SpiderNet's LSTM to Predict Stock Market Prices

Savannah Stephenson, Janavi Bahalala, Christian Duff, Allison Adams, Alicia Mares

***Abstract*—The stock market is a complex and dynamic system, driven by a number of factors including economic indicators, investor behavior, and global events. To address the problem of the inherent volatility and unpredictability of the stock market, the authors will apply AI models to predict stock trends, which may potentially offer a more reliable system of prediction. In this paper, the authors will explore existing research in the field of stock market prediction, outline the team's approach to the problem, give an overview of the methodology used for their AI model, and present their findings on the effectiveness of their AI model in predicting the stock price of individual stocks.**

I. Introduction

THE stock market is a complex and dynamic system, driven by a number of factors including economic indicators, investor behavior, and global events. Predicting its fluctuations is a challenge; traditional methods of stock market prediction often rely on historical trends, statistical analysis, and expert opinion, which can be subject to human biases and errors. However, recent advancements in artificial intelligence (AI) present new opportunities to approach stock market forecasting in a more coherent and unbiased manner.

The authors' project focuses on developing an AI-based time series prediction model for Apple stock prices. The authors will utilize deep learning techniques to analyze historical stock price data from the online dataset repository Kaggle, use TensorFlow in Python to set up and train the model, and use Google Colab for development. The output of the model will be a single numerical value representing the low stock price prediction for that day. The authors will then visualize these predictions against actual stock market values, and provide an assessment of the model's accuracy.

The motivation for utilizing AI in stock market predictions stems from its ability to detect complex patterns that may not be immediately apparent to human analysts, potentially offering a more reliable prediction system. By avoiding emotional biases and increasing the consistency of predictions, AI can provide a wealth of insight to traders, investors, and financial institutions. The authors will address the problem of inherent volatility and unpredictability of the stock market, where applying AI models such as Long short-term memory (LSTM) or Artificial neural networks (ANN) can help improve accuracy in predictions.

This approach holds significance in a world where financial markets have global impacts, and even slight improvements in prediction accuracy can turn into significant financial benefits. In this paper, the authors will explore existing research in the field of stock market prediction, outline the architecture used for their AI model, and present their findings on the effectiveness of their AI model in predicting the stock price of Apple.

II. Related Work

THIS section discusses research done in stock market prediction (SMP) that the authors did before attempting their own prediction model. The knowledge from the papers is grouped into the following categories: Common Methodologies in Stock Market Prediction, Fundamental, Technical, and Sentiment Analysis, Data Representation and Sourcing, and LSTMs in Stock Market Prediction.

A. Common Methodologies in Stock Market Prediction

The stock market is a crucial component of any economy in the world. As such, the prediction of stocks has been researched and written about for nearly as long as the stock market itself, because accurately forecasting stock market movements can enable investors to make more informed and strategic decisions [4]. With so much literature out there, naturally, there have been surveys done over stock market prediction in order to collect and dissect the trends in the field. N. Rouf et al performed a decade-long survey on methodologies, recent developments, and future directions for stock market prediction (SMP) that incorporated machine learning techniques [4]. N. Rouf et al found that many different algorithmic approaches have been used in SMP, listing the Support Vector Machine (SVM), k Nearest Neighbors (KNN), Artificial Neural Networks (ANN), Decision Trees, Fuzzy Time-Series, and Evolutionary Algorithms as examples. In another survey of SMP done by R. Ray et al, a similar list of models was provided with Support Vector Machines (SVM) and Artificial Neural Networks (ANN) at the focus of their survey as the most preferred models [6]. A systematic review of systematic reviews of Artificial Intelligence (AI) models applied to stock market prediction conducted by Chin Yang Lin et al found that support vector machines (SVM), long short-term memory (LSTM), and artificial neural networks (ANN) are the most popular AI methods for stock market prediction. Similarly N.Rouf also narrowed down their options, concluding that SVM is the most popular technique used for SMP. However, techniques like ANN and Deep Neural Networks (DNN) are mostly used, as they provide more accurate and faster predictions [4]. The ANN is a brain-inspired technique in which a large number of artificial neurons are strongly interconnected in order to solve complex problems [4]. DNNs are an improvement on neural networks where more hidden layers and neurons are added for automatic feature extraction and transformation [4]. Moreover, N. Rouf et al [4] wrote that when comparing three Recurrent Neural Network (RNN) models including a basic RNN, a Gated Recurrent Unit (GRU) and an Long Short Term Memory (LSTM), the results revealed that LSTM outperformed other techniques and achieved an accuracy of 72% on a 5-day horizon. R. Ray et

al [6] found that the highly volatile and non-linear patterns in the stock market can only be effectively handled by ANNs and their successor neural networks (like LSTMs) rather than other traditional machine learning techniques. The authors of the systematic review of systematic reviews [2] also reported that LSTM networks have the advantage of capturing the context of the data as they are being trained and can address the gradient vanishing problem, making them the ideal approach for time series forecasting. This review of methodologies steered the authors toward the LSTM model, which will be discussed further in the LSTMs in Stock Market Predictions section later on.

B. Fundamental, Technical, and Sentiment Analysis

Moving on from the models used for SMP, it is also important to consider the different types of analysis that can be done with data relating to the stock market. The stock market is dependent on various parameters, such as the market value of a share, the company's performance, government policies, the country's Gross Domestic Product (GDP), the inflation rate, natural calamities, and so on [4]. Historically there are two theories for predicting stock market movements (i) Efficient Market Hypothesis (EMH) and (ii) Random Walk Approach. Efficient Market Hypothesis was proposed by economist Eugene Fama in 1970. This hypothesis states that the current market itself contains all the information about it and if new information is gathered then it is absorbed by the market and gets reflected in its price. It concludes that the stock market cannot be predicted by any other means by an investor. Efficient Market Hypothesis is classified into weak EMH, semi-strong EMH, and strong EMH. The weak EMH tries to predict the market with past historical data alone. Semi-strong EMH predicts the market using historical data and currently available public information. Strong EMH predicts the market using public information and some private information that is available outside the market. The Random Walk Theory states that future stock prices cannot be predicted by historical and current stock prices since they are highly volatile and independent of each other. Additionally, there is another theory named Inefficient Market Hypothesis (IHM); this theory states that there are other factors that can be used to leverage the future movement of stock prices. Fundamental and Technical Analysis are traditional methods used in finance that involve analyzing a company's financials and stock trends. However, there are three ways of stock market prediction: (i) Fundamental Analysis, (ii) Technical Analysis, and (iii) Sentiment Analysis. Fundamental analysis uses more market information like an annual report, company balance and auditor's report k to predict stock for the company [6]. Technical analysis uses only historical stock price data of the company using time series charts [6]. Sentiment analysis deals with the general public's concerns, beliefs, emotions, perceptions, and sentiments towards stocks, it is the process of analyzing text corpora, e.g., news feeds or stock market-specific tweets, for stock trend prediction [4]. When considering the feasibility of each type of analysis the authors concluded that technical analysis was the most feasible due to the type of data available, discussed further in Data Representation and Sourcing, and the Methodology section of the paper. This was conducive to trends seen through the literature review with Chin Yang Lin et al [2] finding that Fundamental Analysis is less frequently discussed in the SMP literature with Technical Analysis accounting for 66% of the studies examined, while Fundamental and Combination Analyses accounted for 23% and 11%, respectively.

