lm. predict(X_test) - Y_test, c='g', s=40)
plt.hlines(y = 0, xmin=0, xmax = 50)
plt.title('Residual Plot using training (blue) and test (green) data')
plt.ylabel('Residuals')
```

Out[140]: <matplotlib.text.Text at 0x264b1b702e8>



Your turn: Do you think this linear regression model generalizes well on the test data?

Based on the above plot its appears that this linear regression model generalizes well on the test data since the train and test data points appear to mirror each other.

K-fold Cross-validation as an extension of this idea

A simple extension of the Test/train split is called K-fold cross-validation.

Here's the procedure:

- randomly assign your n samples to one of K groups. They'll each have about n/k samples
- For each group k:
  - Fit the model (e.g. run regression) on all data excluding the k^{th} group
  - Use the model to predict the outcomes in group k
  - Calculate your prediction error for each observation in k^{th} group (e.g. $(Y_i - \hat{Y}_i)^2$ for regression, $1(Y_i = \hat{Y}_i)$ for logistic regression).
- Calculate the average prediction error across all samples $Err_{CV} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$

Luckily you don't have to do this entire process all by hand (`for` loops, etc.) every single time, `sci-kit learn` has a very nice implementation of this, have a look at the documentation (http://scikit-learn.org/stable/modules/cross_validation.html).

Your turn (extra credit): Implement K-Fold cross-validation using the procedure above and Boston Housing data set using $K = 4$. How does the average prediction error compare to the train-test split above?

In [ ]:

```

```