**Sentiment Analysis of Product Reviews Using TensorFlow**

By **Aadhitya A** (2019506001), **Balasubramanian KN** (2019506017), **Meyyappan S** (2019506050)

**Sentimental Analysis:**

**Sentiment analysis** is the process of detecting positive or negative sentiment in text. It’s often used by businesses to detect sentiment in social data, gauge brand reputation, and understand customers.

Sentiment analysis in NLP (Natural Language Processing) is about deciphering such sentiment from text. Is it positive, negative, both, or neither? If there is sentiment, which objects in the text the sentiment is referring to and the actual sentiment phrase such as poor, blurry, inexpensive. This is also called aspect-based analysis. But for our project, we considered two sentiments alone.

As a technique, sentiment analysis is both **interesting** and **useful**.

First, to the interesting part. It’s not always easy to tell, at least not for a computer algorithm, whether a text’s sentiment is positive, negative, both, or neither. The cues can be subtle. Overall sentiment aside, it’s even harder to tell which objects in the text are the subject of which sentiment, especially when both positive and negative sentiments are involved.

Next, to the useful part. This is easy to explain. People who sell things want to know how people feel about these things. It is called customer feedback. Ignoring it is bad for business. So, it’s mandatory.

**Uses of Sentimental Analysis:**

Sentiment analysis is useful for quickly gaining insights using large volumes of text data. In addition to the customer feedback analysis use case here are another two exemplary use cases:

1. Stock trading companies who trawl the internet for news. Here, sentiment algorithms can detect particular companies who show a positive sentiment in news articles. This can mean a significant financial opportunity, as this may trigger people to buy more of the company’s stock. Having access to this type of data allows traders to make decisions before the market reacts.

2. How the stock price of a company can be affected by news?. The sentiment expressed in the news of acquisition triggers a stock trading algorithm to buy the stock before the increase in price happens.

Another application of sentiment analysis is measuring social media posts. During the announcement of Brexit, a social media sentiment tool predicted that “remain” polls were incorrect, as much as six hours before the news broke.

**Requirements Needed for Sentimental Analysis:**

***Libraries Used:***

1. Numpy
2. Pandas
3. tensorflow
4. Keras
5. Scikit-learn
6. tensorflow\_text
7. tensorflow\_hub
8. tqdm

***Dataset Needed*:**

Generally, a review dataset (it can be feedback, movie reviews) is preferred for this project.

***Type of model:***

Here, we done the model from scratch using Keras for Neural Network and Pandas for text mining. Alternatives were considered like **BERT**, but it took a lot of time for training even on TPUs.

**How the model is implemented:**

There are multiple factors considered while developing a model for this project.

* Type of model we’ve to implement
* Dataset considered
* Libraries needed (like do we need this for implementation)
* A Text Encoder to convert text into Tensors
* Type of Encoding needed for Train and Test data
* Does it need regularization or not? If yes, what kind?
* Number of layers
* Do we need to deploy via Webpage or leave it as a model?

We tried to answer the following by doing and we luckily got some of the answers!

After training the model, we were in dilemma on how to deploy the project. Many questions were raised, do we need to use **Flask**for deployment via HDF5 or Docker etc. For now, we used Docker

**Where the model is trained? (In which software)**

The model is actually trained using a data science community website called “**Kaggle**” **(it was acquired by Google in 2017)** It has many features like Jupyter Notebook, free GPUs and TPUs, sharing of datasets and much more! Also, we found that it’s much faster than **Google Colab**, and dataset can be directly used there without downloading it.

**Obstacles faced:**

We faced common problems like Overfitting, high loss and other factors. For me, it took almost a day to figure out on bringing high accuracy; first it was around 85%. Then after implementing another hidden layer with **L1 kernel regularizer, the accuracy became 93%-96%** which is pretty high for a model. Then we planned whether to deploy the model on some platform, but it seemed a pretty late and so we decided to go via **Docker. Flask** seemed a good one, but we aren’t trained to do yet!

**What is Docker?**

**Docker** is a set of [platform as a service](https://en.wikipedia.org/wiki/Platform_as_a_service) (PaaS) products that use [OS-level virtualization](https://en.wikipedia.org/wiki/OS-level_virtualization) to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, [libraries](https://en.wikipedia.org/wiki/Library_(computing)) and configuration files; they can communicate with each other through well-defined channels. Because all of the containers share the services of a single [operating system kernel](https://en.wikipedia.org/wiki/Kernel_(operating_system)), they use fewer resources than [virtual machines](https://en.wikipedia.org/wiki/Virtual_machine).

In a nutshell, Docker is a virtual machine but without the need of a Guest OS, so it operates on Kernel level which saves space and time.

**Is Docker useful?**

The answer is yes! Docker is indeed used as a PaaS which helps ML engineers, Data Scientists to store their model in cloud and run the service from start without the need of OS.

**Note:** Demonstration of deploying a model using Docker is considered as a risk one, because it consumes a lot of data and storage involved in creating a container image. Instead, we provided a Dockerfile in which users can run locally and see what’s going on!

**How can I see the notebook?!**

We included a ZIP containing necessary files for review. Plus, if you’re interested in running the model, here’s the Kaggle link (it’s made public) and repo link (at GitHub)

