Project 3: Data Manipulation with Pandas

John Wesley Mathis

Dr. Anthony Choi

June ##, 2024

ECE/SSE 591, Summer 2024

Table of Contents:

[Deliverable Table 5](#__RefHeading___Toc2569_331549091)

[1. Introduction 6](#__RefHeading___Toc2573_331549091)

[2. SIR Model Analysis Updated 7](#__RefHeading___Toc2577_331549091)

[7. Conclusion 17](#__RefHeading___Toc2609_331549091)

[References 18](#__RefHeading___Toc3692_331549091)

Table of Figures:

[*Figure 1: SIR Model Code* 7](#Figure!22|sequence)

[*Figure 2: SIR Model Code (final)* 8](#Figure!23|sequence)

[*Figure 3: SIR Model: Sorting number of infected individuals* 9](#Figure!24|sequence)

[*Figure 4: SIR Model: Boolean indexing to find where infections exceed a threshold* 9](#Figure!25|sequence)

[*Figure 5: SIR Model: Using universal functions `cumsum` and `mean`* 10](#Figure!26|sequence)

[*Figure 6: SIR Model: Using universal functions `cumsum` and `mean` (continued)* 11](#Figure!27|sequence)

[*Figure 7: SIR Model: Using universal functions `max` and `argmax`* 12](#Figure!28|sequence)

[*Figure 8: SIR Model: First 50 days infected array access, mean, and max values* 12](#Figure!29|sequence)

[*Figure 9: SIR Model: Days recovered individuals surpassed infected* 13](#Figure!30|sequence)

[*Figure 10: Monte Carlo Simulation of the SIR Model Code* 14](#Figure!31|sequence)

[*Figure 11: Monte Carlo Simulation Output* 15](#Figure!32|sequence)

# **Deliverable Table**

The purpose of this table is to provide a complete view of the concepts covered in chapter 2 of *"Python Data Science Handbook"* (VanderPlas, 2016) and provide a general page location and/or project name for where the topic was demonstrated. This is not an exhaustive list. Some areas may be covered by multiple projects. This is meant only as a means to show specific areas where the topic is demonstrated and to ensure that every area was covered.

|  |  |
| --- | --- |
| Deliverables | Location |
| Introducing Pandas Objects |  |
| Data Indexing and Selection |  |
| Operating on Data in Pandas |  |
| Handling Missing Data |  |
| Hierarchical Indexing |  |
| Combining Datasets: Concat and Append |  |
| Combining Datasets: Merge and Join |  |
| Aggregation and Grouping |  |
| Pivot Tables |  |
| Vectorized String Operations |  |
| Working with Time Series |  |
| High-Performance Pandas: eval() and query() |  |

