SSE 691: Into to Data Science Project 1

Jonathan Alfred
9-11-2016

Contents

Table of Figures

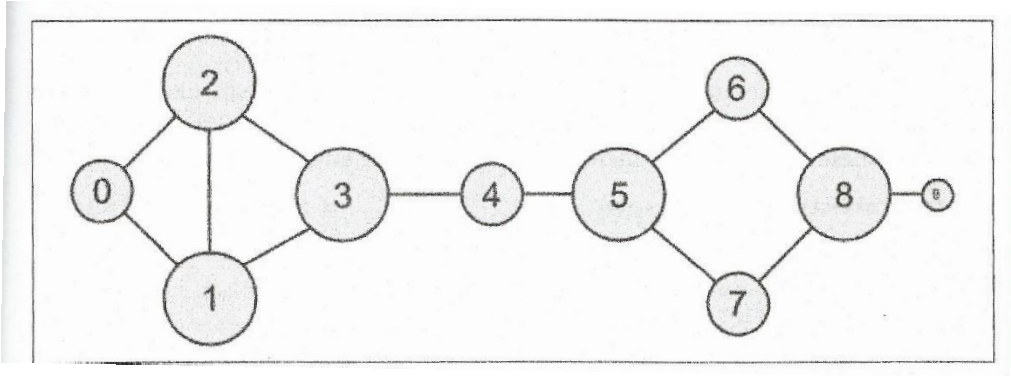
Introduction

This project was meant to demonstrate a student's familiarity with python and data science. Chapter one gave an overview of what data science can be used for with a small example. Chapter two demonstrated many of the functionalities of python. Chapter three showed code for many of the graphical aspects of matplotlib.

Chapter 1

Chapter one give a general description of what data science is. Essentially, it is taking large amount of data and extracting insights from them. Several examples were given in the beginning. Companies such as Facebook and Target use them for interesting purpose. Facebook used hometown and current location to track global migration patterns, while Target utilized data from recent purchase to predict a pregnancy and market baby-related items to them.

This chapter also proposed a hypothetical situation in which the reader worked for a website. This was meant to demonstrate methods in which a social networking website might check its users' friends, suggest other friends they may know, and determine common interests between them. It utilized different mappings, showing how the different users were connected. The following shows the mapping show in the book:



Chapter 2

Chapter two gave a very quick run-through of the Python programming language. It showed off many different Pythonic elements, such as list comprehension and modularity. However, all of the code was written in Python 2.7. For this class, we are using Python 3.5 and were tasked with changing any code that may have been written improperly and would not work in Python 3.5.

Whitespace Formatting

The first section in chapter two went over whitespace formatting. Whitespaces allow python to be much more readable. This section also introduced a frequent difference that will be seen between Python 2.7 and 3.5. The print function in 3.5 requires parentheses, while in 2.7 is not required. There was a small section of code that required fixing, seen below:

```
#Broken Code
for i in [1,2,3,4,5]:
    print i
    for j in [1,2,3,4,5]:
        print j
        print i+j
    print i
print "done looping"

#Fixed Code
for i in [1,2,3,4,5]:
    print(i)
    for j in [1,2,3,4,5]:
        print(j)
        print(i+j)
    print(i)
print("Done Looping")
```

Modules

This section showed the modularity of Python and how to import specific items from different modules. Again, the book utilized the 2.7 methodology of print, resulting in broken code for 3.5. This is seen below:

```
#Broken Code
```

```
match = 10
from re import *
print match

#Fixed Code
match = 10
from re import *
print(match)
```

Arithmetic

In this section, it was noted that Python 2.7 uses integer division by default, requiring a line of code to be included at the beginning of every file. This is not the case in Python 3.5, resulting in the omission of this line from the files used throughout the rest of this class.

Functions

This section demonstrated the use of functions in Python, showing how to create functions and use them. It showed also how to use default values for arguments passed into a function. The 2.7 print function was used in this section, resulting in the following broken code and fix:

```
#Broken Code
def my_print(message="my default message"):
    print message

my_print("hello")
my_print()

#Fixed Code
def my_print(message="my default message"):
    print(message)

my_print("hello")
my_print()
```

Strings

Strings displayed the basic properties of Python string objects. They can be declared with either single or double quotes, allowed for the opposite quotation to be used within the string. It also displayed how to use raw strings and multiline strings. This section had no errors.

