ABALONE: EDA Exercise

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.

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 [11]:
```
import pandas as pd
import numpy as np

header = ['Sex', 'Length', 'Diameter', 'Height', 'Whole Weight', 'Shuc
          'Shell Weight', 'Rings']
df = pd.read_csv('abalone.csv', header=None, names=header)
df.head()
```

Out[11]:

	Sex	Length	Diameter	Height	Whole Weight	Shucked Weight	Viscera Weight	Shell Weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
Sex                 4177 non-null object
Length              4177 non-null float64
Diameter            4177 non-null float64
Height              4177 non-null float64
Whole Weight        4177 non-null float64
Shucked Weight      4177 non-null float64
Viscera Weight      4177 non-null float64
Shell Weight        4177 non-null float64
Rings               4177 non-null int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

In [13]:
```
# Alternate way to load the dataset without pandas
import csv
f = open("abalone.csv")
all_lines = csv.reader(f, delimiter = ',')

# We define a header ourselves since the dataset contains only the raw
dataset = []
header = ['Sex', 'Length', 'Diameter', 'Height', 'Whole Weight', 'Shuc
          '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 [14]:
```
# See first line of dataset
dataset[0]
```

Out[14]:
```
{'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 [15]:
```
# Q: What is the total number of entries in the dataset?
# A: 4177
df.count()
```

Out[15]:
```
Sex               4177
Length            4177
Diameter          4177
Height            4177
Whole Weight      4177
Shucked Weight    4177
Viscera Weight    4177
Shell Weight      4177
Rings             4177
dtype: int64
```

In [17]:
```
# Q: What is the average length of an abalone?
# A: 0.5239920995930099 or 0.524
df['Length'].mean()
```

Out[17]:
```
0.5239920995930094
```

In [18]:
```
# Q: What is the widest abalone in the dataset (diameter)?
# A: 0.65
df['Diameter'].max()
```

Out[18]:
```
0.65
```

In [27]:
```
mean_length = df['Length'].mean()
ageSmall = df[df.Length <= mean_length].Rings.mean()
ageLarge = df[df.Length >= mean_length].Rings.mean()
```

In [35]:
```
df.groupby(by=['Rings']).count()
```

Out[35]:

Rings	Sex	Length	Diameter	Height	Whole Weight	Shucked Weight	Viscera Weight	Shell Weight
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	15	15	15	15	15	15	15	15
4	57	57	57	57	57	57	57	57
5	115	115	115	115	115	115	115	115
6	259	259	259	259	259	259	259	259
7	391	391	391	391	391	391	391	391
8	568	568	568	568	568	568	568	568
9	689	689	689	689	689	689	689	689
10	634	634	634	634	634	634	634	634
11	487	487	487	487	487	487	487	487
12	267	267	267	267	267	267	267	267
13	203	203	203	203	203	203	203	203
14	126	126	126	126	126	126	126	126
15	103	103	103	103	103	103	103	103
16	67	67	67	67	67	67	67	67
17	58	58	58	58	58	58	58	58
18	42	42	42	42	42	42	42	42
19	32	32	32	32	32	32	32	32
20	26	26	26	26	26	26	26	26
21	14	14	14	14	14	14	14	14
22	6	6	6	6	6	6	6	6
23	9	9	9	9	9	9	9	9
24	2	2	2	2	2	2	2	2
25	1	1	1	1	1	1	1	1
26	1	1	1	1	1	1	1	1
27	2	2	2	2	2	2	2	2
29	1	1	1	1	1	1	1	1

