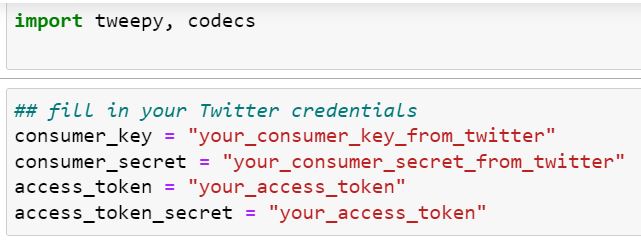
**BDM 3014: PROJECT ON TWITTER TEXT MINING**

**GROUP A**

1. **AADARSHA CHAPAGAIN**
2. **JYOTI SHUKLA**
3. **PIYUSH BHATIA**
4. **PRITI BHALE**
5. **RISHI PHANEENDRA VARMA**
6. **SREYA TREESA JOHNY**

The dataset from Twitter's API has been used for this text mining project. This has been done in three steps. Firstly, we built a corpus, the massive volume of textual content capable of giving an insight. Here we have used Twitter to build a corpus. Secondly, we analyzed the collected text of the Tweets using Sentiment Analysis, the text mining that recognizes and extracts subjective data from the source material. Finally, we visualized our analysis result using matplotlib and pandas' libraries.

1. We imported the tweepy library to get the most recent Tweets containing our search keyword from the Twitter Rapid API and the codecs library to write the text of the Tweets to a .txt file. In order to collect Tweets from the search API, at first, we must obtain Twitter's approval, and for that, we have to sign up as a developer and get the respective consumer keys and access tokens. After getting the credentials, we stored those Twitter credentials in the respective variables.



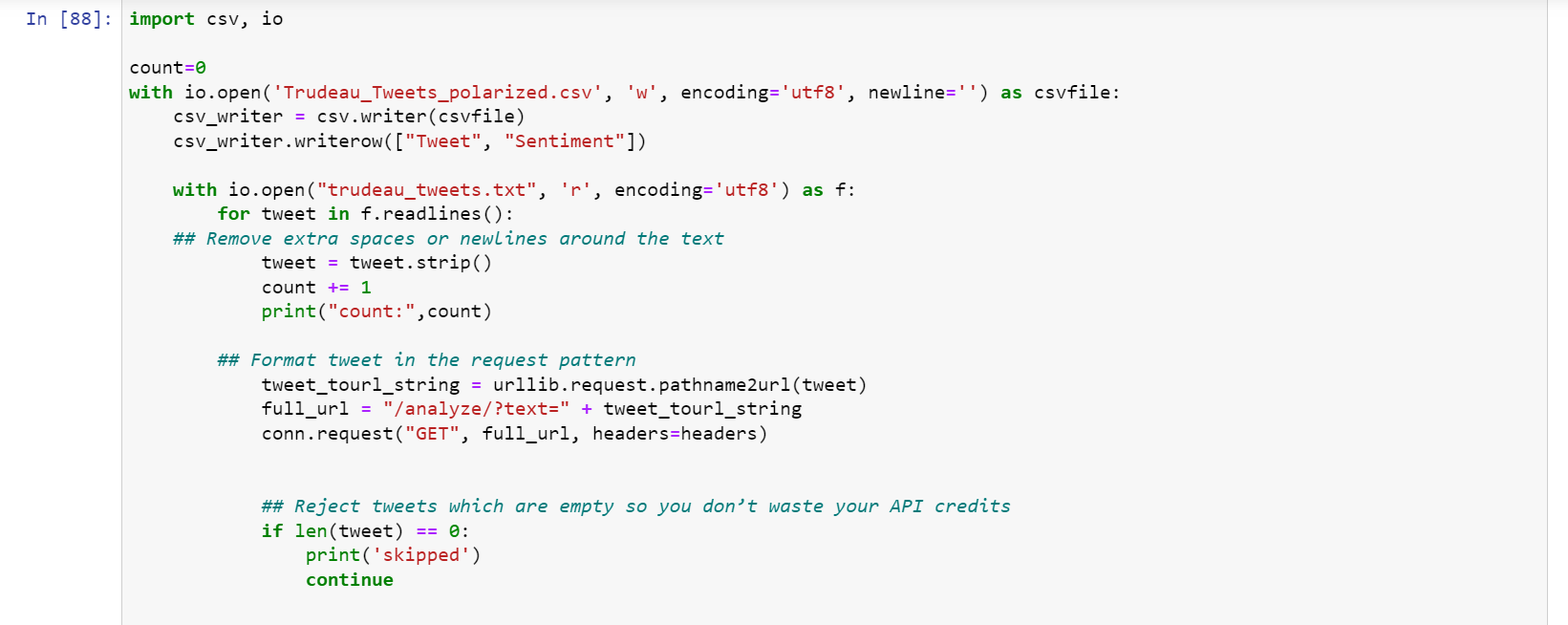
1. After setting up an instance of the REST API, we built a search query by adding our search keyword 'Trudeau to a variable q and added other parameters like language, the number of results we would like to receive, and the time frame for searching. This search query has been stored into a variable as well. Then, using the codecs library, the text of the Tweets was written to a .txt file and cleaned the collected tweets by iterating through it using a for a loop.