Exceptions

Exceptions help to prevent programs from crashing, instead allowing for errors within programs to be handled gracefully. This requires use of try catch blocks. If a computation or declaration within the try block throws an error, the program will catch it, running the catch block, which will usually display a message for the user to fix the problem. This section had a print error seen below:

```
#Broken Code
try:
    print 0/0
except ZeroDivisionError:
    print "cannot divide by zero"

#Fixed Code
try:
    print(0/0)
except ZeroDivisionError:
    print("cannot divide by zero")
```

Lists

Lists are the most fundamental data structure within Python. It is an ordered collection, similar to an array but with more functionality. Arrays are more rigid in structure, while lists allow many more functions to be used with the structure. This section had an error with ranges. In Python 2.7, the built in range function returns a list of the values within the range. However, in Python 3.5, the function returns a range object. In order to fix this, the range must be converted to a list. The broken code along with its fix is seen below:

```
#Broken Code
x = range(10)
zero = x[0]
one = x[1]
nine = x[-1]
eight = x[-2]
x[0] = -1

#Fixed Code
x = list(range(10))
zero = x[0]
one = x[1]
nine = x[-1]
```

```
eight = x[-2]
x[0] = -1
```

Tuples

Tuples are similar to lists. However, while lists are allowed to be changed and are 'mutable', tuples cannot be changed and are 'immutable'. Tuples are often used to return multiple values from a function. In showing the immutablity of tuples, the book used a 2.7 print, resulting in broken code seen below:

```
#Broken Code
my_list = [1,2]
my_tuple = (1,2)
other_tuple = 3,4
my_list[1] = 3
try:
    my_tuple[1]=3
except TypeError:
    print "Cannot modify a tuple"

#Fixed Code
my_list = [1,2]
my_tuple = (1,2)
other_tuple = 3,4
my_list[1] = 3
try:
    my_tuple[1]=3
except TypeError:
    print("Cannot modify a tuple")
```

Dictionaries

Dictionaries are another important data structure. Whereas lists and tuples use indices as keys for accessing their data, dictionaries require specific keys to be given with their values. While demonstrating a KeyError, in which a key is not actually within a dictionary, the 2.7 print function was used.

```
#Broken Code
try:
    kates_grade = grades["Kate"]
except KeyError:
    print "no grade for Kate!"
```

```
#Fixed Code
try:
    kates_grade = grades["Kate"]
except KeyError:
    print("no grade for Kate!")
```

Control Flow

Control flow is fundamental in all languages, requiring conditions for specific actions to occur, creating the logic of a program. This showed how if, elif, else, while, and for are all used within Python. There was a small print error in this code.

Truthiness

This showed how Boolean operations can be assigned to variables. It showed what values are generally accepted as false values. Multiple different methods for doing operations were displayed along with which were more Pythonic than others. There were no errors in this code.

Sorting

Sorting began the more difficult section of this chapter. It showed an important differentiation between sort and sorted. Sorted does not modify the original list while sort while actually change the list. The sort functions within python have multiple different keys that can be used to change how the sort works, along with lambda functionality for values that may not be as easy to sort. There were no errors in this code.

List Comprehensions

List comprehensions are very import in changing one list to another list. This can be anything from selecting some of the values in the list and transferring them to another to modifying all of the values within a list. Comprehensions can make use of multiple for loops, and loops that use results from an earlier loop. These will be used largely throughout the course. There were no errors in this code.

Generators and Iterators

Generators are used when the value within a list is only needed for short period of time. These are useful in cases where very large amounts of numbers are calculated and a list would be inefficient and sluggish to use. This section contained no errors.

Randomness

Random values are often used to test programs by providing a seemingly random set of values to be fed into a program. Python randomness allows for generation of random values with or without a seed. It also can make random decisions and randomize lists as needed. There were no errors in this list.

Regular Expressions

Regular expressions provide a way to check text in a large number of ways. The book does not go into much detail about the use of regular expressions. The different searches can be increasingly complex, and it was state that within the book, any needed regular expressions would be explained. There was a 2.7 print error in this section.

Object-Oriented Programming

This goes over the definition of classes and encapsulations of data and functions. It goes over an example of how to code a class, along with examples of how to use the coded class. This section had a small print error in showing how to use the class, seen below:

```
#Broken code:
s = Set([1,2,3])
s.add(4)
print s.contains(4)
s.remove(3)
print s.contains(3)

#Fixed code:
s = Set([1,2,3])
s.add(4)
print(s.contains(4))
```

```
s.remove(3)
print(s.contains(3))
```

