

Regression in Python

This is a very quick run-through of some basic statistical concepts, adapted from [Lab 4 in Harvard's CS109](#) 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.

In [79]:

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

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

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

```
C:\Users\anands\AppData\Local\Continuum\Anaconda3\lib\site-
packages\IPython\core\magics\pylab.py:161: UserWarning: pylab import has cl
obbered these variables: ['clf']
`%matplotlib` prevents importing * from pylab and numpy
"\n`%matplotlib` prevents importing * from pylab and numpy"
```

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](#) is a method to model the relationship between a set of independent variables X (also known 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, \dots

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

- 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.

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



<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](#) 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](#) 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](#) python module which is how we will access it today.

In [80]:

```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [81]:

```
boston.keys()
```

Out [81]:

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

```
In [82]:
```

```
boston.data.shape
```

```
Out[82]:
```

```
(506, 13)
```

```
In [83]:
```

```
# Print column names
print (boston.feature_names)
```

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

```
In [84]:
```

```
# 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 Carnegie 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 that 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, 236-243, 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 [85]:

```
import pandas as pd
bos = pd.DataFrame(boston.data)
bos.head()
```

Out[85]:

	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 [86]:

```
bos.columns = boston.feature_names
bos.head()
```

Out[86]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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

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 [87]:

```
print (boston.target.shape)

(506,)
```

In [88]:

```
bos['PRICE'] = boston.target
bos.head()
```

Out [88]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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

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 [89]:

```
bos.describe()
```

Out [89]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.561000
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.819753
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.000000

25%	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	L
	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	5
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	1

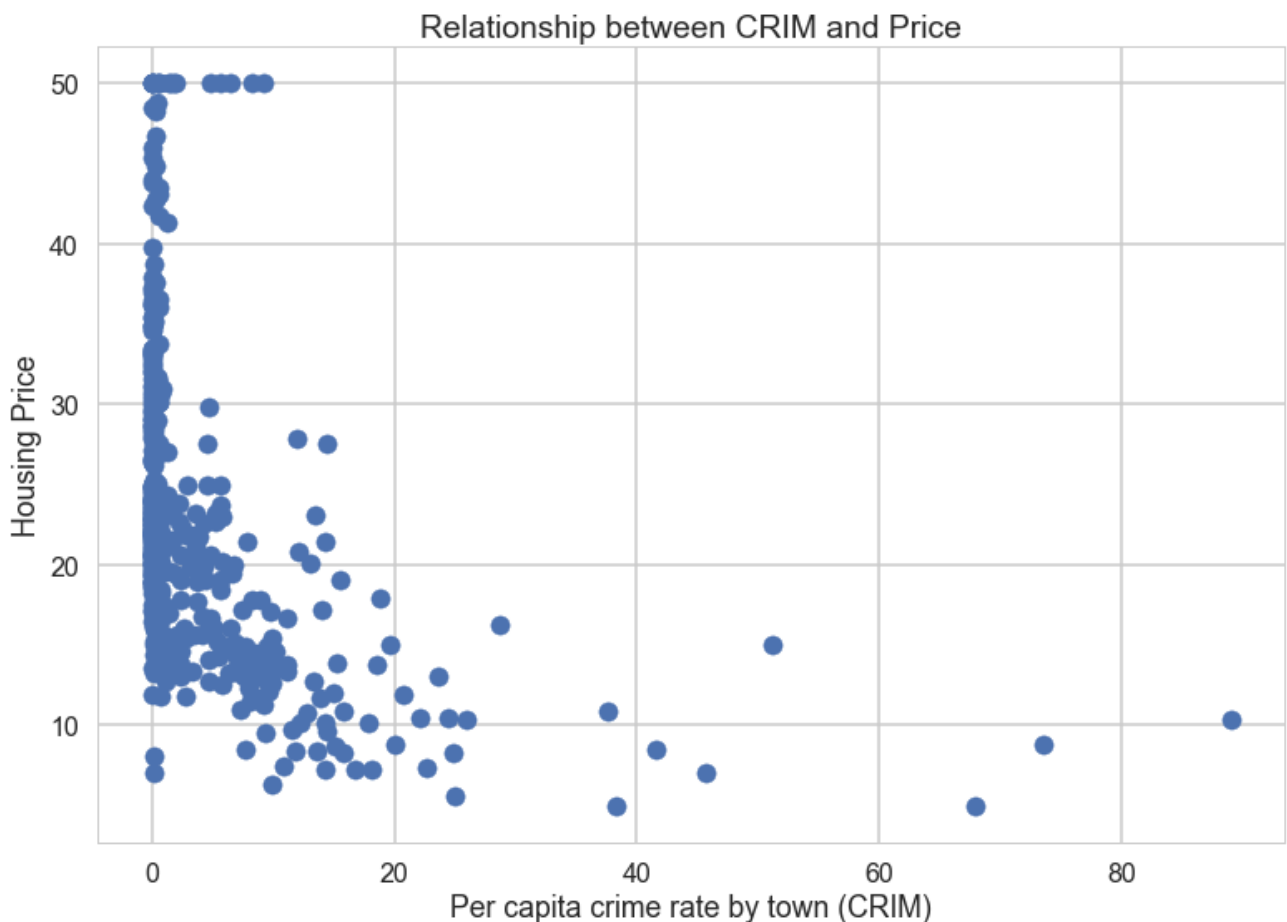
Scatter plots

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 [90]:

```
import matplotlib.pyplot as plt
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")
plt.show()
```



As obvious, there is -ve linear association between Housing Price and Per capita crime rate. House prices tend to get affected as crime rate increases.

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

notice?

In [91]:

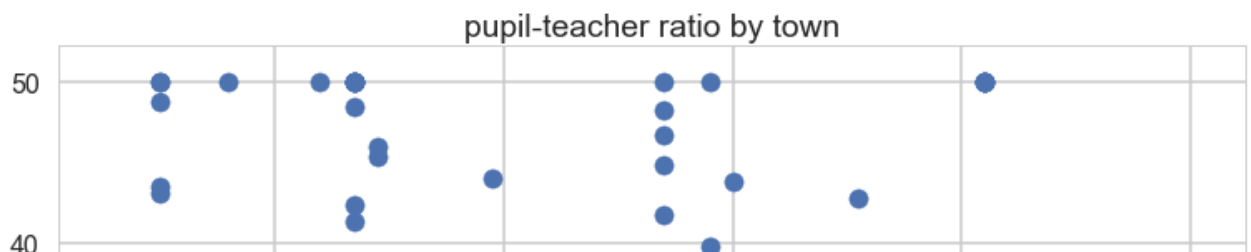
```
#your turn: scatter plot between *RM* and *PRICE*
import matplotlib.pyplot as plt
plt.scatter(bos.RM, bos.PRICE)
plt.xlabel("Average number of rooms per dwelling (RM)")
plt.ylabel("Housing Price")
plt.title("Relationship between RM and Price")
plt.show()
```

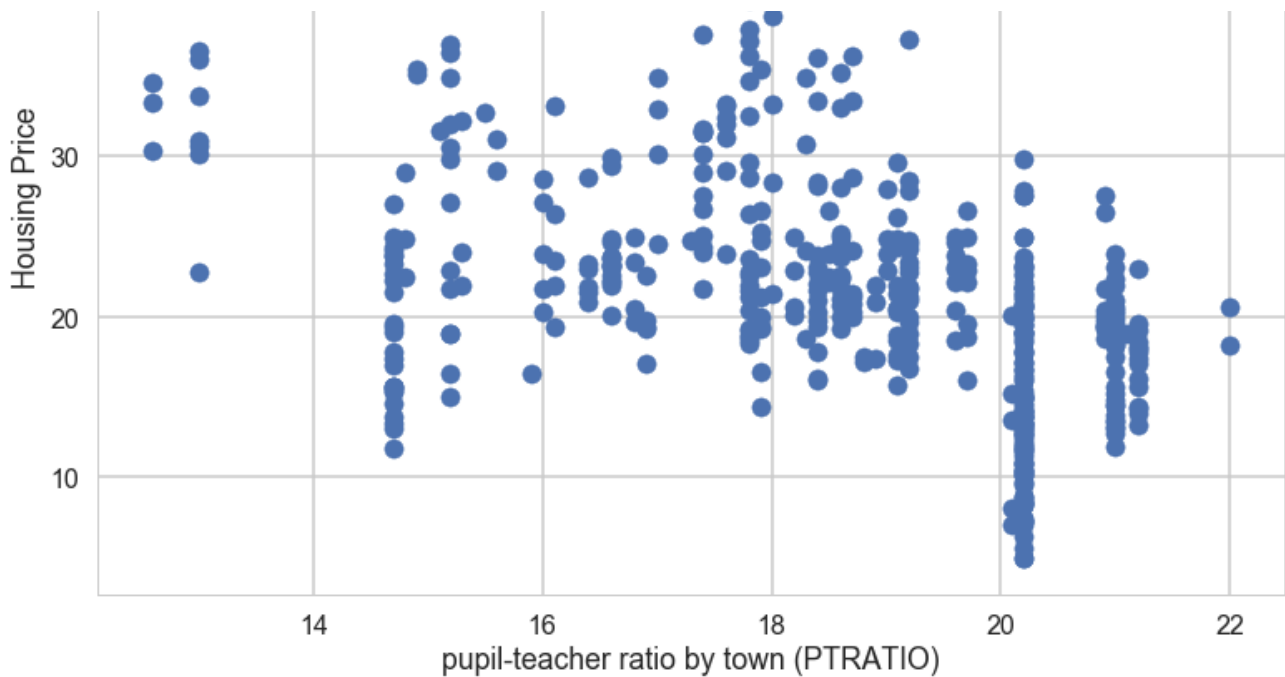


The graph is positively linear. Number of rooms per dwelling is definitely a function of Housing price. As rooms increase , so is the house price.