C. Data Sourcing, Split, and Representation

Both the surveys of SMP work and the individual cases that the authors examined had some references to how data was sourced and then represented to the model. Sources for the data given to models was varied with papers listing the following sources for their numerical data: Taiwan Stock Exchange Capitalization Weighted Stock Index (TAIEX), Yahoo Finance, and the New York Stock Exchange (NYSE), and Kaggle. Chin Yang Lin et al found that most authors choose Yahoo Finance as the informational source because they could easily extract the data that was relevant by using a Python module [2]. The Yahoo Finance data includes numerical values such as the open, close, mid, high, low price, and volume values for a series of days without missing samples. M. Akhtar et al [3], however, used a dataset from Kaggle due to its size and security. The same dataset was used across three different model types SVM, Forest Algorithm, and LSTM. Authors of the review of systematic reviews [2] found that most SMP research utilizes periods of 1000 days, which can be easily handled by most machine-learning algorithms. There is an example of this generalization in A. Moghar's work [8] where the data utilized consisted of the daily opening prices of two stocks in the New York Stock Exchange (NYSE) extracted from Yahoo Finance with ranges spanning far longer than 1000 days. The separation of data described in A.Moghar's LSTM Recurrent Neural Network [8] was a classic 80/20 split, 80% of data for training and the other 20% of data for testing which is an industry standard for models. Beyond the data itself and the amount of it used for each part of the model there is also the problem of how the data will be presented to the model. In most of the approaches the authors read about, the data was fed into the model in its numerical form as given, although perhaps with some massaging in order to fill gaps or order the data in a particular way. However, it is possible to use a symbolic representation of the time series as discussed in S. Elsworth et al's Time Series Forecasting Using LSTM Networks: A Symbolic Approach [7]. The key idea behind symbolic representation is to convert the numerical time series T = [t1, t2, . . . , tN] into a sequence of symbols S = [s1, s2, . . . , sm] where each symbol si is an element of a finite alphabet A = a1, a2, . . . , ak [7]. The authors found that a symbolic approach can lead to significant speed up of the training phase without degrading the forecast accuracy, whilst reducing the sensitivity to certain hyper parameters [8]. However, this was the only source where a symbolic approach was observed and as it is a more complex approach it was deemed inappropriate for the authors first attempt at a SMP model.

D. Common Evaluation Metrics

Evaluation metrics are essential for AI models because they provide a standardized way to assess a model's performance and help determine its suitability for specific tasks. In machine learning (ML) algorithms, a multitude of metrics are available to measure the models' performance. In S. Mokhtari's research on the effectiveness of artificial intelligence in stock market prediction based on machine learning, a variety of potential evaluation metrics are listed, including confusion matrices, receiver operator characteristic (ROC) curves for classification models, and R-squared, explanation variation, mean absolute percentage error (MAPE), root mean squared error (RMSE), and mean absolute error (MAE) for regression [8]. The research from the systematic review of systematics reviews indicated that Mean Squared Error (MSE) was the most frequently employed across the SMP domain, appearing in 42% of the 12 studies, closely followed by Accuracy and Mean Absolute Error (MAE), both of which were used in 33% of the 12 studies [2]. Further detail on evaluation metrics equations will be discussed in the methodology portion of the paper. Regarding the evaluation metrics that were observed in the research done by the authors; S. Elsworth et al used the mean squared error (MSE) loss function for the raw LSTM model, and a categorical cross entropy loss function for the ABBA-LSTM model (the model with symbolic data) [7] and A. Moghar used mean squared error for their LSTM Recurrent Neural Network [1], at least partially proving the trend of MSE being the most frequently employed evaluation metric.

E. Additional Techniques for Improvement

1) Regularization Methods: Regularization methods are critical in machine learning and deep learning because they address a common problem: overfitting. Overfitting occurs when a model learns to perform exceptionally well on the training data but fails to generalize to unseen data. This is particularly relevant to our project because we are training on a single stock and only utilizing numeric data rather than any sentiment analysis. This section will discuss a number of regularization techniques.

In R. Gencay and Min Qi's paper titled "Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging," they discuss a variety of regularization techniques [13]. Starting with early stopping, it is a regularization technique that works by monitoring the model's performance on a validation set during training and halting the training process when the performance stops improving, this point can be seen in Figure 1.

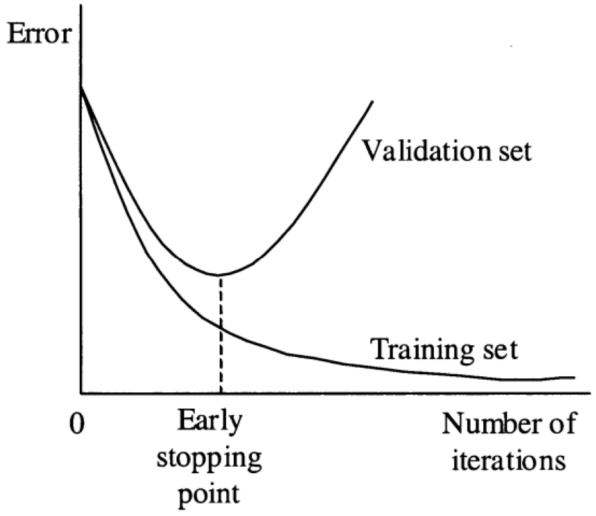

Fig. 1. Early stopping based on cross-validation.

The error on the validation set is monitored during the training session in early stopping [13]. The validation error will normally decrease during the initial phase of training (see Fig. 1), as does the error on the training set. However, when the model begins to overfit the data, the error on the validation set will typically begin to rise. When the validation error increases for a specified number of iterations, the training is stopped, and the weights at the minimum of the validation error are returned [13]. A paper titled " ModAugNet: A new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module" written by Yujin Baek and Ha Young Kim, found that although the early stopping method can prevent overfitting, it cannot improve generalization performance [14]. Reducing the neural network size can also prevent overfitting, but it does not improve generalization performance because larger and deeper networks can solve more complicated problems [14]. [13] also discusses bagging and bayesian regularization and ultimately found that all the methods discussed effectively brought down overfitting. We focused on discussing early stopping here because we found it to be the most easily understood method and because it was discussed in multiple papers that we looked at.

