**NLG Term Project Report**

Group 1

Cesar Castellon Escobar, Robert Fogarty, Braden Weaver, Conrad Testagrose, Michael Moss

University of North Florida

CEN 6940

Dr. Shang

**Contents**

[Introduction 3](#_Toc102322861)

[Related Approaches 3](#_Toc102322862)

[Data and Analysis 4](#_Toc102322863)

[Python NLG & RNN Model Integration 7](#_Toc102322864)

[Tableau Dashboard 14](#_Toc102322865)

[References 18](#_Toc102322866)

[Appendix A 20](#_Toc102322867)

[Appendix B 23](#_Toc102322868)

[Appendix C 25](#_Toc102322869)

[Appendix D 28](#_Toc102322870)

# **Introduction**

*Project Overview*

Our group wanted to create a dashboard utilizing Natural Language Generation (NLG) to analyze and create a narrative for financial data. Proprietary technology like this exists, but there is space for an open-source tool.  The goal, therefore, is to create a robust open-source tool that utilizes NLG to summarize financial data visualizations. Our proof of concept will use publicly sourced financial data to create an NLG powered dashboard that consolidates quarterly financial data and displays it in a simple easy to read format.

*Project Goals*

The following are more specific goals that summarize what our group would like to accomplish:

* Build a visually appealing dashboard.
* Visuals and Natural Language narrative update automatically when new data is entered providing the ability to keep up with large amounts of data.
* Display graphs and important data (revenue, cost of revenue, net income).
* Reports can be accessed at any moment and at any time.
* Ability to navigate between provided companies.

Ultimately, this dashboard would provide value to a company by allowing managers/ decision-makers to analyze a visual representation of relevant financial information immediately. This automated process allows for quick decision making which saves time and reduces costs. This dashboard could also be shared to other functional areas of the company or even to an outside firm enabling efficient and timely sharing of data.

# **Related Approaches**

*Client Requirement Feasibility*

Although there were no specific client requirements, our group was provided resources that were not used. The first resource being the EU Banking dataset. After analyzing the dataset, it was concluded the data was too harsh to fully comprehend and understand. Instead, we opted for a simpler dataset that would allow us to put more focus on the NLG components. Next our group chose to focus on Tableau as a local instance rather than AWS. Processing power local as AWS was deemed to be too costly. Lastly our group initially analyzed Power Bi applications and integration; however, after further analysis found we could not obtain a license to use desired applications.

# **Data and Analysis**

*Summary of Data*

The data source that will be used for this project is Microsoft yearly financial data. Collected directly from yearly *“10K”* reports available on Microsoft's website. In addition, we implemented Apple data to show proof of concept with multiple companies. (See **Appendix A**) We have allocated 3 years of data from the balance sheet, income statement, and the stock price that will be used to develop the dashboard for our project. We have chosen to use Microsoft and Apple data due to there being over ten years of data publicly available and the data is relevant to the project. Additionally, both of their financial statements follow Generally Accepted Accounting Practices (GAAP) allowing us to utilize the dashboard and NLG on additional companies that follow the same standards. The business dashboard will contain graphs of the data we have selected for the project. Alongside the graphs on the dashboard, we also intend to utilize the Tableau extension Api to display our NLG (Natural Language Generation) model output to the user describing the data.

The three main sources of Microsoft/Apple data are the balance sheet, income statement, and historical stock price.  The following describe the primary variables each financial document is comprised of:

* The primary variables in the balance sheet are total assets, total liabilities, and total stockholders’ equity which represent the value of the company.
* The primary variables on the income statement are total revenue, cost of revenue, net income, and diluted eps. These represent the sales and income of the company.
* The primary variable on the historical stock price is the close which represents the value of one share of Microsoft/Apple at the end of each day.

*Tables 1 & 2* describe the amount of data in each company’s financial documents.

*Table 1*: Microsoft Data Description

|  |  |  |  |
| --- | --- | --- | --- |
| Microsoft | | | |
| **Financial Document** | **Columns** | **Rows** | **Total Records** |
| Balance Sheet | 4 | 34 | 136 |
| Income Statement | 4 | 21 | 84 |
| Stock Price | 7 | 757 | 5,299 |

*Table 2:* Apple Data Description

|  |  |  |  |
| --- | --- | --- | --- |
| Apple | | | |
| **Financial Document** | **Columns** | **Rows** | **Total Records** |
| Balance Sheet | 4 | 34 | 136 |
| Income Statement | 4 | 20 | 80 |
| Stock Price | 7 | 757 | 5,299 |

*Data Preprocessing Steps Applied*

Along with the target financial data we also processed and prepared the annual report as an input for further steps concerning the NLG model. The preprocessing steps used on the financial data followed the normal steps of data preprocessing which includes data cleaning, data transformation, data reduction, and quality assessment. The first step was to start by exporting the balance sheet and income statements from the *“10k”* report excluding all other financial data available. Making each of the reports its own file. Then unmerging cells and removing blank lines to format the reports so that they are readable and importable into Tableau. Followed by combining multiple years side by side. Then using Tableau to create each graph and combine them on a dashboard. The general trends (increase/decrease) will be determined and fed to our NLG model.

The NLG is trained on text that follows the context and format that we wish for it to output on the dashboard. We have prepared a .txt document “*training set.txt*” containing some of the text from the Microsoft financial documents and wrote our own short .txt file to obtain text that fits with our project slightly better. (See **Appendix B** for training sets) The text is fed into the model in a character-by-character format. The text data is loaded from the training set and the preprocessing steps just prepare the string of characters to be fed into the model. The NLG should then return the written description of the trend to the user on the dashboard after being fed the text trend we feed it.