In [92]:

```
#your turn: scatter plot between *PTRATIO* and *PRICE*
import matplotlib.pyplot as plt
plt.scatter(bos.PTRATIO, bos.PRICE)
plt.xlabel("pupil-teacher ratio by town (PTRATIO)")
plt.ylabel("Housing Price")
plt.title("pupil-teacher ratio by town")
plt.show()
```



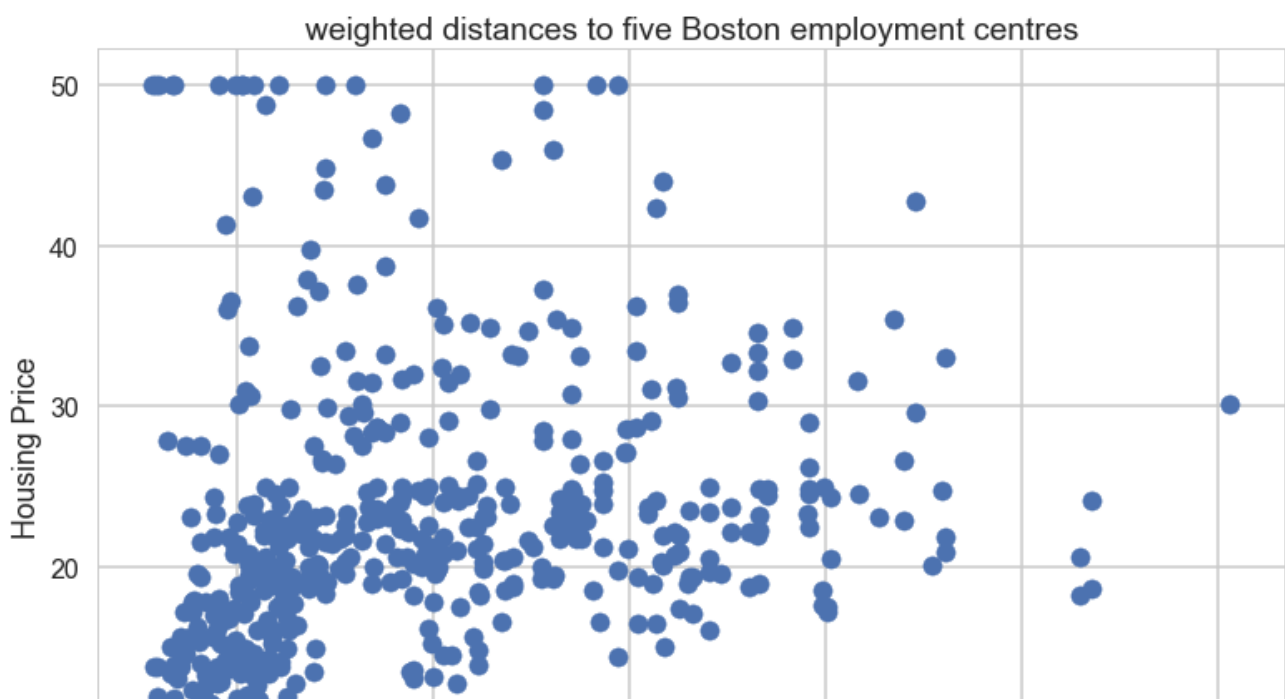


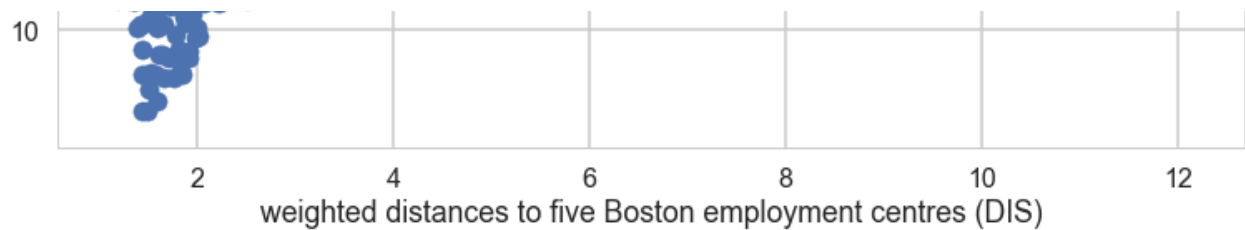
The plot does not depict linear relationship b/w pupil-teacher ratio and housing price. For example , houses are priced lower and higher both when pupil-teacher ratio is 15 or 18. There is no relationship as such. It's non-linear.

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

In [93]:

```
#your turn: create some other scatter plots
#your turn: scatter plot between *DIS* and *PRICE*
import matplotlib.pyplot as plt
plt.scatter(bos.DIS, bos.PRICE)
plt.xlabel("weighted distances to five Boston employment centres (DIS)")
plt.ylabel("Housing Price")
plt.title("weighted distances to five Boston employment centres")
plt.show()
```

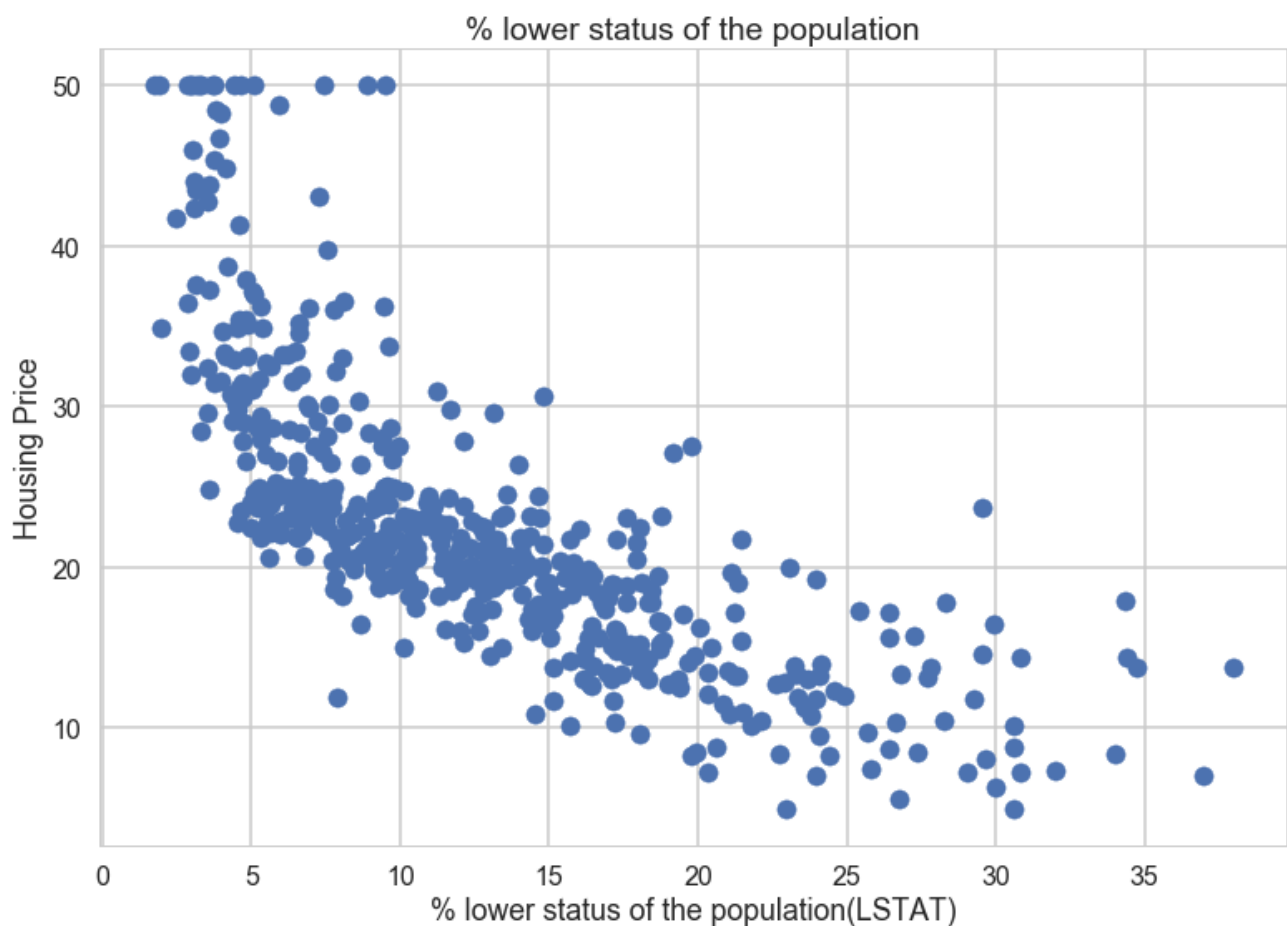




Seems like a linear relationship between house prices and weighted did to employment centres. Proximity to the employment centre also impacts the house prices.

