Assignment 01: Plotting and Graphing using Python

Scott Merkley

January/29/2023

Department of Computer Science, University of Utah, Salt Lake City, 84112, UT, USA

Introduction

In this report I will be explaining results and figures from the written code as well as answering the given questions of Assignment 01 for CS 5635 at the University of Utah. Python was the coding language chosen to be used for this assignment. Packages such as [1], Matplotlib [2], Array, Pandas [3], and Plotly [4] were used to make arrays, manipulate data, and create visualizations of that data. Stackoverlow [5] was the primary reference source as well as Matplotlib, Pandas, and Plotly websites.

Part 1

We were first asked to create two arrays one of 100 numbers uniformly distributed between [0, 1], the other of 200 numbers with Gaussian distribution between [1, 100]. The imported libraries were as such:

```
# Import all libraries
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd
from array import array
import matplotlib.cm as cm
import pandas as pd
import plotly.express as pltex
import plotly.graph_objects as pltgo
```

Then the arrays were created:

```
# Creat an array of 100 random numbers uniformly distributed between [0, 1]
uniformArray = rnd.uniform( 0, 1, 100 )

# Create a second array of 200 random numbers with Gaussian Distribution
# between [1, 100]
gaussianMean = 10
gaussianArray = rnd.normal( gaussianMean, np.sqrt( gaussianMean ), 200 )
```

We were then asked to create a box plot of the arrays. This was done using Matplotlib's boxplot method.

```
# Creat a box plot for visualization of both arrays
plt.title( "Box Plot Visualization: Uniform Distribution" )
plt.xlabel( "" )
plt.ylabel( "Distributed Area" )
plt.boxplot( uniformArray )
plt.grid( linestyle = '--' )
plt.show()

plt.title( "Box Plot Visualization: Gaussian Distribution" )
plt.xlabel( "" )
plt.ylabel( "Distributed Area" )
plt.boxplot( gaussianArray )
```

```
plt.grid( linestyle = '--' )
plt.show()
```

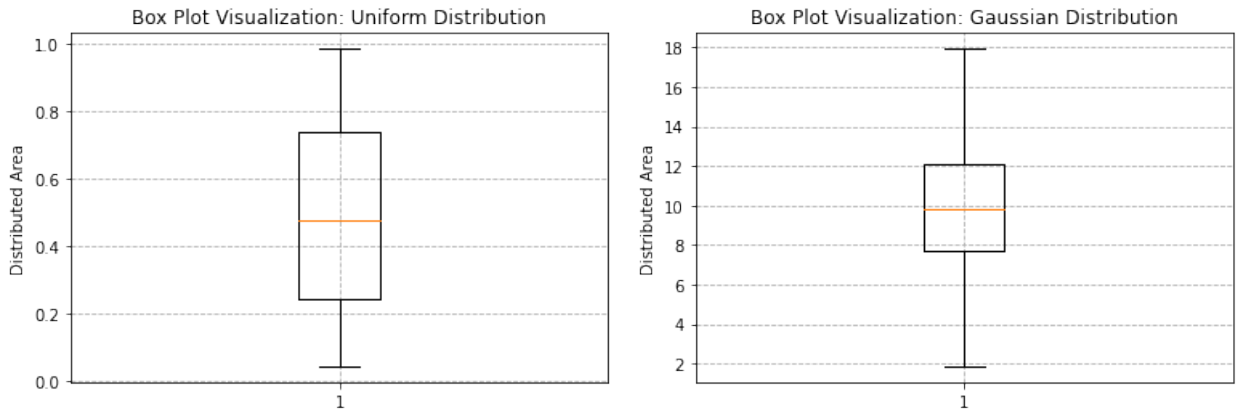

Figure 1: Uniform and Gaussian Distribution Box Plots

The plots generated show the distribution over the area and the relative grouping of points. The uniform distribution has more spread over the entire area that was given, the Gaussian distribution has less spread but still covers a large area with fewer points at the edges.