# 1. Introduction

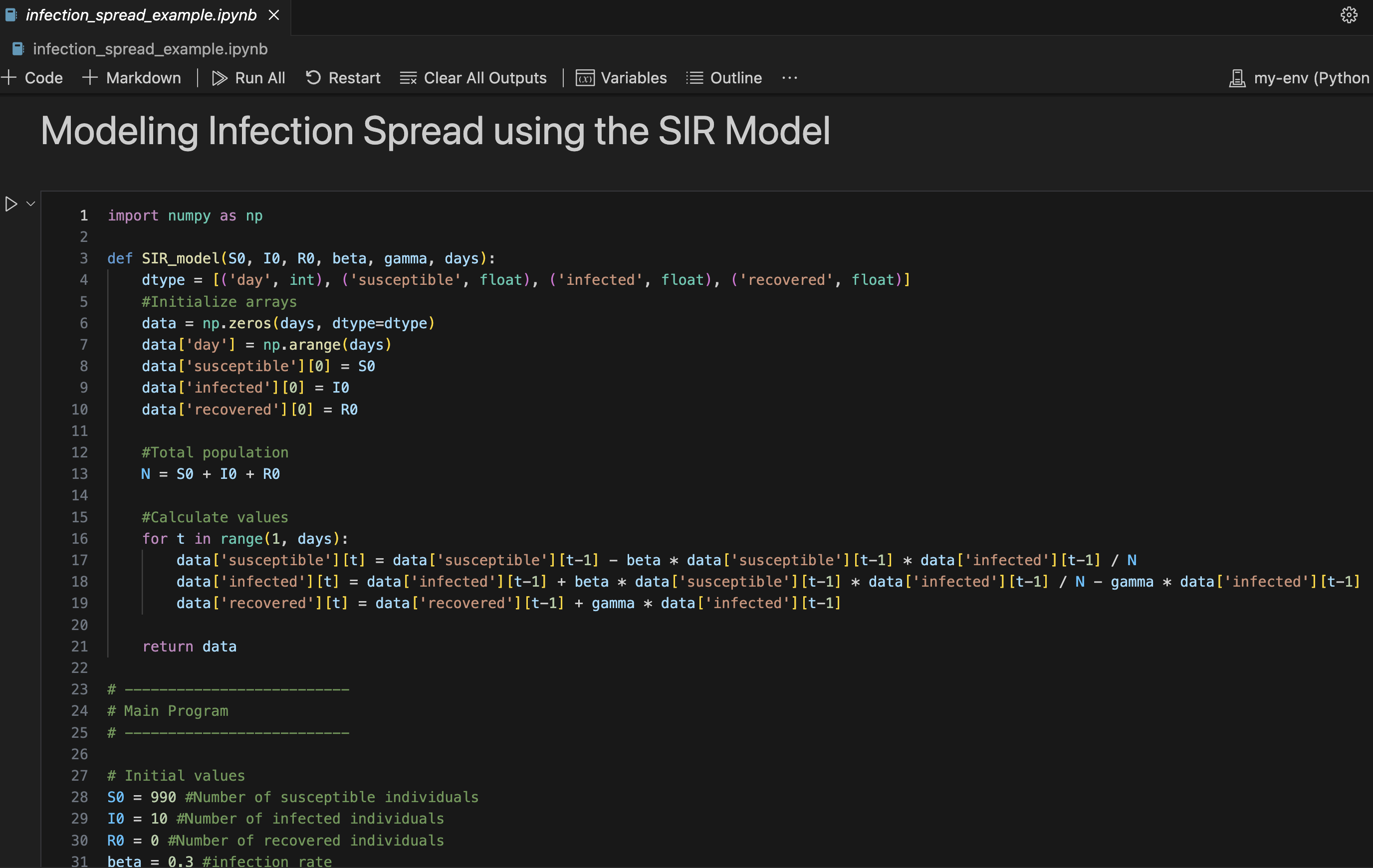
Python is a versatile language with numerous libraries. Because of its easy to understand syntax, it is very popular among the data science community as a useful tool to load, store, and manipulate data. Much of this data is typically managed by converting it into arrays of numbers. However, as useful and easy as it is to use Python programming, managing data efficiently has its many drawbacks. Fortunately, there are specialized tools that have been created to improve Python’s ability to handle such numerical data. One such package that has been created is the Numerical Python package or NumPy as it’s generally known.