*RNN Vs. LSTM*

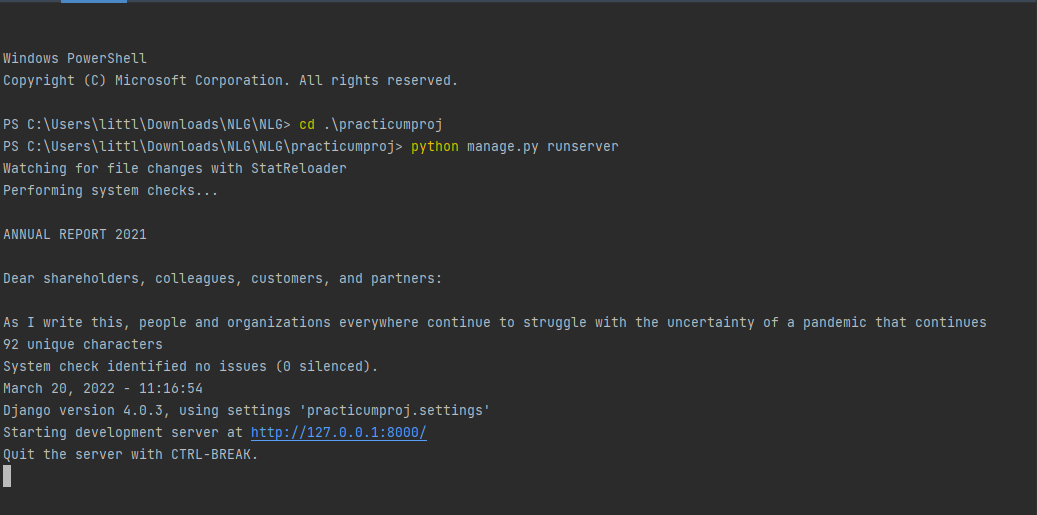
We investigated/coded/implemented using either an LSTM (Long Short-Term Memory) or an RNN (Recurrent Neural Network) to be the NLG machine learning model. We have run a few tests on both LSTM and RNN and the results of the RNN seem to be a better fit for our project.

Below is a sample output from an LSTM model and the bold sentence is the random seed from the text:

“LSTM Output: **if he's going to experiment let him. But I told this guy, "You can't”** tou the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to the wou aan to th”

Our group discovered from the above LSTM output that the text was not as promising as our output for the RNN displayed in *Figure 1*.

*Figure 1*: RNN Output



 The RNN which takes longer to train provides text that is more understandable. The RNN was easier to save and reload without errors or issues on the backend. In almost all areas the RNN was seen as more valuable which is why it was used going forward.

# **Python NLG & RNN Model Integration**

*Django Integration*

Our group chose to integrate the trained model into the tableau html and JavaScript extension framework. One of RNN our trained models with best loss score **(**0.053**)** was deployed via Django framework which serves up the necessary model files and runs them on a webserver (see *Figure 2*). After further implementation, it was discovered that Django contained several bugs. (See **Appendix C** for full Django process and implementation)

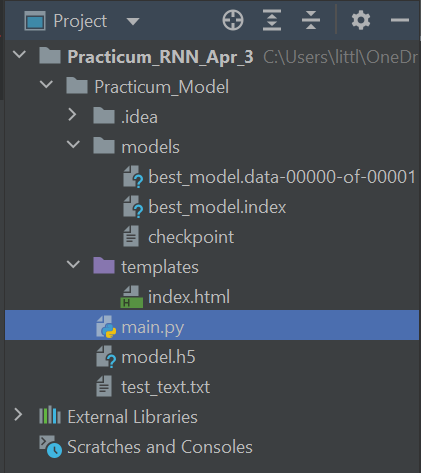
*Figure 2*: Django Framework



*Flask Integration*

After careful examination and further research, the decision to move to another python-html integration library was made. The library used for the new implementation is known as Flask. Using Flask was much more streamlined and only required minor changes to the project directory. It also allowed for a cleaner Project directory allowing for fast implementation and a much more streamlined integration. *Figure 3*displays project directory. Using Flask, the addition of index.html was the only added file to the project directory, making the entire folder much easier to navigate and manage. To run the entire process of text generation to the html file one must run main.py either in a Docker container, an IDE, or in a python terminal. The model is loaded from the saved weights and the text generation can be initiated using a user supplied input string.

*Figure 3*: Flask Project Directory



*Figure 4* displays the only changes needed for the python program to run the script in html using Flask. When using Django there were numerous files needed and many modifications needed for the script to output the text. Using Flask, the output html file with the appropriate calls for the script along with what should be done with the user supplied information, is all that is needed using two methods, home () and result ().

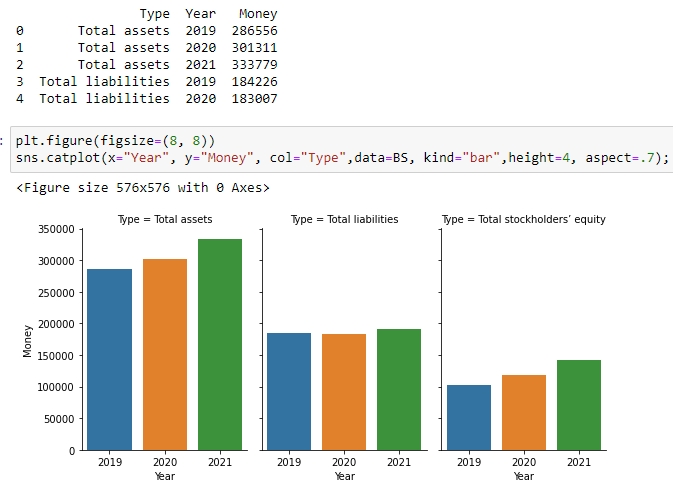
*Figure 4*: Changes to Use Flask



*Graph and Seed Generation*

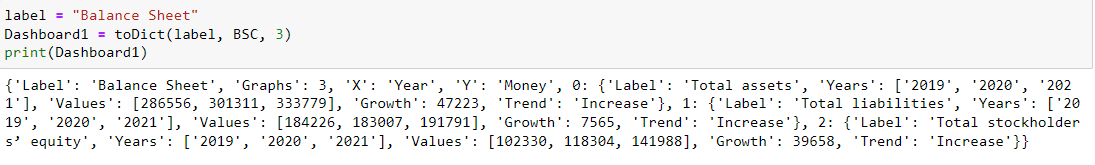
Because our group was unable to get the data we needed directly from Tableau, we recreated the dashboard with Python using the Pandas, Numpy, Matplotlib, and Seaborn libraries. The .csv files are read into Python, processed, plotted, and a Python Dictionary is created to send to the NLG. This Dictionary contains the information needed to seed the NLG. This information includes the title, axis names, labels, values, and trend. Figure 5 is first graph from the dashboard in Python.