In [28]:
```
# Q: What is the average number of rings of smaller abalones compared
#    is, do smaller abalones tend to be younger or older than larger a
#    We will count small abalones as abalones with lengths less than o
#    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
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), 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) 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.

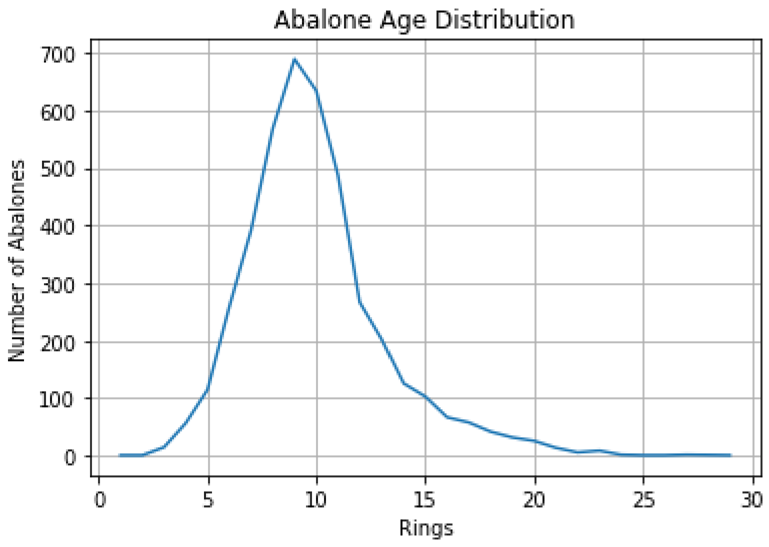
In [29]:
```
import matplotlib.pyplot as plt
from matplotlib import colors
import numpy
from collections import defaultdict
```

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 [30]:
```
# 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[30]:
```
[<matplotlib.lines.Line2D at 0x7fb545045da0>]
```



In [38]:
```
abalone_rings
```

Out[38]:
```
defaultdict(int,
            {1: 1,
             2: 1,
             3: 15,
             4: 57,
             5: 115,
             6: 259,
             7: 391,
             8: 568,
             9: 689,
             10: 634,
             11: 487,
             12: 267,
             13: 203,
             14: 126,
             15: 103,
             16: 67,
             17: 58,
             18: 42,
             19: 32,
             20: 26,
             21: 14,
             22: 6,
             23: 9,
             24: 2,
             25: 1,
             26: 1,
             27: 2,
             29: 1})
```

Histograms

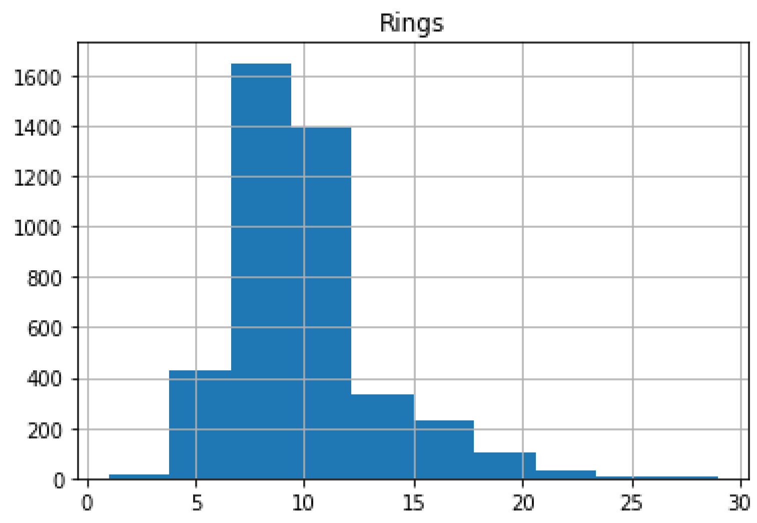
Histograms show the distribution of numeric continuous variables with central tendency and skewness. **Using the line plot data from above, plot a histogram showing the distribution of abalone age.** Feel free to explore matplotlib (https://matplotlib.org/gallery/index.html) on your own to customize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

In [39]:
```
# 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[:20])

# Plot your histogram here
```

```
[1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4]
```

