Final Project Report: Finding the Most Prevalent Element in Type 1a Supernovae

Chris Joseph

Introduction

I have been interested in space since I was six years old, so I have always wanted to do projects that incorporate celestial bodies. When I heard that programming was the backbone of astrophysics research, I wanted to learn Python as soon as I could so I could do more projects. Supernovae have always been a point of interest for me because of their ability to produce elements during their violent explosions. In this report, I deep dive into the motivations, data analysis, curve fitting, and fitting with errors that went into this project.

Chosen Phenomenon and Data Source

In this final project, I utilized the spectral data of 31 Type 1a Supernovae (SNe) to find the most abundant element in each. The data was obtained from the Harvard-Smithsonian Center for Astrophysics Supernova Data Archive. There is a paper describing the data along with the data tables by Matheson, T., et al from 2008. This research group studied the spectral data of Type 1a supernovae from 1977 to 2001. The download came with 32 files, each containing spectra of a supernova from 1977 to 2001.

Next, I needed to compare the SNe observed spectra to the wavelengths of well-defined elements. I utilized the National Institute of Standards and Technology's Atomic Spectra Database to obtain the wavelengths of well-defined elements. When exporting the data, I was left with 8202 cells of wavelengths for many elements.

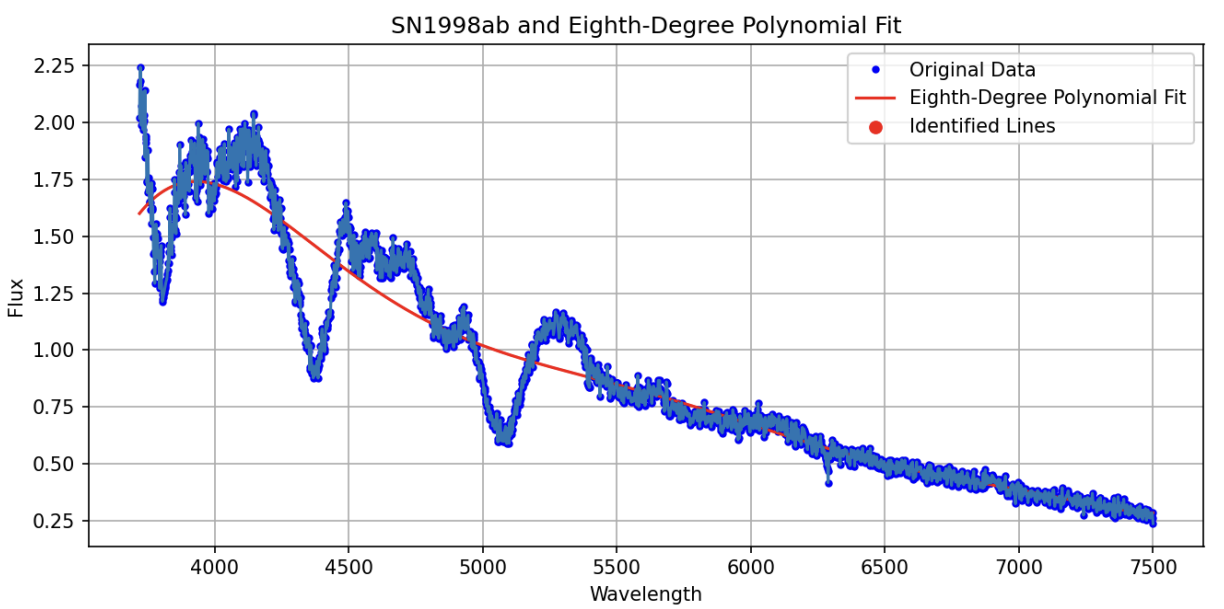
Methods

First, I start by reading the SNe spectra and NIST Atomic spectra files using the pandas libraries. Then, I find the closest element with the find_element function. I find the closest elements by subtracting the Atomic wavelength of an element from the observed SNe