We then created a histogram of the data with 20 bins using a bar chart. We were not allowed to use any histogram libraries so a MakeHistogram helper method was made that intakes an array and a number of bins and computes and returns two arrays. The first array has the histogram indices (the x coordinates), the second array contains the bins of the indices (the y coordinates). Which were as follows:

```
# MakeHistogram is a function that takes in an array to be made into a histogram the
# number of bins for the histogram while creating and plots a histogram array
# with the number of bins.
#
# @param arr            - an array to be made into a histogram
# @param numberOfBins   - number of bins for the array
# @return               - array containing histogram indices array and bins array
def MakeHistogram (arr, numberOfBins):
    histArray = np.zeros(numberOfBins)
    histIndices = np.zeros(numberOfBins)
    for i in range(1, len(histArray) + 1):
        lowerLimit = (i - 1) / len(histArray) * max(arr)
        upperLimit = i / len(histArray) * max(arr)
        histIndices[i - 1] = lowerLimit
        for data in arr:
            if data > lowerLimit and data < upperLimit:
                histArray[i - 1] += 1
    return np.array([histIndices, histArray])

uniformHistArray  = MakeHistogram( uniformArray, 20 )
gaussianHistArray = MakeHistogram( gaussianArray, 20 )

plt.title( 'Histogram: Uniform vs. Gaussian Distributions' )
plt.xlabel('Bins')
plt.ylabel('Number of Items in Bin')
plt.bar(uniformHistArray[0], uniformHistArray[1], width = 0.05 * max(gaussianArray),
                                                  zorder = 5, label = 'Uniform', color = '
                                                  orange' )
plt.bar(gaussianHistArray[0], gaussianHistArray[1], width = 0.05 * max(gaussianArray),
                                                  zorder = 5, label = 'Gaussian', color = '
                                                  dodgerblue' )
plt.grid( linestyle = '--', zorder = 0 )
plt.legend()
plt.show()
```

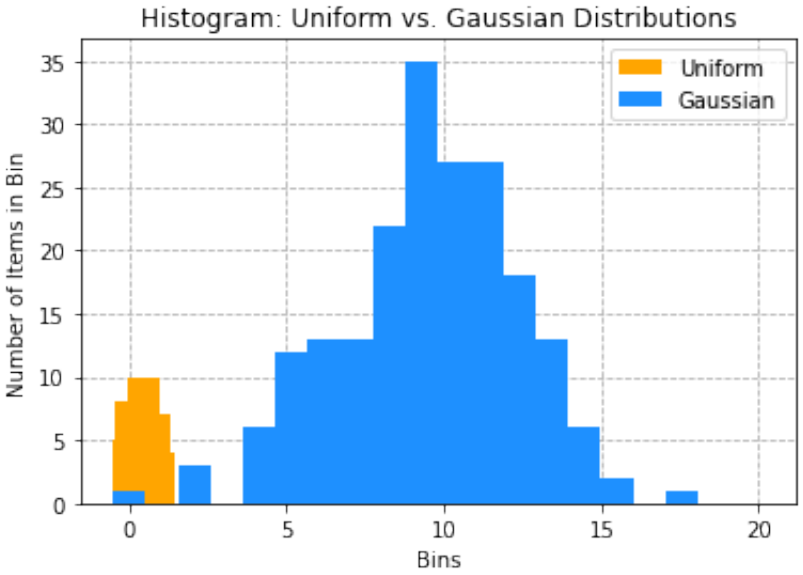

Figure 2: Histogram of Uniform and Gaussian Distributions

Since 200 bins were used for the Gaussian distribution and 100 for the uniform, the Gaussian distribution shows much more prominently than the uniform.

