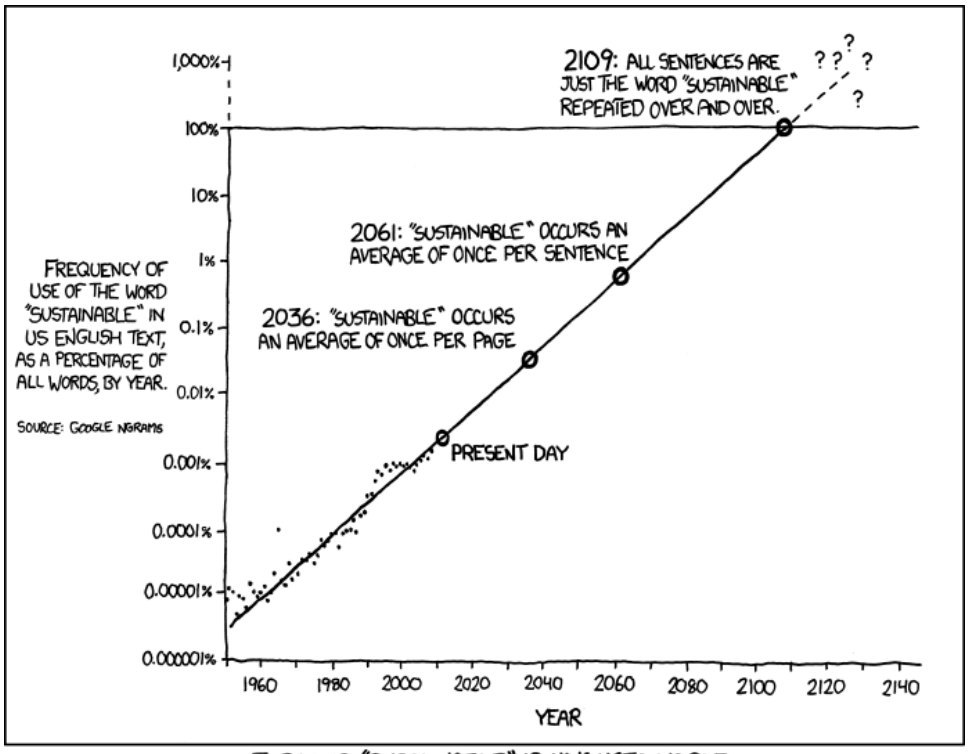
Regression in Python

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

Linear regression is used to model and predict continuous outcomes with normal random errors. There are nearly an infinite number of different types of regression models and each regression model is typically defined by the distribution of the prediction errors (called "residuals") of the type of data. Logistic regression is used to model binary outcomes whereas Poisson regression is used to predict counts. In this exercise, 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.



THE WORD "SUSTAINABLE" IS UNSUSTAINABLE.

In [2]:

```
# special IPython command to prepare the notebook for matplotlib and other libraries
%matplotlib 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")
```

Part 1: Introduction to Linear Regression

Purpose of linear regression

Given a dataset containing predictor variables X and outcome/response variable Y, linear regression can be used to:

- Build a **predictive model** to predict future values of $\hat{Y}$, using new data X^* where Y is unknown.
- Model the **strength of the relationship** between each independent variable X_i and Y
  - Many times, only a subset of independent variables X_i will have a linear relationship with Y
  - Need to figure out which X_i contributes most information to predict Y
- It is in many cases, the first pass prediction algorithm for continuous outcomes.

A Brief Mathematical Recap

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. The most basic linear regression model contains one independent variable X, we'll call this the simple model.
$$ Y = \beta_0 + \beta_1 X + \epsilon$$

where ϵ is considered as an unobservable random variable that adds noise to the linear relationship. In linear regression, ϵ is assumed to be normally distributed with a mean of 0. In other words, what this means is that on average, if we know Y, a roughly equal number of predictions $\hat{Y}$ will be above Y and others will be below Y. That is, on average, the error is zero. The residuals, ϵ are also assumed to be "i.i.d.": independently and identically distributed. Independence means that the residuals are not correlated -- the residual from one prediction has no effect on the residual from another prediction. Correlated errors are common in time series analysis and spatial analyses.

- β_0 is the intercept of the linear model and represents the average of Y when all independent variables X are set to 0.
- β_1 is the slope of the line associated with the regression model and represents the average effect of a one-unit increase in X on Y.
- 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.

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

Image from http://www.learner.org/courses/againstallodds/about/glossary.html. Note this image uses α and β instead of β_0 and β_1.

- 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 we estimate the coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$, we can use these to predict new values of Y given new data X.

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

- Multiple linear regression is when you have more than one independent variable and the estimation involves matrices
  - X_1, X_2, X_3, $\ldots$
- How do you estimate the coefficients?
  - There are many ways to fit a linear regression model
  - The method called **least squares** is the most common methods
  - We will discuss least squares

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

Estimating $\hat\beta$: Least squares

Least squares is a method that can estimate the coefficients of a linear model by minimizing the squared residuals:
$$ \mathscr{L} = \sum_{i=1}^N \epsilon_i^2 = \sum_{i=1}^N \left(y_i - \hat{y}_i \right)^2 = \sum_{i=1}^N \left(y_i - \left(\beta_0 + \beta_1 x_i\right)\right)^2 $$

where N is the number of observations and ϵ represents a residual or error, ACTUAL - PREDICTED.

Estimating the intercept $\hat{\beta_0}$ for the simple linear model

We want to minimize the squared residuals and solve for $\hat{\beta_0}$ so we take the partial derivative of $\mathscr{L}$ with respect to $\hat{\beta_0}$

$ \begin{align} \frac{\partial \mathscr{L}}{\partial \hat{\beta_0}} &= \frac{\partial}{\partial \hat{\beta_0}} \sum_{i=1}^N \epsilon^2 \\ &= \frac{\partial}{\partial \hat{\beta_0}} \sum_{i=1}^N \left(y_i - \hat{y}_i \right)^2 \\ &= \frac{\partial}{\partial \hat{\beta_0}} \sum_{i=1}^N \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right)^2 \\ &= -2 \sum_{i=1}^N \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right) \hspace{25mm} \mbox{(by chain rule)} \\ &= -2 \sum_{i=1}^N (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) \\ &= -2 \left[\left(\sum_{i=1}^N y_i \right) - N \hat{\beta_0} - \hat{\beta}_1 \left(\sum_{i=1}^N x_i \right) \right] \\ & 2 \left[N \hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^N x_i - \sum_{i=1}^N y_i \right] = 0 \hspace{20mm} \mbox{(Set equal to 0 and solve for $\hat{\beta}_0$)} \\ & N \hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^N x_i - \sum_{i=1}^N y_i = 0 \\ & N \hat{\beta}_0 = \sum_{i=1}^N y_i - \hat{\beta}_1 \sum_{i=1}^N x_i \\ & \hat{\beta}_0 = \frac{\sum_{i=1}^N y_i - \hat{\beta}_1 \sum_{i=1}^N x_i}{N} \\ & \hat{\beta}_0 = \frac{\sum_{i=1}^N y_i}{N} - \hat{\beta}_1 \frac{\sum_{i=1}^N x_i}{N} \\ & \boxed{\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}} \end{align} $

Using this new information, we can compute the estimate for $\hat{\beta}_1$ by taking the partial derivative of $\mathscr{L}$ with respect to $\hat{\beta}_1$.