In [94]:

```
#your turn: scatter plot between *LSTAT* and *PRICE*
import matplotlib.pyplot as plt
plt.scatter(bos.LSTAT, bos.PRICE)
plt.xlabel("% lower status of the population(LSTAT)")
plt.ylabel("Housing Price")
plt.title("% lower status of the population")
plt.show()
```

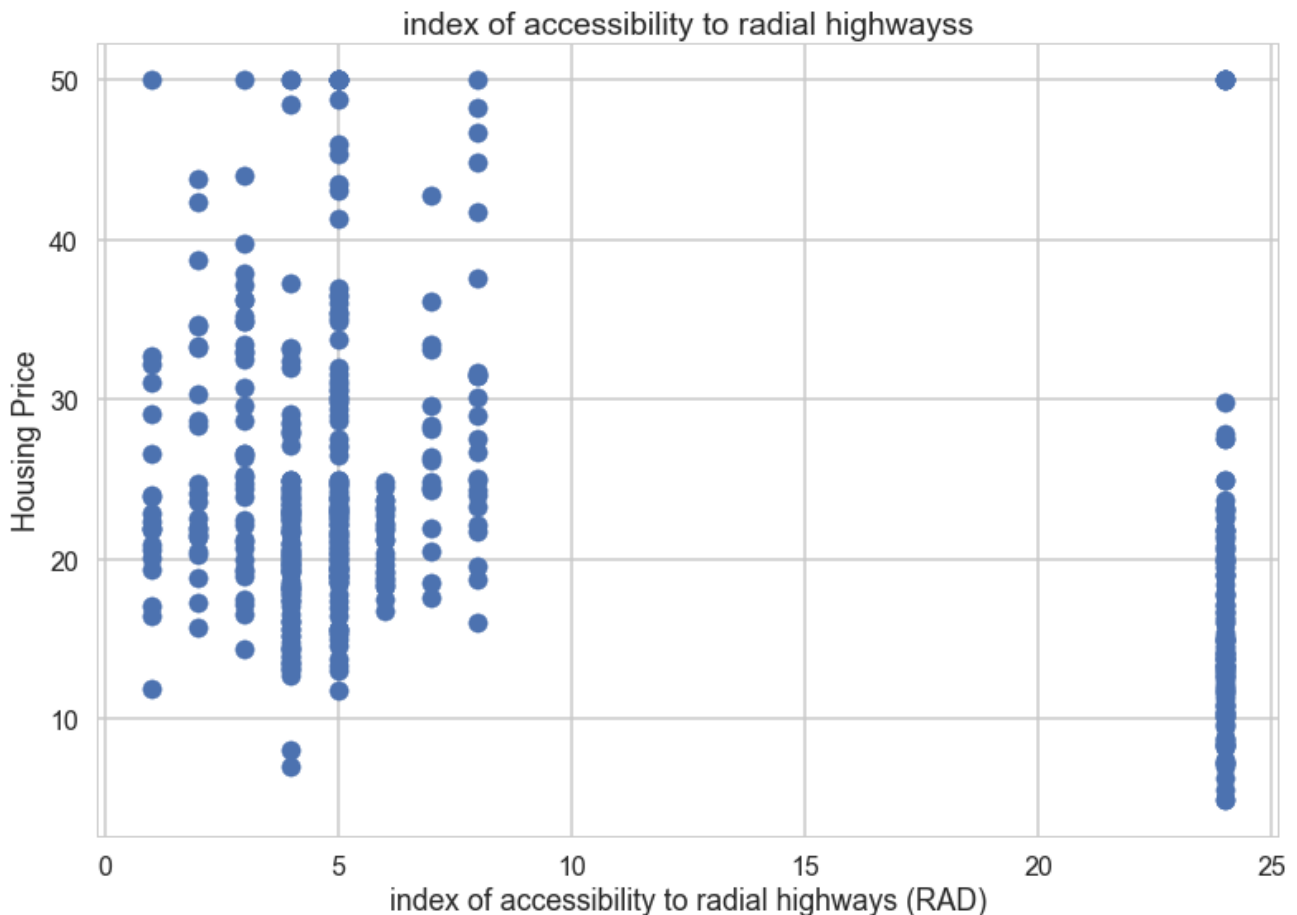


The plot shows -ve linear association between lower status of population and Housing prices. Intuitive enough highly priced houses has a positive linear relationship with affordability/income status. Higher the % lower population status, higher the house prices.

In [95]:

```
#your turn: scatter plot between *RAD* and *PRICE*
import matplotlib.pyplot as plt
plt.scatter(bos.RAD, bos.PRICE)
plt.xlabel("index of accessibility to radial highways (RAD)")
```

```
plt.ylabel("Housing Price")
plt.title("index of accessibility to radial highwayss")
plt.show()
```



Interestingly , accebility to radial highways do not impact home prices. My guess would be that accebility index could be individual dependent meaning it might be an important criteria for home buying for some and might not be as important to others. We see data is quite dispersed and hence it depicts non-linear relationship.

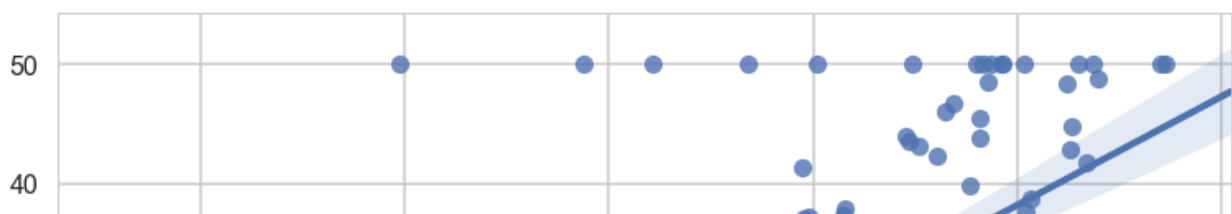
Scatter Plots using Seaborn

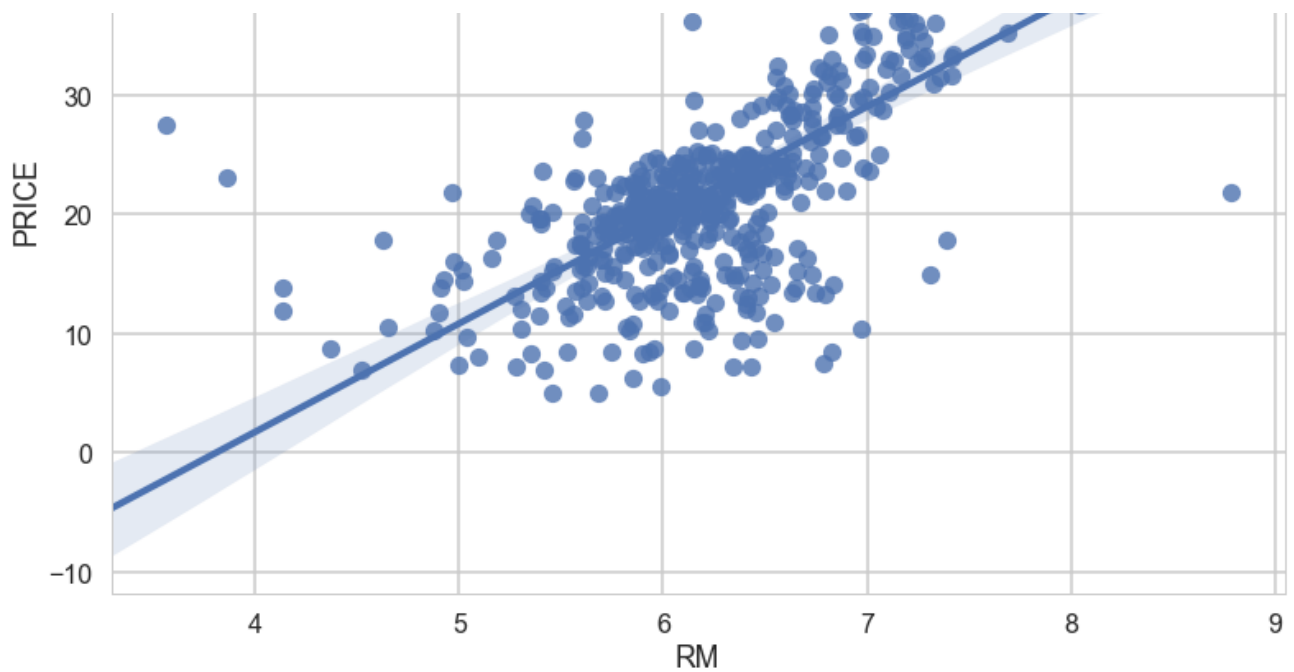
[Seaborn](#) is a cool Python plotting library built on top of matplotlib. It provides convenient syntax and shortcuts for many common types of plots, along with better-looking defaults.

We can also use [seaborn regplot](#) for the scatterplot above. This provides automatic linear regression fits (useful for data exploration later on). Here's one example below.

