## **Distributed Data Processing and Storage Strategy**

In this project, I created a full pipeline using Apache Spark to process data and used both SQL and NoSQL databases to store it. I worked with stock market data because it was required in the assignment, and also because it's commonly used for financial analysis.

I chose to use MongoDB and MySQL for storing the data. I picked these two databases because I noticed they are mentioned in many job descriptions, and I wanted to get more practice with them. This way, I could learn something useful for both this assignment and my future career.

First, I loaded the stock data for five different companies into both MySQL and MongoDB. Then, I used Apache Spark to clean and transform this data. Spark was a good choice because it is fast and can handle large amounts of data. I used PySpark because I already knew Python, and it works well with Spark.

For the next steps, I mainly worked with MongoDB. I loaded the stock price data from MongoDB using Spark with the MongoDB connector. After checking the data (like looking for missing values and making sure the dates were correct), I filtered it to keep only the columns I needed, such as Date and Close price.

I converted the cleaned Spark DataFrame into a Pandas DataFrame, which made it easier to do some time-series and statistical analysis. I did the same for the tweet data, which was also stored in MongoDB. I filtered the tweets by time and company name so I could later match them to the correct stock data for sentiment analysis.

After that, I used Spark to calculate the average sentiment per day. Then I joined this daily sentiment with the daily stock prices. I moved this combined data into Pandas so I could analyze it more deeply and check if there was a correlation between public sentiment and stock prices.

In the end, I calculated the correlation between the sentiment and the stock closing prices for each company. I also used Matplotlib to make graphs that show how sentiment and prices moved over time. This helped to better understand the relationship between what people are saying online and what’s happening in the stock market.

### 

### **1.1 Why I Chose MongoDB**

To decide which database to use for the rest of the project, I ran some tests using a tool called YCSB. I tested MongoDB and MySQL using three types of workloads:

* Workload A: a mix of reading and updating
* Workload B: mostly reading
* Workload C: only reading

Here are the results:

| **Metric** | **Workload A** |  | **Workload B** |  | **Workload C** |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | MongoDB | MySQL | MongoDB | MySQL | MongoDB | MySQL |
| RunTime (ms) | 1168 | 4089 | 1033 | 1554 | 663 | 972 |
| Throughput (ops/sec) | 856.16 | 244.56 | 968.05 | 643.50 | 1508.30 | 1028.81 |

From these numbers, it's clear that MongoDB was faster in every case. For example, in Workload A, MongoDB took only 1168 milliseconds, while MySQL took 4089 milliseconds. Also, MongoDB could handle more operations per second (higher throughput), which means it can deal with more work in less time. This made MongoDB a better option for this project.

Also, MongoDB is good at storing semi-structured data, like tweets and stock data, which don't always have the same format. This made it easier to work with, especially when using it with Spark.

After processing, I saved the cleaned and transformed data back to MongoDB, and also exported it as CSV files so I could use them in other libraries, models and generate forecasts.

This setup shows a complete big data pipeline, with both batch and real-time processing. I also tested streaming Reddit comments using a Python script that sends comments to Spark, which filters them and saves the relevant ones to HDFS in CSV format. That part is explained later.

Overall, this architecture uses popular and powerful tools like Spark, MongoDB, and HDFS. It shows how different technologies can work together to collect, process, and analyze large amounts of data.

**2. Sentiment Analysis**

As informed above I combined the tweet data. I will work with text data as sentiment analysis, that is now often used in financial forecasting. It helps us understand the emotional tone and opinions people share on social media, in the news, and in other places. These emotions can influence how investors behave and what they expect from the market. Many studies have shown that using this kind of data, especially from X (Twitter) we can improve how well financial models predict stock prices (Bollen, Mao, & Zeng, 2011).

To make sure the tweet data could be used correctly with the stock price data, I changed the date column in the tweets to a proper datetime format. This allowed me to group the tweets by day and match them with daily stock prices. It also made it easier to filter and calculate average values for each day, which is important for combining the tweet sentiment with time series models.

To measure the sentiment (positive or negative feeling) of Tesla-related tweets, I used a libary called TextBlob. This is a simple Python library that’s easy to use and works well for basic text analysis. TextBlob gives two main scores: polarity, which shows if the text is negative (-1) or positive (+1), and subjectivity, which shows if the text is more opinion-based or fact-based. In this project, I only used the polarity score to find the average daily mood in Tesla tweets.