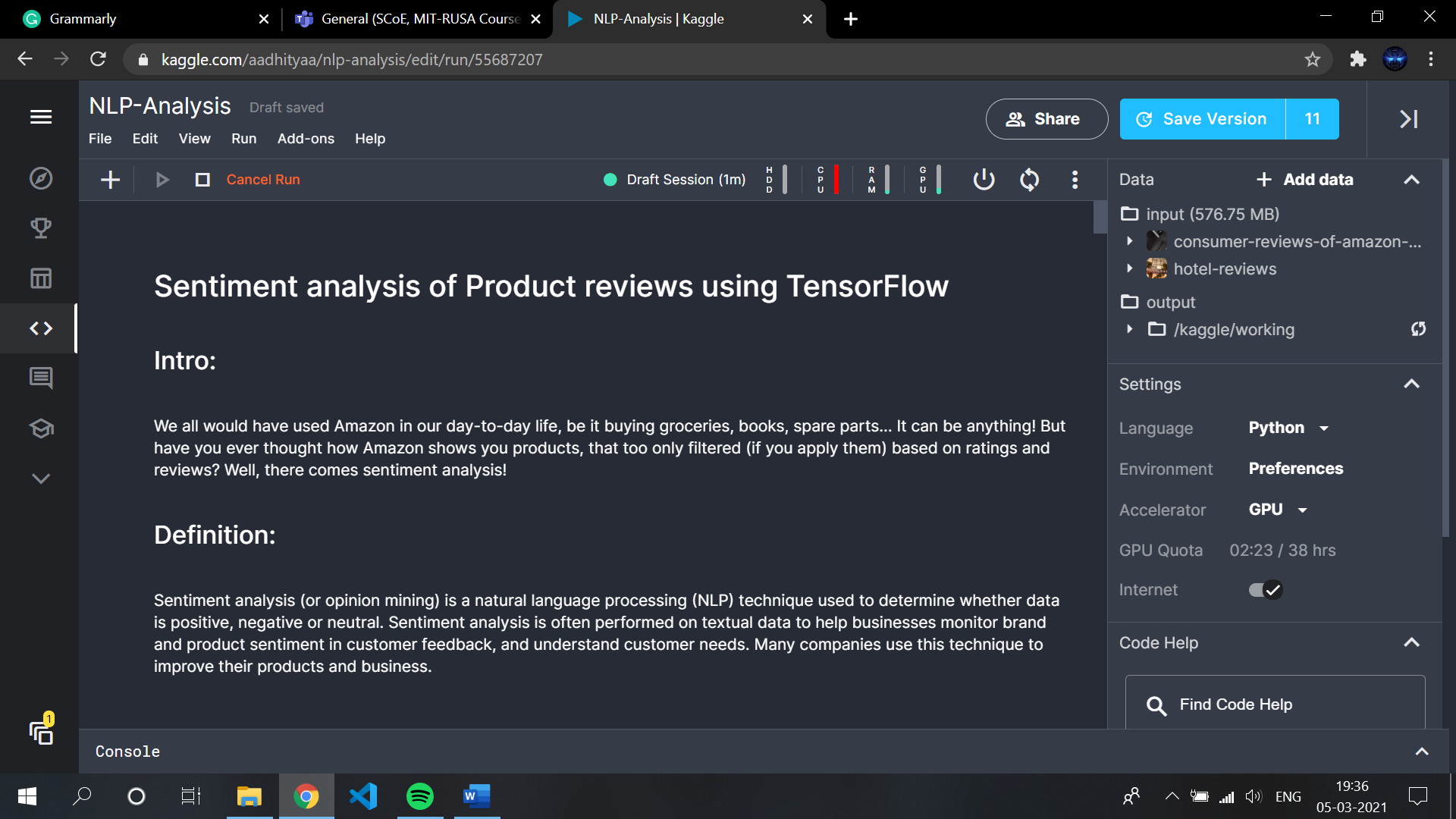
**Kaggle Notebook link**:

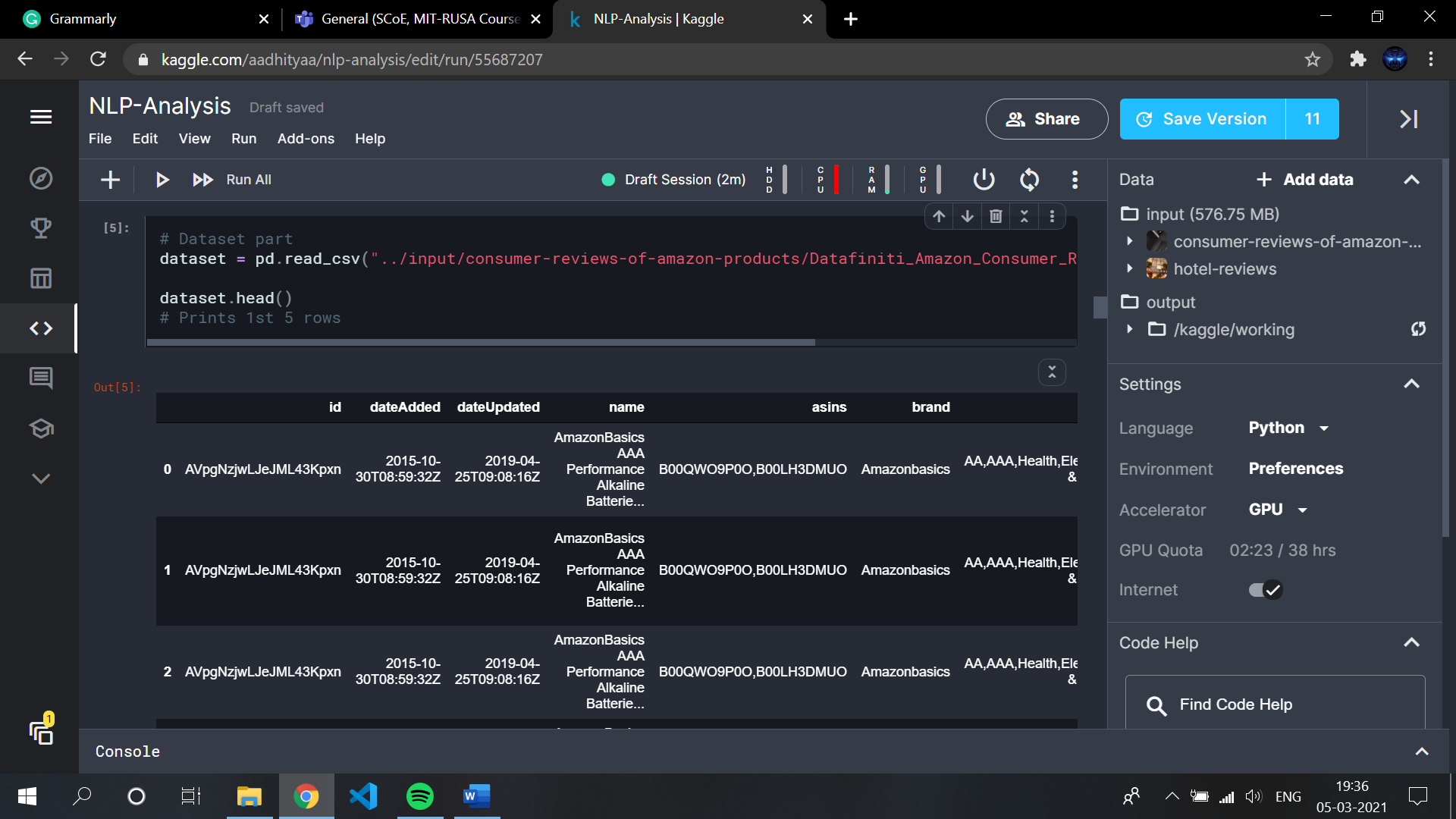
<https://www.kaggle.com/aadhityaa/nlp-analysis>