Authors Saud, Arjun Singh, and Subarna Shakya wrote a paper titled"Analysis of l2 regularization hyperparameter for stock price prediction." Journal of Institute of Science and Technology 26.1 (2021): 83-88 [12]. L2 regularization is one of the most popular and widely used regularization techniques [12]. L2 regularization adds a penalty term to the loss function of a model, proportional to the sum of the squared values of the model's weights. L2 regularization penalizes large weights, which helps the model generalize better to unseen data, reducing overfitting. L2 regularization also tends to distribute weights more evenly across the model rather than allowing a few weights to dominate. Neural networks like LSTMs are highly flexible and prone to overfitting, making L2 regularization useful. The regularization hyperparameter (λ) is

one key parameter to be optimized for a well-generalized machine learning model [12]. Hyperparameters can't be learned by machine learning models during the learning process. We need to find their optimal value through experiments. [12]'s research work analyzed the L2 regularization hyperparameter used with a gated recurrent unit (GRU) network for stock price prediction. The authors experimented with five stocks from the Nepal Stock Exchange (NEPSE) and observed that stock price can be predicted with lower mean squared errors (MSEs) when the value of λ was around 0.0005. Therefore, their research paper recommended using λ=0.0005 with L2 regularization for stock price prediction.

2) Learning Rate: In the training process of the deep neural networks, the learning rate plays an important role in whether the training process can converge and how fast it can achieve convergence.

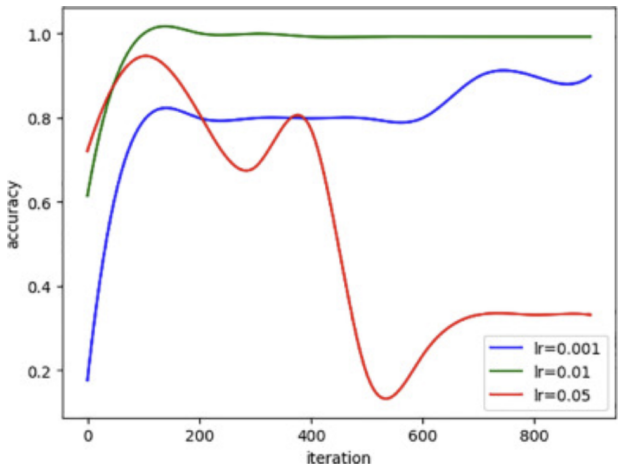

Fig. 2. Comparisons between different learning rates from [15].

Fig. 2 shows the effect of different learning rates on neural network training. If a small learning rate is chosen, it can guarantee that the local minimum is not missed, however, it also means that the network takes a long time to reach the convergence state, especially when the training is trapped in the saddle point. In the context of optimization and machine learning, a saddle point is a point in the loss surface where the gradient is zero (like a minimum or maximum), but the point is neither a true local minimum nor a maximum. Instead, it behaves like a minimum along one dimension and a maximum along another. Saddle points can significantly slow down optimization. Advanced optimization techniques help models escape these points and converge more effectively on the loss surface. On the contrary, if a large learning rate is used, the fluctuation of the loss during training will be relatively large, and it is even difficult to converge in the end. One type of learning rate adjustment strategy is the adaptive learning rate, which means that during the training process, different learning rates are assigned to each parameter according to the current state of the model [15].

3) K-fold Cross Validation: Authors Ogundunmade, Adepoju, and Allam wrote a paper titled Stock Price Forecasting: Machine Learning Models with K-fold and Repeated Cross Validation Approaches [10]. The goal of this paper was to determine if stock market prediction models performed better or worse when utilizing K-fold and K-fold Cross Validation (CV) methods. The authors felt that although various algorithms have been developed since the introduction of Artificial Intelligence (AI) and have been used to forecast equities market movement less attention has been paid to the use of Cross Validation (CV) approaches for better stock price prediction [10]. In our research we noticed this trend as well, none of the papers we have looked into so far used k-folding, choosing instead to split their data traditionally, with 80% for training and 20% for testing and validation. Cross-validation is a resampling strategy for assessing AI models on a sample of data [10]. The interaction incorporates just a single parameter, k, which indicates the number of groups into which a given dataset ought to be divided. Thus, the cycle is oftentimes referred to as k-fold CV. Whenever an exact value for k is determined, it is substituted for k in the model's reference, for instance, k=10 for 10-fold cross-validation. The general process of k-fold CV is as follows: the data is divided into k equally sized folds, the model is trained on k-1 folds and validated on the remaining fold, and this process is repeated k times, with each fold used once as the validation set. This method ensures that every data point is used for both training and validation, reducing the risk of overfitting and underfitting and providing a more robust estimate of the model's performance [10]. In [10]'s paper the machine learning models considered are the linear regression model, SVM, RF, ANN and the Classification and Regression Trees (CART) model. They found that repeated K-fold and K-fold CVs produced favorable forecast performance compared to models with no CV technique involved in modeling stock price and went on to "recommend the use of CV technique in modeling stock exchange price" [10]. When it comes to choosing the k value, Isaac Kofi Nti, Owusu Nyarko-Boateng, and Justice Aning wrote a paper titled "Performance of Machine Learning Algorithms with Different K Values in K-fold Cross-Validation" [11]. They found that in selecting the value of k, "one needs to be cautious" because a lower k value is less expensive in terms of computational complexity, produces less variance, however, includes more bias. In contrast, a higher value of k is computationally expensive but has more variance and lower bias. They concluded that "it is of essence to perform experiments with different k values for a given dataset and algorithm to find the optimal k value for accuracy improvement" [11]. LSTM (Long Short-Term Memory) networks are often used for sequential data, like time series, where the order of data matters. The team thought that K-fold Cross Validation might be beneficial because sequential data often has patterns that can lead to overfitting. For example, Apple stock prices in the 1900s were much lower and increased much slower compared to the more current stocks. Cross-validation helps detect overfitting by ensuring the model is evaluated on unseen data multiple times. Additionally, since we are working with limited data, k-fold ensures all data points contribute to both training and validation, which is especially important for deep learning models that need large datasets.

F. LSTMs in Stock Market Prediction

As discussed in the above sections LSTMs have been shown to perform well with stock market data due to their non-stationary nature. Stock market data exhibits trends and seasonality that LSTMs can handle due to its ability to remember information over extended periods [9]. ProjectPro.io and AnalyticsVidhya.com both have articles relating to SMP models that showcase the Sequential and LSTM modules provided by Tensorflow Keras that can be used to code an LSTM model in Python code, as well as Pandas library functions that can be used to preprocess the data [5] [9]. These articles served as a great starting point for the authors to determine their methodology.

III. Methodology

THIS section reviews the team's methodology for the project including team organization, tools used by the team and a review of the model code.