In [46]:
```
df.hist(column= 'Rings')
```

Out[46]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb54270c320
>]],
      dtype=object)
```



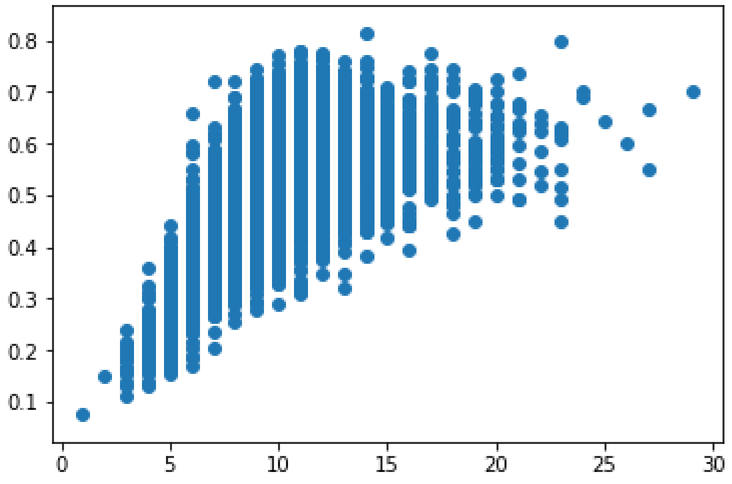
Scatter Plots

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.

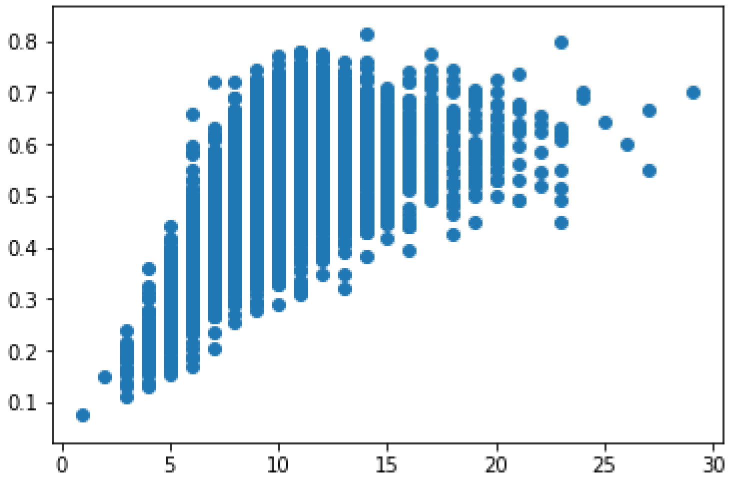
In [48]:
```
plt.scatter(x=df['Rings'], y=df['Length'])
```

Out[48]:
```
<matplotlib.collections.PathCollection at 0x7fb5420b3d30>
```



In [49]:
```
# 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(x=rings, y=length)
```

Out[49]:
```
<matplotlib.collections.PathCollection at 0x7fb541ee6240>
```

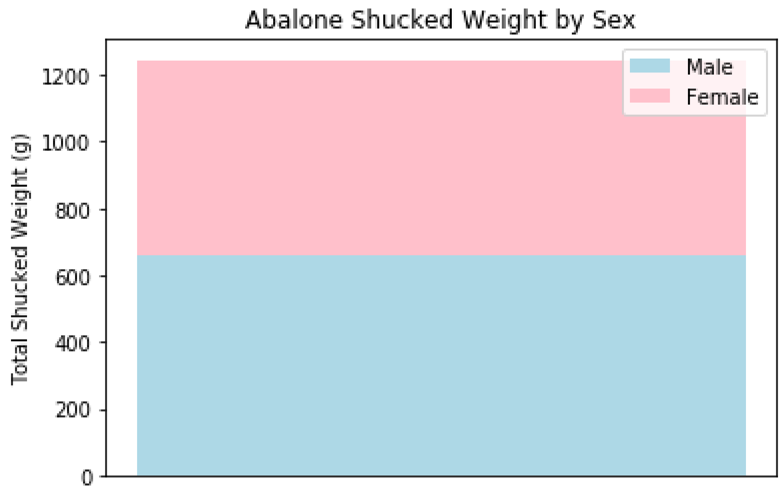


Bar Plots

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.

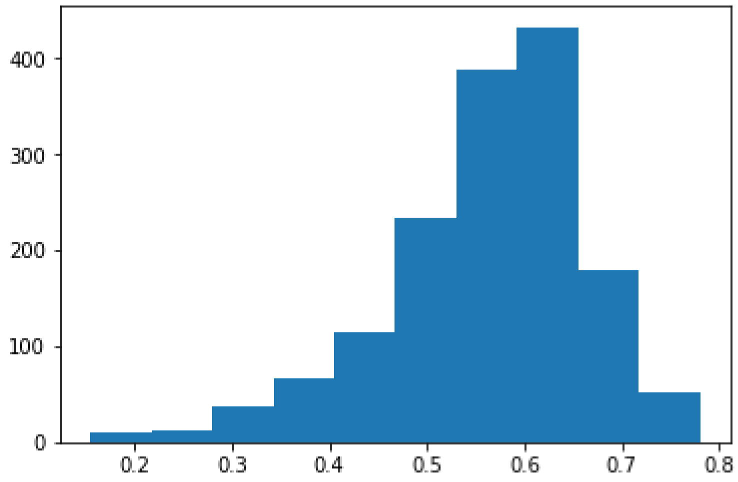
In [50]:
```
# Example Stacked Bar Chart - Comparisons Between Sexes
Mweight = sum([d['Shucked Weight'] for d in dataset if d['Sex'] is 'M'
Fweight = sum([d['Shucked Weight'] for d in dataset if d['Sex'] is 'F'
index = [1]