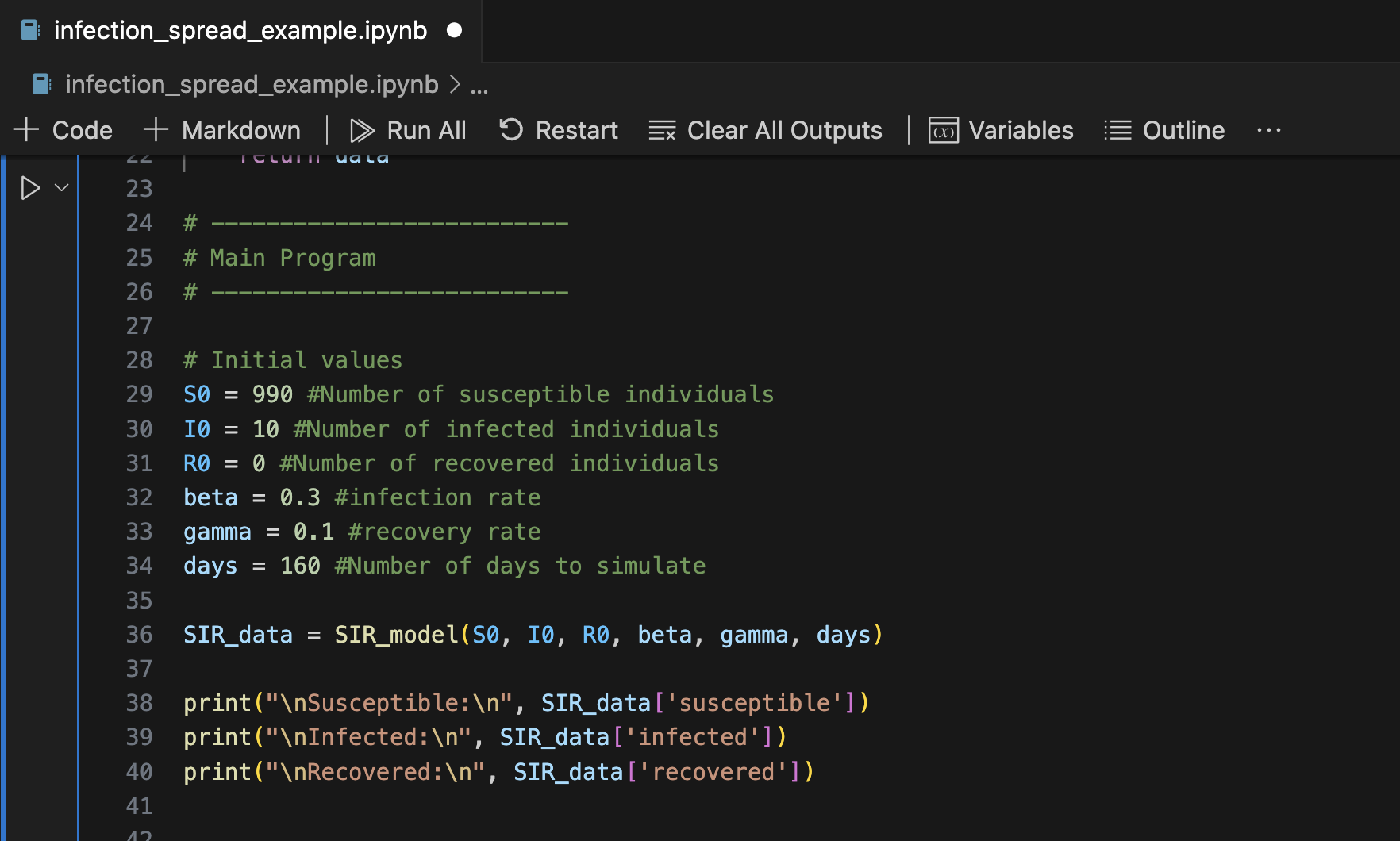
NumPy provides a manner to more efficiently handle data arrays. NumPy’s arrays are similar to Python’s built-in arrays but where it shines is how is handles storage as the data grows larger. As a result, NumPy has become the foundation for many scientific tools that are in use.

This report aims to demonstrate my proficiency in Python fundamentals as well as NumPy fundamentals that were covered in chapter 2 of the “*Python Data Science Handbook”* written by Jake VanderPlas (2016). Each exercise and mini-project is used to illustrate the concepts outlined in the deliverable table from the previous section. The code presented throughout this report was written using Visual Studio Code with Jupyter Notebook extensions. In the following sections an analysis is done on the two mini-projects. The remaining sections cover various exercises, and explanations are detailed to explore the structure and functionality of coding using NumPy packages.