Next, we were asked to write the distribution arrays to a binary file, read the file back to an array, and visualize the data using the cumulative distribution function (CDF) as a line graph. The writing and reading of binary files was done using the Array package.

```
# ArrayToBinary takes an input array and a file name and converts the given array
# to a binary file.
#
# @param arr - array to convert to binary file
# @fileName  - name of file
def ArrayToBinary(_arr, _fileName):
    outputFile = open(_fileName, 'wb')
    _arr.tofile(outputFile)
    outputFile.close()

# BinaryToArray takes a file name reading the file and returning an array.
#
# @fileName - name of file
# return    - array from file
def BinaryToArray(_fileName):
    inputFile = open(_fileName, 'rb')
    arr = array('d')
    arr.frombytes(inputFile.read())
    arr = np.array(arr)
    return arr

ArrayToBinary(uniformArray, 'uniformArray')
ArrayToBinary(gaussianArray, 'gaussianArray')

uniformArray = BinaryToArray('uniformArray')
gaussianArray = BinaryToArray('gaussianArray')
```

The cumulative distribution function shows whether the next point in the array will be greater than or less than the previous point. This is done essentially by summing up the percentages of the current point divided by the previous point which always equals 1 in the end.

```
# Plot the CDF of both distributions
plt.title( "CDF: Uniform vs. Gaussian Distribution" )
plt.xlabel('Data')
plt.ylabel('Probability')
plt.plot(np.arange(len(uniformArray)) / (len(uniformArray) - 1), np.sort(uniformArray) /
                                                    max(uniformArray), zorder = 5, label = '
                                                    Uniform', c = 'orange')
plt.plot(np.arange(len(gaussianArray)) / (len(gaussianArray) - 1), np.sort(gaussianArray
                                                    ) / max(gaussianArray), zorder = 5, label
                                                    = 'Gaussian', c = 'dodgerblue')
plt.grid(linestyle = '--', zorder = 0)
plt.legend()
plt.show()
```

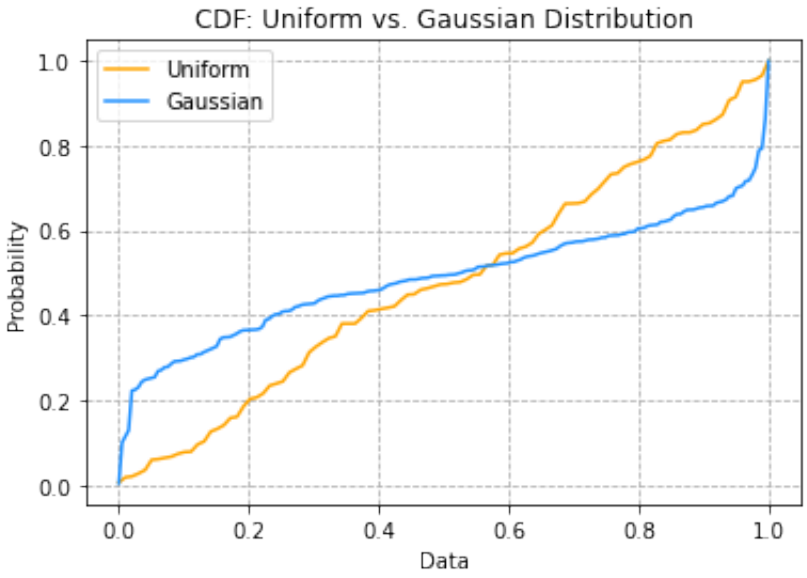

Figure 3: CDF of Uniform and Gaussian Distributions

We were then asked to generate 2D arrays with uniform random sampling and Gaussian random sampling, containing 5000 points from [0, 1]x[0, 1]. The visualization method for these was to be a scatter plot.

```
# Create 2D arrays using random sampling and gaussian random sampling with 5,000 points
uniform2D = rnd.sample((2, 5000))
mean = 0.5
gaussian2D = rnd.normal(mean, 0.1, (2, 5000))
plt.title('Scatter Plot: Uniform vs. Gaussian Distribution')
plt.xlabel('X')
plt.ylabel('Y')
plt.scatter(uniform2D[0], uniform2D[1], c = 'orange', zorder = 5, s = 5, label = '
                                                    Uniform')
plt.scatter( gaussian2D[0], gaussian2D[1], zorder = 5, s = 5, c = 'dodgerblue', label =
                                                    'Gaussian' )
plt.legend(loc = 'upper right')
plt.show()
```