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 Shu
plt.xticks([])

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



In [70]:
```
# Complete this cell with your choice of data
gender_dict = defaultdict(list)
gender_dict2 = defaultdict(int)
for d in dataset:
    gen = d['Sex']
    gender_dict[gen].append(d['Length'])
    gender_dict2[d['Sex']] += 1
```

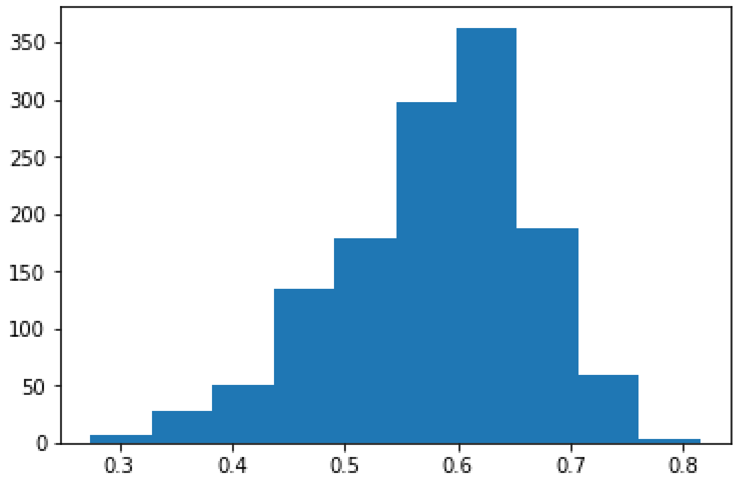
In [62]:
```
plt.hist(x=gender_dict['M'], histtype='bar')
```

Out[62]:
```
(array([ 10.,  13.,  38.,  66., 115., 234., 388., 433., 179.,  52.]),
 array([0.155 , 0.2175, 0.28  , 0.3425, 0.405 , 0.4675, 0.53  , 0.5925,
        0.655 , 0.7175, 0.78  ]),
 <a list of 10 Patch objects>)
```



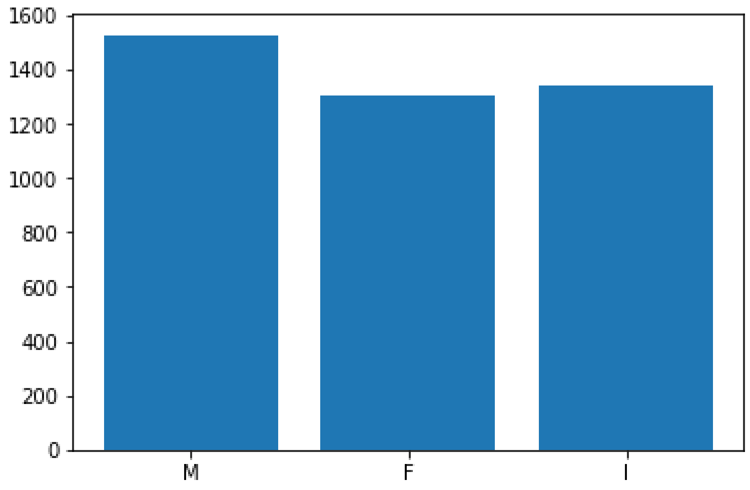
In [63]:
```
plt.hist(x=gender_dict['F'], histtype='bar')
```