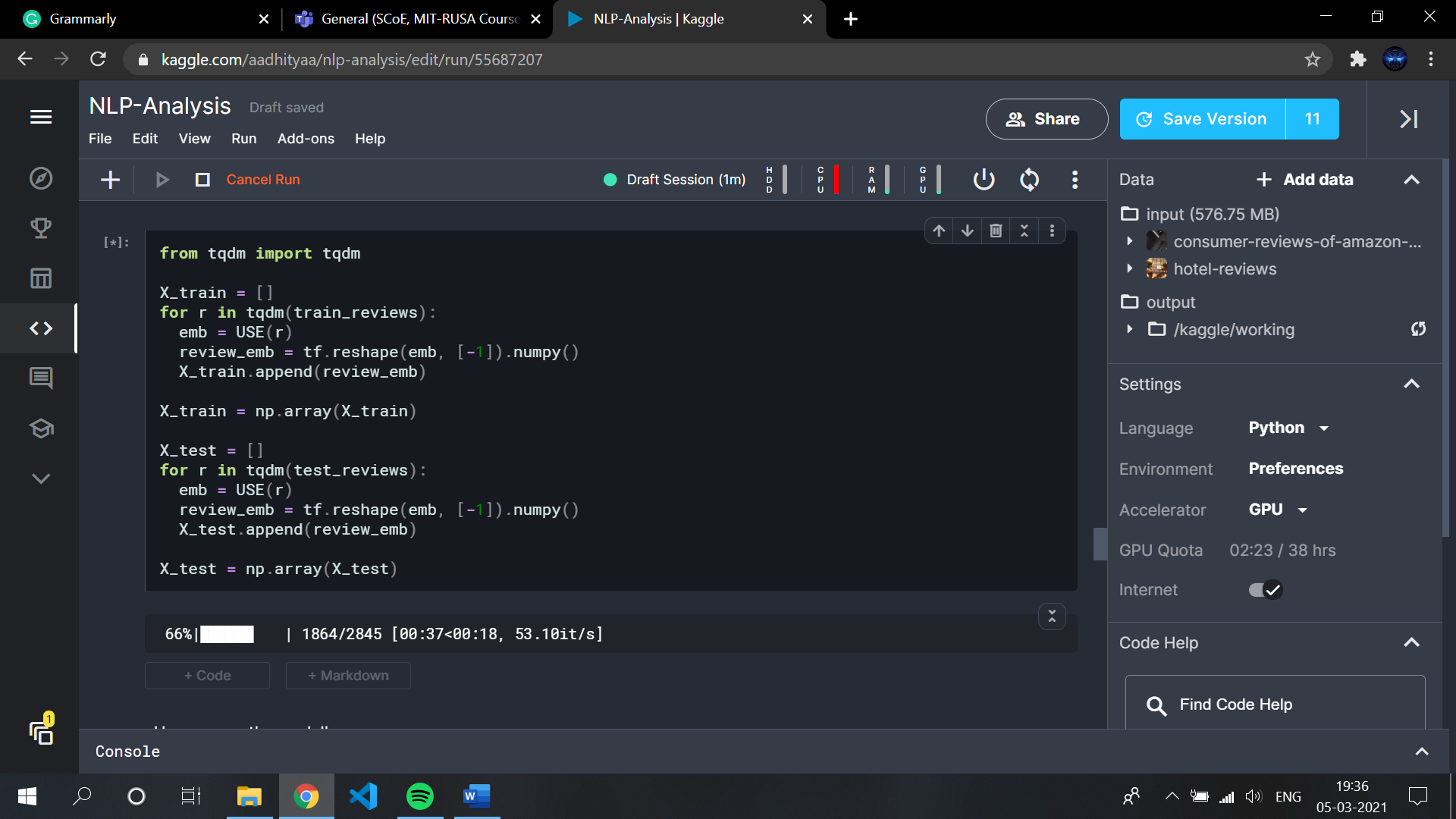
**GitHub Repo link:**

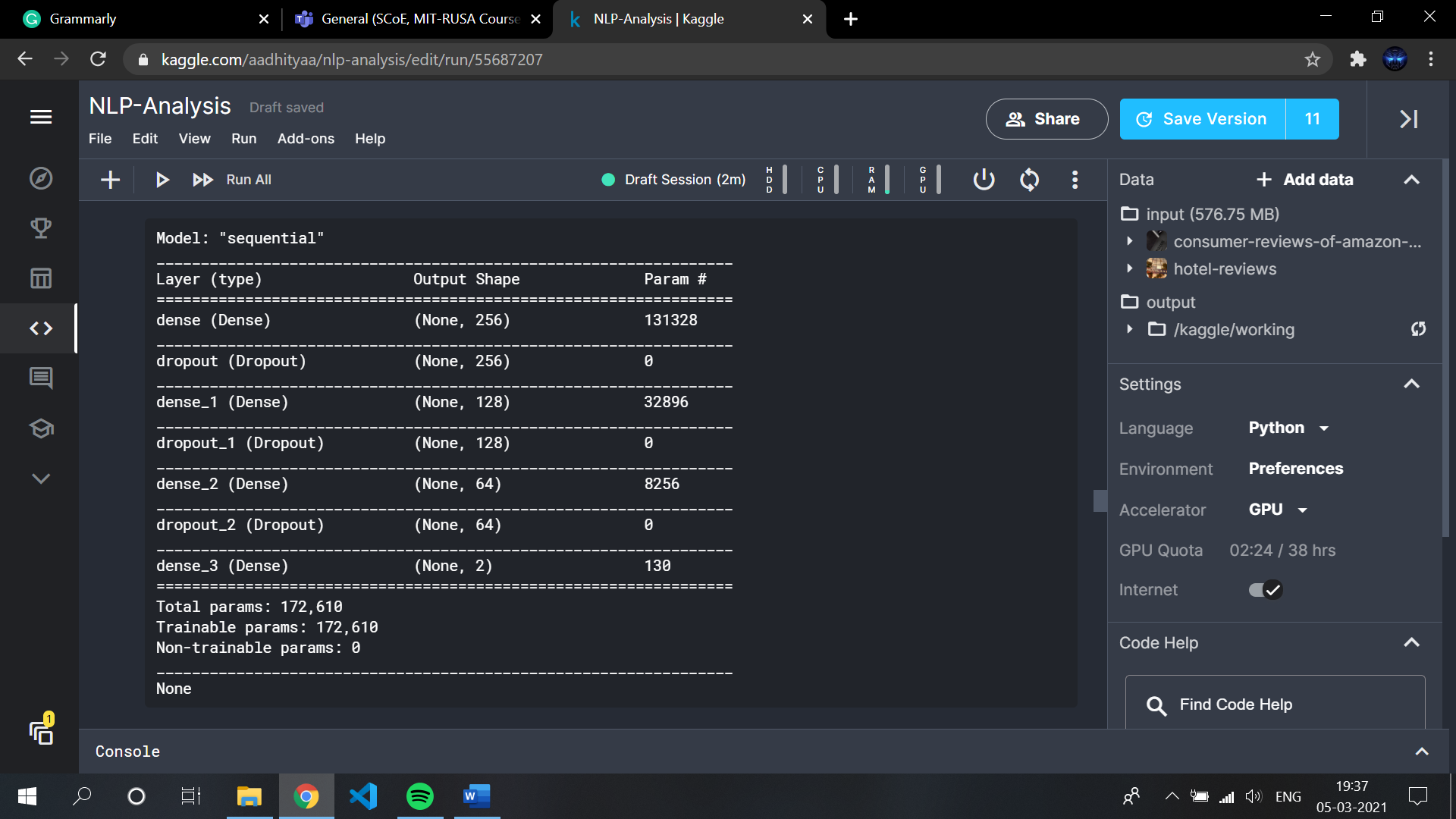
<https://github.com/alphaX86/nlp-demo>

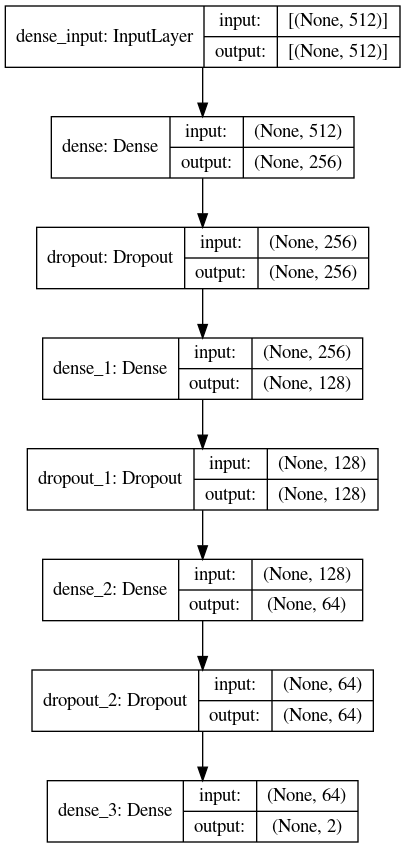
**Images/Screenshots (Code & Output) taken from Kaggle and VS Code:**

