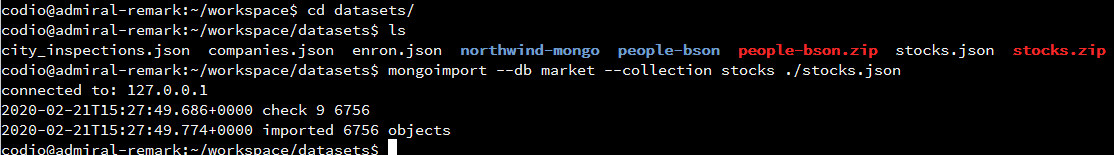
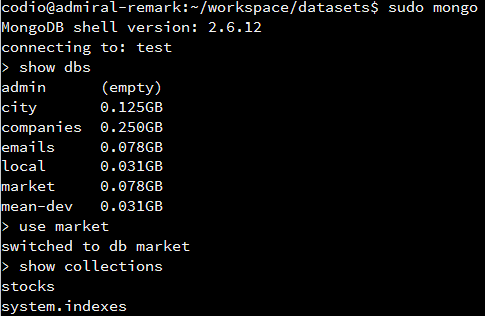
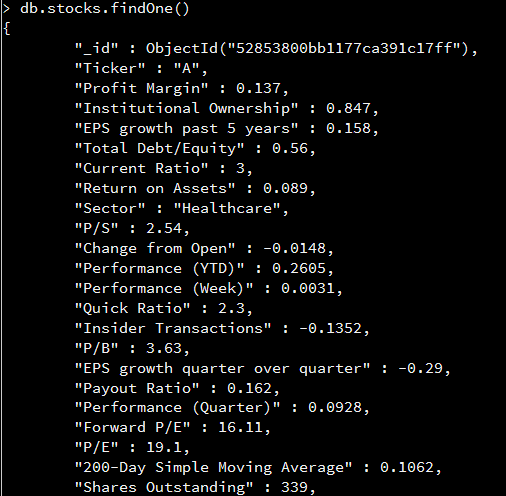
1. **A.** For this assignment, I had to use the mongo import tool to create a database named “market” and a collection named “stocks”, loaded with documents from the stocks.json file. In one terminal, I started by typing “sudo mongod” and this initiated the mongo host. In another terminal, I changed to the “datasets” directory and imported the desired database and collection along with the appropriate documents from the stocks.json file. This can be seen below:



To show that the database and collection were imported correctly, I connected to the mongo client, switched to the appropriate database, and showed the collections in that database, as seen below:

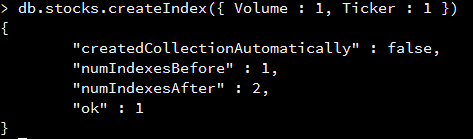


Performing a findOne() query and receiving results proves that the json file was imported correctly too. This can be seen below:

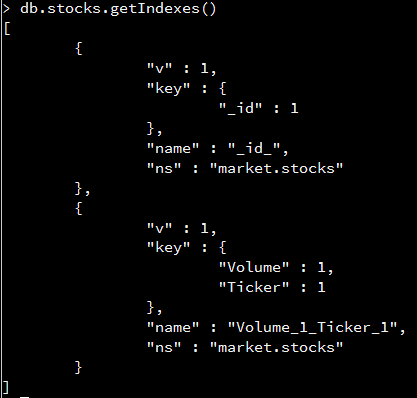


If no results were found, then that would mean that the import was not successful.

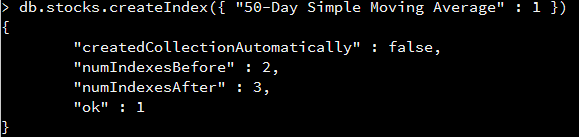
**B.** Indexes can be used to create shortcuts for queries, which results in quicker search times. Single indexes are used for quickly finding a document with one key and compound indexes are used for finding more specific documents that match multiple criteria. Indexes should be created based on the tasks at hand and what makes sense for finding results quickly. Based on this project, creating a compound index for the keys “Volume” and “Ticker” would prove to be beneficial. Creating this compound index with createIndex() can be seen below:



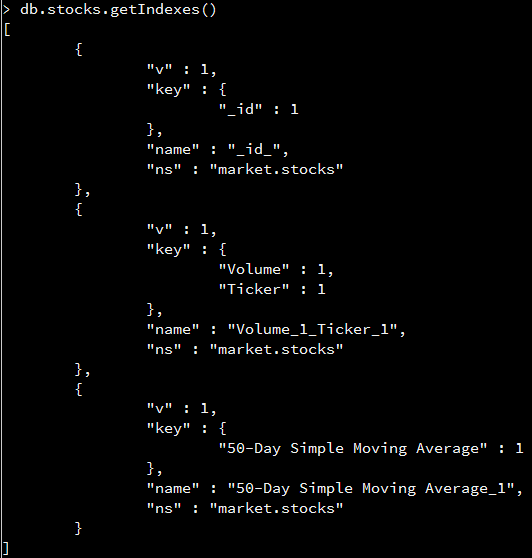
To show that this index was created successfully, I performed a getIndexes() function, which can be seen below:



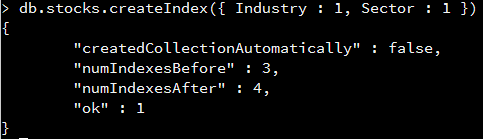
Later on in this project, a single index can be beneficial for “50-Day Simple Moving Average”. The creation of this can be seen below:



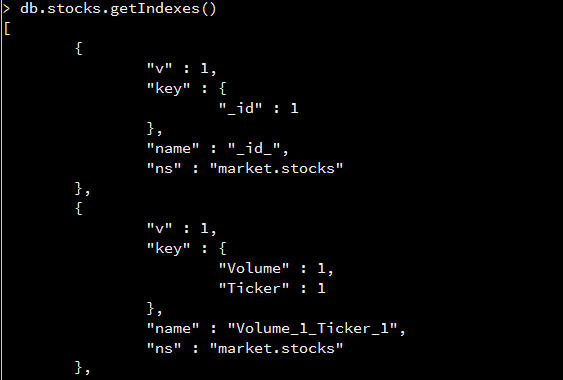
Performing another getIndexes() function shows that this new single index was successful:

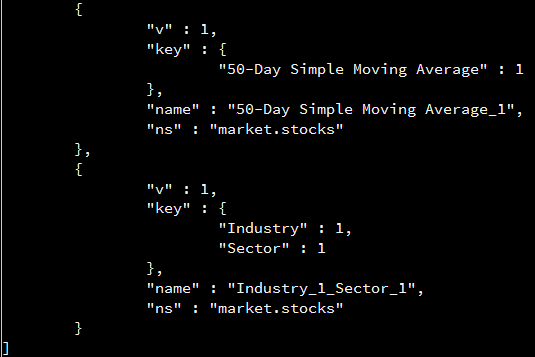


Lastly, I believe that creating a third index would be beneficial to this project, this one being compound for the keys “Industry” and “Sector”. The creation of this can be seen below:

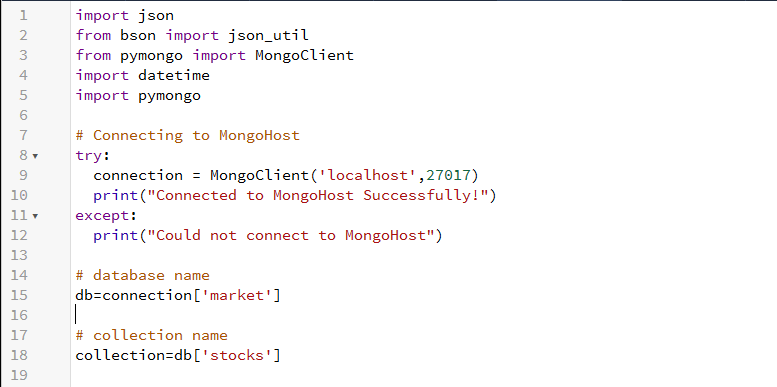


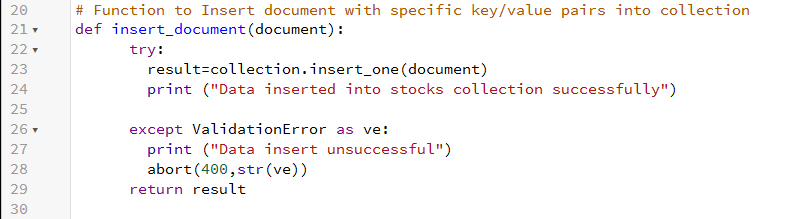
Again, to show this was successful, I performed a getIndexes() function, which can be seen below:



