# CMPG313 Group Project

Documentation

Group members

Julie Odendaal……………………… ..33491127

Adel du Preez………………………… 25111167

Henri Erasmus……………………….. 35314834

Lizzon-Vanique De Villiers………….. 40604012

Shaun Britz…………………………… 40563189

Leon Mostert…………………………. 20805330

# 

# 

# Table of contents

[**CMPG313 Group Project 1**](#_a656map8t0w1)

[**Table of contents 2**](#_cpsscdcflnyv)

[**1. Background 3**](#_5ut8da7873y)

[1.1 Problem Statement 3](#_umid9a79sor6)

[1.2 Research pertaining to problem statement 3](#_h0p8qx83x6zv)

[**2. Choice of Techniques 4**](#_te8xb9o3lr2j)

[2.2 Knowledge representation 4](#_uhv6xnowdk8b)

[2.3 Learning algorithm 5](#_70nevt1zhrx7)

[2.4 Search order and operators, etc. 5](#_nas0rct9xrlj)

[**3. Modeling, Simulation Experiment and Program Design 5**](#_vy3nu0px6nf1)

[3.1 Brief description of our model 5](#_od4s04qs5dz0)

[3.2 Simulation experiment 6](#_lrafg1lkndxw)

[3.3 Parameter setting 7](#_fxe1x2vfq3dr)

[3.4 Program design 7](#_vul6iejb74t5)

[**4. Training/Testing Strategy 9**](#_u43kvra5xxod)

[4.1 Testing and experiments used 9](#_lfbesk9t9ois)

[4.2 Detail of experimental setting and strategies 9](#_1xhd3mw0h4k6)

[4.2.1 Epochs 9](#_ne4bxyq90d1m)

[4.2.2 Datasets 10](#_yaxmq7lruv2k)

[4.2.3 Dataset Length 10](#_m8f2beayv2wj)

[4.2.4 Timestep Size 10](#_rvelw4xscvbn)

[4.2.5 Model Optimizer 11](#_5biw43b7an2m)

[4.2.6 Model Loss 11](#_2hm01p8x43bj)

[4.2.7 Model Layering 11](#_lmjequtbwywg)

[4.3 Improvements and hybrids 11](#_4ay2t350baxy)

[**References 12**](#_o08u3zzhjh66)

# 1. Background

## 1.1 Problem Statement

The goal of this project is to develop an AI-powered application that provides accurate and reliable forecasts of stock prices and movements. The application should utilize machine learning algorithms and historical stock data to analyze patterns and trends, allowing users to make informed investment decisions.

## 1.2 Research pertaining to problem statement

A stock market is a marketplace accessible to the public for trading publicly listed company shares, also known as equities, which represent ownership in the company. The exchange serves as a middleman, facilitating the buying and selling of shares. Stock prediction is primarily defined as the attempt to determine the cost of the stock and provide a sound basis for investors to understand and forecast the future stock price and ultimately return on investments (Akhtar *et al*., 2022). Relying on only one dataset is not sufficient and could result in an inaccurate prediction (Akhtar *et al*., 2022). Artificial intelligence (AI) stock prediction models use AI techniques to examine past stock market data and forecast future stock prices. Due to improvements in machine learning and the accessibility of large-scale datasets, these models have become more and more common. Therefore, we are utilizing machine learning to assist in stock prediction by analyzing numerous datasets in order to forecast the demand and trend of stocks. Nonetheless, predicting the performance of the stock market is a challenging undertaking. Based on various reading and research, the following key points are needed for a successful AI stock prediction model:

Data Collection:

To make accurate predictions, AI stock prediction models need a large amount of historical information, such as stock prices, trading volumes, financial statements, news articles, and sentiment on social media. These datasets are gathered from a variety of places, including stock exchanges, financial databases, and internet news sites.

Machine Learning techniques:

To identify trends in historical data and create forecasts, AI stock prediction models use a range of machine learning techniques. Popular techniques include recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, as well as linear regression, support vector machines (SVMs), random forests, and deep learning models.

Training and Testing:

Training and testing sets are created using the historical data. The training set is used to train the model, which teaches it how to relate the stock prices to the input features. Using the testing set, which offers an estimate of the model's prediction accuracy, the model's performance is assessed.