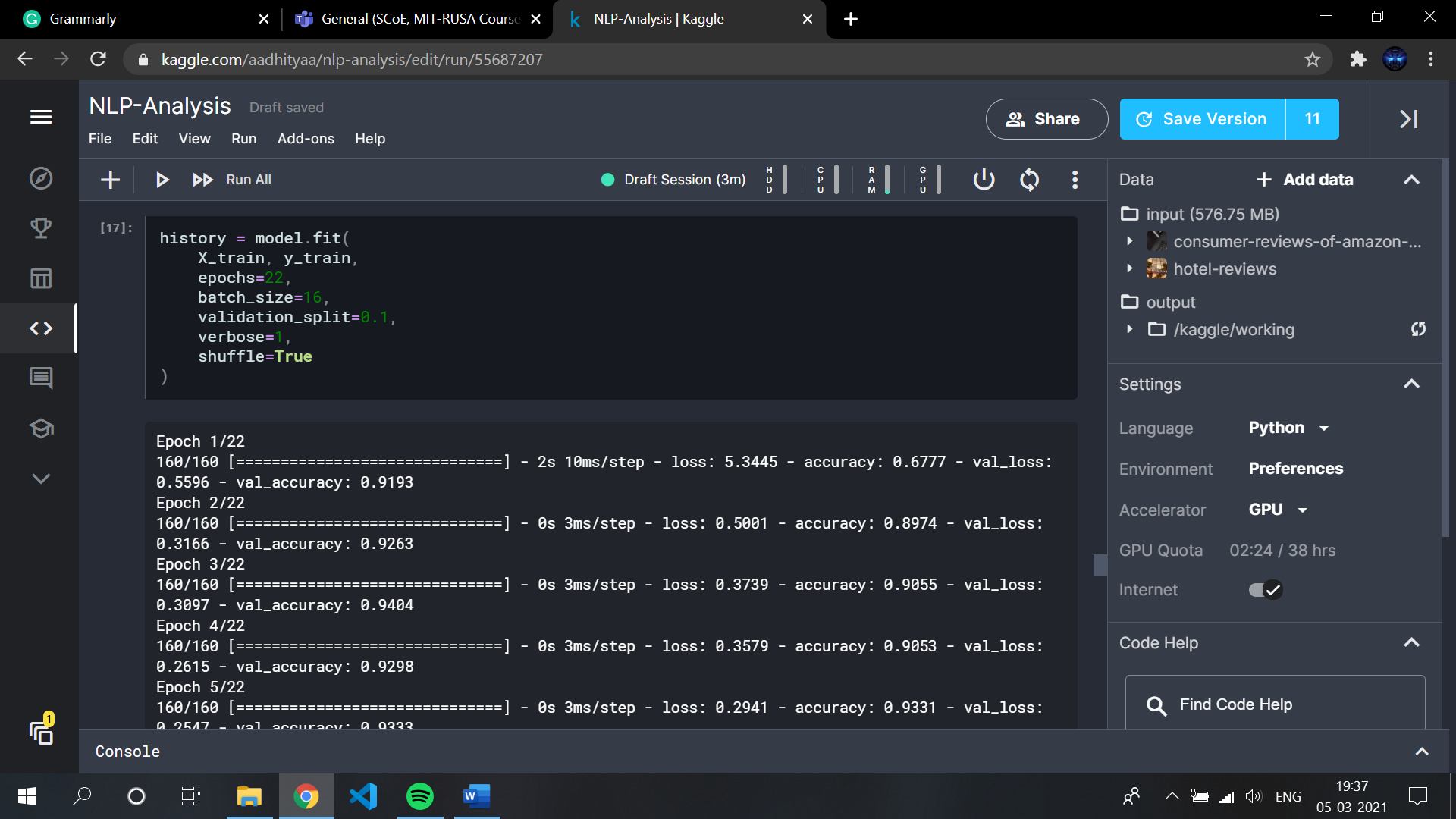


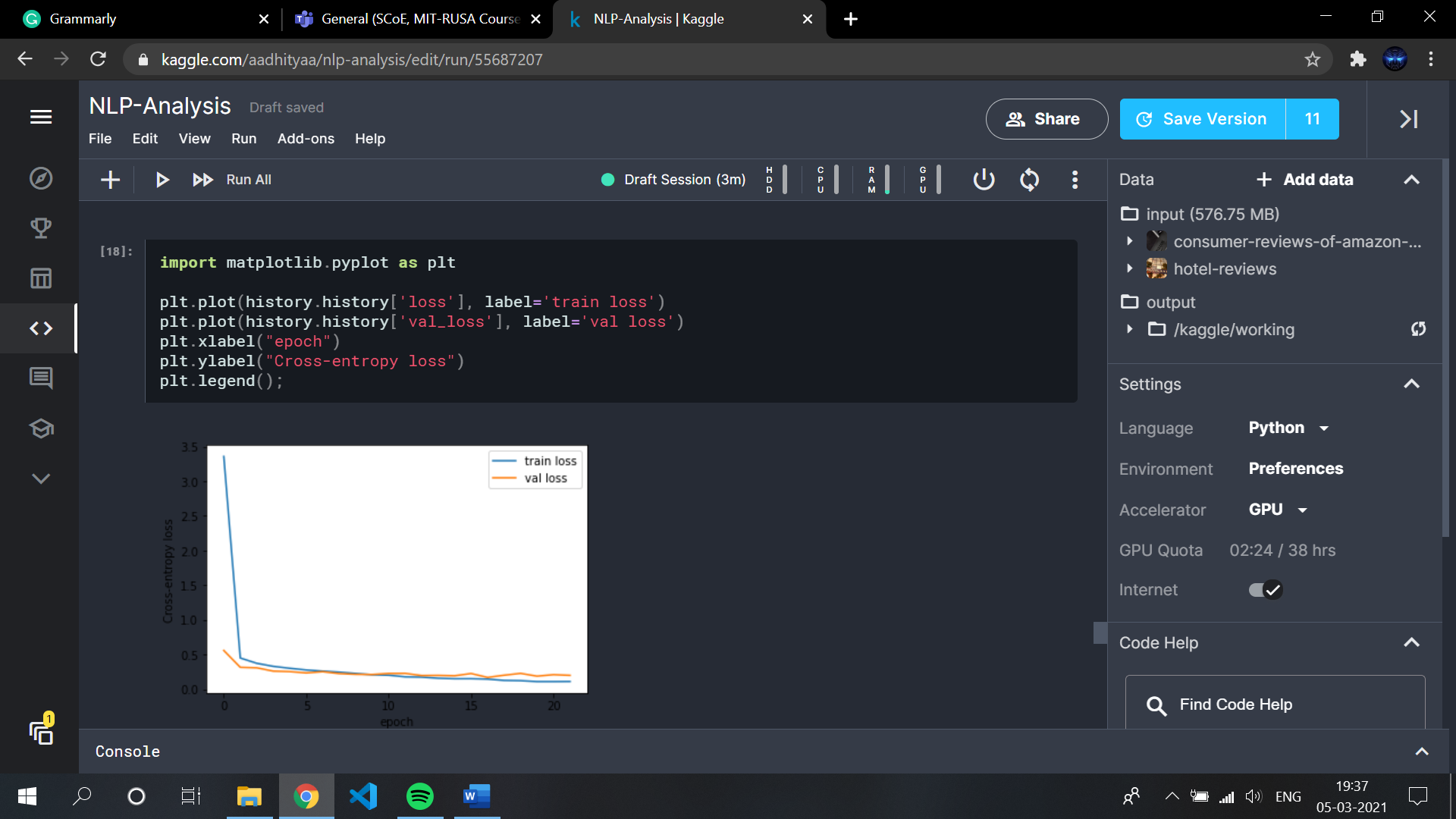


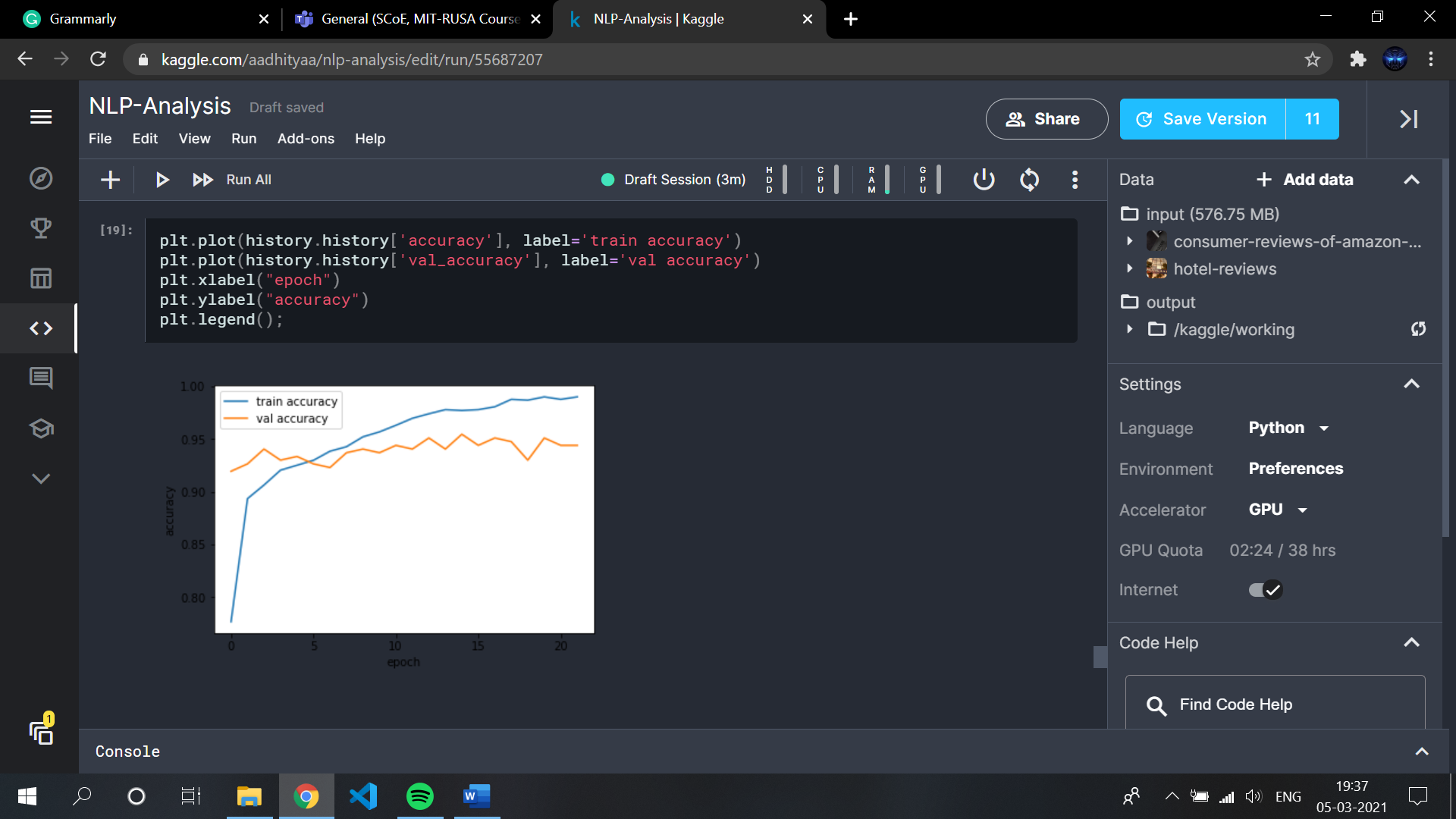


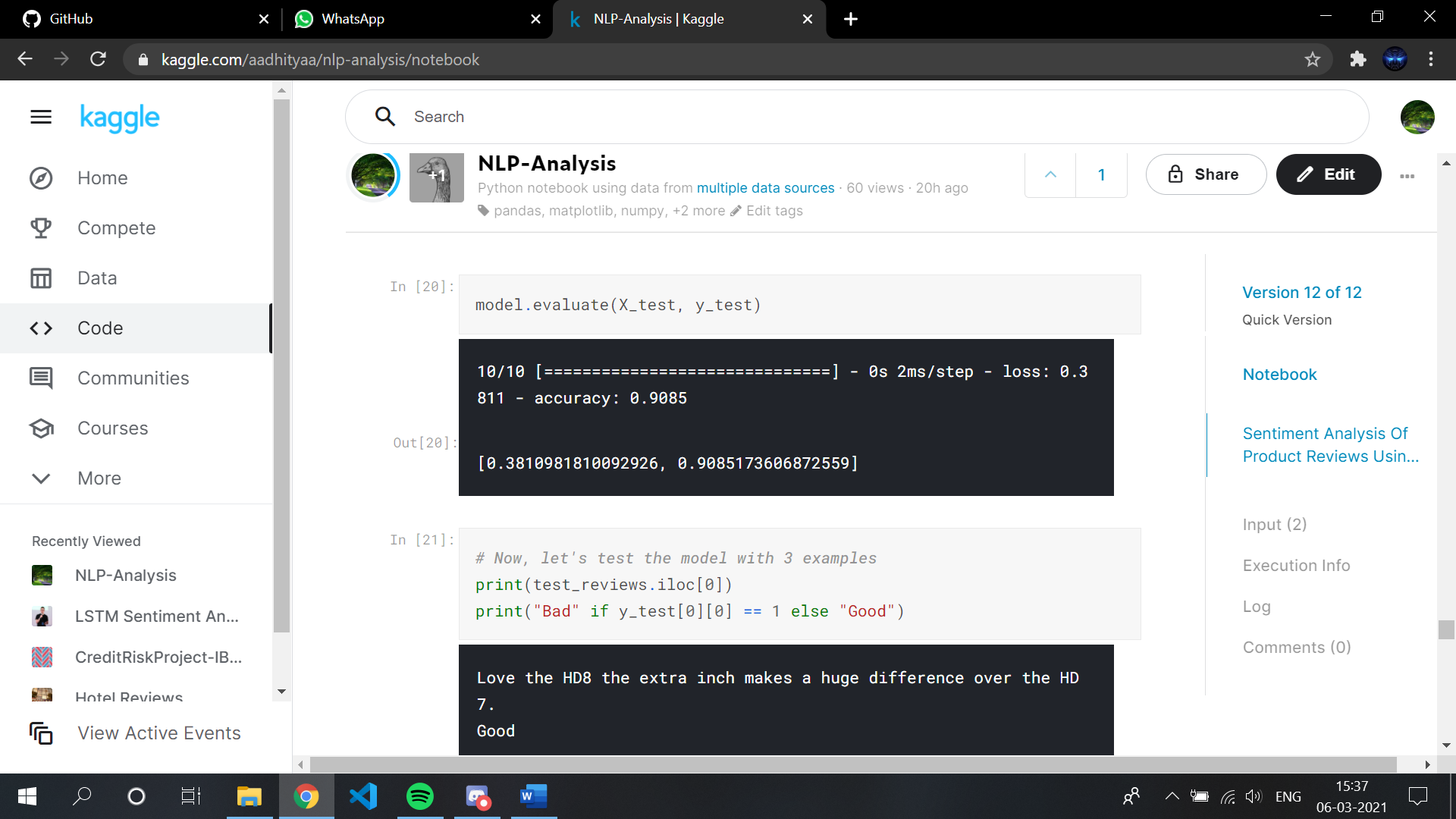


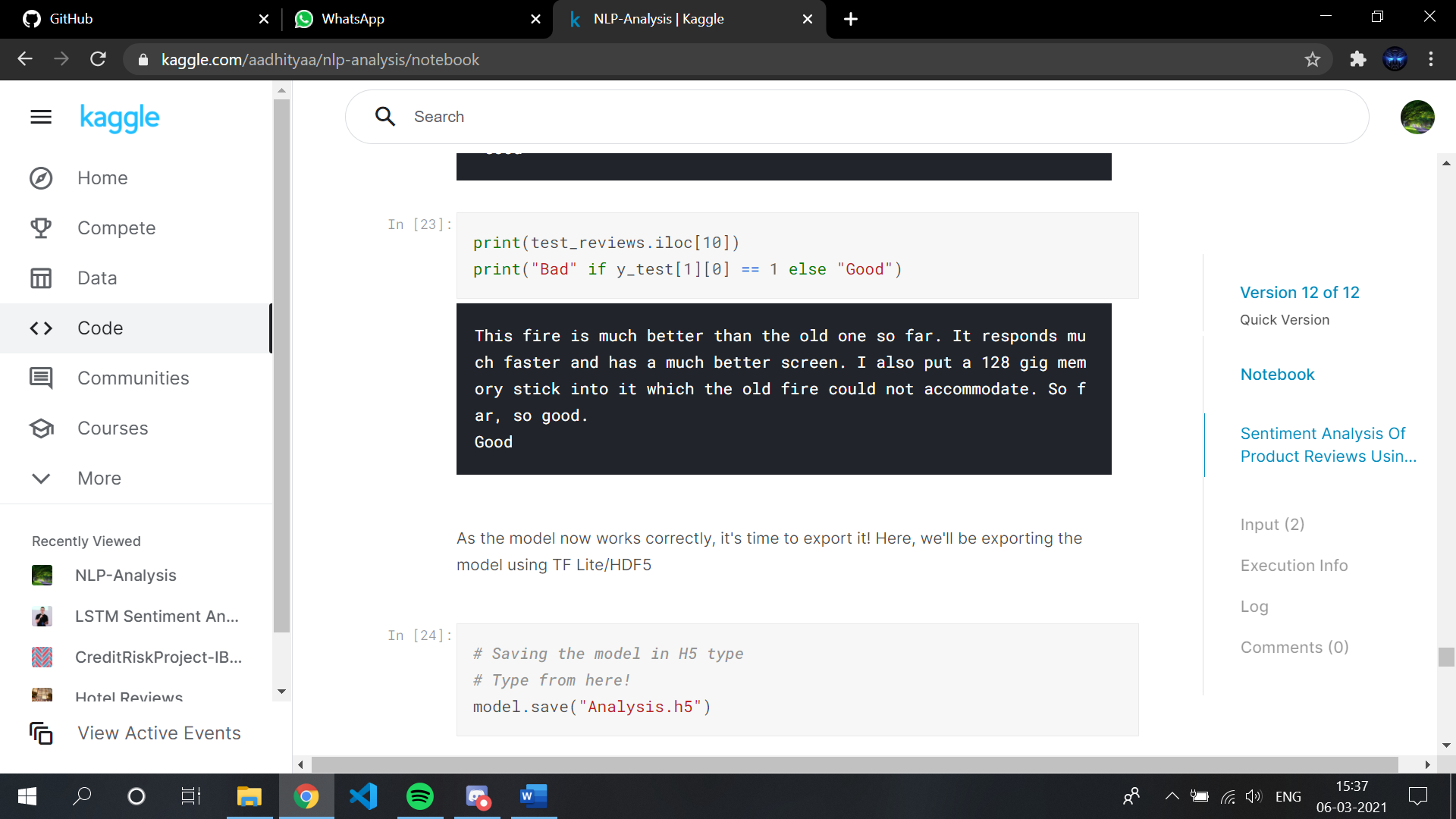


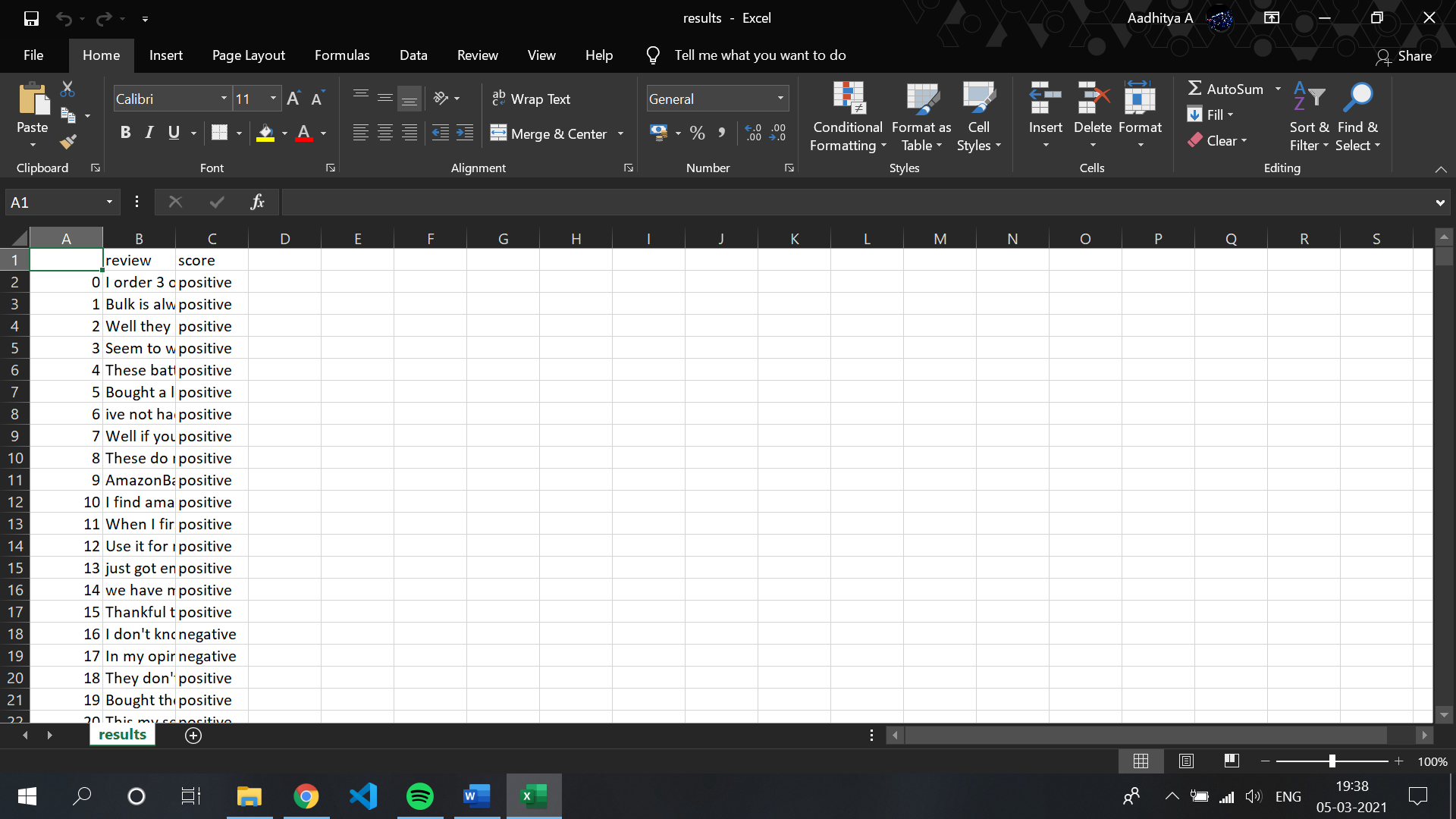


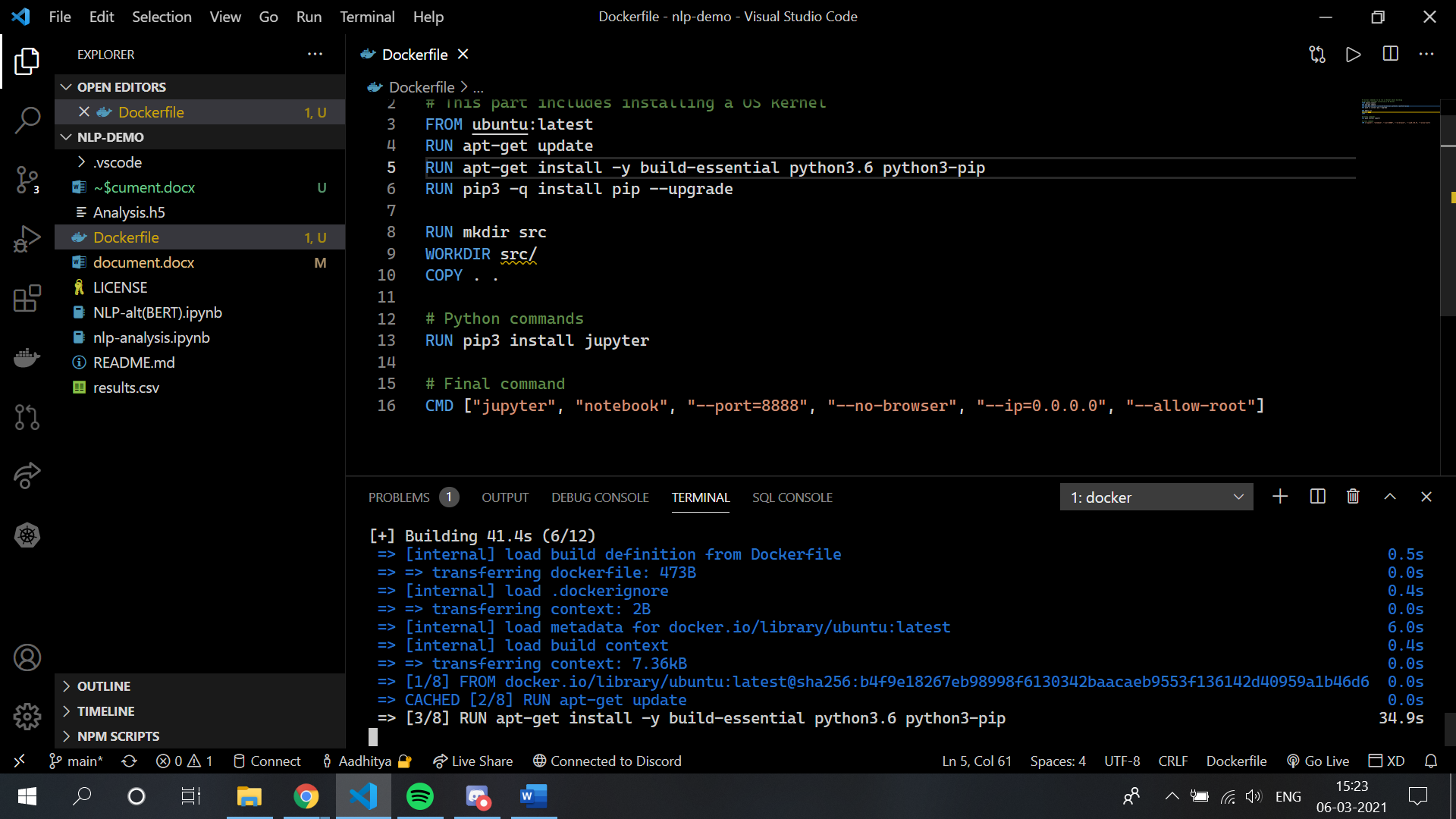












**Alternative models considered:**

***1.BERT Model:***

BERT stands for [**Bidirectional Encoder Representations from Transformers**](https://arxiv.org/pdf/1810.04805.pdf) and it is a state-of-the-art machine learning