



MANIPAL INSTITUTE OF TECHNOLOGY
(A constituent Institute of Manipal Academy of Higher Education)
MANIPAL - 576 104, KARNATAKA, INDIA

June 2020

Submission Report

STONKS
STOCK MARKET PREDICTION
USING STATISTICAL AND SENTIMENT ANALYSIS

By

Devdatta Joshi (170953070)
Prakriti Parihar (170953110)
Utkarsh Chhapekar (170953106)
Kunal Khanwalkar (170953162)

Under the Guidance of
Mrs. Jayashree, Assistant Professor
Dr. Raghavendra Achar, Associate Professor
Department of Information and Communication Technology, MIT, MAHE

1. STATISTICAL ANALYSIS

Initially to get current stock data we run `GetCurrentData.py`, which uses the `yfinance` API to get the stock data for the past 6 years till the present.:

```
no changes added to commit (use "git add" and/or "git commit -a")
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$ python3 GetCurrentData.py
/mnt/d/Codes/venv/lib/python3.6/site-packages/pandas_datareader/compat/__init__.py:7: FutureWarning: pandas.util.testing
is deprecated. Use the functions in the public API at pandas.testing instead.
  from pandas.util.testing import assert_frame_equal
AAPL
[ *****100%***** ] 1 of 1 completed
GOOGL
[ *****100%***** ] 1 of 1 completed
FB
[ *****100%***** ] 1 of 1 completed
AMZN
[ *****100%***** ] 1 of 1 completed
MSFT
[ *****100%***** ] 1 of 1 completed
T
[ *****100%***** ] 1 of 1 completed
INTC
[ *****100%***** ] 1 of 1 completed
WMT
[ *****100%***** ] 1 of 1 completed
GM
[ *****100%***** ] 1 of 1 completed
JPM
[ *****100%***** ] 1 of 1 completed
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
```

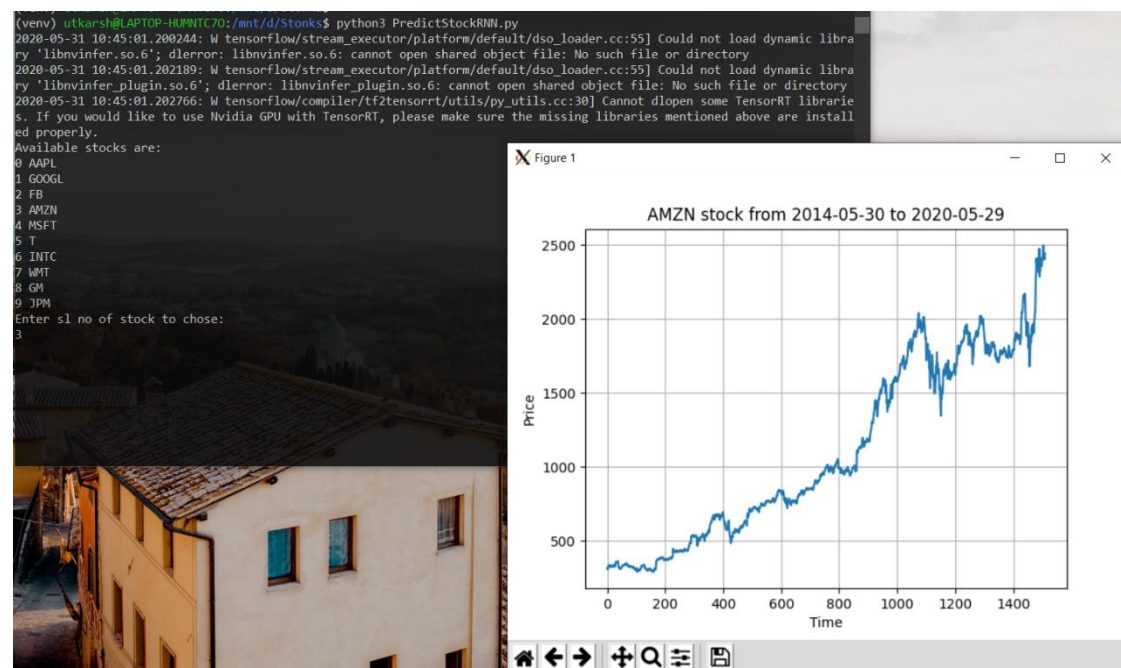
It saves this to the data folder.

To calculate correlation using Pearson's coefficient, we run `CalcCorrelation.py`. It gives us this value on the secondary diagonal. For this there is a primary stock (Amazon) to input and calculates the correlation with respect to the other stocks:

```
[ *****100%***** ] 1 of 1 completed
WMT
[ *****100%***** ] 1 of 1 completed
GM
[ *****100%***** ] 1 of 1 completed
JPM
[ *****100%***** ] 1 of 1 completed
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$ python3 CalcCorrelation.py
Enter the primary stock
AMZN
Enter one or more stock separated by space to be compared with
AAPL GOOGL FB MSFT GM JPM T
[[1.          0.51400635]
 [0.51400635  1.          ]
 [1.          0.64876866]
 [0.64876866  1.          ]
 [1.          0.56815931]
 [0.56815931  1.          ]
 [1.          0.61148441]
 [0.61148441  1.          ]
 [1.          0.29355197]
 [0.29355197  1.          ]
 [1.          0.3536493 ]
 [0.3536493  1.          ]
 [1.          0.265067 ]
 [0.265067  1.          ]]
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
(venv) utkarsh@LAPTOP-HUMNTC70:/mnt/d/Stonks$
```

This data will be used later in the program when we need to find highly correlated stocks and calculate the sentiment of those. So overall, the predictions of those correlated stocks will add weight to the final prediction of the current stock chosen.