*Figure 5*: First Graph from Python Dashboard

**

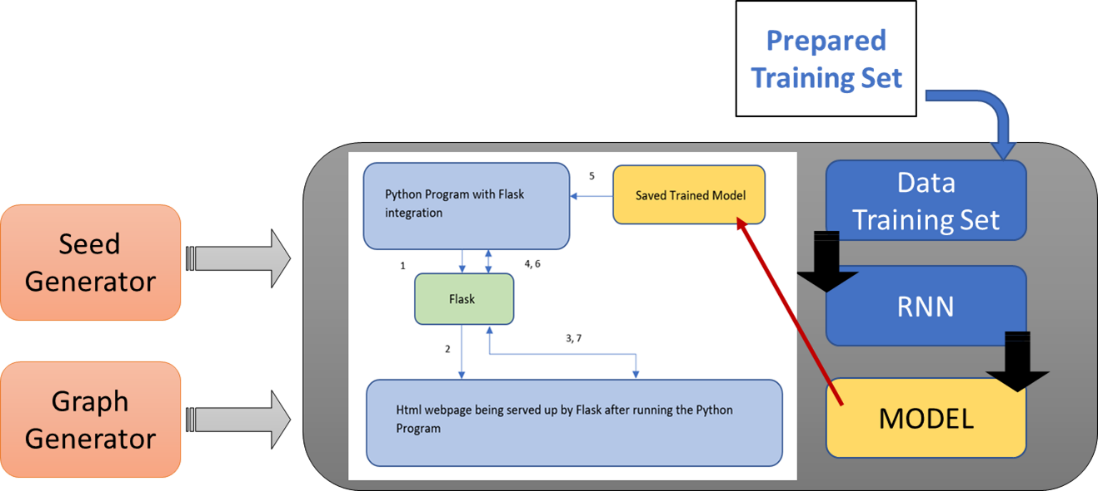
To recreate the dashboard, the data had to be imported and extracted from the Balance Sheet.csv. The simplified data is then used by Matplotlib and Seaborn to produce the graph. Once the graph is recreated in Python, a dictionary can then be constructed and passed over to the NLG side. This Dictionary (see **Appendix C**) will contain all the relevant information that the NLG will need to create a description of the graph or graphs. The toDict function requires the label, modified data frame, and number of subgraphs, and returns the Dict for that graph. *Figure 6* shows the process used to create the graph in *Figure 5*. The function is designed to work for any graph created in python, provided the format of the data frame is structured correctly. For a graph with no subgraphs Column 0 is the X-axis and Column 1 is the Y-axis. If a graph has subgraphs, then Column 0 is, instead, the Subgraph label. Columns 1 and 2 become X and Y respectively. Passing in a ‘0’ for the number parameter means that the Dictionary is for a series, and all of the values do not need to be stored in the Dictionary. Types are explicitly cast as Integer to avoid errors with using JSON to transfer the data to the NLG side. JSON does not get along with the Numpy Int64 type.

*Figure 6*: Balance Sheet Dictionary

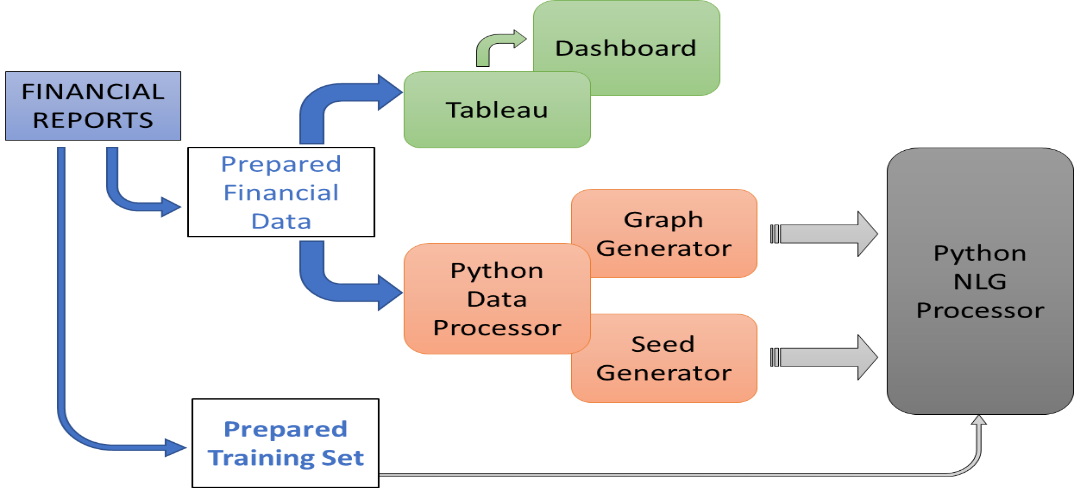


This structure also allows for easy descriptions of multipart graphs. A dict.keys() operation will return dict\_keys([‘Label’,’Graphs’,’X’,’Y’,'0', '1', '2']), meaning that there are 3 subgraphs to describe. Dict[‘0’].keys() returns dict\_keys(['Label', 'Years', 'Values', 'Trend', 'Growth']) allowing access to the unique subgraph information. This info can then be fed as a seed to the NLG to generate the descriptive text and fill in for values that cannot be generated. To integrate the seed generator and graphs with the NLG in the dashboard, we used JSON files for the Dictionaries and .png images for the graphs. *Figures 7 & 8* show the process of preparing the data so that the Python NLG Processor is fully functional.