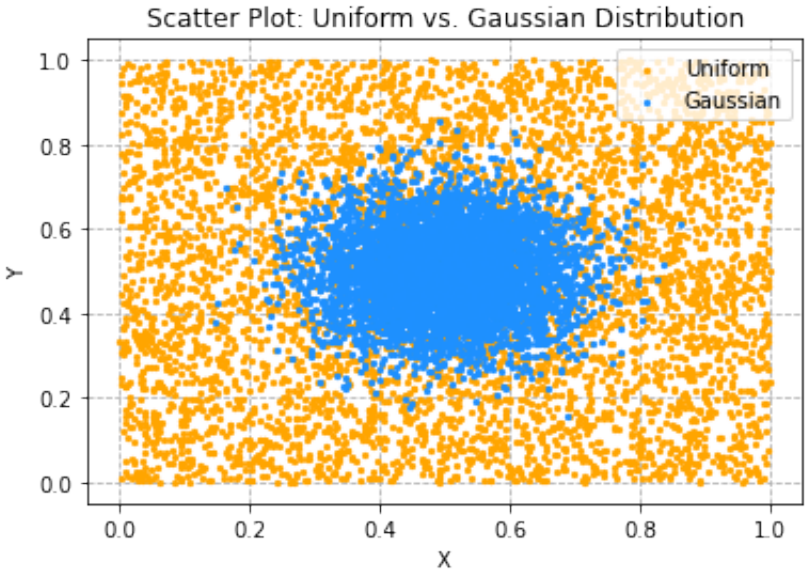

Figure 4: 2D Scatter Plot of Uniform and Gaussian Distributions

The Gaussian distribution is shown to be located tightly around the center [0.5, 0.5] with a slight spread. The uniform distribution covers the whole area evenly with some slight white space.

For the last of Part 1 we were asked to generate a 2D histogram with the dimensions of 100x100, show the arrays as images, and then plot the sampled arrays as contour plots with 10 levels.

```
# Generating Arrays and Array Images
low = 0
high = 1
numberOfPoints = 5000
counts, xBins, yBins, image = plt.hist2d( rnd.uniform( low, high, numberOfPoints ), rnd.
                                          uniform( low, high, numberOfPoints ), bins
                                           = 100 )
plt.title( 'Uniform Distribution 2D Histogram' )
plt.xlabel( 'X' )
plt.ylabel( 'Y' )
plt.show()

mean = 0.5
spread = 0.5
counts, xBins, yBins, image = plt.hist2d( rnd.normal( mean, spread, numberOfPoints ),
                                          rnd.normal( mean, spread, numberOfPoints )
                                          , bins = 100 )
plt.title( 'Gaussian Distribution 2D Histogram' )
plt.xlabel( 'X' )
plt.ylabel( 'Y' )
plt.show()
```