Now for the RNN prediction we run PredictStockRNN.py. Here we choose the stock that needs to be predicted (Amazon) and we can see the graph as an output:



Now the model is trained for 100 epochs on the current data to find an optimum learning rate. The model uses a convolutional layer, two layers of LSTM and a few dense layers to give the prediction:

Model: "sequential"

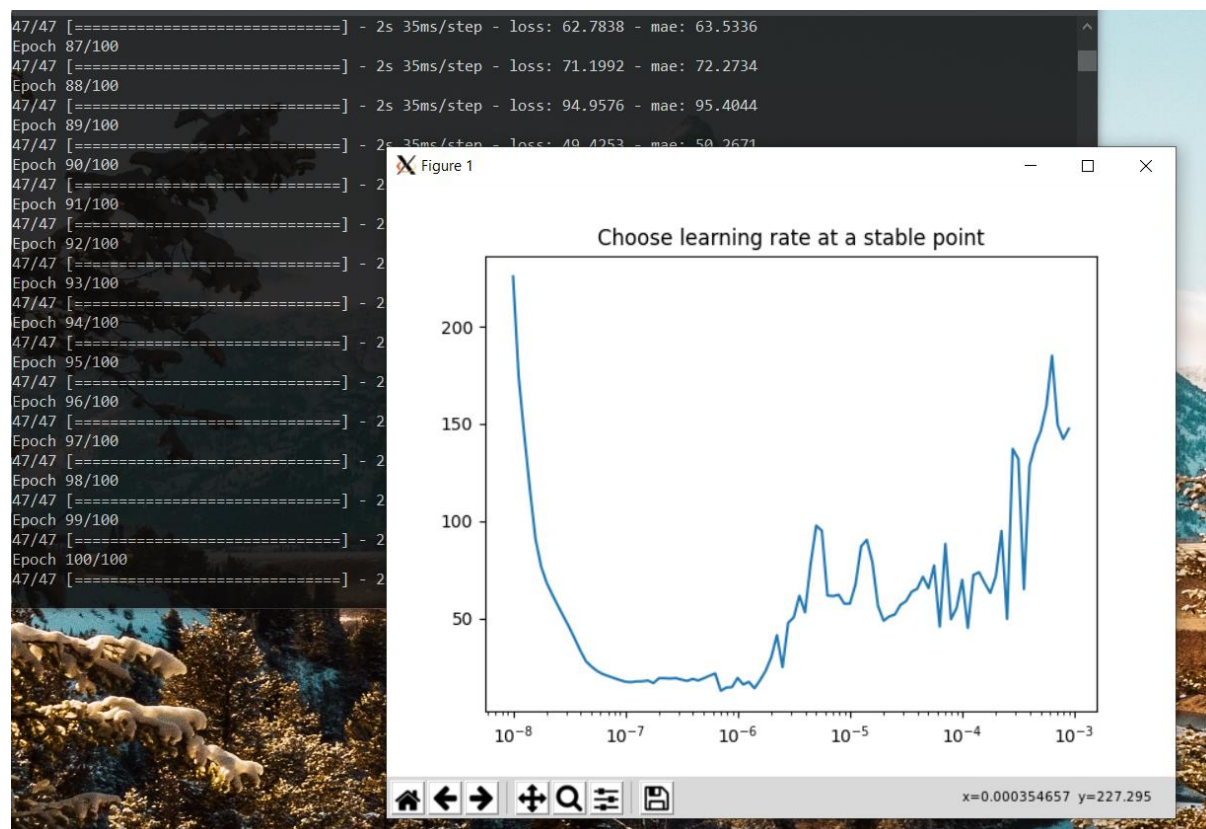
Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, None, 32)	192
lstm (LSTM)	(None, None, 64)	24832
lstm_1 (LSTM)	(None, None, 64)	33024
dense (Dense)	(None, None, 30)	1950
dense_1 (Dense)	(None, None, 10)	310
dense_2 (Dense)	(None, None, 1)	11
lambda (Lambda)	(None, None, 1)	0
Total params: 60,319		
Trainable params: 60,319		
Non-trainable params: 0		

```

Epoch 5/100
47/47 [=====] - 3s 66ms/step - loss: 92.4998 - mae: 91.1478
Epoch 6/100
47/47 [=====] - 3s 66ms/step - loss: 78.4685 - mae: 77.1764
Epoch 7/100
47/47 [=====] - 3s 63ms/step - loss: 69.8981 - mae: 68.6978
Epoch 8/100
47/47 [=====] - 3s 55ms/step - loss: 63.8062 - mae: 62.6617
Epoch 9/100
47/47 [=====] - 3s 58ms/step - loss: 57.9716 - mae: 56.9687
Epoch 10/100
47/47 [=====] - 3s 53ms/step - loss: 52.2722 - mae: 51.4345
Epoch 11/100
47/47 [=====] - 2s 40ms/step - loss: 46.7839 - mae: 45.9649
Epoch 12/100
47/47 [=====] - 2s 50ms/step - loss: 40.6836 - mae: 39.9729
Epoch 13/100
47/47 [=====] - 3s 62ms/step - loss: 34.4185 - mae: 33.8244
Epoch 14/100
47/47 [=====] - 3s 62ms/step - loss: 28.8435 - mae: 28.3941
Epoch 15/100
47/47 [=====] - 3s 58ms/step - loss: 26.0660 - mae: 25.6546
Epoch 16/100
47/47 [=====] - 2s 49ms/step - loss: 23.8102 - mae: 23.4048
Epoch 17/100
47/47 [=====] - 3s 56ms/step - loss: 22.1801 - mae: 21.8647
Epoch 18/100
47/47 [=====] - 3s 59ms/step - loss: 21.0696 - mae: 20.8167
Epoch 19/100
15/47 [=====>.....] - ETA: 2s - loss: 4.9845 - mae: 5.4439

```

The learning rate is calculated based on the number of epochs and will increase at every epoch. This ensures that we can find a stable point in the learning rate. This in turn ensures that the model can fit on the training data accurately. (Here the training is done just to find the optimum learning rate)



Here we can see that the learning rate is stable at 10^{-7} , so we give that as input:

```
Epoch 98/100
47/47 [=====] - 2s 34ms/step - loss: 148.7155 - mae: 150.0306
Epoch 99/100
47/47 [=====] - 2s 35ms/step - loss: 141.0713 - mae: 142.7006
Epoch 100/100
47/47 [=====] - 2s 35ms/step - loss: 145.7427 - mae: 148.0755
Enter the learning rate in exponential form: (eg: 8e-7 = 8 * 10^-7):
1e-7
```

