Project – Stock Market Sentiment

1) Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.

This is a data set on "stock news" from twitter, labelled as either "positive" or "negative". I will try to train a model that accurately predicts the impact of tweets on stock market moves.

Data-set: https://www.kaggle.com/yash612/stockmarket-sentiment-dataset

There are 5791 rows of data, with 2 columns (the "text" of the tweet, and the labelled corresponding sentiment).

Running data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5791 entries, 0 to 5790
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Text       5791 non-null   object
 1   Sentiment  5791 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 90.6+ KB
```

To have an early grasp of the data, I have inspected its first 10 rows, which look like the following:

Out[2]:

	Text	Sentiment
0	Kickers on my watchlist XIDE TIT SOQ PNK CPW B...	1
1	user: AAP MOVIE. 55% return for the FEA/GEED i...	1
2	user I'd be afraid to short AMZN - they are lo...	1
3	MNTA Over 12.00	1
4	OI Over 21.37	1
5	PGNX Over 3.04	1
6	AAP - user if so then the current downtrend wi...	-1
7	Monday's relative weakness. NYX WIN TIE TAP IC...	-1
8	GOOG - ower trend line channel test & volume s...	1
9	AAP will watch tomorrow for ONG entry.	1

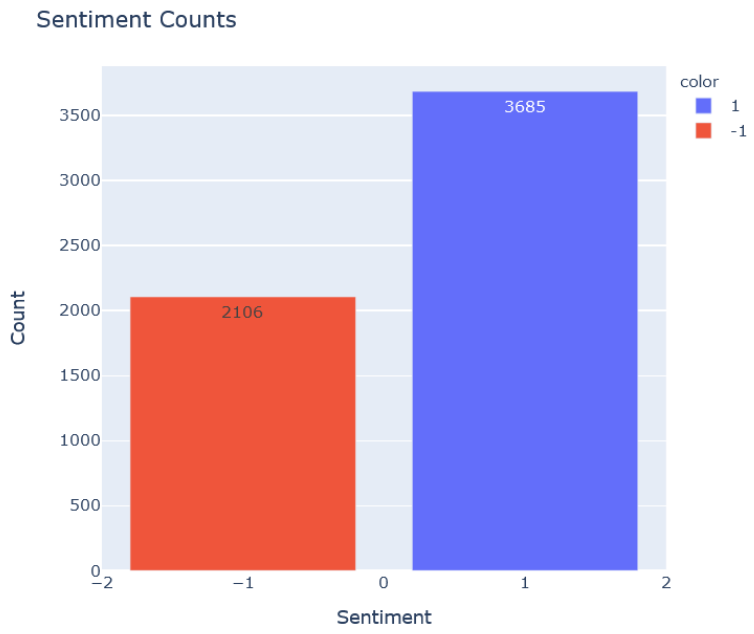
2) Main objective of the analysis that also specifies whether your model will be focused on a specific type of Deep Learning or Reinforcement Learning algorithm and the benefits that your analysis brings to the business or stakeholders of this data.

