Week 4: Mini Project

This notebook will guide you through smaller portions of your final project. For this notebook, we will be using the Abalone dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Abalone) (originating from the Marine Research Laboratories – Taroona). This dataset should already be in your folder (under `abalone.csv`) or you can download it at the above link.

Abalone

A Brief History of Abalones

An abalone is a sea snail belonging to one of a range of 30 to 130 species (depending on which scientist you ask). It is commonly prized for its mother-of-pearl shell, pearls, and delicious flesh by a variety of cultures and has long been a valuable source of food in its native environments. Sadly, wild populations of abalone have been overfished and poached to the point where commercial farming supplies most of abalone flesh nowadays. It now sits on the list of current animals threatened by extinction.

Source: https://en.wikipedia.org/wiki/Abalone (https://en.wikipedia.org/wiki/Abalone)

Part 1: Familiarize Yourself With the Dataset

The purpose of this dataset is to predict the age of an abalone through physical characteristics, determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Good thing it's already been done for us!

Below is the dataset description from the UCI Machine Learning Repository.

Name	Data Type	Measure	Description
Sex	nominal		M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer		+1.5 gives the age in years

Run the cells below to examine the dataset.

In [70]:
```
# Load Abalone dataset
# Remember to change the file location if needed
import csv
path = "C:/Users/justm\Documents/Python Training/BDPV - Mini Project/abalone.csv'
f = open(path)
all_lines = csv.reader(f, delimiter = ',')

# We define a header ourselves since the dataset contains only the raw numbers.
dataset = []
header = ['Sex', 'Length', 'Diameter', 'Height', 'Whole Weight', 'Shucked Weight'
          'Shell Weight', 'Rings']
for line in all_lines:
    d = dict(zip(header, line))
    d['Length'] = float(d['Length'])
    d['Diameter'] = float(d['Diameter'])
    d['Height'] = float(d['Height'])
    d['Whole Weight'] = float(d['Whole Weight'])
    d['Shucked Weight'] = float(d['Shucked Weight'])
    d['Viscera Weight'] = float(d['Viscera Weight'])
    d['Shell Weight'] = float(d['Shell Weight'])
    d['Rings'] = int(d['Rings'])
    dataset.append(d)
```

In [71]:
```
# See first line of dataset
dataset[0]
```

Out[71]:
```
{'Sex': 'M',
 'Length': 0.455,
 'Diameter': 0.365,
 'Height': 0.095,
 'Whole Weight': 0.514,
 'Shucked Weight': 0.2245,
 'Viscera Weight': 0.101,
 'Shell Weight': 0.15,
 'Rings': 15}
```

Part 2: Simple Statistics

This dataset is already cleaned for us and relatively straightforward, without strings or time data. In your final project, you will have to take care of missing or tricky values yourself.

Fill in the following cells with the requested information about the dataset. The answers are given so you can check the output of your own code. For floating numbers, don't worry too much about the exact numbers as long as they are quite close -- different systems may have different rounding protocols.

Feel free to `import numpy` if you want more practice with it, or just use Python's native structures to play around with the numbers.

In [72]:
```
# Q: What is the total number of entries in the dataset?
# A: 4177
len(dataset)
```

Out[72]: 4177

In [73]:
```
# Q: What is the average length of an abalone?
# A: 0.5239920995930099 or 0.524
data_length=[]
for d in dataset:
    data_length.append(d['Length'])

import numpy
numpy.average(data_length)
```

Out[73]: 0.5239920995930094

In [74]:
```
# Q: What is the widest abalone in the dataset (diameter)?
# A: 0.65
data_Diameter=[]
for d in dataset:
    data_Diameter.append(d['Diameter'])

import numpy
numpy.max(data_Diameter)
```

Out[74]: 0.65

In [75]:
```
# Q: What is the average number of rings of smaller abalones compared to that of
#    is, do smaller abalones tend to be younger or older than larger abalones?
#    We will count small abalones as abalones with lengths less than or equal to
#    an abalone. The average length of an abalone is 0.524.
# A: Small Abalones have on average 8.315645514223196 rings.
#    Large Abalones have on average 11.192848020434228 rings.

# Change variable name if necessary

data_Ringssmall=[]
data_Ringslarge=[]
for d in dataset:
    if d['Length']<=0.524:
        data_Ringssmall.append( d['Rings'] )
    elif d['Length']>0.524:
        data_Ringslarge.append( d['Rings'] )

import numpy
ageSmall=numpy.average(data_Ringssmall)
ageLarge=numpy.average(data_Ringslarge)

print('Small Abalones have on average', ageSmall, 'rings.')
print('Large Abalones have on average', ageLarge, 'rings.')
```

```
Small Abalones have on average 8.315645514223196 rings.
Large Abalones have on average 11.192848020434228 rings.
```