Now the model is trained over the training data with that fixed learning rate for 150 epochs:

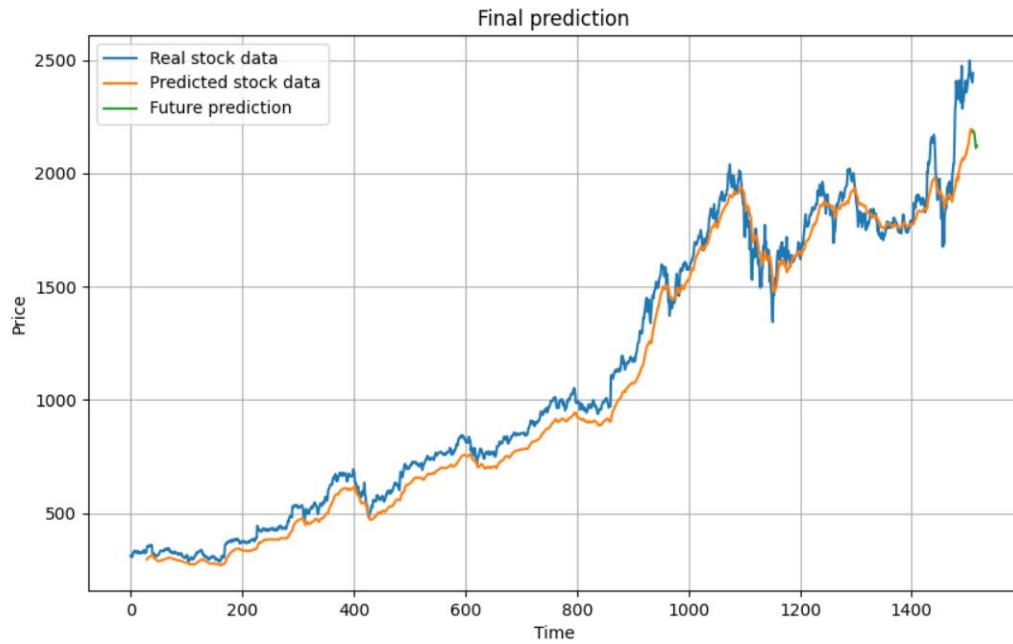
```
Epoch 136/150
47/47 [=====] - 2s 36ms/step - loss: 6.5247 - mae: 6.6555
Epoch 137/150
47/47 [=====] - 2s 36ms/step - loss: 6.5016 - mae: 6.6491
Epoch 138/150
47/47 [=====] - 2s 35ms/step - loss: 6.5411 - mae: 6.6750
Epoch 139/150
47/47 [=====] - 2s 36ms/step - loss: 6.4451 - mae: 6.5844
Epoch 140/150
47/47 [=====] - 2s 35ms/step - loss: 6.4109 - mae: 6.5356
Epoch 141/150
47/47 [=====] - 2s 36ms/step - loss: 6.4731 - mae: 6.6307
Epoch 142/150
47/47 [=====] - 2s 37ms/step - loss: 6.3837 - mae: 6.5640
Epoch 143/150
47/47 [=====] - 3s 57ms/step - loss: 6.2603 - mae: 6.3855
Epoch 144/150
47/47 [=====] - 2s 40ms/step - loss: 6.4334 - mae: 6.5744
Epoch 145/150
47/47 [=====] - 2s 40ms/step - loss: 6.4132 - mae: 6.5678
Epoch 146/150
47/47 [=====] - 2s 40ms/step - loss: 6.1734 - mae: 6.3390
Epoch 147/150
47/47 [=====] - 2s 40ms/step - loss: 6.3055 - mae: 6.4554
Epoch 148/150
47/47 [=====] - 2s 51ms/step - loss: 6.3890 - mae: 6.5438
Epoch 149/150
47/47 [=====] - 2s 43ms/step - loss: 6.2201 - mae: 6.3678
Epoch 150/150
21/47 [=====>.....] - ETA: 1s - loss: 3.2163 - mae: 3.6704
```

The model trained above is a different model but has the same configuration as the previous model used to find the learning rate. This model is solely used for the future prediction of the stock data.

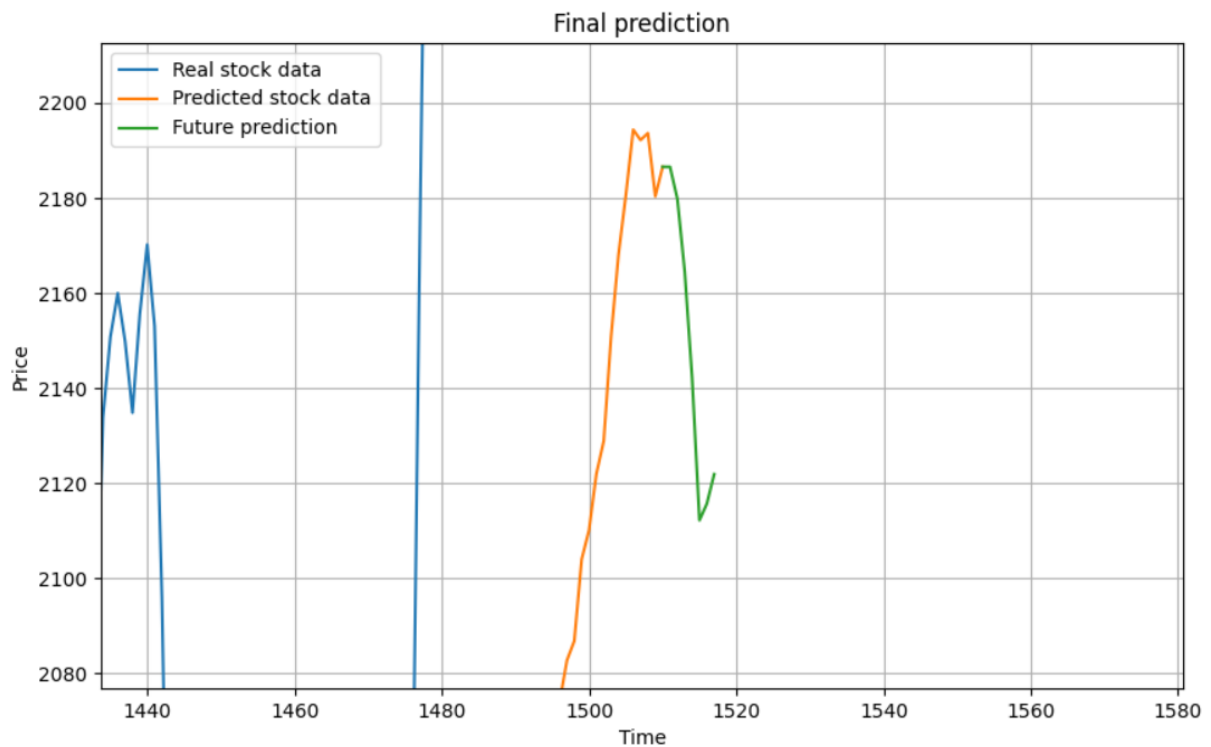
After the training is done, we need to specify the number of days ahead to predict:

```
Epoch 150/150
47/47 [=====] - 2s 40ms/step - loss: 6.1473 - mae: 6.3160
2020-05-31 10:57:20.199758: W tensorflow/core/common_runtime/base_collective_executor.cc:217] BaseCollectiveExecutor::StartAbort Out of range: End of sequence
[[{{node IteratorGetNext}}]]
(Note: Increasing number of days increases error)
Enter no of days in the future to predict: 7
```