The main objective of the analysis would be to train a model that accurately classify the impact of relevant tweets on stock-market sentiment (i.e., either positive or negative). Thus, given our data-set, the model will be trained using some sort of Neural Networks (Deep Learning).

If successful, this model would benefit an investor using an active trading strategy (either helping to design a new investment strategy, or to fine-tune an existing one, providing additional insights from "real-time and high-frequency" information such as tweets).

3) Brief summary of data exploration and actions taken for data cleaning or feature engineering.

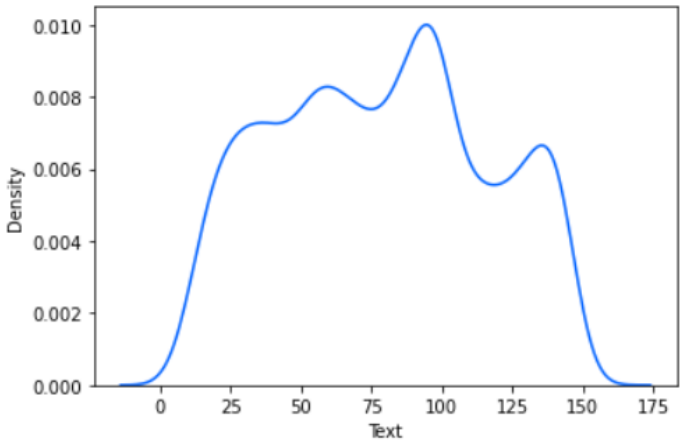
For data exploration of our 5791 rows of data, I have decided to inspect how much of these were classified as "positive = 1" or "negative = -1":



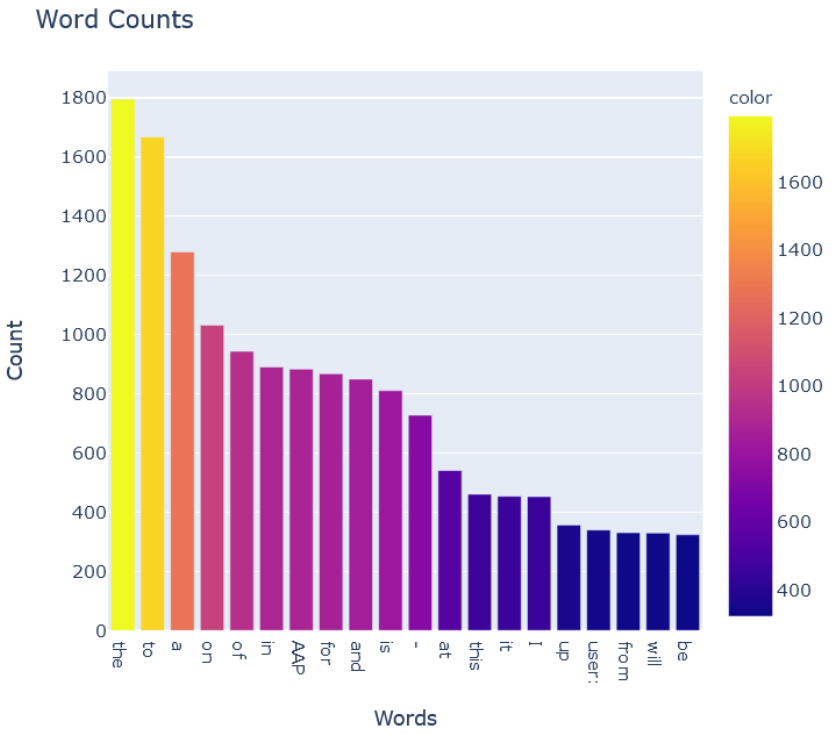
That is, 64% of tweets in our data-set were labelled as having a "positive sentiment", while the remaining 36% of tweets were labelled as having a "negative sentiment".

I have also decided to inspect for the length of the tweets in the data-set, with the following density-output:

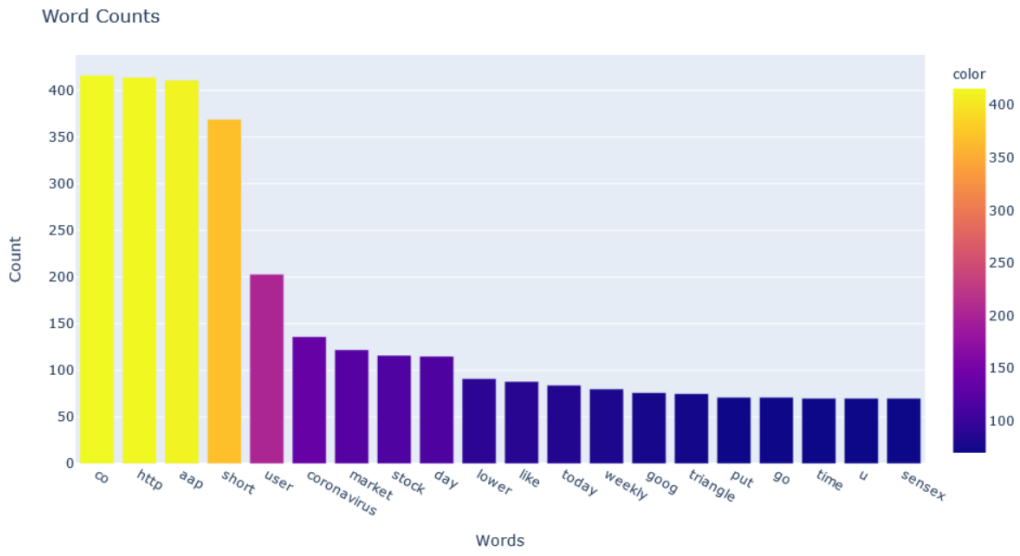
```
Out[7]: <AxesSubplot:xlabel='Text', ylabel='Density'>
```



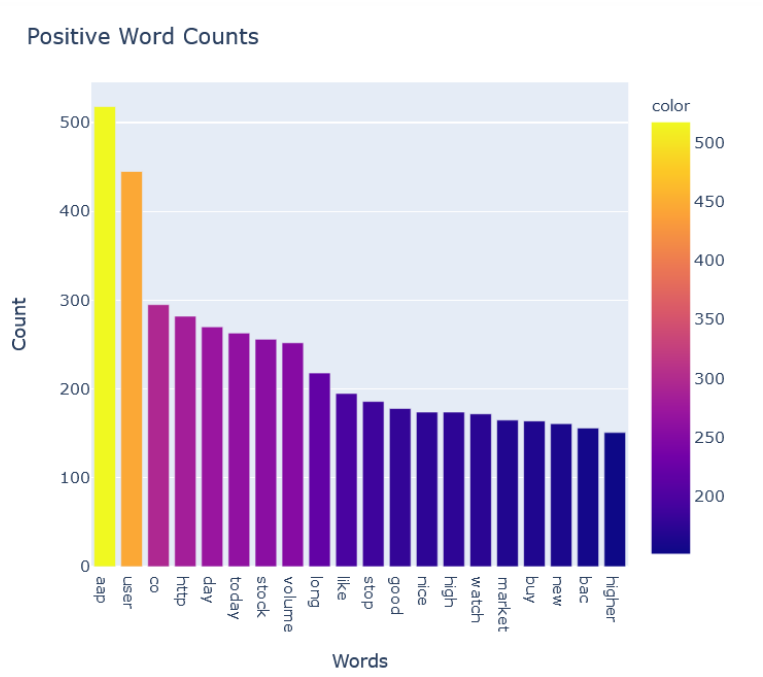
I have also decided to plot a “Word Counter” for the 20 more common words:



So a lot of *stopwords* such as “the”, “to”, “a” or “on”, for instance. Therefore, importing some appropriate libraries from nltk, I cleared special characters and stopwords. After this “cleaning”, our Word Counter looks as follows:

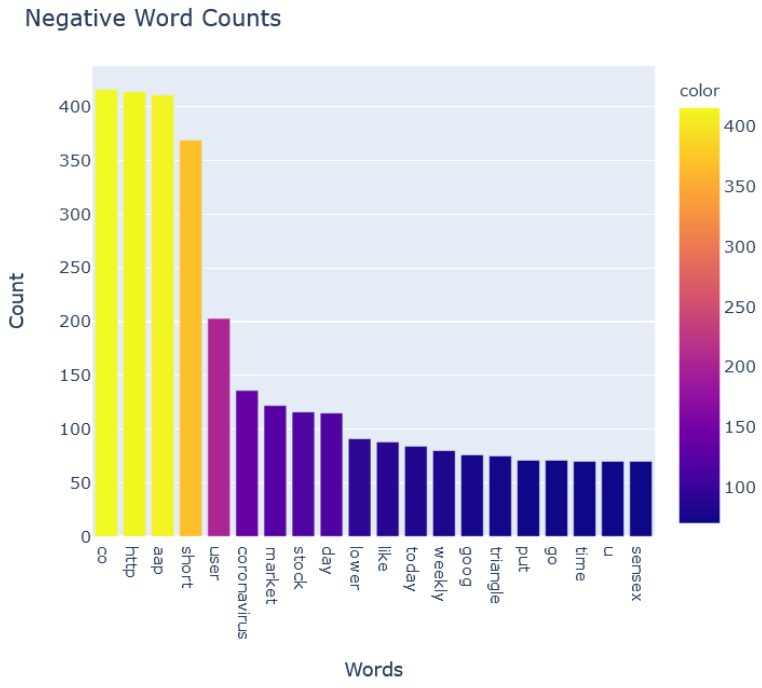