*Figure 7*: Processes Needed for Python Dashboard

**

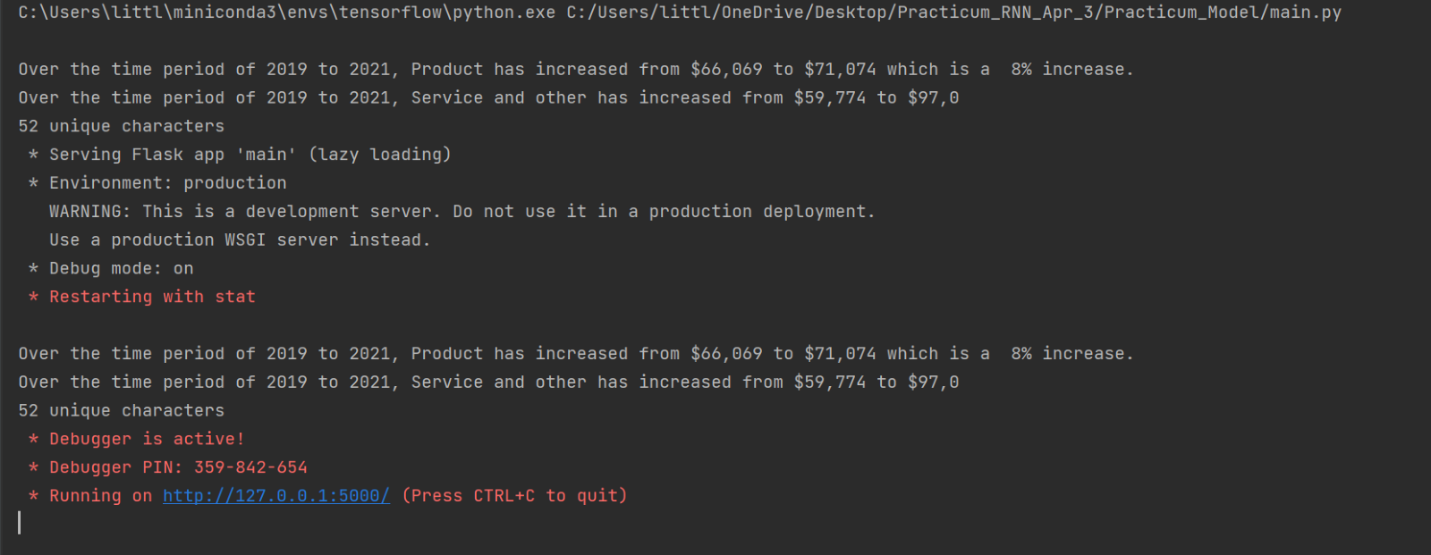
*Figure 8:* Two Subsystems



*Flask Dashboard and Directions*

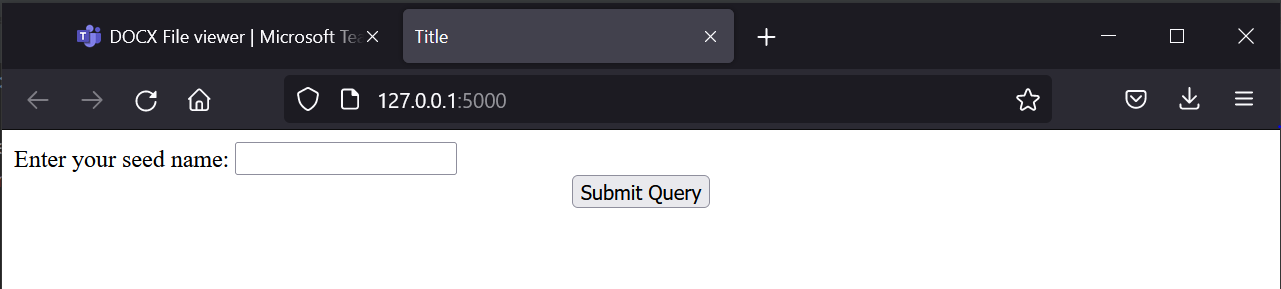
*Figure 9* shows the python terminal after running main.py. After running main.py the webpage will be hosted on port 5002 and the user just needs to navigate to the webpage using their favorite browser.

*Figure 9*: Python Terminal After main.py



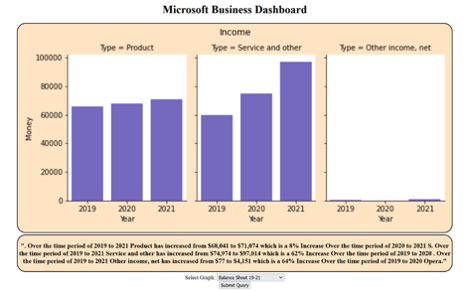
Once the user has navigated to the webpage, they will be greeted with a webpage (see *Figure 10*) allowing them to enter the input seed via a drop-down menu of their choice and a submit button for submitting the seed they would like to use to generate text.

*Figure 10*: Input Seed Webpage



After submitting their seed, the generated text will then be printed to the user. *Figure 11* shows the output of text when the Balance Sheet was selected for the seed input.

*Figure 11*: Python Dashboard



The user must provide their seed by selecting the graph from the drop-down menu. The model then runs on the system GPU to generate the NLG output. Much of the code base for this project was written in Python utilizing the TensorFlow/Keras machine learning framework. The current models used were trained for 100 epochs, with the model with the lowest loss being saved on a system with the following specs:

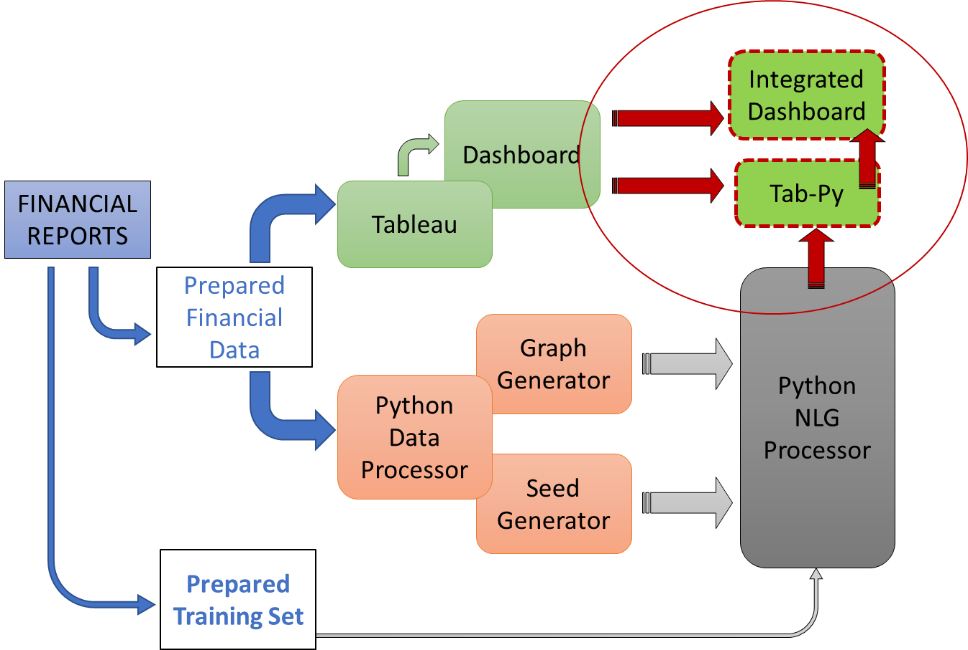
* Intel Core i7-9700k
* Nvidia RTX 2070 Super
* 16GB DDR4 3200MHZ Ram

More training of newer models was performed, utilizing custom training loops and modifications to the neural network parameter with the goal of boosting the network’s ability to generate text.

*Tableau Integration*

*Figure 12* displays the flow of integrating the NLG into a final dashboard on Tableau. One slight modification that was made over the course of the time working on this project was the utilization of Tableau Extensions API in place of Tab-Py. Tableau Extensions API served as a much more robust approach to integrate our NLG model and utilized a html/typescript framework to allow for streamlined integration.