There are more advanced tools for sentiment analysis, like VADER or AI models such as BERT, which can give better results with complex text. However, I chose TextBlob because it’s easy to use and was good enough for this project. It has been used in many studies and is a good option when working with smaller datasets. (Loria, 2018; Kumar & Jaiswal, 2020).

**3. Preparing Data for Neural Network**

Before training the neural network model, it was necessary to prepare the dataset in a way that the model could understand and learn from. One important step was to normalize the data using a MinMaxScaler, which transforms both the stock closing prices and the sentiment scores to a range between 0 and 1. This is done because neural networks are sensitive to the scale of the input features. If the values in one column are much larger than the other, the model may focus too much on one feature and ignore the other. By scaling both features to the same range, we help the model learn more effectively.

The dataset was then organized into sequences, where each sequence included data from the previous 28 days. The choice of 28 days, which is approximately one trading month, was made to allow the model to learn patterns over a meaningful short-term period. For each sequence, the model was trained to predict the closing price on the following day based on the past 28 days of closing prices and sentiment values.

After preparing the input and output data, the sequences were converted into a format that could be used by the PyTorch neural network. This involved turning the data into tensors, which are the basic data structure in PyTorch. These tensors were then grouped into a dataset and loaded into a DataLoader. The DataLoader feeds the data into the model in small batches, in this case 32 sequences at a time, which helps the training process run more efficiently. Importantly, the data was not shuffled, because this is a time series problem where the order of the data matters. Shuffling would break the time sequence, which would make it harder for the model to learn how the past affects the future.

**4. Long Short-Term Memory (LSTM) model**

A Long Short-Term Memory (LSTM) model was used for the prediction of Tesla stock price on the basis of past price values and tweet sentiment information. LSTM is a class of recurrent neural network ideally suited for time series forecasts because it can learn patterns and dependencies over time. Standard RNNs face the vanishing gradient problem and thus become impractical when trying to remember long sequences; in contrast, LSTMs remember longer sequences using memory cells with gating mechanisms (Hochreiter & Schmidhuber, 1997). This renders LSTM models very much applicable for financial use, wherein both recent as well as old data points can impact the future values.

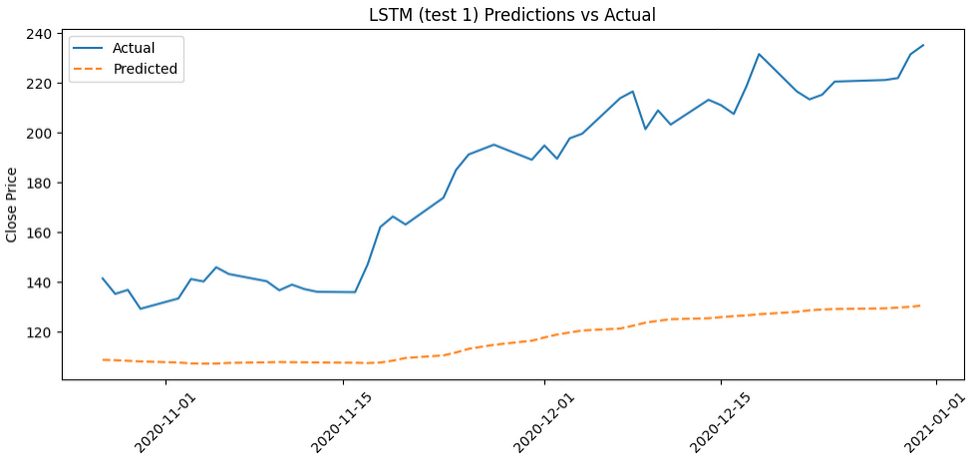
The LSTM model took two input features: the normalized closing price and the daily sentiment score. These inputs were fed into the model in the form of sequences of 28 time steps (i.e., 28 days). The model was defined with two layers of LSTM and had a hidden size of 64 units, which is a standard setup that captures moderate complexity without overfitting. No dropout was added since the dataset size was manageable and the main objective was to learn sequences.The Adam optimizer was then used to update model weights efficiently, with the learning rate set to 0.001, which is considered a general starting point for time series tasks.

The model was trained for 20 epochs with a batch size of 32, with training performed on a GPU whenever one was available. In every epoch, the model attempted to minimize the distance between its prediction and the actual stock prices. After training, the model was tested on a separate test set so that the generalizability to unseen data could be evaluated.