Finally, I decided to plot a “Positive Word Count”, that is, counting the frequency of words in the tweets labelled as “positive”:



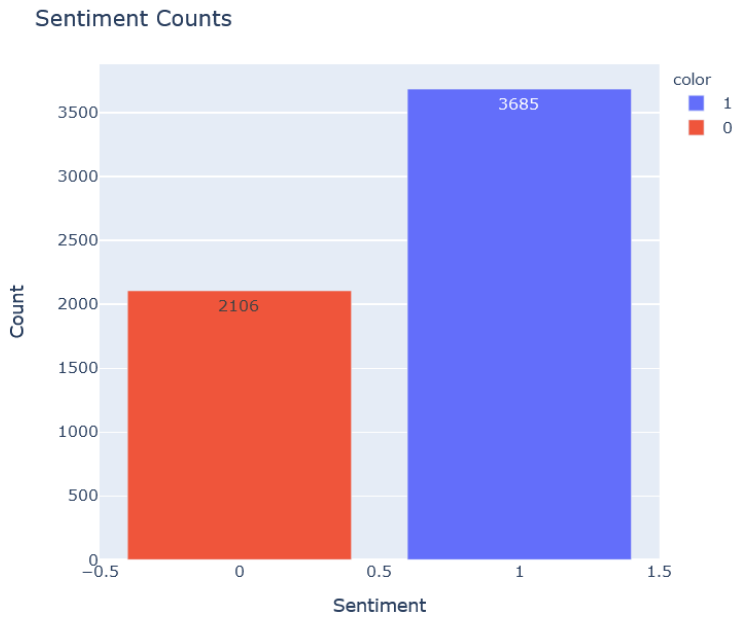
With some intuitive words such as “long”, “good”, “nice”, “high” or “buy” appearing.

And for the “Negative Word Counter”:



Words such as “short”, “coronavirus” or “lower” frequently appear in tweets labelled as “Negative Sentiment”, which makes sense.

In the meantime, regarding the “Negative Sentiment” label, I have changed it from “-1” to “0”:



I have applied a Count Vectorizer (max_features=100) to the Text data, extracting features from the text in our data-set.

Finally, a last preparation just before modeling was to split the data into Training Data and Testing Data, and for that I have decided to use a Stratified Shuffle Split, given that as we have seen before the data-set is fairly unbalanced (towards a higher percentage of positively labelled tweets at the expense of negatively labelled tweets…).

4) **Summary of training at least three variations of the Deep Learning model you selected. For example, you can use different clustering techniques or different hyperparameters.**

Model I – Neural Network (Adam Optimizer)

I started with the following Neural Network, with three dense layers, the first two using a 'relu' activation function and the last using a 'sigmoid' activation function. I have chosen to start with an Adam Optimizer. Model's summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                1616
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,905
Trainable params: 1,905
Non-trainable params: 0
_________________________________________________________________
```