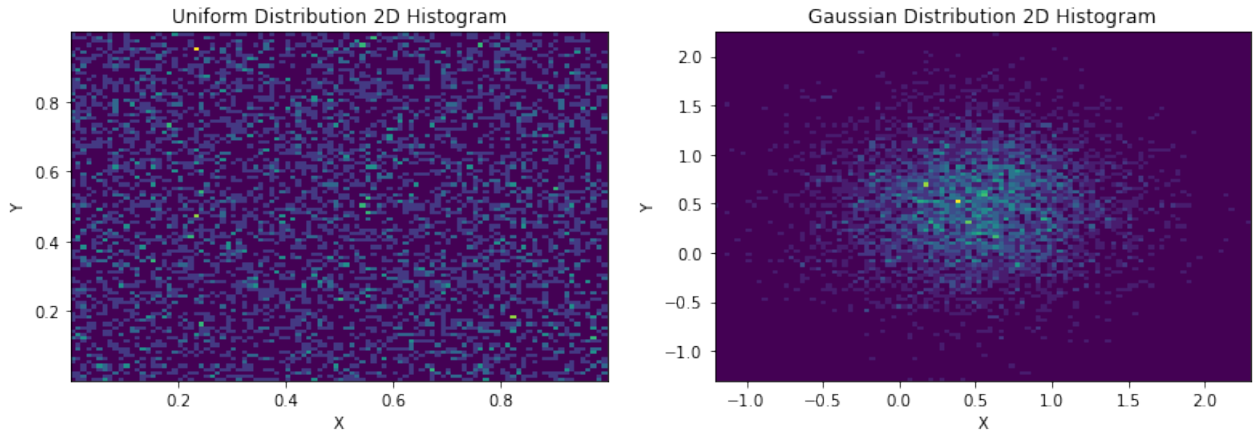

Figure 5: Uniform and Gaussian Distribution 2D Array Images

The contour plots:

```
    # Generating Contour Plots
    plt.contourf( counts, extent = [0, 1, 0, 1], zorder = -1 , cmap = cm.jet, levels = np.
                                                    linspace(counts.min(), counts.max(), 10) )
    plt.title( 'Uniform Distribution 2D Histogram Contour' )
    plt.grid( color = 'w', linestyle = '--', zorder = 0 )
    plt.xlabel( 'Uniform X Bin Level' )
    plt.ylabel( 'Uniform Y Bin Level' )
    plt.colorbar()
    plt.show()

    plt.contourf( counts, extent = [0, 1, 0, 1], zorder = -1 , cmap = cm.jet, levels = np.
                                                    linspace(counts.min(), counts.max(), 10) )
    plt.title( 'Gaussian Distribution 2D Histogram Contour' )
    plt.grid( color = 'w', linestyle = '--', zorder = 0 )
    plt.xlabel( 'Gaussian X Bin Level' )
    plt.ylabel( 'Gaussian Y Bin Level' )
    plt.colorbar()
    plt.show()
```

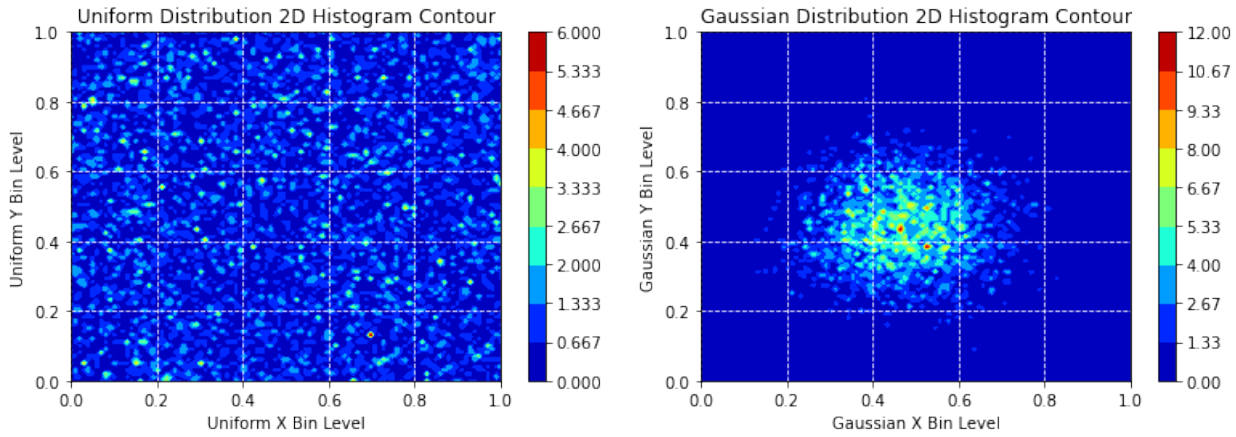

Figure 6: Uniform and Gaussian Distribution Contour Plots

Part 2

For the second part of this assignment, we are downloading data sets and creating visualizations to get a more intuitive view at some of the underlying trends. First, we looked at the NOAA Land Ocean Temperature Anomalies Data Set to create a bar plot showing temperature changes and coloring them red or blue based on whether they were positive or negative respectively.

```
# 1 - Create a box plot for visualization of NOAA Temperatures
tempsNOAA = pd.read_csv('NOAA-Temperatures.csv', header = 4)
plt.bar(np.asarray(tempsNOAA['Year'], int), tempsNOAA['Value'], color = (tempsNOAA['
                                                    Value'] > 0).map({True: 'r', False: 'b'}),
                                                     zorder = 5)
plt.title("NOAA: Global Land and Ocean Temperature Anomalies")
plt.xlabel('Year')
plt.ylabel("Degrees C+/-")
plt.grid(linestyle = '--', zorder = 0)
plt.show()
```