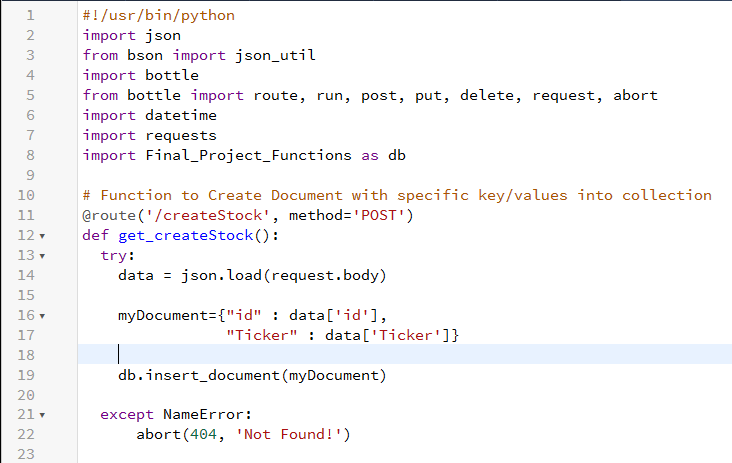


1. **A.** For this next part, I had to use an insert method to create new key-value pairs into documents. The following code shows how this is set up:



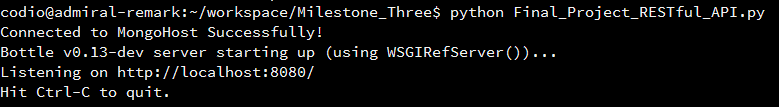


This shows how to connect to the mongo host and then references the market database and the stocks collection. The insert\_document() function uses the insert\_one() function, which is part of the RESTful API. The insert\_one() function can be seen in the code below:



The above code shows how get\_createStock() utilizes the insert\_document() function mentioned before. This method is used to create documents and store data within that document. To test this out, I opened up one terminal and typed in “sudo mongod”, which initiated the mongo host. In another terminal, I typed in “sudo mongo” to initiate the mongo client. In a third terminal, I changed my directory to my project and then executed the file for my RESTful API. This last part can be seen below:





By going back to the terminal with the mongo host, we can see that there are now two connections:



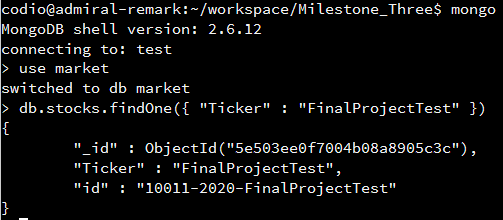
I opened up a new terminal and used a curl command along with POST to create a new document, which can be seen below:



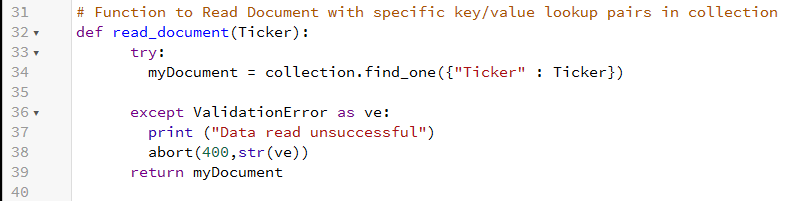
This can be confirmed by viewing the terminal running the Bottle framework and seeing the following output:



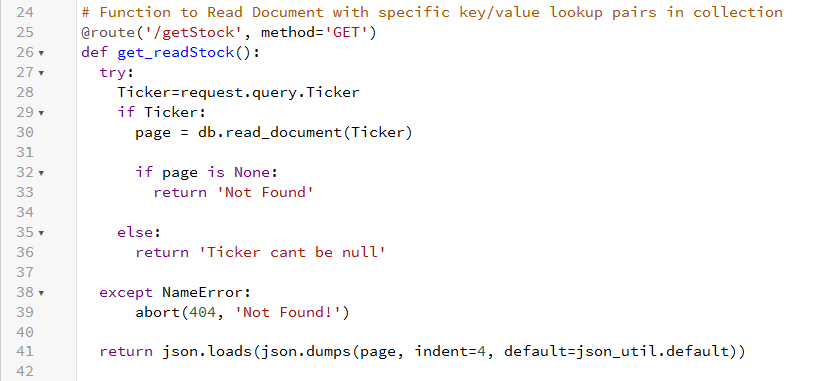
To prove this was a success, simply accessing the mongo client and performing a query on the “Ticket” key can reveal the document that was just added:



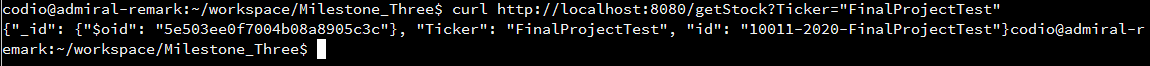
Another way to confirm this creation is to perform a read function. The read function’s code can be seen below:



The RESTful API for this code can be seen below:



