The Machine learning portion of this project consisted of cleaning, tokenizing, and lemmatizing the posts that were in the Kaggle database. The first step was to check to see how many posts there were and how many terms would that be. Then we decided to clean the data to eliminate the three vertical bars that separate the posts, commas, apostrophes, and uppercase letters. For this process, we used the NLTK python library as well as the regular expression (re) syntax. Once we cleaned the data, we then tokenized the posts. Tokenized means that we inserted commas in all the posts so that those posts can individually be evaluated. After that, we then filtered the posts using stopwords. Stopwords is a NLTK.corpus dictionary that is further used to clean up our data of common words that would only be noise for the dataset. We used both for loops and list comprehensions to achieve this goal. Once the data was loaded, we found out the data which is in a list format and not a dataframe has 87000 words. This would not be feasible given to collect given our computer memory. After this we decided to filter this down to the first entry that would allow us to lemmatize that post at a sacrifice of accuracy, but it would allow the machine learning code to be run successfully in Jupyter Notebooks. This list would be 4343 words long, and after that we used the tf-idf feature generation to vectorize the list. This creates a list that finds which words are only in a certain post. We then created a dataframe with the columns being the feature names and the list being the count. We also created a sentiment column using TextBlob so that each word would have a sentiment from negative 1 to positive 1 for being negative or positive. After that, due to memory concerns what we did was create a CSV file that had the tf-idf dataframe to then analyze in a different notebook.

After the new notebook was created, the first thing we did in the new notebooks was to create a feature and a target which we did for our train test split. We also had to convert the dataframe back to arrays for this. We then ran the machine learning models Logistic Regression model. We also ran the Decision Tree, Random Forrest, Ada Boost, Gradient Boosting, Extreme Gradient Boosting (XGB), K-nearest neighbors (KNN), Light Gradient Boosting, and support vector classification models. Because of the sacrifice in accuracy that was mentioned above, the most accurate model that we found was 21% accurate. Also, all the models had that accuracy, so we just chose the logistic regression model for our predictor. After we chose which model to use for the predictions, we decided to save the model as a python pickle file to be opened and used in the predictor. Our predictor was us using the pickle file to create a prediction-based model that would take any input text and take it through the system to bring back a prediction of a personality type. For that we followed some of the same processes as the NLTK notebook. Cleaning the user inputted text, looping through the text to create an input, creating a sentiment analysis for that input, and making a prediction on the personality type based on that input.

Limitations:

The limitations of this project when it comes to machine learning is that we had a lot of data to process, and it ended up eating a lot of computer memory. There were a lot of times when I tried to run code in Jupyter and the computer or even Google Colab could not process all the data and it threw a memory error. I then tried to use a different library to process all of it. I used the spaCy library to process the data much faster and then that ended up creating a csv of all the data without having to cut any of it. All the data ended up being 4.32GB and that would destroy computer memory. I also wanted to try to check to see if I could save the data off to a JSON file so that it would be easier to load and try running that but for the scope of this project I ran out of time to do that. In the future, I would suggest making sure that the computer can handle the data and using some of the other strategies to find out if all the data can be processed. This would give the user much more accurate results. I also think that a user could use other libraries that are designed for this. One of the libraries I came across was GenSim and using Google BERT to transform the data.