**Promoting Ease of Model Output Data Analysis**

1. Definition of Problem and Overview of Current Project

Environmental models generate output data, which describes the model’s predictions for the concentration of a particular species at a particular location and time. Generating output data is one of the core purposes of models, and these data are extremely useful for model-based decision-making. For example, when combined with data on social factors in the model coverage area, output data can be used to examine how pollution interacts with demographics such as population density, socioeconomic status, and race/ethnicity. These data can then inform policy solutions to promote public health and environmental justice (for one example, see Grineski, Bolin, & Boone, 2007).

The amount of output data generated by environmental models is often extremely large. For this reason, data management and analysis can be both time-intensive and computationally intensive. Inefficient code to manage and analyze model output data can severely hinder and delay the process of developing data-informed public health policy solutions. For this reason, any researchers working with model output data ought to be proactive in creating efficient code to manage and analyze these data.

In the current project, I worked with output data for Robeson County, North Carolina from the Community Multiscale Air Quality Modeling System (CMAQ) (for further information, see https://www.epa.gov/cmaq). The data covered 45 days from 12AM UTC+0 on June 1, 2013, to 12AM UTC+0 on July 16, 2013. Each day of output data was contained in its own netCDF (.nc) file. The model contained near-full coverage of Robeson County with 30 output boxes covering 16km2 (4x4km) each (Figure 1). Coordinates for the center points of each box were contained in separate netCDF4 (.nc4) files. I will refer to these center points as “sites” in the remainder of this paper. One observation per hour per pollutant per site is recorded in these data. Dannys Ayala Terrones built the foundation for this project by writing code to extract relevant model outputs and to manage data. His code will be referred to as “existing code” in the remainder of this paper.

**The goal of my project was twofold: (1) to run the Robeson County model output data through the existing code and compile the data into an easily analyzable format, and (2) to make the code more efficient.**

The intended deliverable for Goal 1 was a “tidy” dataset containing concentrations for five important pollutants at every output site. The term “tidy” refers to a specific, tabular data format that facilitates analysis, in which each variable is captured in a column and each observation is captured in a row (Wickham, 2014). The intended audience for Deliverable 1 was a group of my classmates whose project was to examine pollution and social factors in Robeson County. This group will be referred to as the “social factors group” for the remainder of this paper.

The intended deliverables for Goal 2 were as follows: (a) to implement object-oriented programming (OOP) and other small updates in existing code to make it easier to adapt the code to new data, and (b) to write new code that compiles multiple output data files into one tidy dataset. The intended audience for Deliverables 2a and 2b was future researchers using any similarly formatted output data.

Map

Description automatically generated

*Figure 1. Map of Robeson County (red outline) with center points for the 30 output boxes (yellow placemarks). Distance between center points is shown in the bottom-left. Image taken using GoogleEarth.*

2. Methods

To accomplish Goal 1, I used both Python 3.9.7 (Van Rossum & Drake, 2009) and R/RStudio (R Core Team, 2022; RStudio Team, 2022), including the tidyverse package in R (Wickham et al., 2019). All my work in Python for this project was done using the UNC Longleaf Cluster (see https://its.unc.edu/research-computing/longleaf-cluster/ for information). I used the existing code (in Python) to accomplish the first part of this goal. The existing code was designed to extract model output data for one pollutant from one day at all sites. My colleagues in the social factors group requested data for five pollutants: O3, CO, NO, NO2, and SO2; in response to that request, I expanded the existing code to run through data for multiple pollutants at once. I ran this code 45 times (once for each day of data). Each run took approximately 45 minutes due to two code chunks that required significant processing time. The end result of this expanded existing code was 225 csv files (one csv file per pollutant per day). This expanded existing code file (“ExtractingDataNew.ipynb”) can be accessed via GitHub (see Appendix A).