A. Team Organization

During the research process, the authors focused on sharing information efficiently and accessibly. Each member of the team found a paper or article whose topic was similar to their proposed project. The papers were gathered by the team members individually in order to obtain a wide spread of authors and paper types. Each paper was then read and summarized for the group at large so that all the authors could gain a better understanding of the challenge they were tackling. The team member responsible for summarizing each paper took notes in a document stored in the team's shared Google Drive so that all members had access to the information even after the meeting where the papers were discussed. Following this broad literature search, the team assigned two different machine learning (ML) algorithms per team member in order to identify the best approach for our stock market prediction problem. Each team member summarized their assigned algorithm in a few sentences, outlining the knowledge of how it worked and what kind of tasks it has been utilized for in the past. Based on these findings, the team selected four model approaches that were deemed suitable for further investigation: Artificial neural network (ANN), Long Short-Term Memory (LSTM), Random Forest, and Linear Regression. Following this testing period, the authors selected the model architecture that they wanted to go forward with and then thoroughly reviewed the code to ensure each member understood what was being done. One of the authors was also tasked with editing the initial code with improvements, in order to get a model that was as accurate as possible. Following this work, the authors focused on documenting their progress.

B. Team Tools

The SpiderNet team used Python to code their AI Models due to the accessibility of multiple libraries in Python that are specifically designed for creating, training, and evaluating AI, including TensorFlow Keras and Scikit-Learn. Python offers graph-plotting libraries such as plotly and matplotlib, as well as data manipulation libraries such as numpy and Pandas that are invaluable when working with stock market data. The coding interface that the team used was Google Colab. Google Colab offers easy collaboration across users and through some of the author's past experience, it was known that colab has been utilized for AI models for similar projects. The team is also utilizing HTML and CSS for the project website, because they allow for high customization. Lastly, the team is using Overleaf to format their paper into IEEE formatting.

C. Dataset

SpiderNet's dataset was sourced from Kaggle, a web platform containing a giant collection of high-quality, already validated datasets. These datasets are curated for machine learning tasks, which is good because it reduces the initial overhead of cleaning the data. Kaggle is also freely accessible, which was a primary factor in out decision to utilize it. It offered a detailed, complete, updated, and easily accessible record of trading data for Apple (SpiderNet's selected stock to predict) so it was selected over paid platforms like Yahoo Finance or Google Finance.

The author's dataset (Fig. 4) consisted of 10,080 entries, spanning nearly 40 years of stock data, with the first entry dated September 7, 1984, and the last on September 6, 2024. Impressively, the entire dataset required only 602 KB of storage.. SpiderNet's data also contained six columns: Date, Open, High, Low, Close, and Volume. The Open column represents the stock price (in dollars) at which trading begins for the day. The High column represents the highest price at which the stock was traded during the day. The Low column represents the lowest price at which the stock was traded during the day. The Close column represents the final stock price at which trading ends for the day. Finally, the Volume column represents the total number of shares traded during the day.

This model focuses on predicting the Low price as a more stable indicator of the stock's daily performance. There was not any reference for this choice from other related work, as most papers we read didn't go greatly in depth on what they fed into the model. Only the Date and Low columns were used for prediction, to emulate the strategy of a trader or analyst focusing on the stock's minimum daily price as a less volatile measure of stock movement.

The authors removed the Volume column from the raw data, and it was not used in the training dataset. It does provide information on trading activity, but ultimately fails to influence stock price trends. It was unnecessary and was cut on the merit that it would introduce noise to the model. Furthermore, also removing Open, High, and Close reduces the risk that the model picks up on irrelevant patterns that deter from the goal of making accurate predictions.

	A	B	C	D	E	F
1	Date	Open	High	Low	Close	Volume
2	1984-09-07	0.100763	0.101999	0.0995464	0.100763	97676041.84
3	1984-09-10	0.100763	0.101071	0.0983401	0.100165	75812543.28
4	1984-09-11	0.101071	0.103814	0.101071	0.101999	178770477
5	1984-09-12	0.101999	0.102597	0.0989283	0.0989283	156171258.2
6	1984-09-13	0.104432	0.10473	0.104432	0.104432	243230959.4
7	1984-09-14	0.10473	0.108369	0.10473	0.105937	289611904
8	1984-09-17	0.108689	0.110204	0.108689	0.108689	226123797.6
9	1984-09-18	0.108689	0.109586	0.10473	0.10473	114152469.6
10	1984-09-19	0.10473	0.105937	0.102597	0.102597	124690010.7
11	1984-09-20	0.102896	0.103814	0.102896	0.102896	77625794.27

Fig. 3. Excerpt of contents of Kaggle's Apple stock data.

	A	B	C
1	Date	Low	Predictions
2	2016-09-01	24.583	24.636688
3	2016-09-02	24.8662	24.562134
4	2016-09-06	25.0247	24.54052
5	2016-09-07	24.92	24.567892
6	2016-09-08	24.4963	24.603132
7	2016-09-09	24.0048	24.584837
8	2016-09-12	23.8652	24.48088
9	2016-09-13	24.9599	24.330355
10	2016-09-14	25.2819	24.314068
11	2016-09-15	26.4144	24.405989

Fig. 4. Excerpt of contents of the valid_data DataFrame

D. Data Preprocessing

Chronological sorting was done before preprocessing to make sure the temporal sequence remained intact. Otherwise, the data would interpret unsorted data as chronological, which would throw off everything; leading to poor predictions. Although the data was already sorted by date from the Kaggle CSV file, the authors still converted it to a datetime format using pd.to_datetime, to guarantee temporal integrity.

This conversion was necessary to make sure the Date column is recognized as a datetime object. The authors used the Pandas DataFrame structure to preprocess the stock market data, which was integrated with numpy. SpiderNet's transformation method, scaler.fit_transform(), operates on DataFrames and its format aligns with the CSV file. This method helped the authors transition from the raw data to preprocessed data. The scaled_data numpy array can be seen in Fig 5.

The data was normalized by scaling to a range of [0, 1] using the MinMaxScaler from the sklearn library. This estimator scales and translates each feature individually so that it is in the given range on the training set (i.e. between zero and one). The main idea behind normalization is that variables measured at different scales do not contribute equally to the model fitting and model-learned function, which can lead to unintended bias in the results. Thus, to deal with this potential problem, feature-wise normalization was used prior to model fitting. The scaling was applied to the Low column of the dataset (the target variable for prediction). This enhanced the model's sensitivity to trends rather than numerical magnitudes, calming down the amount of noise and anomalies in the data (where spikes occur). As a result, the model became more accurate during training.

```
scaler = MinMaxScaler(feature_range=(0, 1));
scaled_data = scaler.fit_transform(data);
```

```
0.0001920036137785627
0.0001868271425223142 4
0.0001985459731230677 5
0.0001893512247513827 4
0.0002129686871181363 3
0.0002142474639019895
0.0002312363138592868 2
0.0002142474639019895
0.0002050943401437385 5
0.0002063774081248530 2
```

Fig. 5. Excerpt of contents of the scaled_data numpy array.