Here we have chosen 7. That means that the model will predict the stock prices for the next 7 days, starting the day after the present day. The final output will be a graph as shown:



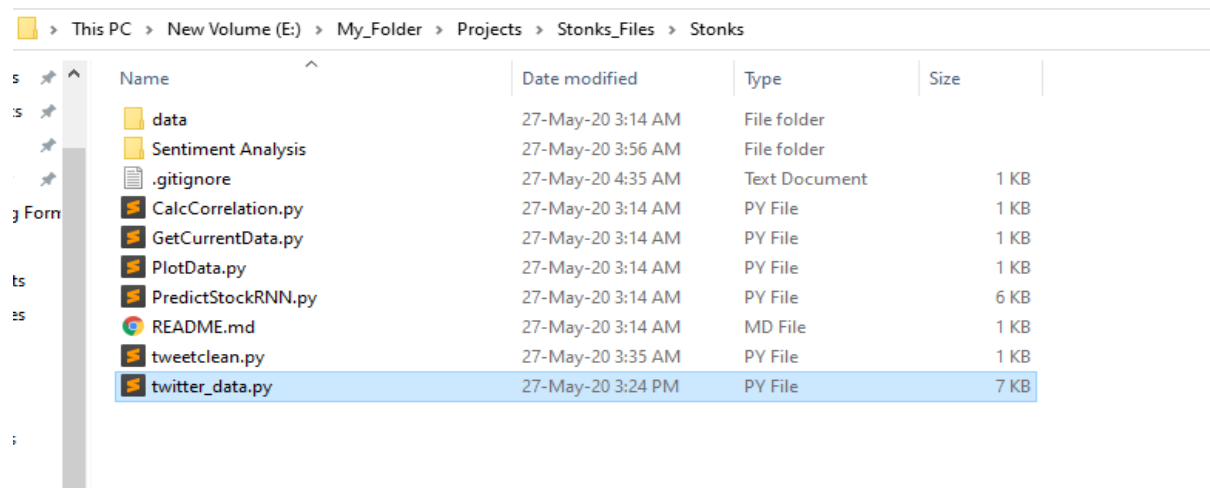
The model takes a window size of 30 so the orange graph is the prediction of the model from taking 30 days of the training data at a time. For the future prediction, the *predict* function inputs the last 30 days to the model and gives an output which corresponds to tomorrow. If more days are specified, the previous output is appended to the new input to the model and a prediction is made again. If we zoom in near the end of the graph (i.e. present day), we can see the green graph:



We can see that the graph predicts a decrease in the stock price and then a slight increase in the last two days. But the general trend of the amazon stock is downwards. As we know with the case of RNNs, the more number of days we predict, the lesser the accuracy of the future prediction will be as the errors of the previous prediction get added up.

2. SENTIMENT ANALYSIS DATASET GENERATION

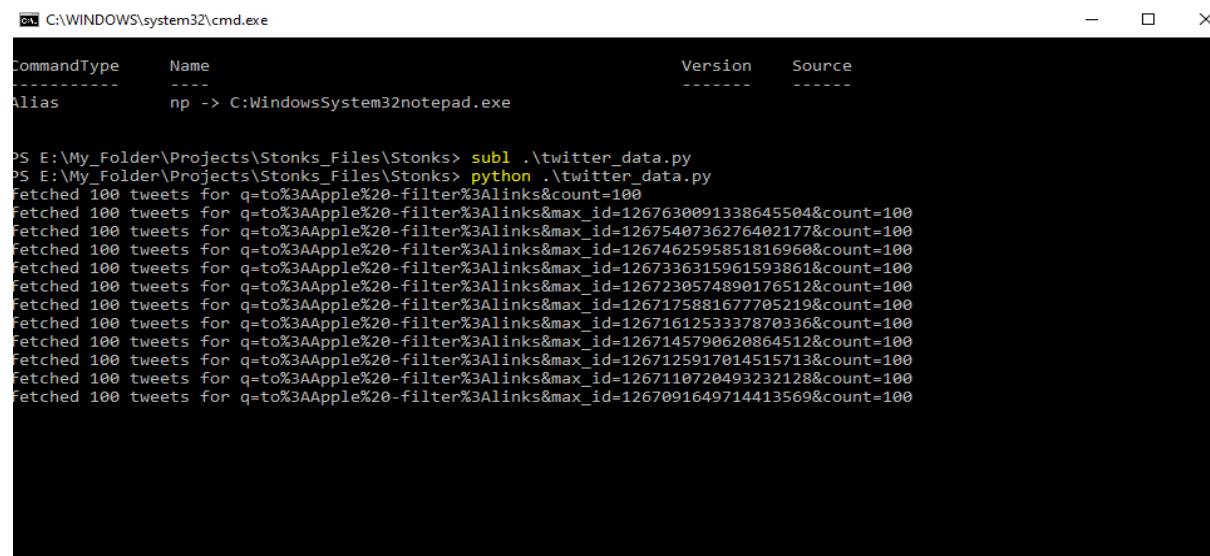
To generate the Training, Validation and Testing datasets for the Sentiment analysis' classifier, we used Twitter as our primary source. Two python based REST Clients, namely Python-Twitter and Tweepy, were used to scrape the respective tweets for dataset generation.