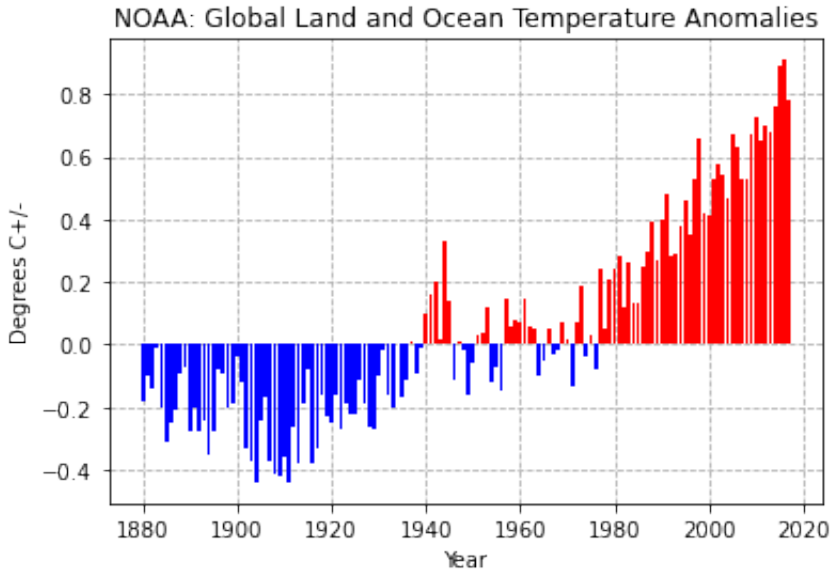

Figure 7: Visualizing NOAA Data Set Using a Bar Graph

The data from Figure 7 seems to have been negatively trending towards colder temperatures from 1880 until about 1910. After 1910, the trend in temperature began to rise until the 1940's when the trend first broke the 0.0 degree equilibrium in the data set. Since about 1975 the trend in temperature has been getting hotter every year, seeing a max of about 0.9 degrees Celsius. The graph looks to not be slowing anytime soon and with a trend that will likely continue.

For the next visualization, we were given a excel spreadsheet containing a list of breakfast cereals and nutritional information about their contents. Some of the nutritional information listed is Calories, Protein, Fat, Sodium, Fiber, Carbohydrates, Sugars, etc. We were asked to make a radar chart containing 8 nutritional statistics for 3 cereals. The statistics are as show:

```
# 2 - Generate a Radar Chart with 8 Nutritional Statistics for 3 Cereals
breakfastCereals = pd.read_excel('Breakfast-Cereals.xls', header = 0)
cerealNames = ['Cinnamon Toast Crunch', 'Trix', 'Cheerios'] # breakfastCereals['Cereal']
categories = ['Calories', 'Sugars', 'Fat', 'Protein', 'Fiber', 'Carbohydrates', '
                                        Vitamins', 'Potassium']
fig = pltgo.Figure()
for cereal in cerealNames:
    fig.add_trace(pltgo.Scatterpolar(r = breakfastCereals.iloc[np.where(breakfastCereals
                                            ['Cereal'] == cereal)][categories].
                                            to_numpy()[0], theta = categories,
                                            fill = 'toself', name = cereal))
fig.update_layout(title = 'Cereal Nutritional Information')
fig.show()
```