Out[63]:
```
(array([  6.,  28.,  50., 135., 178., 297., 363., 187.,  60.,   3.]),
 array([0.275, 0.329, 0.383, 0.437, 0.491, 0.545, 0.599, 0.653, 0.707,
        0.761, 0.815]),
 <a list of 10 Patch objects>)
```



In [75]:
```
X = gender_dict2.keys()
y = [gender_dict2[x] for x in X]
plt.bar(X,y)
```

Out[75]:
```
<BarContainer object of 3 artists>
```



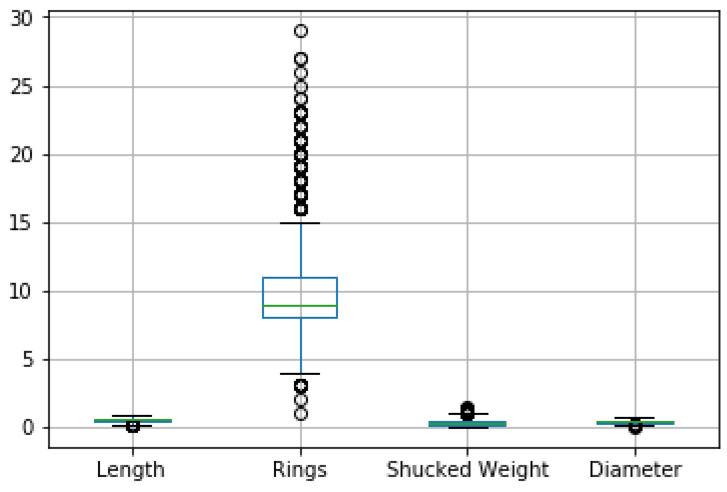
Box Plots

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 [76]:
```
# Complete this cell with multiple box plots
df.boxplot(column=['Length', 'Rings', 'Shucked Weight', 'Diameter'])
```

Out[76]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb53c7c30b8>
```

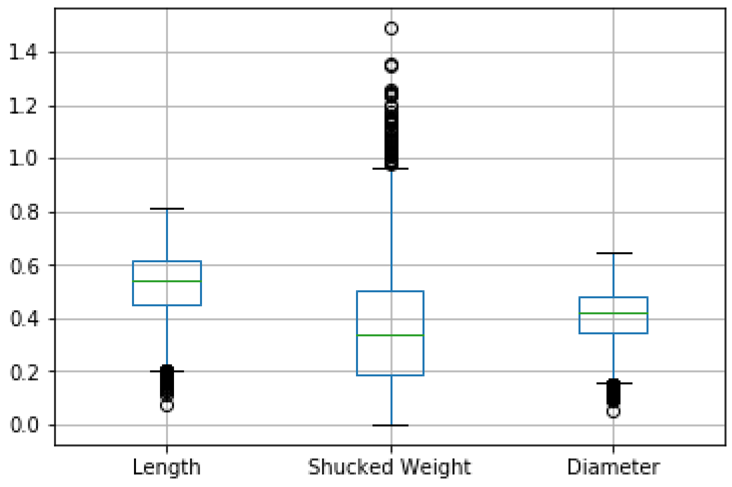


Free Response (optional)

Experiment and create visualizations of your own here.

In [77]:
```
# Description of visualization - Since Rings have too many outliers, v
df.boxplot(column=['Length', 'Shucked Weight', 'Diameter'])
```

Out[77]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb53c7c26d8>
```