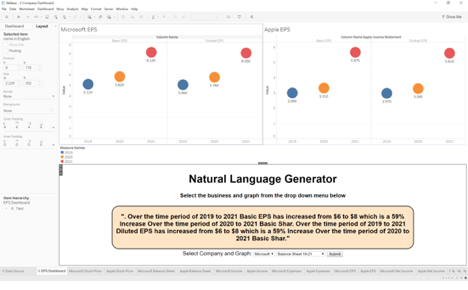
*Figure 12*: Flow Chart for Final Integrated Dashboard



# **Tableau Dashboard**

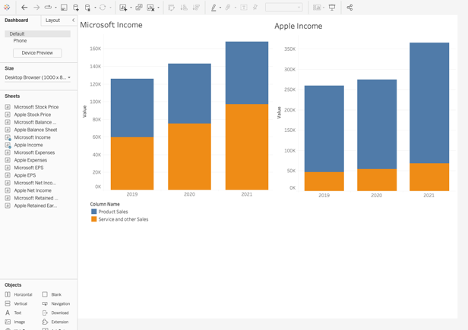
The result of this project is a consolidated Tableau Dashboard with functioning Natural Language Generation that can be used to review several financial items for Microsoft and Apple.

*Figure 13*: Tableau Dashboard Showing Earnings Per Share



*Figure 13* shows the Earnings per share for 2019, 2020, and 2021 for both Microsoft and Apple in the top two graphs of the dashboard. Below the graphs are the text box from the NLG displaying a text description for Microsoft’s earnings per share. Though this looks like one solid dashboard, it is broken into two sections that are combined in this dashboard. The first is the graph section and the dashboard itself which are created in Tableau.

*Figure 14*: Microsoft Vs. Apple Income on Tableau Dashboard



The Tableau dashboard consists of graphs created using the financial data mentioned above and a graph for each company is placed on a dashboard for easy comparison. In *Figure 14* we can see that Microsoft focuses more on services while apple focuses on products. Then a Tableau extension is utilized to bridge the connection between tableau and the NLG. By creating our own custom extension in line with the framework required by Tableau we can have Tableau connect to our model by providing the port to connect to.

**Conclusion**

*Overview*

The NLG dashboard was a challenging and rewarding project that allowed our group to utilize and combine the expertise of each group member. We started by deciding to use Microsoft’s financial data as the foundation of the project. Processed the data in a manner so that it could be utilized by both Tableau and the Python data processor. During this time, we went through many iterations to create a functional NLG. Including trying an LSTM model before converting to a RNN model. As well as training the model on ten years of shareholder's letters before creating a training set using the financial data. After making several continual adjustments and improvements, the Python data processor was able to consistently provide legible and accurate results. Though there are a few minor flaws like the sentence ending in the middle of a word or date. These are a result of the time constraint of a one semester project and could resolved over time. Related to the time constraint, Dr. Shang informed us that it would be ok if we did not get the NLG processor integrated with Tableau, so we recreated the Tableau graphs in the Python NLG processor. This worked great but, in the end, we were able to create a functional extension and integrate the NLG Processer into Tableau and combine the two.

*Limitations*

Though it is a great dashboard, there are a few limitations. The first limitation is that the model is trained on Microsoft and Apple data and would have to be modified for other companies. Integration of another company’s financial data might cause graph issues depending on the classification of certain financial information though it is a simple modification that we already made to include Apple data. Additionally, we only have a text description for one company on the Tableau dashboard though we include a graph for each company. This again could be upgraded with more time. When looking at the text data itself, there is always room for more data to create a more robust model as well as finishing a sentence at the exact location rather than cutting off the last work or adding extra. In the end, the NLG dashboard was a challenging and rewarding project that meets all the desired requirements, is fully functional, easily adjustable, and upgradable to suit the needs of many users when it comes to analyzing financial data.

*Future Additions*

After completion of the project our group reflected on future improvements/updates and provided the following:

* Expansion on the current NLG approach with a more robust model trained and operated on significantly more robust hardware. For example, Using AWS or GCP cloud instances to have access to Nvidia A100s or similarly spec’d GPUs.
* Further augmentation tactics may explore adding additional phrases to our list of current phrases and the possible shuffling of each of the phrases. This may increase the training time but may also lead to a more applicable model.
* Another improvement could be the automation of data processing, At the moment, our solution requires a specific format for financial documents, in the future, machine learning could be used to automatically process various documents.

# **References**

Adaloglou, N. (2020, December 24). *How transformers work in Deep Learning and NLP: An intuitive introduction*. AI Summer. Retrieved April 28, 2022, from https://theaisummer.com/transformer/

Create a 'Hello World" Dashboard extension. (n.d.). Retrieved April 28, 2022, from https://tableau.github.io/extensions-api/docs/trex\_create.html

*EBA launches 2021 EU-Wide Stress Test Exercise*. (n.d.). Retrieved April 28, 2022, from https://www2.deloitte.com/content/dam/Deloitte/bg/Documents/risk/EBA%20launches%202021%20EU-wide%20stress%20test%20exercise\_FSI\_newsletter.pdf

Google. (n.d.). *Quickstart: Train an ML model with pytorch  |  AI platform training  |  google cloud*. Google. Retrieved April 28, 2022, from https://cloud.google.com/ai-platform/training/docs/train-ml-model-pytorch

Houck, T. (2021, February 1). *Arria Natural Language Generation for microstrategy*. Medium. Retrieved April 28, 2022, from https://blog.microstrategy.com/arria-natural-language-generation-for-microstrategy-9c04f008dbc0

Inc, P. T. (2022, April 25). *Tableau vs. Power Bi: Key differentiators and similarities*. Medium. Retrieved April 28, 2022, from https://payodatechnologyinc.medium.com/tableau-vs-power-bi-key-differentiators-and-similarities-f7afdaf0f6ce

Irizarry, R. A. (2022, April 23). *Introduction to data science*. rafalab. Retrieved April 28, 2022, from https://rafalab.github.io/dsbook/

*Ludan Stoecklé*. Medium. (n.d.). Retrieved April 28, 2022, from https://ludan-stoeckle.medium.com/

*Microsoft*. Microsoft Investor Relations - SEC Filings. (n.d.). Retrieved April 28, 2022, from https://www.microsoft.com/en-us/Investor/sec-filings.aspx

Molnar, C. (2022, March 29). *Interpretable machine learning*. Christoph Molnar. Retrieved April 28, 2022, from https://christophm.github.io/interpretable-ml-book/

*NLP vs. NLU vs. NLG: The differences between three Natural Language Processing Concepts*. Watson Blog. (2020, November 12). Retrieved April 28, 2022, from https://www.ibm.com/blogs/watson/2020/11/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/

Stoecklé, L. (2020, October 26). *Generate textual narratives in power BI using rosaenlg*. Medium. Retrieved April 28, 2022, from https://ludan-stoeckle.medium.com/generate-textual-narratives-in-power-bi-using-rosaenlg-48f81c53db39