1. So once we have collected the text of the Tweets that you want to analyze, we can use more advanced NLP tools to extract information from it. Sentiment analysis is an excellent example of this since it tells us whether people were expressing positive, negative, or neutral sentiments in the text. For sentiment analysis, we are going to use RAPID API. Once we have got the App key and Application ID, insert them into the code below to start our first call to the API from the Python shell.

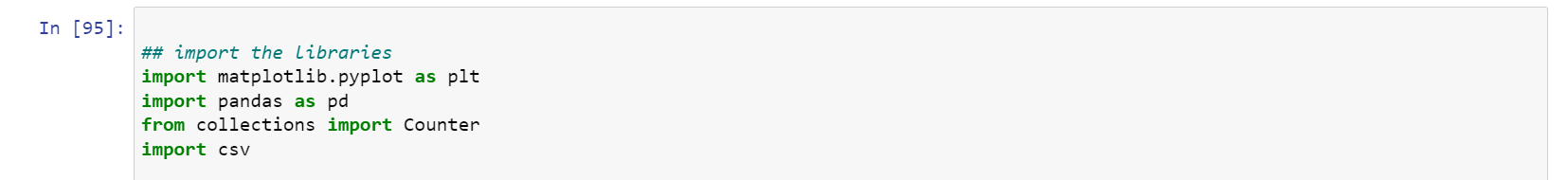


1. So now we need to analyze our corpus from step 1. To do this, we need to analyze every Tweet separately. The script below uses the [io module](https://docs.python.org/2/library/io.html) to open up a new .csv file and write the column headers "Tweet" and "Sentiment," Then, it opens and reads the .txt file containing our Tweets. Then, each Tweet in the .txt file sends the text to the RAPID API, extracts the sentiment prediction from the JSON that the RAPID API returns, and writes this to the .csv file beside the Tweet itself. This will give us a .csv file with two columns — the text of a Tweet and the sentiment of the Tweet, as predicted by the RAPID API. We can look through this file to verify the results and visualize our results to see some metrics on how people felt about our search query.