Part 3: Data Visualizations

In this course, we learned about [Matplotlib (https://matplotlib.org)](https://matplotlib.org), a "Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms". There are a [variety of plots and figures (https://matplotlib.org/gallery/index.html)](https://matplotlib.org/gallery/index.html) we can make with Matplotlib, and in conjunction with NumPy, becomes a powerful and versatile tool in your skillset.

In lectures, we covered the basics of line plots, histograms, scatter plots, bar plots, and box plots. Let's try out a few below.

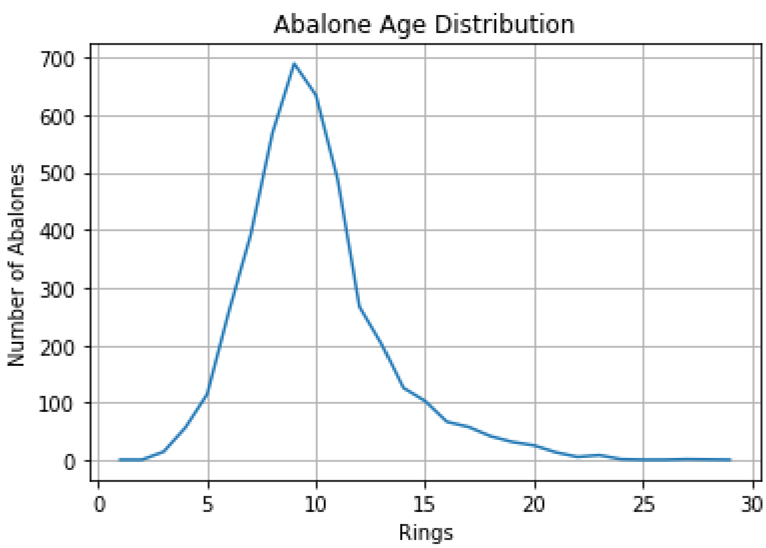
In [76]:
```
import matplotlib.pyplot as plt
from matplotlib import colors
import numpy
from collections import defaultdict
```

Line Plots

Line plots show the change in data over time. The example Line Plot below plots the change in density as abalones age (i.e. the distribution of rings). **Note that a line plot is not necessarily the best way to show this data since it doesn't deal with a trend!** Use a histogram (next step) to better showcase this data.

In [77]:
```
# Parse out Rings column from dataset
rings = [d['Rings'] for d in dataset]
rings.sort()

# Count number of abalones with each number of rings with defaultdict
abalone_rings = defaultdict(int)
for r in rings:
    abalone_rings[r] += 1
X = list(abalone_rings.keys())
Y = list(abalone_rings.values())

# Customize plot
plt.gca().set(xlabel='Rings', ylabel='Number of Abalones',
       title='Abalone Age Distribution')
plt.grid()

# Show the plot of Rings vs Number of Abalones
plt.plot(X, Y)
```

Out[77]: `[<matplotlib.lines.Line2D at 0x1e9fa5dde08>]`



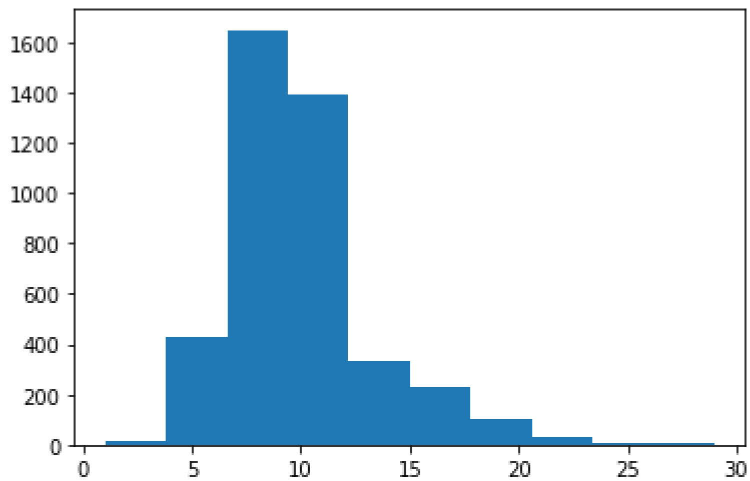
Histograms

Histograms show the distribution of numeric continuous variables with central tendency and skewness. **Using the line plot data from above, plot a histogram showing the distribution of abalone age.** Feel free to explore [matplotlib (https://matplotlib.org/gallery/index.html)](https://matplotlib.org/gallery/index.html) on your own to customize your histogram and the following visualizations.