Name	Date modified	Type	Size
data	27-May-20 3:14 AM	File folder	
Sentiment Analysis	27-May-20 3:56 AM	File folder	
.gitignore	27-May-20 4:35 AM	Text Document	1 KB
CalcCorrelation.py	27-May-20 3:14 AM	PY File	1 KB
GetCurrentData.py	27-May-20 3:14 AM	PY File	1 KB
PlotData.py	27-May-20 3:14 AM	PY File	1 KB
PredictStockRNN.py	27-May-20 3:14 AM	PY File	6 KB
README.md	27-May-20 3:14 AM	MD File	1 KB
tweetclean.py	27-May-20 3:35 AM	PY File	1 KB
twitter_data.py	27-May-20 3:24 PM	PY File	7 KB

The twitter_data.py file contains the scraper class and all the required member methods for creating the training dataset and the testing dataset.






















To generate the initial Training dataset, tweets concerning the specific companies are fetched from specific financial accounts as well as from the general population.



```
C:\WINDOWS\system32\cmd.exe

CommandType      Name              Version          Source
-----
Alias            np -> C:\Windows\System32\notepad.exe

PS E:\My_Folder\Projects\Stonks_Files\Stonks> sub1 .\twitter_data.py
PS E:\My_Folder\Projects\Stonks_Files\Stonks> python .\twitter_data.py
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267630091338645504&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267540736276402177&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267462595851816960&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267336315961593861&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267230574890176512&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267175881677705219&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267161253337870336&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267145790620864512&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267125917014515713&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267110720493232128&count=100
Fetched 100 tweets for q=to%3AApple%20-filter%3Alinks&max_id=1267091649714413569&count=100
```

 Amazon_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	52 KB
 Amazon_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	401 KB
 Apple_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	80 KB
 Apple_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	239 KB
 corpus.csv	27-May-20 3:14 AM	Microsoft Excel C...	232 KB
 Facebook_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	42 KB
 Facebook_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	418 KB
 GoldmanSachs_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	1 KB
 GoldmanSachs_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	44 KB
 Google_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	30 KB
 Google_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	274 KB
 Intel_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	15 KB
 Intel_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	9 KB
 jpmorgan_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	11 KB
 jpmorgan_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	12 KB
 Microsoft_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	26 KB
 Microsoft_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	136 KB
 Samsung_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	4 KB
 Samsung_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	44 KB
 Tesla_finance.csv	27-May-20 3:14 AM	Microsoft Excel C...	55 KB
 Tesla_twitter.csv	27-May-20 3:14 AM	Microsoft Excel C...	70 KB

The tweets from the financial accounts are stored in the {Company_name}_finance.csv files, and the tweets from the general public are stored in the {Company_name}_twitter.csv files.

Data Pre-Processing

Raw tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people's usage of social media. Tweets have certain special characteristics such as re- tweets, emoticons, user mentions, etc. which have to be suitably extracted. Therefore, raw twitter data has to be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. We first do some general pre-processing on tweets which is as follows.

- Convert the tweet to lower case.
- Replace 2 or more dots (.) with space.
- Strip spaces and quotes (" and ') from the ends of tweet.
- Replace 2 or more spaces with a single space.

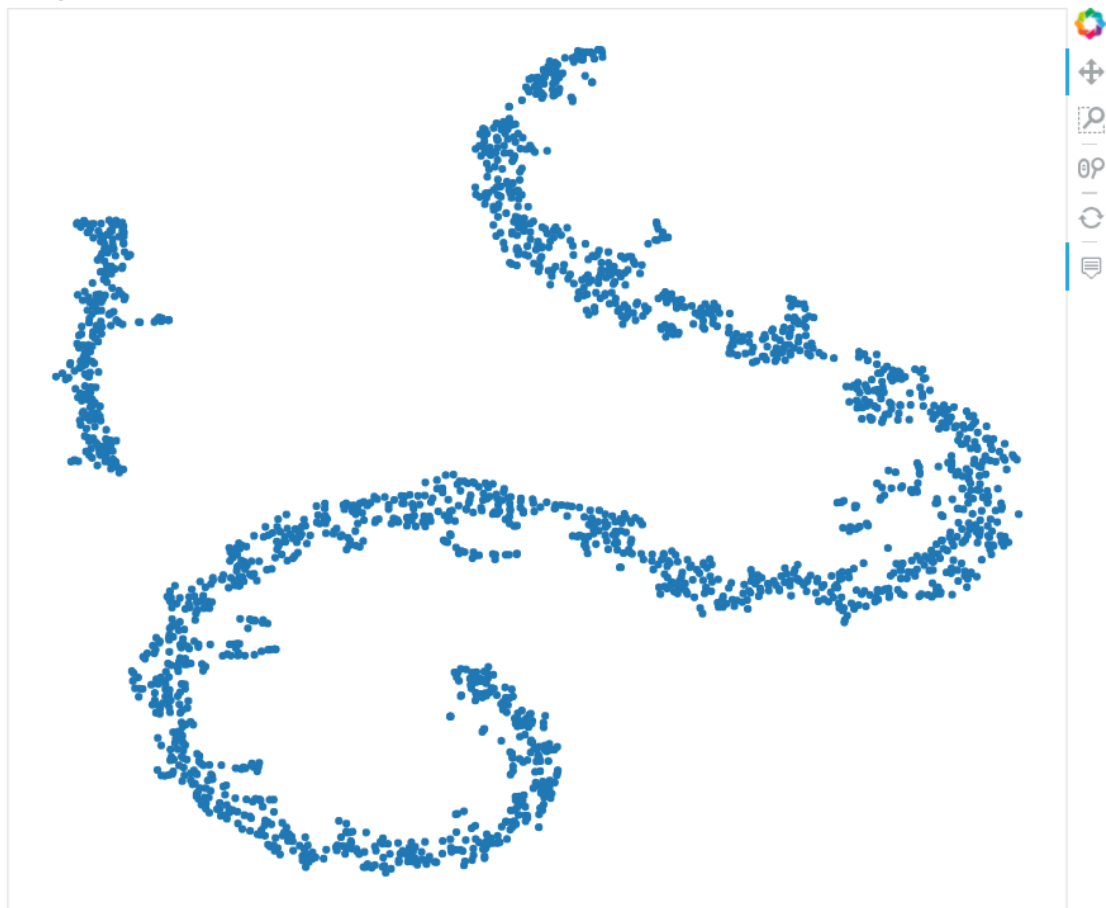
The 'cleaned up' tweets are then pre-processed using REGEX and NLP libraries; all the URLs and links in the tweets are removed, all the '@'s from the accounts mentions are removed, and finally the remaining words in the tweet are tokenized and stored in a List of Sets, and are finally ready to be used by word2Vec.