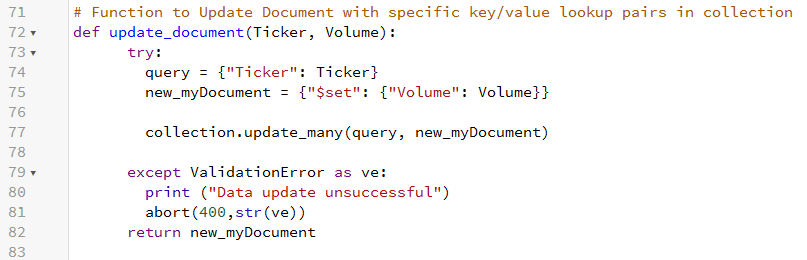
Instead of creating a document, this gets the document based on the criteria specified. This alternate method can be seen below:



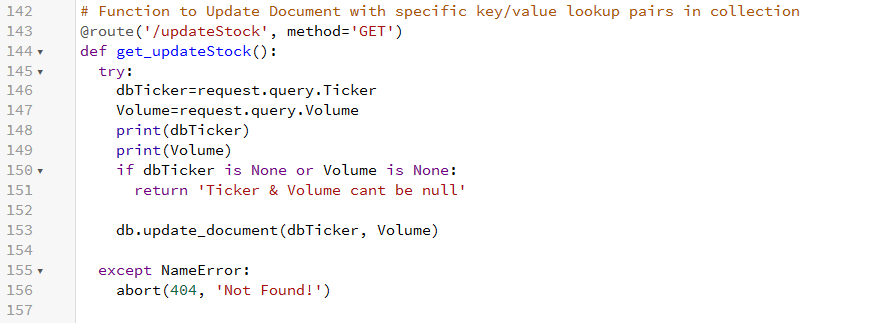
Again, to confirm this, navigate to the Bottle framework terminal and view the results:



**B.** This next task requires us to update the value of the key “Volume” by identifying it with “Ticker”. To update just the “Volume” when searching for “Ticker”, the update function below can be used:



In order for this to work, we must update the recently created document using the following RESTful API:



The following command finds the “Ticker” and adds the key “Volume” with a value of 25:



The Bottle framework shows this was a success:



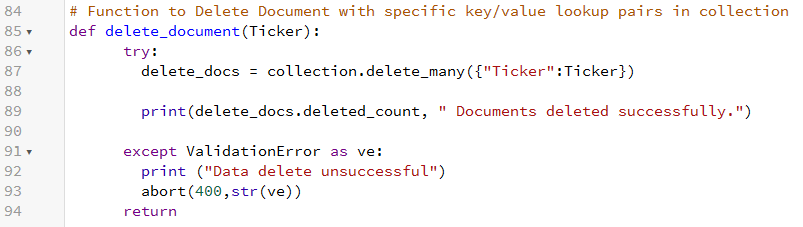
Performing another curl command to get information shows the update document:



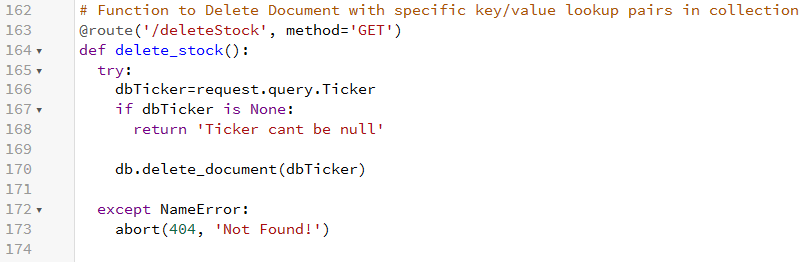
Again, this proved to be a success:



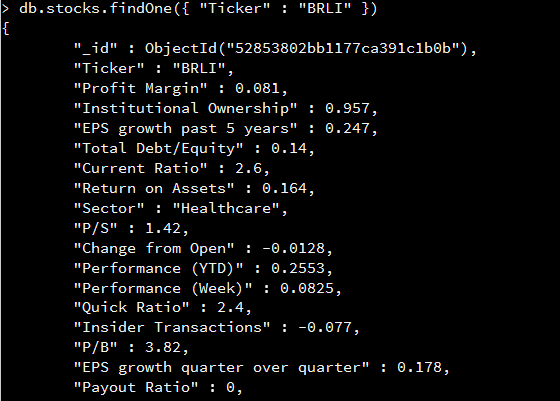
**C.** The next task asked us to delete a document using the delete function, which was created below:



The delete\_many() function above uses “Ticker” as the identifier. The RESTful API that goes along with this can be seen below:



This deletes a document, providing that it is not null. Since the rubric asked us to delete a document with the “Ticker” as “BRLI”, we can access the mongo client and perform a query, seen below:



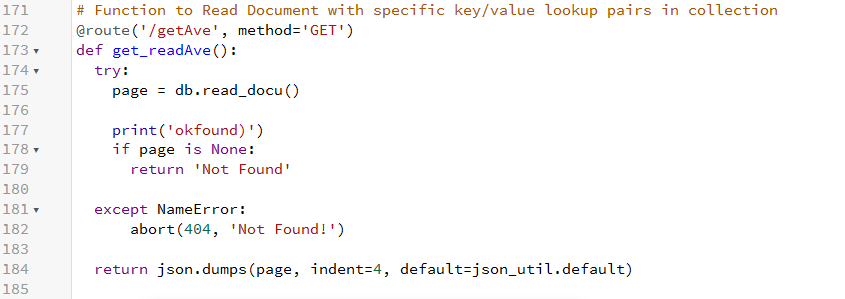
To delete this document, we can perform the following command:



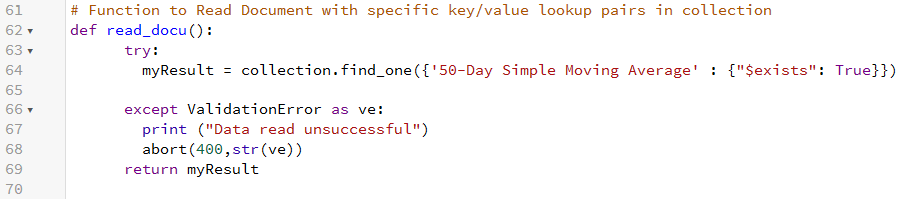
Performing the same query as before will show that the document no longer exists:



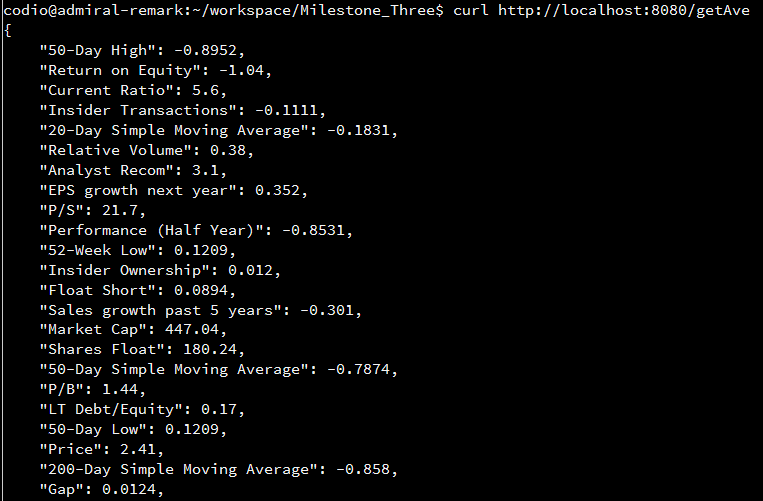
1. **A.i.** For this next task, we had to find key/value pairs for “50-Day Simple Moving Average”. This was done using the Python code below:



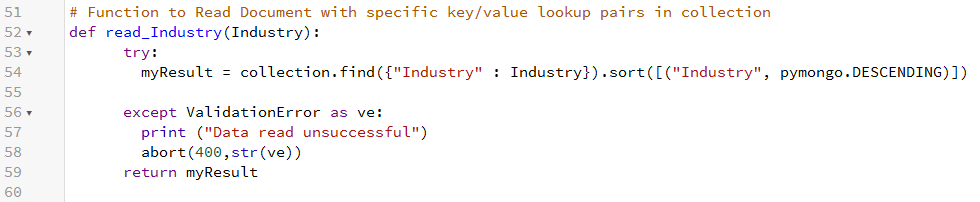
The following image shows the RESTful API that goes along with this:



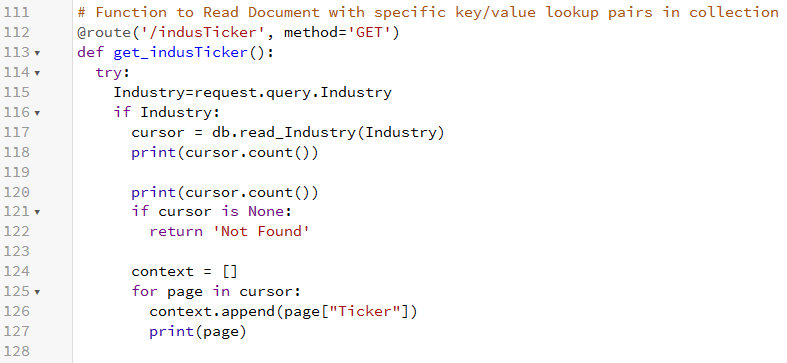
The command below helps show how this function works:

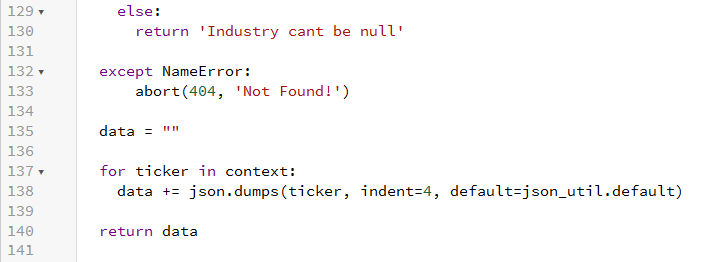


**A.ii.** For the next task, we were asked to create a function that takes input as a string and match the key “Industry” and then return the “Ticker” values based on those matched items. The code below shows how this is accomplished:

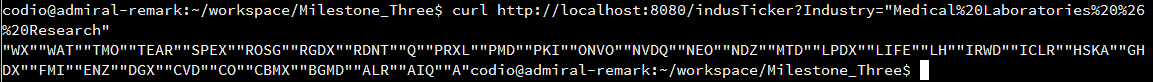


The RESTful API that goes along with this function can be seen below:



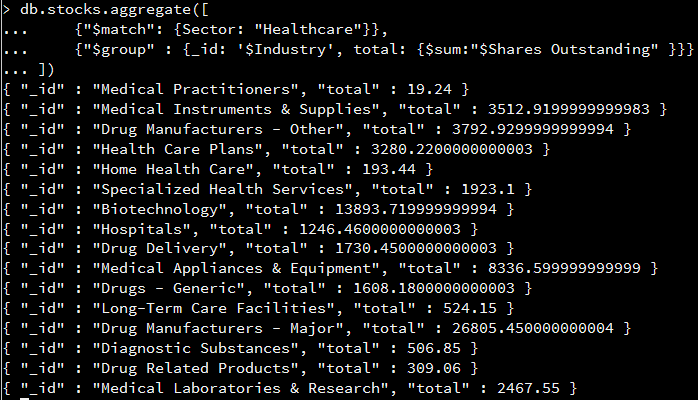


According to the rubric, we can use the “Industry” value as “Medical Laboratories & Research”. Therefore, the following curl command can produce the desired output:



In the above command, %20 represents a space and %26 represents the “&” sign. These results show all the “Ticker” entries that match the “Industry” value entered.

**B.** For this next task, we had to use an aggregate pipeline to find documents for the key “Sector” and also returns the total outstanding shares grouped by the key “Industry”. This can be seen below:

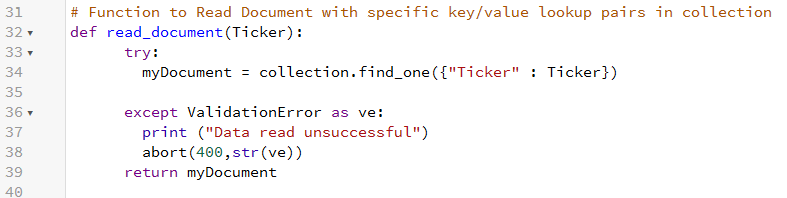


In this pipeline, the returned documents had to match the “Healthcare” sector and then they were grouped by the “Industry” along with the total sum of the outstanding shares.

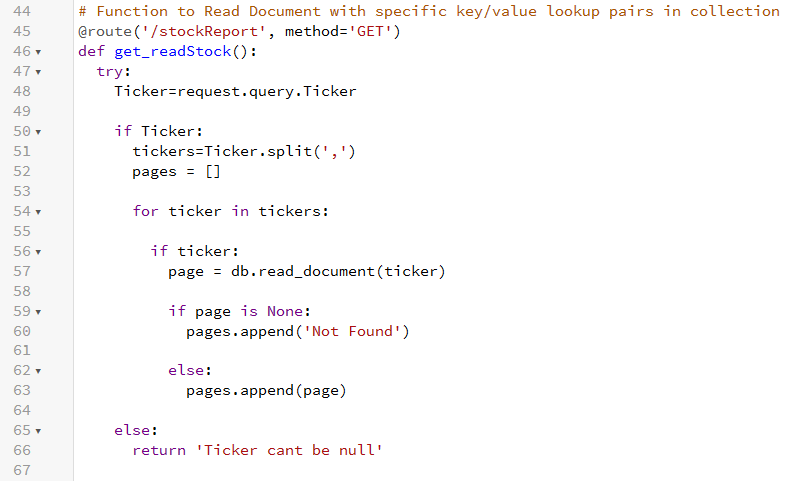
1. **A.** In question 2 of this assignment, I showed how to develop a RESTful API with Python. The images in question 2 showed just how I did this, and the provided file shows the Python code used with the “market” database and the collection “stocks”.