I fitted the model to the train_data for 100 epochs, with the following results:

```
Train on 4053 samples, validate on 1738 samples
Epoch 1/100
4053/4053 [==============================] - 1s 178us/step - loss:
0.6920 - accuracy: 0.6346 - val_loss: 0.6907 - val_accuracy: 0.6364
Epoch 2/100
4053/4053 [==============================] - 0s 65us/step - loss: 0.
6887 - accuracy: 0.6363 - val_loss: 0.6863 - val_accuracy: 0.6364
Epoch 3/100
4053/4053 [==============================] - 0s 59us/step - loss: 0.
6820 - accuracy: 0.6363 - val_loss: 0.6777 - val_accuracy: 0.6364
Epoch 4/100
4053/4053 [==============================] - 0s 74us/step - loss: 0.
6699 - accuracy: 0.6363 - val_loss: 0.6638 - val_accuracy: 0.6364
Epoch 5/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
6535 - accuracy: 0.6363 - val_loss: 0.6485 - val_accuracy: 0.6364
Epoch 6/100
4053/4053 [==============================] - 0s 86us/step - loss: 0.
6373 - accuracy: 0.6363 - val_loss: 0.6358 - val_accuracy: 0.6364
```

[…]

```
4053/4053 [==============================] - 0s 78us/step - loss: 0.
5128 - accuracy: 0.7461 - val_loss: 0.5760 - val_accuracy: 0.7048
Epoch 96/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
5126 - accuracy: 0.7469 - val_loss: 0.5759 - val_accuracy: 0.7054
Epoch 97/100
4053/4053 [==============================] - 0s 83us/step - loss: 0.
5123 - accuracy: 0.7469 - val_loss: 0.5758 - val_accuracy: 0.7048
Epoch 98/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
5122 - accuracy: 0.7466 - val_loss: 0.5759 - val_accuracy: 0.7066
Epoch 99/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
5119 - accuracy: 0.7464 - val_loss: 0.5756 - val_accuracy: 0.7037
Epoch 100/100
4053/4053 [==============================] - 0s 86us/step - loss: 0.
5116 - accuracy: 0.7478 - val_loss: 0.5756 - val_accuracy: 0.7043

<keras.callbacks.callbacks.History at 0x158177ef2e8>
```

So, stabilizing with a validation_accuracy around 70%.