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 [80]:
```
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

In [81]:
```
# 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]
```

Out[81]:
```
'b\'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"\\r\\n
"http://www.w3.org/TR/xhtml1/DTD/x'
```

In [82]:
```
# Convert our HTML object to a BeautifulSoup object and make it readab
soup = BeautifulSoup(html, 'html.parser')
print(soup.prettify())
```

```
b'
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"\r\n    "h
ttp://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
\r\n
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtm
l">
 \r\n
 <head>
  \r\n
  <meta content="text/html;charset=utf-8" http-equiv="Content-Typ
e"/>
  \r\n
  <meta content="text/css" http-equiv="Content-Style-Type"/>
  \r\n
  <title>
   \r\n      The Project Gutenberg eBook of Little Boy, by Harry N
eal.\r\n
  </title>
  \r\n
```

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

In [83]:
```
# Number of paragraph tags
len(soup.find_all('p'))
```

Out[83]:
```
165
```

In [84]:
```
# Create list of all paragraphs
paragraph_list = soup.find_all('p')
paragraph_list[100]
```

Out[84]:
```
<p>Slowly he felt his own lips curl back into an expression he could\r
\nhardly remember. He felt the way he felt sometimes late at night whe
n,\r\nsafe and alone in his room, he would play a little with his toys.
He\r\ndidn\'t feel like killing her any more. He felt like ... like <i>
friends</i>.</p>
```

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

In [85]:
```
# Extract all text from page
text = soup.get_text()
text[:100]
```

Out[85]:
```
"b'\\r\\n\\r\\n  \\r\\n    \\r\\n    \\r\\n    \\r\\n      The Project
Gutenberg eBook of Little Boy, by Harry Ne"
```

In [86]:
```
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()
```

Out[86]:
```
dict_items([('b', 606), ('r', 3584), ('n', 3875), (' ', 8120), ('T', 16
0), ('h', 2200), ('e', 4941), ('P', 119), ('o', 2988), ('j', 116),
('c', 981), ('t', 3686), ('G', 119), ('u', 1049), ('g', 866), ('B', 6
1), ('k', 453), ('f', 802), ('L', 74), ('i', 2376), ('l', 1371), ('y',
759), ('H', 138), ('a', 2751), ('N', 74), ('d', 1741), ('m', 844),
('1', 68), ('0', 23), ('2', 16), ('x', 54), ('s', 1993), ('p', 660),
('5', 28), ('4', 19), ('9', 12), ('w', 849), ('3', 25), ('6', 10),
('7', 18), ('I', 135), ('v', 315), ('z', 12), ('8', 16), ('E', 130),
('U', 53), ('S', 181), ('Y', 40), ('A', 110), ('R', 71), ('D', 49),
('J', 7), ('C', 48), ('O', 71), ('F', 75), ('K', 11), ('W', 34), ('M',
33), ('—', 54), ('q', 42), ('V', 8), ('X', 2), ('Q', 1)])
```

Creating Our Own Dataset

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 [87]:
```
# 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())
```

```
b'
<!DOCTYPE doctype html >
\n
<!--[if IE 8]><html class="ie8" lang="en"> <![endif]-->
\n
<!--[if IE 9]><html class="ie9" lang="en"> <![endif]-->
\n
<!--[if gt IE 8]><!-->
<html lang="en-US" prefix="og: http://ogp.me/ns#"> (http://ogp.me/
ns#">)
 <!--<![endif]-->
 <head>
  <title>
   Top 10 Chefs In The World - The Best in 2018 - Gazette Review
  </title>
  <meta charset="utf-8"/>
  <link href="\'https://fonts.googleapis.com/css?family=Open+Sans%
3A300italic%2C400italic%2C600italic%2C400%2C600%2C700\'" id="\'goo
gle_font_open_sans-css\'" media="\'all\'" rel="\'stylesheet\'" typ
e="\'text/css\'"/>
```

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 [88]:
```
# 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])
```