3. SENTIMENT ANALYSIS

I. Word2Vec

Word2vec is a group of Deep Learning models developed by Google with the aim of capturing the context of words while at the same time proposing a very efficient way of pre-processing raw text data. This model takes a large corpus of documents like tweets or news articles as input and generates a vector space of typically several hundred dimensions. The powerful concept behind word2vec is that word vectors that are close to each other in the vector space represent words that are not only of the same meaning but of the same context as well. What is interesting about the vector representation of words is that it automatically embeds several features that we would normally have to handcraft ourselves. Since word2vec relies on Deep Neural Nets to detect patterns, we can rely on it to detect multiple features on different levels of abstractions.

A map of 2659 word vectors



II. Naive Bayes

Naive Bayes is a simple model which can be used for text classification. In this model, the class \hat{c} is assigned to a tweet t , where

$$\hat{c} = \underset{c}{\operatorname{argmax}} P(c|t)$$
$$P(c|t) \propto P(c) \prod_{i=1}^n P(f_i|c)$$

In the formula above, f_i represents the i -th feature of total n features. $P(c)$ and $P(f_i|c)$ can be obtained through maximum likelihood estimates.

We used MultinomialNB from `sklearn.naive_bayes` package of scikit-learn for Naive Bayes classification. We used Laplace smoothed version of Naive Bayes with the smoothing parameter α set to its default value of 1. We used sparse vector representation for classification and ran experiments using both presence and frequency feature types. We found that presence features outperform frequency features because Naive Bayes is essentially built to work better on integer features rather than floats. We also observed that addition of bigram features improves the accuracy. We obtain a best validation accuracy of 79.68% using Naive Bayes with presence of unigrams and bigrams.

III. CNN

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They can also be used in NLP when words are presented as vectors.

Convolutional neural networks ingest and process images as tensors, and tensors are matrices of numbers with additional dimensions.

In simpler words, Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM / Softmax) on the last (fully-connected) layer and all the tips/tricks developed for regular Neural Networks still apply.

In this model, we use embeddings for the Word2Vec vectors we have generated before. After training in convolutions, we use MaxPooling to downsize the samples and then pass it through a Dense layer to get our outputs. Backpropagation is then carried out using

'categorical_crossentropy' loss function(since there are 3 categories). 'Adam' optimizer is used since it is very efficient for NLP.

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 200, 64)	9600000
conv1d_1 (Conv1D)	(None, 200, 32)	6176
max_pooling1d_1 (MaxPooling1D)	(None, 100, 32)	0
dropout_1 (Dropout)	(None, 100, 32)	0
conv1d_2 (Conv1D)	(None, 100, 32)	3104
max_pooling1d_2 (MaxPooling1D)	(None, 50, 32)	0
dropout_2 (Dropout)	(None, 50, 32)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_1 (Dense)	(None, 2)	3202

```
Total params: 9,612,482
Trainable params: 9,612,482
Non-trainable params: 0
```

```
2020-06-03 17:47:48.465166: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library curand64_10.dll
2020-06-03 17:47:48.620919: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cusolver64_10.dll
2020-06-03 17:47:48.755578: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cusparse64_10.dll
2020-06-03 17:47:49.094115: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudnn64_7.dll
2020-06-03 17:47:49.288539: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1697] Adding visible gpu devices: 0
2020-06-03 17:47:49.325675: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX AVX2
2020-06-03 17:47:49.342899: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1555] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: GeForce MX130 computeCapability: 5.0
coreClock: 1.2415GHz coreCount: 3 deviceMemorySize: 4.00GiB deviceMemoryBandwidth: 14.30GiB/s
2020-06-03 17:47:49.350279: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudart64_101.dll
2020-06-03 17:47:49.353935: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cublas64_10.dll
2020-06-03 17:47:49.357645: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cufft64_10.dll
2020-06-03 17:47:49.361026: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library curand64_10.dll
2020-06-03 17:47:49.364833: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cusolver64_10.dll
2020-06-03 17:47:49.368288: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cusparse64_10.dll
2020-06-03 17:47:49.372432: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudnn64_7.dll
2020-06-03 17:47:49.376914: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1697] Adding visible gpu devices: 0
2020-06-03 17:48:14.556375: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1096] Device interconnect StreamExecutor
with strength 1 edge matrix:
2020-06-03 17:48:14.567444: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102] 0
2020-06-03 17:48:14.570626: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] 0: N
2020-06-03 17:48:14.668657: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1241] Created TensorFlow device (/job:loc
alhost/replica:0/task:0/device:GPU:0 with 3041 MB memory) -> physical GPU (device: 0, name: GeForce MX130, pci bus id: 0
000:01:00.0, compute capability: 5.0)
C:\Users\Medha Joshi\Anaconda3\lib\site-packages\tensorflow_core\python\framework\indexed_slices.py:433: UserWarning: Co
nverting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
"Converting sparse IndexedSlices to a dense Tensor of unknown shape."
2020-06-03 17:48:21.006328: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cublas64_10.dll
2020-06-03 17:48:25.380954: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudnn64_7.dll
2020-06-03 17:48:34.589173: W tensorflow/stream_executor/gpu/redzone_allocator.cc:312] Internal: Invoking GPU asm compil
ation is supported on Cuda non-Windows platforms only
Relying on driver to perform ptx compilation. This message will be only logged once.
Logloss score: 0.44
Validation set Accuracy: 0.90
```