Once I had the 225 csv files, I used R to compile them into one tidy dataset. I used R due to my increased familiarity with data management using this software compared to Python. Since I wanted to quickly get analyzable data to the social factors group, I decided to use software that would enable me to do the work in less time. The end result of this code was one tidy csv file containing all the data from the 225 csv files (Figure 2). This file comprised Deliverable 1 and was sent to the social factors group. The R code file (“ENVR451\_management.R”) and the csv file (“Robeson Output Data.csv”) can be accessed via GitHub (see Appendix A).

Table

Description automatically generated

*Figure 2. Six rows of the Deliverable 1 tidy dataset. In this case, the tidy format means that all concentrations are in one column so that they can be easily analyzed together and grouped by day, species, hour, or site.*

To accomplish Goal 2, I used Python. Deliverable 2a involved implementing OOP in the existing code in order to decrease the number of edits needed to adapt this code to different data. In OOP, objects called “classes” are created which contain all the operations needed to accomplish the purpose of the code. Then, all these operations can be run at once using far fewer lines of code. The advantage of OOP is that the code within a class should not need to be changed in order to adapt the code to new data, the only inputs that the user needs to change are the few lines outside the class that actually run the functions within the class.

In the code file I created to implement OOP on the existing code, I created a class called “Extract” which includes six functions that encapsulate every step of the process contained in the existing code. Notes on what each of these functions do are included in the class code. Only the first and last functions require the user to specify function arguments. The other four functions do not require user-specified arguments (although they rely on the user-specified arguments from the first function) and are only separated into different functions so that the user can stop at various points in the process to look at the data and ensure that the code is running smoothly.

The benefits of the OOP code compared to the existing code are examined in the Results section. The OOP code file (“ExtractingDataOOP\_CLEAN.ipynb”) can be accessed via GitHub (see Appendix A). The current state of this file runs through O3 data for July 14, 2013, at the 30 Robeson County sites mentioned previously. An Extract class object called “df” is created and all six functions are run on it, resulting in the creation of a csv file that contains hourly O3 output data for July 14, 2013, for all 30 sites.

Deliverable 2b involved adapting the code I wrote in R for Goal 1 into Python in order to facilitate smoother data management for future projects. As previously mentioned, I used R for Goal 1 to quickly manage the data given my higher familiarity with R compared to Python. However, in order for this code to be of maximal use for future research, all code should be in one programming language so that researchers do not need to know multiple languages to manage their data.

I wrote the Deliverable 2b Python code file using OOP to facilitate easy future use and to match the Deliverable 2a code file in style. In the Deliverable 2b code file, I created a class called “Combine” which contains 7 functions that encapsulate the R code I wrote for Goal 1. Only the first and third functions require the user to specify function arguments. The other five functions do not require user-specified arguments (although they rely on the user-specified arguments from the first function). Unlike in the OOP class from the Deliverable 2a file, not all the functions in this class have to be run. To combine all data files for a specific pollutant, the first four functions can be used. To bring together combined data files from multiple pollutants, the first function and the final three functions are used. This is explained in further detail in the comments within the Combine class code.

The Deliverable 2b code file (“CombineCSVs\_OOP.ipynb”) can be accessed via GitHub (see Appendix A). The current state of this file creates six Combine class objects. The first function in the class initiates these objects: one for each of the five pollutants requested by the social factors group (“COcombine”, “NOcombine”, “O3combine”, “NO2combine”, and “SO2combine), and one to combine data from all five pollutants (“ALLcombine”). The file runs through the next three functions of the class for the first five Combine class objects, resulting in five separate csvs (one per pollutant) that each contain hourly data for all 45 days in the study period and for all 30 sites. The file then runs through the final three functions for the “ALLcombine” class object, which creates a tidy dataset in csv form that is nearly identical to the Deliverable 1 csv.

3. Results

The holistic results of this project are twofold: (1) a tidy dataset for CMAQ model output data in Robeson County, and (2) an improved set of code files to manage future CMAQ output data in netCDF form.