Functional Tools

Functional tools allow for new partially completed functions to be created from existing functions or for functions to be applied to entire lists. This makes use of several different tools from functools. This includes partial, map, reduce, and filter. There were a few errors in this section. The map and filter functions create map and filter objects in Python 3.5 whereas it used to create lists within Python 2.7. Reduce also needed to imported and was missing from the 2.7 code in the book. The broken code with fixes is seen below:

```
#Broken Code
from functools import partial

def double(x):
    return x*2

xs = [1,2,3,4]
twice_xs = [double(x) for x in xs]
twice_xs = map(double,xs)
list_doubler = partial(map, double)
twice_xs = list_doubler(xs)

def is_even(x):
    return x % 2 == 0

x_evens = [x for x in xs if is_even(x)]
x_evens = filter(is_even, xs)
list_evener = partial(filter, is_even)
x_evens = list_evener(xs)

def multiply(x, y):
    return x*y

x_product = reduce(multiply, xs)


#Fixed Code
from functools import partial

def double(x):
    return x*2

xs = [1,2,3,4]
twice_xs = [double(x) for x in xs]
```

```
twice_xs = list(map(double,xs))
list_doubler = partial(map, double)
twice_xs = list(list_doubler(xs))

def is_even(x):
    return x % 2 == 0

x_evens = [x for x in xs if is_even(x)]
x_evens = list(filter(is_even, xs))
list_evener = partial(filter, is_even)
x_evens = list(list_evener(xs))

def multiply(x, y):
    return x*y

from functools import reduce

x_product = reduce(multiply, xs)
```

Enumerate

Enumerate allows a user to both iterate over an object and use whatever content is within the object. This is done by returning both the index of a value and the value within a tuple for use by the user. This section does not contain any errors.

Zip and Argument Unpacking

Zipping and unpacking arguments is used to combine lists together or to take them apart into multiple different objects or lists. Argument unpacking is done using the zip function, except with an asterisk to denote unpacking rather than zipping. This section had the same issue as mapping and filters, requiring the zip/unpacked object to be converted to a list. This is seen below.

```
#Broken Code
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
zip(list1, list2)

zip(('a', 1), ('b', 2), ('c', 3))

#Fixed Code
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
list(zip(list1, list2))

tuple(zip(('a', 1), ('b', 2), ('c', 3)))
```

Args and kwargs

Args and kwargs shows how to create functions that are able to take multiple arguments within functions that have keys and that do not have keys. This section had a few print errors, but overall worked fine.

Chapter 3

Chapter 3 began to show how to matplotlib.pyplot and the various ways that it can be used. Pyplot allows data to be easily manipulated a visualized on screen. Figure can be displayed and shown via use of its functionality. Visualizing data is a key element in analyzing it, along with other functions that make that analysis easier. A first example was given, displaying the following:

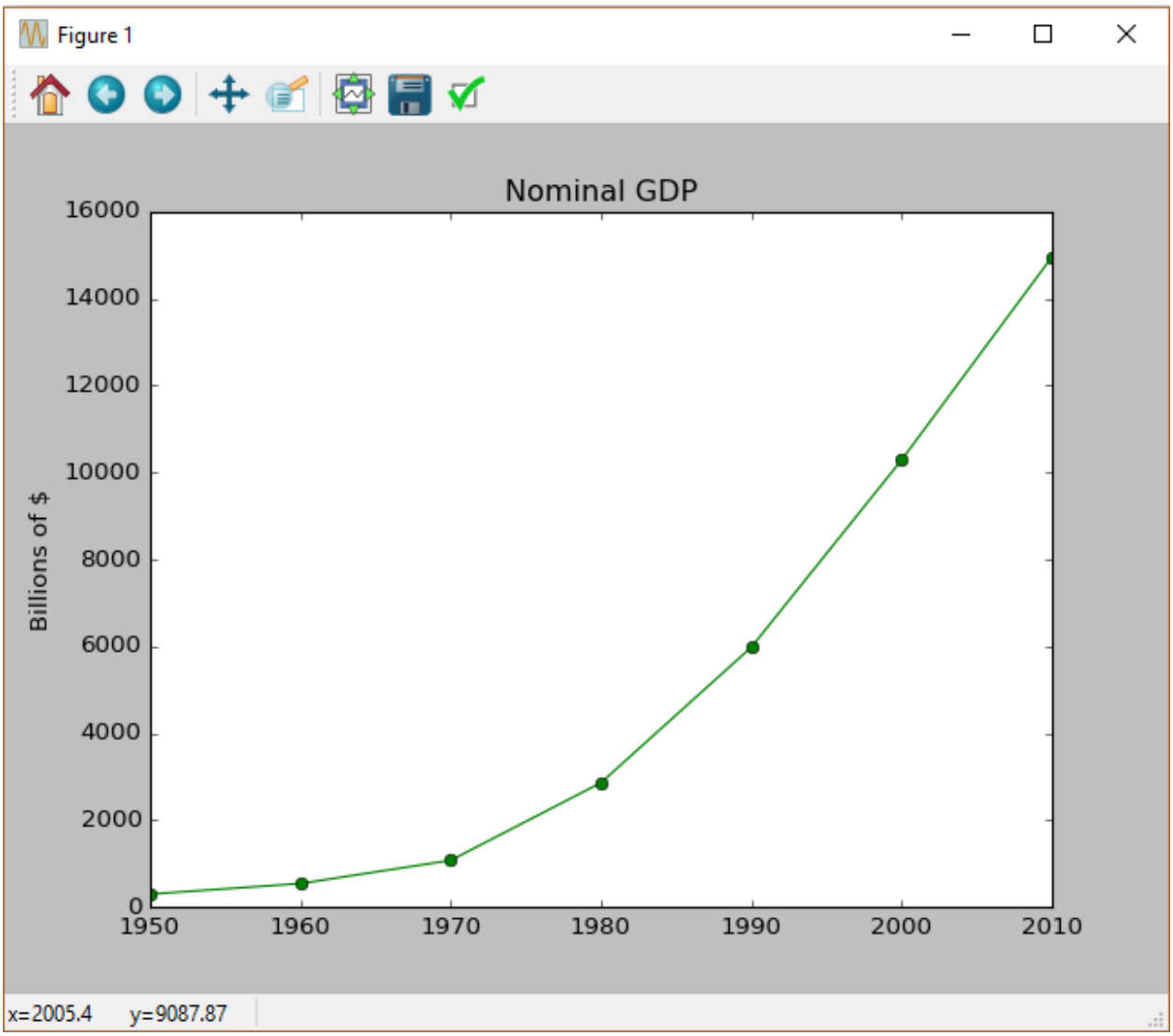


Figure 1 - Simple Line Graph

This graph is just an example of the many different types of graphs that can be used. This is a simple line graphed using seven points of data to display an upward trend in GDP. Other examples included showed how to use bar graphs, how to modify axes, how to plot multiple lines on a graph and how to plot a scatter plot. These are seen in the following figures.