In [78]:
```python
# Complete this cell with a histogram of abalone age distribution

# Flatten distribution list into frequency distribution
age_freq = []
for key in abalone_rings.keys():
    for i in range(0, abalone_rings.get(key)):
        age_freq.append(key)
print(age_freq[:10])

# Plot your histogram here
plt.hist(age_freq)
```

```
[1, 2, 3, 3, 3, 3, 3, 3, 3, 3]
```

Out[78]:
```
(array([  17.,  431., 1648., 1388.,  329.,  228.,  100.,   29.,    4.,
           3.]),
 array([ 1. ,  3.8,  6.6,  9.4, 12.2, 15. , 17.8, 20.6, 23.4, 26.2, 29. ]),
 <a list of 10 Patch objects>)
```



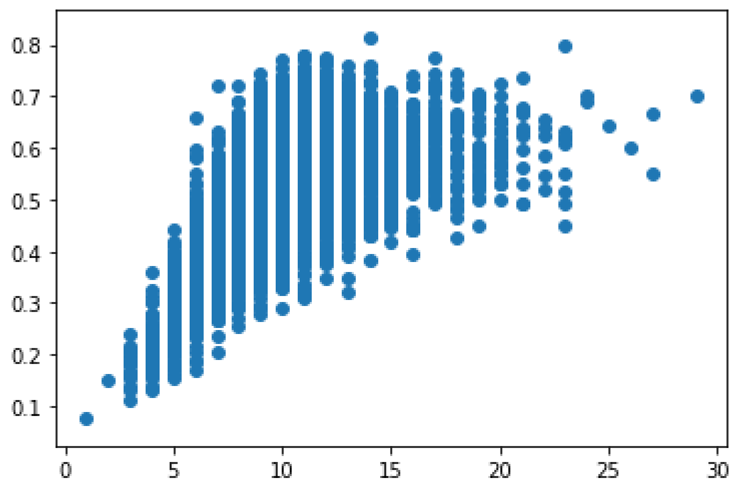
Scatter Plots

Scatter plots show the strength of a relationship between two variables (also known as correlations). From *Part 2: Simple Statistics*, we see that larger abalones tend to be larger, at least from a numbers perspective. **Let's see if this is actually true by creating a scatter plot showing the relationship between `Rings` and `Length`.**

On Your Own: Read up on `sciPy` and how you can calculate and graph the correlation as well.

In [79]:
```
# Complete this cell with a scatter plot of age vs length
rings = [d['Rings'] for d in dataset]
length = [d['Length'] for d in dataset]
plt.scatter(rings, length)
```