Model Evaluation and Improvement:

The performance of the AI stock prediction model is evaluated using a variety of evaluation metrics, including mean squared error (MSE). The feature set can be adjusted, various methods can be investigated, and hyperparameters can be tuned if the model's performance is not adequate.

Real-time Prediction:

After the model has been trained and put to the test, it may be used to make predictions based on fresh data that hasn't yet been viewed. To make predictions about future stock prices or market patterns, this entails extracting the pertinent elements from the available market data and feeding them to the trained model.

We are aware that great fluctuations in the stock market can be caused by unforeseen circumstances and unexpected events, even events such as terrorist incidents, natural disasters, and pandemics (Aslam *et al*., 2021:1149). As a result, the model aims to use advanced algorithms to continuously learn and adjust, guaranteeing that the projections change to reflect shifting market conditions. Our goal is to provide precise information to individual investors so they may make wise decisions, maximize their investment plans, and perhaps improve their financial prospects.

# 2. Choice of Techniques

2.1 Why we chose our techniques

Following extensive experimentation, simulations with various parameters, and thorough research on best practices in developing stock prediction models, we made a deliberate decision to employ the techniques that we implemented in our solution. These techniques were selected based on their proven effectiveness and widespread adoption within the field. Our aim was to strike a balance between the time required for our model to generate predictions and the accuracy of those predictions.

## 2.2 Knowledge representation

The code represents the data using pandas DataFrames, which provide a convenient and efficient way to handle and manipulate tabular data. The stock price data is stored in CSV files, which are read and processed using pandas. This is to provide easy and constant access to the data across multiple devices. Training data is received from an API (yahoo finance) to allow for the most accurate and up-to-date training data. We display the results of the model using a graph in order to quickly see how accurate our model's predictions are in comparison to the actual stock prices.

## 2.3 Learning algorithm

Learning algorithm: The program uses a Long Short-Term Memory (LSTM) model with a sequence of 60 time steps and 3 layers. LSTM models are a type of recurrent neural network (RNN) that can capture long-term dependencies in sequential data, making them suitable for time series forecasting tasks. Thus, the choice of the LSTM model is appropriate for this type of problem because it can learn long-term dependencies and handle time-series data well. We also used a sequence of 60 time steps as it allows the model to learn patterns over a longer time horizon, but not too long as it requires a lot of memory.

The data is normalized using a MinMaxScaler from scikit-learn, this is a good way to normalize the data since it scales the data to a range between 0 and 1, which is suitable for neural network models. It also helps improve the convergence and stability of the LSTM model during training. The scaler is saved for later use during prediction to transform the test data in the same way.

We also made use of the Adam optimizer with mean absolute error loss, this is because it is a widely used optimization algorithm that works well with deep learning models. The mean squared error is a good choice of loss function because it is a standard metric for regression problems, displaying a good measure of how confident our model is of its predictions.

## 2.4 Search order and operators, etc.

There is no explicit search order or operators used in our model. The training process involves preparing the input sequences, defining the LSTM model architecture, compiling the model with an optimizer and loss function, and fitting the model to the training data and thus has no need for a search order method or operators. However, within the LSTM model architecture, dropout layers are used as a regularization technique to prevent overfitting. Dropout randomly sets a fraction of input units to 0 during training, which helps in reducing the model's reliance on specific input features and promotes generalization.