$ \begin{align} \frac{\partial \mathscr{L}}{\partial \hat{\beta_1}} &= \frac{\partial}{\partial \hat{\beta_1}} \sum_{i=1}^N \epsilon^2 \\ &= \frac{\partial}{\partial \hat{\beta_1}} \sum_{i=1}^N \left(y_i - \hat{y}_i \right)^2 \\ &= \frac{\partial}{\partial \hat{\beta_1}} \sum_{i=1}^N \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right)^2 \\ &= 2 \sum_{i=1}^N \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right) \left(-x_i \right) \hspace{25mm}\mbox{(by chain rule)} \\ &= -2 \sum_{i=1}^N x_i \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i \right) \\ &= -2 \sum_{i=1}^N x_i (y_i - \hat{\beta}_0 x_i - \hat{\beta}_1 x_i^2) \\ &= -2 \sum_{i=1}^N x_i (y_i - \left(\bar{y} - \hat{\beta}_1 \bar{x} \right) x_i - \hat{\beta}_1 x_i^2) \\ &= -2 \sum_{i=1}^N (x_i y_i - \bar{y}x_i + \hat{\beta}_1\bar{x}x_i - \hat{\beta}_1 x_i^2) \\ &= -2 \left[\sum_{i=1}^N x_i y_i - \bar{y} \sum_{i=1}^N x_i + \hat{\beta}_1\bar{x}\sum_{i=1}^N x_i - \hat{\beta}_1 \sum_{i=1}^N x_i^2 \right] \\ &= -2 \left[\hat{\beta}_1 \left\{ \bar{x} \sum_{i=1}^N x_i - \sum_{i=1}^N x_i^2 \right\} + \left\{ \sum_{i=1}^N x_i y_i - \bar{y} \sum_{i=1}^N x_i \right\}\right] \\ & 2 \left[\hat{\beta}_1 \left\{ \sum_{i=1}^N x_i^2 - \bar{x} \sum_{i=1}^N x_i \right\} + \left\{ \bar{y} \sum_{i=1}^N x_i - \sum_{i=1}^N x_i y_i \right\} \right] = 0 \\ & \hat{\beta}_1 = \frac{-\left(\bar{y} \sum_{i=1}^N x_i - \sum_{i=1}^N x_i y_i \right)}{\sum_{i=1}^N x_i^2 - \bar{x}\sum_{i=1}^N x_i} \\ &= \frac{\sum_{i=1}^N x_i y_i - \bar{y} \sum_{i=1}^N x_i}{\sum_{i=1}^N x_i^2 - \bar{x} \sum_{i=1}^N x_i} \\ & \boxed{\hat{\beta}_1 = \frac{\sum_{i=1}^N x_i y_i - \bar{x}\bar{y}n}{\sum_{i=1}^N x_i^2 - n \bar{x}^2}} \end{align} $

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. Basically, $X^T X$ is full rank if all rows and columns are linearly independent. This has a loose relationship to variables and observations being independent respective. **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.

Part 2: Exploratory Data Analysis for Linear Relationships

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 [3]:

```
from sklearn.datasets import load_boston
```

```
import pandas as pd

boston = load_boston()
```

In [4]:

```
boston.keys()
```

Out[4]:

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

In [5]:

```
boston.data.shape
```

Out[5]:

```
(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 [7]:

```
# Print description of Boston housing data set
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**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.
https://archive.ics.uci.edu/ml/machine-learning-databases/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
```

```
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.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of C
ollinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the T
enth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst.
Morgan Kaufmann.
```

Now let's explore the data set itself.

In [8]:

```
bos = pd.DataFrame(boston.data)
bos.head()
```

Out[8]:

	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 [9]:

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

Out[9]:

	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

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 [10]:

```
print(boston.target.shape)
```

```
(506,)
```

In [11]:

```
bos['PRICE'] = boston.target
```

```
bos.head()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

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 [12]:

```
bos.describe()
```

Out[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTR
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.45
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.16
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.60
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.40
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.05
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.20
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.00

Scatterplots

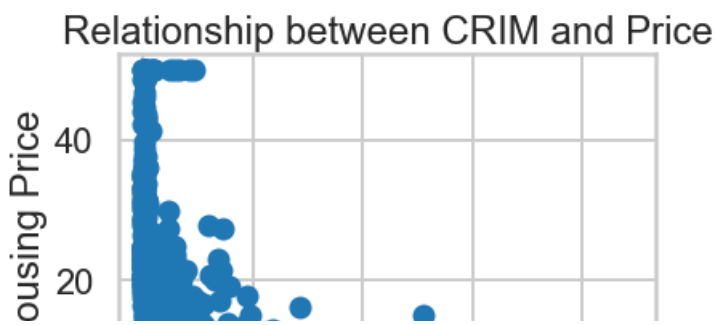
Let's look at some scatter plots for three variables: 'CRIM' (per capita crime rate), 'RM' (number of rooms) and 'PTRATIO' (pupil-to-teacher ratio in schools).

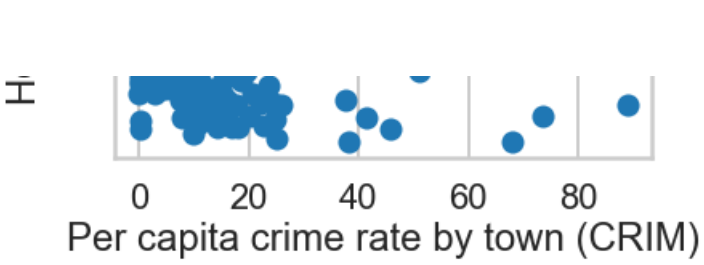
In [13]:

```
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[13]:

```
Text(0.5, 1.0, 'Relationship between CRIM and Price')
```





Part 2 Checkup Exercise Set I

Exercise: What kind of relationship do you see? e.g. positive, negative? linear? non-linear? Is there anything else strange or interesting about the data? What about outliers?

Exercise: Create scatter plots between *RM* and *PRICE*, and *PTRATIO* and *PRICE*. Label your axes appropriately using human readable labels. Tell a story about what you see.

Exercise: What are some other numeric variables of interest? Why do you think they are interesting? Plot scatterplots with these variables and *PRICE* (house price) and tell a story about what you see.

In [14]:

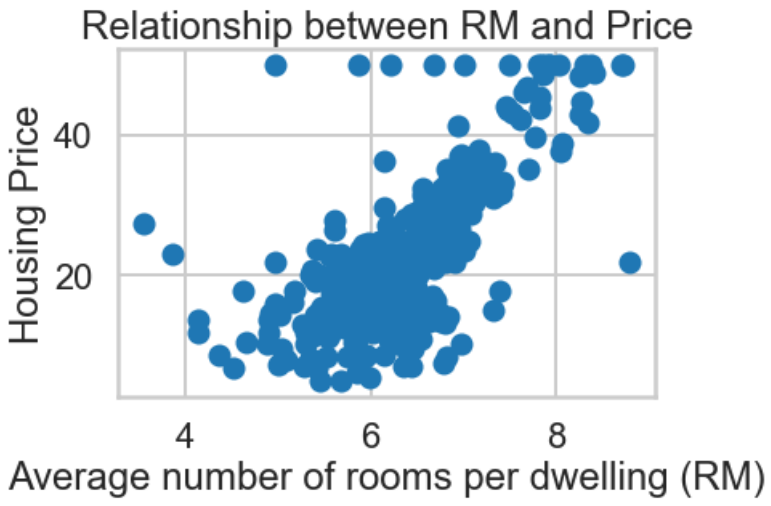
```
# your turn: describe relationship
```

In [15]:

```
# 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[15]:

```
Text(0.5, 1.0, 'Relationship between RM and Price')
```