```
<class 'bs4.element.ResultSet'>
<h2>10. Anthony Bourdain</h2>
```

In [89]:
```
# Clean and strip spaces and numbers from the bs4 element and turn it
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 let
    chef = ''.join(chef[2:len(chef) - 1])
    chef_name.append(chef)

chef_name
```

Out[89]:
```
['Anthony Bourdain',
 'Paul Bocuse',
 'Alain Ducasse',
 'Emeril Lagasse',
 'Vikas Khanna',
 'Marco Pierre White',
 'Heston Blumenthal',
 'Wolfgang Puck',
 'Jamie Oliver',
 'Gordon Ramsay']
```

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).

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 [90]:
```
descriptions = soup.find_all('p')
del descriptions[-12:]
del descriptions[0]
print("The number of paragraphs is:", len(descriptions))
descriptions[:2]
```

```
The number of paragraphs is: 20
```

Out[90]:
```
[<p><img alt="" class="size-medium wp-image-65278 alignleft" height="30
0" sizes="(max-width: 200px) 100vw, 200px" src="https://gazettereview.c
om/wp-content/uploads/2017/04/Anthony-Bourdain-200x300.jpg" srcset="htt
ps://gazettereview.com/wp-content/uploads/2017/04/Anthony-Bourdain-200x
300.jpg 200w, https://gazettereview.com/wp-content/uploads/2017/04/Anth
ony-Bourdain-280x420.jpg (https://gazettereview.com/wp-content/uploads/
2017/04/Anthony-Bourdain-280x420.jpg) 280w, https://gazettereview.com/w
p-content/uploads/2017/04/Anthony-Bourdain.jpg (https://gazettereview.c
om/wp-content/uploads/2017/04/Anthony-Bourdain.jpg) 320w" width="200"/>
<br/>\nIt\xe2\x80\x99s hard to believe that the world renowned chef, wr
iter, and television personality Anthony Bourdain\xe2\x80\x99s career s
tarted out with him washing dishes as a college dropout. He is now one
 of the most popular travel and food personalities. Although he is no l
onger officially a chef, his career spanned several decades. He was a c
hef at elite restaurants in New York such as the Supper Club, Sullivan
\xe2\x80\x99s, and One Fifth Avenue.</p>,
 <p>Bourdain has written several successful novels about his culinary a
dventures. His shows are well known by his comedic and often profane co
mmentary. He is also famous for the travel and food series No Reservati
ons. Bourdain also has a blue belt in Brazilian Jiu Jitsu.</p>]
```

In [91]:
```
# 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)] + ' '
    i += 1

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

Out[91]:
```
' \\nIt\\xe2\\x80\\x99s hard to believe that the world renowned chef, w
riter, and television personality Anthony Bourdain\\xe2\\x80\\x99s care
er started out with him washing dishes as a college dropout. He is now
one of the most popular travel and food personalities. Although he is n
o longer officially a chef, his career spanned several decades. He was
a chef at elite restaurants in New York such as the Supper Club, Sulliv
an\\xe2\\x80\\x99s, and One Fifth Avenue. Bourdain has written several
successful novels about his culinary adventures. His shows are well kno
wn by his comedic and often profane commentary. He is also famous for t
he travel and food series No Reservations. Bourdain also has a blue bel
t in Brazilian Jiu Jitsu.'
```

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 [92]:
```
chef_data = {}

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

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

Out[92]:
```
' \\nIt\\xe2\\x80\\x99s hard to believe that the world renowned chef, w
riter, and television personality Anthony Bourdain\\xe2\\x80\\x99s care
er started out with him washing dishes as a college dropout. He is now
one of the most popular travel and food personalities. Although he is n
o longer officially a chef, his career spanned several decades. He was
a chef at elite restaurants in New York such as the Supper Club, Sulliv
an\\xe2\\x80\\x99s, and One Fifth Avenue. Bourdain has written several
successful novels about his culinary adventures. His shows are well kno
wn by his comedic and often profane commentary. He is also famous for t
he travel and food series No Reservations. Bourdain also has a blue bel
t in Brazilian Jiu Jitsu.'
```

(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) 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 [ ]:
```

```