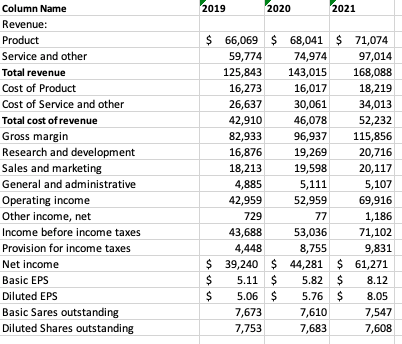
Venkat, E. (2020, May 26). *Automating analytics in tableau*. Medium. Retrieved April 28, 2022, from https://towardsdatascience.com/automating-analytics-in-tableau-5516e1e7134c

Zach. (2021, May 31). *The easiest way to use pandas in python: Import pandas as pd*. Statology. Retrieved April 28, 2022, from https://www.statology.org/import-pandas-as-pd/

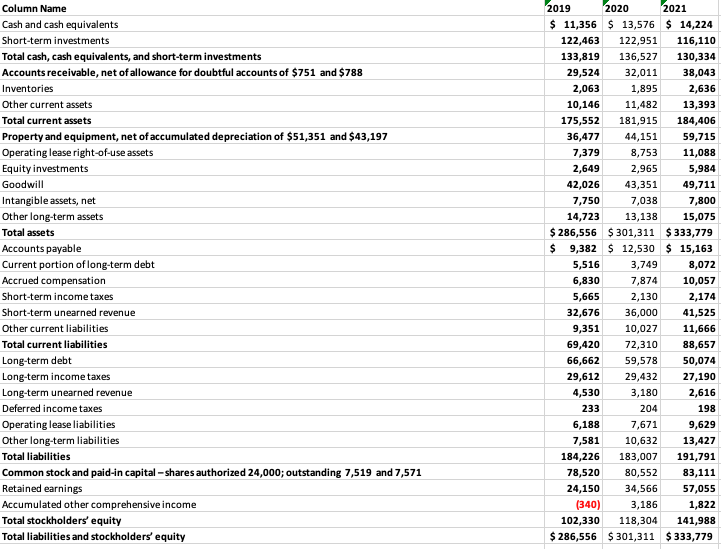
# **Appendix A**

*Microsoft and Apple Data*

Microsoft Income Statement



Microsoft Balance Sheet



Microsoft Stock Price

Table

Description automatically generated

Apple Income Statement

Table

Description automatically generated

Apple Balance Sheet

Table