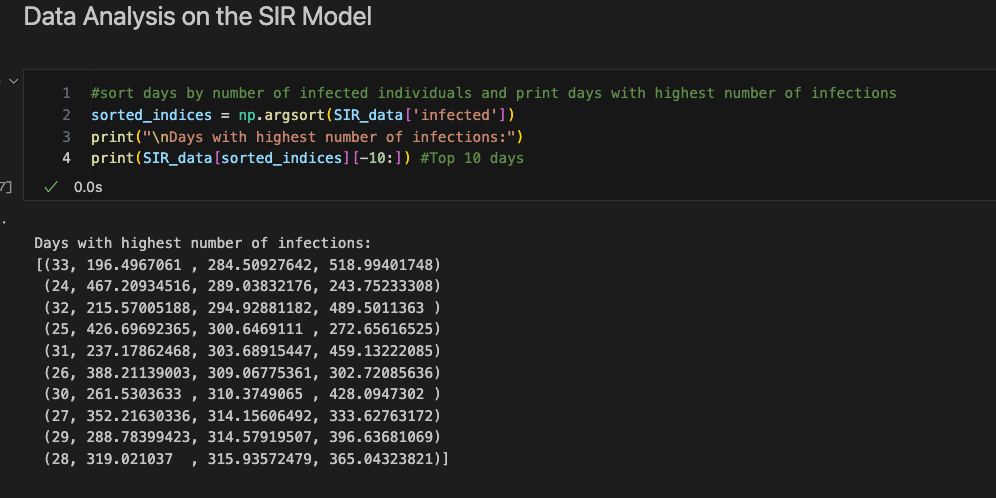
# 2. SIR Model Analysis Updated

The next project imagines a scenario where there is an outbreak of an infectious disease, such as a zombie apocalypse (Lowe, S., Mathis, J., & Wall, N. (2019)), and creates a Python script to model the spread of the zombie disease. The differential equations used to model the spread of the disease is the SIR model that was developed by Kermack and McKendrick. SIR stands for susceptible, infected, and recovered. For this project I considered two populations: the humans (susceptible group) and the zombies (infected group). The disease can be transmitted via scratching or biting. To simplify the project and modeling, I made the assumptions that the disease spreads rapidly where the change in populations can only be attributed to the infection; and that there are no births, so that the populations stayed constant. I used a structured NumPy array to store and access data in a more organized manner. Four data types were defined: *`day`* as type *int, ‘susceptible`, infected`* and *`recovered`* as type *float*. The structured array *`data`* was initialized with pre-defined initial conditions described above. A *`for`* loop was used to update the data for each data type by using the general equations for a SIR model starting with day 1 and ending with the simulations defined variable of *`days`*. The population is calculated by adding together the initial conditions *`S0`, `I0`,* and *`R0`*. This is all stored in the function *`SIR\_model(S0, I0, R0, beta, gamma, days)`.* The variable *`beta`*  defines the infection rate and the variable *`gamma`* defines the recovery rate. The function returns the structured array *`data`* after using Numpy broadcasting to calculate the values for *`susceptible`, `infected`, and `recovered`* data types for the *`data`* array*.* Figure 23 and 24 show the final code for the SIR model.

