2023 UTSR Gas Turbine Industrial Fellowship Program

By: Robert Greene

I. Background

Currently, the Combustion Department in Siemens Energy uses a variety of commercial tools like MATLAB to process large amounts of their data. However, they are currently transitioning to new cloud-based tools that will enable faster access and analysis of data. I was tasked with develop python-based tools that could a user to access, process, and visualize the fleet engine data in a reliable and timely manner. I was also asked to make this code accessible for a wide range of end users, such that any new and/or novice python programmer could have easy access to the basic tools used in the analysis, but that I did not cut any potential avenues of analysis for any experienced users.

II. Outline of Code

To keep the code accessible for a wide range of users, it was important to develop not only a user-friendly code but also a simple outline that they could follow to avoid getting lost. For the final version of the code a simple outline was developed to help guide new users throughout the process, the steps are to:

1) Import all relevant python packages,
2) Import relevant fleet engine data,
3) Clean the data,
4) Visualize the data.

Please note that some details are omitted by design to protect any proprietary information.

<u>Step 1: Import all relevant python packages -</u>

Since the goal of this project was to develop accessible code, I wanted to keep the number of packages needed for the python environment down to a minimum. To this end, the code was built around using the pandas package for python as much as possible. This meant that even if an alternate package could have provided a more optimal method for solving a problem, I attempted to maintain as much of the code within functions built into the pandas package. Ultimately, I was forced to use several other packages but again, I wanted to maintain these to few in number as well. The code was developed to rely mostly on pandas, NumPy, matplotlib, and seaborn. The code also utilized a Siemens Energy in-house package developed for allowing fleet engine data to be pulled from the data cloud that stores the information.

<u>Step 2: Import relevant fleet engine data –</u>

Importing the relevant fleet engine data utilized an already developed in-house package. However, the code that was already developed to do this could appear intimidating to new and/or novice users, so I was tasked with creating a wrapper for the pre-existing code that simplified the user requirements and made the code mush more friendly to use. The code does not sacrifice any input options for users, but does significantly reduce the amount of code needed to pull the desired data.

Step 3: Cleaning the Data –

Due to the nature of measurements, it is sometimes desired by the engineers to perform various methods of aggregation and data cleaning on the raw data pulled from the cloud. To this end, simplified code was developed to allow any user to perform rolling statistics on the data. These statistics include maximums, minimums, averages, percent differences, ranges, etc. and could be performed at any desired time interval specified by the user. This allows for the removal of any spurious oscillations in the data set. In addition, the department wanted a simple way to filter any data. To accomplish this in a user-friendly manner, I wrote a function that enables the code to easily filter out functions based on a simple criterion. While the code does act very mush as a wrapper for built in filtering in the pandas package, it does present the options in a much more user-friendly experience.

Step 4: Visualize the Data –

While data visualization based on the matplotlib and seaborn libraries is not particularly challenging in and of itself, the amount of coding that is required to generate a single plot can be intimidating, especially to new and/or novice users. Therefore, the primary goal of this was to write a function that could generate a standard plot with minimal inputs. Easy plotting commands were written for line, scatter, and bar plots all with varying degrees of complexity. While the interface with these plots remained simple, I also included optional arguments to change many of the facets of the plots (like size, coloring, etc.) that one may want to have during the creation of the plots.

III. Conclusion

The improvement offered by the newly developed code may best be quantized as time saved during analysis of data. While total time saved over the lifetime of this code by all users may be difficult to quantify, a simple example may best illustrate the improvement of this code over the MATLAB based codes currently used. Looking at figure 1, we can see that we have a scatter plot in which 2 sets of data are plotted against each other on the x- and y-axes and are colored by the amplitude of some third data set on the z-axis. Creating a scatter plot that uses large quantities of data from a wide variety of sensors like this in MATLAB could take a significant amount of time, that is drastically reduced using the newly developed tools.

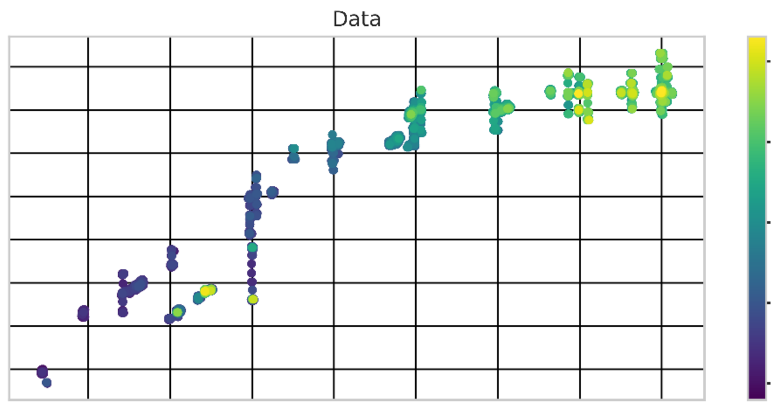

Figure 1: Example graph showing some example data sets plotted as a scatter plot, and colored by some third set of data