The authors did not use raw data, as that would risk introducing bias and misinterpretations of the data, leading to prediction inaccuracy. All data was reshaped, using the codes above, to conform with the input format required by the LSTM model.

E. Data Splitting

In order to test how accurate the model was in its ability to predict the Low stock price, the authors divided the data into training and testing/validating sets, with train_data including the first 8,064 rows (making up 80% of the dataset) and valid_data including row 8,064 onward (making up 20% of the dataset.) This is an incredibly standard ratio in machine learning; another common ratio is 70% for training and 30% for testing. The model learned from historical data, and its performance was validated using 'unseen' data. In other words, it makes for a ratio that ensures there is enough data to train, and enough unseen data to test the training on. Additionally, this split was done chronologically to maintain the temporal order of the stock price—shuffling would disrupt the sequence and make the model ineffective.

```
train_data = final_dataset[0:8064, :]
valid_data = final_dataset[8064:, :]
```

Referencing the code below, the authors used a sliding window technique to generate sequences for training the model and making sure there were no gaps in the training set. Each of the following sequences corresponds to a sliding window of a fixed length over this time-series data. This is to see any temporal dependencies—it looks for relationships between data points in the sequence; in other words, the value the value of one point will rely on previous points in the series. The window size figures out how many previous days the model uses to predict the next day's Low price. That way the data will stay sequential and best reflect how the stock behaves.

For the model, the authors set their number of days (window size) to 60, (days = 60) so that the first sequence included data points 1 to 60, with 61 being the target. The

sliding window logic is a loop that iterates from days to the length of train_data. It skips the first 60 rows because the very first prediction must have 60 previous days of data to pull from. In the loop, it appends a window of those 60 consecutive Low values to x_train_data, as well as appending the next day's Low value to y_train_data.

```
for i in range(days, len(train_data)):
x_train_data.append(scaled_data
  [i-days:i, 0])
y_train_data.append(scaled_data[i, 0])
```

The sliding window approach prevents overfitting by validating that each training sequence is unique and covers the entire dataset comprehensively. The model further aims to solve overfitting by utilizing Dropout layers after each LSTM layer. Each Dropout layer randomly removes 20% of the training data, which helps the model generalize by preventing reliance on too much of the same patterns. The 50 units per LSTM layer balances between getting enough context and avoiding getting too complex. The model's Dense layer produces a single output at the end that aligns with the prediction of the next day's Low price.

```
lstm_model.add(LSTM(units=50,
  return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(units=50))
lstm_model.add(Dropout(0.2))

  lstm_model.add(Dense(1))
```

F. Model Architecture

The final code for this project is designed to predict stock prices using a Long Short-Term Memory (LSTM) deep learning model. Data preprocessing, training, validation, and prediction stages are all included in the code workflow. The code uses a range of libraries to handle various tasks. The Pandas library is used for data manipulation, particularly for reading, cleaning, and managing tabular data from a CSV file. The NumPy facilitates efficient numerical computations and array handling. The matplotlib.pyplot is used to visualize the data and produce understandable and instructive graphs.

The Sequential model is used to construct a neural network layer by layer. Layers such as LSTM, Dense, Dropout, and Input help define the architecture of the LSTM model. This allows it to capture temporal patterns, make predictions, and apply regularization. Fig. 6 shows a diagram of the general architecture we used.

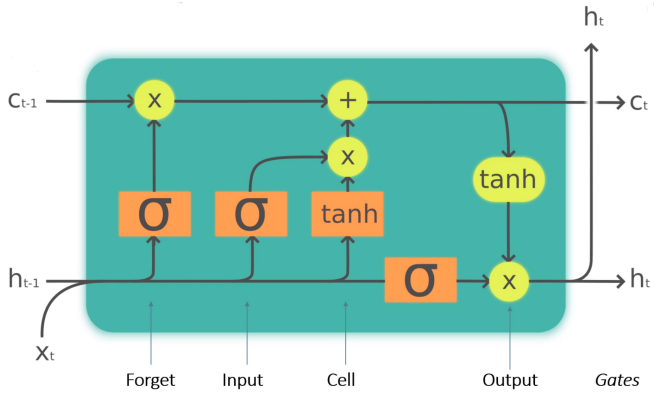

Fig. 6. Enter Caption

The dataset is split into training and validation sets with 80% of the data used for training and 20% reserved for validation. The MinMaxScaler is used to scale the Low column to a range of 0 to 1. The LSTM model then generates data sequences by utilizing the next data price as the output label and days of previous data as input. To adhere to the LSTM model's required format, the input and output arrays are transformed into NumPy arrays and reshaped.

The LSTM model itself is built using Sequential API from Keras. It starts with an input layer that can take sequences of scaled Low prices for 60 days. Then, two stacked LSTM layers with 50 units are added to capture time dependencies in the data. The code "return_sequences=true" in the initial LSTM layer allows a sequence output to be passed to the following layer. A forecast value is generated by the last dense layer, which represents the Low price for the following day. The model is compiled with mean squared error as the loss function, suitable for regression problems, and the adam optimizer for efficient training.

For testing and validation, the data points from the last days are extracted from the dataset, scaled, and reshaped to match the input format required by the LSTM model. The trained model is then used to predict future Low prices. These predictions are rescaled back to their original scale using the inverse transformation of MinMaxScaler. The validation data and predictions are combined in a DataFrame, ensuring that their lengths match. The results are visualized by plotting the training data, actual validation, and predicted values on the same graph to evaluate the model's performance.

G. Model Evaluation

To evaluate the performance of the proposed LSTM model, the authors employed a set of standard metrics commonly used in time series predicting identified through the literature review as well as the authors past experience. These metrics were chosen to provide a comprehensive assessment of the model's accuracy, robustness, and overall predictive performance. Below, the authors describe each metric and its relevance to the evaluation process:

1) Mean Squared Error (MSE): MSE is defined as the average of the squared differences between the predicted

and actual values. It is sensitive to large prediction errors, making it suitable for highlighting the model's ability to minimize significant deviations. The formula for MSE is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where y_i is the ground truth, $\hat{y}_i$ is the predicted value, and n is the number of observations.

2) Mean Absolute Error (MAE): MAE measures the average magnitude of errors in the predictions, providing a linear score that is less sensitive to outliers compared to MSE. The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAE is particularly useful for interpreting the model's performance in the context of absolute prediction errors.

3) Root Mean Squared Error (RMSE): RMSE is defined as the square root of the average of the squared differences between the predicted and actual values.The formula for RMSE is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the ground truth, $\hat{y}_i$ is the predicted value, and n is the number of observations.

4) R-squared (R^2): The R^2 metric evaluates the proportion of variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with higher values indicating better model performance. The formula for R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $y\bar{y}y$ is the mean of the actual values.

Future work for model evaluation would involve, in addition to these metrics, performing a comparative analysis of the LSTM model against baseline models such as feed forward neural networks. The same metrics would be applied to these baselines to ensure a fair comparison. To establish the reliability of the performance improvements, the authors want to conduct statistical significance tests on the metric scores across multiple evaluation datasets. This would ensure that the observed differences in performance were not due to random variation.

IV. Results and Discussion

SPIDERNET'S project demonstrated the feasibility of utilizing an LSTM-based AI model for predicting daily low stock prices for Apple. The model was trained using historical stock price data obtained from Kaggle, specifically ranging from Sept. 7th, 1984 - Sept. 6th, 2024. After implementing an 80/20 train-test split, the model's performance was evaluated using Mean Squared Error (MSE), Mean Absolute Error (MAE) and R-squared (R^2) as key evaluation metrics.

From the literature review, the authors had a picture of the types of models that would be appropriate for stock market prediction. However, before making a final decision on what model type that the authors wanted to revise and perfect, they felt it was prudent to code and test a selection of models effectiveness against their chosen dataset. Each team member selected a model from a list of models mentioned in their research, and the authors agreed to individually write the code for their chosen models. The models selected included Long Short-Term Memory (LSTM), Backpropagation using Artificial Neural Networks (ANN), Linear Regression, and Random Forest. Once the coding was complete, the authors compared the performance graphs of all models and chose LSTM. The authors selected LSTM firstly because the code written produced the best results, but also because of its ability to handle sequential data and capture long-term dependencies, which are common in time-series problems. Here are the graphs the authors got from the models that they selected:

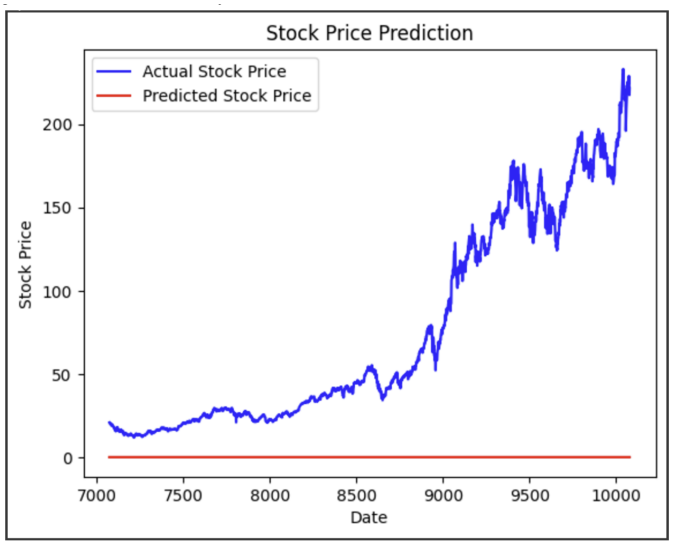

Fig. 7. Backpropagation using Artificial Intelligence (ANN

The method for constructing the backpropagation model might have been slightly rushed, leading to the odd prediction of only 0 that you see in Fig. 7.

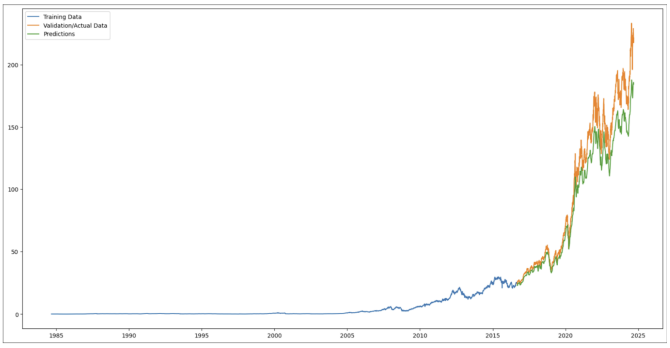

Fig. 8. Long Short-Term Memory (LSTM)

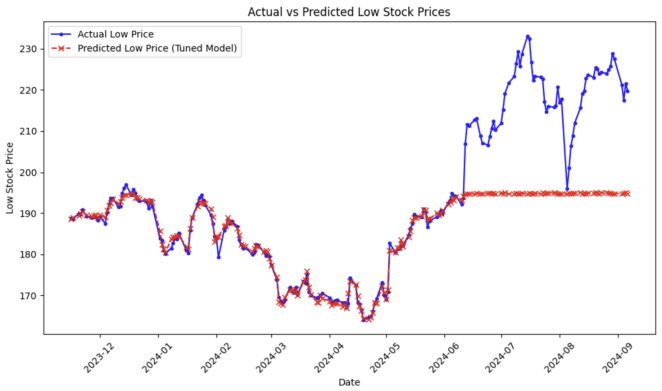

Fig. 9. Random Forest with training data at 98/2 split

Fig 9. uses a 98/2 data split, as that was when the best results were found. The available resources for constructing a Random Forest model were lower than thess for the LSTM which also contributed to the choice of model despite Random Forests also being used abundantly throughout the research for the project.

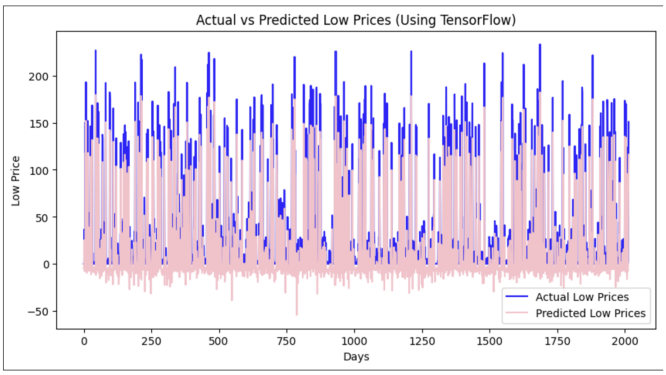

Fig. 10. Linear Regression

Below is a Fig. 11 showing the initial results from the team's evaluation metrics for the LSTM model described in Model Architecture.

Fig. 11. Initial results for LSTM model.

We chose to evaluate the model over multiple lead times because different lead times often have different levels of accuracy. For example, a model might be very accurate in predicting short-term outcomes (e.g., 1-day forecast) but less accurate for longer-term predictions (e.g., 10-day forecast). Evaluating the model across multiple lead times allows us to identify how performance degrades as the forecast horizon increases and to understand the model's behavior and limitations at different time horizons. Additionally, a model that performs well at one lead time might be overfitting or relying on specific patterns that only hold for that time frame. Evaluating over multiple lead times helps to identify whether the model is robust across time and ensures that the model is generalizing well to different forecast horizons, not just optimized for a single lead time.