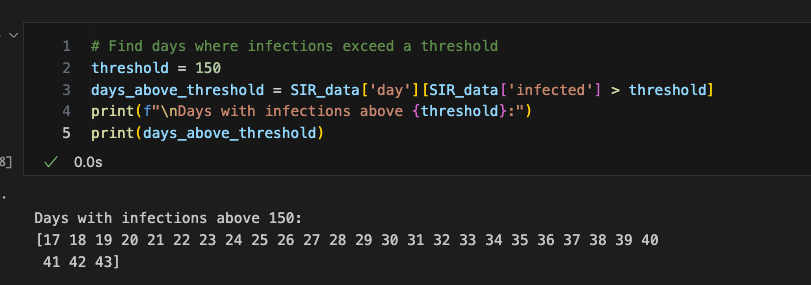
*Figure 1: SIR Model Code*

*Figure 2: SIR Model Code (final)*

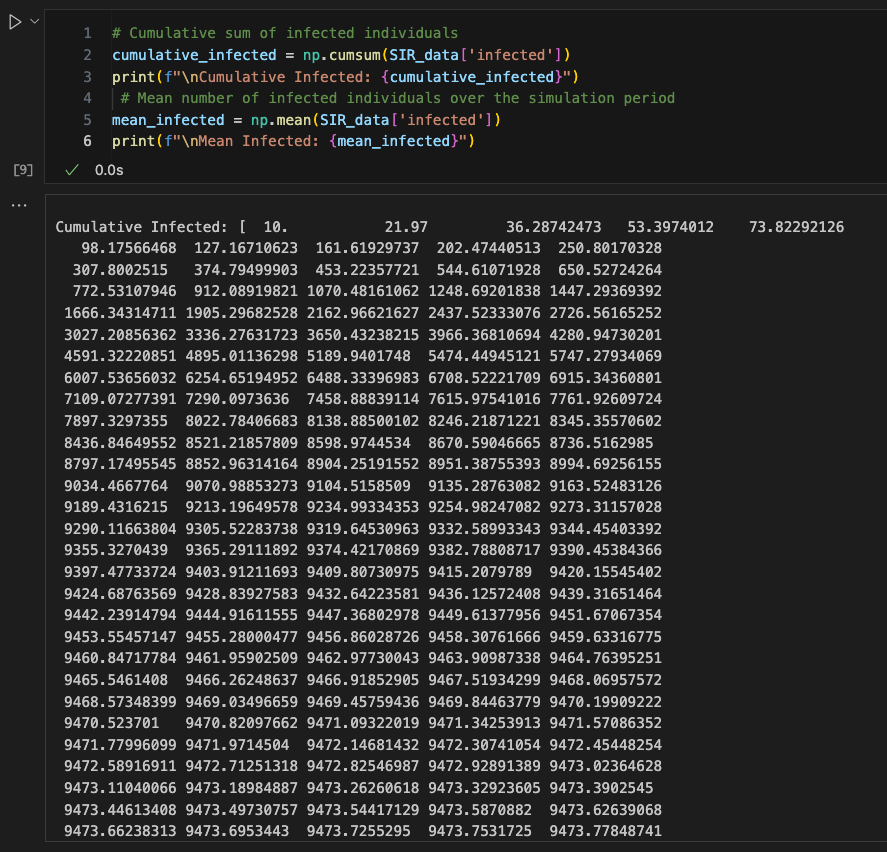
To show the benefit of using Python for modeling disease infection, various data analyses were performed. The first analysis was using *`np.argsort`* to sort the days by the number of infected individuals. With this information stored in a variable, using array indexing, the program prints to the screen the days with the highest number of infections. Figure 25 shows the code and output.

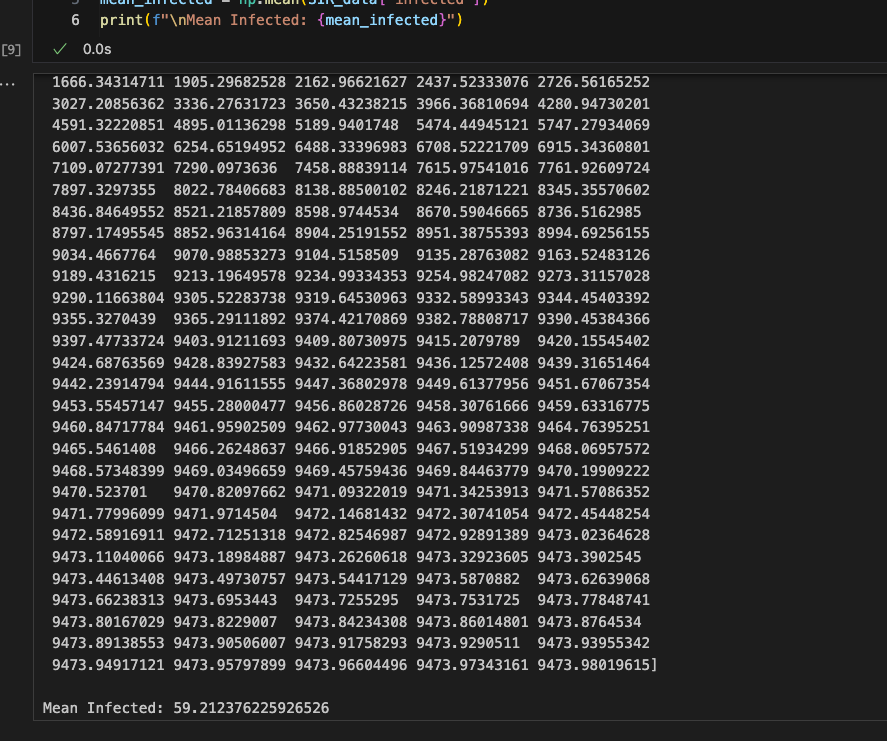
*Figure 3: SIR Model: Sorting number of infected individuals*

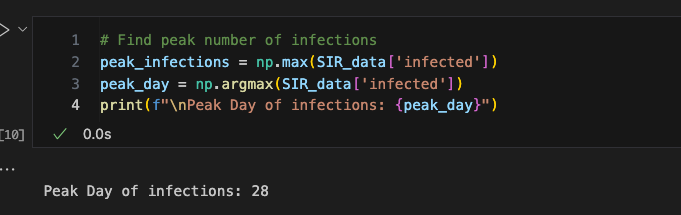
The next analysis uses fancy indexing and boolean indexing to access array elements to find the days where the number of infected individuals exceeds a certain threshold. Since fancy indexing allows access to elements in an arbitrary order, it is best used here. The boolean indexing is used to set the condition that must be met. Figure 26 shows the code and the output.