# 3. Modeling, Simulation Experiment and Program Design

## 3.1 Brief description of our model

* Keras: Is a neural network library that acts as an interface for the TensorFlow library.
* TensorFlow: Is a library for machine learning and artificial intelligence, with a focus on neural networks.
* keras.model.Sequential: Is a recurrent neural network model that facilitates the stacking and ordering of layers. It is ideally suited for dealing with sequential data, making it very applicable for the time series data format of stock data.
* keras.model.LSTM: Is the LSTM layer where the learning takes place. It has the following parameters:
  + units: an integer value that describes the dimension of the output space of the LSTM layer. In other words, it determines how many data points are going into the next layer.
  + return\_sequences: Determines whether the output of the layer will be kept as an array of sequential data (when set as ‘true’) or not (when omitted or set as ‘false’).
  + input\_shape: Defines the shape/dimension of the input data. In our case our input into the initial LSTM layer is a variable length array (depending on the size of the training data set) of timestep-sized arrays, which each contain a single element.
* keras.model.Dropout: Determines the factor of the amount of neurons in the model that will be randomly turned off/ignored. It helps to avoid overfitting data and ending up with a model that is suited only for a specific situation, and not applicable to any other situations.
* keras.model.Dense: Sets the dimension of the output layer. Setting it to the value of one combines the output into a single value.
* optimizer: An optimizer is an algorithm that influences how the model assigns weights to data points and rate at which it learns (how quickly it can reduce the losses).
* loss: Determines how the model calculates the error between its own predictions and the true values for the training data provided to it during the fitting step.

Our final model is layered as follows:

model = Sequential()

model.add(LSTM(units=50,return\_sequences=True,input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(units=50,return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50))

model.add(Dropout(0.2))

model.add(Dense(units=1))

model.compile(optimizer='adam',loss='mean\_absolute\_error')

## 3.2 Simulation experiment

For our simulations, an LSTM model is provided with a dataset containing the historical opening stock price values of a given stock (the data has been normalized and arranged into timesteps). The model trains itself on this dataset, and outputs its loss value for each training epoch (iteration). It is then provided with another dataset, containing the historical opening stock price values of a given stock (also normalized and arranged into timesteps). The model does predictions on this testing dataset, providing a corresponding predicted value for every stock value. The normalization on the predicted values is reversed. The actual stock prices and the corresponding predicted prices are visually displayed by plotting them both on the same graph. As a measure of accuracy, both the root mean squared error and the mean absolute percentage error are calculated between the actual and predicted stock prices.

## 3.3 Parameter setting

* Model: 3 LSTM (units = 50) layers and 2 Dropout (0.2) layers, which we defined as (3, 2)
* Model optimizer: adam
* Model loss: mean absolute error
* Timesteps: 60
* Epochs: 200
* Training and testing on the same stock, by splitting the dataset up into a 0.75/0.25 training/testing dataset length ratio.