In [ ]:

```

```

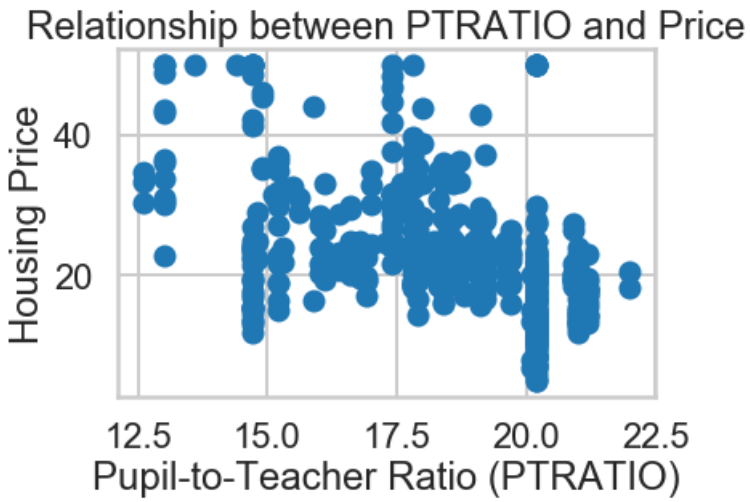
In [16]:

```
# your turn: scatter plot between *PTRATIO* and *PRICE*
plt.scatter(bos.PTRATIO, bos.PRICE)
plt.xlabel("Pupil-to-Teacher Ratio (PTRATIO)")
plt.ylabel("Housing Price")
plt.title("Relationship between PTRATIO and Price")
```

Out[16]:

```
Text(0.5, 1.0, 'Relationship between PTRATIO and Price')
```

Text(0.5, 1.0, 'Relationship between PTRATIO and Price')

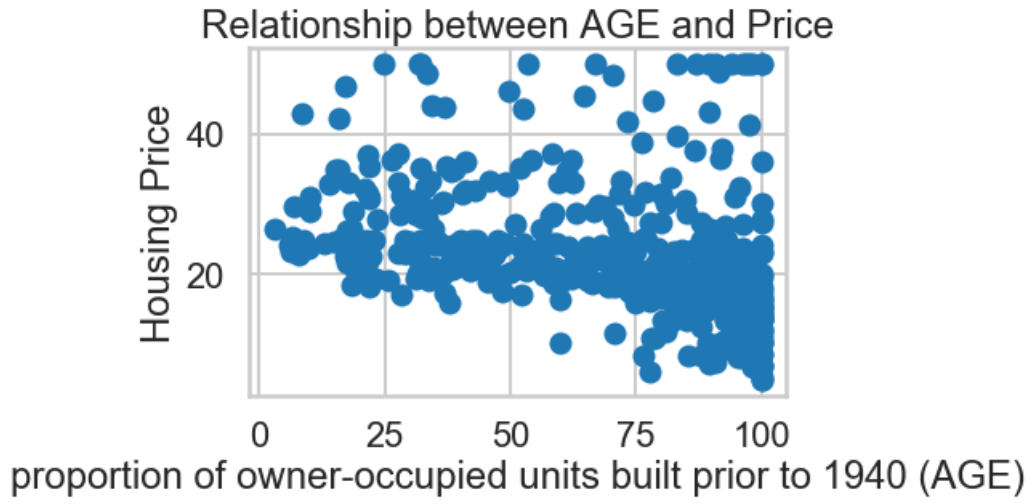


In [17]:

```
# your turn: create scatter plots between "AGE" and "PRICE"
plt.scatter(bos.AGE, bos.PRICE)
plt.xlabel("proportion of owner-occupied units built prior to 1940 (AGE)")
plt.ylabel("Housing Price")
plt.title("Relationship between AGE and Price")
```

Out[17]:

```
Text(0.5, 1.0, 'Relationship between AGE and Price')
```

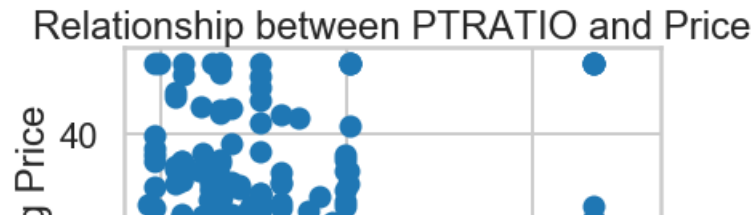


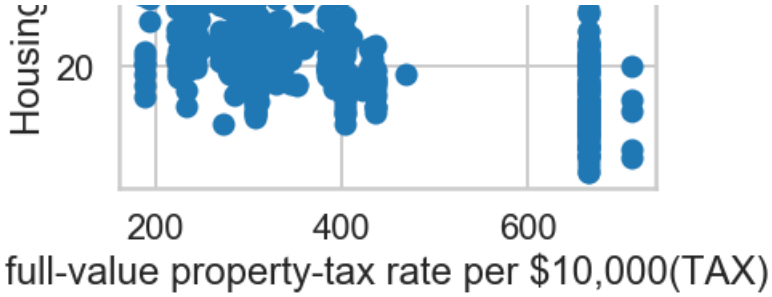
In [18]:

```
# your turn: scatter plot between *TAX* and *PRICE*
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 PTRATIO and Price")
```

Out[18]:

```
Text(0.5, 1.0, 'Relationship between PTRATIO and Price')
```





In [ ]:

```
```

In [ ]:

```
```

Scatterplots 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.

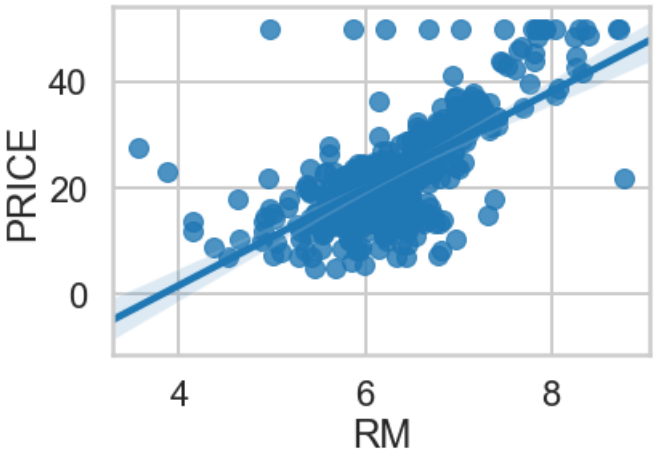
We can also use seaborn regplot for the scatterplot above. This provides automatic linear regression fits (useful for data exploration later on). Here's one example below.

In [19]:

```
sns.regplot(y="PRICE", x="RM", data=bos, fit_reg = True)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x206dead8278>
```

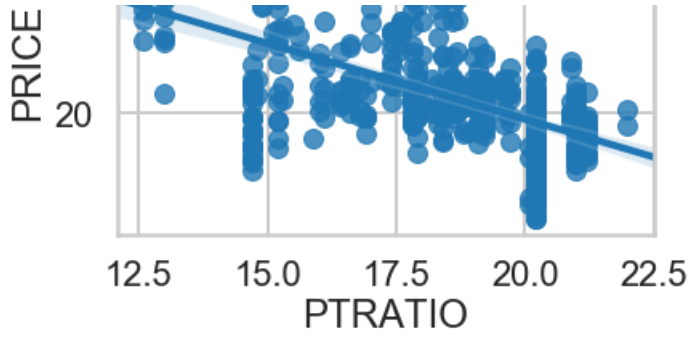


In [20]:

```
sns.regplot(y="PRICE", x="PTRATIO", data=bos, fit_reg = True)
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x206deb28898>
```