model used for NLP tasks. Jacob Devlin and his colleagues developed BERT at Google in 2018. Devlin and his colleagues trained the BERT on English Wikipedia (2,500M words) and BooksCorpus (800M words) and achieved the best accuracies for some of the NLP tasks in 2018. There are two pre-trained general BERT variations: The base model is a 12-layer, 768-hidden, 12-heads, 110M parameter neural network architecture, whereas the large model is a 24-layer, 1024-hidden, 16-heads, 340M parameter neural network architecture.

The model is already pre-trained and can be used directly. The only disadvantage is that training the dataset can take an hour for one epoch (**Reason:** Each part of the text is analysed and trained for better results). The same case goes with **OpenAI’s GPT-3 model.**

***2.Bi-LSTM (Recurrent Neural Network):***

***RNN***: RNNs are designed to make use of sequential data when the current step has some kind of relationship with the previous steps. This makes them ideal for applications with a time component (audio, time-series data) and natural language processing. RNN’s perform very well for applications where sequential information is important because the meaning could be misinterpreted or the grammar could be incorrect if sequential information is not used. Applications include image captioning, language modelling and machine translation.

***LSTM***: Long Short-Term Memory (LSTM) stores historical information by constructing a memory unit, each temporal state saves the previous input information, which can effectively alleviate the long-distance dependence problem of Recurrent Neural Networks (RNN).

***Bi-LSTM***: LSTM ignores future information. The Bi-LSTM contributes to the solution of obtaining both historical information and future information by using the bidirectional propagation mechanism, which helps to achieve better performance in such tasks.

This type of Neural Network is really efficient and can be used if anyone wants to train a model with only less effort (of course, effort is needed for accuracy, but compared to ours it’s efficient)