Accuracy of 90% was achieved during training.

```

2020-06-04 00:11:17.626755: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudnn64_7.dll
2020-06-04 00:11:17.636311: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1697] Adding visible gpu devices: 0
2020-06-04 00:11:19.016854: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1096] Device interconnect StreamExecutor
with strength 1 edge matrix:
2020-06-04 00:11:19.024249: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102] 0
2020-06-04 00:11:19.031526: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] 0: N
2020-06-04 00:11:19.037259: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1241] Created TensorFlow device (/job:loc
alhost/replica:0/task:0/device:GPU:0 with 3041 MB memory) -> physical GPU (device: 0, name: GeForce MX130, pci bus id: 0
000:01:00.0, compute capability: 5.0)
C:\Users\Medha Joshi\Anaconda3\lib\site-packages\tensorflow_core\python\framework\indexed_slices.py:433: UserWarning: Co
nverting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
TESTING
Model: "sequential_1"

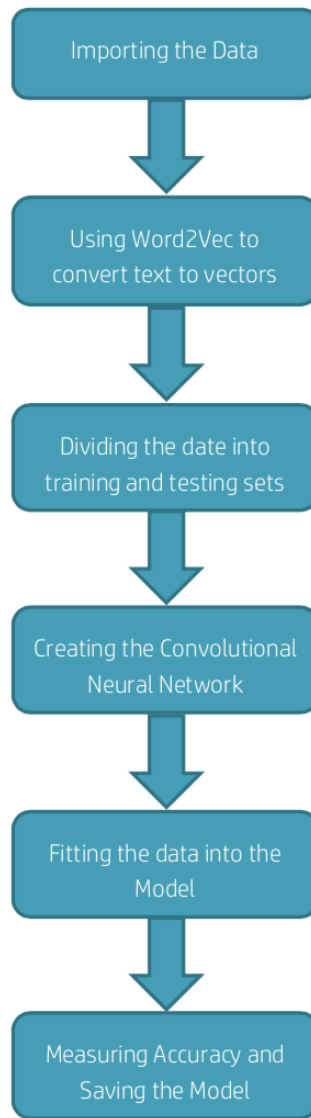
Layer (type)                 Output Shape                 Param #
=====
embedding_1 (Embedding)      (None, 200, 64)             9600000
conv1d_1 (Conv1D)            (None, 200, 32)             6176
max_pooling1d_1 (MaxPooling1 (None, 100, 32)             0
dropout_1 (Dropout)          (None, 100, 32)             0
conv1d_2 (Conv1D)            (None, 100, 32)             3104
max_pooling1d_2 (MaxPooling1 (None, 50, 32)             0
dropout_2 (Dropout)          (None, 50, 32)             0
flatten_1 (Flatten)          (None, 1600)                0
dense_1 (Dense)              (None, 2)                   3202
=====
Total params: 9,612,482
Trainable params: 9,612,482
Non-trainable params: 0

2020-06-04 00:11:20.806969: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cublas64_10.dll
2020-06-04 00:11:21.096663: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudnn64_7.dll
2020-06-04 00:11:22.016008: W tensorflow/stream_executor/gpu/redzone_allocator.cc:312] Internal: Invoking GPU asm compil
ation is supported on Cuda non-Windows platforms only
Relying on driver to perform ptx compilation. This message will be only logged once.
Current Sentiment: [0 0 0]

```

When the model was used to predict sentiment based on tweets in the last week(27/04/2020-4/05/2020) with respect to Amazon, the sentiment was shown as neutral.