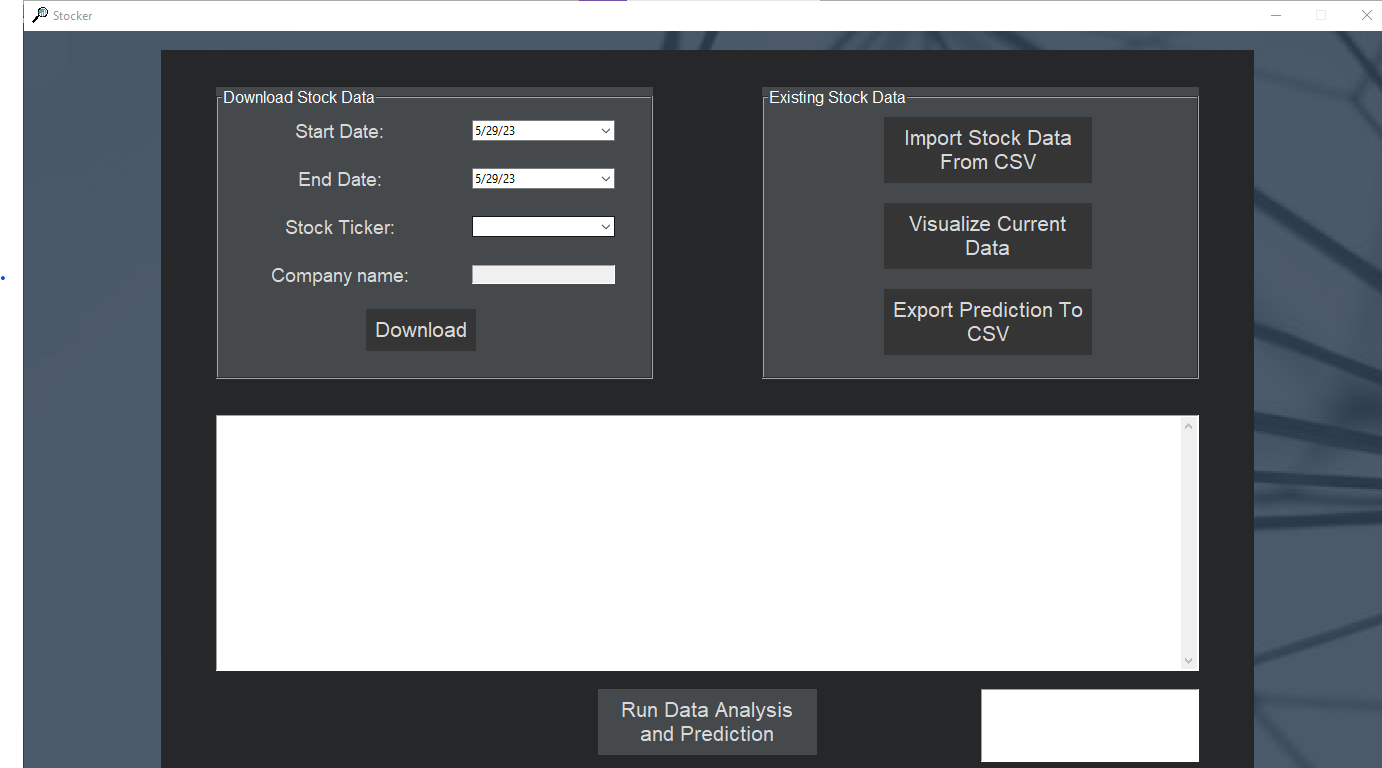
## 3.4 Program design

The program is a Python-based desktop application that utilizes the tkinter library, amongst others, to create a user interface for analyzing stock data. The GUI can be divided into different sections. On the left side, there is a section for downloading stock data. Here, the user can select a start date and end date to specify the time range for data retrieval from the API. They can also choose a stock ticker or symbol to fetch data for a specific company, and the selected company's name is displayed. By clicking the download button, the user can retrieve the data from the API.

On the right side, there are operations available for existing stock data. Users can import stock data from a CSV file, visualize the currently displayed stock data (which is shown in a listbox), and export the analyzed data to a CSV file if any analysis results are available. The listbox displays either the imported/downloaded stock data or the analysis results generated by the model.

After conducting the data analysis and prediction, the program proceeds to calculate the confidence levels, namely MAPE (Mean Absolute Percentage Error) and RMSE (Root Mean Square Error). These calculated values are then displayed within the listbox located at the bottom-left section of the screen.

Furthermore, the resulting stock data is graphed and presented as a plot. This plot allows users to visualize the analyzed data, including the actual values and the predicted values generated by the model. The plot provides a graphical representation of the stock data, aiding in the interpretation and understanding of the analysis outcomes.



Examining the code structure, the project is organized into three classes: the main class, the model class, and the GUI class. Each class serves a specific purpose within the program.

The main class primarily contains code to execute and run the program. It coordinates the interaction between the model class and the GUI class, ensuring the proper flow of data and actions.

The model class encompasses all the relevant code for training the model and performing predictions on the provided data. It includes several noteworthy functions:

* \_read\_data: This function reads the stock data from a CSV file that has been either downloaded or provided by the user.
* \_save\_model and save\_scaler: These functions save the trained model and scaler for future use.
* \_save\_training\_data: This function saves the training data as a CSV file, allowing it to be utilized later.
* \_extract\_data: This function facilitates the division of data into a 1:4 ratio, with four years of data for training and one year for testing.
* train\_model: This function handles the creation and training of the model using the training data.
* predict\_stock\_prices: This function employs the trained model to analyze the given data and make predictions for stock prices one year into the future.
* showGraph: This function generates a graph to visually represent the predictions made by the model.

The GUI class concentrates on the creation and management of the graphical user interface. It contains the necessary code for constructing various widgets, windows, and user interaction elements, enabling users to interact with the program and visualize the results of the data analysis and prediction.