Model II – Neural Network (RMSprop Optimizer)

Then, I decided to train the exact same structure of Neural Network but using a RMSprop Optimizer instead. Model's summary (obviously like previous model's):

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 16)                1616
_________________________________________________________________
dense_5 (Dense)              (None, 16)                272
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,905
Trainable params: 1,905
Non-trainable params: 0
```

Fitting the data for 100 epochs:

```
Train on 4053 samples, validate on 1738 samples
Epoch 1/100
4053/4053 [==============================] - 1s 165us/step - loss:
0.6919 - accuracy: 0.6353 - val_loss: 0.6908 - val_accuracy: 0.6364
Epoch 2/100
4053/4053 [==============================] - 0s 72us/step - loss: 0.
6895 - accuracy: 0.6363 - val_loss: 0.6881 - val_accuracy: 0.6364
Epoch 3/100
4053/4053 [==============================] - 0s 76us/step - loss: 0.
6862 - accuracy: 0.6363 - val_loss: 0.6844 - val_accuracy: 0.6364
Epoch 4/100
4053/4053 [==============================] - 0s 74us/step - loss: 0.
6818 - accuracy: 0.6363 - val_loss: 0.6796 - val_accuracy: 0.6364
Epoch 5/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
6762 - accuracy: 0.6363 - val_loss: 0.6737 - val_accuracy: 0.6364
Epoch 6/100
4053/4053 [==============================] - 0s 82us/step - loss: 0.
6692 - accuracy: 0.6363 - val_loss: 0.6665 - val_accuracy: 0.6364
```

[…]

```
Epoch 95/100
4053/4053 [==============================] - 0s 79us/step - loss: 0.
5201 - accuracy: 0.7417 - val_loss: 0.5789 - val_accuracy: 0.7117
Epoch 96/100
4053/4053 [==============================] - 0s 77us/step - loss: 0.
5200 - accuracy: 0.7417 - val_loss: 0.5788 - val_accuracy: 0.7106
Epoch 97/100
4053/4053 [==============================] - 0s 82us/step - loss: 0.
5198 - accuracy: 0.7419 - val_loss: 0.5788 - val_accuracy: 0.7117
Epoch 98/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
5197 - accuracy: 0.7419 - val_loss: 0.5788 - val_accuracy: 0.7112
Epoch 99/100
4053/4053 [==============================] - 0s 78us/step - loss: 0.
5195 - accuracy: 0.7414 - val_loss: 0.5790 - val_accuracy: 0.7117
Epoch 100/100
4053/4053 [==============================] - 0s 83us/step - loss: 0.
5194 - accuracy: 0.7417 - val_loss: 0.5790 - val_accuracy: 0.7117
```

So, even though accuracy on training_data is marginally lower, accuracy on the validation data is a bit higher (around 71%, above the 70% reached with the Adam optimizer), suggesting less overfitting issues and a slightly more robust out-of-sample performance than with Model I.

Model III – Recurrent Neural Network (RMSprop Optimizer)

Finally, I decided to train a recurrent neural network. Intuitively, this kind of neural network should arguably be more suited for text data. I have added an embedding layer, a simple RNN layer, and then a dense layer with a ‘sigmoid’ activation function, as follows:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          3200
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 10)                430
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 11
=================================================================
Total params: 3,641
Trainable params: 3,641
Non-trainable params: 0
_________________________________________________________________
```

