Appendix #1
Documentation of Implementation Challenges and Solutions

Making NASA's Open-Innovation Data More Machine-Learning Friendly: A Case for the MAVEN Datasets

A 2021 LURA Project

Ashton C. Fremin
Dhan Lord B. Fortela, PhD

Issues	Solutions	References
There were some issues getting PyDIVIDE to be imported during some of the preliminary testing of the code.	***PyDIVIDE*** requires the computer system to be running python 3.5 or above. During some trial runs with different areas of the code researchers downgraded the version of python being used to version 2.8 to import and use some packages that were not compatible with other, newer versions of python. When trying to use the ***PyDIVIDE*** package again, the computer was defaulting into running python version 2.8 not allowing ***PyDIVIDE*** to be imported. A simple switch back to the newer version of python fixed this issue.	- https://pydivide.readthedocs.io/en/latest/getting_started.html#system-requirements
Since the goal was to analyze the datasets over a single day, to eliminate redundancies in the code and dataset file researchers removed the first column indicating the date.	Researchers were able to select and remove the first column based on its location using the pandas function ***pandas.DataFrame.iloc***, which allows for position indexing and modification using integers.	- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iloc.html
To analyze the data using SOMs, the dataset to be used must contain no null values for any variable. The dataset selected to work with had many variables	The solution to this problem involved manipulating the datasets using a few separate ***pandas*** functions including ***pandas.DataFrame.dropna*** and ***pandas.DataFrame.isnull*** which removes observations with no entries and	- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isnull.html - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html

without values for certain parameters being measured.	removes the rows and columns that were left completely blank, respectively.	
For a few days researchers received errors when trying to get the cleaned-up datasets which where devoid of missing entries, blank rows, and columns extracted as Excel files.	The original issue was with the line of code being used to save the dataset as an Excel file. The line of code used originally had a variable named ***"pd"***, which was not originally defined as a variable. The researchers eventually were able to find a simpler pandas function, ***pandas.DataFrama.to_excel***, which was able to save the desired datasets as Excel files.	- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html
There was minor difficulty in calling the datasets that were manipulated and exported as Excel files back into the program to be fed into the SOM to be trained.	This error was easily rectified by changing the backwards slashes into forward slashes. The file path was pasted into the code with backslashes, which in python indicates and escape line. Changing the backslashes into forward slashes eliminated this error.	- https://pc.net/helpcenter/answers/backslash_vs_forward_slash - https://pro.arcgis.com/en/pro-app/latest/arcpy/get-started/setting-paths-to-data.htm
In attempting to train the SOM, it was a challenge to successfully import certain elements needed for SOM training from the *sompy* package.	This error was resolved by deleting and reinstalling the ***sompy*** package. It is believed the originally downloaded package file may have been corrupted or an older version of the package.	
To train the SOM using the dataset previously	Researchers were able to get around this issue by first manually analyzing	- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.astype.html

manipulated, the protocol is to first call the dataset into the SOM. There was trouble with this step, specifically getting the SOM to recognize all the objects within the dataset, as well as using the proper syntax when filling out arguments.	the data within the dataset of interest to make sure it was of the correct data type. Since the original dataset was quite large, it was decided to focus on the SWEA subset of the data. Using this subset of data requires that all our variables be of the ***'float64'*** data type. After verifying the data was of the correct type, researchers then used a list function to extract and list the column names from the dataset for component plane rendering. Completing all the above steps allowed for successful importation of the dataset into the SOM algorithm to begin working with the dataset.	
One of the central challenges to overcome when utilizing SOMs to elucidate patterns amongst various data is selecting an initial guess as to the shape and size of the graphical output.	This can be accomplished with ease by using the ***map_size*** function included in the ***sompy.SOMFactory*** package(s). This function allows one to import a dataset of choice and select the map shape desired. It then outputs the optimal map size based on the dataset, the shape selected, and other parameters that deal more with the theory behind selecting the optimal map size.	• https://github.com/sevamoo/SOMPY/blob/master/sompy/sompy.py
Once the data is successfully imported into the SOM, the algorithm	Researchers were able to use two native functions of the ***sompy*** package,	• https://www.sciencedirect.com/science/article/pii/S089360801930231X • https://www.intechopen.com/chapters/69305

begins training the dataset, and once the dataset is trained the next challenge to overcome is to calculate the topographic and quantization error.	***som.calcaulate_topographic_error ()*** and ***som.calculate_quantization_error ()*** to calculate the topographic and quantization errors, respectively. This step involves some manual iterations requiring changing of certain parameters withing the SOM arguments to try and minimize both the topographic and quantization error.	• https://github.com/sevamoo/SOMPY/blob/master/sompy/sompy.py
After the SOM has been trained, the challenge is to now find different graphical methods that will allow researchers to more easily and quickly elucidate different patterns that may appear within the dataset.	Within the sompy package, there contained many preinstalled functions for representing and interpreting the results generated by the SOM algorithm. An important application of the graphical methods is clustering. Clustering manipulates the data visually and allows researchers to clearly see the patterns the data makes.	• https://github.com/sevamoo/SOMPY • https://github.com/ldocao/sompy
Clustering allows researchers to better see various patterns contained within the trained dataset; however, this visual representation does not allow one to easily tell which variables and values are being represented to in a specific cluster. There needed to be a way to overcome this issue which would	This problem was able to be solved by first creating a new dataset which the researchers will add to the value of the BMU and cluster. We then created a dictionary to relate the data's BMU with the cluster of the BMU. This then allows the researchers to map the raw data against the clusters, so one can easily see which raw data falls into which cluster.	• https://github.com/sevamoo/SOMPY/blob/master/sompy/sompy.py

allow researchers to attach raw data to the graphical representation already generated.		
The final objective of the project was the successful deployment of the Jupyter Notebook in an interactive online format. Finding the proper online service to meet the needs of the project did present minor issues for researchers to overcome.	Researchers initially thought of deploying the Jupyter notebook using Microsoft's Azure Cloud Computing Services. Azure is a paid service that offers many different products with different pricing based on the product and computing power necessary. Researchers also looked at Google's Colab service which offers a free base plan, with the option to upgrade between two subscription options. Colab's subscription options offer faster GPU times and more RAM; however, it was believed that the base level service would be satisfactory in meeting our needs for initial deployment of the Jupyter Notebook. Researchers also briefly looked into using Amazon Web Services which was like Azure, offering many different products and similar pricing protocols. The researchers decided on using Google's Colab services due to an easier startup as well as a valuable free tier suitable for initial deployment of the Jupyter Notebook.	• https://colab.research.google.com • https://azure.microsoft.com/en-us/ • https://aws.amazon.com/?nc2=h_lg