*Figure 4: SIR Model: Boolean indexing to find where infections exceed a threshold*

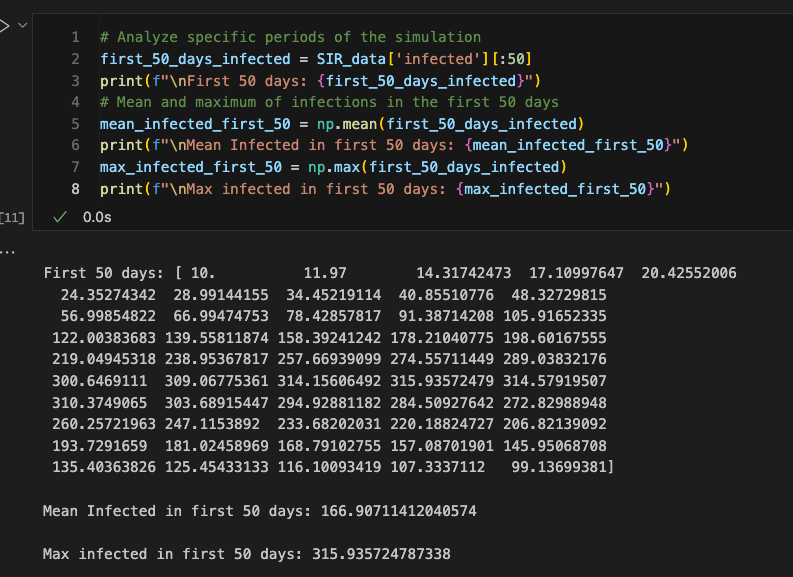
The next analyses provide statistical information. It uses the built in NumPy cumulative summation and mean methods to calculate the cumulative sum of infected individuals and the mean number of infected individuals over the simulation period, and prints the results to the screen. Figure 27 and 28 shows the code and output. Using *`np.max`* and *`np.argmax`,* the peak number of infections are able to be found and the day this occurred is able to be quickly produced. Figure 29 shows the code and output.

*Figure 5: SIR Model: Using universal functions `cumsum` and `mean`*

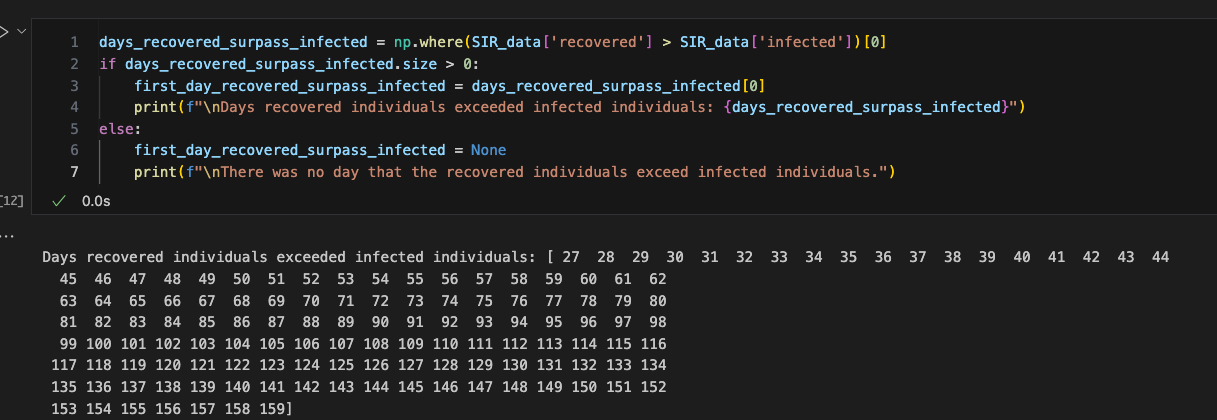
*Figure 6: SIR Model: Using universal functions `cumsum` and `mean` (continued)*

*Figure 7: SIR Model: Using universal functions `max` and `argmax`*

To further demonstrate the capabilities of using NumPy, the program was coded to analyze specific periods of the simulation. For this example, the first 50 days were analyzed. Using array slicing and aggregation, the data of the infected for the first 50 days was accessed and stored in the variable *`first\_50\_­days­\_infected`*. With this information, the aggregation functions, *`mean`* and *`max`* were called and the results printed to the screen. Figure 30 below shows the code and output.