In [96]:

```
import seaborn as sns
sns.regplot(y="PRICE", x="RM", data=bos, fit_reg = True);
plt.show()
```



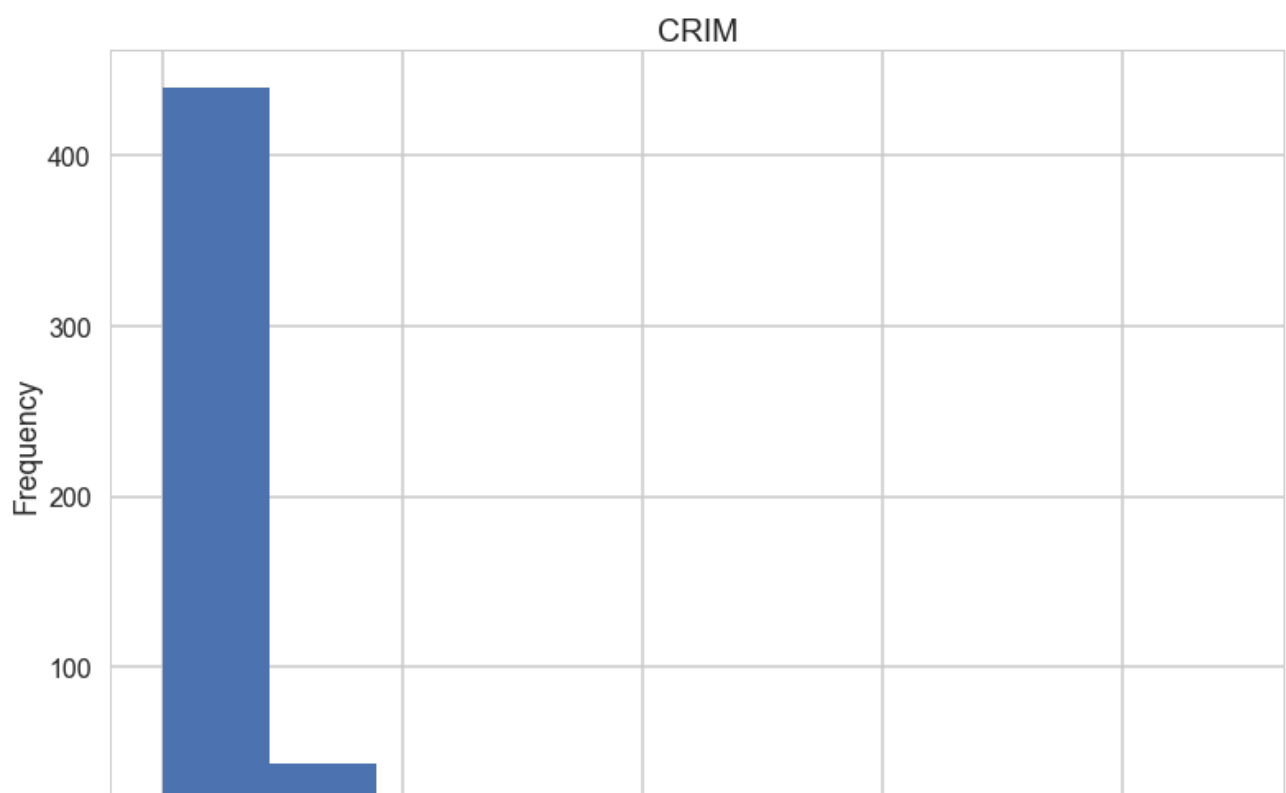


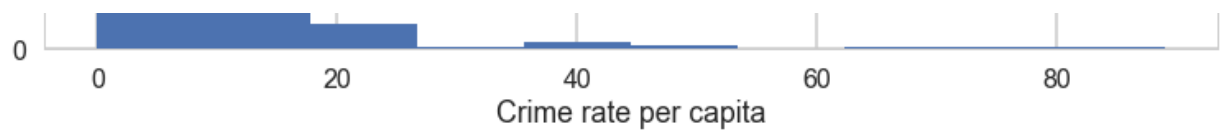
Histograms

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 [97]:

```
plt.hist(bos.CRIM)
plt.title("CRIM")
plt.xlabel("Crime rate per capita")
plt.ylabel("Frequency")
plt.show()
```

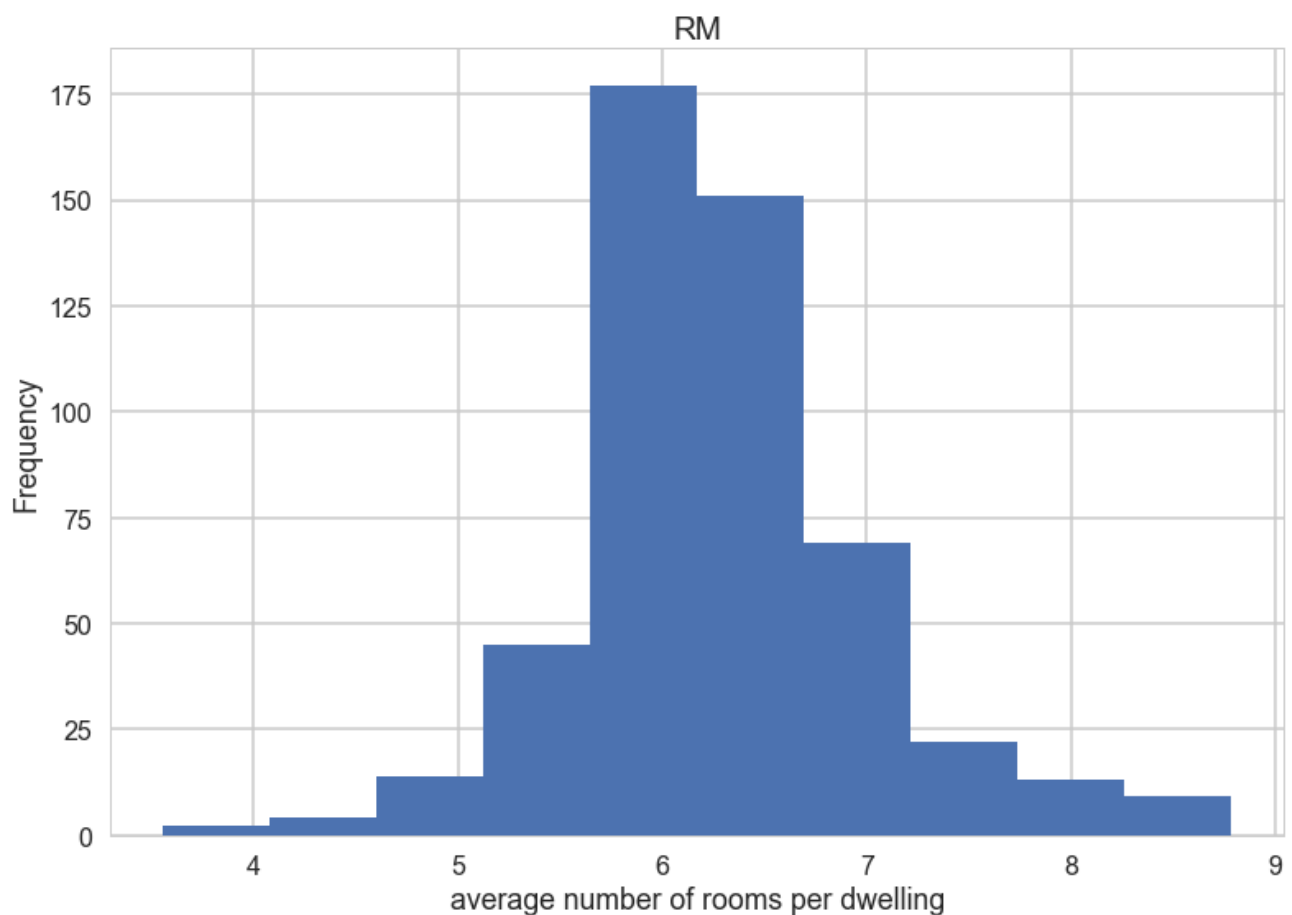




Your turn: Plot separate histograms and one for *RM*, one for *PTRATIO*. Any interesting observations?

In [98]:

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

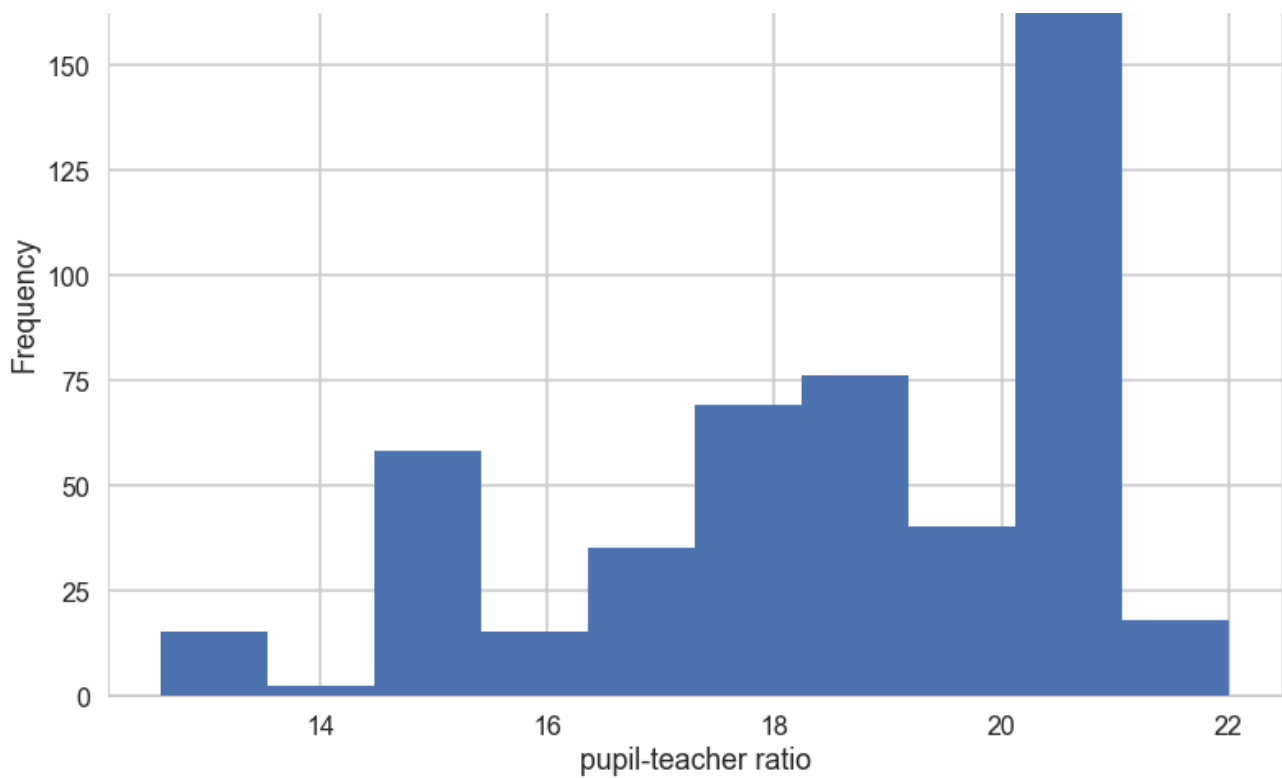


Observation:- Histogram is taking almost a bell-shaped curve for RM hinting “normal distribution

In [99]:

```
#your turn
plt.hist(bos.PTRATIO)
plt.title("PTRATIO")
plt.xlabel("pupil-teacher ratio")
plt.ylabel("Frequency")
plt.show()
```





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](#) 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 [100]:

```
# 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 [101]:

```
# 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:          0.4
4
Model:                  OLS      Adj. R-squared:      0.4
3
Method:                 Least Squares    F-statistic:      47.
8
Date:                   Sat, 19 Aug 2017    Prob (F-statistic):      2.49e-
74
Time:                   16:48:10    Log-Likelihood:      -167.
1
No. Observations:      506    AIC:                  33.
.
Df Residuals:          504    BIC:                  33.
.
Df Model:               1
Covariance Type:       nonrobust
```

```
=====
              coef      std err          t      P>|t|      [95.0% Conf. Int
]
-----
Intercept    -34.6706      2.650     -13.084      0.000     -39.877     -29.4
65
RM           9.1021      0.419      21.722      0.000       8.279      9.1
5
=====
```

```
Omnibus:          102.585    Durbin-Watson:          0.4
4
Prob(Omnibus):    0.000    Jarque-Bera (JB):
612.449
Skew:            0.726    Prob(JB):          1.02e-
3
Kurtosis:        8.190    Cond. No.          58
4
=====
```

Warnings:

```
[1] Standard Errors assume that the covariance matrix of the errors is corr
ectly specified.
```

Interpreting coefficients

There is a ton of information in this output. But we'll concentrate on the coefficient table (middle

There is a lot 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 gives us a range of plausible values for this difference, about (\$8,279, \$9,925\$), definitely 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`, ...), but 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 be able to handle more complex models, for example multiple regression is done like this:

```
Outcome ~ X1 + X2 + X3
```

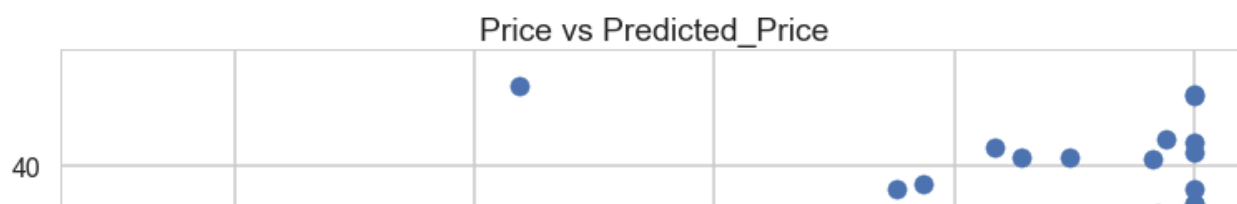
This is the very basic structure but it should be enough to get you through the homework. Things can get much more complex, for a quick run-down of further uses see the `statsmodels` [help page](#).

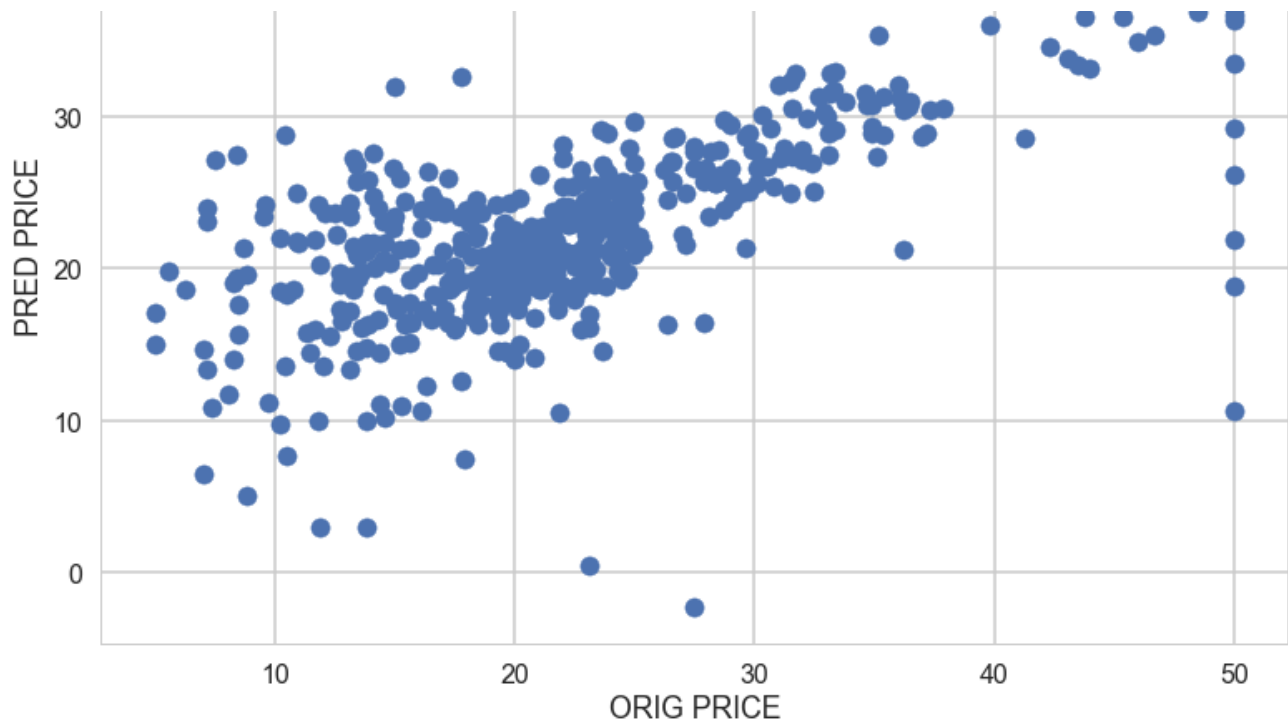
Let's see how our model actually fit our data. We can see below that there is a ceiling effect, we should probably look into that. Also, for large values of \$Y\$ we get underpredictions, most predictions are below the 45-degree gridlines.

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