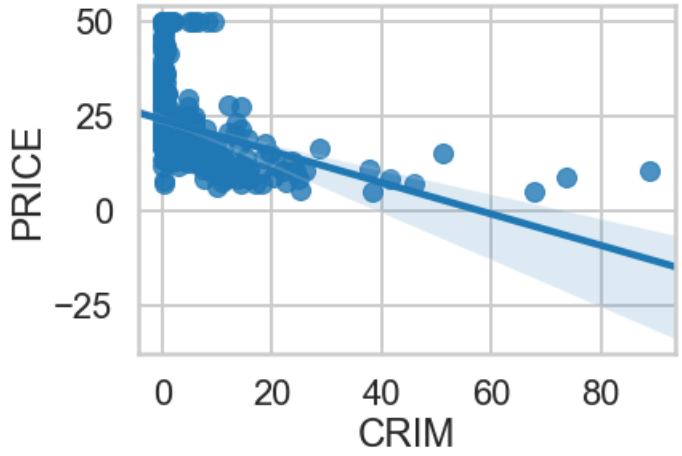


In [21]:

```
sns.regplot(y="PRICE", x="CRIM", data=bos, fit_reg = True)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x206dea78a58>
```



In [ ]:

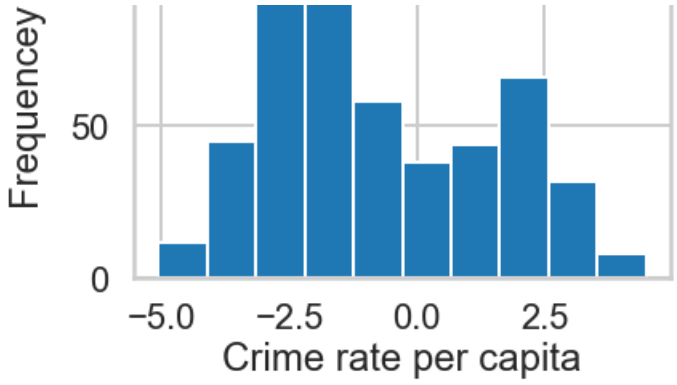
In [ ]:

In [ ]:

Histograms

In [22]:

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



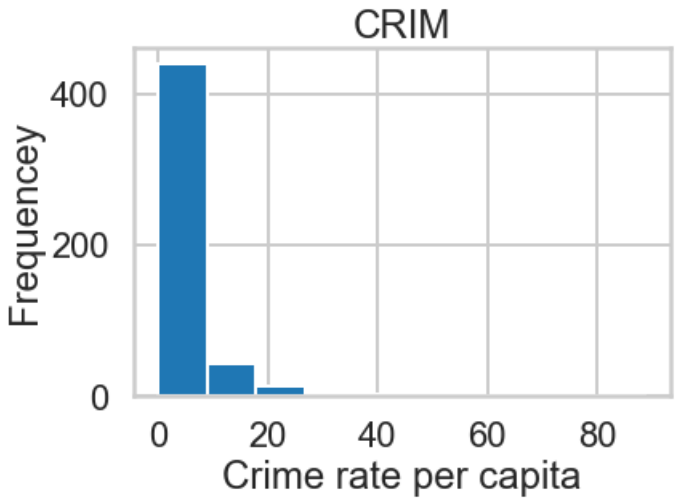
Part 2 Checkup Exercise Set II

Exercise: In the above histogram, we took the logarithm of the crime rate per capita. Repeat this histogram without taking the log. What was the purpose of taking the log? What do we gain by making this transformation? What do you now notice about this variable that is not obvious without making the transformation?

Exercise: Plot the histogram for *RM* and *PTRATIO* against each other, along with the two variables you picked in the previous section. We are looking for correlations in predictors here.

In [23]:

```
#your turn

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

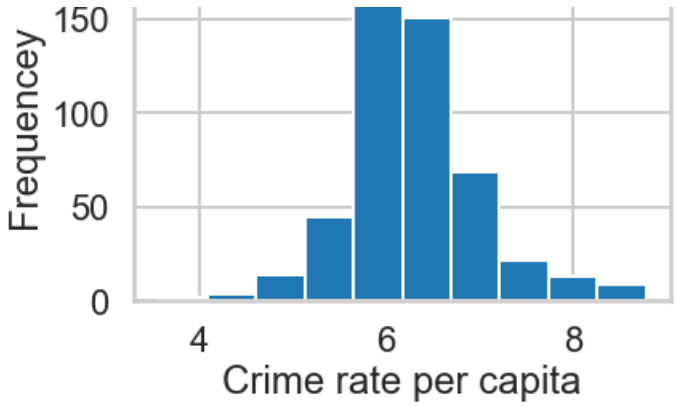


In [ ]:

```

```

In [24]:

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

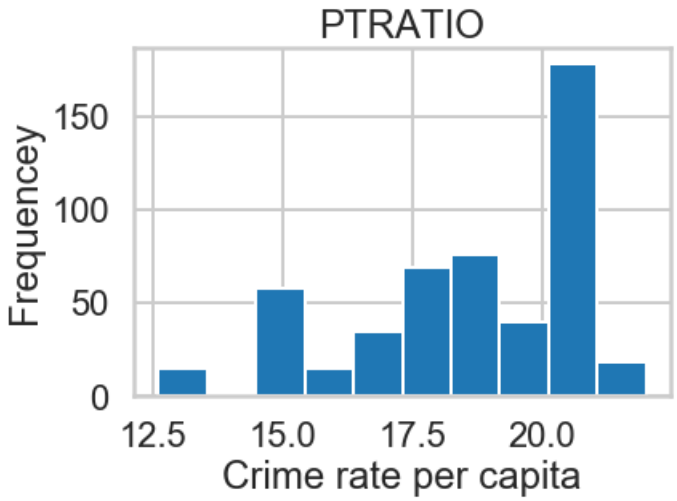


In [ ]:

```
```

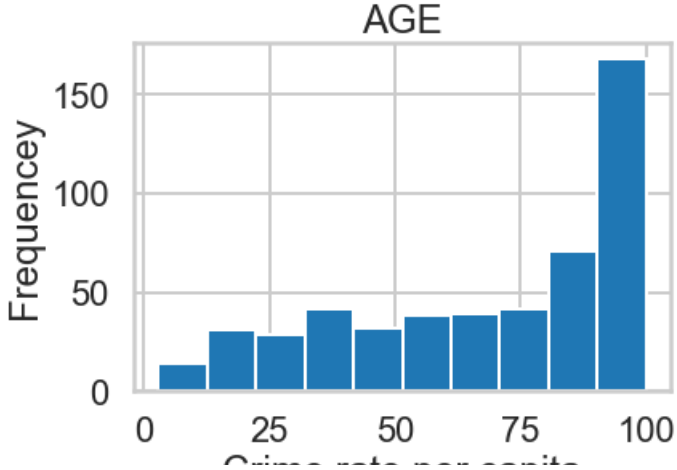
In [25]:

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



In [26]:

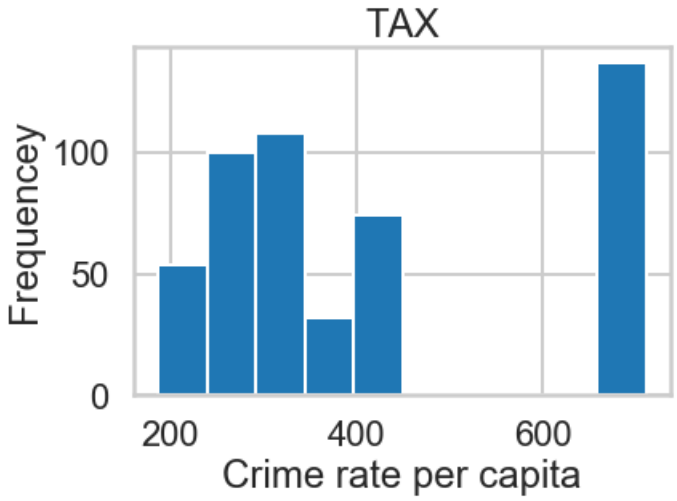
```
plt.hist(bos.AGE)
plt.title("AGE")
plt.xlabel("Crime rate per capita")
plt.ylabel("Frequencey")
plt.show()
```



Crime rate per capita

In [27]:

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



In [ ]:

In [ ]:

In [ ]:

Part 3: Linear Regression with Boston Housing Data Example

Here,

Y = boston housing prices (called "target" data in python, and referred to as the dependent variable or response variable)

and

X = all the other features (or independent variables, predictors or explanatory variables)

which we will use to fit a linear regression model and predict Boston housing prices. We will use the least-squares method 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 workshop, it's a good library to have in your toolbox. Here's a quick example of what you could do with it. The version of least-squares we will use in statsmodels is called *ordinary least-squares (OLS)*. There are many other versions of least-squares such as partial least squares (PLS) and weighted least squares (WLS).

In [28]:

```
# Import regression modules
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [82]:

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

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  PRICE   R-squared:                       0.594
Model:                            OLS   Adj. R-squared:                  0.592
Method:                 Least Squares   F-statistic:                     245.2
Date:                Mon, 18 Nov 2019   Prob (F-statistic):           6.15e-98
Time:                        23:18:15   Log-Likelihood:                -1612.0
No. Observations:                 506   AIC:                             3232.
Df Residuals:                     502   BIC:                             3249.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -3.3707      4.034     -0.836      0.404     -11.296       4.555
RM             7.3804      0.402     18.382      0.000       6.592       8.169
PTRATIO       -1.0695      0.133     -8.051      0.000      -1.331      -0.809
CRIM          -0.2050      0.032     -6.399      0.000      -0.268      -0.142
==============================================================================
Omnibus:                      234.656   Durbin-Watson:                   0.830
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2020.689
Skew:                           1.815   Prob(JB):                         0.00
Kurtosis:                      12.092   Cond. No.                         311.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly 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. This means that the number of rooms, `RM`, is a statisticall significant predictor of `PRICE`. The regression coefficient for `RM` of 9.1021 means that *on average, each additional room is associated with an increase of $\$9,100$ in house price net of the other variables*. The confidence interval gives us a range of plausible values for this average change, about ($\$8,279, \$9,925$), definitely not chump change.

In general, the $\hat{\beta_i}, i > 0$ can be interpreted as the following: "A one unit increase in x_i is associated with, on average, a $\hat{\beta_i}$ increase/decrease in y net of all other variables."

On the other hand, the interpretation for the intercept, $\hat{\beta}_0$ is the average of y given that all of the independent variables x_i are 0.

`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 to be able to handle more complex models, for example multiple regression is doone like this:

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

In general, a formula for an OLS multiple linear regression is

```
Y ~ X1 + X2 + ... + Xp
```

This is the very basic structure but it should be enough to get you through the homework. Things can get much more complex. You can force statsmodels to treat variables as categorical with the `C()` function, call numpy functions to transform data such as `np.log` for extremely-skewed data, or fit a model without an intercept by including `- 1` in the formula. 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.

Part 3 Checkup Exercise Set I

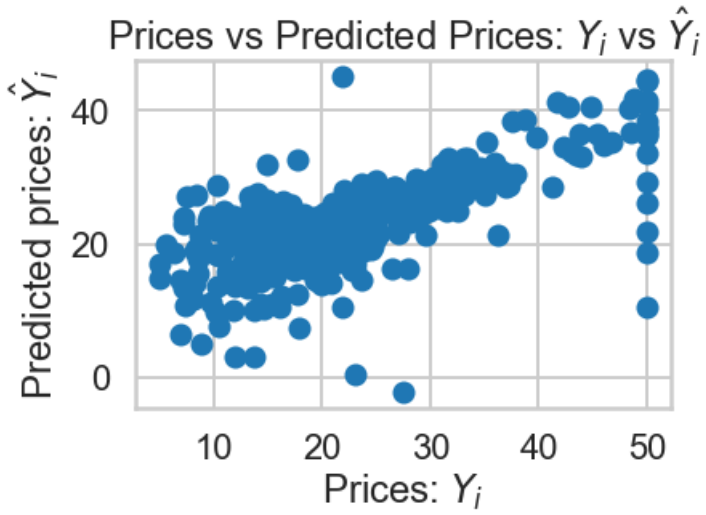
Exercise: Create a scatterplot between the predicted prices, available in `m.fittedvalues` (where `m` is the fitted model) and the original prices. How does the plot look? Do you notice anything interesting or weird in the plot? Comment on what you see.

In [30]:

```
# your turn
plt.scatter(bos['PRICE'], m.fittedvalues)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted Prices: $Y_i$ vs $\hat{Y}_i$")
```

Out[30]:

```
Text(0.5, 1.0, 'Prices vs Predicted Prices: $Y_i$ vs $\\hat{Y}_i$')
```



Fitting Linear Regression using `sklearn`

In [31]:

```
from sklearn.linear_model import LinearRegression
X = bos.drop('PRICE', axis = 1)

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

Out[31]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

normalize=False)

What can you do with a LinearRegression object?

Check out the scikit-learn docs here. We have listed the main functions here. Most machine learning models in scikit-learn follow this same API of fitting a model with `fit`, making predictions with `predict` and the appropriate scoring function `score` for each model.

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 [32]:

```
# Look inside lm object
# lm.<tab>
```

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 [33]:

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

Out[33]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Part 3 Checkup Exercise Set II

Exercise: How would you change the model to not fit an intercept term? Would you recommend not having an intercept? Why or why not? For more information on why to include or exclude an intercept, look [here](https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faq-what-is-regression-through-the-origin/).

Exercise: One of the assumptions of the linear model is that the residuals must be i.i.d. (independently and identically distributed). To satisfy this, is it enough that the residuals are normally distributed? Explain your answer.

Exercise: True or false. To use linear regression, Y must be normally distributed. Explain your answer.

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 [35]:

```
print('Estimated intercept coefficient: {}'.format(lm.intercept_))
```

```
Estimated intercept coefficient: 36.45948838509015
```

In [36]:

```
print('Number of coefficients: {}'.format(len(lm.coef_)))
```

```
Number of coefficients: 13
```

In [37]:

```
# The coefficients
pd.DataFrame({'features': X.columns, 'estimatedCoefficients': lm.coef_})[['features',
'estimatedCoefficients']]
```

Out[37]:

	features	estimatedCoefficients
0	CRIM	-0.108011
1	ZN	0.046420
2	INDUS	0.020559
3	CHAS	2.686734
4	NOX	-17.766611
5	RM	3.809865
6	AGE	0.000692
7	DIS	-1.475567
8	RAD	0.306049
9	TAX	-0.012335
10	PTRATIO	-0.952747
11	B	0.009312
12	LSTAT	-0.524758

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 [38]:

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

Out[38]:

```
array([30.00384338, 25.02556238, 30.56759672, 28.60703649, 27.94352423])
```

Part 3 Checkup Exercise Set III

Exercise: Histogram: Plot a histogram of all the predicted prices. Write a story about what you see. Describe the shape, center and spread of the distribution. Are there any outliers? What might be the reason for them? Should we do anything special with them?

Exercise: Scatterplot: Let's plot the true prices compared to the predicted prices to see they disagree (we did this with `statsmodels` before).

Exercise: We have looked at fitting a linear model in both `statsmodels` and `scikit-learn`. What are the advantages and disadvantages of each based on your exploration? Based on the information provided by both packages, what advantage does

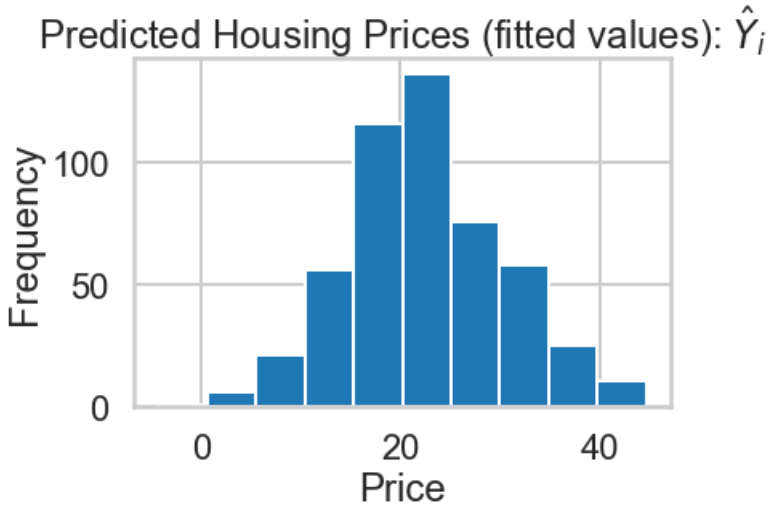
disadvantages of each based on your exploration? Based on the information provided by both packages, what advantage does `statsmodels` provide?

In [39]:

```
# your turn

plt.hist(lm.predict(X))
plt.title('Predicted Housing Prices (fitted values): $\hat{Y}_i$')
plt.xlabel('Price')
plt.ylabel('Frequency')
```

Out[39]:

```
Text(0, 0.5, 'Frequency')
```



In [ ]:

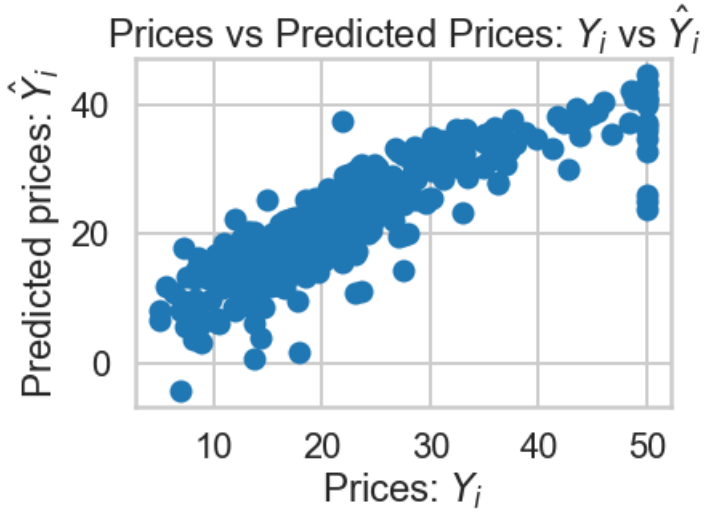
```

```

In [40]:

```
plt.scatter(bos.PRICE, lm.predict(X))
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted Prices: $Y_i$ vs $\hat{Y}_i$")
```

Out[40]:

```
Text(0.5, 1.0, 'Prices vs Predicted Prices: $Y_i$ vs $\\hat{Y}_i$')
```



In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

Evaluating the Model: Sum-of-Squares

The partitioning of the sum-of-squares shows the variance in the predictions explained by the model and the variance that is attributed to error.
$$TSS = ESS + RSS$$

Residual Sum-of-Squares (aka RSS)

The residual sum-of-squares is one of the basic ways of quantifying how much error exists in the fitted model. We will revisit this in a bit.
$$ RSS = \sum_{i=1}^N r_i^2 = \sum_{i=1}^N \left(y_i - \left(\beta_0 + \beta_1 x_i\right)\right)^2 $$

In [41]:

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

```
11078.784577954977
```

Explained Sum-of-Squares (aka ESS)

The explained sum-of-squares measures the variance explained by the regression model.
$$ESS = \sum_{i=1}^N \left(\hat{y}_i - \bar{y} \right)^2 = \sum_{i=1}^N \left(\left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) - \bar{y} \right)^2$$

In [42]:

```
print(np.sum((lm.predict(X) - np.mean(bos.PRICE)) ** 2))
```

```
31637.510837065056
```

Evaluating the Model: The Coefficient of Determination (R^2)

The coefficient of determination, R^2, tells us the percentage of the variance in the response variable Y that can be explained by the linear regression model.
$$ R^2 = \frac{ESS}{TSS} $$

The R^2 value is one of the most common metrics that people use in describing the quality of a model, but it is important to note that *R^2 increases artificially as a side-effect of increasing the number of independent variables.* While R^2 is reported in almost

that R^2 *increases artificially as a side-effect of increasing the number of independent variables.* While R^2 is reported in almost all statistical packages, another metric called the *adjusted R^2* is also provided as it takes into account the number of variables in the model, and can sometimes even be used for non-linear regression models!
$$R_{adj}^2 = 1 - \left(1 - R^2 \right) \frac{N - 1}{N - K - 1} = R^2 - \left(1 - R^2 \right) \frac{K}{N - K - 1} = 1 - \frac{\frac{RSS}{DF_R}}{\frac{TSS}{DF_T}}$$

where N is the number of observations, K is the number of variables, $DF_R = N - K - 1$ is the degrees of freedom associated with the residual error and $DF_T = N - 1$ is the degrees of the freedom of the total error.

Evaluating the Model: Mean Squared Error and the F-Statistic

The mean squared errors are just the *averages* of the sum-of-squares errors over their respective degrees of freedom.
$$MSE = \frac{RSS}{N-K-1}$$$$MSR = \frac{ESS}{K}$$

Remember: Notation may vary across resources particularly the use of R and E in RSS/ESS and MSR/MSE. In some resources, E = explained and R = residual. In other resources, E = error and R = regression (explained). **This is a very important distinction that requires looking at the formula to determine which naming scheme is being used.**

Given the MSR and MSE, we can now determine whether or not the entire model we just fit is even statistically significant. We use an F-test for this. The null hypothesis is that all of the β coefficients are zero, that is, none of them have any effect on Y. The alternative is that *at least one* β coefficient is nonzero, but it doesn't tell us which one in a multiple regression:
$$H_0: \beta_i = 0, \mbox{for all i} \\ H_A: \beta_i > 0, \mbox{for some i}$$

$$F = \frac{MSR}{MSE} = \left(\frac{R^2}{1 - R^2} \right) \left(\frac{N - K - 1}{K} \right)$$

Once we compute the F-statistic, we can use the F-distribution with $N-K$ and $K-1$ degrees of degrees of freedom to get a p-value.

Warning! The F-statistic mentioned in this section is NOT the same as the F1-measure or F1-value discused in Unit 7.

Part 3 Checkup Exercise Set IV

Let's look at the relationship between `PTRATIO` and housing price.

Exercise: Try fitting a linear regression model using only the 'PTRATIO' (pupil-teacher ratio by town) and interpret the intercept and the coefficients.

Exercise: Calculate (or extract) the R^2 value. What does it tell you?

Exercise: Compute the F-statistic. What does it tell you?

Exercise: Take a close look at the F-statistic and the t-statistic for the regression coefficient. What relationship do you notice? Note that this relationship only applies in *simple* linear regression models.

In [43]:

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

Out[43]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [44]:

```
rss = np.sum((bos.PRICE - lm.predict(X[['PTRATIO']])) ** 2)
print(rss)
ess = np.sum((lm.predict(X[['PTRATIO']]) - np.mean(bos.PRICE)) ** 2)
print(ess)
tss = ess+rss
print(tss)
```

```
31702.013206967255
11014.282208052502
42716.295415019755
```

In [45]:

```
Rsquare = (ess/tss)
print(Rsquare)
```

```
0.2578473180092227
```

In [ ]:

```

```

In [46]:

```
mse = (rss/(506-1-1))
msr = (ess/1)
f = msr/mse
f
```

Out[46]:

```
175.10554287569522
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
```

Part 3 Checkup Exercise Set V

Fit 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)

Exercise: Compute or extract the F-statistic. What does it tell you about the model?

Exercise: Compute or extract the R^2 statistic. What does it tell you about the model?

Exercise: Which variables in the model are significant in predicting house price? Write a story that interprets the coefficients.

In [47]:

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

Out[47]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [48]:

```
rss = np.sum((bos.PRICE - lm.predict(X[['CRIM']])) ** 2)
print(rss)
ess = np.sum((lm.predict(X[['CRIM']]) - np.mean(bos.PRICE)) ** 2)
print(ess)
tss = ess+rss
print(tss)
```

```
36275.512356275096
6440.783058744659
42716.295415019755
```

In [49]:

```
Rsquare = (ess/tss)
print(Rsquare)
```

```
0.1507804690497569
```

In [50]:

```
mse = (rss/(506-1-1))
msr = (ess/1)
f = msr/mse
f
```

Out[50]:

```
89.486114757681
```

In [51]:

```
lm = LinearRegression()
lm.fit(X[['RM']], bos.PRICE)
rss = np.sum((bos.PRICE - lm.predict(X[['RM']])) ** 2)
```

```
print(rss)
ess = np.sum((lm.predict(X[['RM']]) - np.mean(bos.PRICE)) ** 2)
print(ess)
tss = ess+rss
print(tss)
Rsquare = (ess/tss)
print(Rsquare)
mse = (rss/(506-1-1))
msr = (ess/1)
f = msr/mse
f
```

```
22061.879196211798
20654.416218807964
42716.29541501976
0.4835254559913341
```

Out[51]:

```
471.8467398763866
```

In [52]:

```
lm = LinearRegression()
lm.fit(X[['RM']], bos.PRICE)
rss = np.sum((bos.PRICE - lm.predict(X[['RM']])) ** 2)
print(rss)
ess = np.sum((lm.predict(X[['RM']]) - np.mean(bos.PRICE)) ** 2)
print(ess)
tss = ess+rss
print(tss)
Rsquare = (ess/tss)
print(Rsquare)
mse = (rss/(506-1-1))
msr = (ess/1)
f = msr/mse
f
```

```
22061.879196211798
20654.416218807964
42716.29541501976
0.4835254559913341
```

Out[52]:

```
471.8467398763866
```

In [ ]:

```

```

Part 4: Comparing Models

During modeling, there will be times when we want to compare models to see which one is more predictive or fits the data better. There are many ways to compare models, but we will focus on two.

The F-Statistic Revisited

The F-statistic can also be used to compare two *nested* models, that is, two models trained on the same dataset where one of the models contains a *subset* of the variables of the other model. The *full* model contains K variables and the *reduced* model contains a subset of these K variables. This allows us to add additional variables to a base model and then test if adding the variables helped the model fit.
$$F = \frac{\left(\frac{RSS_{reduced} - RSS_{full}}{DF_{reduced} - DF_{full}} \right)}{\left(\frac{RSS_{full}}{DF_{full}} \right)}$$

where $DF_x = N - K_x - 1$ where K_x is the number of variables in model x.

Akaike Information Criterion (AIC)

Akaike Information Criterion (AIC)

Another statistic for comparing two models is AIC, which is based on the likelihood function and takes into account the number of variables in the model.
$$AIC = 2 K - 2 \log_e{L}$$

where L is the likelihood of the model. AIC is meaningless in the absolute sense, and is only meaningful when compared to AIC values from other models. Lower values of AIC indicate better fitting models.

`statsmodels` provides the AIC in its output.

Part 4 Checkup Exercises

Exercise: Find another variable (or two) to add to the model we built in Part 3. Compute the F-test comparing the two models as well as the AIC. Which model is better?

Part 5: Evaluating the Model via Model Assumptions and Other Issues

Linear regression makes several assumptions. It is always best to check that these assumptions are valid after fitting a linear regression model.

- **Linearity**. The dependent variable Y is a linear combination of the regression coefficients and the independent variables X. This can be verified with a scatterplot of each X vs. Y and plotting correlations among X. Nonlinearity can sometimes be resolved by [transforming](https://onlinecourses.science.psu.edu/stat501/node/318) one or more independent variables, the dependent variable, or both. In other cases, a [generalized linear model](https://en.wikipedia.org/wiki/Generalized_linear_model) or a [nonlinear model](https://en.wikipedia.org/wiki/Nonlinear_regression) may be warranted.
- **Constant standard deviation**. The SD of the dependent variable Y should be constant for different values of X. We can check this by plotting each X against Y and verifying that there is no "funnel" shape showing data points fanning out as X increases or decreases. Some techniques for dealing with non-constant variance include weighted least squares (WLS), [robust standard errors](https://en.wikipedia.org/wiki/Heteroscedasticity-consistent_standard_errors), or variance stabilizing transformations.
- **Normal distribution for errors**. The ϵ term we discussed at the beginning are assumed to be normally distributed. This can be verified with a fitted values vs. residuals plot and verifying that there is no pattern, and with a quantile plot. $$ \epsilon_i \sim N(0, \sigma^2)$$ 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

There are some other issues that are important investigate with linear regression models.

- **Correlated Predictors:** Care should be taken to make sure that the independent variables in a regression model are not too highly correlated. Correlated predictors typically do not majorly affect prediction, but do inflate standard errors of coefficients making interpretation unreliable. Common solutions are dropping the least important variables involved in the correlations, using regularlization, or, when many predictors are highly correlated, considering a dimension reduction technique such as principal component analysis (PCA).
- **Influential Points:** Data points that have undue influence on the regression model. These points can be high leverage points or outliers. Such points are typically removed and the regression model rerun.

Part 5 Checkup Exercises

Take the reduced model from Part 3 to answer the following exercises. Take a look at [this blog post](http://mpastell.com/2013/04/19/python_regression/) for more information on using statsmodels to construct these plots.

Exercise: Construct a fitted values versus residuals plot. What does the plot tell you? Are there any violations of the model assumptions?

Exercise: Construct a quantile plot of the residuals. What does the plot tell you?

Exercise: What are some advantages and disadvantages of the fitted vs. residual and quantile plot compared to each other?

Exercise: Identify any outliers (if any) in your model and write a story describing what these outliers might represent.

Exercise: Construct a leverage plot and identify high leverage points in the model. Write a story explaining possible reasons for the high leverage points.

Exercise: Remove the outliers and high leverage points from your model and run the regression again. How do the results change?

In [53]:

```
# Your turn.
X_train, X_test, Y_train, Y_test = sklearn.model_selection.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)
lm = LinearRegression()
lm.fit(X_train, Y_train)
pred_train = lm.predict(X_train)
pred_test = lm.predict(X_test)
```

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

In [54]:

```
print (np.mean((Y_train - lm.predict(X_train)) ** 2))
print (np.mean((Y_test - lm.predict(X_test)) ** 2))
```

```
19.546758473534663
28.530458765974604
```

In [55]:

```
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')
```

Out[55]:

```
Text(0, 0.5, 'Residuals')
```

In [64]:

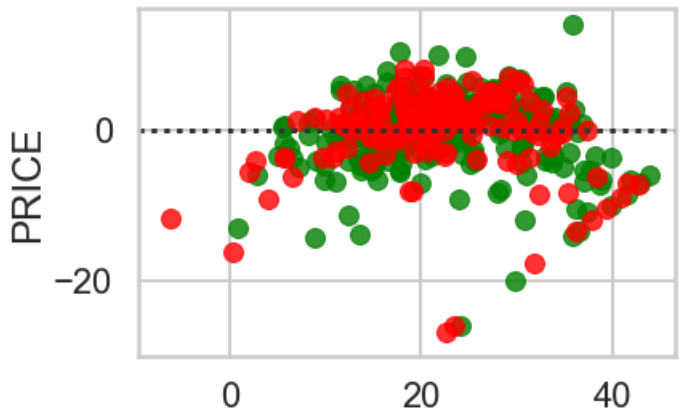
```
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from yellowbrick.datasets import load_concrete
from yellowbrick.regressor import ResidualsPlot
```

```
#Load a regression dataset
X, y = load_concrete()

# Create the train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Instantiate the linear model and visualizer
model = Ridge()
visualizer = ResidualsPlot(lm)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.show()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-64-ad15fa90e37d> in <module>
      2 from sklearn.model_selection import train_test_split
      3 from sklearn.linear_model import Ridge
----> 4 from yellowbrick.datasets import load_concrete
      5 from yellowbrick.regressor import ResidualsPlot
      6

ModuleNotFoundError: No module named 'yellowbrick'
```

In [77]:

```
# Import plotting modules
import matplotlib.pyplot as plt
import seaborn as sns

# Generate a green residual plot of the regression between 'hp' and 'mpg'
sns.residplot(x=pred_train, y=(pred_train - Y_train), color='green')
sns.residplot(x=pred_test, y=(pred_test - Y_test), color='red')
# Display the plot
plt.show()
```



In [81]:

```
residuals = Y_train - pred_train.reshape(-1)
residuals_1 = Y_test - pred_test.reshape(-1)

plt.figure(figsize=(7,7))
stats.probplot(residuals, dist="norm", c='green', plot=plt)
stats.probplot(residuals_1, dist="norm", c='blue', plot=plt)
plt.title("Normal Q-Q Plot")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-cf8cefe0e18c> in <module>
      3
      4 plt.figure(figsize=(7,7))
----> 5 stats.probplot(residuals, dist="norm", c='green', plot=plt)
      6 stats.probplot(residuals_1, dist="norm", c='blue', plot=plt)
      7 plt.title("Normal Q-Q Plot")
```

```
TypeError: probplot() got an unexpected keyword argument 'c'
```

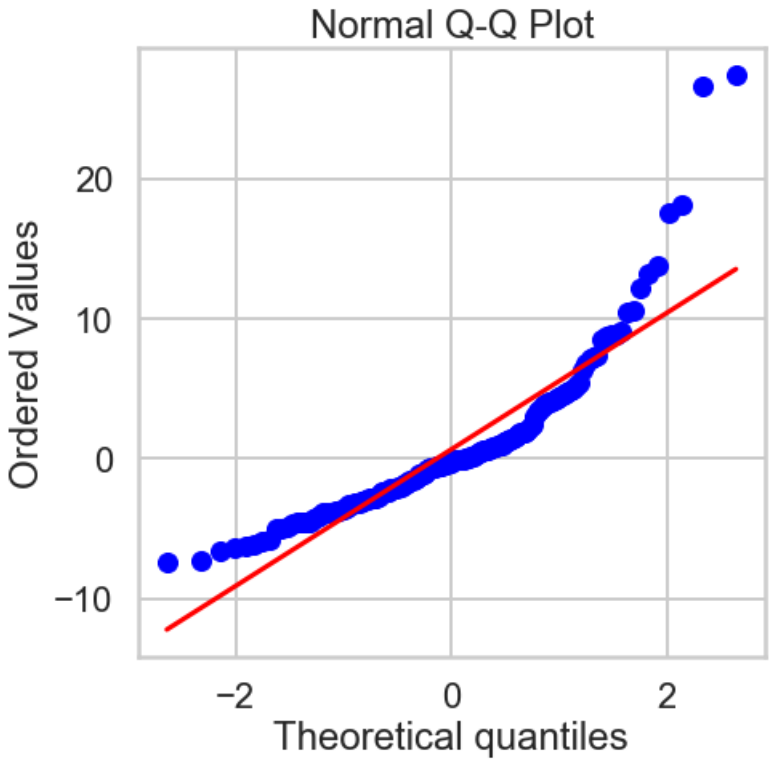
```
<Figure size 504x504 with 0 Axes>
```

In [72]:

```
residuals = Y_test - pred_test.reshape(-1)
residuals

plt.figure(figsize=(7,7))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Normal Q-Q Plot")
```

Out[72]:

```
Text(0.5, 1.0, 'Normal Q-Q Plot')
```

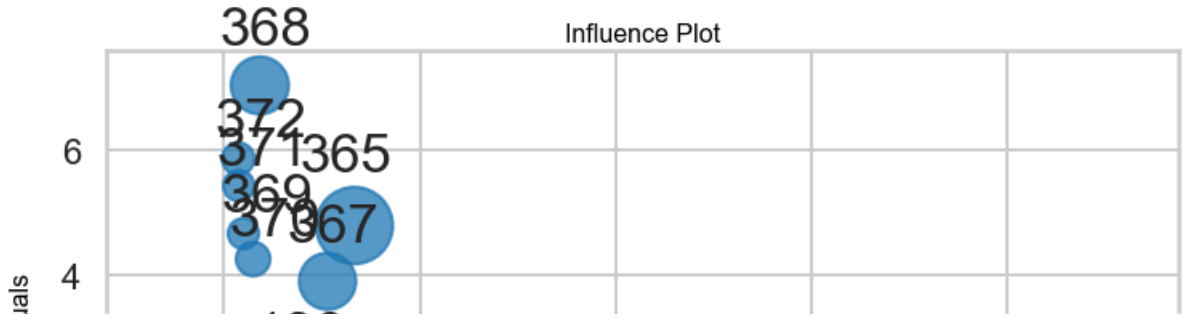


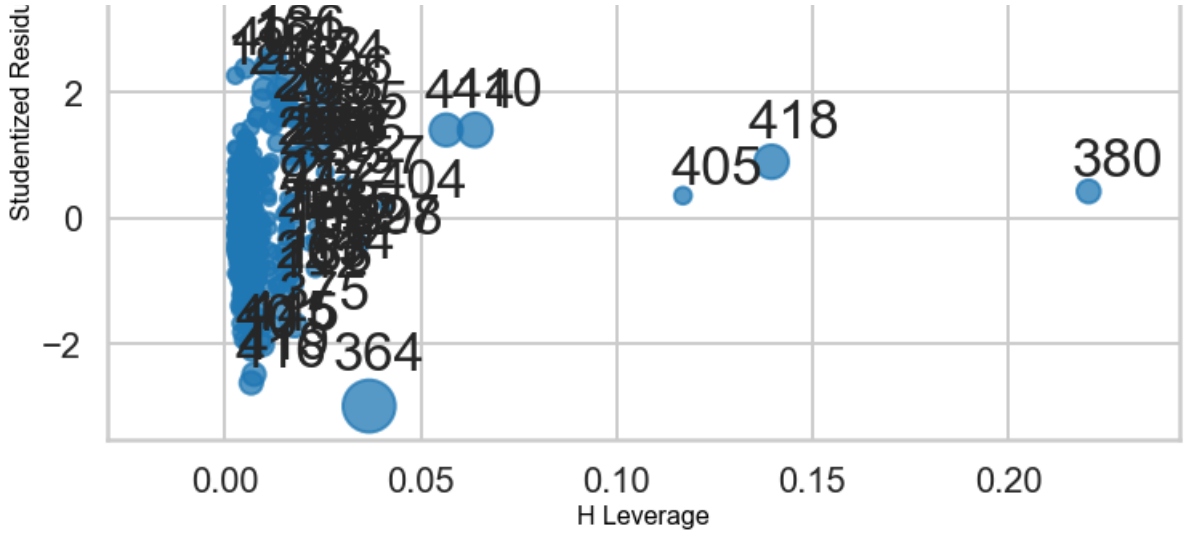
In [83]:

```
from statsmodels.compat import lzip
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In [84]:

```
fig, ax = plt.subplots(figsize=(12,8))
fig = sm.graphics.influence_plot(m, ax=ax, criterion="cooks")
```





In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: