Athens University of Economics and Business

MSc Business Analytics

Machine Learning and Content Analytics

Streamlit Trading Assistant

Web-App

Team Bravo
Ilias Dimos (f2822102)
Charilaos Petrakogiannis (f2822112)
Ioannis Triantafyllakis (f2822115)
Nikolaos Tsekas (f2822116)

Athens,

September 2022

Table of Contents

Table of Figures

The Idea Behind Our Project

Although the existence of the stock market has been around for many decades, in recent years there has been an increase in the number of people involved in this particular sector. The reason behind this interesting phenomenon consists of two parts. On the one hand we have the entry of cryptocurrencies into the stock market, an event that shook the waters of the stock market, while on the other hand we have the evolution of technology and therefore of the tools that help traders make the appropriate decisions that will determine their investments.

Our team, influenced by the international trends related to the development of the aforementioned tools, decided to create an environment through which potential traders will be able to make the best possible decisions about the stocks they are interested in. More specifically, we have focused on combining the already existing knowledge around the neural network models of sentiment analysis and stock price prediction that exist on the world wide web, so as to create an innovative dashboard through which everyone will be able to derive important information about the future evolution of the price of each stock.

To create this particular dashboard, we had to combine two separate neural network models. The first of these two is a FinBERT model that was trained as best as possible to be able to perform sentiment analysis on article headlines related to the stock that the user will select each time. The result of this analysis is the classification of the respective articles as negative, positive, or neutral and, at the end, this ranking is visualized through a line plot which helps the user to receive this specific information easier and more understandable, comparing one by one the selected articles that refer to the stock in which they are interested.

On the other hand, the second part of the dashboard concerns an LSTM model which has been trained so that taking as input information (from yahoo finance) the course of each stock up to today, would be able to identify any periodic patterns of the Adjusted Closing price and based on them to predict the future stock performance for the next 7 days. This result is again visualized through a line plot which clearly separates the forecast values from those of the historical data.

Finally, our project is perfected by adding some mini tools (technical analysis using the Bollinger Bands and MACD methods) which, although they do not derive from any of the above neural network models, provide extra information (in the form of a table) to the user regarding the most appropriate decision that s/he can make on the respective stock. On the next chapter, we present the dashboard’s tools with more details.

An Introduction to our Dashboard

Our project is a simple Streamlit stock trading assistant web-app, which includes 3 tools that can help a trader have more accurate and profitable trades. You can try our app here: App Link in HugginFace Spaces. An overview of our dashboard – like web app is presented in Figure 1. For a more detailed view of the dashboard, you can use the Figure in the Appendix.

The assistant consists of the following 3 tools:

- Stock Headers' of marketwatch articles sentiment analysis with a FinBERT model
- 7-day stock price prediction with an LSTM model
- Outlooks of four different technical analysis methods for stock trading

When a user chooses a stock from the app's drop-down menu, each of these 3 tools will return their advice/outlook.

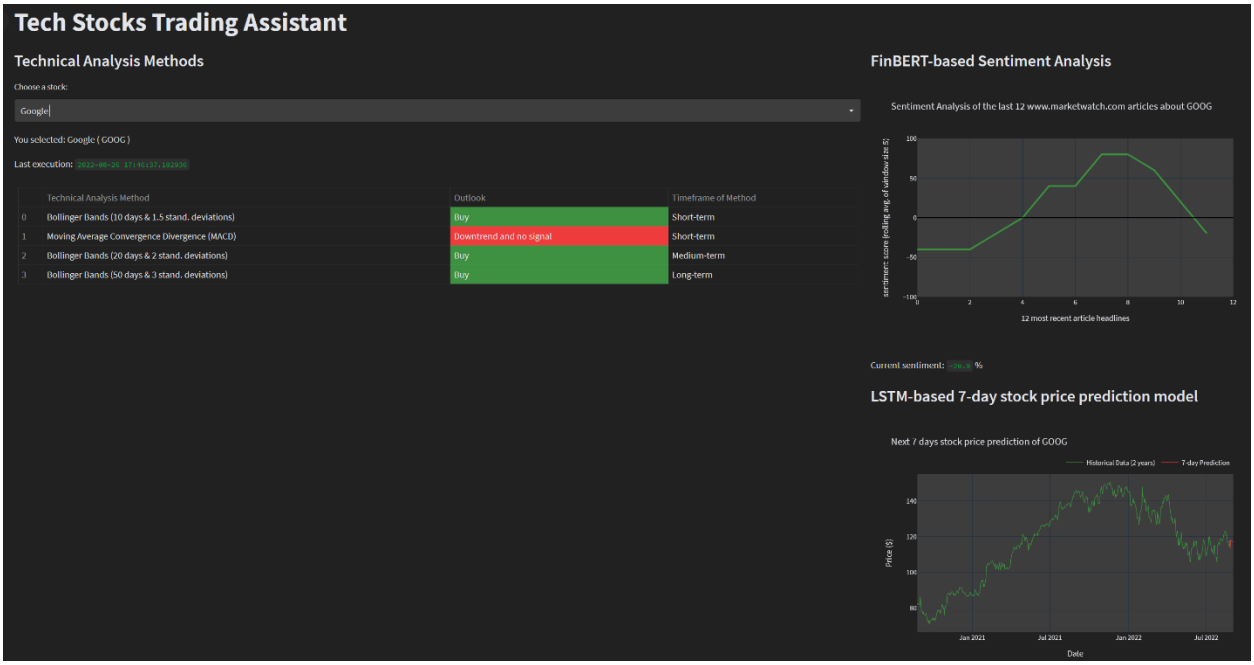


Figure 1: Streamlit Trading - Assistant Dashboard

FinBERT-based Sentiment Analysis (1st tool)

The first tool is the Sentiment Analysis line plot (upper-right side of dashboard) which is based on a FinBERT model. When the user chooses a stock, our code scraps the last 17 articles' headlines of this stock from marketwatch and labels them as either "Positive", "Negative", or "Neutral", which are then encoded as scores (+100, 0, -100 correspondingly). As there is no standard way to quantify the sentiment of something, we use a scale of (-100, +100). It must be noted that even though we get the last 17 articles' headers in each execution, only the last 12 headers' sentiment can be seen in the plot. That is because we use a rolling average with a sliding window of size 5, to "smooth" the line. When this line is mostly negative (below 0), we expect that most of the last articles have a negative sentiment, when they are mostly positive (above 0),

we expect that there are many articles with positive sentiment, while when the line is close to 0, or revolves around it with no specific trend, we expect mixed outlook articles.

`Interpretation for traders`

Many recent negative news may lead to stock price decline, which means that the stock now is cheaper than it was recently, and the explanations are the following:

- Either the bad news are minor and temporary, so the stock will soon recover (Often a 👍 Buy Signal)
- Or the bad news are major and non-temporary, so the stock price may decline even more (Often a 👎 Sell Signal)

Many recent positive news may lead to stock price increase which means that the stock now is more expensive than it was recently, and the explanations are the following:

- Either the good news are minor and temporary, so the stock will soon drop back to its usual price soon (Often a 👎 Sell Signal)
- Or the good news are major and non-temporary, so the stock price may increase even more in the next days or hours (Often a 👍 Buy Signal)

LSTM-based 7 days stock price prediction (2nd tool)

The second tool is an LSTM model that uses historical data of adjusted stock prices of a chosen stock (data for last 2 years) and predicts the stock price for the next 7 days. The output of these predictions can be seen in a time-series plot (lower-right side of dashboard). The data are acquired by yfinance library that gets data from Yahoo's API. It must be noted that yfinance returns the following fields:

- Date: The date in which the record corresponds to
- Open: The stock's opening price of the day
- High: The stock's highest price of the day
- Low: The stock's lowest price of the day
- Close: The stock's closing price of the day
- Adjusted Close: The stock's closing price of the day, adjusted to stock's splits, dividends, and other corporate actions

In our analysis, we will use Adjusted Close price only. It is notable that no model weights need to be saved, and in each stock selection from the user, the data acquisition and model training happen on the go really fast.

`Interpretation for traders`

- Upwards projection could be a potential 👍 Buy Signal (especially for large R-squared scores, which can be seen in the plots)
- Downwards projections could be a potential 👎 Sell Signal (especially for large R-squared scores, which can be seen in the plots)

Technical Analysis Methods (3rd tool)

This set of four methods are out of the scope of the Machine Learning and Content Analytics course but they were added because we believe that they can be used supplementary with the previous two tools and provide a more realistic overview for the future of a stock.

The Technical Analysis Methods are Statistical Methods for stock trading which usually use price rolling averages and standard deviations, in order to return their outlook (e.g., Buy, or Hold, or Sell). These methods usually belong to one of the following trading timeframes:

- Short term trading (Buying and Selling Stocks within hours, minutes or even seconds)
- Medium term trading (Buying and Selling Stocks within days or weeks)
- Long term trading (Buying and Selling Stocks within months)

To use the following methods, we have to get the real time stock price when the NYSE and NASDAQ markets are open by scraping the current price of the chosen stock from marketwatch.

1st method: Bollinger Bands of 20 days and 2 standard deviations

This method is a "Medium-term" trading method (days or weeks) and relies on 3 lines:

- the 1st one is the 20-day rolling average of the stock price (middle line)
- the 2nd one is the 20-day rolling average + 2 standard deviations (Upper Limit)
- the 3rd one is the 20-day rolling average - 2 standard deviation (Lower Limit)

`Interpretation for traders`

- When a stock's current price is between the rolling average and the Upper Limit, it is considered as Overbought (which is a 👎 Sell Signal)
- When a stock's current price is between the rolling average and the Lower Limit it is considered as Oversold (which is a 👍 Buy signal)
- When a stock's current price is equal to the rolling average, it is considered as neither Overbought or Oversold, (which is a ✊ Hold Signal)
- When a stock's current price is higher than the upper limit or lower than the lower limit, it is considered as Unusual Event (which signals ❗ Caution)

In Figure 2 on the next page, we can see that Apple's stock is considered as a "Buy" currently, according to the Bollinger Bands of 20 days and 2 standard deviations.

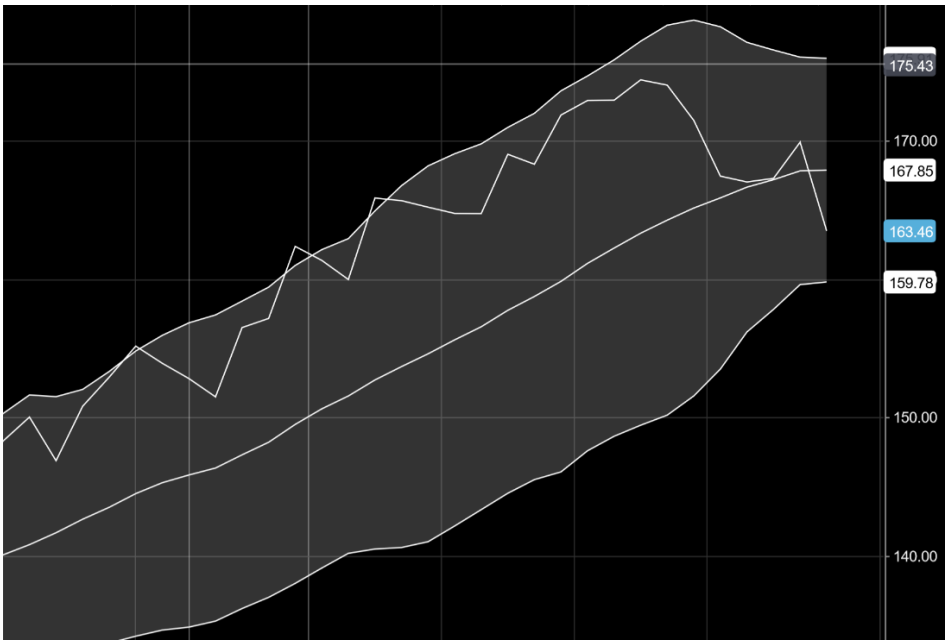


Figure 2: Bollinger Bands of 20 days and 2 standard deviations for Apple stock

2nd method: Bollinger Bands of 10 days and 1.5 standard deviations

This method is a "Short-term" trading method (hours, minutes or even seconds) and relies on 3 lines:

- the 1st one is the 10-day rolling average of the stock price (middle line)
- the 2nd one is the 10-day rolling average +1.5 standard deviations (Upper Limit)
- the 3rd one is the 10-day rolling average -1.5 standard deviation (Lower Limit)

`Interpretation for traders`

- When a stocks current price is between the rolling avg. and the Upper Limit, it is considered as Overbought (which is a 👎 Sell Signal)
- When a stocks current price is between the rolling avg and the Lower Limit it is considered as Oversold (which is a 👍 Buy signal)
- When a stocks current price is equal to the rolling average, it is considered as neither Overbought or Oversold, (which is a ✊ Hold Signal)
- When a stocks current price is higher than the upper limit or lower than the lower limit, it is considered as Unusual Event (which signals ❗ Caution)

In Figure 3 on the next page, we can see that Lyft's stock is considered as a "Buy", according to the Bollinger Bands of 10 days and 1.5 standard deviations.

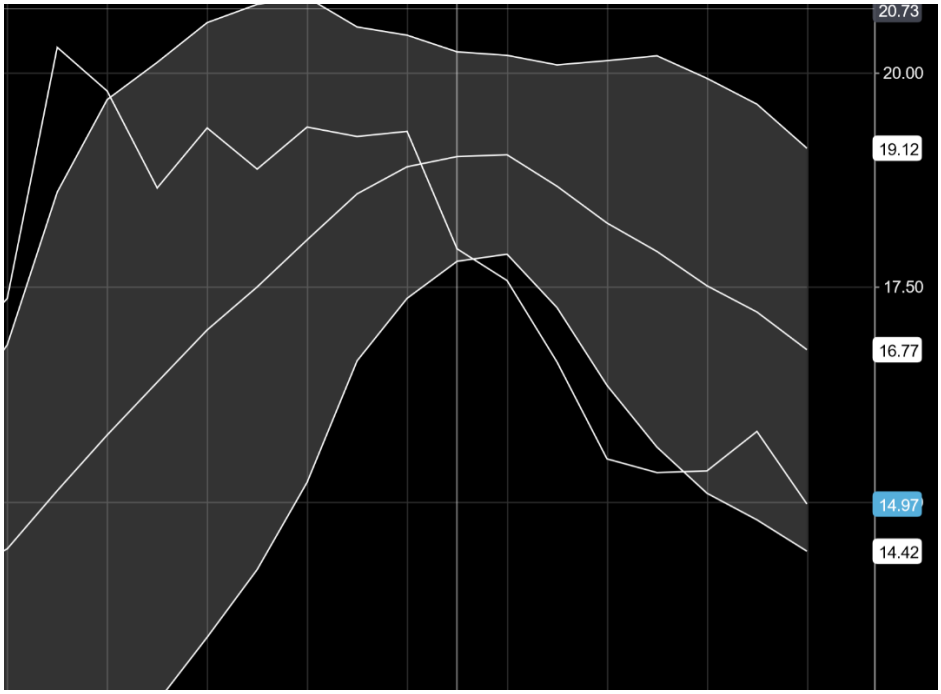


Figure 3: Bollinger Bands of 10 days and 1.5 standard deviations for Lyft stock

3rd method: Bollinger Bands of 50 days and 3 standard deviations

This method is a "Long-term" trading method (weeks) and relies on 3 lines:

- the 1st one is the 50-day rolling average of the stock price (middle line)
- the 2nd one is the 50-day rolling average +3 standard deviations (Upper Limit)
- the 3rd one is the 50-day rolling average -3 standard deviation (Lower Limit)

`Interpretation for traders`

- When a stocks current price is between the rolling avg. and the Upper Limit, it is considered as Overbought (which is a 👎 Sell Signal)
- When a stocks current price is between the rolling avg and the Lower Limit it is considered as Oversold (which is a 👍 Buy signal)
- When a stocks current price is equal to the rolling average, it is considered as neither Overbought or Oversold, (which is a ✊ Hold Signal)
- When a stocks current price is higher than the upper limit or lower than the lower limit, it is considered as Unusual Event (which signals ❗ Caution)

In Figure 4 on the next page, we can see that MongoDB's stock is considered as a "Sell" currently, according to the Bollinger Bands of 50 days and 3 standard deviations.

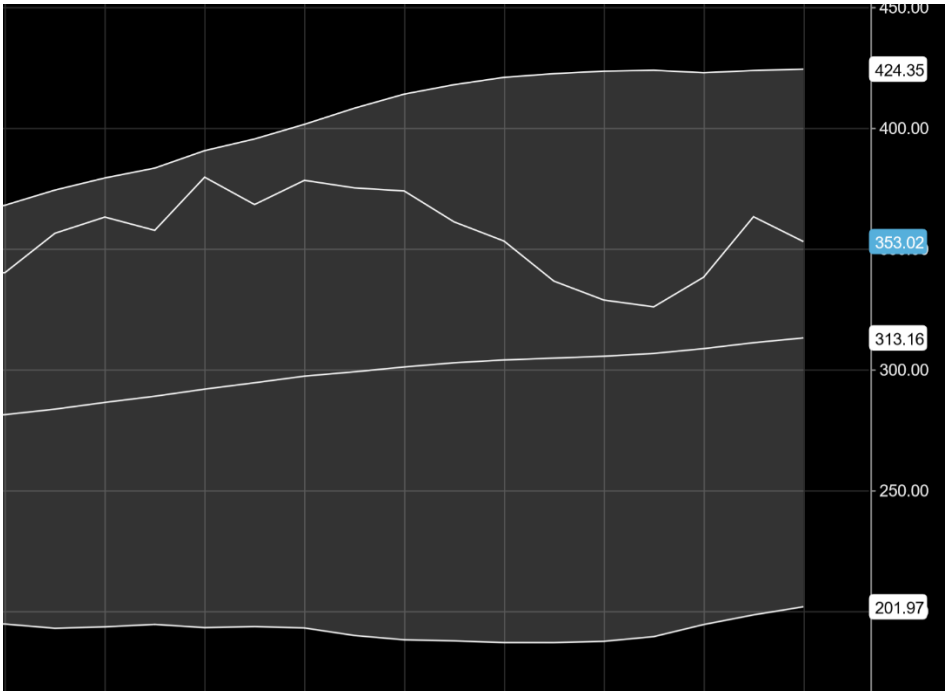


Figure 4: Bollinger Bands of 50 days and 3 standard deviations for MongoDB stock

4th method: Moving Average Convergence Divergence (MACD)

MACD is a trend following momentum indicator based on 2 lines:

- The 1st one is the difference between the 26-day exponential moving average (EMA) and the 12-day EMA (usually called White Line)
- The 2nd one is the 9-day EMA (usually called Red Line)

`Interpretation for traders`

- When White Line > Red Line the stock is considered to have a Downtrend and No Signal outlook (which is a 👎 Sell Signal)
- When White Line < Red Line the stock is considered to have an Uptrend and No Signal (which is a 👍 Buy signal)
- When White Line > Red Line and the former crossed the latter from upwards today or yesterday, the stock's outlook is Downtrend and Sell (which is a 👎 Sell Signal)
- When White Line < Red Line and the former crossed the latter from downwards today or yesterday, the stock's outlook is Uptrend and Buy (which is a 👍 Buy signal)

In Figure 5 on the next page, we can see that PayPal's stock has a "Downtrend and No Signal" outlook currently (since the white line < red line), according to the MACD.

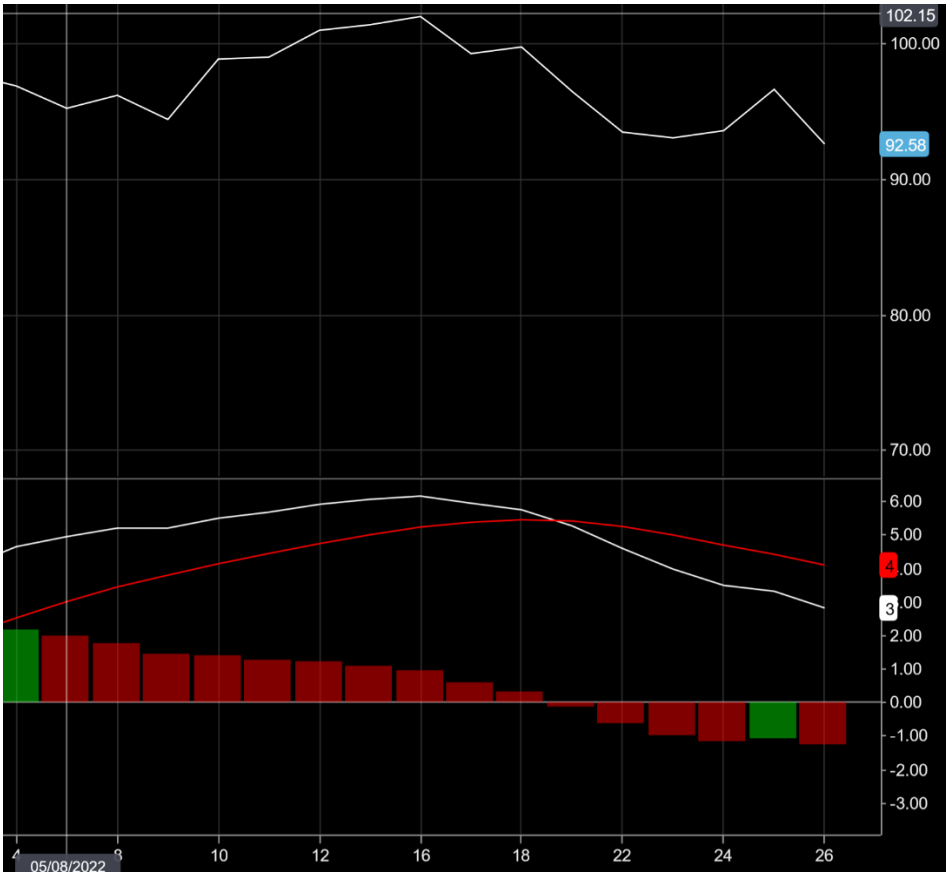


Figure 5: Moving Average Convergence Divergence (MACD) for PayPal stock

Final Thoughts

None of the above methods is used alone for stock trading, so we expect that using them all together when deciding the next trade is the best solution. After manually testing this simple, yet intuitive web app, we concluded that it can be valuable mostly when all 3 tools "agree" on their outlooks.

Our Vision and Goals

Regarding the vision of our team, we can say with confidence that from the beginning of this project it was divided into two categories, the one concerning our academic goals and the one that has to do with the business objectives of our initiative. To be more specific:

ACADEMIC GOALS

1. **Deep understanding of NN models:** Through this project, we set out to understand as best as possible the way Neural Network models work, as well as how they can be used to achieve/solve a specific problem. For this reason, we studied a multitude of articles concerning the broader framework of construction and operation of NN models and managed to achieve a decent result through constant experimentation with them.

2. **Combining models to achieve a common and complex problem:** To achieve our plan, it was necessary to create a dashboard which contains two different NN models. Through this project we set from the beginning the goal of learning to combine different codes in a common environment so that their result brings a powerful solution to our original problem.

BUSINESS GOALS

1. **Providing an innovative business tool to the public:** As the initial business goal of our project, our team set the creation and free availability of a "smart" dashboard that would achieve what other platforms provide separately and/or for a fee. Although there are currently various models on the internet that work in a similar way, there are few free options available to users, that combine both sentiment analysis and stock price prediction techniques in a single environment along with various technical analysis methods to advice the potential traders for their next move in a fast and simple way.

2. **Providing an immediate solution to ourselves, through a tool on which we can base our future investment moves:** Due to the active involvement of our team members in the stock trading part, we decided from the beginning that one of the business goals of our venture would be to create a tool that we could trust without second thoughts. Even if some platforms provide corresponding services with great reliability, this dashboard is a product of our own effort, which means that we not only know its weaknesses and strengths, but we can also modify it at any time and moment as we wish, without relying on third parties.

Introduction to Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory Networks (LSTMs) are special kind of RNN, and they are capable of learning long term dependencies. They were created and designed to avoid long term dependency problems and remembering information for long periods is their strong advantage against the other Deep Learning Models.

LSTM neural networks have a chain of repeating modules of neural networks like all the RNN models. But in their case, the module has a different structure: unlike the rest of the recurrent neural networks that have a single neural network layer, LSTMs have 4.

LSTMs have feedback connections meaning that they can process entire sequences of data without treating each point in the sequence independently, but rather, keeping useful information about previous data in the sequence to help with the processing of new data points. For this reason, LSTM models are extremely good at processing large sequences of data such as text, speech, and general time-series to make valid and useful predictions.

LSTM Architecture

LSTMs use a series of gates that they control how the information in a sequence of data comes into, saved, and leaves the network. There are 3 gates in an LSTM Network:

1) The Forget gate
2) The Input gate
3) The Output gate

These gates can be considered as filters that each one of them contains its own neural network.

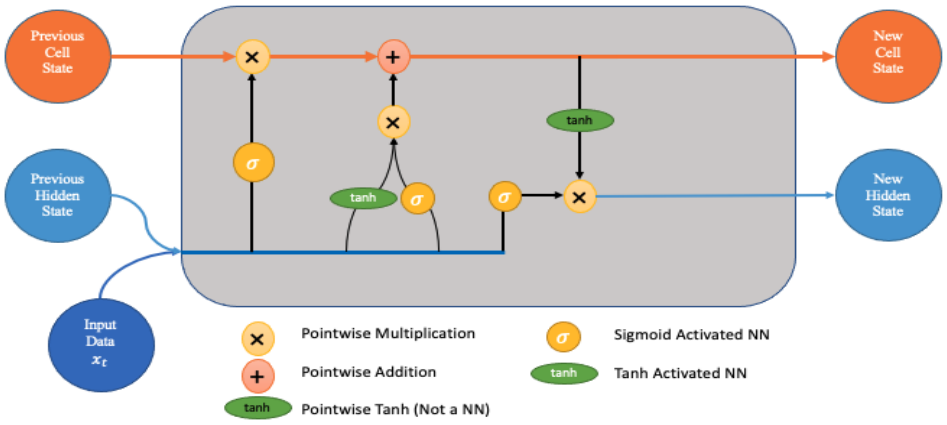


Figure 6: LSTM Diagram

1. Forget gate

In this step it will be decided which of the bits of the long-term memory of the network are useful given both the previous hidden state and new input data.

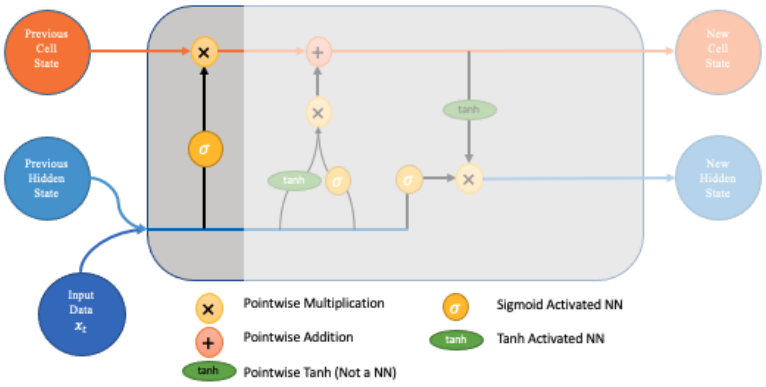


Figure 7: LSTM Forget Gate

The previous hidden state of the model and the new input data are fed into the neural network. The network creates a vector that every element is between 0 and 1 . The network is trained so that every input component to be 0 if it is irrelevant and closer to 1 if it is relevant.

The output values are sent and pointwise multiplied with the previous cell state. This multiplication means that the components that the have been considered irrelevant by the forget gate will be multiplied with a number close to 0 making them have less influence on the following steps of the procedure.

To sum up, forget gate decides which of the components of the long, short memory should be forgotten (they usually have the smallest weight among the other components).

2. **Input Gate**

The next step involves the new memory network and the input gate. The goal of this gate is to decide which of the new information should be added in the long-term memory of the network given the previous hidden state and the new data that has been imported.

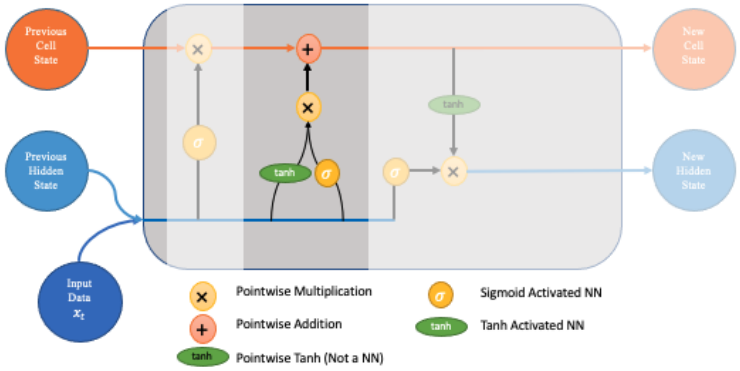


Figure 8 : LSTM Input Gate

The new memory network and the input gate are also neural networks and they both take the same inputs, hidden states, and the new input data . The inputs in input gate are the same with the ones in the forget gate.

The new memory network is a Tanh activated neural network that has learned to combine the previous hidden state and the new input data to create a new memory update vector. This vector contains information from the new input data given the state of the previous hidden state. It is an informative vector as it informs the user how much to update each component of the long - short memory.

3. **Output gate**

Given that the updates of the long-term memory of the network have been completed we can proceed with the last step, the output gate, where the new hidden state of the network will be decided.

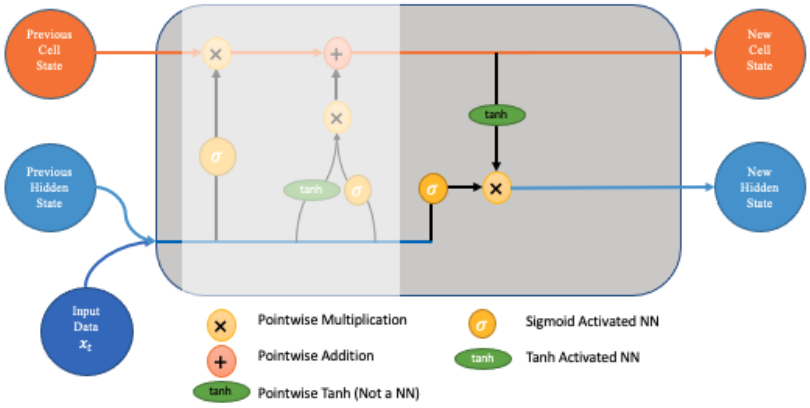


Figure 9: LSTM Output gate

The output gate has the same inputs with the forget gate and the activation function is also sigmoid (since we want the filter property gained from outputs in [0,1]). The purpose of the output gate is to apply the filters to the newest updated cells meaning that only the necessary information will be extracted , that is why we pass the cell state into a Tanh function making the values into the interval of -1 and 1.

To be more specific the process of the output gate is as follows:

1) The Tanh function is applied to the current cell state to acquire a new squished cell state which lies in [-1,1].
2) Pass the previous hidden state and the input data through the sigmoid neural network to obtain the filtered vector.
3) By making pointwise multiplication apply the filter vector to the squished cell state.
4) Output the new hidden state.

Introduction to BERT Models

Bidirectional Encoder Representations from Transformers (BERT) is a Machine Learning model for natural language processing. Natural Language Processing (NLP) is the field of artificial intelligence aiming to make computers read, analyse, interpret, and derive meaning from text and spoken words. NLP is behind Google Translate, voice assistants (Alexa, Siri, etc.), chatbots, Google searches, voice-operated GPS, and more.

BERT model was developed by Google and can be used for the most common language tasks such as sentiment analysis and named entity recognition. This model revolutionized the NLP space by solving better than its predecessors many tasks. BERT can be used on a wide variety of tasks such as:

- Sentiment Analysis
- Question answering
- Text prediction
- Text generation

- Summarization
- Polysemy resolution

At this point you may wonder why you are reading about BERT as in our project we used FinBERT. The answer is pretty simple, FinBERT is a language model based on BERT to tackle NLP tasks in financial domain. So, we will first present how BERT models work and then we will dive into the world of FinBERT.

How does BERT Work?

BERT works by leveraging:

1. **Large amounts of training data**

BERT was trained on Wikipedia (~2.5B words) and Google's Books Corpus (~800M words). Training on a dataset this large takes a long time. BERT's training was made possible thanks to the Transformer architecture and sped up by using TPUs (Tensor Processing Units - Google's custom circuit built specifically for large ML models). More specifically, 64 TPUs trained BERT over the course of 4 days.

2. **Mask Language Model**

Mask Language Model enables a bidirectional learning from text by masking (hiding) a word in a sentence and forcing the model to bidirectionally use the words on either side of the covered word to predict the masked word.

3. **Next Sentence Prediction**

NSP (Next Sentence Prediction) is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not.

4. **Transformers**

The Transformer architecture makes it possible to parallelize machine learning training extremely efficiently. Transformers use an attention mechanism to observe relationships between words and work by leveraging attention, a powerful deep-learning algorithm. Machine Learning models need to learn how to pay attention only to the things that matter and not waste computational resources on irrelevant information. Transformers solve this problem by creating differential weights signalling which words in a sentence are the most critical to further process. A transformer does this by successively processing an input through a stack of transformer layers, usually called the encoder. If necessary, another stack of transformer layers, the decoder, can be used to predict a target output. Note that, BERT does not use a decoder.

BERT Architecture

BERT's model architecture is based on Transformers. It uses multilayer bidirectional transformer encoders for language representations. BERT comes in many sizes, but we will focus on $BERT_{BASE}$ as FinBERT is based on this model. The $BERT_{Base}$ model uses 12 layers of transformers blocks with a hidden size of 768, number of self-attention heads as 12 and has around 110M trainable/learnable parameters. Figure 10 presents the architecture of $BERT_{BASE}$ model.

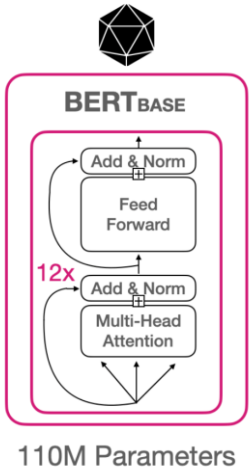


Figure 10: $BERT_{Base}$ Architecture

The transformer block transforms a sequence of word representations to a sequence of contextualized words. The hidden size of the model represent the layers of mathematical functions located between the input and the output that assigns weights to the words (tokens) to produce the desired result while, the attentions heads represent the size of a Transformer block. BERT uses the same model architecture for all NLP tasks with minimal change such as adding an output layer for classification.

The input to the model has to be given in a single sequence. BERT uses special tokens [CLS] and [SEP] to understand input properly. [SEP] token has to be inserted at the end of a single input. When a task requires more than one input (e.g., Question – Answering tasks), [SEP] token helps the model to understand the end of one input and the start of another input in the same sequence input. [CLS] is a special classification token and the last hidden state of BERT corresponding to this token (h[CLS]) is used for classification tasks. BERT uses Wordpiece embeddings input for tokens. Along with token embeddings, BERT uses positional embeddings and segment embeddings for each token. Positional embeddings contain information about the position of tokens in sequence. Segment embeddings help when model input has sentence pairs. Tokens of the first sentence will have a pre-defined embedding of 0 whereas tokens of the second sentence will have a pre-defined embedding of 1 as segment embeddings. Figure 11 presents the input – output format of the BERT model.

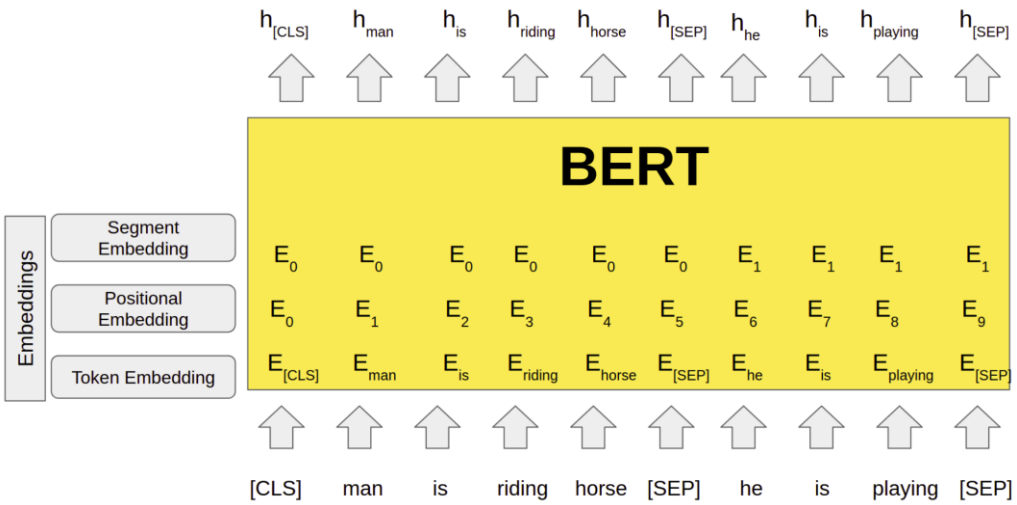


Figure 11: Input-Output Format of BERT Model

The Final Embeddings used by model architecture are the sum of token embedding, positional embedding as well as segment embedding. The final embeddings are then fed into the deep bidirectional layers to get output. The output of the BERT is the hidden state vector of pre-defined hidden size (in $BERT_{Base}$ case the hidden size is 768) corresponding to each token in the input sequence. These hidden states from the last layer of the BERT are then used for the desired NLP task.

FinBERT: A BERT Model for Financial Sentiment Analysis

Financial sentiment analysis is a challenging task due to the specialized language and the lack of labelled data in this domain. Pre-trained language models can help with this problem because they require fewer labelled examples and can be further trained on domain – specific corpora. FinBERT is a pre-trained language model from HuggingFace (the link of the model can be found here) based on BERT and can tackle NLP tasks in the financial domain. The sentiment classification with FinBERT is made by adding a dense layer after the last hidden state of the [CLS] token as this is the recommended practice for using BERT for any classification task. Then, the classifier is trained on the labelled sentiment dataset. Figure 12 on the next page, presents an overview of the hole classification process to extract the sentiment of financial texts.

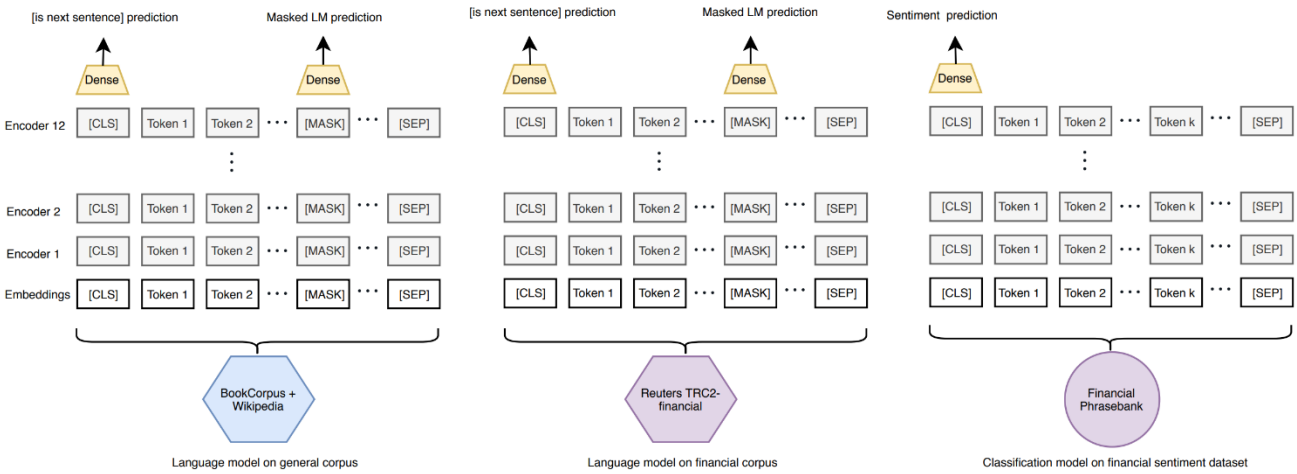


Figure 12: Process Overview for Sentiment Prediction with FinBERT

Methodology

Our models target to 10 specific tech stocks. These stocks are:

1. **AAPL** (Apple)
2. **NVDA** (Nvidia)
3. **PYPL** (PayPal)
4. **MSFT** (Microsoft)
5. **AMZN** (Amazon)
6. **SPOT** (Spotify)
7. **TWTR** (Twitter)
8. **ABNB** (Airbnb)
9. **UBER** (Uber)
10. **GOOG** (Google)

We chose to use a limited number of stocks for model predictions as in this way we can manipulate much easier the development of our models.

LSTM Price Prediction Model Methodology

The idea behind the creation of the LSTM model was to scrap the yahoo finance website to download the data of the stock of our liking for the last 2 years. After that we transformed the data and split them into training and testing datasets. Also, we made a 3D array as this is the form that the model needs to process the data. As every stock has a unique data pattern, it would be wrong if we just used a single LSTM model for all the 10 stocks. To avoid any wrong calculations, we created 10 different LSTM models. Every single model is hyper-tuned with the best parameters and layers to have the bests results for the specific stock. With the help of an IF ELSE function, we were able to train each model in real-time and return accurate predictions to our client for every predefined stock, he wishes for.

Finally, after training and testing our model, we were ready to provide our customer with predictions for the next 7 days. To predict tomorrow's value, we had to feed into the model the past 60 (lookback) days' values, and we get tomorrow's value as output. To get the day after tomorrow's value, we fed-in past 59 days' values along with tomorrow's value and the model output day after tomorrow's value. At the end of the forecasting procedure for the next 7 days a browser window will be opened with an interactive visualization if someone chooses to not use the Streamlit dashboard. Otherwise, the predictions visualization for this model is embedded in the lower-right corner of our dashboard.

FinBERT Sentiment Analysis Model Methodology

To use the FinBERT model we import it from HuggingFace Transformers library. We chose not to use the model as is but to experiment with it and tune its hyper-parameters. For this reason, we import the Financial PhraseBank dataset to use it for further training (more about the dataset in Dataset Overview chapter). After importing the data, we spit them in training, test and validation and we map the labels of each observation in order to convert the categorical labels (Positive, Negative, Neutral) to numerical labels (0, 1, 2). For the splitting we choose to use stratification to force the distribution of the target variable (classes) among the different splits to be the same. After this step we load the tokenizer that we will use alongside with our model, and we be responsible to convert the inputs of the model into a format that can be processed by FinBERT. The tokenizer creates distinct token for each observation in order to feed them to the model.

The model returns a logit probability for each record and then based on the highest value classifies the record as Positive, Negative or Neutral. For the evaluation of the model, we use the accuracy score which is calculated based on the true label of each record and the predicted label. After tuning the model, we are ready to put it into production. The trained model takes as input the 17 latest headlines of the stock that the user chooses and then predicts the class of each headline. Last but not least, based on the predicted labels of each headline the visualization that was described in the An Introduction to our Dashboard chapter is constructed.

Data Collection

The data we used for the models is from the yahoo finance and marketwatch webpage. Yahoo finance is an open-source Python library that allows us to acquire stock data from Yahoo Finance without any cost. With the help of the yahoo finance library in python we were able to acquire the data from the last 2 years of every single day till today and use them for the LSTM training and prediction. **We used only the last 2 years because we wanted as little data as possible that has been affected by the covid pandemic**. After collecting our data, we focused on the Adjustive Close Price of each stock as its value contains factors of anything that might affect the stock price after the market closes. From our perspective it would be more accurate that the close price. As far as the data for the FinBERT model, as we have already mentioned above for the training, we used the famous Financial PhraseBank dataset while for the prediction we scrap the headlines of the latest 17 articles from marketwatch for the stock that the user chooses.

Dataset Overview

At this chapter, we present the data used for the training of each model.

LSTM Price Prediction Model Dataset

After acquiring the data from yahoo finance web page, we can visualize the price history of every stock to have an overview of how the price stock is moving between the years.

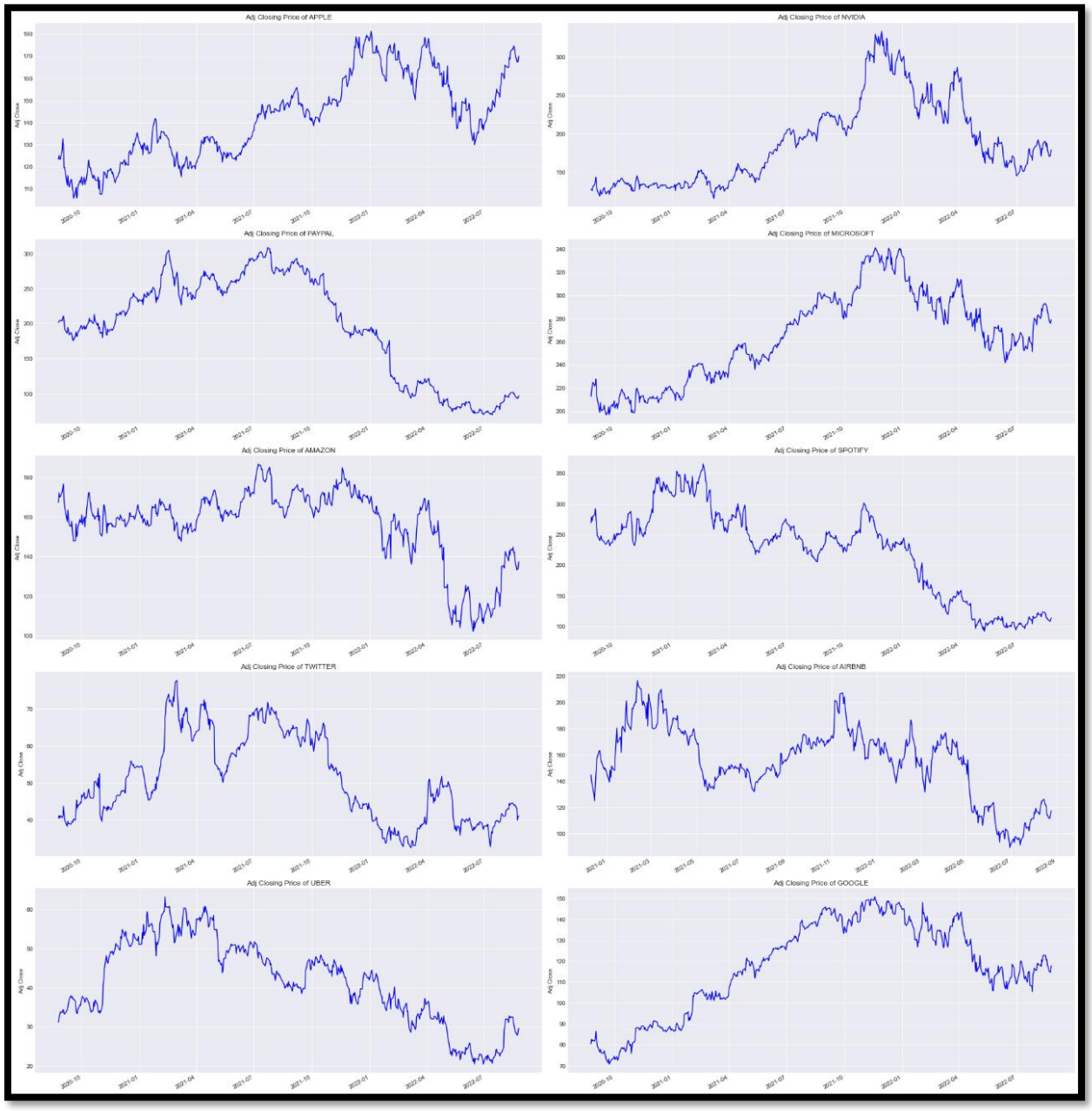


Figure 13: Stock Price History (LSTM Dataset)

It must be noted that stock market prices are highly unpredictable, volatile and they are affected from a great number of factors. This means that there are no consistent patterns in the data that allow us to model stock prices over time near-perfectly, and this is visible in the plots above (Figure 13). However, we expect that LSTM will be able to capture the basic trend of the stock price even approximately.

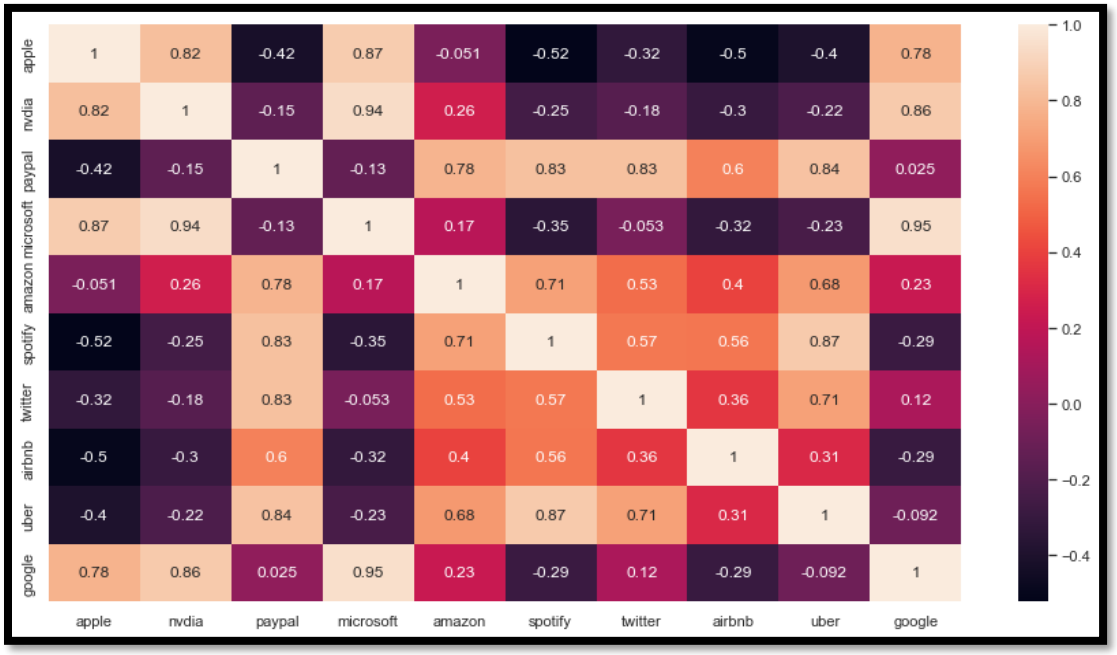


Figure 14: Company Stock Correlation

From Figure 14 we can see that the highest linear correlation exists between **Microsoft** and **Google** with score of 0.95 . That means that when the price of one will increase, the price of the other will also increase. The same thing applies and for the **Microsoft** with the **Nvidia**. The reason between this high correlation is that these 3 companies are partners in the Technology Field. For the best diversification strategy, we would advise our client to buy shares of one of these 3 companies if the stock of one of them goes down and sell accordingly if it goes up.

FinBERT Sentiment Analysis Model Dataset

For the training of the FinBERT model we use the Financial PhraseBank dataset which consists of 4846 English sentences selected randomly from financial news found in LexisNexis database. These sentences were annotated by 16 people with background in finance and business. The annotators were asked to label each sentence according to how they think this information might affect the mentioned company stock price. It must be noted that the majority of labels in the dataset are Neutral and Negative (2879 Negative, 1368 Neutral and 604 Positive) as these classes are the most important to classify correctly.

Data Processing/Annotation/Normalization

In this chapter we present all the required steps for data preparation to convert the inputs of each model into a usable format.

LSTM Price Prediction Model Data Preparation

In order to feed the LSTM model with the data we had to perform some procedures to transform the data in a way a respective model would accept.

To build an LSTM model, we had to separate our stock prices data into a training set and a test set. Besides, we had also to normalize our data so that all the values would be ranged from 0 to 1. For our goal we only needed the adjusted closing prices from our dataset in order to train our LSTM model. We were able to extract 80% of the adj. closing prices from our stock data as our training dataset. With the help of the **Scikit-Learn MinMaxScaler** we were able to normalize our data ranging from 0 to 1 and after that we reshaped it into a 2-Dimensional array.

After that we created a 60-days window of historical prices (i-60) as our feature data (x_train) and the following 60-days window as label data (y_train).

Moreover, we transformed our data into a **NumPy** array because is the format that is accepted by **TensorFlow** for the training of the LSTM model. Finally, in order to feed our model with the transformed data we had to reshape the x_train and y_train into a 3-Dimensional array.

We implemented the same procedure and for the 20% of the data that we used as testing dataset.

FinBERT Sentiment Analysis Model Data Preparation

For the FinBERT model the procedure is simpler as we use a Tokenizer. After splitting the data, separating the target variable, and mapping the classes so as to have numerical labels, we feed the records into the tokenizer. The tokenizer will map our record in a form that is acceptable by our model. The Adjusted Close that we use is based on the class PreTrainedTokenizerFast. This base class implements the common methods for encoding string inputs in model inputs and instantiating or saving python and “Fast” tokenizers either from a local file or directory or from a pretrained tokenizer provided by the HuggingFace library.

In our case, the tokenizer is imported from the HuggingFace library.

The “Fast” (PreTrainedTokenizerFast class) implementation allows:

- a significant speed-up when doing batched tokenization
- additional methods to map between the original string (character and words) and the token space (e.g., getting the index of the token comprising a given character or the span of characters corresponding to a given token).

PreTrainedTokenizerFast implements the main methods for using all the tokenizers:

- Tokenizing (splitting strings in sub-word token strings), converting tokens strings to ids and back, and encoding/decoding (i.e., tokenizing and converting to integers).
- Adding new tokens to the vocabulary in a way that is independent of the underlying structure
- Managing special tokens (like mask, beginning-of-sentence, etc.): adding them, assigning them to attributes in the tokenizer for easy access and making sure they are not split during tokenization.

The output of the tokenizer is the input data of our model.

Algorithms and NLP Architectures

This chapter presents the algorithms developed for model creation as well as their architectures. For a more detailed overview of the code and the architecture of each model feel free to check the LSTM_EDA_&_EVALUATION.ipynb and the FinBERT_EDA_&_EVALUATION.ipynb files.

LSTM Price Prediction Model

With the help of the TensorFlow machine learning library in python we were able to build our LSTM model. The next figure is providing us with insides about the creation of the model. We will present one of the 10 stock models as the methodology of the model creation is the same for all of them. To avoid having a lot of lines of code for the creation of the models we created a function (**LSTM trainer()**) that takes as input the **seed**, **dropout**, **LSTM_units**, **patience**, **batch size**, and the **epochs.**

```
def LSTM_trainer(seed, DROPOUT, LSTM_units,patience,batch_size, epochs):

    tf.random.set_seed(seed)
    DROPOUT = DROPOUT
    global model_lstm
    model_lstm = keras.Sequential()
    model_lstm.add(layers.LSTM(LSTM_units, return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model_lstm.add(Dropout(rate=DROPOUT))
    model_lstm.add(layers.LSTM(LSTM_units, return_sequences=False))
    model_lstm.add(Dropout(rate=DROPOUT))
    model_lstm.add(layers.Dense(25))
    model_lstm.add(Dropout(rate=DROPOUT))
    model_lstm.add(layers.Dense(1))
    model_lstm.add(Activation('linear'))

    print('\n')
    print("Compiling the LSTM Model for the "  + str(ticker) + " stock....\n")
    t0 = time.time()
    model_lstm.compile(optimizer='adam', loss='mean_squared_error',metrics=['mae'])
    callback=EarlyStopping(monitor='val_loss',
                           min_delta=0,
                           patience=patience,
                           verbose=1, mode='auto')
```

Figure 15: LSTM Model creation

To be able to feed the function we created an **if-else** procedure that every time the user chooses a specific stock, applies the best hyperparameters to the function for the creation of the best model possible for the given stock.

```
if ticker == 'AAPL':
    LSTM_trainer(1, 0.2, 100,2, 20, 30)
elif ticker == 'NVDA':
    LSTM_trainer(2, 0.2, 100,2, 30, 50)
elif ticker == 'PYPL':
    LSTM_trainer(6, 0.2, 100,10,25, 30)
elif ticker == 'MSFT':
    LSTM_trainer(4, 0.1, 80, 2,20, 40)
elif ticker == 'AMZN':
    LSTM_trainer(6, 0.1, 120,2, 20, 25)
elif ticker == 'SPOT':
    LSTM_trainer(9, 0.2, 200,5, 20, 40)
elif ticker == 'TWTR' :
    LSTM_trainer(15, 0.2, 100,4,20, 40)
elif ticker == 'UBER':
    LSTM_trainer(15, 0.2, 100,7,20, 40)
elif ticker == 'ABNB':
    LSTM_trainer(15, 0.2, 120,8,20, 40)
elif ticker == 'GOOG':
    LSTM_trainer(15, 0.2, 100,3,20, 25)
```

Figure 16: LSTM Hyperparameters of each stock

Commenting Figures 15 – 16 for the Apple Stock

1. We fed the model with 100 network units and set the return sequence = True to be able to have the output of the layer as a sequence of the same length.
2. We have also added a second LSTM layer with 100 units but this time, we set the return sequence = False to return the last output.
3. As a third dense layer we used a neural network layer with 25 units (a dense layer feeds all outputs from the previous layer to all its neurons, each neuron providing one output to the next layer).
4. As a fourth layer we added a densely connected layer that specifies the output of 1 network unit.
5. Set the activation function to be linear as we have a Regression Problem.
6. At last, to avoid overfitting, we used the dropout parameter equal with 0.2.
7. To avoid our model to be overfitted we inserted the Early Stopping function. The main idea behind early stopping is to stop training before the model starts to overfit on the training data. In Apple's case the **patience = 2** is the number of epochs with no improvement after which training will be stopped.
8. In every model the best parameters have been found manually, so for every stock we set a seed in the creation of the LSTM model to keep the same weights in every iteration.
9. From Figure 16 we can see that the hypermeters are different for every stock, but the explanation is the same for all of them.

FinBERT Sentiment Analysis Model

As we mentioned above FinBERT is based on Bert and specifically on $BERT_{Base}$.

$BERT_{Base}$ base consists of 12 stacked Transformer encoders. The transformer is an attention-based architecture for modeling sequential information and includes an encoder - decoder mechanism. Encoders consist of multiple identical Transformer layers where each layer has a multi-headed self-attention layer and a fully connected feed-forward network. For one self-attention layer, three mappings from embeddings are learned (key, query, value). Using each token's key and all tokens' query vectors, a similarity score is calculated with dot product. These scores are used to weight the value vectors to arrive at the new representation of the token. With multi-headed self attention, these layers are concatenated together, so that the sequence can be evaluated from varying perspectives. Then the resulted vectors go through fully connected networks with shared parameters. FinBERT for Sentiment Classification is conducted by adding a dense layer after the last hidden state of the [CLS] token. Then, the classifier network is trained on the labelled sentiment dataset.

To conclude the overall architecture of our model is the following:

- 1 Embeddings Layer

- 12 Transformer Encoder Layers

- 1 Classification Layer

For the training of the model, we set the hyperparameters as seen in Figure 17.

```
train_args = TrainingArguments(
    './Finbert Trained/',
    per_device_train_batch_size=16,
    per_device_eval_batch_size=2*16,
    num_train_epochs=5,
    learning_rate=2e-5,
    weight_decay=0.01,
    warmup_ratio=0.1,
    do_eval=True,
    do_train=True,
    do_predict=True,
    evaluation_strategy='epoch',
    save_strategy="no",
)
```

Figure 17: FinBERT Hyperparameters

More specifically we chose to set:

- The training batch size to 16 (controls the number of training samples to work through before the model's internal parameters are updated)
- The evaluation batch size to 32 (controls the number of evaluation samples to work through before the model's internal parameters are updated)
- The number of epochs to 5 (the number of complete passes through the training dataset)

- The learning rate to 2e-5 (defines how quickly the neural network updates the concepts it has learned)
- The weight decay to 0.01 (is used to reduce the complexity of a model and prevent overfitting)
- The warmup ratio to 0.1 (the proportion of initial learning rate in relation to the actual rate. Here we start at 10% of our learning rate)

LSTM Setup and Configuration

After proper feeding our model, we are ready to train our LSTM model by fitting it with the training sub dataset. We had to setup an optimizer and a loss function that measures the error of the model.

```
model_lstm.fit(x_train,
               y_train,
               batch_size= batch_size,
               epochs=epochs,
               validation_split=0.1,# ...holding out 10% of the data for validation
               shuffle=True,verbose=0,callbacks=[callback])
```

Figure 18: Compiling and fitting the LSTM model into our data for Apple Stock

We used Adam optimizer. Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.

We used as loss function the **Mean Squared Error**. Mean squared error is calculated as the average of the squared differences between the predicted and actual values. The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0. The squaring means that larger mistakes result in more error than smaller mistakes, meaning that the model is punished for making larger mistakes

We initiate the batch size of the model to 20. The batch size is the hyperparameter that defines the number of samples to work through before updating the internal model parameters. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model.

Last but not least we set the number of epochs to 30. The number of epochs is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. As the number of epochs increases, a greater number of times the weight is changed in the neural network and the curve goes from underfitting to optimal and to overfitting curve.

Finally, Keras can separate a portion of our training data into a validation dataset and evaluate the performance of our model on that validation dataset at each epoch. That is why we set the

validation split parameter equal to 0.1 meaning that in every epoch the model keeps the 10% of the training dataset for evaluation.

FinBERT Setup and Configuration

After setting the hyperparameters presented in Algorithms and NLP Architectures chapter we are ready to train our model. For the training we use the Trainer class which is optimized for Transformers models.

```
trainer = Trainer(
    model,
    train_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
```

Figure 19: Training the FinBERT model into our data

```
def compute_metrics(eval_preds):
    logits, labels = eval_preds
    preds = np.argmax(logits, axis=-1)
    return {'accuracy': accuracy_score(labels, preds)}
```

Figure 20: Computing the accuracy of our FinBERT model

The Trainer class uses the training arguments that was presented in Figure 17, a data collator, and the tokenizer we imported and evaluates the model based on the accuracy score. Data collators are objects that will form a batch by using a list of dataset elements as input. These elements are of the same type as the elements of train_dataset or eval_dataset. To be able to build batches, data collators may apply some processing (like padding). In our case the data collator that was used has padding. Moreover, as we did not determine an optimizer Adam optimizer is used by default. After setting the compute_metrics() function as well as the Trainer class we are ready to train the model.

Results & Quantitative Analysis

LSTM Price Prediction Model

In sample Predictions

The next task we have to perform is the evaluation of the model for each stock. To do that we had to take the trained models and apply them into the predict function to predict the stock prices based on the test sub datasets we have created. After that we denormalized the predicted prices.

In order to evaluate the predictive abilities of the model we computed the r^2 score of each stock and we visualized the predictions to have a clearer image about them.

Stock	R^2 Score
APPLE (AAPL)	0.81
NVDIA (NVDA)	0.73
PAYPAL (PYPL)	0.76
MICROSOFT (MSFT)	0.52
AMAZON (AMZN)	0.58
SPOTIFY (SPOT)	0.55
TWITTER (TWTR)	0.60
UBER (UBER)	0.79
GOOGLE (GOOG)	0.68
AIRBNB (ABNB)	0.64

The r^2 gives an estimation on the fraction of our model's variance that is explained by our model's independent features. If the r^2 is high indicates more variability is explained by the model. Its not a metric that we allow us to choose if the models performs well or not but it is a good indicator to guide the user in his stock selection for investment.

Note: It is important to mention that because every day new data is added in the dataset the r^2 values might change if the new adjusted price of the stock has big influence. The outcomes that are mentioned in this report are based on the end date of 25/8/2022.

Last but not least, we created visualizations to see the prediction in the validation and test set of our model in order to decide if the model performs well or not.

In figures 21 to 30 you can see the prediction visualisation for each stock. The training dataset (80%) is visualized with blue colour, the validation with orange (10% of the training data set) and the testing - Predictions with green (20%).

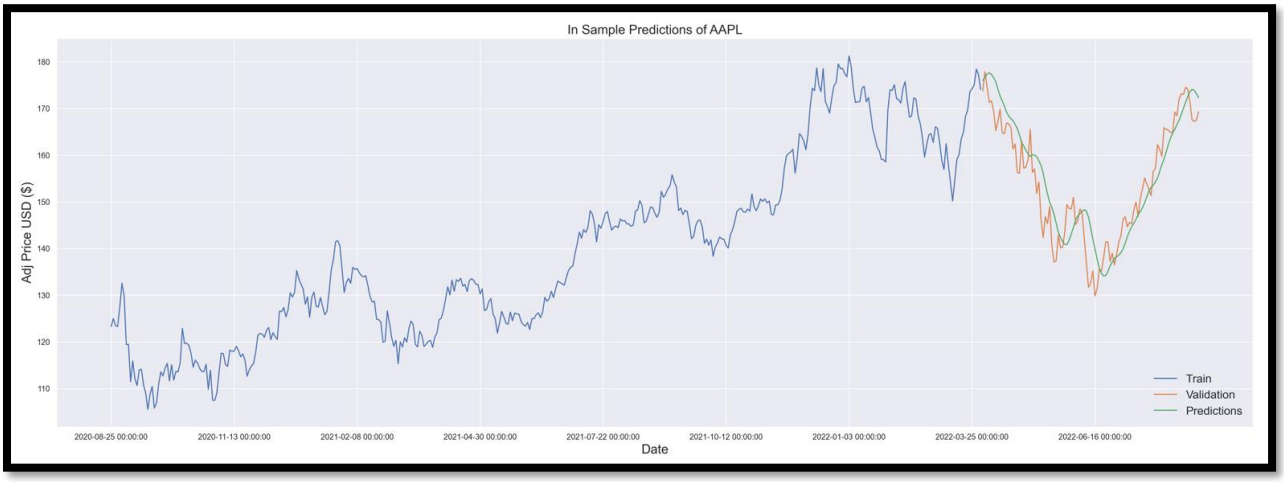


Figure 21: In sample Predictions for the APPLE Stocks

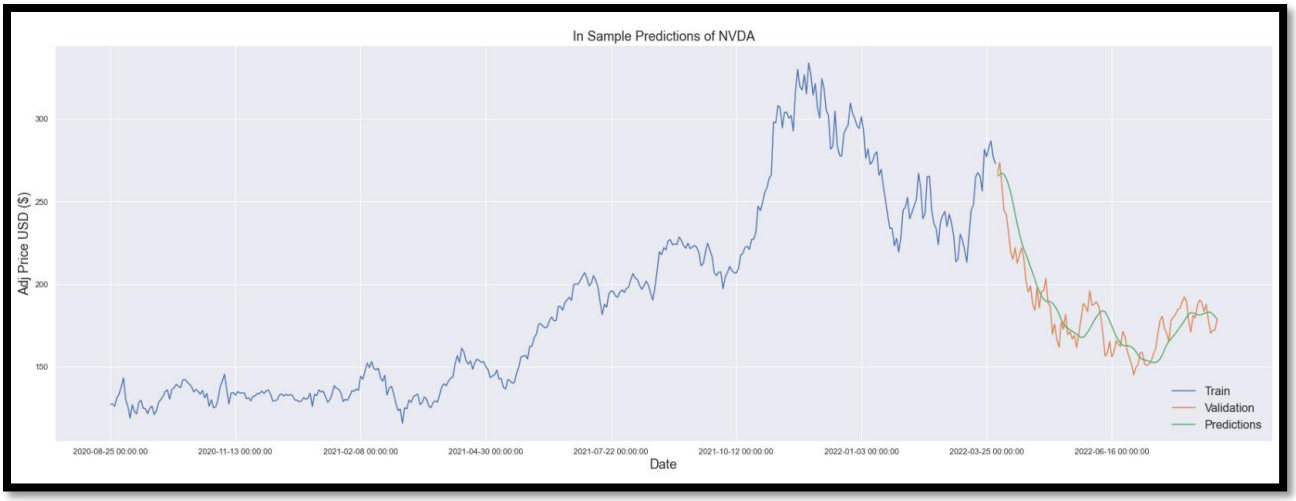


Figure 22: In sample Predictions for the NVDIA Stocks

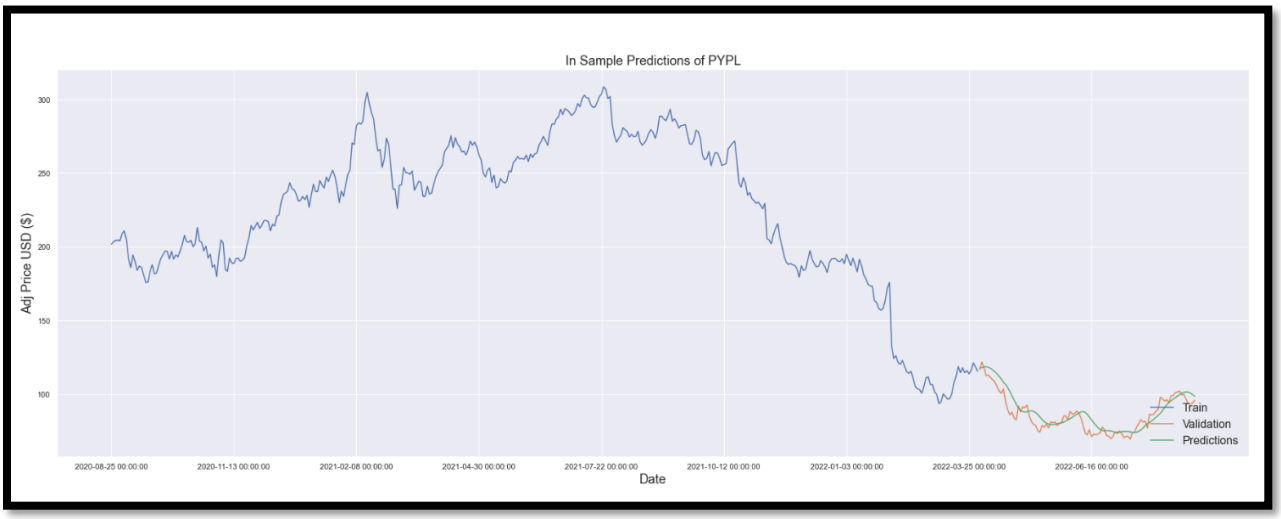


Figure 23: In sample Predictions for the PAYPAL Stocks

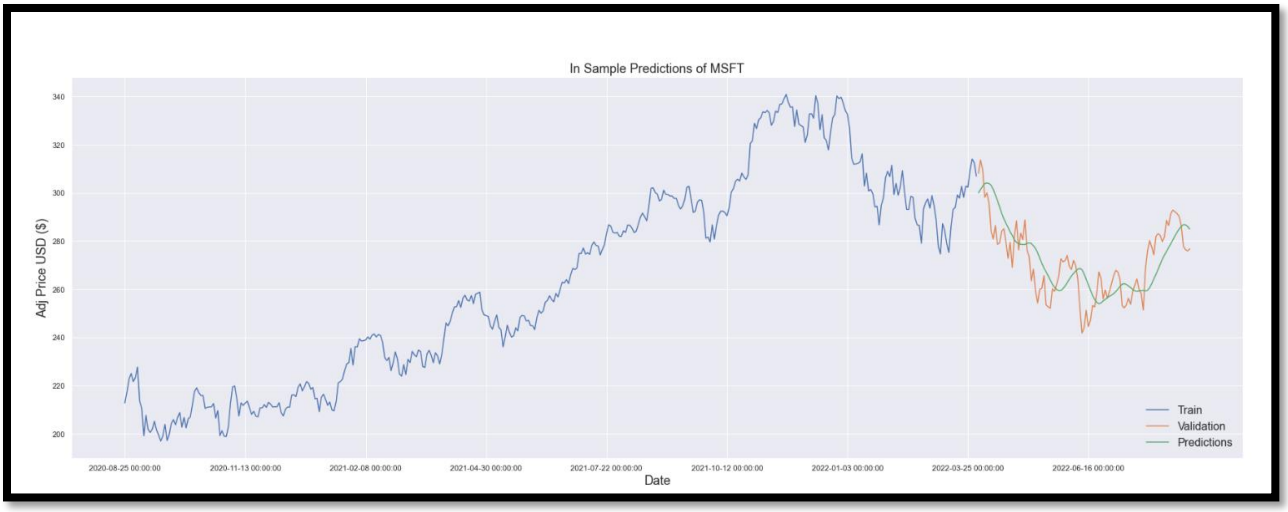


Figure 24: In sample Predictions for the MICROSOFT Stocks

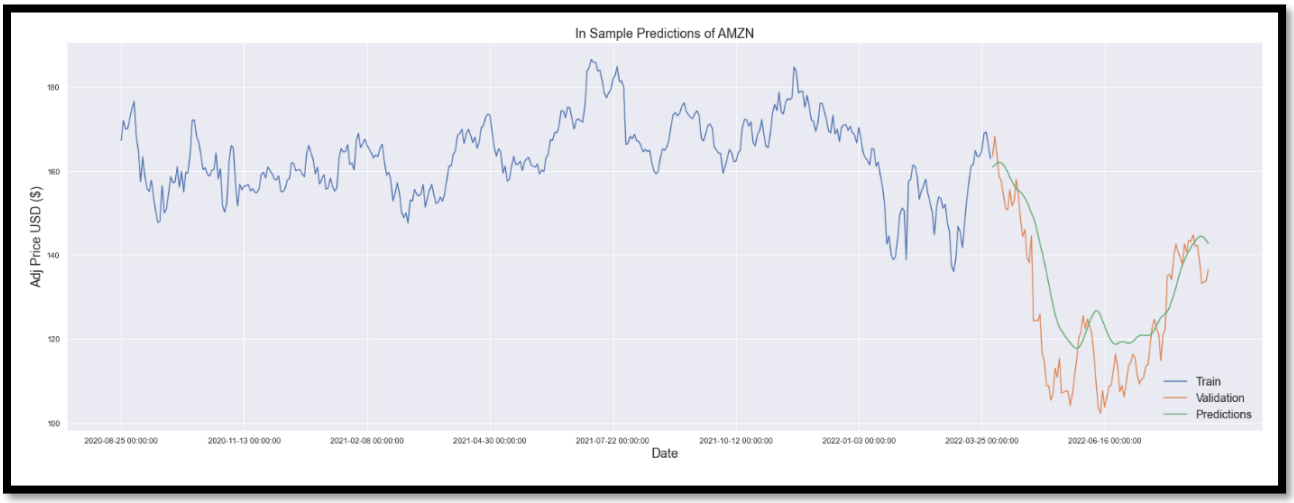


Figure 25: In sample Predictions for the AMAZON Stocks

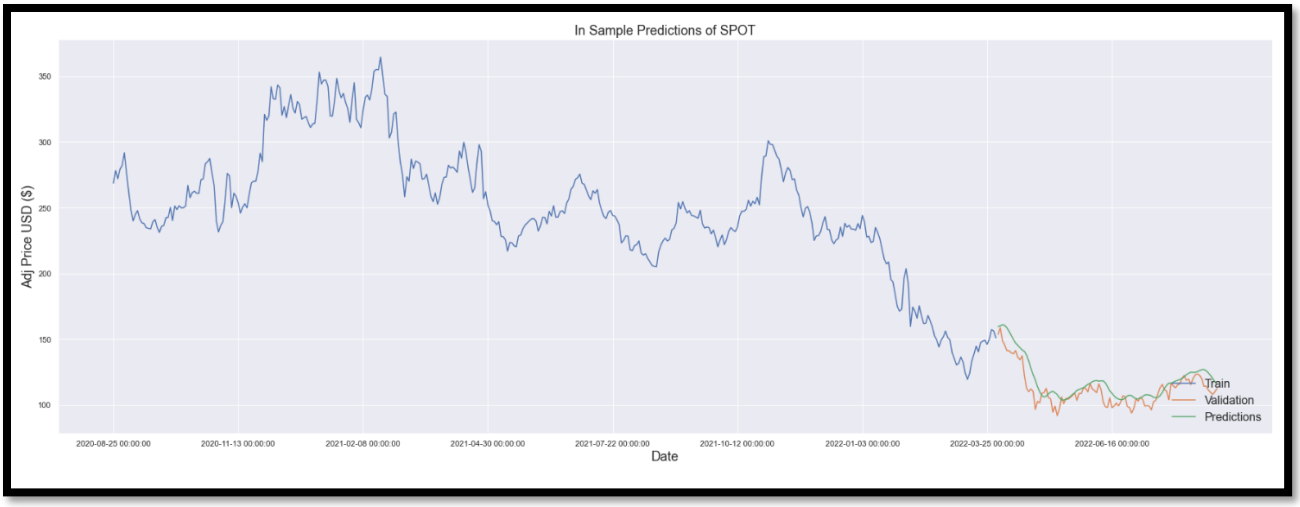


Figure 26: In sample Predictions for the SPOTIFY Stocks

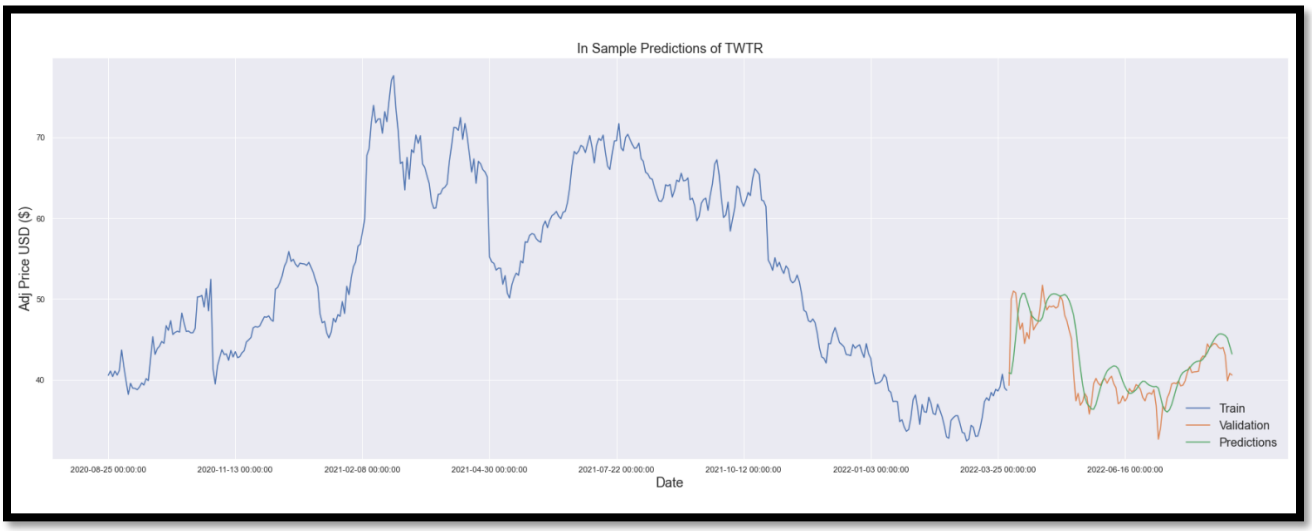


Figure 27: In sample Predictions for the TWITTER Stocks

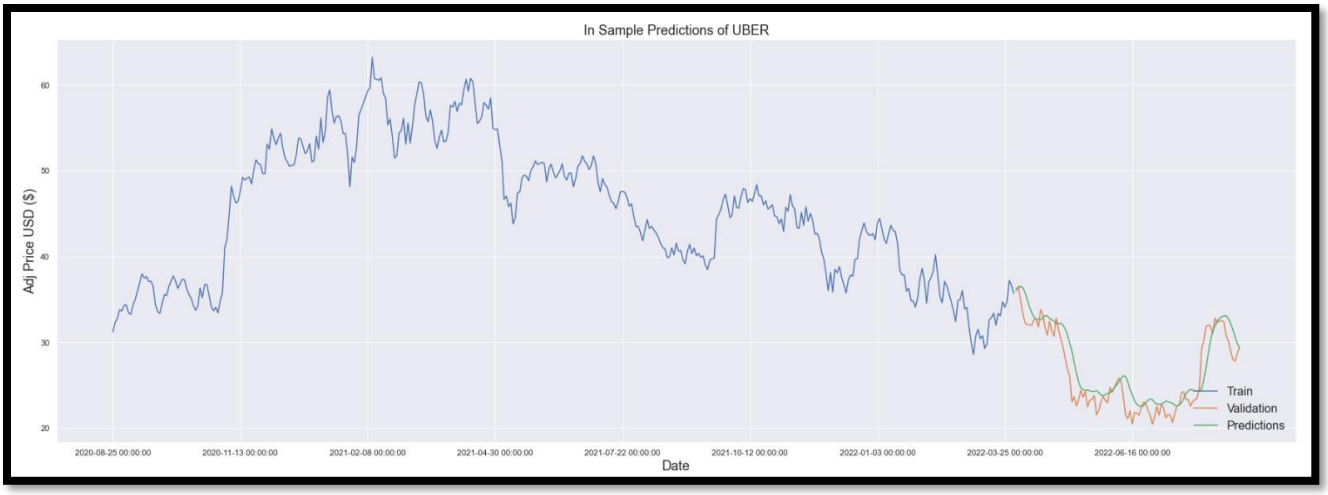


Figure 28: In sample Predictions for the UBER Stocks

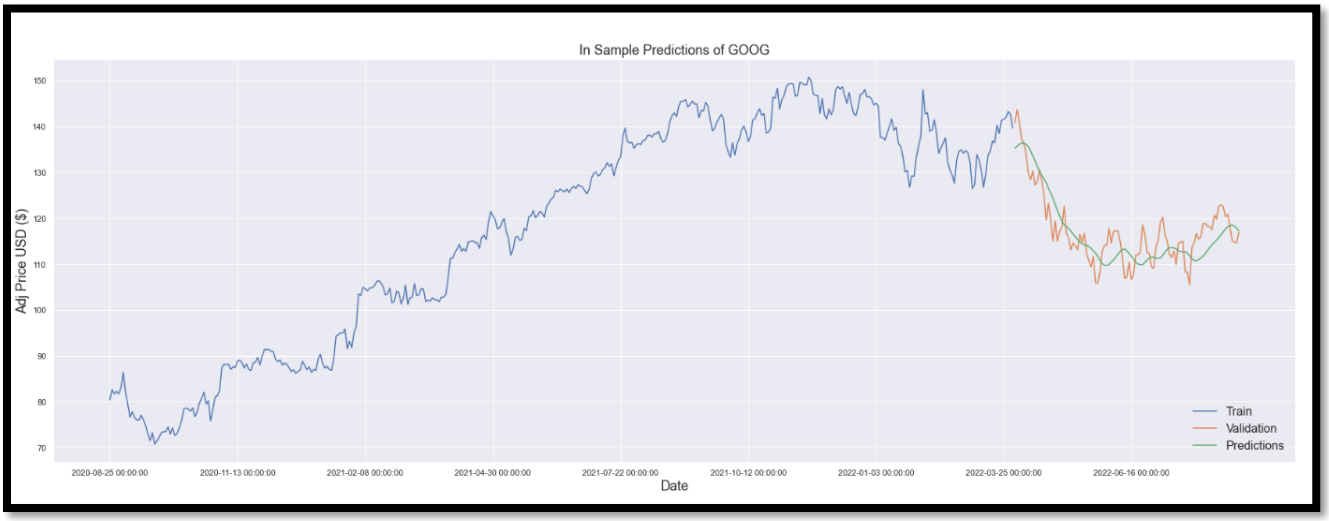


Figure 29: In sample Predictions for the GOOGLE Stocks

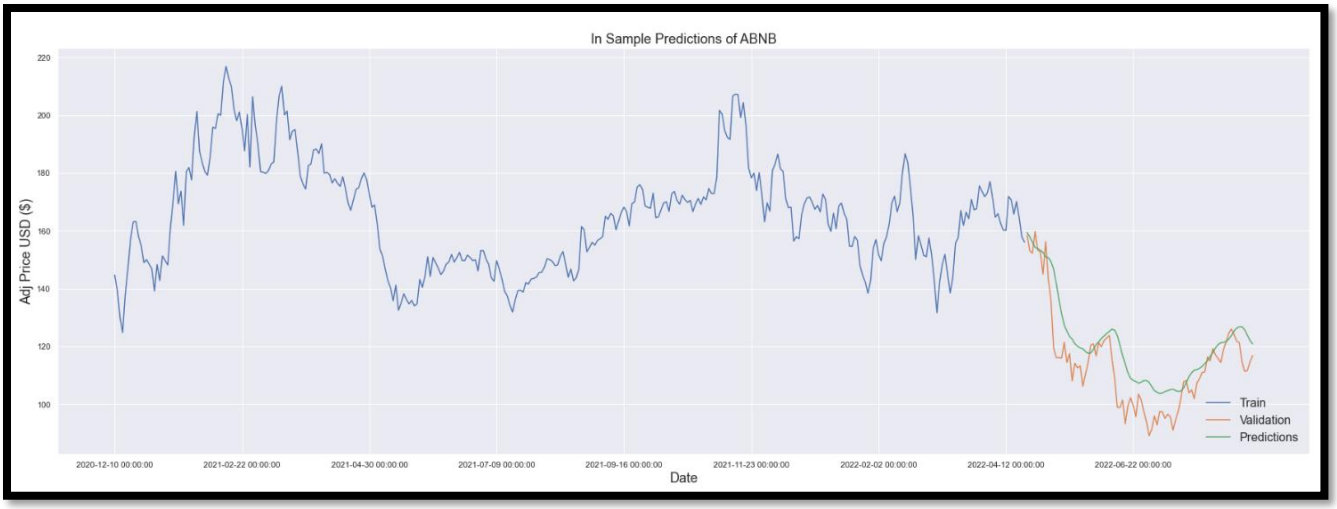


Figure 30: In sample Predictions for the AIRBNB Stocks

We can observe that the prediction line of every stock follows the trend of the real stock pretty accurately. The predictions are important not be exactly like the real stock values as in this case our LSTM model would be overfitted. We can say that the LSTM model seems to be very effective predicting sequential data (the trend) like the stock prices.

Out of sample predictions

In this part of the project, we will check the predictive abilities of our LSTM models in data that they have never seen before. Specifically, we created a function that takes as input the 60 last days of the stock prices and tries to predict the prices for the next 7 days (From 25/8/2022 to 1/9/2022) using every time the trained LSTM model of each stock.

DISCLAIMER: Since we are attempting to predict the future, there will be a great amount of uncertainty in the prediction and that is why we aim to predict the trend of the stock price and not the actual price of the stock in the future. We aim to inform our client about ONLY the trend of the stock in order to gain insights and decide if the stock is bullish or bearish.

Furthermore, we took the out of sample predictions and we concatenate them with the actual dataset in order to visualize them with the rest of the data. In figures 31 to 40 we can observe the trend of the price of each stock. The historical data about a stock's price is visualized with green colour and the 7-day prediction with red colour.

Figure 31: Out of Sample predictions for the APPLE Stock

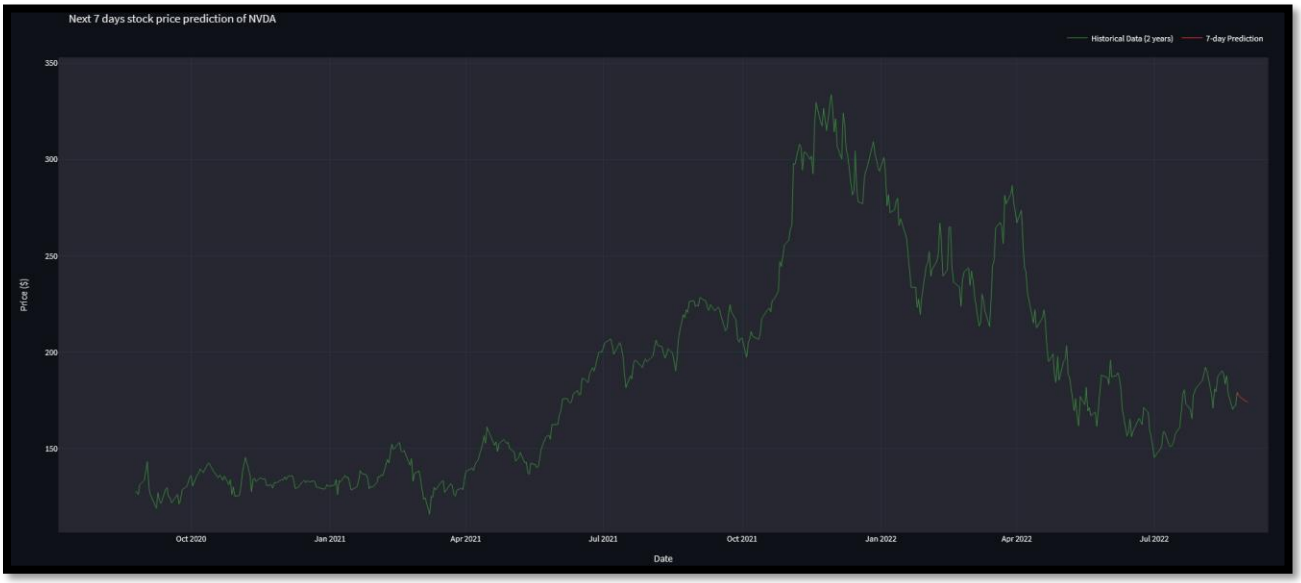


Figure 32: Out of sample predictions for the NVIDIA Stocks

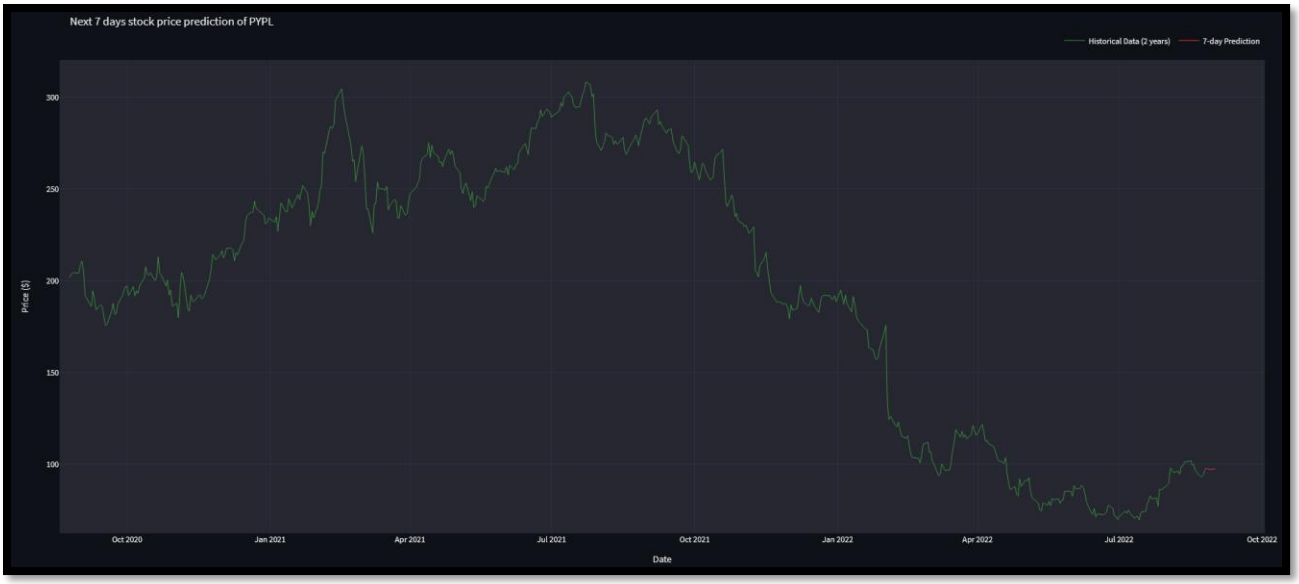


Figure 33: Out of sample predictions for the PAYPAL Stocks

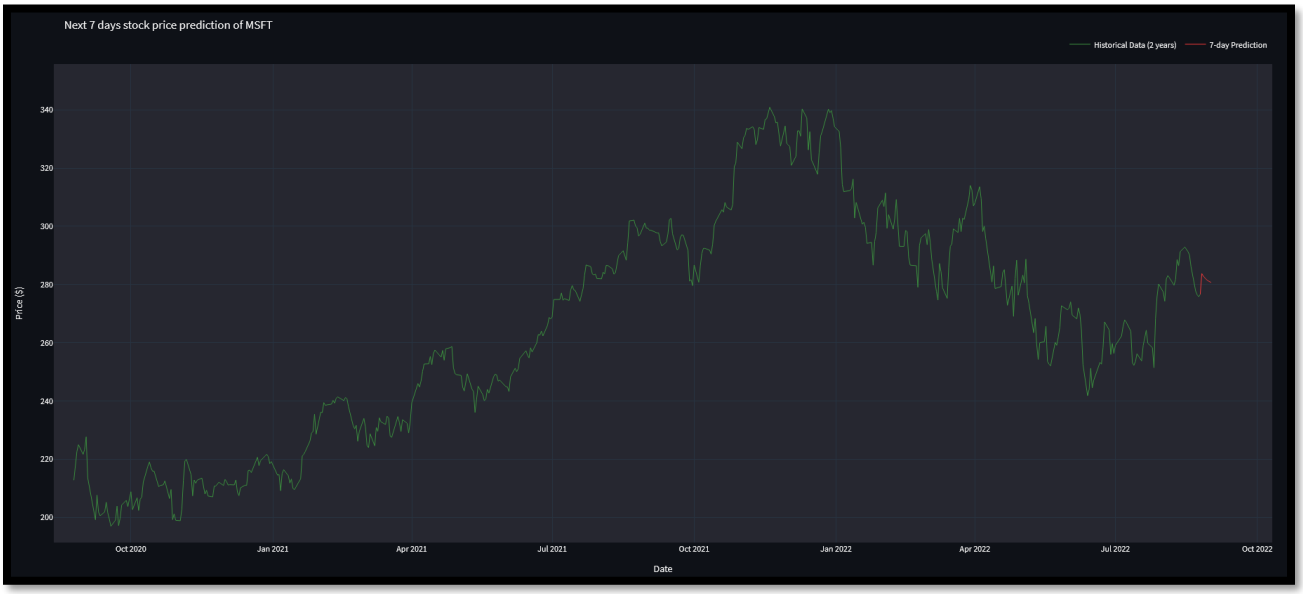


Figure 34: Out of sample predictions for the MICROSOFT Stock

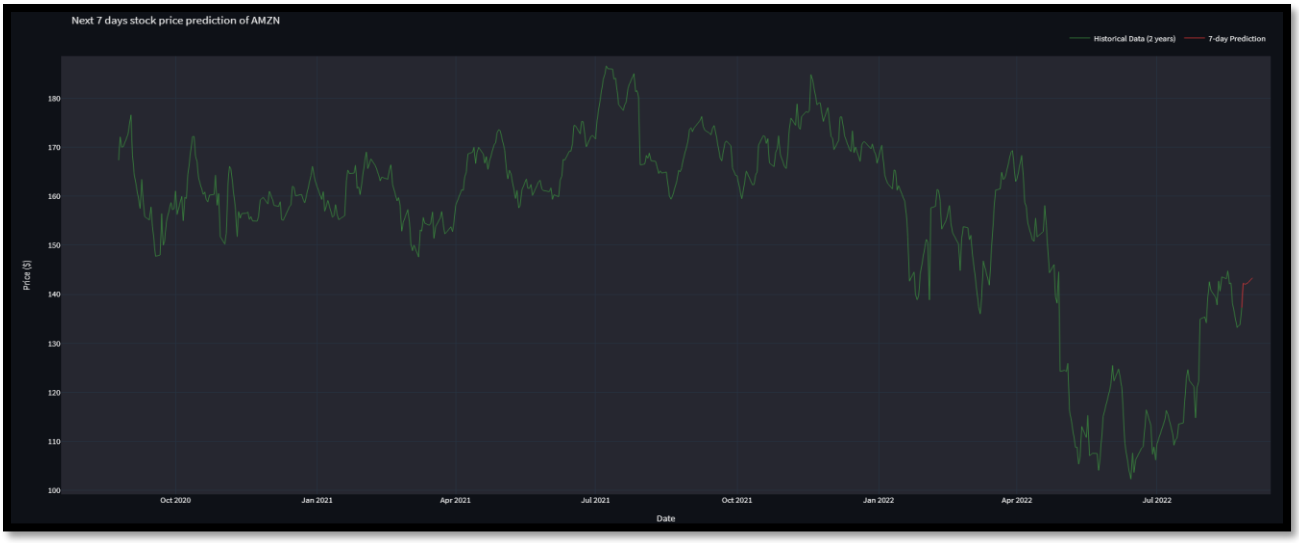


Figure 35: Out of sample predictions for the AMAZON Stocks

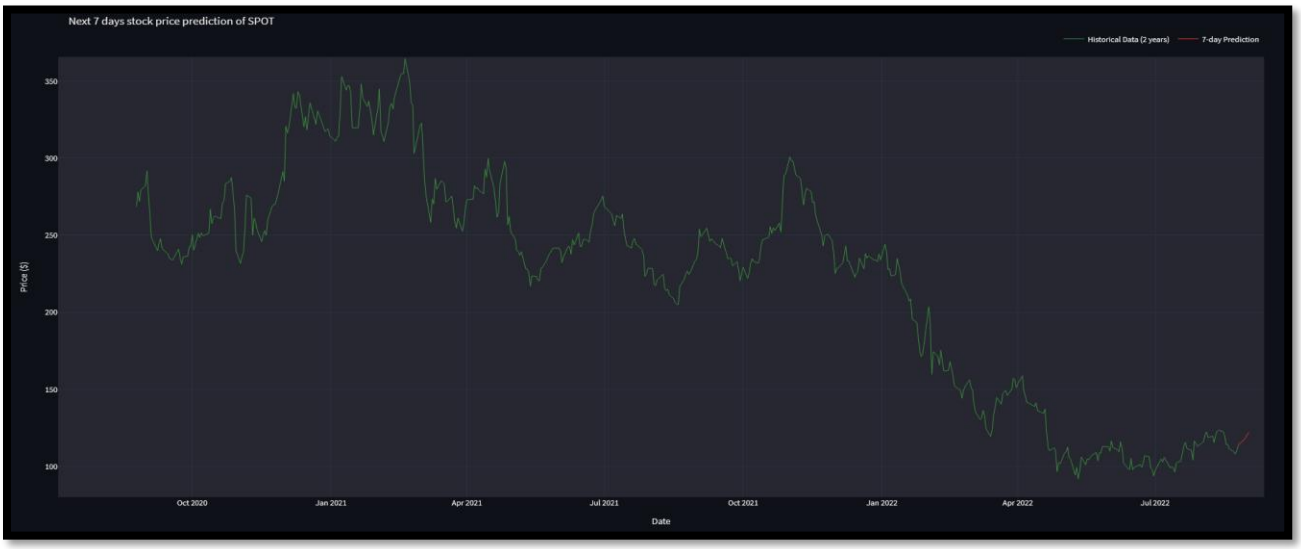


Figure 36: Out of sample predictions for the SPOTIFY Stocks

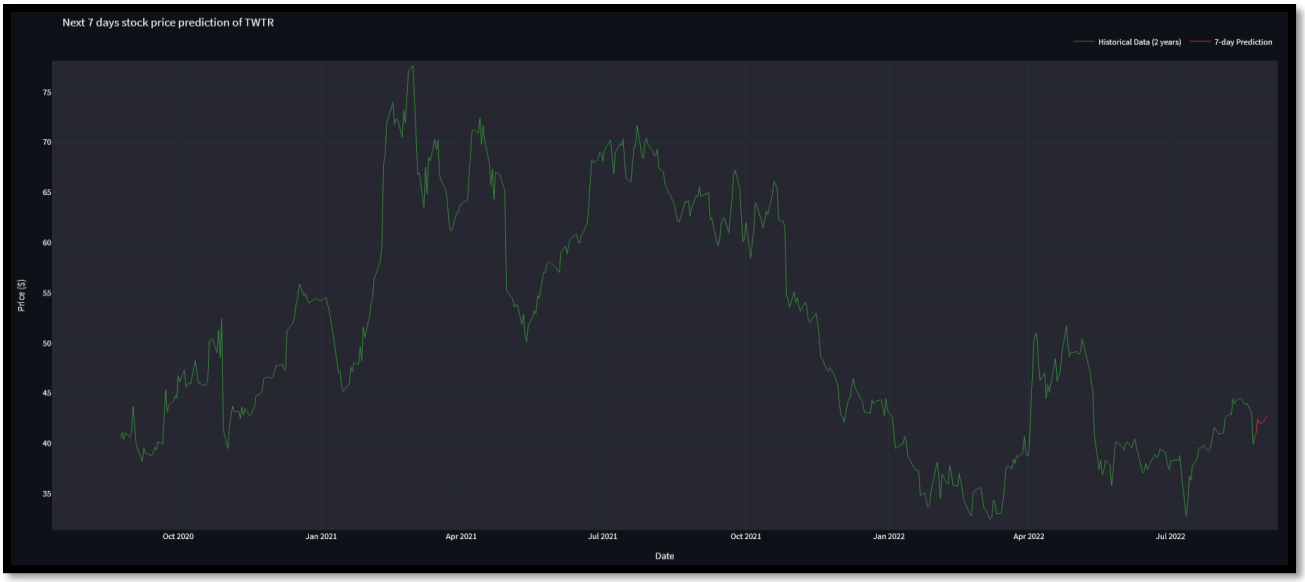


Figure 37: Out of sample predictions for the TWITTER Stocks

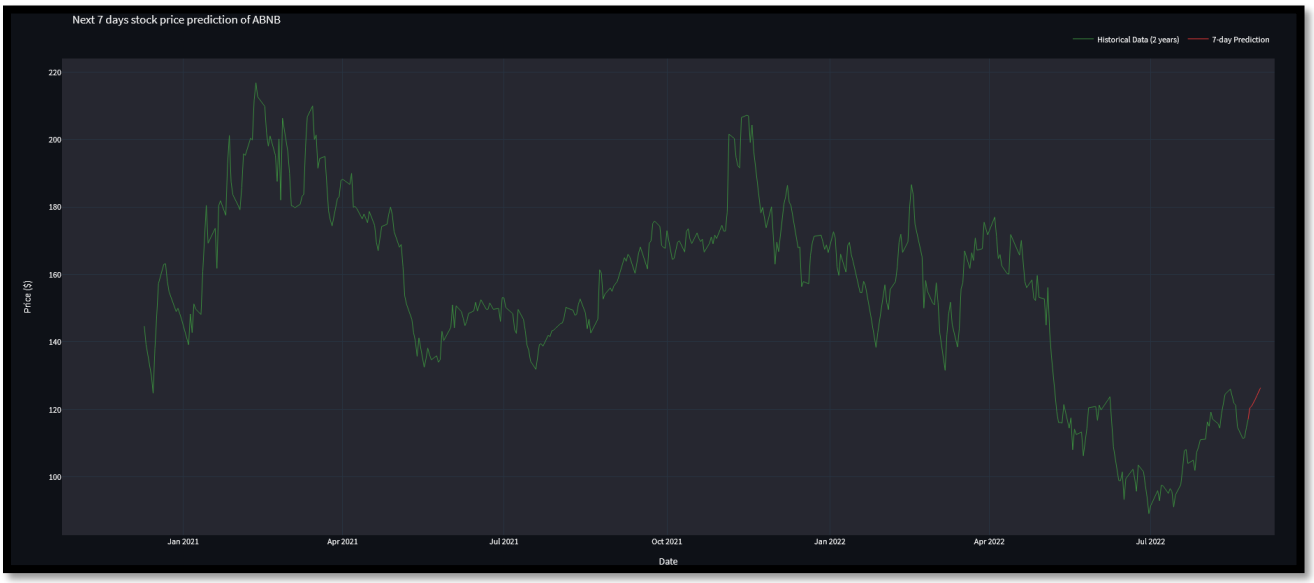


Figure 38: Out of sample predictions for the AIRBNB Stocks

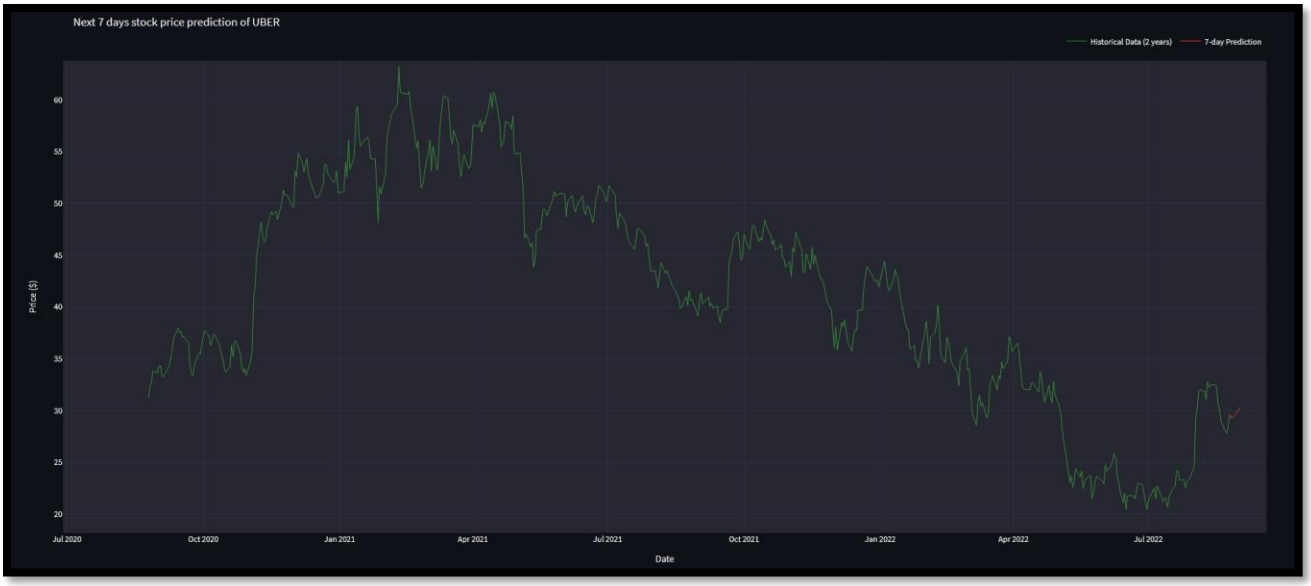


Figure 39: Out of sample predictions for the UBER Stocks

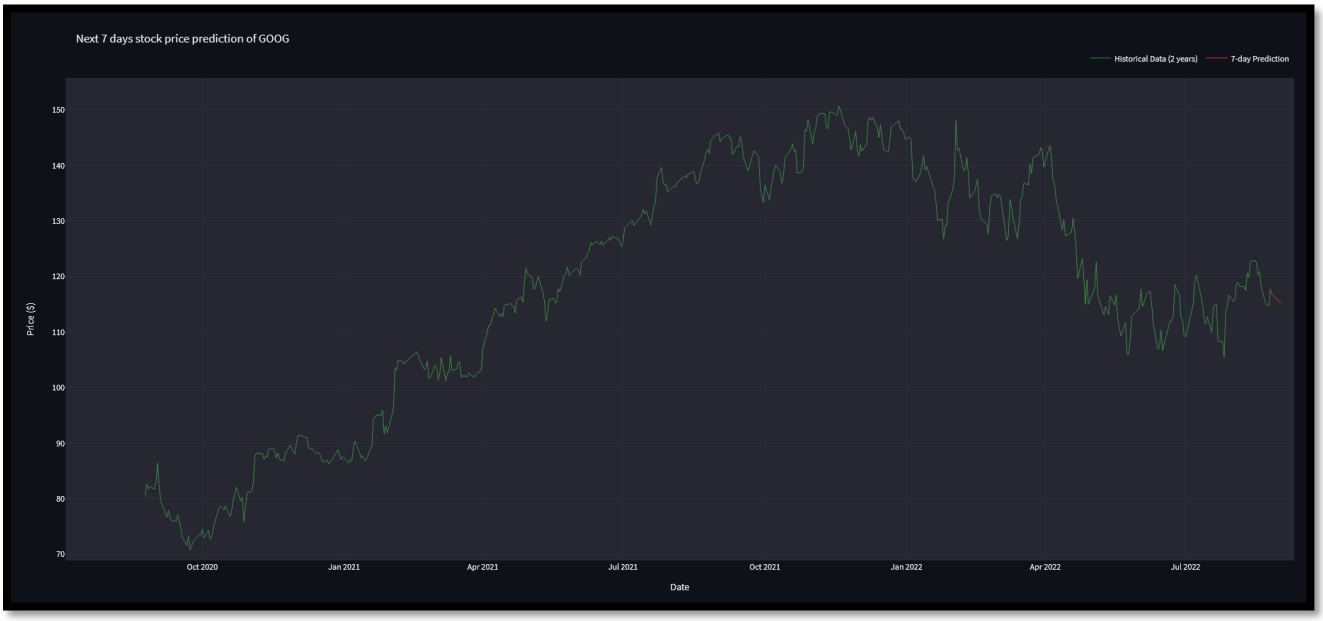


Figure 40: Out of sample predictions for the GOOGLE Stocks

By observing the Figures 31 - 40, we can have an idea of how the stock will be in the next 7 days. It is important to mention that we would advise our client for only the next 2 days as it is more accurate however the accuracy for 7-days prediction is not compromised a lot.

As we can see the stock price of the Uber, Airbnb, Amazon, Twitter, Spotify, and Apple will be increased so would advise our customer to sell the stocks. On the other hand, the stock prices of Google, PayPal, Nvidia and Microsoft will be decreased in the next 2 days so we would advise our customer to buy some of their stocks.

FinBERT Sentiment Analysis Model

To evaluate the model, we have to first tokenize the data that we will use with our tokenizer. Then, the predictions for each record are calculated and the maximum probability is extracted. The predictions are made within a pipeline which implements the tokenizer to the data and based on our trained model it calculates the probabilities of Positive, Negative and Neutral classes for each observation. To get the desired metrics we print the classification report for the chosen data split (test and validation). This report contains the precision, recall, f1-score, support, and accuracy metrics as well as the micro average and the weighted average of metrics as we have multiclass classification. Moreover, we construct the confusion matrix and the ROC curve.

In sample predictions

As we can see from Figure 41 the **accuracy** of our model in the test dataset is 87% which is quite good. From the **precision** metric we can see that 88% of Positive labels, 91% of Negative labels and 78% of Neutral labels were correct. The macro average of this metric is 86% while the weighted average is 87%. From the **recall** metric we can see that the 87% of all true Positive phrases, the 88% of all true Negative and the 85% of all Neutral phrases were found correctly. The macro average of this metric is 86% while the weighted average is 87%. **F1 score** is the harmonic mean between precision and recall. It is used to rate performance. In other words, an F1-score is a mean of performance based on precision and recall. As we can see the f1-score for our classes is 88%, 89% and 81% for Positive, Negative and Neutral classes, respectively. The macro average of this metric is 86% while the weighted average is 87%. Last but not least, **Support** is the number of actual occurrences of each class in our dataset. As we can see, there are 61 Positive phrases, 288 Negative and 136 Neutral.

Note that the Neutral phrases are the hardest to predict!

	precision	recall	f1-score	support
Positive	0.88	0.87	0.88	61
Negative	0.91	0.88	0.89	288
Neutral	0.78	0.85	0.81	136
accuracy			0.87	485
macro avg	0.86	0.86	0.86	485
weighted avg	0.87	0.87	0.87	485

Figure 41: In sample Classification Report for the FinBERT model

Apart from the classification report we create the confusion matrix for the test dataset. A confusion matrix is a table that is used to define the performance of a classification algorithm. Confusion matrices represent counts from predicted and actual values. Figure 41 presents the matrix for the in sample predictions.

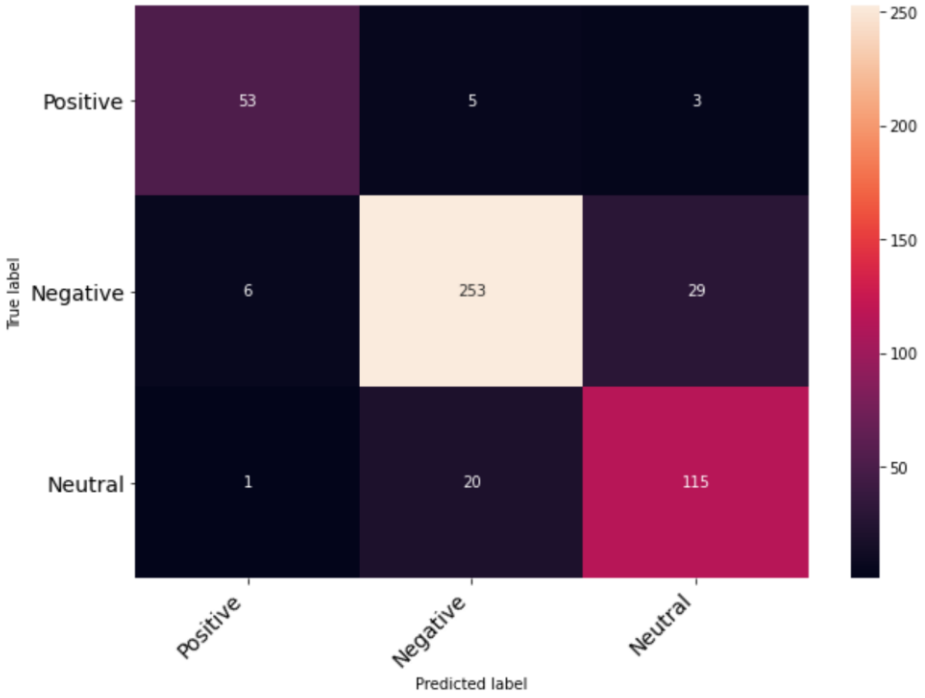


Figure 42: In sample FinBERT confusion matrix

As we can see from Figure 42, 53 out of 61 Positive classes were predicted correctly with the majority of mistakenly labelled Positive classes being labelled as Negative (5 of them). 253 out of 288 Negative classes were predicted correctly with the majority of mistakenly labelled Negative classes being labelled as Neutral (29 of them). Last but not least, 115 out of 136 Neutral classes were predicted correctly with the majority of mistakenly labelled Neutral classes being labelled as Negative (20 of them).

Finishing our evaluation in the test dataset we create the ROC curve. **ROC curve** (receiver operating characteristic curve) shows the performance of a classification model at all classification thresholds. This curve plots two parameters the True Positive Rate (TPR) and the False Positive Rate (FPR). The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). Figure 43 presents the ROC curve of our model for the in-sample predictions.

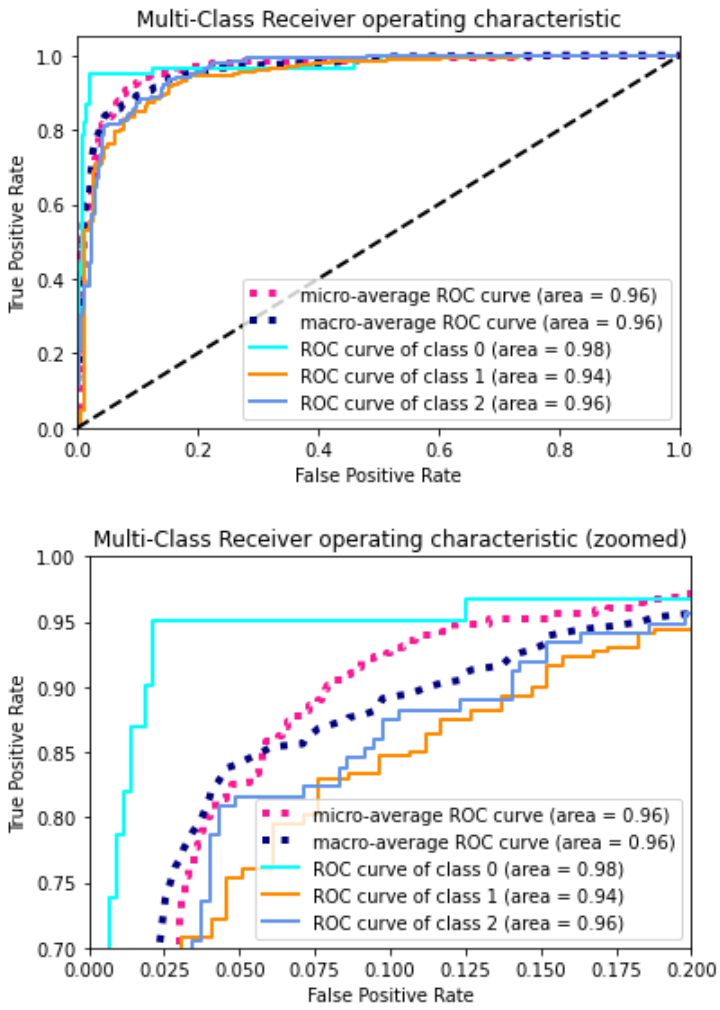


Figure 43: In sample predictions ROC curve for FinBERT model

The upper plot in Figure 43 shows the ROC curve for the test dataset while the lower one simply zooms to the top-left corner to study better each ROC - curve. In our case as we have a multi-class classification, we have 5 curves. Three for the classes (class 0: Positive class, class 1: Negative class and class 2: Neutral class) and two for the averages (macro and weighted average).

As we can see, the Positive class presents the best performance while the Neutral the worst. The overall performance of our model is quite good as all curves are pretty close to the top-left corner with the micro - average and weighted - average ROC curves True Positive Rates being close to ~ 97%.

Out of sample predictions

For the out of sample predictions, we will evaluate the model with the same metrics and methods. As we can see from Figure 44, the **accuracy** of our model in the validation dataset is 88% quite close to the accuracy of the test dataset which makes us believe that overfitting is not present. From the **precision** metric we can see that 85% of Positive labels, 92% of Negative labels and 82% of Neutral labels were correct. The macro average of this metric is 86% while the weighted average is 89%. From the **recall** metric we can see that the 92% of all true Positive phrases, the 89% of all true Negative and the 85% of all Neutral phrases were found correctly. The macro average of this metric is 89% while the weighted average is 88%. The **f1-score** for our classes is 88%, 91% and 84% for Positive, Negative and Neutral classes, respectively. The macro average of this metric is 88% as well as the weighted average too. In addition, we can see that there are 109 Positive phrases, 519 Negative and 245 Neutral.

	precision	recall	f1-score	support
Positive	0.85	0.92	0.88	109
Negative	0.92	0.89	0.91	519
Neutral	0.82	0.85	0.84	245
accuracy			0.88	873
macro avg	0.86	0.89	0.88	873
weighted avg	0.89	0.88	0.88	873

Figure 44: Out of sample Classification Report for the FinBERT model

Figure 45 presents the confusion matrix for the out of sample predictions.

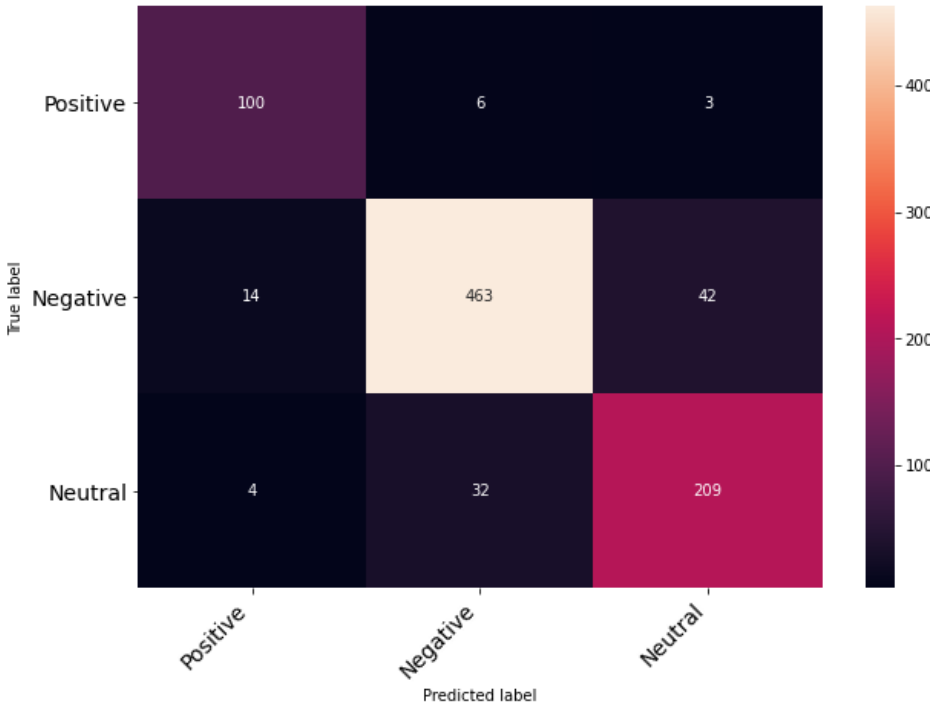


Figure 45: Out of sample FinBERT confusion matrix

As we can see from Figure 45, 100 out of 109 Positive classes were predicted correctly with the majority of mistakenly labelled Positive classes being labelled as Negative (6 of them). 463 out of 519 Negative classes were predicted correctly with the majority of mistakenly labelled Negative classes being labelled as Neutral (42 of them). Last but not least, 209 out of 245 Neutral classes were predicted correctly with the majority of mistakenly labelled Neutral classes being labelled as Negative (32 of them).

Finishing our evaluation in the out of sample predictions we create the ROC curve. As we can see from Figure 46, the Positive class presents the best performance while the Negative this time the worst. The overall performance of our model is quite good as all curves are pretty close to the top-left corner with the micro - average and weighted- average ROC curves True Positive Rates being really close to 97%.

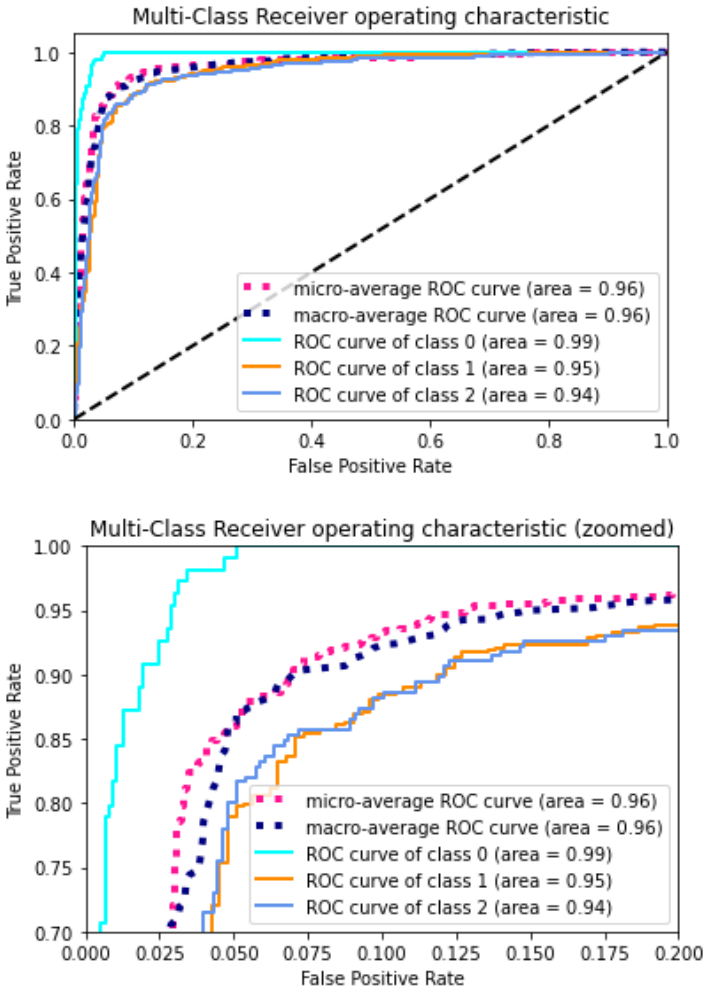


Figure 46: Out of sample predictions ROC curve for FinBERT model

LSTM Error Analysis

Because we have a regression problem the absolute metric, we can rely on to measure the error of the predictions is the Root Mean Square Error. Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Because our models have been trained in real time data, we expect to have bigger error that usual as the stock prices are usually affected by a lot of factors. The most common factor is the **Split Procedure** that a lot of companies apply in order to make their stocks affordable for everyone. When this happens the prices are dramatically reduced which may cause problems to the LSTM models which are built for forecasting in general. As we mentioned before we are training the models with real time data , so we cannot expect low RMSE Scores (< 0). The following table presents the RMSE scores for each stock model.

Stock	RMSE Score
APPLE (AAPL)	***5.483***
NVDIA (NVDA)	***12.945***
PAYPAL (PYPL)	***6.056***
MICROSOFT (MSFT)	***10.469***
AMAZON (AMZN)	***11.236***
SPOTIFY (SPOT)	***9.124***
TWITTER (TWTR)	***2.811***
UBER (UBER)	***2.069***
GOOGLE (GOOG)	***4.384***
AIRBNB (ABNB)	***9.956***

Moreover, in Figures 47 – 56 the model loss in the train set and the test of every stock can be observed.

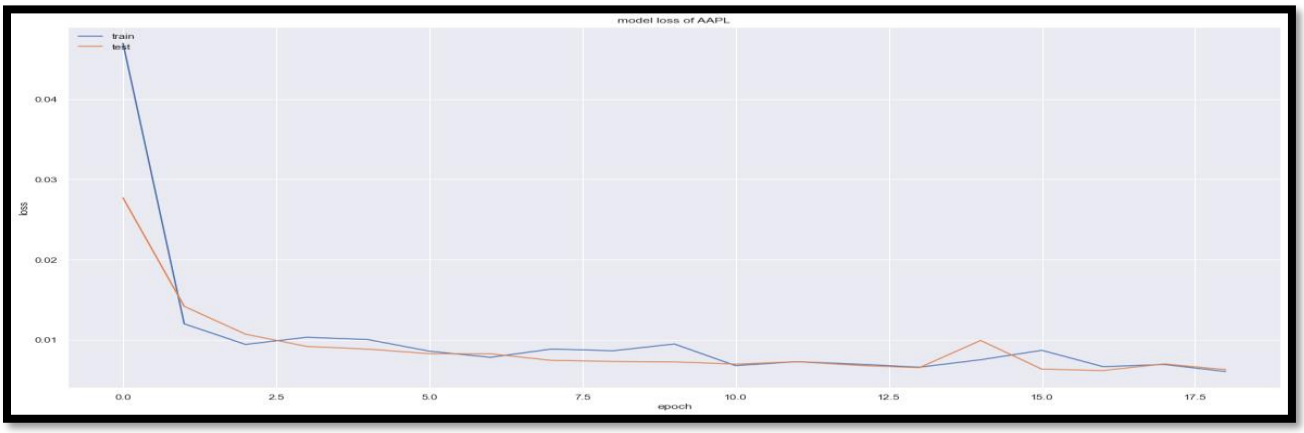


Figure 47: Apple Model Loss

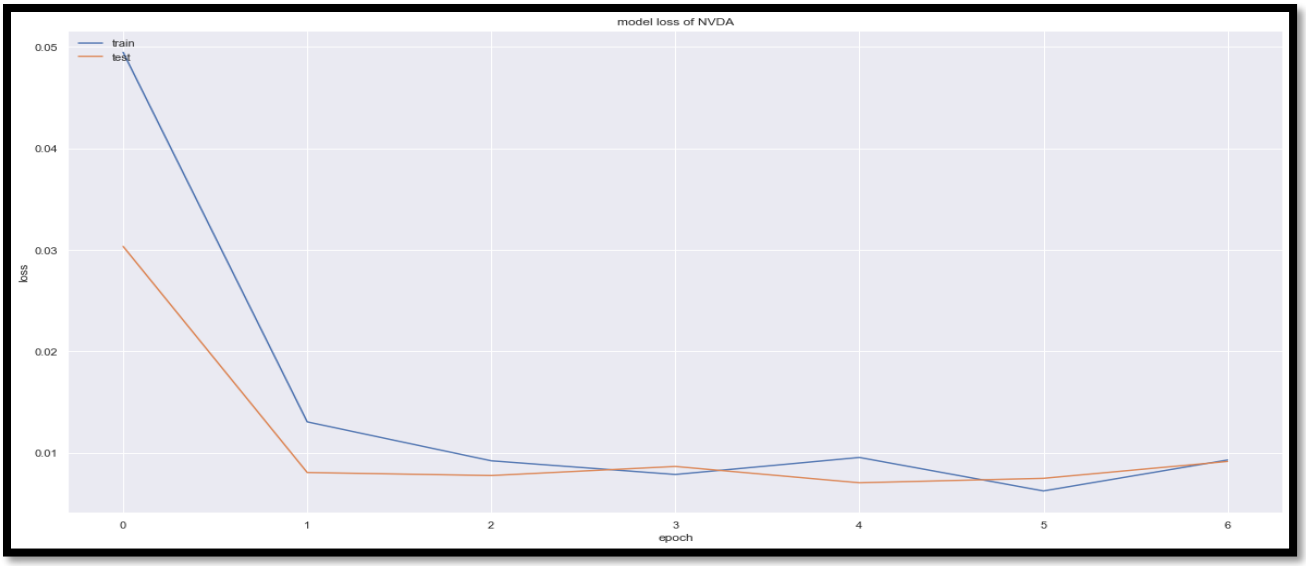


Figure 48: Nvidia Model Loss

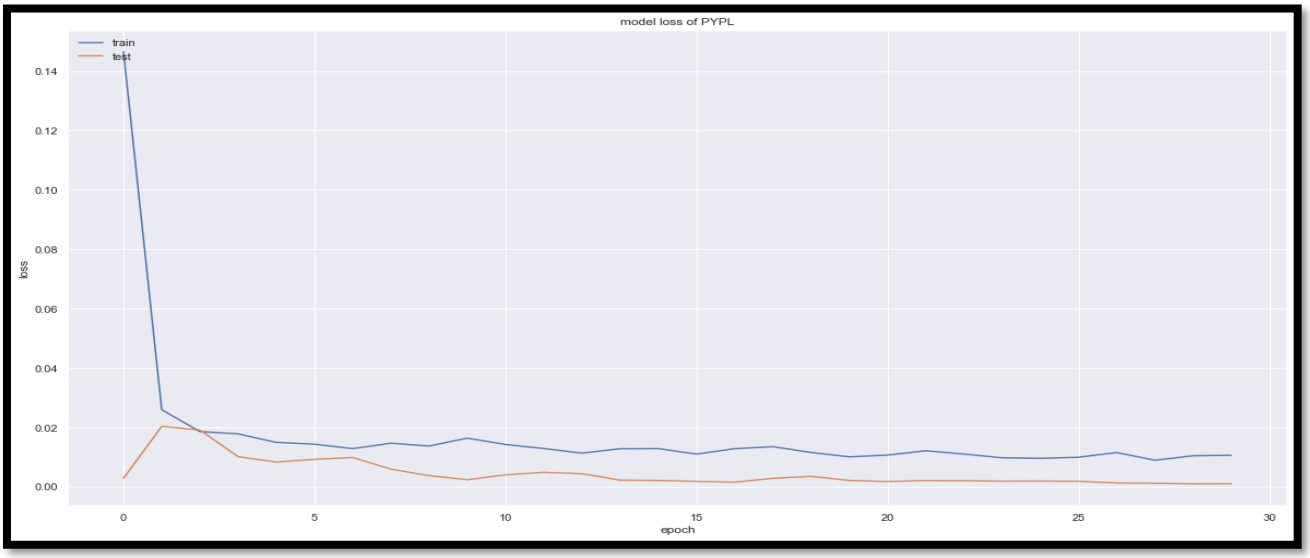


Figure 49: PayPal Model Loss

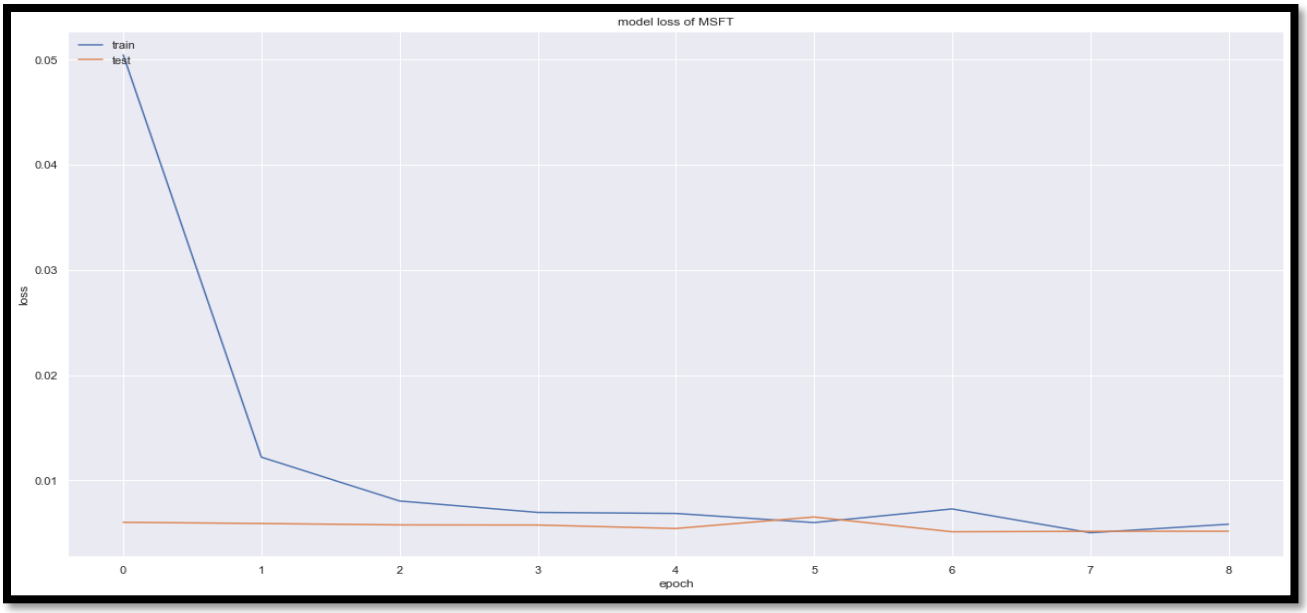


Figure 50: Microsoft Model Loss

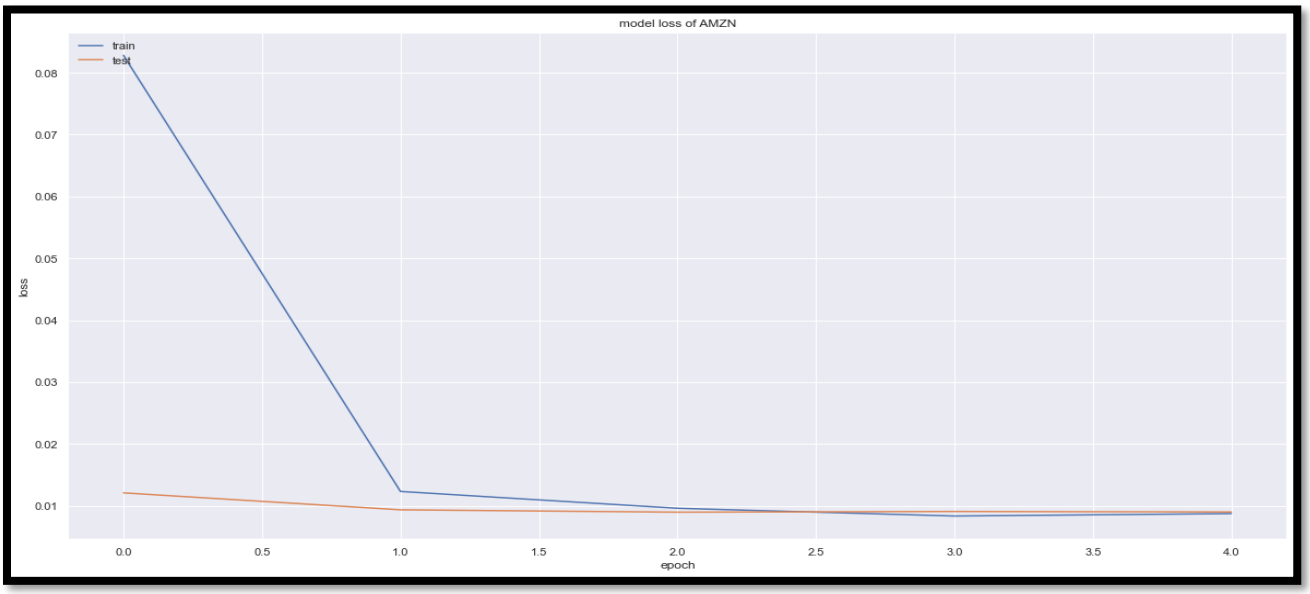


Figure 51: Amazon Model Loss

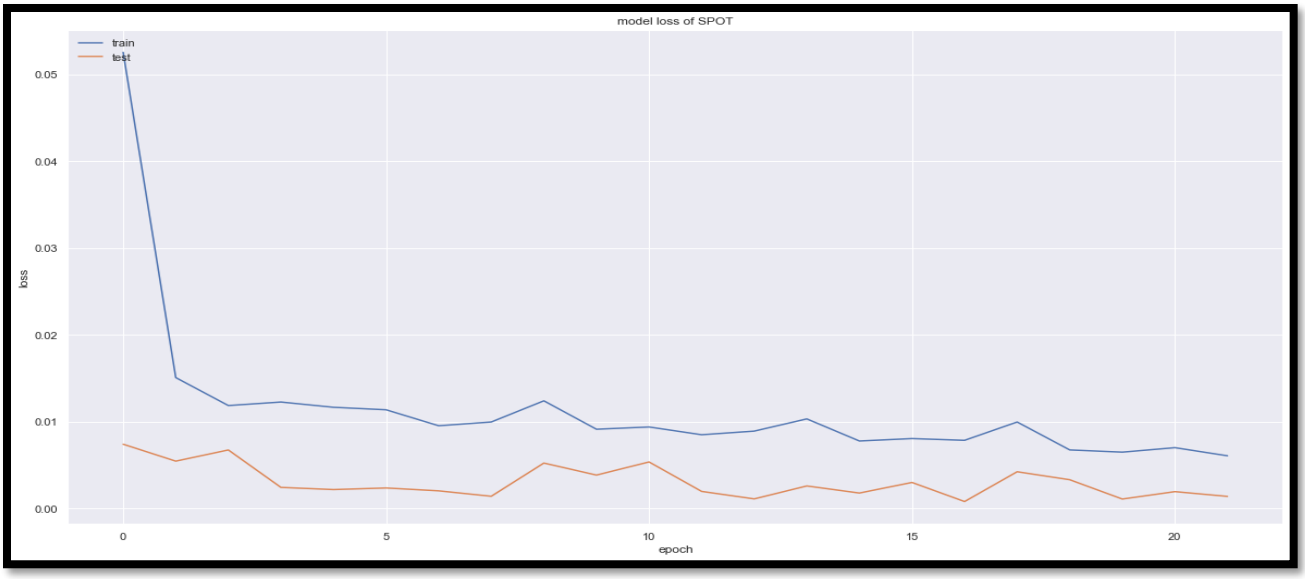


Figure 52: Spotify Model Loss

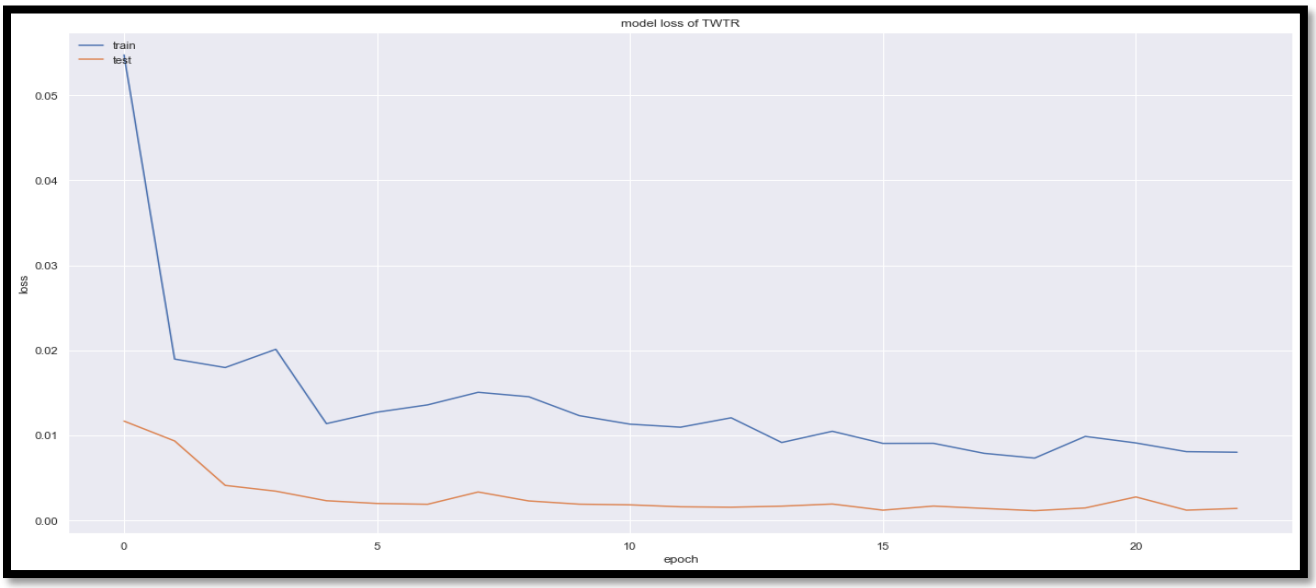


Figure 53: Twitter Model Loss

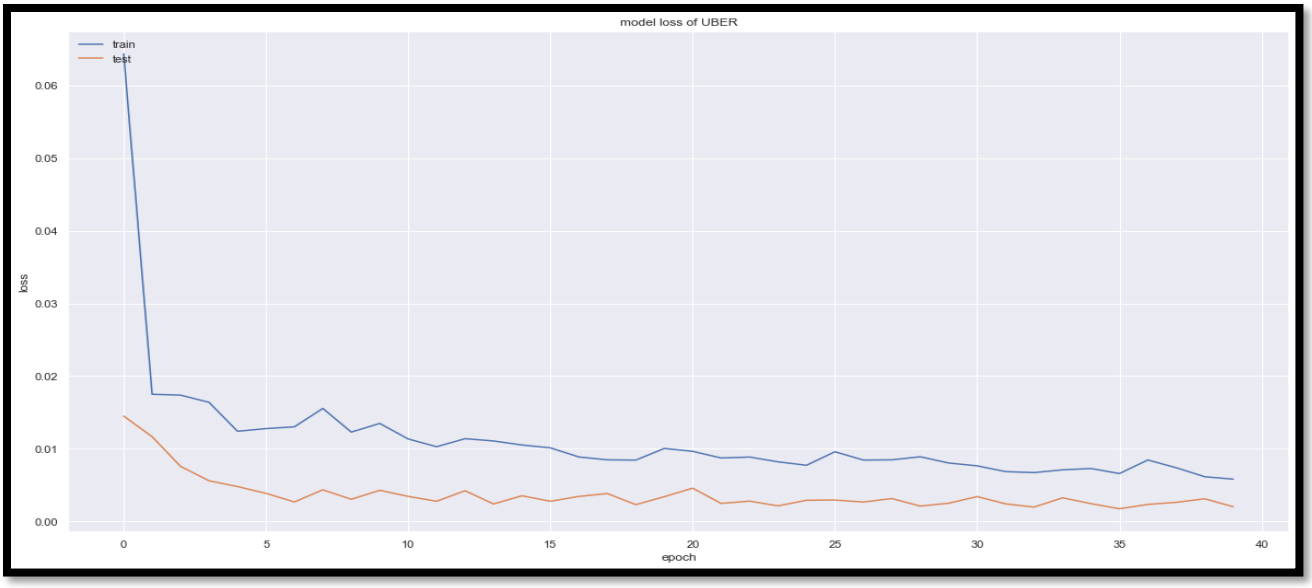


Figure 54: Uber Model Loss

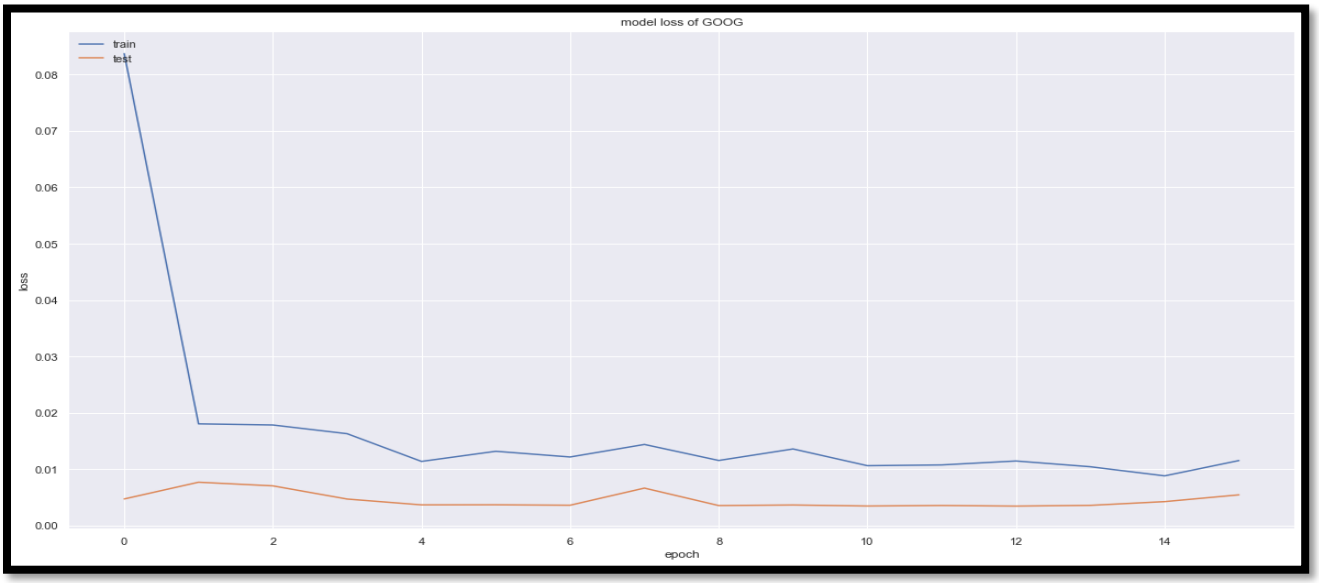


Figure 55: Google Model Loss

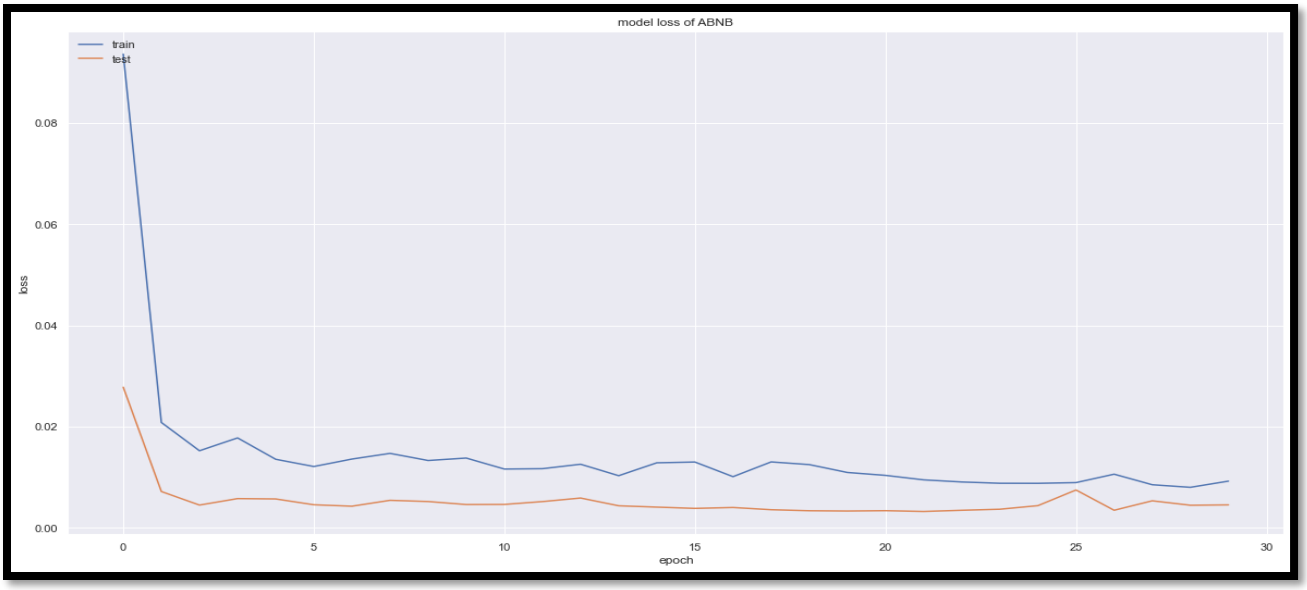


Figure 56: Airbnb Model Loss

Discussion, Comments, Notes and Future Work

Comments

1) LSTM neural networks are great tools for stock price prediction, but it is important to mention that they should not be used as a guide to make a decision about a future investment without further analysis. This is because the prediction is only based on past historical data that in a lot of cases are not the only factor that determines the future price of the stock. A good example is the split technique that a lot of companies do to their stocks. Companies often split shares of their stock to make them more affordable to investors. Also, a lot of factors should be under consideration of the client like :
   - News releases on earnings and profits, and future estimated earnings.
   - Announcement of dividend.
   - Introduction of a new product or a product recall.

   That is the reason why a further fundamental – market analysis is required to support the investment decision making.

2) In our future prediction of the stock prices, we choose to predict only the 7 next days. That is why we considered that the outcome of the predictions would be more plausible and more accurate rather that predicting for the next 30 days. Another reason was that our model will serve informational purposes.
3) In the acquisition of the LSTM data, we used only data from the last 2 years. That was purposeful as we wanted to include as little data as possible that was affected by covid , as during the pandemic the decreasing of the stock prices was unprecedented.
4) The reason why we choose the Adjusted Close Price was that it is the closing price after adjustments for all applicable splits and dividend distributions. Data is adjusted using

appropriate split and dividend multipliers, adhering to Center for Research in Security Prices (CRSP) standards

5) The report has been written on 25/8/2022 that means that the outcomes that we present may be slightly changed due to the new data that will have entered when the assignment will be graded.
6) But to cover all possible scenarios , in the **stocks.py** file in **line 97** we have commented 2 extra lines of code that give freedom to the user to predict the price of the stock from the specific date up to 7 days (By default it is uncommented). For example, if the script will be runned on 1st of the September the script will collect data up to this date and predict the future stock prices up to 8th of September. On the other hand, we give the opportunity to the user to see the evaluation of the model by uncommented the lines referred below the : ***Prediction in the data we evaluate the model***.
7) For the LSTM we did not save any weights of the models as we want a real time prediction of the data. The execution time for every stock is about 30 seconds.
8) It is essential to mention that in every decision the customer makes, the score of R – Squared and the Root Mean Square Error should be considered (not to a great extent).
9) Finally, another good idea would be to scrape Headlines from more sources (Reuters, Financial Times etc.) and use them in the Sentiment Analysis Model. Moreover, in the near future we could modify our code and use more than 17 headlines for the sentiment analysis if we have more sources.

Future Work

- First of all, a very needed add on for the assistant would be to be able to predict more stocks. In our case we used the most well-known stocks that we have an idea about their trend of their price via news, web etc. With the proper knowledge for the majority of the stocks we would be able to tune the model and predict a much wider variety of stocks.
- Secondly, with the proper education and knowledge about this field we would be able to build a model that can take into account a larger amount of factors that can affect the price of the stock such as :
    - Global Phenomena
    - The Open Stock Price
    - The Volume
    - Exchange Rate Fluctuations
    - Interest Rates

  Also, that would allow us to use even more data to train our models.

- Thirdly, additionally with the second with the proper knowledge we could predict the trend of the stock price for a longer period of days without any misclassifications.
- Finally, a grid search procedure could be applied in our model in future in order to be optimized every day with the absolute best parameters and return even more accurate outcomes.

Members and Roles

The entire project has been implemented by :

- Ilias Dimos (f2822102)
- Charilaos Petrakogiannis (f2822112)
- Ioannis Triantafyllakis (f2822115)
- Nikolaos Tsekas (f2822116)

Ioannis Triantafyllakis was the leader of the project as he had a very wide knowledge about stocks and was able to guide us in making accurate decisions in the implementation of the Neural Network Models.

We all contributed to both the creation of Bert and the creation of LSTM model as we were exchanging ideas for their creation. But in order to carry out the project faster we splited responsibilities. In more details Ilias Dimos , Charilaos Petrakogiannis were assigned in the creation of the Long Sort Term Memory model (LSTM) and Ioannis Triantafyllakis , Nikolaos Tsekas in the creation of the FinBERT model. When the creation of the models had been completed , we all contributed to the creation of the web application (stream lit) in order to provide our customers with an interactive environment where they can observe the trend of each stock and its sentiment analysis.

Time Plan

Figure 57, on the next page presents the time plan of activities that were followed for this project.

Stage 1 contains the Brainstorming and the Idea Generation. In this phase we decided the final topic of the project and started talking about our vision and goals. The duration of Stage 1 was approximately 1 week. With the final idea and vision at hand, we proceeded to **Stage 2** which contains the research and the requirements setting for the project. For about 2 weeks we searched the internet for relevant trading services, papers about Financial Sentiment Analysis and Price Prediction and we also studied the class notes and notebooks to familiarize with Machine Learning and Neural Networks. In this step, we also decided how we want to be our App and what it should contain (Technical Analysis Tools, Streamlit Dashboard, User Friendly UI/UX). After, finishing this phase we were ready to proceed to **Stage 3** which is nothing else but Data Preparation. In this stage for about 1 week, we searched for quality data, and we prepared them for our models. With the data ready, we started creating our models (**Stage 4**). Stage 4 and **Stage 5** (Training and Models Evaluation) lasted for more than 4 weeks as this was the most important phase of our project. In these steps we found the best models and we concluded how we will present the outcomes of each model and tool. At the end of our project, we constructed the Streamlit Dashboard (**Stage 6**) which is essentially the model deployment (model saving, model loading) in a way that each trader will be able to make use of the produced insights. The model deployment lasted about 1 week. The overall duration of the project was more than 2 months.

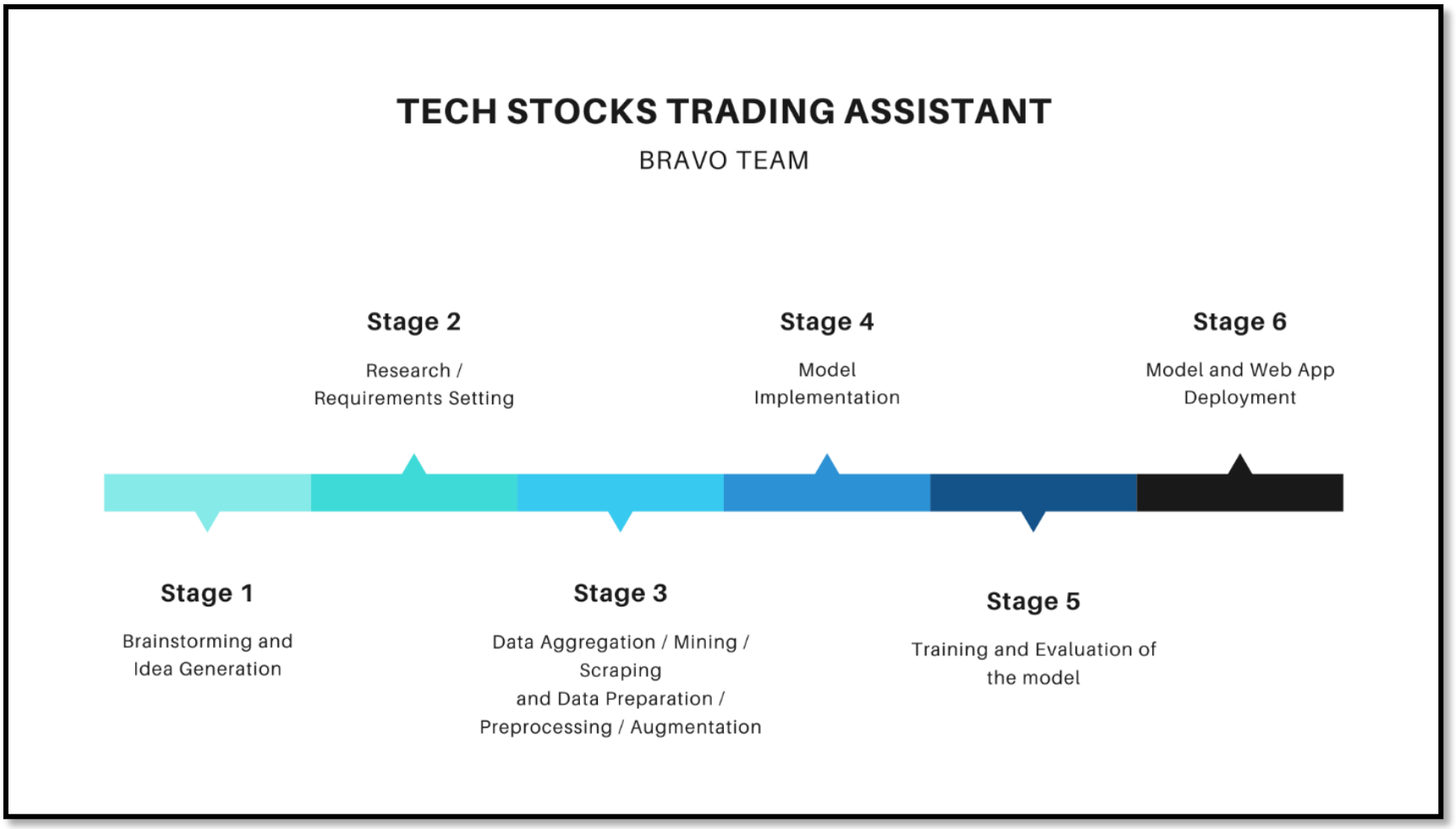


Figure 57: Project Time Plan

Bibliography

Class Notes and Jupyter Notebooks: Machine Learning and Content Analytics. (2022). MSc in Business Analytics Educational Portal. https://e-mscba.dmst.aueb.gr/

Stack Overflow - Where Developers Learn, Share, & Build Careers. (2022). Stack Overflow. https://stackoverflow.com/

Hugging Face - Documentation. (2022). HuggingFace. https://huggingface.co/docs

Dolphin, R. (2022). LSTM Networks | A Detailed Explanation - Towards Data Science. Medium. https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

Solegaonkar, V. (2022). Error Analysis in Neural Networks - Towards Data Science. Medium. https://towardsdatascience.com/error-analysis-in-neural-networks-6b0785858845

Chowdhury, K. (2022). 10 Hyperparameters to keep an eye on for your LSTM model — and other tips. Medium. https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4

Vijay, U. (2021). Early Stopping to avoid overfitting in neural network- Keras. Medium. https://medium.com/zero-equals-false/early-stopping-to-avoid-overfitting-in-neural-network-keras-b68c96ed05d9

Yi Yang, Mark Christopher Siy UY, Allen Huang (2020). FinBERT: A Pretrained Language Model for Financial Communications.

Dogu Araci (2019). FinBERT: Financial Sentiment Analysis with Pre-trained Language Models

Appendix

Tech Stocks Trading Assistant

Technical Analysis Methods

Choose a stock:

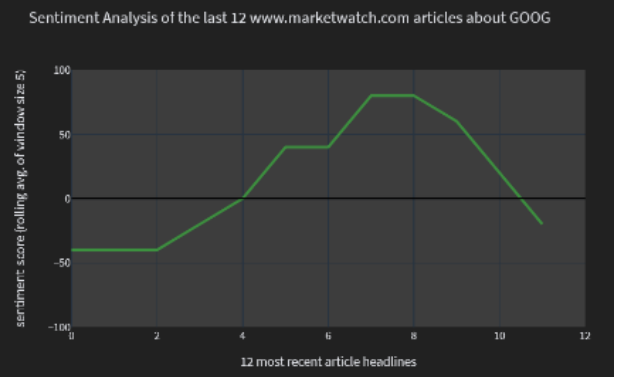
Google

You selected: Google (GOOG)

Last execution: 2022-08-26 17:46:37.102936

	Technical Analysis Method	Outlook	Timeframe of Method
0	Bollinger Bands (10 days & 1.5 stand. deviations)	Buy	Short-term
1	Moving Average Convergence Divergence (MACD)	Downtrend and no signal	Short-term
2	Bollinger Bands (20 days & 2 stand. deviations)	Buy	Medium-term
3	Bollinger Bands (50 days & 3 stand. deviations)	Buy	Long-term

FinBERT-based Sentiment Analysis



Current sentiment: -20.0 %

LSTM-based 7-day stock price prediction model