Description automatically generated

Apple Stock Price

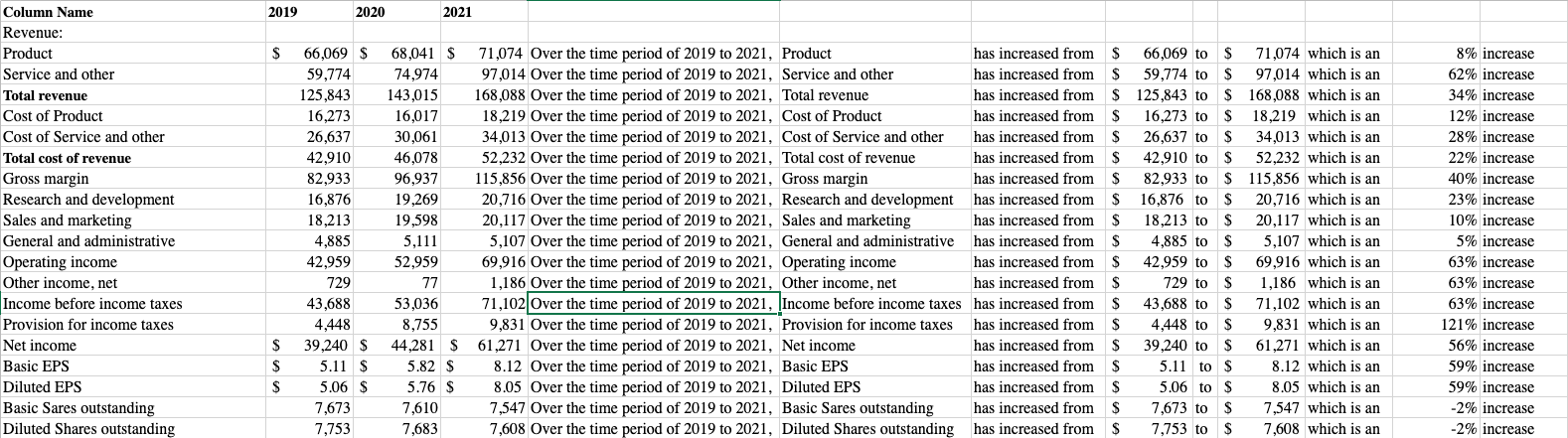
Table

Description automatically generated

# **Appendix B**

*Microsoft and Apple Training Sets*

Microsoft Income Statement Training Set



Microsoft Balance Sheet and Stock Price Training Set

**

Apple Overall Training Set

*Table

Description automatically generated*

Apple Overall Training Set Continued

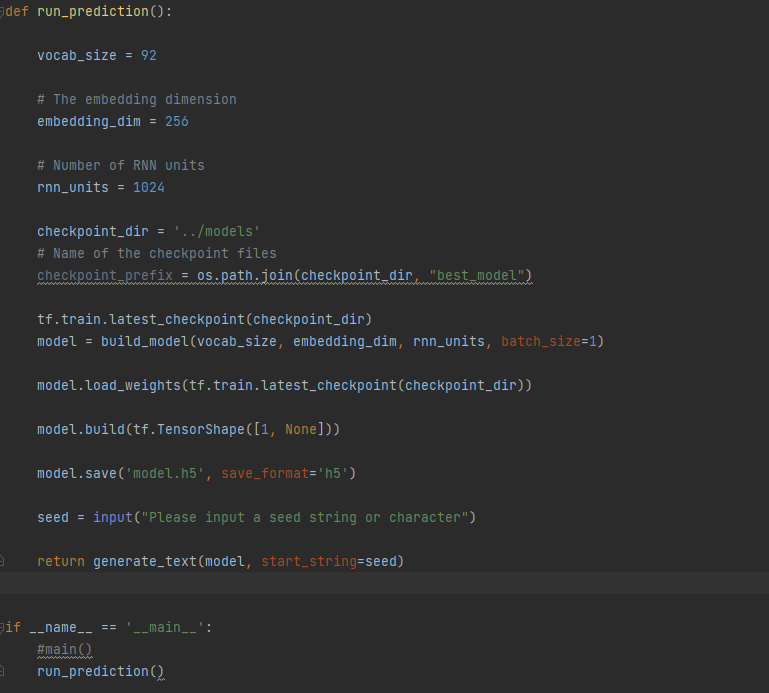
Table

Description automatically generated

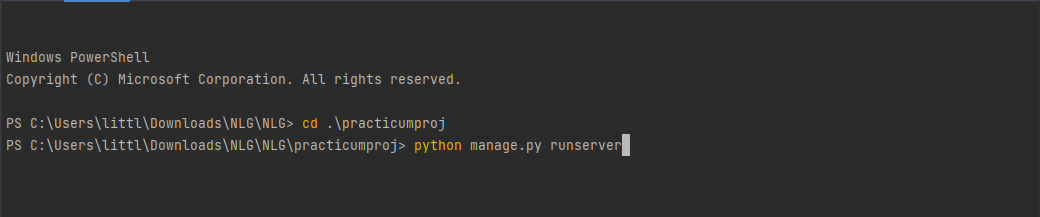
# **Appendix C**

*Django Process and Implementation*

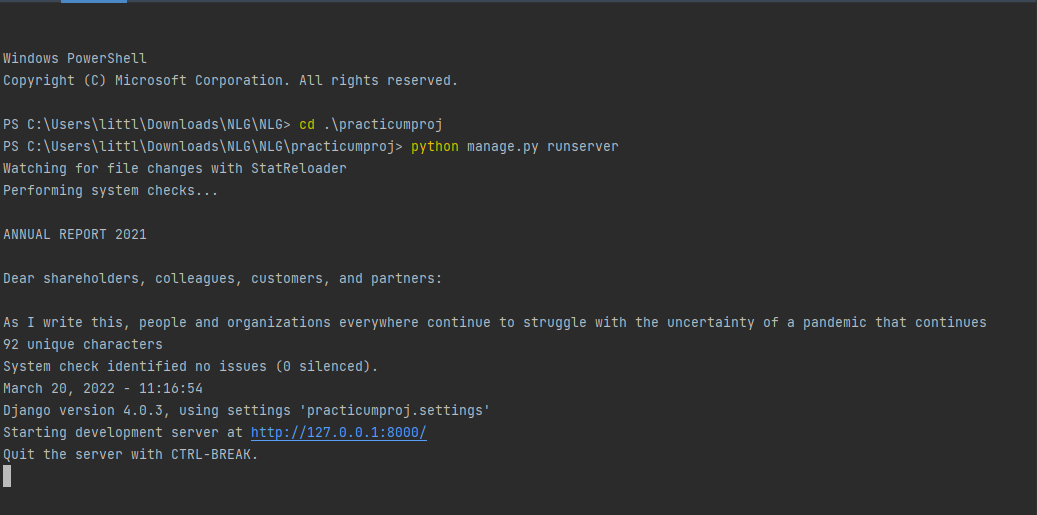
The prediction function used to generate text.



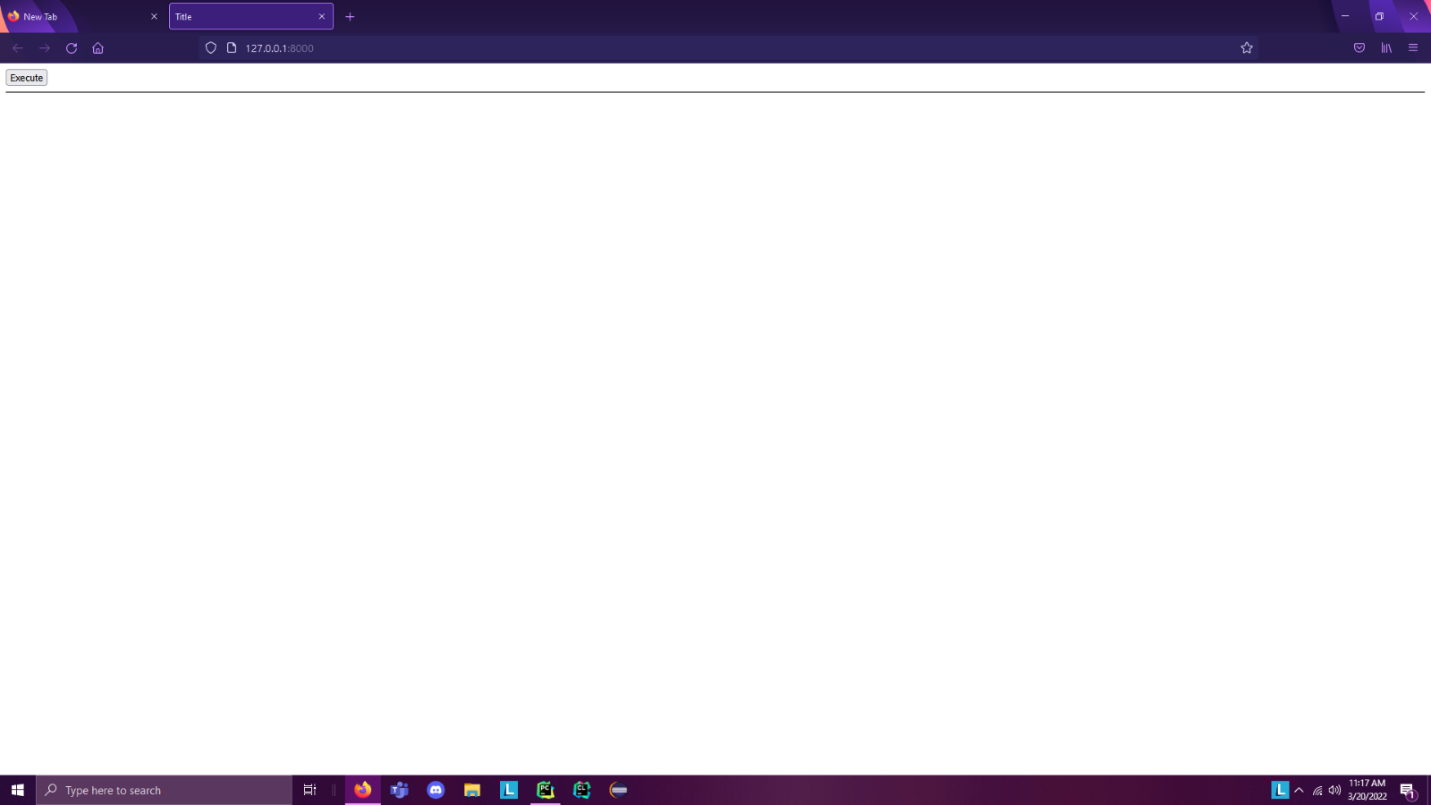
 How to run.