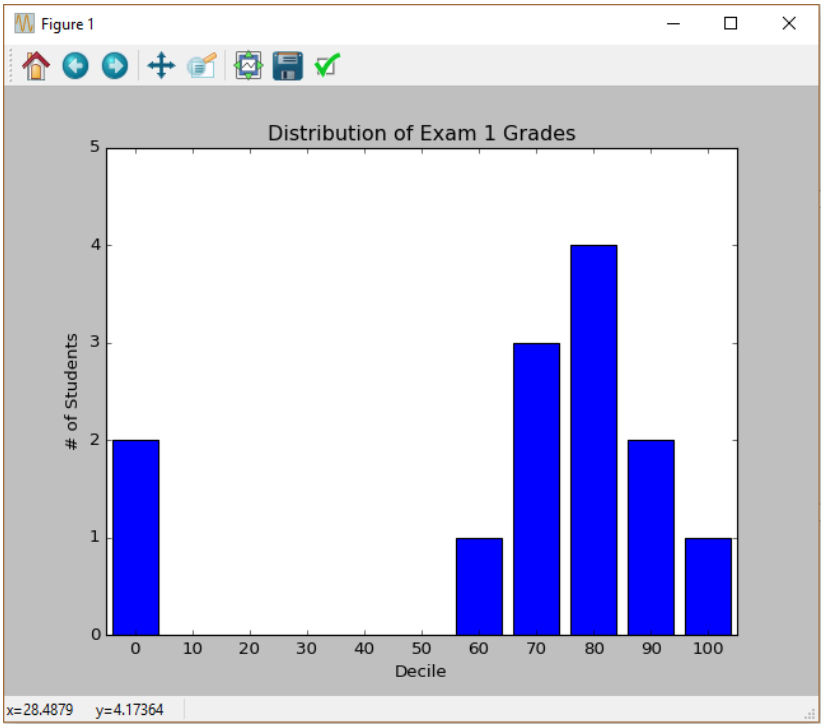


Figure 2 - Simple Bar Graph

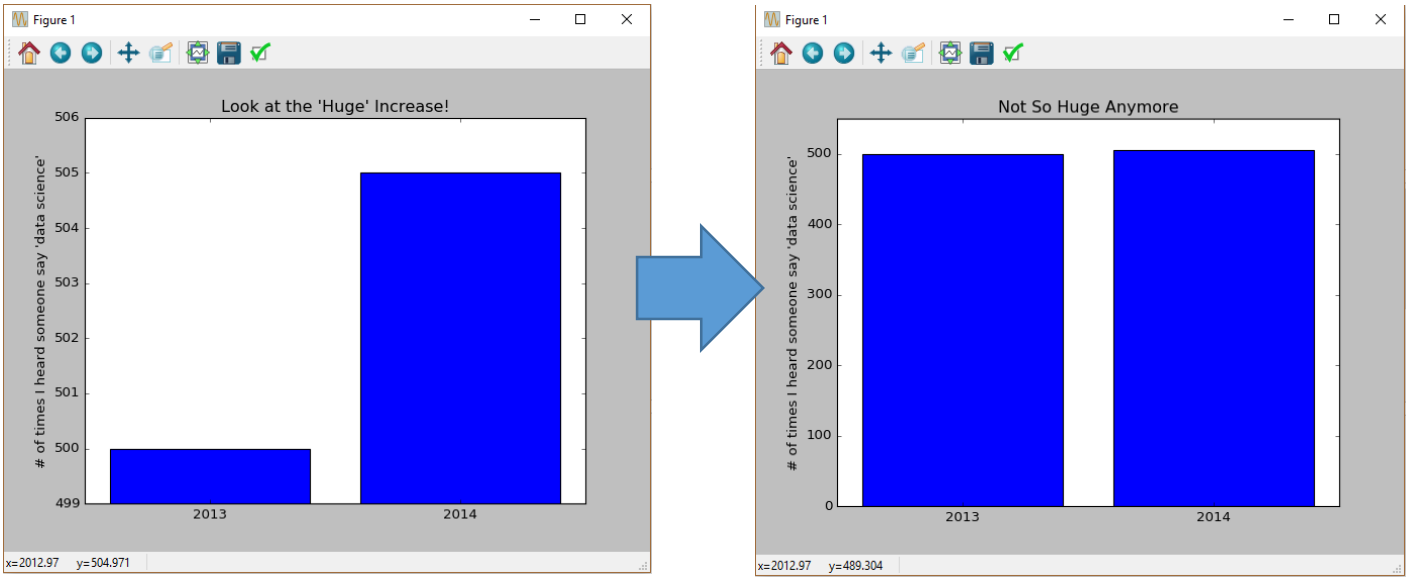


Figure 3 - This displays how axes can cause data to be misrepresented when differences between the data may not be so large.

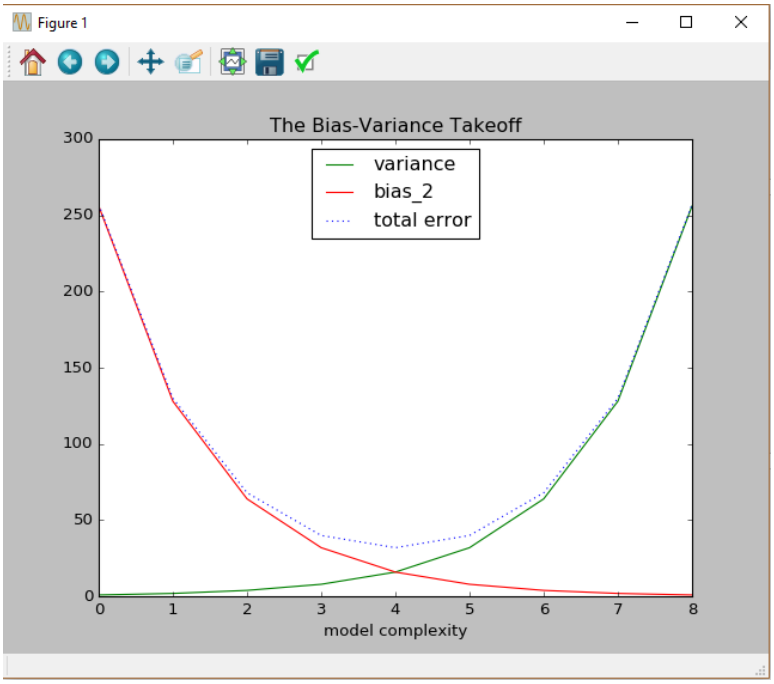


Figure 4 - This graphs shows mutiple lines being displayed on a single graph.