The social factors group utilized Deliverable 1 to conduct GIS analyses of pollution and demographic factors (e.g., race/ethnicity, socioeconomic status) in Robeson County. See their paper (ENVR451 final paper by Onyekachi & Wilson, 2022) for more details.

The most important benefit of the OOP code that I implemented for Deliverable 2a is that it significantly decreases the number of manual changes required to look at different pollutants compared to the existing code. In the existing code, changing the pollutant (e.g., from O3 to NO) would require at least 180 manual edits. In the OOP code, the number of manual edits required to examine a different pollutant is at most 2: the species argument in the first function must be changed, and the name of the csv file at the end of the code should also be changed to reflect the different pollutant. Using OOP to examine model output data thus is a superior approach because it automates most of the heavy lifting requires to adapt the code to new data. I hope that this approach will decrease the time and effort required to adapt this code to new data in the future.

No results have yet taken place as a result of Deliverable 2b, but the intended result is a smoother data management process for future researchers.

I believe that benefits related to Deliverables 2a and 2b can be magnified with further code improvements that were outside the scope of this project. These are discussed in further detail in the next section.

4. Future Directions and Conclusion

I believe that the code I created and/or improved for this project can be further improved to facilitate even quicker data management for model output data in Robeson County and elsewhere. These areas for future work fall into two categories: processing time and adaptability.

Regarding processing time, one major edit that I did not have the time to accomplish in this project was to edit two code chunks in the Deliverable 2a code file (“ExtractingDataOOP\_CLEAN.ipynb”, see Appendix A). These two code chunks require significant processing time, as mentioned in the first paragraph of the methods section. I believe that the actions accomplished by these code chunks can be accomplished using code that requires (a) far fewer lines of code and (b) much less processing time. I spent a significant amount of time looking into possible answers on the Internet, but I was unable to put all the pieces together in time to accomplish them in the Deliverable 2a code before the deadline. I believe that researchers with more extensive Python knowledge than myself and a bit more time could solve these puzzles and significantly reduce the amount of time needed to manage model output data using the Deliverable 2a code file. This would be an enormous benefit to future researchers as they could quickly run through multiple pollutants on multiple days, spending less time managing data and more time analyzing data, drawing conclusions, and developing valuable public health solutions.

Both of the code chunks that create processing time issues are found in the “Extract” class code. The first is found within the “countobs” function. The purpose of this code is to iterate across an entire xarray dataset that contains the output data and count all non-missing observations by hour. Currently, this is accomplished using repeated if-else statements that look for non-missing observations for each hour of the data and add one to a counter each time one is found. I suspect that these if statements and the iterative nature of this function cause the high processing time, and I imagine that this action could be accomplished with less code and could also perhaps involve less processing time. There are simple functions in Python to count non-missing observations, and I was able to use only a couple lines of code to find the total number of non-missing observations throughout the dataset (in this case, 720, or 24 hours by 30 sites). This code took less than five seconds to run, compared to the 15 minutes of processing time required by the original code. The crux of the issue that I was unable to resolve in the time given is how to group the non-missing count by hour (in this case, it should be 30 for each hour, since there are 30 sites in the study dataset that each record hourly observations). I suspect that only a couple more lines of code could resolve this issue and keep the processing time measured in seconds instead of minutes. I hope that this description of the problem will enable future researchers to solve this problem for the benefit of future data management and analysis.

The second code chunk that requires significant processing time is found in the second half of the “arraycreate” function within the “Extract” class code. Similar to the other problematic code chunk, this one also involves multiple if else statements, each corresponding to an hour of the day. For each non-missing value in each hour, this function fills in the values of the concentration and coordinates data to the xarray dataset. This function was a bit more difficult for me to grasp, so I did not get as far in trying to solve the processing time puzzle. However, I do wonder whether the entire set of actions accomplished by these two code chunks could be accomplished more simply. I am not very familiar with the xarray data type, so I am not one hundred percent confident in this. However, I do believe that the multiple if-else statements could be simplified, as with the previous problematic code chunk, to reduce number of lines and amount of time required to run the code in the Deliverable 2a code file. Once again, I think solving this processing time problem would be extremely valuable for future model output data analysis and data-informed solution-making.