```
In [102]:
```

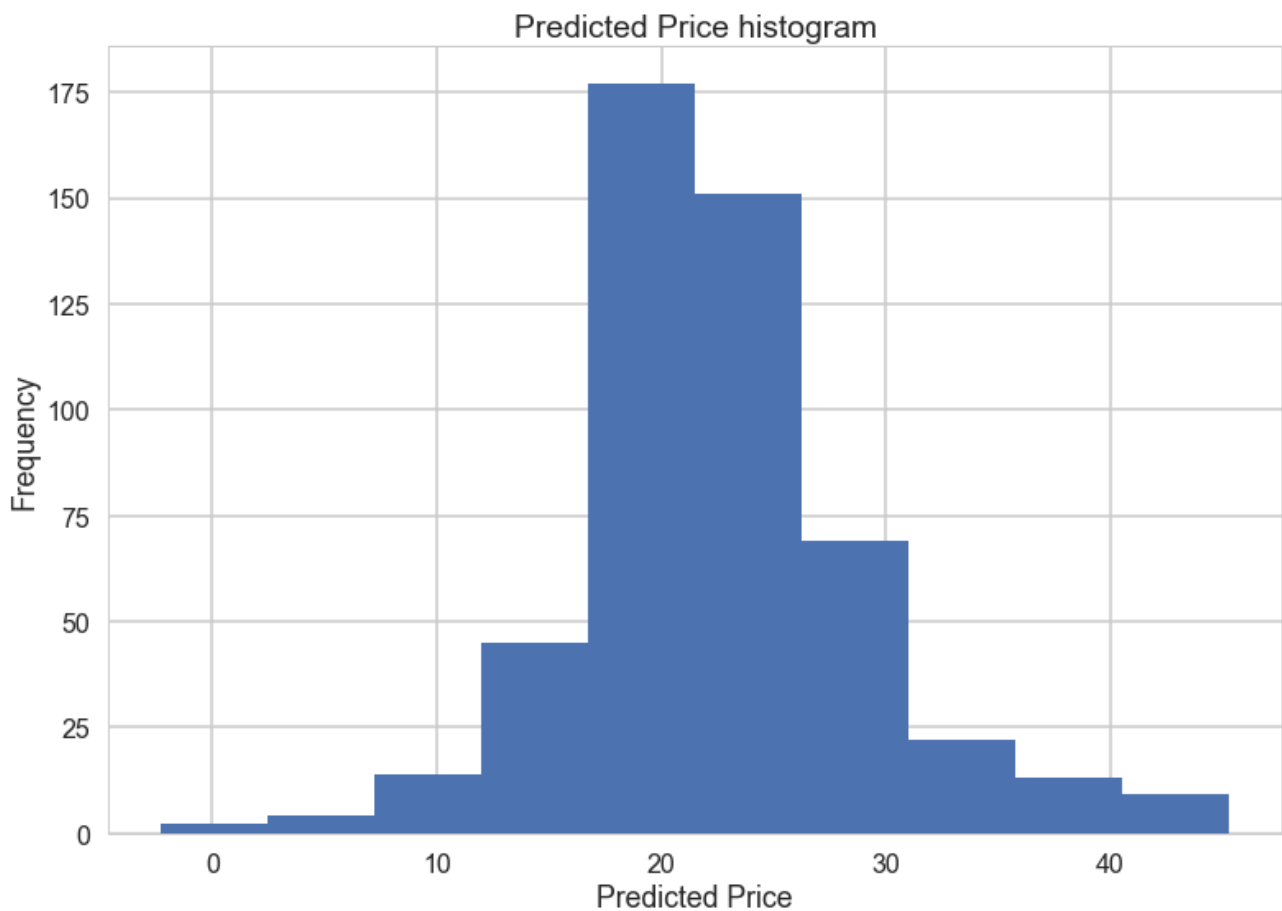
```
# your turn
a = m.fittedvalues
bos['PRED_PRICE'] = m.fittedvalues
plt.xlabel('ORIG PRICE')
plt.ylabel('PRED PRICE')
plt.scatter(bos.PRICE, bos.PRED_PRICE)
plt.title("Price vs Predicted_Price")
plt.show()
```





In [103]:

```
#your turn
plt.hist(bos.PRED_PRICE)
plt.title("Predicted Price histogram")
plt.xlabel("Predicted Price")
plt.ylabel("Frequency")
plt.show()
```



Fitting Linear Regression using sklearn

In [104]:

```
from sklearn.linear_model import LinearRegression
X = bos.drop(['PRICE', 'PRED_PRICE'], axis = 1)
y = bos['PRICE']
# # This creates a LinearRegression object
lm = LinearRegression()
print(lm)
```

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

What can you do with a LinearRegression object?

Check out the scikit-learn [docs here](#). We have listed the main functions here.

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

What output can you get?

In [105]:

```
# Look inside lm object
# dir(lm)
# lm.predict()
```

Output	Description
<code>lm.coef_</code>	Estimated coefficients
<code>lm.intercept_</code>	Estimated intercept

Fit a linear model

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

In [106]:

```
# Use all 13 predictors to fit linear regression model
lm.fit(X, bos.PRICE)
print(lm.coef_)
```

```
[ -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]
```

Your turn: How would you change the model to not fit an intercept term? Would you recommend not having an intercept? `print(lm.coef_)`

In [107]:

```
# Create a LinearRegression obj with no intercept
lm_no_intercept = LinearRegression(fit_intercept = False)
lm_no_intercept.fit(X, bos.PRICE)
```

Out[107]:

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

A regression without an intercept means that the regression line should run through the origin, which in turn means that there is a point where both the predictor variable and target variable = 0. In this case, when there are 0 rooms in a house, the house price should not mean anything and it's equal to 0 as well. So, when we run the regression without a constant in the model, we are declaring that the expected value of Y is also 0 when x is equal to 0. It can be taken off for this model.

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 $\beta_0, \beta_1, \dots, \beta_{13}$:

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

In [108]:

```
print ('Estimated intercept coefficient:', lm.intercept_)
```

```
Estimated intercept coefficient: 36.4911032804
```

In [109]:

```
print ('Number of coefficients:', len(lm.coef_))
```

```
Number of coefficients: 13
```

In [110]:

```
X.head()
```

Out[110]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
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

◀													▶
---	--	--	--	--	--	--	--	--	--	--	--	--	---

In [111]:

```
# The coefficients
pd.DataFrame(list(zip(X.columns, lm.coef_)), columns = ['features',
'estimatedCoefficients'])
```

Out[111]:

	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`.

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

In [112]:

```
# first five predicted prices
lm.predict(X)[0:5]
```

Out[112]:

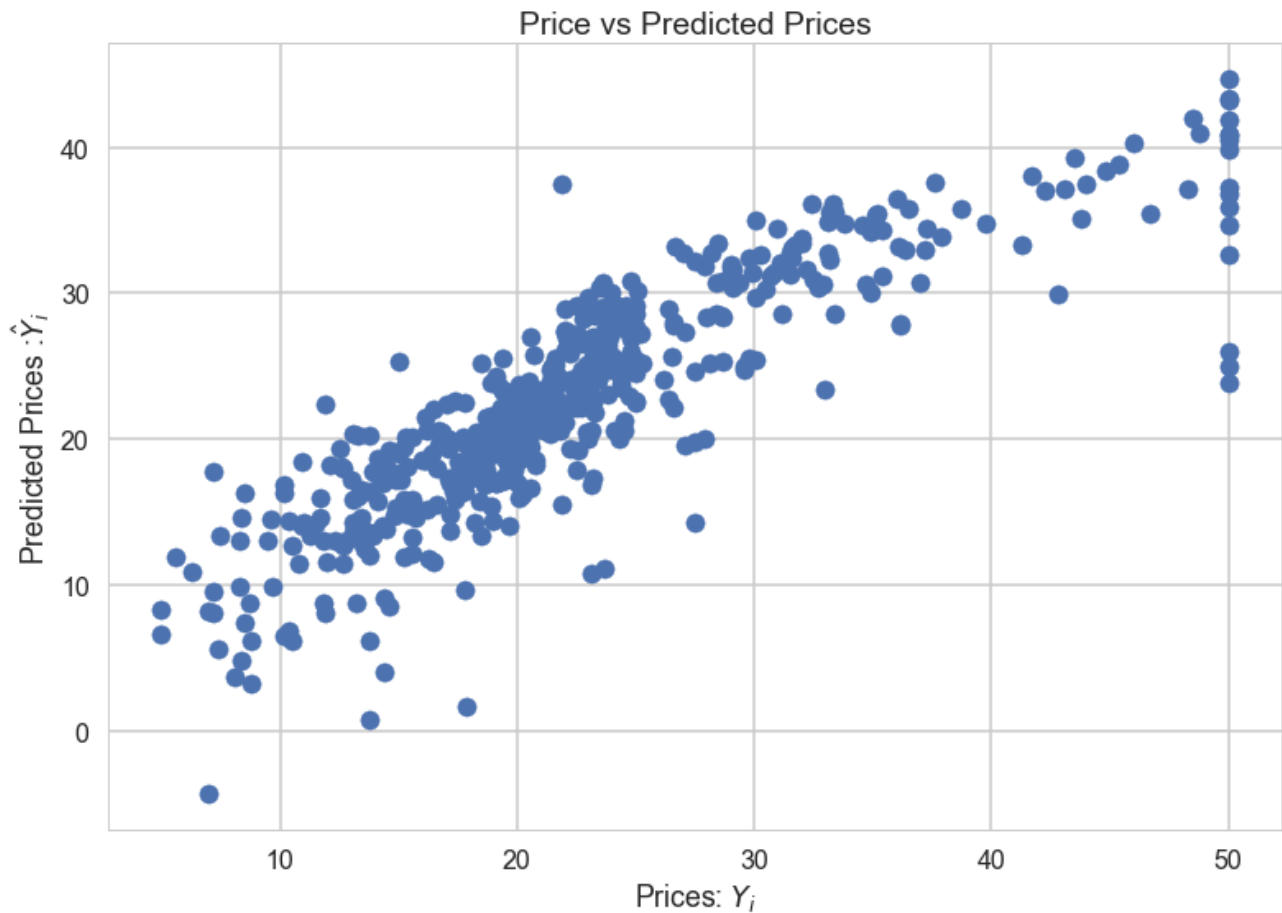
```
array([ 30.00821269,  25.0298606 ,  30.5702317 ,  28.60814055,
 27.94288232])
```

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 [113]:

```
# your turn
#your turn
plt.scatter(bos.PRICE,lm.predict(X))
plt.title("Price vs Predicted Prices")
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted Prices :$\hat{Y}_i$")
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 [114]:

```
import numpy as np
print (np.sum((bos.PRICE - lm.predict(X)) ** 2))
```

11080.276284149868

Mean squared error

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

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

In [115]:

```
#your turn
mse= np.mean((bos.PRICE -lm.predict(X))**2)
print(mse)
```

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 [116]:

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

Out[116]:

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

In [117]:

```
import numpy as np
msePTRATIO = np.mean((bos.PRICE - lm.predict(X[['PTRATIO']])) ** 2)
print (msePTRATIO)
```

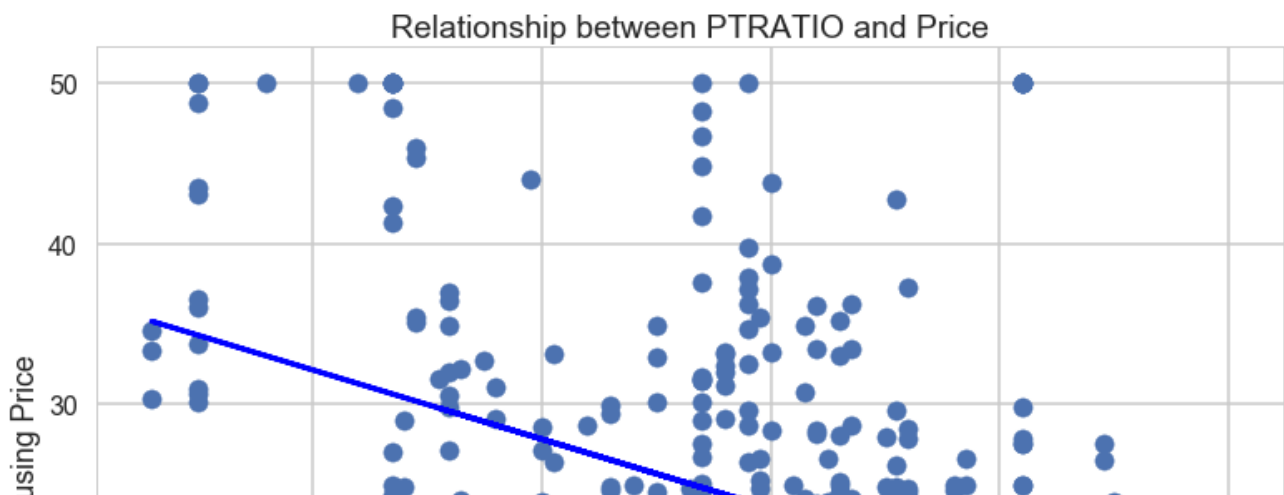
62.65220001376927

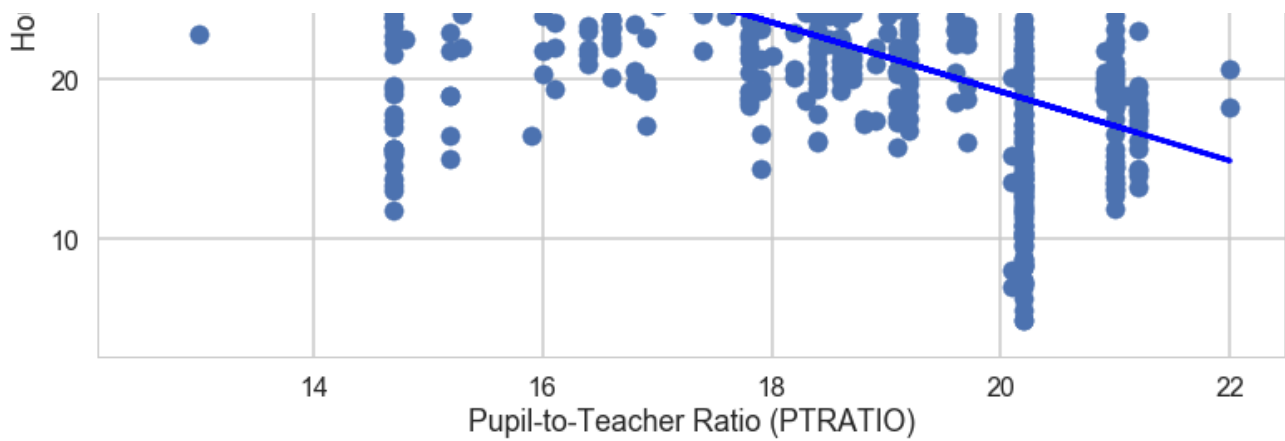
We can also plot the fitted linear regression line.

In [118]:

```
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 [119]:

```
# your turn
lmRM= LinearRegression()
lmRM.fit(X[['RM']], bos.PRICE)
```

Out[119]:

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

In [120]:

```
import numpy as np
mseRM= np.mean((bos.PRICE - lm.predict(X[['RM']])) ** 2)
print (mseRM)
```

795.3728404375607

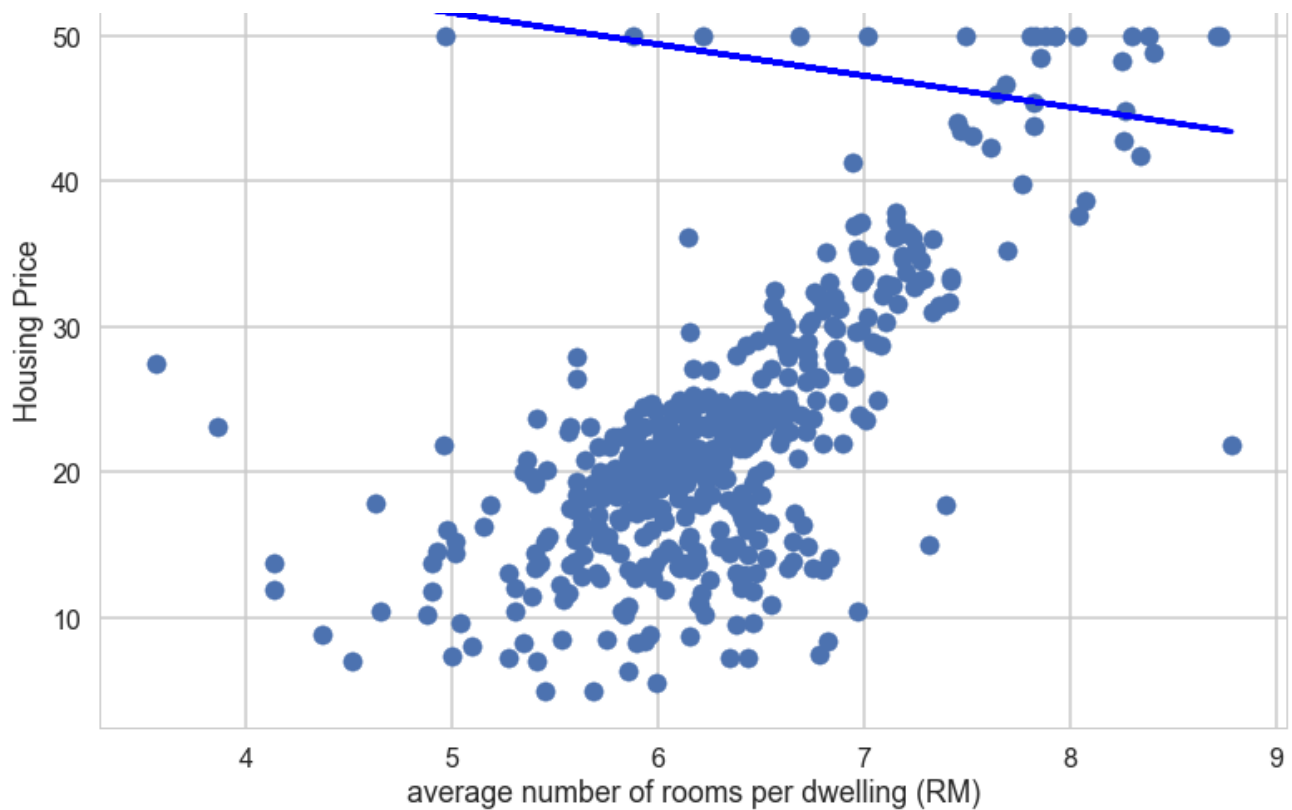
In [121]:

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

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

Relationship between RM and Price





In [122]:

```
lmCRIM= LinearRegression()
lmCRIM.fit(X[['RM']], bos.PRICE)
```

Out[122]:

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

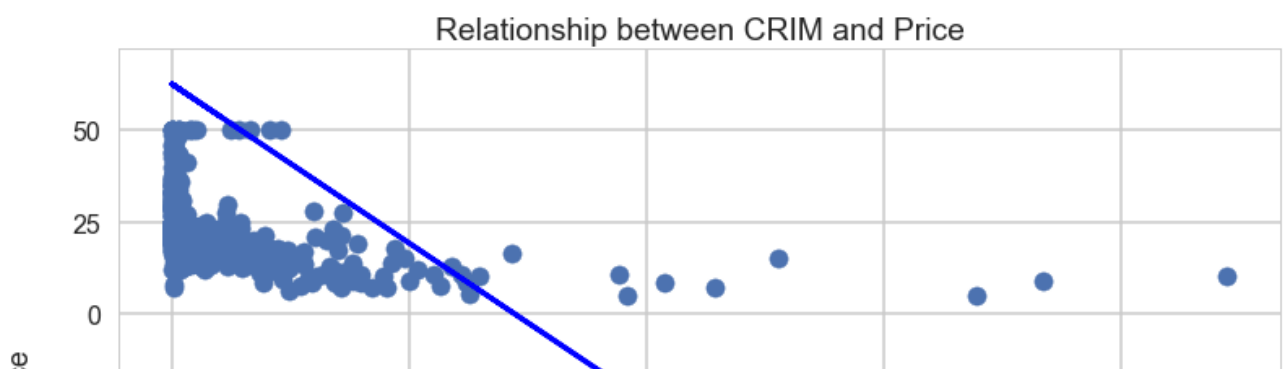
In [123]:

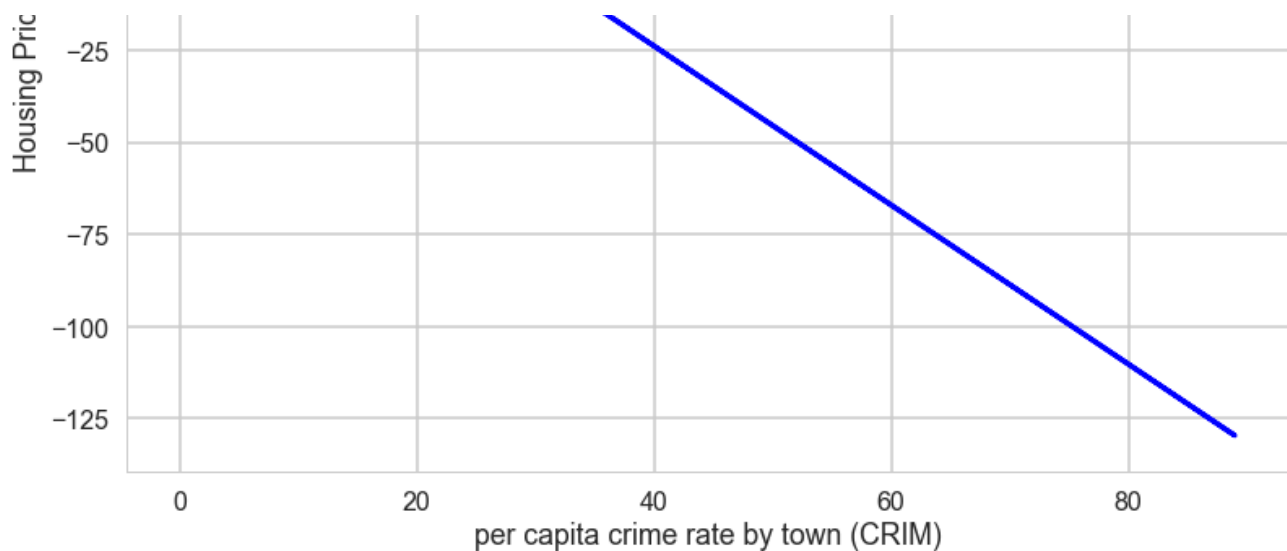
```
import numpy as np
mseCRIM= np.mean((bos.PRICE - lm.predict(X[['CRIM']])) ** 2)
print (mseCRIM)
```

1324.103195821284

In [124]:

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





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 [125]:

```
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](#).

In [126]:

```
from sklearn import cross_validation
X_train, X_test, Y_train, Y_test =
sklearn.cross_validation.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 [127]:

```
# your turn
lm = LinearRegression()
lm.fit(X_train,Y_train)
pred_train = lm.predict(X_train)
pred_test = lm.predict(X_test)
```

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 [128]:

```
# your turn
tr_mean = np.mean((Y_train - lm.predict(X_train))**2)
print ("mse for train set : " + str(tr_mean))

ts_mean = np.mean((Y_test - lm.predict(X_test))**2)
print ("mse for test set : " + str(ts_mean))
```

mse for train set : 19.54675847353466

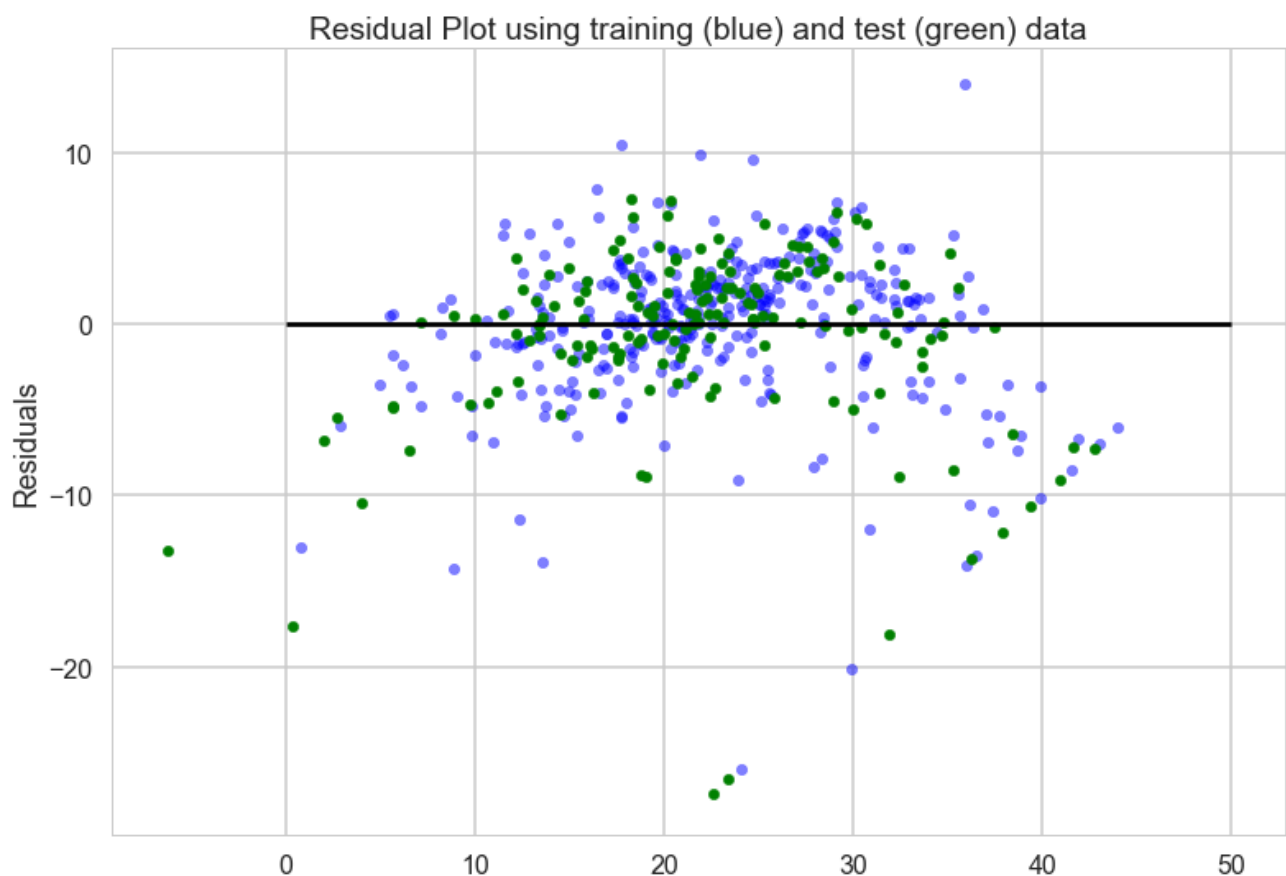
mse for test set : 28.541367275619013

model performs better on the train set with mse of 19.5 over mse of 28.5 on the test set.

Residual plots

In [129]:

```
plt.scatter(lm.predict(X_train), lm.predict(X_train) - Y_train, c='b', s=40,
            , alpha=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')
plt.show()
```



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

The residual plot shows a fairly random pattern, there is a good spread of positive and negative residuals on this plot. Lot of residuals are close to 0. This random pattern indicates that a linear model provides a decent fit to the data.

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, $\mathbb{1}(Y_i \neq \hat{Y}_i)$ for logistic regression).
- Calculate the average prediction error across all samples $\text{Err}_{\text{CV}} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$

Luckily you don't have to do this entire process all by hand (for loops, etc.) every single time, `scikit learn` has a very nice implementation of this, have a look at the [documentation](#).

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 [130]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [131]:

```
from sklearn import linear_model, cross_validation
import math
from sklearn import metrics

X_train = X[:-50]
X_test = X[-50:]
Y_train = bos.PRICE[:-50]
Y_test = bos.PRICE[-50:]

regr = linear_model.LinearRegression()
clf = regr.fit(X_train, Y_train)
scores = cross_validation.cross_val_score(clf, X, y, cv=4)
scores_mse = cross_validation.cross_val_score(clf, X, y, scoring =
'neg_mean_squared_error', cv=4)

print('Coefficients: \n', regr.coef_)
print("Residual sum of squares: %.2f \n" % np.mean((regr.predict(X_train) -
Y_train) ** 2))
```

```
#Explained variance score: 1 is perfect prediction
print('Variance score: %.2f \n' % regr.score(X_train, Y_train))

print(scores)

print(scores_mse)
```

Coefficients:

```
[ -1.04755725e-01  4.91233643e-02  3.24299720e-02  2.51517135e+00
 -1.76585750e+01  3.81259444e+00  1.06438518e-02 -1.43651798e+00
  3.60959247e-01 -1.54635990e-02 -9.13025678e-01  9.94705988e-03
 -5.55769911e-01]
```

Residual sum of squares: 23.19

Variance score: 0.74

```
[ 0.60157044  0.60368748  0.35870794 -1.12403993]
[-11.69977729 -39.0720141  -57.59307099 -61.59301573]
```