Out[79]:
```
<matplotlib.collections.PathCollection at 0x1e9fa64c508>
```

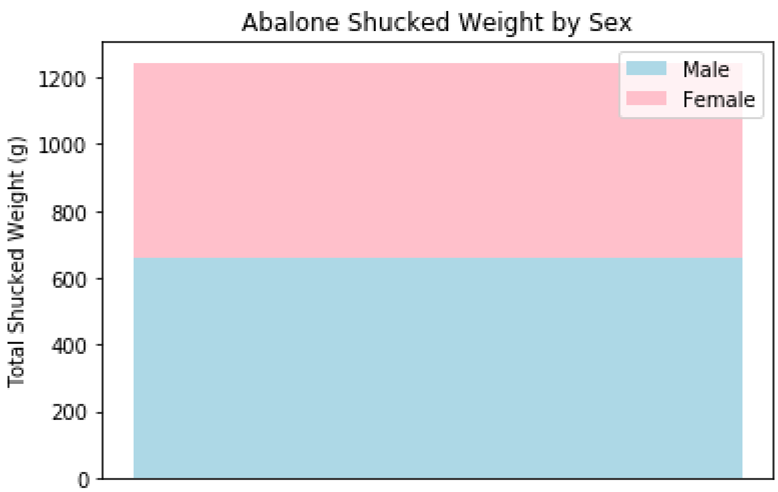


Bar Plots

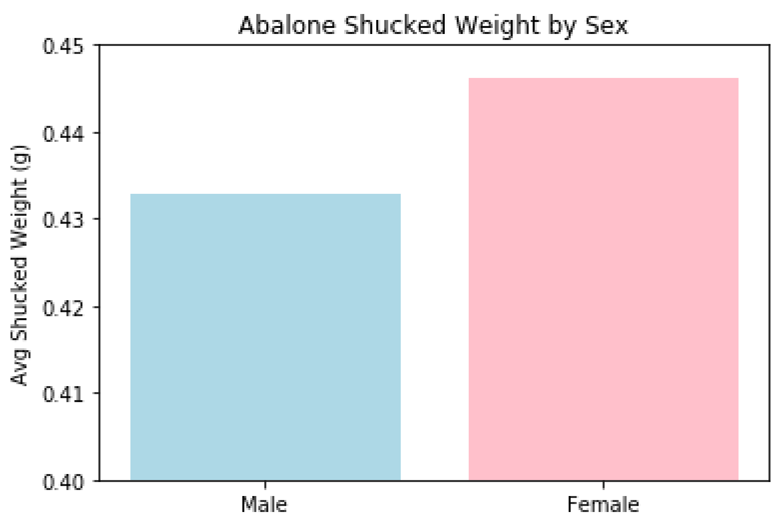
Bar plots are great for comparing categorical variables. There are a few subtypes of bar plots, such as the grouped bar chart or stacked bar chart. Since we have the `Sex` field to play with, we can compare data across `M` and `F` abalones. Below is a simple stacked bar chart comparing the `Sex` category with the `Shucked Weight` data. **Create a bar chart of your choice of data.**

You may refer to the cell below to parse out fields by sex.

In [80]:
```
# Example Stacked Bar Chart - Comparisons Between Sexes
Mweight = sum([d['Shucked Weight'] for d in dataset if d['Sex'] is 'M'])
Fweight = sum([d['Shucked Weight'] for d in dataset if d['Sex'] is 'F'])
index = [1]

p1 = plt.bar(index, Mweight, color='lightblue')
p2 = plt.bar(index, Fweight, bottom=Mweight, color='pink')
plt.gca().set(title='Abalone Shucked Weight by Sex', ylabel='Total Shucked Weight
plt.xticks([])

plt.legend((p1[0], p2[0]), ('Male', 'Female'))
plt.show()
```



In [81]:
```
# Complete this cell with your choice of data
import numpy
Mweight = numpy.mean([d['Shucked Weight'] for d in dataset if d['Sex'] is 'M'])
Fweight = numpy.mean([d['Shucked Weight'] for d in dataset if d['Sex'] is 'F'])
index = [1]

p1 = plt.bar([1], Mweight, color='lightblue')
p2 = plt.bar([2], Fweight, color='pink')
plt.gca().set(title='Abalone Shucked Weight by Sex', ylabel='Avg Shucked Weight (
plt.xticks([1,2], ['Male', 'Female'])
plt.ylim(0.4,0.45)

#plt.legend((p1[0], p2[0]), ('Male', 'Female'))
plt.show()
```



Box Plots

Box plots are useful for comparing distributions of data and are commonly found in research papers. The box portion of a box plot represents 50% of the data, and there are versions where you can mark outliers and other extremes. We have the distribution of rings already from the line plot example under the variable name `age_freq` , assuming you haven't modified it. **Find the distribution of another field of your choice and create one or more box plots with both of these fields.**

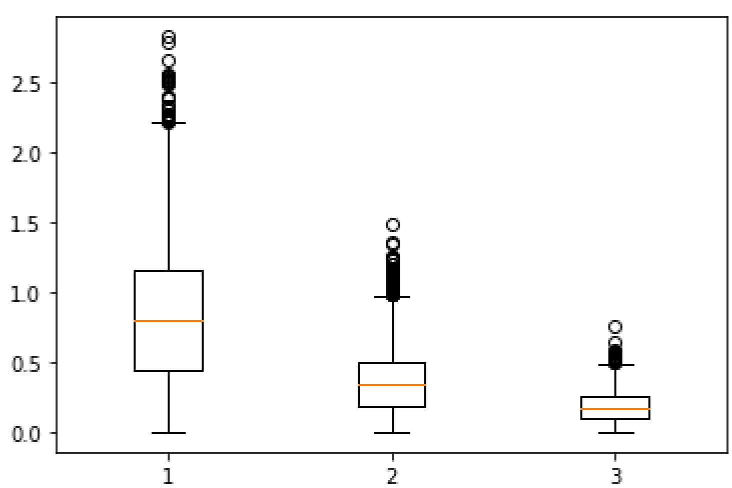
Hint: You can plot multiple box plots with the command `plt.boxplot([plot1, plot2, ..., plotn])` or use `subplots()` to draw multiple separate plots at the same time. See [this matplotlib example (https://matplotlib.org/gallery/statistics/boxplot_demo.html#sphx-glr-gallery-statistics-boxplot-demo-py)](https://matplotlib.org/gallery/statistics/boxplot_demo.html#sphx-glr-gallery-statistics-boxplot-demo-py) for more.