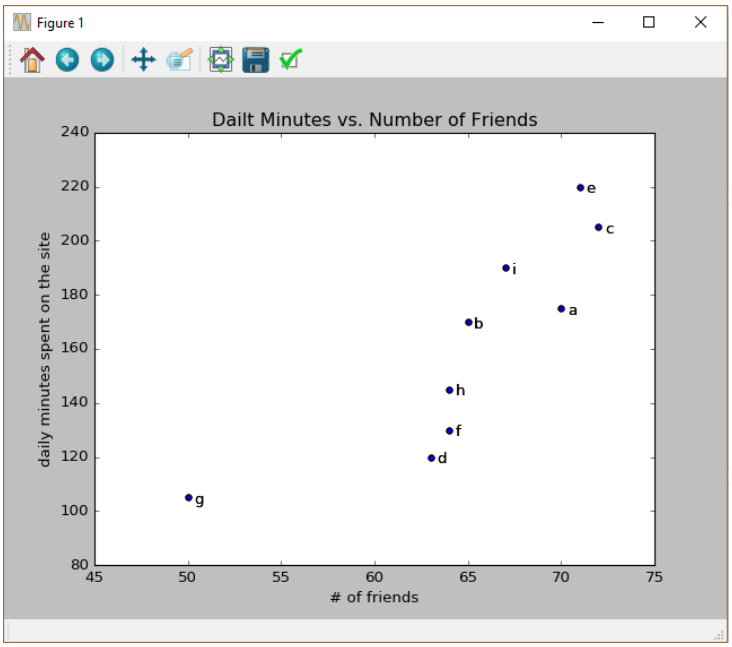


Figure 5 - This graph displays a scatter plot, each individual point representing a person and their number of friends based on the number of minutes they spend on a website.

Extended Code

I did not have any extended code from this chapter. However, I do have code that has made use of pyplot and data manipulation, putting the data into a much easier to visual graph. It made use of inline plots within an iPython notebook, with data taken from data.gov. It shows all infectious diseases over the course of 2001 to 2014. Analysis was completed to display the diseases based off of both men and women, the percentage of the population affected, and the average, with a few other minor statistics. Images of the graphs, using Bokeh, are seen below.

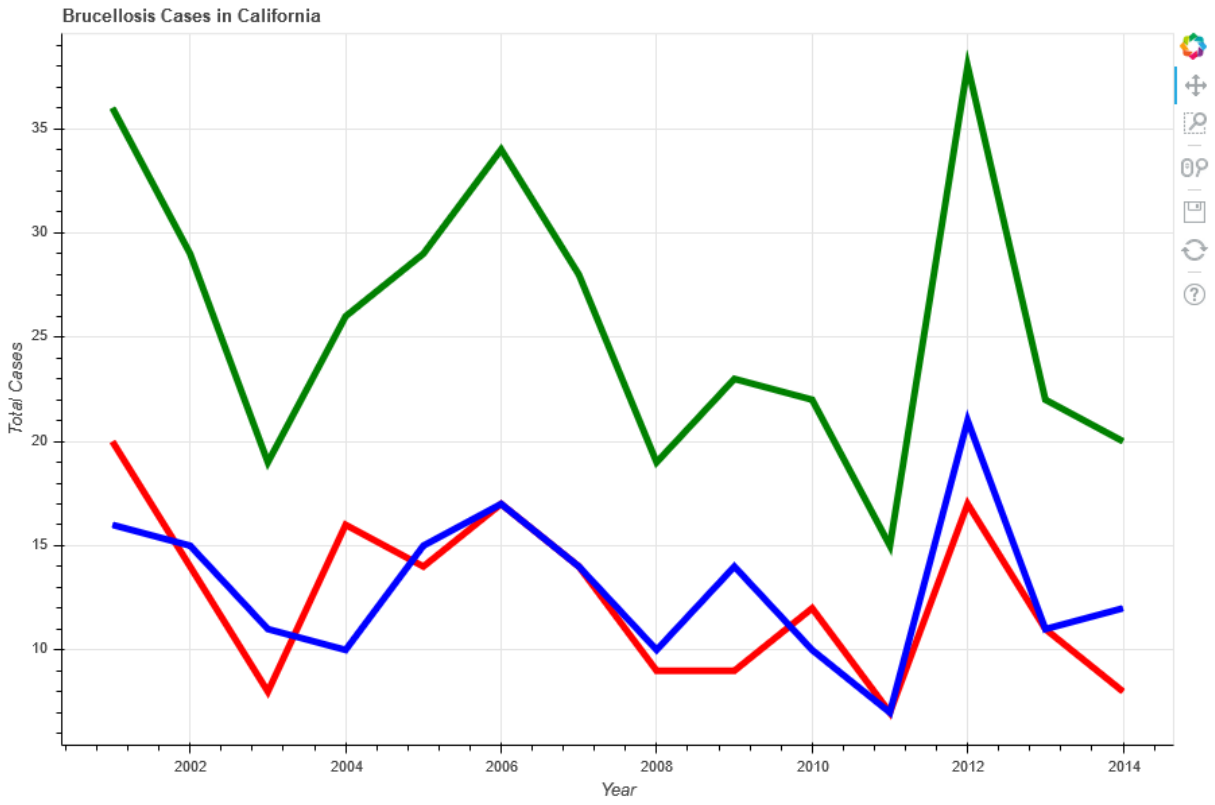


Figure 6 - Displays the graph showing the disease brucellosis over the years. Green represents the total population, blue represent men, and red represents women.