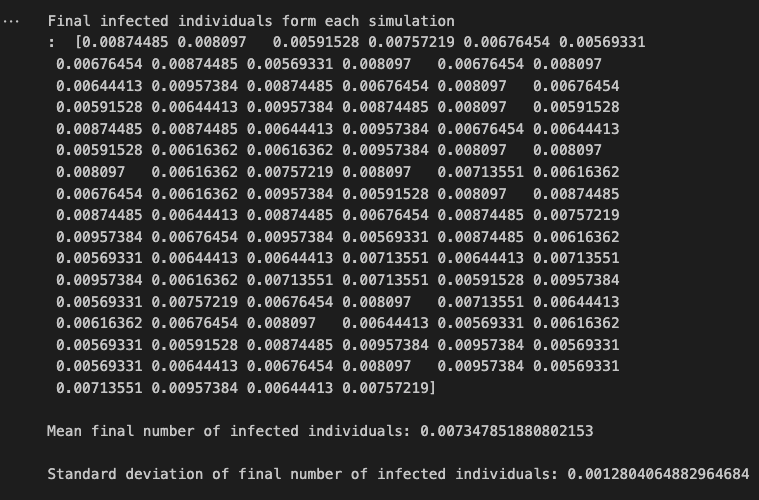
*Figure 8: SIR Model: First 50 days infected array access, mean, and max values*

The next analysis using advanced NumPy indexing and boolean statements. Using *`np.where`*, the program determines the day when the number of recovered individuals surpasses the number of infected individuals. If it occurred, then the days where this occurred are printed to the screen, otherwise a statement alerts the user that this condition never occurred with the provided data. Figure 31 shows the code and output.

*Figure 9: SIR Model: Days recovered individuals surpassed infected*

The last analysis is a Monte Carlo simulation. The Monte Carlo simulation is used to estimate the impact of a random initial condition. For this example, only the initial number of infected, *`I0`*, was randomized. A new function, *`monte\_carlo­\_simulation()`*, was defined. It declares a new array *`final\_infected`* using *`np.zeros`.* With the given number of simulations to run through, the code uses a *`for`* loop to initialize the initial condition of *`I0`* and run the *`SIR\_model`* function. The final number of infected individuals is accessed and stored in the *`final­\_infected`* array. Once the results are collected in an array, the *`final\_infected\_results`* are printed to the screen along with the mean and standard deviation using the appropriate universal functions. Figure 32 and 33 below shows the code and the results.

*Figure 10: Monte Carlo Simulation of the SIR Model Code*

*Figure 11: Monte Carlo Simulation Output*

# **7. Conclusion**

This report documents my journey in mastering NumPy which involved grasping fundamental concepts such as arrays, computations using universal functions and broadcasting, incorporating comparisons and boolean logic into arrays, indexing and sorting arrays, and learning how to create structured arrays.

Through the process of coding, many errors were encountered. Due to the simplicity of the projects and exercises, all of the errors were syntax errors. My experience is mainly in C and C++ so I am used to the nuance of adding semicolons to the end of each line. Many times in Python, I found myself accidentally adding semicolons to the ends of blocks and statements, confused why my code block continually showed errors. Other examples of syntax errors I encountered include forget a quotation, forgetting a colon when defining a function or loop, or simply misspelling a variable that I named.

Engaging in mini-projects and working through the exercises has proved to be a valuable source in aiding me to take the theory of Python fundamentals and the NumPy package and put the concepts into practice and think about practical data science applications. This has allowed me to grasp the concepts that make the NumPy package a valuable tool for efficiently managing large sets of data.

# References

1. Rougier, N.P. (n.d.). *Numpy 100.* Retrieved from <https://github.com/rougier/numpy-100>
2. Lowe, S., Mathis, J., & Wall, N. (2019). *Humans vs. Zombies Lab.* Unpublished paper, Mercer University.
3. VanderPlas, J. (*2016*).  *Python Data Science Handbook*. O’Reilly Media. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/index.html