In [82]:

```python
# Complete this cell with multiple box plots

Whole_weight = [d['Whole Weight'] for d in dataset]
Shucked_weight = [d['Shucked Weight'] for d in dataset]
Viscera_weight = [d['Viscera Weight'] for d in dataset]


data = [Whole_weight, Shucked_weight,Viscera_weight]

# Multiple box plots on one Axes
fig, ax = plt.subplots()
ax.boxplot(data)
```

Out[82]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1e9fbabf248>,
  <matplotlib.lines.Line2D at 0x1e9fba96048>,
  <matplotlib.lines.Line2D at 0x1e9faa13288>,
  <matplotlib.lines.Line2D at 0x1e9fa7ea848>,
  <matplotlib.lines.Line2D at 0x1e9f8eacb88>,
  <matplotlib.lines.Line2D at 0x1e9f8eacb08>],
 'caps': [<matplotlib.lines.Line2D at 0x1e9fba96e88>,
  <matplotlib.lines.Line2D at 0x1e9fa5f6b48>,
  <matplotlib.lines.Line2D at 0x1e9fa80f388>,
  <matplotlib.lines.Line2D at 0x1e9f8e9d988>,
  <matplotlib.lines.Line2D at 0x1e9f8eb4cc8>,
  <matplotlib.lines.Line2D at 0x1e9f8eb4c48>],
 'boxes': [<matplotlib.lines.Line2D at 0x1e9fbabfb08>,
  <matplotlib.lines.Line2D at 0x1e9fa70c7c8>,
  <matplotlib.lines.Line2D at 0x1e9fa4ac9c8>],
 'medians': [<matplotlib.lines.Line2D at 0x1e9fa5f6688>,
  <matplotlib.lines.Line2D at 0x1e9f8e9da88>,
  <matplotlib.lines.Line2D at 0x1e9f8ebcdc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x1e9fa4ac608>,
  <matplotlib.lines.Line2D at 0x1e9f8ea6a48>,
  <matplotlib.lines.Line2D at 0x1e9f8ebcec8>],
 'means': []}
```



Free Response (optional)

Experiment and create visualizations of your own here.

In [ ]:
```
# Description of visualization
```

Part 4: Web Scraping (Optional)

BeautifulSoup Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

This part of the notebook is not graded, but still contains some valuable tips for web-scraping! You were introduced to a method of creating your own dataset by parsing a webpage in lecture videos and this week's notebook. Here is another way to parse a webpage with BeautifulSoup. We will be using a short story from Project Gutenberg (*Little Boy* (http://www.gutenberg.org/files/58743/58743-h/58743-h.htm) by Harry Neal, 1954) as an example.

On Your Own: Read this page on webscraping and try out a project! https://automatetheboringstuff.com/chapter11/ (https://automatetheboringstuff.com/chapter11/)

Introduction to Beautiful Soup

Below are a few useful commands we will be using throughout the next section as we parse a webpage.

In [ ]:
```
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

In [ ]:
```
# Open and extract HTML from the webpage
f = urlopen("http://www.gutenberg.org/files/58743/58743-h/58743-h.htm")
html = str(f.read())

# First 100 characters of the HTML we grabbed
html[:100]
```

In [ ]:
```
# Convert our HTML object to a BeautifulSoup object and make it readable
soup = BeautifulSoup(html, 'html.parser')
print(soup.prettify())
```

With a BeautifulSoup object, we can easily search through HTML and create lists and other structures.

In [ ]:
```python
# Number of paragraph tags
len(soup.find_all('p'))
```

In [ ]:
```python
# Create list of all paragraphs
paragraph_list = soup.find_all('p')
paragraph_list[100]
```

We can also extract all the text from a page and use it to create a bag of words or other measures.