# 4. Training/Testing Strategy

## 4.1 Testing and experiments used

The initial layering and parameters, as per the most widely used configuration found whilst conducting research for the model, were set up in the following way:

* Model: 4 LSTM (units = 50) layers and 4 Dropout (0.2) layers, which we defined as (4, 4)
* Model optimizer: adam
* Model loss: mean squared error
* Timesteps: 60
* Epochs: 100
* Training and testing on different stock types

We then started assessing the impact of each parameter by adjusting it while holding the parameters constant. For each adjusted value of the parameter being inspected, the training and prediction was repeated six times. As a measure of the effect that a parameter has on the performance of the model, the following values were recorded:

* The loss after the final epoch when training the model.The root mean squared error (RMSE) between the actual and the predicted stock prices.
* The mean absolute percentage error (MAPE) between the actual and the predicted stock prices.
* The relative standard deviation of each of the six repeated runs.

Lower values are considered more desirable for all four metrics. The values for the metrics, and the graphs for all the experimental runs can be found in the “Experimental Results.xlsx” file among the supplementary submission files.

## 4.2 Detail of experimental setting and strategies

## 4.2.1 Epochs

The first parameter to be assessed was the number of epochs, in other words, the number of times that the model iterates through the training dataset while it is being taught. The following values were tested: 1, 10, 20, 50, 100, 200, and 300.

Based on the results, it is apparent that the loss, MAPE, and RSD improves with a higher number of epochs. However, computation time also increases with the number of epochs. Since there seems to be a diminishing return on the improvement of the metrics with an increase of the number of epochs, we have selected the value of 200 to be used for the final model.

## 4.2.2 Datasets

Different combinations of datasets used for training and testing were tested. The first was to train and test on the same stock dataset, by splitting the dataset up into training and testing sections. Next was training on one stock dataset, whilst testing on a different stock dataset. A similar test was done, except using two different stocks for training, and one for testing. Lastly, three different stock datasets were used for training, and one for testing. For interest sake, a test was also done on training and testing on the same data. In all the cases, the same amount of data points were used for training, regardless of the dataset combination.

As expected, training and testing on the exact same data yields the best results. However, this would not be considered correct practice, and was merely done in the interest of confirming the outcome. Otherwise, across all metrics, training and testing on the same stock by splitting up the dataset into a training and testing portion, yielded the best results. Therefore it has been chosen as the approach for our final model.

## 4.2.3 Dataset Length

The effect length of the datasets used for training was also assessed. Despite not being a fixed parameter for the model, it was assessed to get a better understanding of its effect on the predictions, as well as to provide a guideline length to use for datasets. We looked at using 6 months, 1 year, 3 years, and 5 years of training data.

Looking at the results, it is clear that in terms of accuracy (MAPE), using 3 and 5 years perform better. The variance (RSD) amongst the individual runs seems to increase with increasing training data length. A possible explanation could be that the effect of increasing long-term dependencies introduces more inconsistency whilst the model is being trained, with it not being able to assign the same importance to certain trends that happened long ago relative to the dataset length every time that it trains the model.

We decided that between 3 and 5 years of training data seems to provide the best combination of accuracy and consistency as well as computation time, since longer training sets result in longer runs. It is, however, merely a guideline.

## 4.2.4 Timestep Size

The timestep size determines the length of the sequence of values that the model uses for every prediction that it makes. We looked at timestep sizes of 10, 30, 60, and 120. From the results, only a marginal increase in accuracy is seen by increasing the timestep size to 60 and then further to 120. The value for the loss also goes up when using 120 timesteps, indicating that it can still benefit from increasing the epoch value. The variance for 60 and 120 timesteps are very comparable. We decided to go with a timestep size of 60, since it performs just as well as 120 timesteps at 200 epochs, with the added benefit of having shorter runs.

## 4.2.5 Model Optimizer

We looked at the following four model optimizers: Adam, Stochastic Gradient Descent (SGD), Nadam, and Root Mean Square Propagation (RMSprop). All the optimizers performed comparably, with the exception of SGD, which performed significantly worse than the others across all metrics. Since Adam seems to be the most widely used optimizer among the models considered whilst conducting research for the model, we opted to go for it for the final model.