The selection of LSTM is backed by previous research in the financial time series application evidencing the model to be an efficacious choice. For instance, Fischer and Krauss (2018) found that LSTM models perform better than usual machine learning models in predicting stock returns. Similarly, Nelson et al. (2017) showed that LSTM models capture the nonlinear and temporal dynamics of stock price movements much better than standard models.

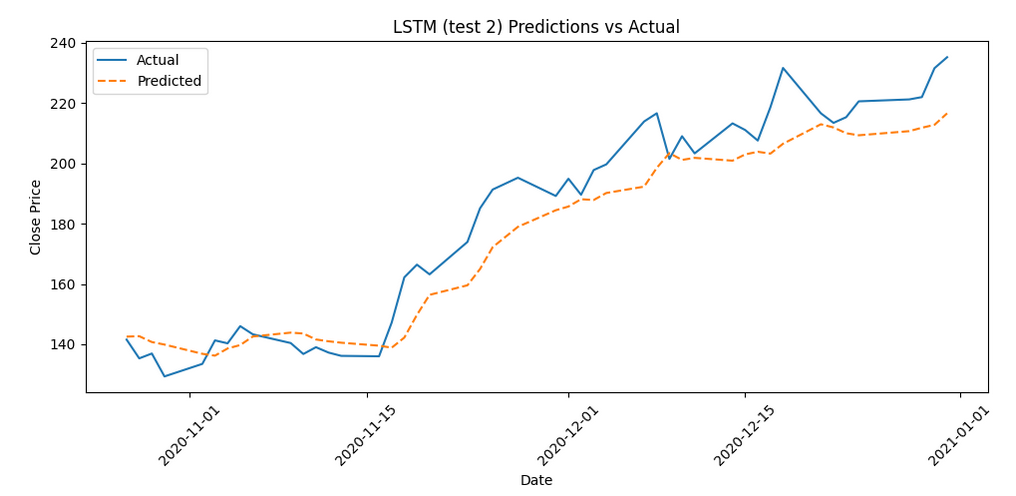
The first LSTM test got:

MSE: 3245.342773 R²: -1.6843



To improve the performance of the LSTM model, I implemented a basic grid search to test different combinations of hyperparameters, including the number of layers, hidden units, dropout rates, learning rates, and training epochs. This helped identify the configuration that resulted in the lowest mean squared error on the test

MSE: 79.746460 R²: 0.9340



Due I got a pretty good score in LSTM, I will try another model.

**5. Multilayer Perceptron (MLP)**

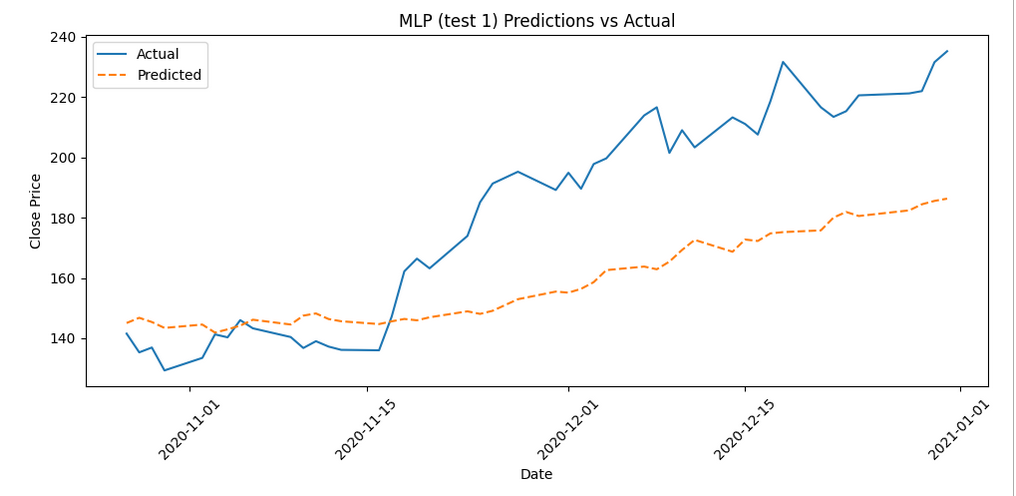
Apart from the LSTM model, a Multilayer Perceptron (MLP) was implemented to act as a baseline to evaluate performance across different types of neural networks. MLPs are among the most basic and commonly studied architectures in the so-called deep-based paradigm for learning. They consist of fully connected layers, and typically they do not consider modeling time dependencies between the input and output, but rather shape very sophisticated non-linear relationships between the two (Goodfellow, Bengio & Courville, 2016).

While MLPs were not intentionally engineered for time series data, it can still be used for forecasting problems whenever the input sequences are flattened as fixed length feature vectors. In particular, in this project, each 28-day sequence of stock price and sentiment score was reshaped to be a single vector to be fed into the MLP. Such a method discards all temporal ordering information but allows the model to learn generalized patterns within fixed size input.

The MLPs had two hidden layers, one with 64 neurons and the other with 32, while both used ReLU activation functions. Such sizes were chosen so as to provide enough capacity for data modeling without overfitting. A final output layer provided a single predicted value, which was the closing price for the next day. The model used the Adam optimizer, a 0.001 learning rate, and the Mean Squared Error (MSE) as the loss function, along with 30 training epochs.

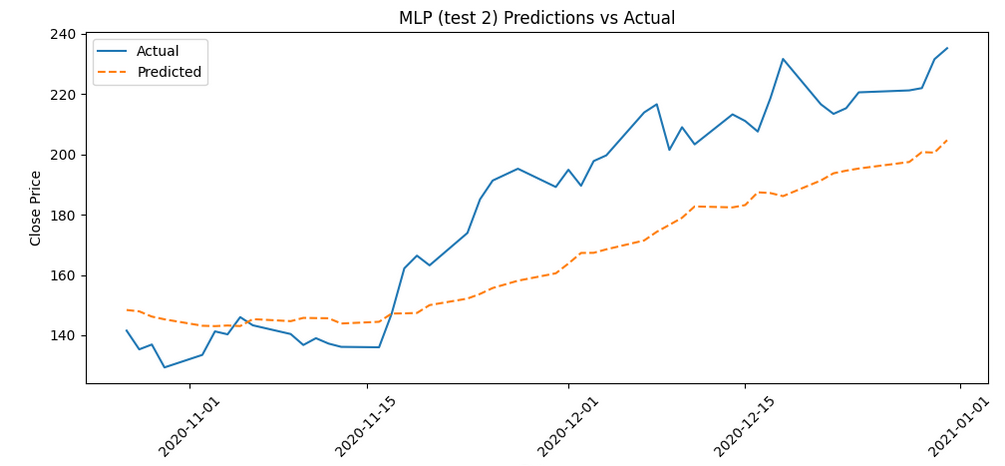
MLPs, lacking memory mechanisms such as those found in LSTM networks, are computationally efficient and are put in as a strong baseline in many forecasting tasks. Research shows that MLPs can indeed still achieve competitive results when structured well for time series prediction, particularly when the dataset is not outrageous in size and the problem does not lend itself to modeling long-range dependencies (Zhang, Eddy Patuwo & Hu, 1998).

MLP Test MSE: 1004.363525 R²: 0.1693



To improve the performance of the MLP model, I tested different configurations of hyperparameters such as the number of hidden layers, the number of neurons in each layer, and the number of training epochs. Although a full grid search was not performed, several combinations were tried to find a structure that balanced learning ability and model simplicity. This process helped identify a configuration that produced stable training and a low mean squared error on the test data.

MLP Test MSE: 725.071167 R²: 0.4003



**6. ARIMAX**

Alongside neural network models, there are ARIMAX (AutoRegressive Integrated Moving Average with Exogenous Variables) models as a traditional statistical approach for time series forecasting. ARIMAX is an extension of standard ARIMA, including the external predictor-known as an exogenous variable. In this case, that is the daily sentiment score. It allows the model to learn from past values of the time series (stock prices) but, on the other hand, it also takes into account external influences that can sway the market behavior, such as public sentiment.

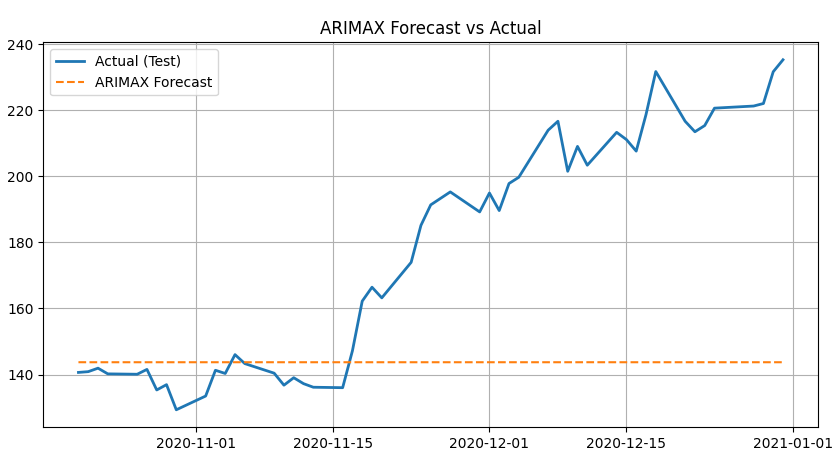
ARIMAX is an ideal case when the time series is autocorrelated and influenced by outside variables. It has often been used in financial forecasting because it is clear-cut, easy to interpret, and able to model linear dependence (Box, Jenkins& Reinsel, 2008). ARIMAX, in simple terms, spells out how much explains the future value based on past values and external inputs in contrast to black-box models of neural networks.

Before applying the ARIMAX model, the time series was differenced once to make it stationary. This is important because ARIMA-based models assume constant statistical properties over time. The original stock price data showed a trend, which violates this assumption. After one round of differencing, the series appeared stable and ready for modeling.

The ACF plot shows a significant spike only at lag 1, followed by values that quickly fall within the confidence bounds. This suggests that a **moving average (MA) term of order 1** is appropriate. Similarly, the PACF plot shows a significant value at lag 1, with no clear pattern beyond that, indicating that an **autoregressive (AR) term of 1** may also be sufficient. Together, this supports choosing a simple ARIMAX model with parameters **(p=1, d=1, q=1)** after first-order differencing.

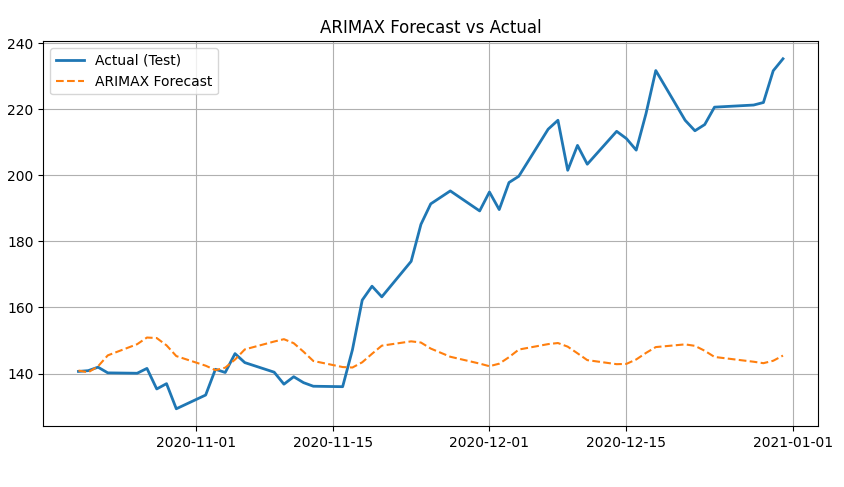
**6.1 Fair comparison**

Although ARIMAX models can be trained using the full dataset, I applied the same train-test split used for the neural network models to ensure a consistent and fair comparison. This approach allowed all models to be evaluated on unseen data, using the same performance metrics under the same conditions.

MSE: 2352.980229 R²: -0.9058

It didnt improve even after trying best hyper parameter

MSE: 2226.948318 R²: -0.8037



When evaluating the ARIMAX model, both the in-sample predictions and out-of-sample forecasts appeared almost flat, indicating the model struggled to capture meaningful variability in stock prices. This may be due to the differenced series being overly smooth, causing the model to focus on the short-term mean rather than detecting fluctuations.

Despite testing multiple configurations, including ARIMAX(1,1,1) and (2,1,2), results remained largely unchanged. This suggests that the model structure, combined with a single exogenous input—average daily sentiment—was insufficient to represent the complex, non-linear nature of stock price movements. It is also possible that sentiment had limited or delayed influence on prices within this linear setup.

As a result, the model defaulted to near-constant predictions, highlighting a limitation of ARIMAX when applied to noisy financial time series. More flexible models like LSTM may be better suited for capturing such dynamics.

### **7. Model Performance Evaluation**

To compare how good different models are in predicting stock prices, I used LSTM, MLP, and ARIMAX models. I tested them with stock data from five companies: Tesla, Disney, British Airways, Amazon, and Apple. For each company, I tried two different setups (Test 1 and Test 2) to see how results change.