In addition to solving issues related to processing time in Deliverable 2a, I think that a worthy route for future work on model output data management would be to further increase the adaptability of the OOP classes I have created for a wide range of data, particularly in the Deliverable 2b code file. Currently, the “Combine” class in this file requires inputting one species of pollutant at a time. I was unable to solve this problem in the time I had, but I think that for a more experienced Python programmer, it would not be too difficult to improve this class code so that an undefined number of pollutants could be input at once and seamlessly combined into one tidy data file. Along with these code improvements, the class code should be edited so that users only need to enter the “path\_in”, “path\_out1”, and “path\_out2” arguments once for all pollutants (these arguments specify where to get the individual files from and where to save the combined files).

Another method to potentially improve the Deliverable 2b code file is to separate the code that compiles multiple days of data for a single pollutant from the code that compiles these single-pollutant compiled files into a multi-pollutant compiled file. Currently, the “Combine” class contains the code for both these actions, and the overlap may be confusing to users. Having two separate classes for these actions may improve user experience, and with more time in hand, I would have liked to implement this change.

Taken as a whole, the improvements I have suggested in this section will build upon my work in the course of this project to create and improve code to manage model output data. The goal of this project, and my goal in describing these future directions in detail, is to dramatically improve future researchers’ ease of data management. Applying both the present work and these future action items to model output data management in Robeson County and elsewhere will free up more time to analyze data, draw conclusions, and develop public health solutions. Effective data science lays the groundwork for intervention development and public health policy, and the present work outlined here is one step in a process of building efficient modeling data management pipelines to facilitate future leaps and bounds in research that uses environmental models.

5. Acknowledgements

Thank you to Dr. Will Vizuete for teaching this class, to Dannys Ayala Terrones for creating the original code files, and for all past and present members of the Vizuete lab who worked to generate this data.

6. References

GoogleEarth (n.d.). [Image of Robeson County, NC]. Retreived December 5, 2022, from GoogleEarth Pro 7.3.6.9285.

Grineski, S., Bolin, B., & Boone, C. (2007). Criteria air pollution and marginalized populations: Environmental inequity in metropolitan Phoenix, Arizona. *Social Science Quarterly, 88*(2), 535-554. https://doi.org/10.1111/j.1540-6237.2007.00470.x.

﻿R Core Team (2022). *R: A language and environment for statistical computing.* R

Foundation for Statistical Computing, Vienna, Austria. Accessed from: https://www.R-project.org/.

RStudio Team (2022). *RStudio: Integrated Development for R.* RStudio, PBC, Boston, MA. Accessed from: http://www.rstudio.com/.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace.

Wickham, H. (2014). Tidy data. *Journal of Statistical Software, 59*(10), 1-23. https://doi.org/10.18637/jss.v059.i10.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., …, Yutani, H. (2019). “Welcome to the tidyverse.” *Journal of Open Source Software*, *4*(43), 1686. doi:10.21105/joss.01686.

Appendix A: Data Access Information

All project files mentioned by name in this report can be found on GitHub at https://github.com/adolwick/ENVR451\_2022.

CMAQ output files were accessed using Longleaf from the following folder: /ms/depts/ese/airqual/received/from\_Quazi\_SOAS2013\_v521runs\_output/output\_SOAS2013\_PhaseSepSchmeddingACP2019\_ros3\_1\_5ISOP\_Basecase\_sapracae7iaq\_v53.

NetCDF4 files containing coordinates for Robeson County were shared with me by Dannys Ayala Terrones.