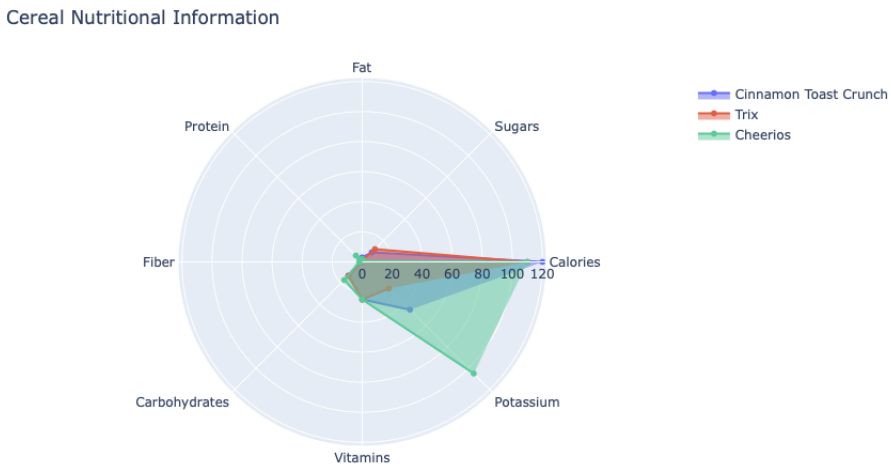

Figure 8: Visualizing Nutritional Statistics of Cinnamon Toast Crunch, Trix, and Cheerios

The radar chart allows it to easily be seen that Cheerios have the most Potassium and Calories while Trix has more sugars, all have similar amounts of carbohydrates and vitamins. This form of chart makes it easy to quickly realize which breakfast cereal contains different amounts of a nutrient but also makes it harder to see some statistics because all the information overlaps each other. This form of visualization works well for small amounts of data, and to more front end users but lacks the ability to show large quantities of data between many different cereals.

For the last question of part 3 we were asked to find two data sets from the repository FiveThirtyEight, plot one using parallel coordinates and the other with a scatter plot. I chose to analyze a data set containing US daily births from 2000-2014 for the first data set, and marital statistics on men for the second. Here are the visualizations.

```
# 3 - Choose two different data sets to visualize from
# https://github.com/fivethirtyeight/data.
# Scatter Plot: US Daily Births from 2000-2014
filename = "https://raw.githubusercontent.com/fivethirtyeight/data/master/births/
                                        US_births_2000-2014_SSA.csv"
data = pd.read_csv(filename)
data['date'] = data['year'].apply(str) + '/' + data['month'].apply(str) + '/' + data['
                                        date_of_month'].apply(str)
data['date'] = pd.to_datetime(data['date'])
data['total_growth'] = data['births']
for i in range(1, len(data['births'])):
    data['total_growth'][i] = data['births'][i] + data['total_growth'][i - 1]
# data.plot.scatter(x = 'date', y = 'births', c = 'births', colormap = 'viridis')
plt.scatter(data['date'], data['births'], c = data['births'])
plt.title('US Births')
plt.xlabel('Year')
```

```
plt.ylabel('Daily Births')
plt.colorbar()
plt.show()

# Parallel Coordinates: Men in Marriage
filename = 'https://raw.githubusercontent.com/fivethirtyeight/data/master/marriage/men.
                                                    csv'
data = pd.read_csv(filename)
data = data.drop('date', axis = 1)
data = data.drop('Unnamed: 0', axis = 1)
columns3544 = [col for col in data.columns if '3544' in col]
columns3544.append('year')
data = data[columns3544]
pd.plotting.parallel_coordinates( data, 'year' ).legend( loc = 'center left',
                                                    bbox_to_anchor = ( 1.05, 0.49 ) )
plt.title( 'Men in Marriage Statistics' )
plt.ylabel( 'Percentage %' )
plt.xlabel( 'Statistic Categories' )
plt.xticks( rotation = 45 )
plt.show()
```