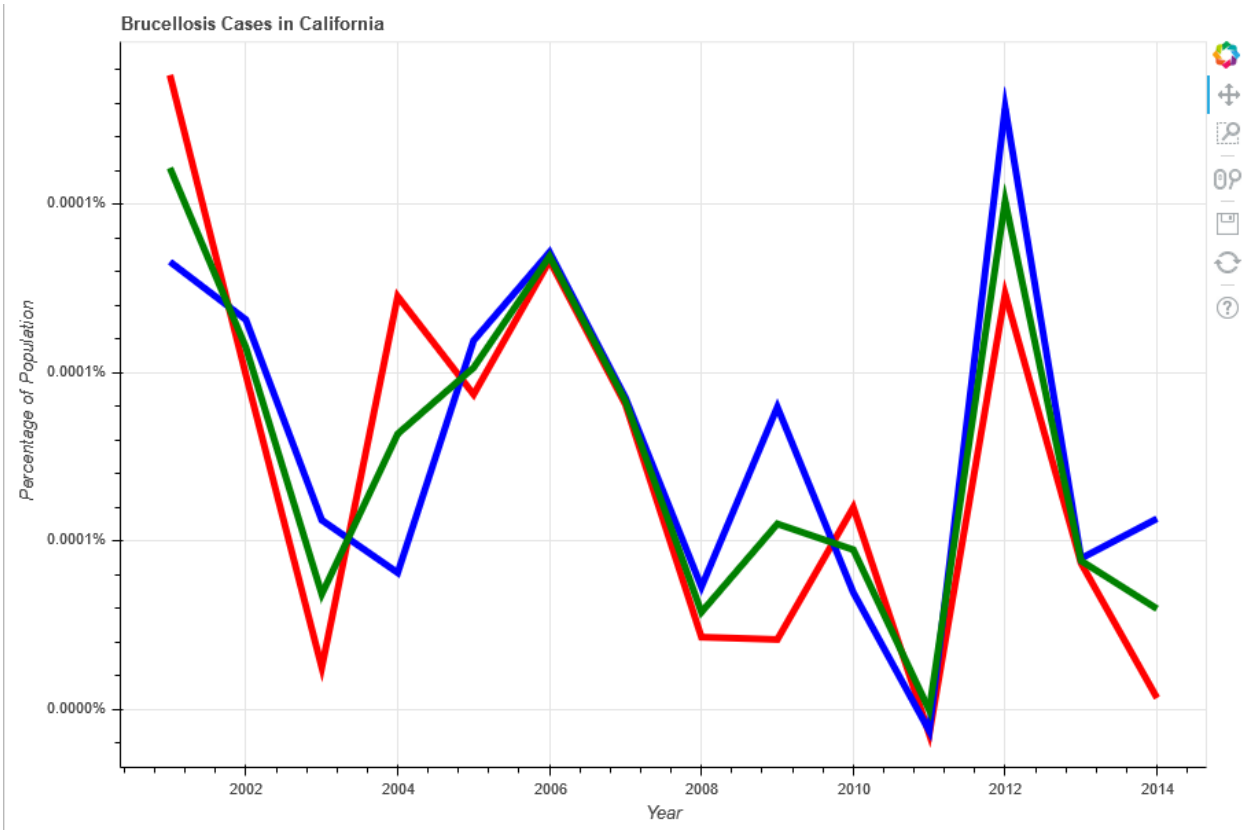


Figure 7 - Percentage of the population affected by Brucellosis in California

Discussion

This has largely been a review for me, as I have used Python in the past for several different project. For creating my code and graphs, I utilized iPython notebooks for each of the different chapters. This allowed for easy integration of code into explanations as well. This was also how I had done my prior classes. I will likely continue to use this medium for generating my code for later chapters.

Indirect Logs

Date	Minutes	Activity Description
9/5/2016	180	Went through chapter 1. Began Chapter 2 code.