Sentiment Analysis

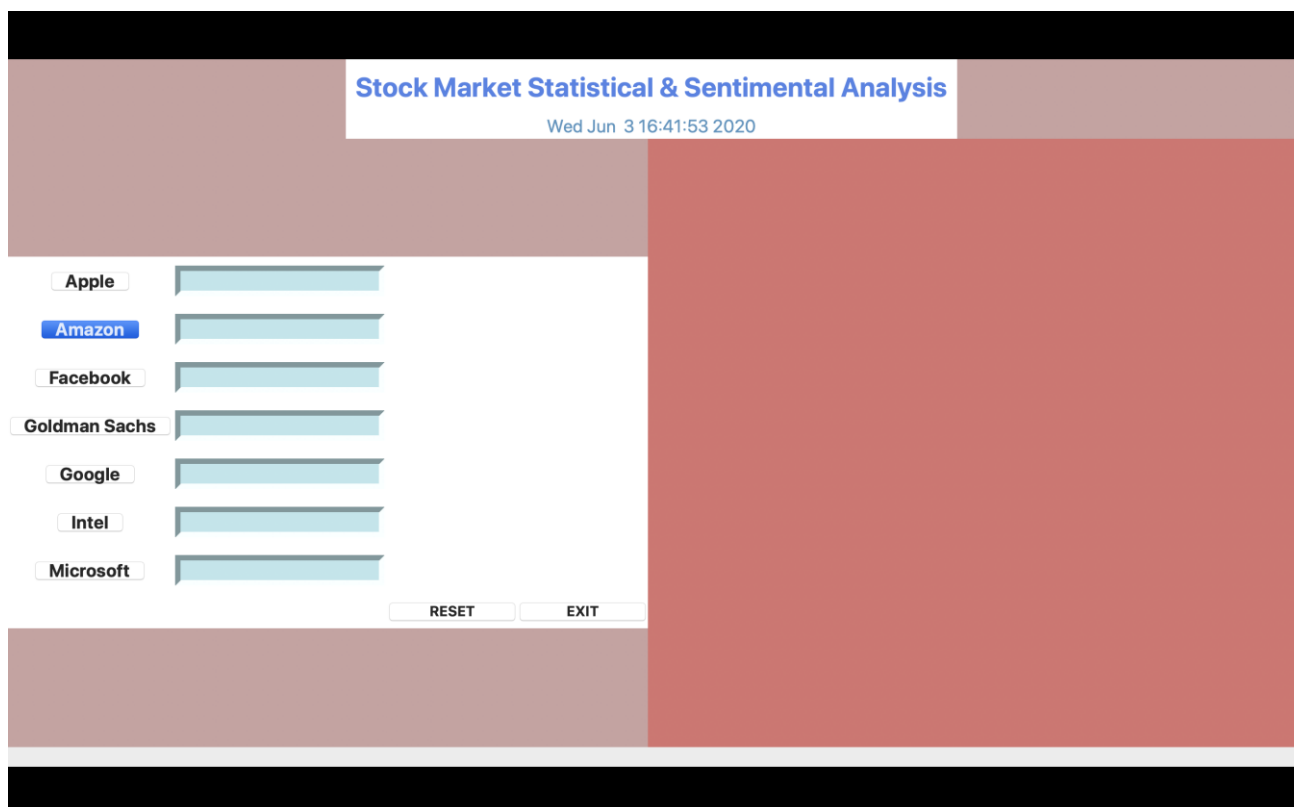
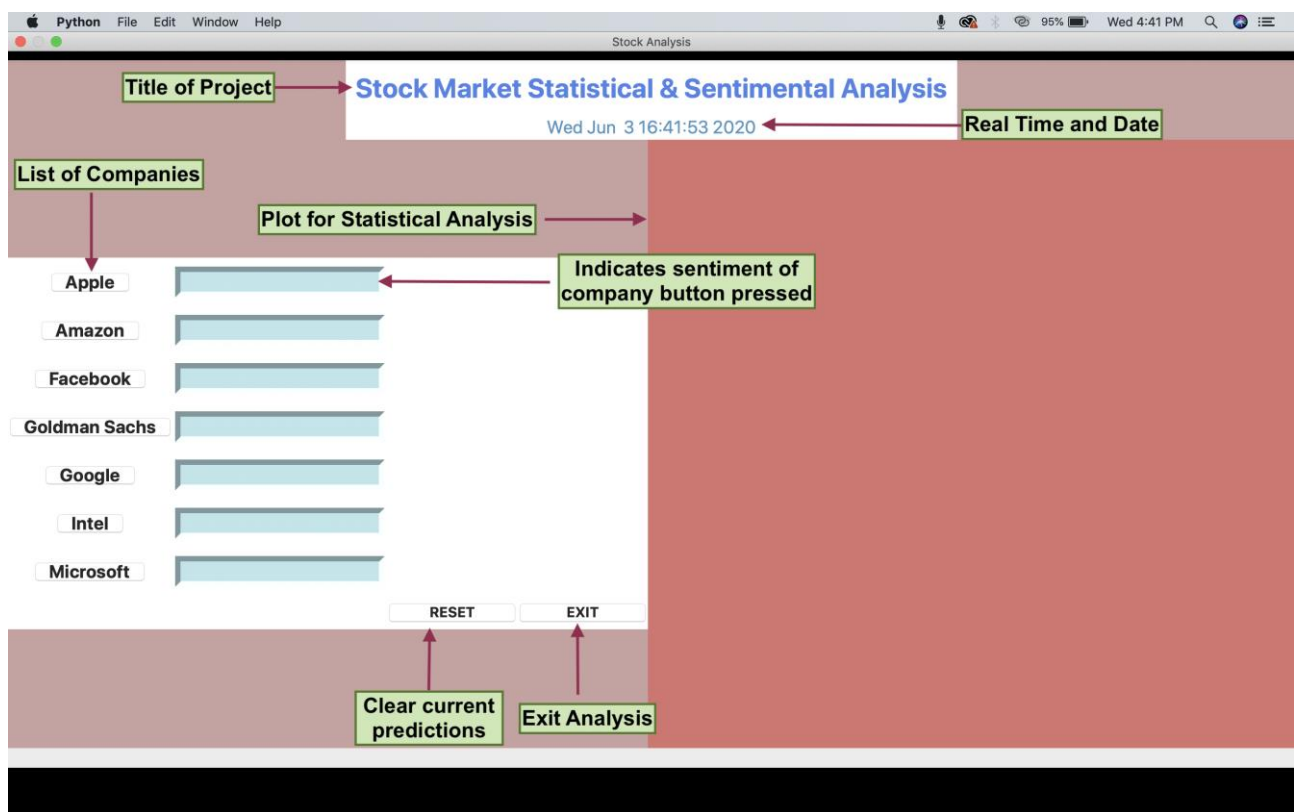


4. USER INTERFACE

To display the results of the analysis company wise, we made an User friendly GUI to access and predict the stocks of the company.

The functionality of the UI are as follows:

- **Title and Time Stamp:** The title provides an overview of the project. The timestamp shows the real time when the predictions are made. This ensures the predictions made are of recent Tweets or data.
- **Company Buttons:** Click on each button leads to display of Result for its sentiment analysis as well as plotting of graph for predicting the stocks based on recent trends.
- **Reset Button:** Helps you clear all the company buttons you have clicked making it null again.



For example, on clicking company button of AMAZON, We get the Sentiment Analysis

Stock Market Statistical & Sentimental Analysis

Wed Jun 3 16:41:53 2020

Apple

Amazon

Facebook

Goldman Sachs

Google

Intel

Microsoft



Output for the company and the stock prediction graph for the same as shown below:

CODES : <https://github.com/DraconioSchiffer/Stonks>

Work Submitted by:

Devdatta Joshi (170953070)
Prakriti Parihar (170953110)
Utkarsh Chhapekar (170953106)
Kunal Khanwalkar (170953162)

-----X-----