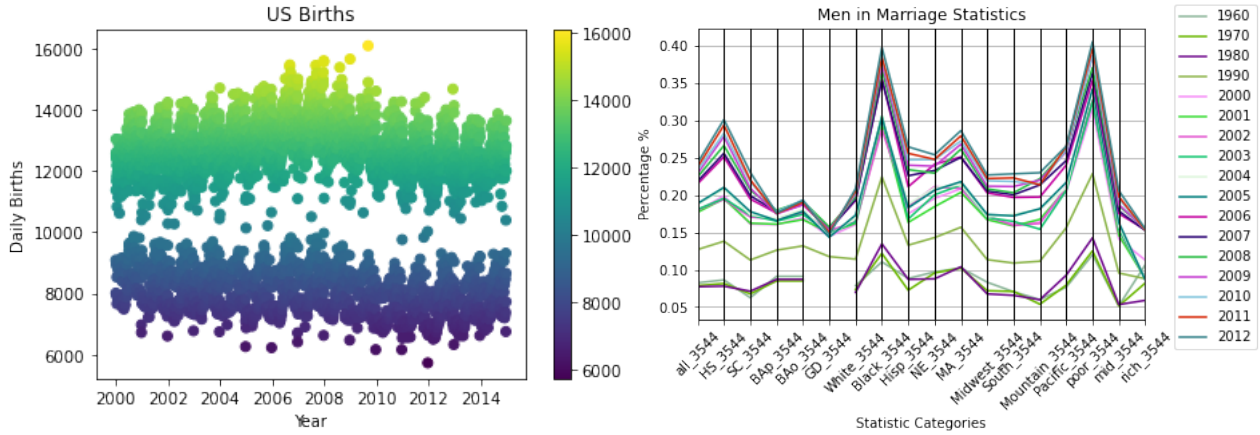

Figure 9: Visualizations of Daily Births in the US from 2000-2014 and Statistics of Men in Marriage

The US daily births shows an odd trend where on any given day the US will not have 10,000 babies born. It is strange that the data is not uniform across the whole plot from 14,000 to 6,000. There could be some reason for the population with a probability of having a baby that lead to the chances of 10,000 to be unlikely but it is unknown from the data and need further research.

Part 3

Question 1: For the final section of this homework, we were asked to read The Value of Visualization by Jarke J. van Wijk and answer four questions about the text. Value of Visualization talked on the importance of visualization and what value they bring. Two measures that help decide the value of visualization are the ability to estimate the amount of time saved (efficiency) and the amount of effort in finding the desired information (effectiveness) in any visualization process. Being human, we don't have endless time to give to any project, and having a visualization that allows you to see an underlying trend faster and with less effort is going to help bring productivity to your workflow and keep the user focused on making the visualization more specific which in turn would allow the for better visualizations, perspective of the data, and knowledge from that data.

Question 2: In figure one there is a block visualization, showing how data is first collected, then visualized, allowing for a image which is perceived by the user leading to a change in knowledge with respect to time. This knowledge can further enhance the perception of the user or lead to interactive exploration. This exploration leads to a change in the specification of the visualization with respect to time which then changes the visualization and a positive feedback loop is created.

This model closely resembles a SIRS model in epidemiology where a certain portion of a population is healthy but is susceptible to being infected by a virus. An individual is at first susceptible, then infected, and after an amount of time the person can go back to being susceptible. These rates of change have different constants attached depending on severity of virus and susceptibility of the individual. Like the model above, this model has a positive feedback. One of the big differences is that the SIRS model shows a decline in the susceptibility of a person over time. This means that someone cannot just become infected over and over again, due to their immune system. Figure 1 seems to be missing such a constant to show that users are not only gaining more knowledge, perception, and interactive exploration. There should also be an option to move out of this model.

Question 3: There are certain costs associated with any visualization technique, here are four to list a few. 1) Number of Users (n), 2) Frequency of Use (m), 3) the use of texture based visualizations, and 4) straightforward cross sections.

Question 4: Some of the pros and cons of visualizations are , if users are allowed to modify S freely this will lead to being subjective where a user will think that they have found important data when really they have found a blip in the system. Sometimes you want to tune the mapping so that a desired result comes out strongly. This is misleading and should not be done. Pros are that the user can look at whatever specific areas of the data that interest them. They are not locked down to looking at one area and this can lead to more exploration and specification which in turn leads to better visualizations.

References

[1] "Numpy" : https://numpy.org

[2] "Matplotlib" : https://matplotlib.org

[3] "Pandas" : https://pandas.pydata.org

[4] "Plotly" : https://plotly.com/python/radar-chart/

[5] "Stackoverflow": https://stackoverflow.com