Interestingly, a longer lead time does not always make the prediction worse. For example, with MSE 360 days there is a better prediction than 330 days. There are situations where more data or longer horizons can improve model performance, especially if long-term trends or seasonal patterns are present. And because LSTM is known to capture longer trends it is possible that sometimes there is a better prediction for a longer lead time because it falls into a more long-term trend that the LSTM is capturing.

Following these initial results, the team decided to incorporate three methods to enhance model performance and protect from over-fitting. These changes included K-fold Cross Validation, a learning rate reducer, and early stopping. K-fold cross-validation ensures robust performance evaluation by using all data for both training and validation, the learning rate reducer adjusts the training speed to prevent overshooting the optimal solution, and early stopping halts training when performance no longer improves, avoiding overfitting. Fig. 12 has a portion of the code for the learning rate reducer and early stopping.

```
# adding a learning rate reducer
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor=call_back_monitor, min_delta=0.001,
                                                 factor=0.1, patience=15, min_lr=0.00001)

# Defining the early stopping
early_stopping = EarlyStopping(monitor=call_back_monitor,
                               min_delta=0.001,
                               patience=25,
                               verbose=2,
                               mode='auto',
                               restore_best_weights=True)
```

Fig. 12. Code that was added for including a learning rate reducer and early stopping

Fig. 12 is the code that was added for including a learning rate reducer and early stopping. ReduceLROnPlateau adjusts the learning rate dynamically during training if the monitored metric (e.g., validation loss) stops improving. A smaller learning rate can help fine-tune the model during convergence.The monitor parameter, which specifies the metric to track, is set to "loss" (call_back_monitor="loss"). As for the other parameters min_delta=0.001 defines the minimum change in the monitored metric to qualify as an improvement, factor=0.1 reduces the learning rate by this factor. patience=15 waits for 15 epochs without improvement before reducing the learning rate and, min_lr=0.00001 sets the minimum possible learning rate to prevent it from becoming excessively small.

EarlyStopping halts training if the monitored metric shows no significant improvement for a specified number of epochs. It helps prevent overfitting and saves training time. The parameters for EarlyStopping are monitor which specifies the metric to track, min_delta=0.001 which defines the minimum change in the monitored metric to qualify as an improvement, patience=25 which waits for 25 epochs without improvement before stopping training, verbose=2 which provides detailed logging about the early stopping behavior, mode='auto' which automatically determines whether to minimize (e.g., loss) or maximize (e.g., accuracy) the monitored metric and restore_best_weights=True which is used to restores the model's weights to the point of best performance, ensuring the final model is optimal.

7 Day Predicitons	MAE	MSE	RMSE	R^2
K-Fold 1	4.198593485	53.83948693	7.337539569	0.991791489
K-Fold 8	1.026904159	1.83E+00	1.353434274	0.968956986
K-Fold 9	1.778581691	6.135720149	2.477038585	0.983888469

14 Day Predicitons	MAE	MSE	RMSE	R^2
K-Fold 1	3.922741659	46.48543363	6.818022706	0.99290678
K-Fold 8	0.778604488	1.21E+00	1.098796598	0.97950513
K-Fold 9	2.148204888	11.68640475	3.418538394	0.96955807

30 Day Predicitons	MAE	MSE	RMSE	R^2
K-Fold 1	10.27613077	271.1274168	16.46594719	0.958716904
K-Fold 8	1.25277999	3.25E+00	1.80262445	0.944568047
K-Fold 9	3.122764448	22.90044204	4.785440632	0.94172169

Fig. 13. Enter Caption

K-Fold 8 consistently outperforms other folds across all prediction horizons, indicating its configuration or data split may be most favorable for training and testing. As the prediction horizon increases (from 7 to 30 days), errors (MAE, MSE, RMSE) generally increase, and R^2 values slightly decrease. This is expected because predicting further into the future is inherently more challenging. The high R^2 values across all folds and horizons suggest the model captures a significant portion of the variance in the data, though errors indicate room for improvement in precision.

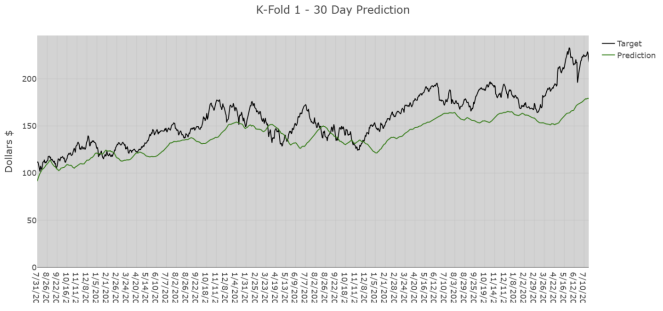

Fig. 14. K-fold 1 for 30 day prediction of stocks.

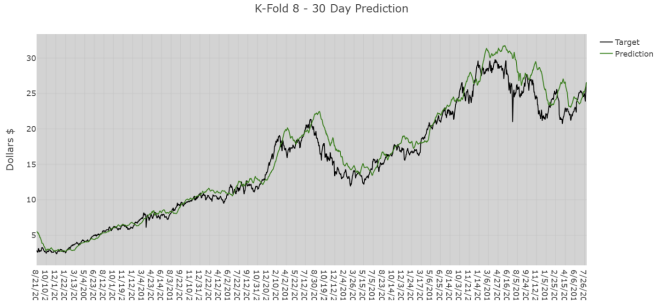

Fig. 15. K-fold 8 for 30 day prediction of stocks.

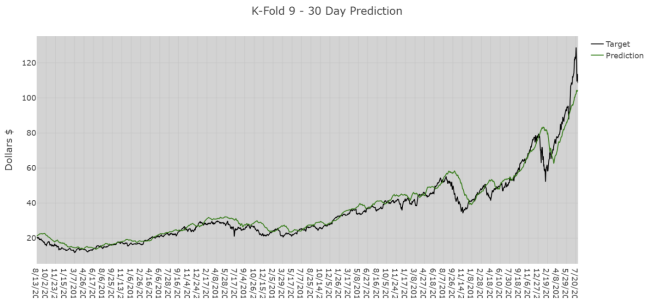

Fig. 16. K-fold 9 for 30 day prediction of stocks.

The 30-day prediction graphs provide valuable insight into the LSTM model's performance across three K-fold cycles, reflecting differences in its ability to generalize and adapt to variations in the data. These graphs align with the metrics table, which quantifies the performance for each cycle using key indicators such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2. By visually analyzing the prediction and target trends, we can see varying degrees of alignment, which highlight how the model performs better in certain cycles compared to others.

Cycle 8 (as seen in Fig. 15) emerges as the best-performing cycle among the three. It achieves the lowest MAE, MSE, and RMSE values, indicating fewer prediction errors and more accurate performance. Additionally, its R^2 value of 0.944568047 confirms that Cycle 8 has the highest predictive power, capturing the variance in the data more effectively than Cycles 1 and 9. These metrics and the corresponding graph for Cycle 8 show the model's ability to closely track the actual stock price trends over the 30-day prediction horizon, demonstrating its relative robustness and consistency during this fold.

In contrast, Cycle 1 (Fig. 14) and Cycle 9 (Fig. 16) show larger deviations from the target values, as evidenced by their higher error metrics and lower R^2 values. While the model still captures the overall trends in these cycles, the differences in alignment highlight potential variability due to the data used during training and validation. This variability underscores the importance of employing K-fold cross-validation to evaluate the model's performance comprehensively. The combination of visual analysis through graphs and numerical validation through metrics provides a holistic understanding of the model's strengths and areas for improvement. This ensures that any enhancements made to the model will be based on insights drawn from multiple cycles, rather than relying on a single validation outcome.

A. Potential for Improvement

Future iterations of the SpiderNet model could benefit from:

1) Incorporating external factors such as news sentiment or economic indicators.
2) Employing hybrid models that combine LSTMs with other architectures.
3) Expanding the dataset beyond Apple stock to evaluate model generalizability across different sectors and provide more options for end users.

V. Conclusion

THE SpiderNet project successfully developed an LSTM-based AI model capable of predicting daily low stock prices for Apple Inc. with moderate accuracy. The results validate the viability of using AI for stock market prediction, particularly with deep learning techniques like LSTMs. However, unexplored avenues underlined by the related works section show the need for more complex and multifaceted approaches.

Through the SpiderNet project, the authors gained valuable insights into the challenges and intricacies of stock market prediction using artificial intelligence. They developed a deeper understanding of advanced machine learning models, particularly Long Short-Term Memory (LSTM) networks, and their applicability in time-series forecasting. The process of sourcing, preprocessing, and splitting historical stock market data enhanced their data handling skills, while implementing and optimizing the AI model sharpened their coding and problem-solving abilities. The authors also learned the importance of selecting appropriate evaluation metrics, such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), to effectively assess model performance.

VI. Acknowledgment

THIS article reflects the views of the authors and does not necessarily reflect those of the COSC 4330 Introduction to Artificial Intelligence instructor who assigned the project.

References

[1] A. Moghar and M. Hamiche, "Stock Market Prediction Using LSTM Recurrent Neural Network," Procedia Computer Science, vol. 170, pp. 1168–1173, 2020, doi: https://doi.org/10.1016/j.procs.2020.03.049.

[2] Chin Yang Lin, João Alexandre Lobo Marques, "Stock market prediction using artificial intelligence: A systematic review of systematic reviews", Social Sciences Humanities Open, Volume 9, 2024, 100864, ISSN 2590-2911, https://doi.org/10.1016/j.ssaho.2024.100864.

[3] Md. M. Akhtar, A. S. Zamani, S. Khan, A. S. A. Shatat, S. Dilshad, and F. Samdani, "Stock market prediction based on statistical data using machine learning algorithms," Journal of King Saud University - Science, vol. 34, no. 4, p. 101940, Mar. 2022, doi: https://doi.org/10.1016/j.jksus.2022.101940.

[4] N. Rouf et al., "Stock Market Prediction Using Machine Learning Techniques: A Decade Survey on Methodologies, Recent Developments, and Future Directions," Electronics, vol. 10, no. 21, p. 2717, Nov. 2021, doi: https://doi.org/10.3390/electronics10212717.

[5] P. Raval, "Stock Price Prediction using Machine Learning with Source Code," ProjectPro, Apr. 09, 2024. https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571

[6] R. Ray, P. Khandelwal and B. Baranidharan, "A Survey on Stock Market Prediction using Artificial Intelligence Techniques," 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2018, pp. 594-598, doi: 10.1109/ICSSIT.2018.8748680. keywords: Stock markets;Support vector machines;Predictive models;Artificial neural networks;Neurons;Companies

[7] S. Elsworth and S. Güttel, "Time Series Forecasting Using LSTM Networks: A Symbolic Approach," arXiv.org, Mar. 12, 2020. https://arxiv.org/abs/2003.05672 (accessed Aug. 23, 2023).

[8] S. Mokhtari, K. K. Yen, and J. Liu, "Effectiveness of Artificial Intelligence in Stock Market Prediction based on Machine Learning," International Journal of Computer Applications, vol. 183, no. 7, pp. 1–8, Jun. 2021, doi: https://doi.org/10.5120/ijca2021921347.

[9] "Stock price using LSTM and its implementation," Analytics Vidhya, Dec. 06, 2021. https://www.analyticsvidhya.com/blog/2021/12/stock-price-prediction-using-lstm/

Appendix A
SpiderNet Website

https://cakewaterr.github.io/intro-to-ai-website/index.html

Appendix B
Youtube Link

https://youtu.be/TjmQNXvj4rk?si=dRKuoLSMV3DsOztQ

Appendix C
Project Timeline

- Project Proposal:
  - Deadline: September 29th
  - Select single stock for model implementation and choose a dataset.
  - Initial draft of project paper.
- Project Checkpoint I:
  - Deadline: October 27th
  - Extend the project proposal to 5 pages, including a progress report for each team member.
    - Initial literature review.
      - Each team member reads and summarizes an article relevant to the paper topic.
    - Proposed approach
      - Each team member looks into an ml architecture based off of the literature review and the group makes a decision on a few models to compare after attempting to code them.
      - Each team member codes their selected architecture and creates a graph with some kind of performance measure to share with the group, ending with the authors choosing the model that the paper will focus on
  - Post it on the project webpage.
- Project Checkpoint II:
  - Deadline: November 24th
  - Extend the project proposal to 7 pages, including a progress report for each team member
    - Restructure literature review according to feedback.
    - Write code walkthrough to add length to report and start on results.
    - Make improvements and summarize reasoning behind changes made to initial model code.
  - Post it on the project webpage.
- Final Project Report and Presentation:
  - Deadline: December 8th
  - Final report (IEEE two-column style format, Times New Roman, Font 10): 9 pages (excluding references and appendix) for a 4-member team, 10 pages for a 5-member team (named: Report GroupNumber.pdf).
    - Brainstorm on what sections to include to meet page length, consider creating another model for a different data set and comparing the models against each other or maybe looking at different

lead times for stock prediction. These ideas will be discussed in a meeting.

- YouTube video link
  - Create a presentation, schedule a time to record, record, and then edit video
- Project presentation slides (named: Presentation GroupNumber.ppt).
  - Create a presentation and assign sections to each member that they will present on during the class presentation.
- One-page project summary (named:Summary GroupNumber.ppt).
- Project source codes/data/readme file (named:Code GroupNumber.zip).
- Submit a peer evaluation form to assess team members.