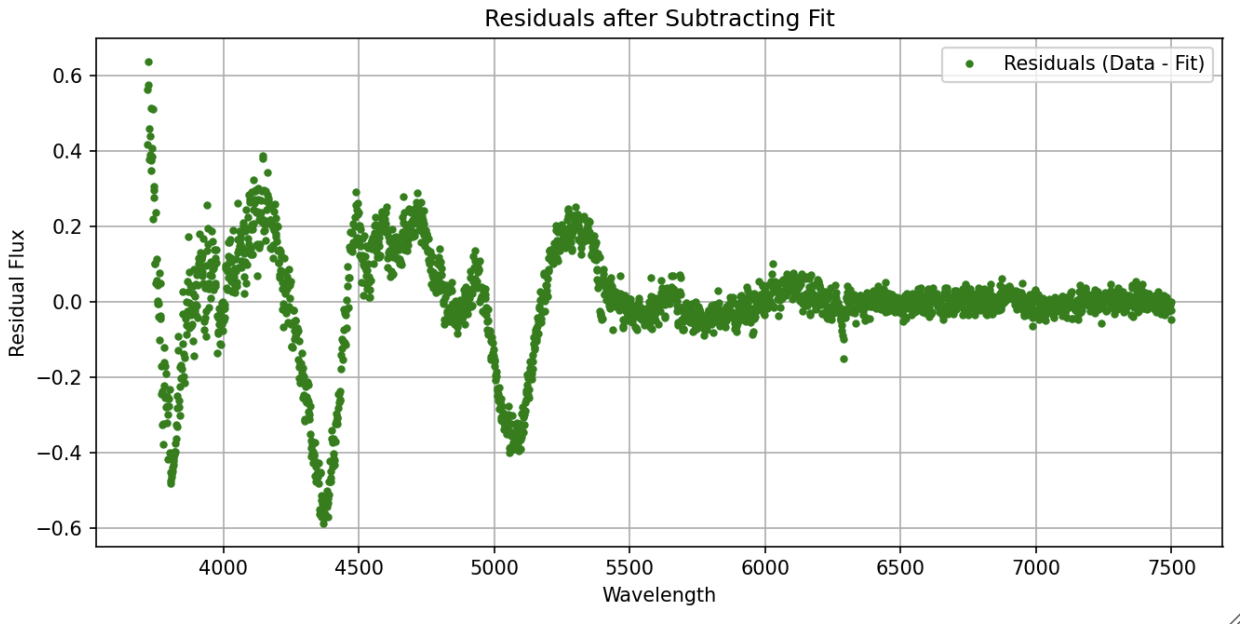
wavelength. I was able to find most of the commands that I used on the internet by searching up the descriptions of the commands. This proved to be extremely useful when trying to fulfill the goals of the function. It checks to see if there are any identified elements in a list that were not missing from the data. If there are some elements found, it finds the element that appears most often and prints the name of the element.

Data Fitting

Next, I conducted an eighth-degree polynomial function curve fit on the observed data. I came to the eighth degree by simply doing trial and error with the lower-degree curve fits. I found that the eighth-degree curve fit modeled the observed spectra the best. For example, for SN1998ab, the eighth-degree curve fit was a very good fit. Error bars were added to the observed spectra plot.



After committing to the eighth-degree curve fit, I created a residual graph to show the distance between the observed points and the modeled fit. Below, I have the residual plot for the same supernova:



Data Filtration

While there isn't a specific "filtering" operation applied to the entire dataset in the traditional sense of removing or selecting subsets of data based on certain criteria. However, several operations effectively filter the data:

1. The use of .notna() in the context of checking if there are any non-missing (NaN) values in the element column indirectly filters the data. This step doesn't remove any data from the DataFrame but checks for the presence of valid (non-missing) data entries. This ensures that subsequent operations like finding the mode are performed on valid data.

Conclusion

In the end, all the SNe showed to have no prevalent elements in their observed spectra. Throughout the project, I ran into multiple problems, some were due to a lack of attention. One of these occurred because I attempted to do subtraction with a float and a string. To ensure data types are correctly formatted before performing operations, I used pd.to_numeric() to convert data to floats, handling non-numeric values appropriately. Another error I encountered was a ValueError and KeyError which was related to the indexing. It had said that -1 was not in range. This issue arose from attempting to access an array at an index of -1, which was possible because of empty data issues or mismanagement of index values. To make sure all the indexing was

correct and checked for non-empty data, I used np.argmin(). I learned a lot from this process of working with spectra and applying programming skills to a topic that I love.