## 4.2.6 Model Loss

We looked at the loss – that is how the model calculates the error between its own predictions and the true values for the training data provided to it during the fitting step. Four different techniques were used, Mean Squared Error (MSE), Mean Absolute Error (MAE), Huber Loss, and Mean Squared Logarithmic Error (MSLE). MSE and MAE performed marginally better than the other two in terms of accuracy, with MAE also outperforming MSE in terms of variance. Based on that, we decided to go use MAE for the final model. We did, however, notice an increase in the loss for MAE, indicating that it can benefit further from increasing the number of epochs.

## 4.2.7 Model Layering

For model layering, we looked at different layering combinations in terms of how many LSTM, and Dropout layers to add. We made use of a naming convention for different layered models as (L, D), where L is the amount of LSTM layers (with units = 50), and D is the amount of Dropout layers (0.2) present in the model. We looked at the following differently layered combinations: (4,4), (4,0), (2,4), (2,0), (8,4), (4,8), (8,8), (3,2). From the results, we determined that the (3,2) layered model yields the best combination of accuracy and variance.

## 4.3 Improvements and hybrids

* Initially we had an artifact (unexpectedly high value) at the start of the prediction data, this was caused by using different datasets for training and testing, the first timestep in our prediction was composed of training data which was unrelated to the testing data. This was fixed by not including any training data with the testing data.
* We also improved the program by allowing the user to download the most accurate and up to date information from an API (yfinance) instead of having to use possibly outdated CSV files. However we still kept the functionality of CSV importing to allow the user to import older data if needed.

# References

Aslam, S., Rasool, A., Jiang, Q. and Qu, Q., 2021, July. LSTM based model for real-time stock market

prediction on unexpected incidents. In *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)* (pp. 1149-1153). IEEE.

Akhtar, Md.M., Zamani, A.S., Khan, S., Shatat, A.S.A., Dilshad, S. and Samdani, F. 2022. Stock market prediction based on statistical data using machine learning algorithms. *Journal of King Saud University - Science*, 34(4), 101940. <https://doi.org/10.1016/j.jksus.2022.101940>.

**Resources used in the construction of our application:**

YouTube Video - "Machine Learning & Predictive Modeling in Finance" by CodeEmporium (2020). [Online]. Available: https://youtu.be/QIUxPv5PJOY. [Accessed: 29 March 2023].

YouTube Video - "Stock Prediction Using Machine Learning" by AlgoTrading101 (2020). [Online]. Available: https://youtu.be/KYc0EFN-VnM. [Accessed: 29 March 2023].

YouTube Video - "Stock Price Prediction using Machine Learning" by Dataquest (n.d.). [Online]. Available: https://www.youtube.com/watch?v=1O\_BenficgE&ab\_channel=Dataquest. [Accessed: 29 March 2023].

Simplilearn. (n.d.). "Stock Price Prediction using Machine Learning." [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/stock-price-prediction-using-machine-learning. [Accessed: 29 March 2023].

ProjectPro. (n.d.). "Stock Price Prediction using Machine Learning Project." [Online]. Available: https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571. [Accessed: 29 March 2023]

YouTube Video - "Stock Market Prediction Using Machine Learning" by Krish Naik (2019). [Online]. Available: https://www.youtube.com/watch?v=6bzkh\_sOCoE. [Accessed: 29 March 2023].

ProjectPro. (n.d.). "Stock Price Prediction using Machine Learning Project." [Online]. Available: https://www.projectpro.io/article/stock-price-prediction-using-machine-learning-project/571. [Accessed: 29 March 2023].

Dataconomy. (2023, January). "Stock Prediction with Machine Learning." [Online]. Available: https://dataconomy.com/2023/01/stock-prediction-machine-learning/. [Accessed: 29 March 2023].

Towards Data Science. (n.d.). "Predicting Stock Prices Using a Keras LSTM Model" by Aman Dalmia (n.d.). [Online]. Available: https://towardsdatascience.com/predicting-stock-prices-using-a-keras-lstm-model-4225457f0233. [Accessed: 29 March 2023].

GeeksforGeeks. (n.d.). "Stock Price Prediction using Machine Learning in Python" by Ishan Garg (n.d.). [Online]. Available: https://www.geeksforgeeks.org/stock-price-prediction-using-machine-learning-in-python/. [Accessed: 29 March 2023].