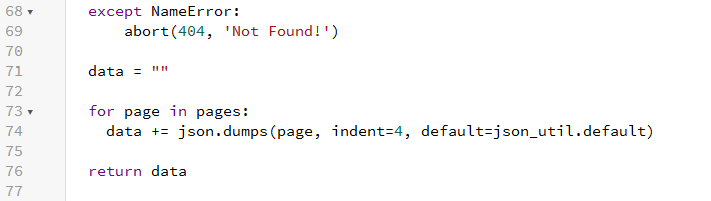
**B.** In question 2 of this assignment, I showed how to implement CRUD functions with a RESTful API. Please refer back to this question above to see images or review the provided Python code.

**C.i.** This task required us to select and present specific stock summary information by a user-derived list of ticker symbols. A GET command is used with the following Python code:

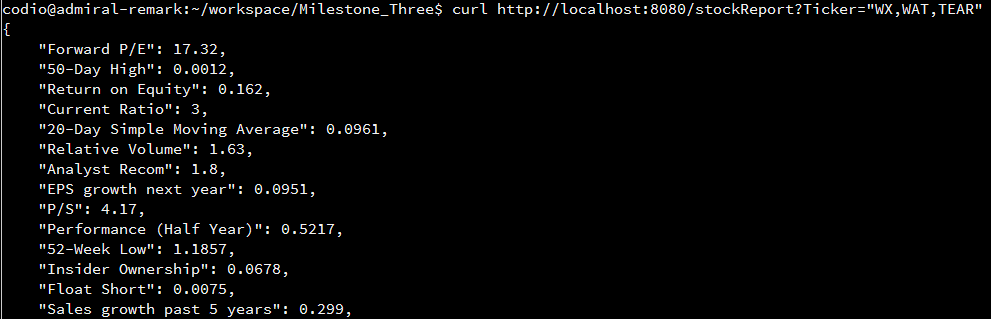


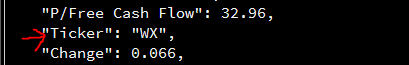
The following code shows the RESTful API for presenting this information:

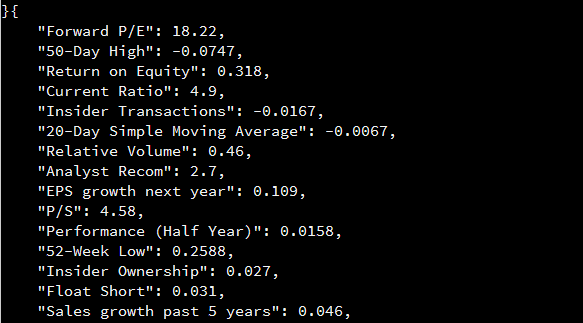


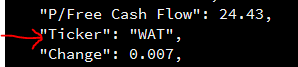


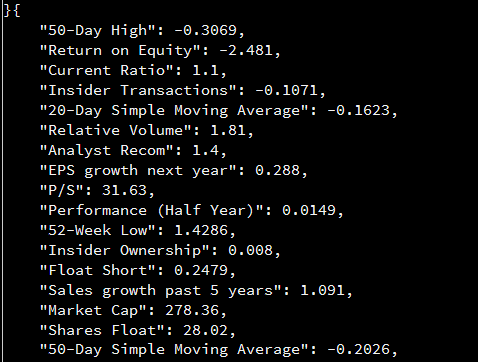
The get\_readStock() function uses a split() which helps separate each Ticker by a comma. The read\_document() function then stores data into an array which is returned at the end of this function. Using the symbols, “WX”, “WAT”, and “TEAR”, the following command can return the results:

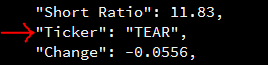






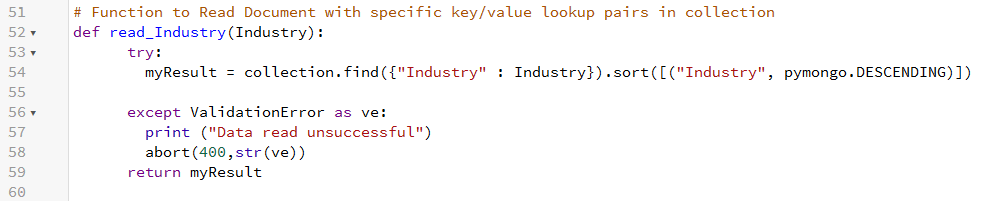




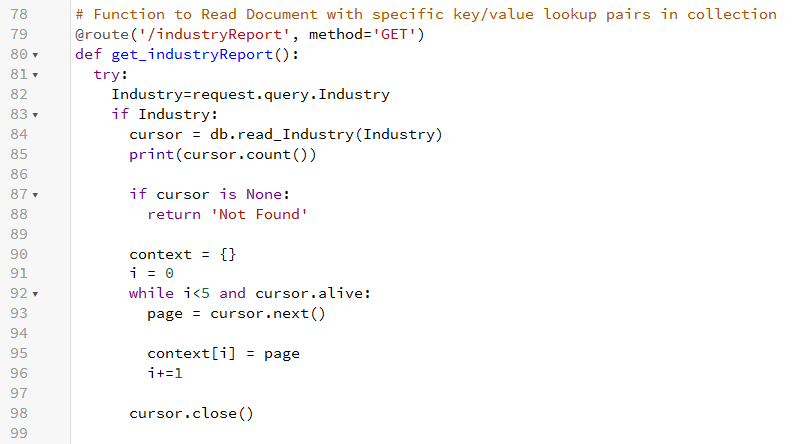


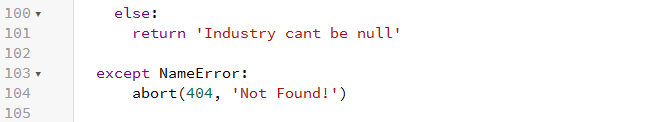
Each pair of images above shows the beginning of the results and the specific “Ticker” that corresponds to “WX”, “WAT”, and “TEAR”, respectively.

**C.ii.** For this next task, we were required to report a portfolio of five top stocks by a user-derived industry selection. The Python code I used for this is below:



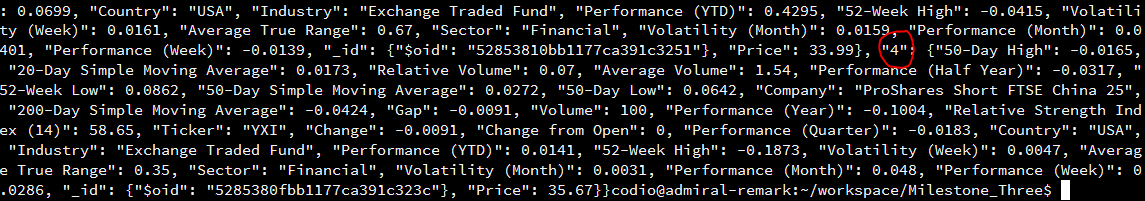
This function performs a find() query so that multiple documents can be returned. Within this function, “Industry” was used as a parameter and was sorted in descending order. The RESTful API that goes along with this can be seen below:



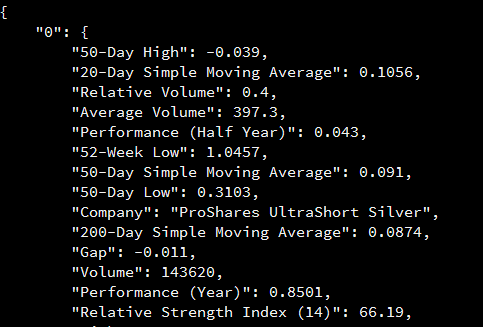


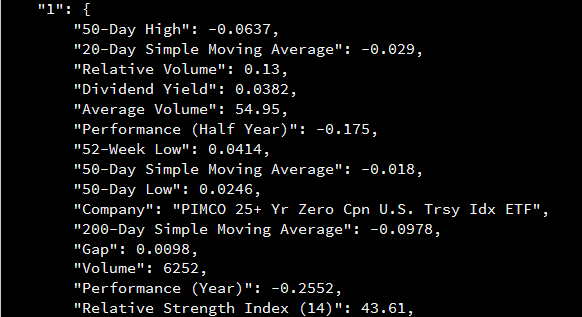
The get\_industryReport() function used an empty dictionary to add pages as long as there is less than 5 in the while loop. Once there is enough data added to this dictionary, the function returns the data and ends. The following command shows the results:





If we look at the Bottle framework terminal, we can see that these results are placed in index order in a much friendlier readout:





The last printout in this terminal shows the success:

