Ian Nelson

12/7/2021

CS 112

**Project Description**

For the CS 112 Final project, I opted to create a correlation visualizer using python.

This program is designed to compare annualized data, with similarly scaled statistics. Such a process is useful for determining defined numeric relationships between relevant datasets, such as marriage and divorce rates, or housing and gas prices.

Upon running the program, users are greeted with a few basic options, allowing said user to tailor the range of the program to their liking. Options include a range of years in which to analyze data, as well as the option to generate random or user made data. When generating user data, input prompts direct the user to submit a name for their data, in addition to individually submitted floats, in accordance to the range of years specified earlier. This data is written as lines of strings into a .txt file, and saved within a subfolder containing a directory of similarly formatted txt files, adjacent to the .py files. Generating random data uses a similar process, just with randomized ints instead of user input.

Following this process, users are directed to choose 2 from any available premade, randomized, or custom data, provided it was generated during the current runtime. Ints of the chosen year range, and strings containing the names of the chosen are then sent to an ArrayProcesser module. ArrayProccesser does the meat of the code, retrieving specified files from .txt files within the directory, reading the lines within and creating a 2d array of each year, and its corresponding data. A trimmer function is used to create new arrays of only the years and points found between the user specified timeframe. Next, the x/y axes in these arrays are switched using a NumPy function, creating a 2d array of all years, and all data. Both data halves from each formatted array are sent to a third module, MathSolver.py. In MathSolver.py, a mix of self-written logic and NumPy functions are used to find the averages, variances, standard deviations, covariance, and correlation coefficient for the two 1D arrays. These results of these calculations are returned to the ArrayProcesser, which are then returned to finalProject3.py alongside the 1D arrays as an object of type XY. Using this object, finalProject.py generates a graph of both used datasets, as seen below.

Chart, line chart

Description automatically generated

Additionally, all numbers utilized, and all math calculated are printed into the console, to show correlation and other interesting data.

Text

Description automatically generated

Following this output, the object graphed is stored in a list array for future use. Having completed an initial run of all functions, finalProject3.py will loop through an epilogue function, in which more mid-runtime datasets can be generated, more graphs can be visualized, and previously created XY objects can be re-visualized. This loop will repeat, until users specifically exit it. Even after completion, all custom and user made files are made to the user within the directory subfolder, for potential re-use.

The inspiration for this project came from a website with a similar premise: Tyler Vigen’s [Spurious Correlations (tylervigen.com)](https://tylervigen.com/spurious-correlations).

Additional premade datasets were generously compiled and provided by Meredith Olson.

**Lessons Learned**

**A majo**r difficulty encountered with this project was the implementation of statistics in the arithmetic section of my code. In order to correlate my data, I had to use a correlation coefficient, and in the final version of my code, I utilized a NumPy function to properly retrieve the correct results. This process was made harder, as I did not know that “correlation coefficient” was the name of the formula I was trying to replicate. My only hint to go off of was the results of correlations posted on Tyler Vigen’s Spurious Correlations website. In my research, I actually found the correlation coefficient formula, but disregarded it, thinking that it only correlated the x and y values of a single dataset, rather than correlating 2 different datasets (labeled x and y respectively). In addition to a lot more research I had to go to the Brooks Library Math Resources Center in order to receive tutoring in statistics: a class I am not in. This tutoring helped me to create a handwritten correlation coefficient calculator, which is ultimately still in my code, despite going unused. I tested this calculator on an example from Spurious Correlations, but unbeknownst to me at the time, this set of numbers just happened to be the only set that my calculator was able to correlate correctly. I continued to develop the rest of the program around the broken calculator, only noticing discrepancies while doing some final last minute test runs. I knew I wouldn’t have enough time to find the error I had made, much less fix it, but in the process of reviewing the arithmetic and thinking back on both my tutoring and my solo research, I realized that the correlation coefficient formula is what I needed to use after all. From here, I quickly found a NumPy function that did the same thing, and replaced my comparatively inaccurate calculator with it.

Aside from higher math, this project taught me a lot more about creating objects in python. While not part of the directions, creating and recalling objects was a skill I wanted to master through this lab. Objects and modules was a principle used frequently in CS 111, and I was somewhat underwhelmed that the concept was only covered briefly in CS 112. This attempt to further dive into an area I had limited experience was very difficult, and I spent a lot of this projects time writing re-writing the ArrayProccesser module. The main problem I repeatedly failed to address was how to call both sets of 1D data through all of the object’s methods. Somewhere in the web of code I had created, I somehow forgot that I could just… call the same methods multiple times. While I did get the program working, I fear that I may have given ArrayProccesser a narcissus complex, on account of the amount of .self tags I included. Despite throwing myself in the deep end in an attempt to better understand a concept, I am now confident that I fully understand object creation in python (or at least to the same level of understanding as was achieved in CWU’s java courses).

If given a chance to expand this project, there are a few changes I would make. First off, I originally intended to display all of the data values and arithmetic in a prettyTable, but I ran out of time to implement this feature. As it stands, I think my output is a bit ugly, but its still functional, so I suppose I will have to settle for it. Another change I would like to make is to find a way to normalize the heights of both line plots to one another. I’m not certain if this would be accomplished through a different implementation of Plotly, or if I would have to manually adjust the datasets to each other, reflecting the scale changes in the title, but either way, such normalization would be beneficial for clearer comparison between the two plots.