But the dataset was quite small, only around 245 rows for each company. Because of that, the forecasts from all models came out almost flat — not many changes or clear trends in prediction. That made it hard for the models to learn the real behavior of the stock prices.

Still, we can compare the models using two values:

* MSE (Mean Squared Error): Shows how far the prediction is from the real value. Lower is better.
* R² (R-squared): Shows how well the model explains the data. Closer to 1 is better.

#### Results:

* LSTM Test 2 usually gave the best R² scores — like 0.9088 for Tesla and 0.9197 for Disney — which shows the model could capture something from the data.
* MLP Test 2 also had good results for British Airways and Apple, with R² around 0.76 and 0.65.
* Amazon results were very weak for all models. R² was negative in most cases, meaning the predictions were worse than just guessing the average.
* ARIMAX performed the worst in general. It gave negative R² values for almost every company, sometimes even very low like -2.49 for Apple.

Even if the results look good in numbers for some companies, the actual forecasts were nearly flat, probably because the data was too small. LSTM and MLP gave better numbers, but with more data, they could perform better. ARIMAX didn't work well with this kind of dataset.

**8. Spark Streaming**

I used Apache Spark and Hadoop to collect and analyze Reddit comments about Trump in real time. I connected to Reddit using its API and sent the live comments through a socket to Spark Streaming. Spark is used because it can process streaming data fast and handle a large amount of text, which is important when collecting many comments from Reddit. Spark is good for filtering, cleaning, and transforming data quickly, and it works well with Python libraries like Pandas and TextBlob. I filtered the comments to keep only the ones that mention "Trump", then save the result to CSV files. These files are stored using Hadoop Distributed File System (HDFS), because Hadoop is good for storing big data across many machines. Hadoop helps to keep the data safe and available even if some parts of the system fail. After the data is saved, we use TextBlob to do sentiment analysis, checking if the comment is positive, negative, or neutral. Finally, we show the results in a bar chart. Using Spark and Hadoop together makes the system strong, fast, and ready for big or growing data (Zaharia et al., 2012; White, 2012).

**References**

Bollen, J., Mao, H. and Zeng, X., 2011. *Twitter mood predicts the stock market*. Journal of Computational Science, 2(1), pp.1–8.<https://doi.org/10.1016/j.jocs.2010.12.007>

Box, G.E.P., Jenkins, G.M. and Reinsel, G.C., 2008. *Time Series Analysis: Forecasting and Control*. 4th ed. Hoboken: Wiley.

Fischer, T. and Krauss, C., 2018. *Deep learning with long short-term memory networks for financial market predictions*. European Journal of Operational Research, 270(2), pp.654–669.<https://doi.org/10.1016/j.ejor.2017.11.054>

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning*. Cambridge: MIT Press.

Hochreiter, S. and Schmidhuber, J., 1997. *Long short-term memory*. Neural Computation, 9(8), pp.1735–1780.<https://doi.org/10.1162/neco.1997.9.8.1735>

Kumar, A. and Jaiswal, A., 2020. *Sentiment Analysis: A Comparative Study of TextBlob, VADER, and Word2Vec on IMDB Dataset*. International Journal of Scientific & Technology Research, 9(2), pp.1016–1019.

Loria, S., 2018. *TextBlob: Simplified Text Processing*. [online] Available at:<https://textblob.readthedocs.io> [Accessed 24 May 2025].

Nelson, D.M.Q., Pereira, A.C.M. and de Oliveira, R.A., 2017. *Stock market's price movement prediction with LSTM neural networks*. In: 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017. IEEE, pp.1419–1426.<https://doi.org/10.1109/IJCNN.2017.7966019>

Rao, T. and Srivastava, S., 2012. *Twitter Sentiment Analysis: How To Hedge Your Bets In The Stock Markets*. arXiv preprint arXiv:1212.1107.<https://arxiv.org/abs/1212.1107>

White, T., 2012. *Hadoop: The Definitive Guide*. 3rd ed. Sebastopol, CA: O’Reilly Media. Available at: <https://www.oreilly.com/library/view/hadoop-the-definitive/9781449311520/>

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S. and Stoica, I., 2012. *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*. In: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI). USENIX Association. Available at: <https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf>

Zhang, G., Eddy Patuwo, B. and Hu, M.Y., 1998. *Forecasting with artificial neural networks: The state of the art*. International Journal of Forecasting, 14(1), pp.35–62. <https://doi.org/10.1016/S0169-2070(97)00044-7>