9/10/2016	120	Worked on Chapter 2 code.
9/11/2016	600	Finished Chapter 2 code and Chapter 3 code. Began report.
9/12/2016	180	Finished Report.
Total	1080	

Code

```
from matplotlib import pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 542.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]

plt.plot(years, gdp, color='green', marker='o', linestyle='solid')

plt.title("Nominal GDP")

plt.ylabel("Billions of $")
plt.show()

movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story" ]
num_oscars = [5, 11, 3, 8, 10]

xs = [i + 0.1 for i, _ in enumerate(movies)]

plt.bar(xs, num_oscars)

plt.ylabel("# of Academy Awards")
plt.title("My Favorite Movies")

plt.xticks([i + 0.5 for i, _ in enumerate(movies)], movies)

plt.show()

from collections import Counter
grades = [83, 95, 91, 87, 70, 0, 85, 82, 100, 67, 73, 77, 0]
decile = lambda grade: grade // 10*10
histogram = Counter(decile(grade) for grade in grades)
plt.bar([x-4 for x in histogram.keys()], histogram.values(), 8)
plt.axis([-5, 105, 0, 5])

plt.xticks([10*i for i in range(11)])
plt.xlabel("Decile")
```

```
plt.ylabel("# of Students")
plt.title("Distribution of Exam 1 Grades")
plt.show()

mentions = [500, 505]
years = [2013, 2014]

plt.bar([2012.6, 2013.6], mentions, 0.8)
plt.xticks(years)
plt.ylabel("# of times I heard someone say 'data science'")

plt.ticklabel_format(useOffset=False)

plt.axis([2012.5, 2014.5, 499, 506])
plt.title("Look at the 'Huge' Increase!")
plt.show()

mentions = [500, 505]
years = [2013, 2014]

plt.bar([2012.6, 2013.6], mentions, 0.8)
plt.xticks(years)
plt.ylabel("# of times I heard someone say 'data science'")

plt.ticklabel_format(useOffset=False)

plt.axis([2012.5, 2014.5, 0, 550])
plt.title("Not So Huge Anymore")
plt.show()

variance = [1,2,4,8,16,32,64,128,256]
bias_squared = [256, 128, 64, 32, 16, 8, 4, 2, 1]
total_error = [x + y for x, y in zip(variance, bias_squared)]
xs = [i for i, _ in enumerate(variance)]

plt.plot(xs, variance, 'g-', label='variance')
plt.plot(xs, bias_squared, 'r-', label='bias_2')
plt.plot(xs, total_error, 'b:', label='total error')

plt.legend(loc=9)
plt.xlabel("model complexity")
plt.title("The Bias-Variance Takeoff")
plt.show()

friends = [70, 65, 72, 63, 71, 64, 50, 64, 67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

plt.scatter(friends, minutes)

for label, friend_count, minute_count in zip(labels, friends, minutes):
    plt.annotate(label, xy=(friend_count, minute_count),
                 xytext=(5, -5),
                 textcoords='offset points')
plt.title("Dailt Minutes vs. Number of Friends")
```

```
plt.xlabel("# of friends")
plt.ylabel("daily minutes spent on the site")
plt.show()
```

Extended Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import bokeh.plotting as bkh
import collections

from bokeh.models import NumeralTickFormatter
from ipywidgets import Dropdown
from IPython.display import display
from IPython.display import clear_output

bkh.output_notebook()
%matplotlib inline

filename = "Infectious_Disease_Cases_by_County__Year__and_Sex__2001-2014.csv"
df = pd.read_csv(filename, index_col = ("Disease", "Year", "Sex"))
df2 = pd.read_csv(filename)
df = df.loc[df['County'] == "California"]
df = df.unstack("Sex")
df1 = df[["Count"]]
df3 = df[["Population"]]
x = pd.unique(df2.Disease.ravel())
y = collections.OrderedDict(sorted(dict(zip(x,x)).items()))

dw = Dropdown(options=y)

def on_value_change(name, val):
    clear_output();
    df4 = df1.loc[val]["Count"]/df3.loc[val]["Population"]
    p = bkh.figure(title=val+" Cases in California",
                   plot_width=900,
                   plot_height=600,
                   x_axis_label="Year",
                   y_axis_label="Total Cases")
    p.multi_line(xs=[df1.loc[val].index.get_level_values("Year").unique()]*3,
                 ys=[df1.loc[val]["Count"][name].values for name in
df1.loc[val]["Count"]],
                line_color=["Red", "Blue", "Green"],
                line_width=5)
    p2 = bkh.figure(title=val+" Cases in California",
                   plot_width=900,
                   plot_height=600,
                   x_axis_label="Year",
                   y_axis_label=" Percentage of Population")
    p2.multi_line(xs=[df4.index.get_level_values("Year").unique()]*3,
                 ys=[df4[name].values for name in df4],
                line_color=["Red", "Blue", "Green"],
                line_width=5)
```

```
    p2.yaxis[0].formatter = NumeralTickFormatter(format="0.0000%")

    bkh.show(p)
    display(df1.loc[val].describe())
    bkh.show(p2)
    display(df4)


dw.on_trait_change(on_value_change, 'value')

dw.value = dw.options["Amebiasis"]

display(dw)
```