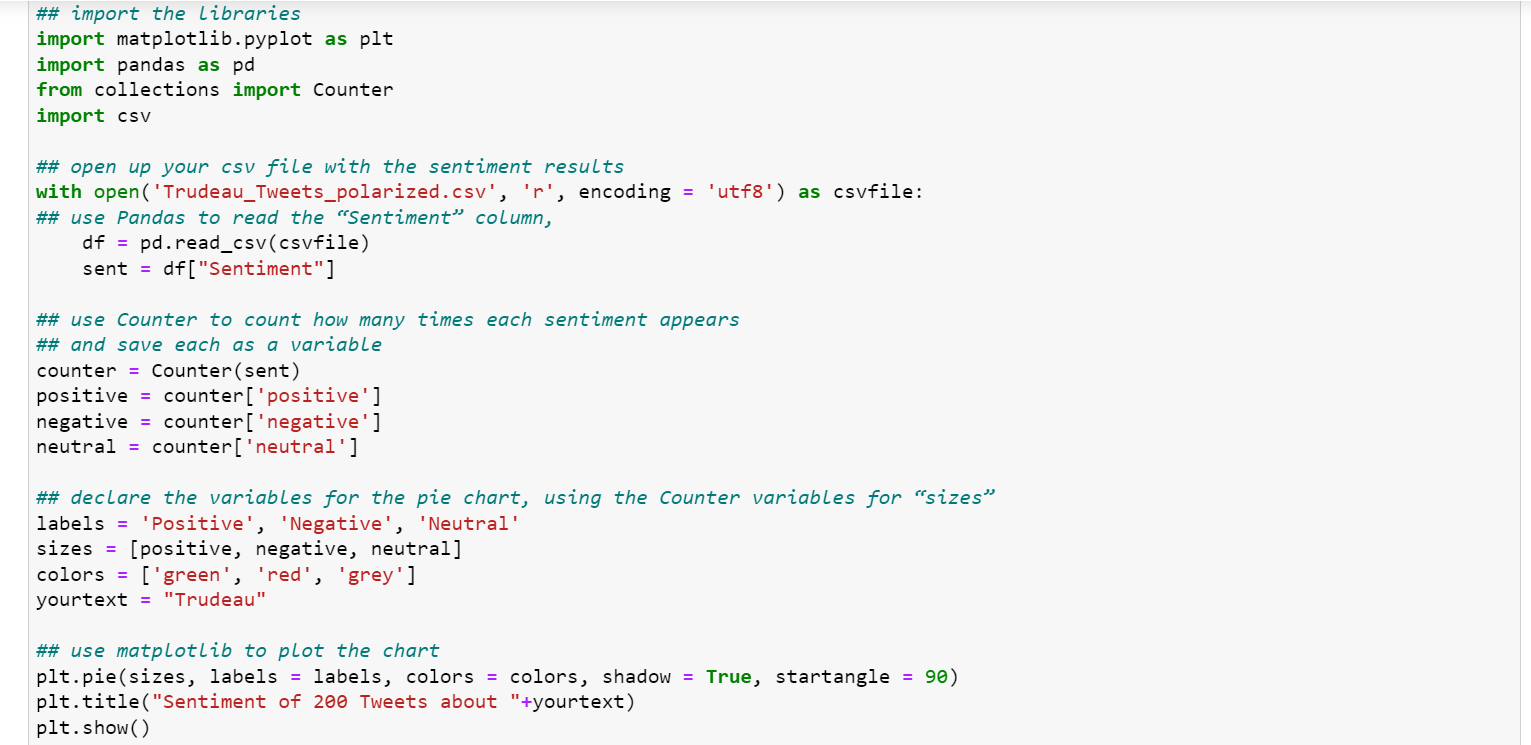




1. So far we have used an API to gather text from Twitter and used our Text Analysis API to analyze whether people were speaking positively or negatively in their Tweet. At this point, we have a couple of options with what you do with the results. You can feed this structured information about sentiment into whatever solution we are building, anything from a simple social listening app or even an automated report on the public reaction to a campaign. We could also use the data to build informative visualizations, which we will do in this final step. For this step, we're going to use [matplotlib](https://matplotlib.org/) to visualize our data and [Pandas](http://pandas.pydata.org/) to read the .csv file, two Python libraries that are easy to get up and running. You will create a visualization from the command line or save it as a .png file.



The script below opens up our .csv file and then uses Pandas to read the column titled "Sentiment." It uses Counter to count how many times each sentiment appears, and then matplotlib plots Counter's results to a color-coded pie chart.



If you want to save your chart to a .png file instead of just showing it, replace plt.show on the last line with saving fig('your chart name.png'). Below is the visualization we ended up with (we searched "Trudeau” in step 1).