I compiled it with a RMSprop optimizer, given that it seemingly performed better for the above models. For 100 epochs, the results for the Recurrent Neural Network were the following:

```
Train on 4053 samples, validate on 1738 samples
Epoch 1/100
4053/4053 [==============================] - 5s 1ms/step - loss: 0.6
555 - accuracy: 0.6316 - val_loss: 0.6430 - val_accuracy: 0.6352
Epoch 2/100
4053/4053 [==============================] - 4s 958us/step - loss:
0.6415 - accuracy: 0.6417 - val_loss: 0.6345 - val_accuracy: 0.6404
Epoch 3/100
4053/4053 [==============================] - 4s 967us/step - loss:
0.6355 - accuracy: 0.6489 - val_loss: 0.6370 - val_accuracy: 0.6341
Epoch 4/100
4053/4053 [==============================] - 4s 970us/step - loss:
0.6373 - accuracy: 0.6420 - val_loss: 0.6411 - val_accuracy: 0.6341
Epoch 5/100
4053/4053 [==============================] - 4s 966us/step - loss:
0.6329 - accuracy: 0.6447 - val_loss: 0.6325 - val_accuracy: 0.6467
Epoch 6/100
4053/4053 [==============================] - 4s 979us/step - loss:
0.6332 - accuracy: 0.6450 - val_loss: 0.6307 - val_accuracy: 0.6617
```

[…]

```
Epoch 95/100
4053/4053 [==============================] - 4s 1ms/step - loss: 0.5
847 - accuracy: 0.7029 - val_loss: 0.6028 - val_accuracy: 0.6847
Epoch 96/100
4053/4053 [==============================] - 4s 998us/step - loss:
0.5839 - accuracy: 0.7049 - val_loss: 0.6132 - val_accuracy: 0.6669
Epoch 97/100
4053/4053 [==============================] - 4s 925us/step - loss:
0.5892 - accuracy: 0.6894 - val_loss: 0.6132 - val_accuracy: 0.6789
Epoch 98/100
4053/4053 [==============================] - 4s 999us/step - loss:
0.5906 - accuracy: 0.6953 - val_loss: 0.6052 - val_accuracy: 0.6749
Epoch 99/100
4053/4053 [==============================] - 4s 1ms/step - loss: 0.5
840 - accuracy: 0.6985 - val_loss: 0.6069 - val_accuracy: 0.6835
Epoch 100/100
4053/4053 [==============================] - 4s 1ms/step - loss: 0.5
914 - accuracy: 0.6884 - val_loss: 0.6040 - val_accuracy: 0.6899
```

So, accuracy disappointingly lower than with the previous two models! I tried different characteristics for the layers and different optimizers, but the conclusions did not change materially. I strongly welcome your feedback on ways to improve the RNN model!

5) **A paragraph explaining which of your Deep Learning models you recommend as a final model that best fits your needs in terms of accuracy or explainability.**

I would chose Model II, the Neural Network using RMSprop Optimizer, because it scored the highest accuracy on test data (around 71%!). Still, I would like to dig deeper on why the RNN model isn't performing better, since I initially suspected (hoped…) that it would be the one that would be the best-performing one since we are dealing with text data.

6) **Summary Key Findings and Insights, which walks your reader through the main findings of your modeling exercise.**

First, I found that the initial data exploration was already somewhat insightful, given that the "Word Counters" highlighted the most common words associated with positive or negative sentiment. Still, some words such as "market" were quite frequent in both positive and negative tweets, thus a deep learning model was attempted to do better than a "bag of words" approach.

Further insights:

- A 71% validation accuracy is achievable with a multi-layer perceptron, using RMSprop Optimizer.
- Beware, though, that 71% accuracy is not spectacularly high, given that 64% of our data set featured tweets labelled as "positive" (i.e., a model with 64% accuracy could be achieved just by classifying every tweet as "positive").
- A Recurrent Neural Network didn't perform as accurately as hoped for, but I would not dismiss it, looking at ways to improve it and eventually leverage its capabilities.

7) **Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model or adding specific data features to achieve a better model.**

I would start by looking into ways to improve the RNN model, given my suspicion that it *should* have performed better given that we are dealing with text data. So there might be room for improvement, either by changing layer configuration or its hyperparameters. In this regard, a LSTM memory model could also be tried.

Regarding feature engineering, other ways of cleaning "stopwords" could also be tried. For instance, "http" is a *word* most frequently identified in both the "Positive Counter" and the "Negative Counter", but it might well be adding just noise without much predictive power, given that it looks like the beginning of a link to a website (that could have either meaning!).

Tiago Monteiro Rabaça