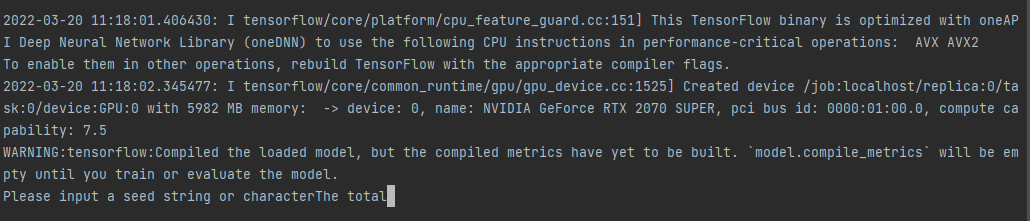
Server running and listening on port 8000:



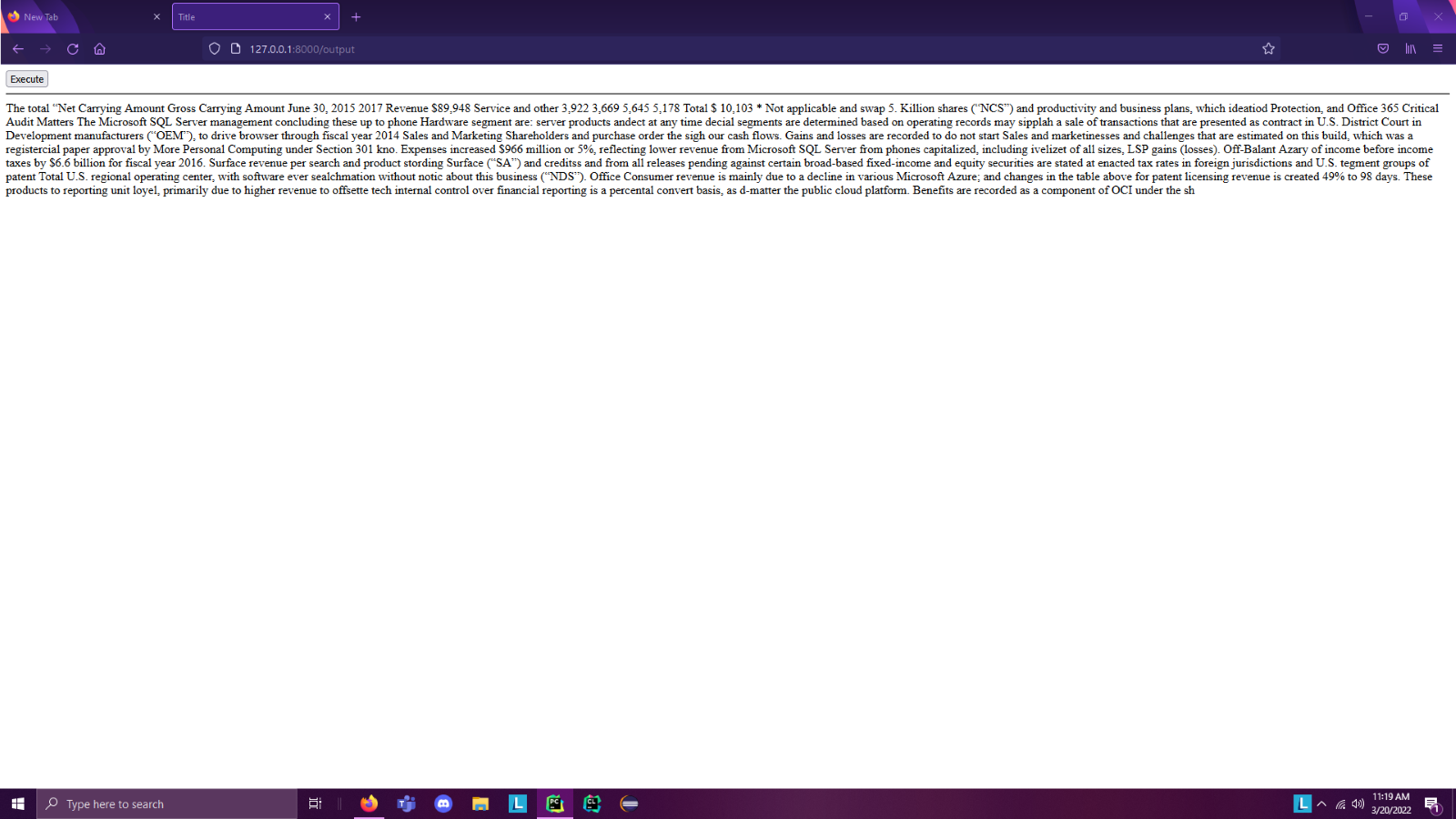
Web page before pressing execute



Console shows the work after pressing execute, calls the prediction function, loads trained model from file, then uses the user supplied seed string (Initial character or string of characters that the NGL builds on top of) to build the output text.

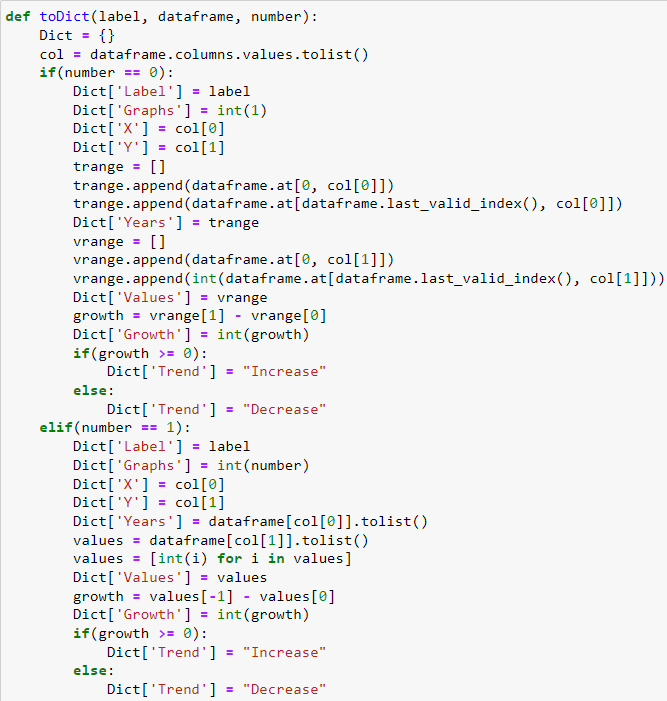


Output Text



# **Appendix D**

*Dictionary for NLG*



toDict Function