In [ ]:
```python
# Extract all text from page
text = soup.get_text()
text[:100]
```

In [ ]:
```python
import string
from collections import defaultdict

letters = defaultdict(int)
punctuation = set(string.punctuation)

for char in text:
    if char not in punctuation:
        letters[char] += 1

letters.items()
```

Creating Our Own Dataset

Cooking

In previous lectures and notebooks, we wrote our own parser method to extract parts of the text. Here is a trivial example of how you can do the same with BeautifulSoup using a list of [Top 10 Chefs by Gazette Review (https://gazettereview.com/2017/04/top-10-chefs/)](https://gazettereview.com/2017/04/top-10-chefs/).

In [ ]:
```python
# Open and extract HTML from the webpage
f = urlopen("https://gazettereview.com/2017/04/top-10-chefs/")
html = str(f.read())
soup = BeautifulSoup(html, 'html.parser')
print(soup.prettify())
```

Note that all the names of the chefs are between `<h2>` and `</h2>` tags and the descriptions are between `<p>` and `</p>` tags. We can get the names of the chefs quite easily, as seen below.

In [ ]:
```
# List of chef names
# Note that find_all() returns a bs4 object, rather than a Python list.
# The HTML tags are also part of the object.
chefs = soup.find_all('h2')
print(type(chefs))
print(chefs[0])
```

In [ ]:
```
# Clean and strip spaces and numbers from the bs4 element and turn it into a Pyth
import string
letters = set(string.ascii_letters)
chef_name = []

# Grab relevant letters/spaces and remove extra HTML tags and spaces
for chef in chefs:
    chef = [letter for letter in str(chef) if letter in letters or letter is ' ']
    chef = ''.join(chef[2:len(chef) - 1])
    chef_name.append(chef)

chef_name
```

Getting the list of chef names is trivial with the `find_all()` function (and a little Python cleaning), but what about the descriptions? This is a little trickier since there may be overlapping uses for the `<p>` and `</p>` tags, so let's try [navigating the BeautifulSoup tree (https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-the-tree)](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-the-tree).

This website is simple in that every chef has a two-paragraph description in the same format. We can use this to our advantage once we know what to look for. Let's say we want to extract just the text from these two paragraphs. How can we do so? With the `.contents` attribute, we can access the children of each tag.

In [ ]:
```
descriptions = soup.find_all('p')
del descriptions[-12:]
del descriptions[0]
print("The number of paragraphs is:", len(descriptions))
descriptions[:2]
```

In [ ]:
```
# Set up the loop
i = 0
chef_description = [''] * 10
chef_image = []

# Grab description text from paragraphs
for d in descriptions:
    curr_desc = []
    if i % 2 == 0:
        curr_desc = d.contents[2]
        chef_image.append(d.contents[0]['src']) # Get images as well
    else:
        curr_desc = d.contents[0]
    # Append relevant parts to corresponding index
    chef_description[int(i / 2)] = chef_description[int(i / 2)] + ' ' + curr_desc
    i += 1

# Voila! We have combined 2 paragraphs into 1.
chef_description[0]
```

We now have lists with the names, descriptions, and images of the chefs! You can arrange this however you want; `chef_data` below is arranged like a JSON object but you can modify this section to make the data look more like a traditional dataset.

In [ ]:
```
chef_data = {}

chef_data['Name'] = chef_name
chef_data['Description'] = chef_description
chef_data['Image'] = chef_image

chef_data['Description'][0]
```

(Optional) Your Turn: Web-Scraping

Now that you've run through this section of the notebook, feel free to experiment with web-scraping on your own. Choose a site and get some raw data out of it!

Note: If you run into a `HTTP error 403 (Forbidden)`, this means that the site probably blocks web-scraping scripts. You can get around this by modifying the way you request the URL (see [StackOverflow (https://stackoverflow.com/questions/28396036/python-3-4-urllib-request-error-http-403)](https://stackoverflow.com/questions/28396036/python-3-4-urllib-request-error-http-403) for some useful tips) or try another site.

In [ ]:
```
# Start parsing here
```

All Done!

In this notebook, we covered loading a dataset, simple statistics, basic data visualizations, and web-scraping to round out your toolset. These will be immensely helpful as you move forwards in building your skills in data science.

By now, you hopefully feel a little more confident with tackling your final project. It is up to you to find your own data, build your own notebook, and show others what you have achieved. Best of luck!

In [ ]:

```

```