Deep Learning for Stock Prediction A Study of The Theory & Implementation: Preliminary Report

CM3015 Machine Learning and Neural Networks
Project Idea Title 1: Deep Learning on a public dataset
University of London, Computer Science bachelor degree Final Project CM3070 by Amar Allaham.

Table of contents

1. Introduction and motivation

Introduction and motivation section word count: 960

Note: throughout this draft report, anything enclosed with "/* */" is meant to be a reminder for me (the writer) to go back to and address.

/*

By reading this section, the reader should be able to:

1. Understand what algorithmic trading is.
2. Why is it important?
3. Why did we specifically choose the stock markets as a test case out of all other financial markets?
4. What are we going to accomplish by this work?
5. What are the technical limitations and ethical concerns related to this project?

Use passive voice moderately.

Use shorter sentences as it sounds a little dense for most people.

*/

1.1 Introduction

What is Algorithmic Trading and Why is it Important?

Let's start by explaining what a stock is. "it's a security that represents a fractional ownership in a company. When you buy a company's stock, you're purchasing a small piece of that company, called a share"[1]. Stock trading involves buying and selling these shares at various prices. Investors typically buy stocks and hold onto them, selling for a profit if the stock price rises up or for a loss if the stock price falls down. Perfectly timing the execution of buying, or selling is essential in making positive returns on investment and managing risks.

Algorithmic trading automates the process of stock trading, allowing for transactions to be executed with minimal time and cost. Leveraging technological advancements from the last few decades, computers can now be utilised to buy and sell stocks on behalf of investors, aiming to maximise returns and minimise risks. This form of trading is highly prevalent, accounting for approximately 60-73% of overall US equity trading[2].

Algorithmic trading is a complex domain with many different components that must function harmoniously to achieve optimal results. One of those components is stock price prediction. This involves forecasting future stock prices and movements to make informed trading decisions.

This project will concentrate on the stock price prediction component within algorithmic trading systems. Accurate stock price predictions are essential for the effective operation of algorithmic trading systems, guiding the buying and selling decisions that ultimately determine the success of the trading strategy.

A stock prediction component can include many sub-components, such as a technical analysis based model, a fundamental analysis based model, quantitative analysis based

models, and sentiment analysis based model. For the scoop of this project we will only work on the technical analysis based model. We will build a deep learning model and train it on the historical stock price data for companies within the US stock exchange. We are aiming to achieve a reasonably accurate model that can make predictions regarding the future price of a given stock.

1.2 Motivations

By convention stock trading is a multi dimensional process that requires deep analysis of both real-time and historical data, the calculation of risk and return, the placement of orders through reliable brokers, and continuous market surveillance for potential position[3] adjustments.

Some investors delegate these responsibilities to financial firms or analysts, but this approach is not without its own set of risks and costs. For instance, the financial analyst may lack the desired qualifications, and they are still prone to human errors which could result in substantial losses.
Furthermore, the motivations of financial firms and analysts may not always align with the best interests of the investors. Jordan Belfort book ‘The Wolf of Wall Street’ (2007), provides an alarming narrative of the potential consequences when an investor entrusts their funds to a party that profits from the investor’s losses.

Algorithmic trading automates stock trading with minimal time and cost. Computers can quickly analyse large data sets with high precision, execute strategies based on these analyses, and evaluate risks and rewards of different strategies simultaneously. Completing these complex tasks with relative ease can make the stock market more accessible to the average person; which may potentially lead to the democratisation of the stock market where people from all financial and cultural backgrounds can participate.

There are four essential components in an algorithmic trading pipeline, these are: [4]
Pre-trade analysis component.
Trading signal generation component.
Trade execution component.
Post-trade analysis component.

Our work will only involve the first two components as we are focusing on stock prediction. However it’s important to note that there are many dimensions to a stock prediction component.
Since the stock market is a speculative market, the price of a stock is not only determined by fundamentals of the associated company, but also by the general precipitation of said company, so if the company leadership is unpopular, if the latest product the company has released is reviewed to have a poor quality, if the company is facing legal issues, all these might have a negative impact on the company value as a whole, and by extension; the value of its stocks.

Based on that we must acknowledge that a good stock prediction system must also include a sentiment analysis component as well as stock prices data analysis component . In an

ideal stock prediction system we will have a technical analysis based model, fundamental analysis based model, quantitative analysis based model, financial news sentiment analysis based model, and social media sentiment analysis based model. All of these models will have some weight on the final prediction of the whole prediction system.

For this project we are only going to focus on building a prediction model based on historical stock price data technical analysis, so it's important to keep in mind that this is not a full prediction system and therefore it's not enough to make an investment decision based on its output, but nevertheless, it's still a very essential component of a stock prediction system.

Even though for the scope of this project we will only focus on stock prediction within the US stock exchange, the work we will do can be applied and tested against other stock exchanges, as well as other commodities in the different financial markets such as energy, metals, and crops futures contracts, as these usually share the same underlying principles of supply and demand that influence pricing.
My vision is to create something that is worthy of being a part of an open source algorithmic trading system which enables people from various social levels to participate in the financial markets in a calculated and risk measured way.

2. Literature reviews

Literature reviews section word count: 2332

/*
In this section I want the reader to have a summarised overview of all the previous works and then an explanation in bullet points of each work strength and weakness.

This work is intended to be understandable by people who are not very familiar with the topic of financial markets and algorithmic trading, and that's why I choose [Nuti, Giuseppe (11/2011), Algorithmic Trading] to be the first previous work to have an overview of in this section as it will help the reader get the fundamentals of the topic and the problem I'm trying to solve.

One of the weaknesses of the second and third previous work in this section is that they both expect a high level of familiarity with financial markets and algorithmic trading from the reader. So I should address that more.

I might have been a little too harsh in my wordings on the forth previous work so I should try to address that.

I should use passive voice in moderation in my rewriting.

Use shorter sentences as the section sounds dense for most people. Try to use bullet points more.

I might consider adding more previous work to this section, depending on the word limit after the rewriting.

*/

In this section, we will review four studies, each focusing on different aspects of algorithmic trading. These studies were selected to provide a comprehensive view of the topic, highlighting its potentials, limitations, and risks, and offering both ideal and flawed approaches to algorithmic trading.

The first study, "Algorithmic Trading," lays the foundation by explaining what algorithmic trading is and how it operates from a less technical perspective. This study does not include the implementation of specific techniques but thoroughly explores and summarises various methodologies within algorithmic trading systems. Its straightforward approach makes it accessible even to those without a background in algorithms or finance.

The second study, "Deep Learning in Stock Portfolio Selection and Predictions," and the third study, "Deep Learning Models for Stock Market Prediction Using Optimization Approach," serve as excellent examples of practical implementations of algorithmic trading. They specifically demonstrate how deep learning can be applied to the stock market for predicting future stock values.

The fourth study, "Machine Learning for Stock Trading," is a public notebook on Kaggle and represents a less successful attempt at applying deep learning to stock market price prediction. This study contains noticeable miscalculations and makes bold conclusions based on these errors. Despite its shortcomings, it is included in this review because it uses the same data source as our work and includes both implementation and experimental results, which will allow us to have a direct comparison with it. We believe that analysing both successful and flawed research can provide valuable insights.

2.1 Nuti, Giuseppe (11/2011), Algorithmic Trading[4]

Summary

Algorithmic trading is a method that automates one or more stages of the trading process. It has become a major concern for regulators more than a decade ago due to events like the 6 May 2010 Flash Crash, where in 5 minutes $600 billion of the market value of US corporate stocks were wiped out. This event highlighted the lack of knowledge about algorithmic trading.

Trade Execution
In algorithmic trading, bids (buy) and asks (sell) orders are converted to entries in the order book. This record ranks the bids by price from highest to lowest and then by the earliest order timestamp. Asks are ordered by price low to high, where orders with the same price are ordered by the earliest timestamp. These entries in the order book will later be matched

together from top to bottom. Designing a good algorithmic trading system requires a deep understanding of the process of how ask and bid orders get executed in a specific exchange.

Trading Objectives
The objective of algorithmic trading is to always make a profit. However, the approach to achieving that profit may vary depending on which party is taking the risk. Some financial firms operate only as middlemen—they execute trades using their customers' money. These firms use algorithmic trading strategies that aim to minimise the cost of trading for the available capital. On the other hand, firms such as investment banks use their own money as well as their customers' money. These firms use algorithmic trading strategies that aim to maximise profits against some measure of financial risk.

Trading Process:

- An algorithmic trading pipeline consists of four essential components: pre-trade analysis, trading signal generation, trade execution, and post-trade analysis. Algorithms can automate one or more components of the trading pipeline.

- Pre-trade analysis, the most common use of algorithms, involves using stock market data and news data to generate a forecast of future stock prices. Pre-trade analysis techniques include fundamental analysis, technical analysis, and quantitative analysis.

- Trading signal generation can use algorithms to automate generating ask and bid orders given the output of a pre-trade analysis system. Trading execution can use algorithms to minimise trading impact and timing risk by optimally scheduling trades and selecting between different markets.

The Future of Algorithmic Trading
The future of algorithmic trading is being shaped by three key areas: Dark Pools, Ultrahigh-frequency trading, and Exchange Traded Funds (ETFs). Dark Pools allow for trading large blocks of shares without public disclosure until completion. Ultrahigh-frequency trading uses powerful computers to execute thousands of trades per second. ETFs, a new multi-asset instrument, pose a greater technical challenge for algorithmic trading. Dark pools and ultrahigh frequency trading raise serious concerns about how they affect the financial markets.

Strengths and Weaknesses

The study provides a solid introduction to many financial market concepts, making it accessible even to those without a finance background. It helps us understand the essential components of an algorithmic trading pipeline, starting from pre-trade analysis and trading signal generation, to trade execution and post-trade analysis.

However, the work is too general and doesn't attempt to show any examples for a methodology, implementation, or results of any specific algorithmic trading pipeline. It also doesn't compare algorithmic trading to traditional stock trading in terms of requirements,

cost, risk, and return. Therefore, the value that algorithmic trading provides is not clear just from reading this study.

Moreover, the paper was published in 2011, so it might not cover everything a researcher would need to know if they want to work on algorithmic trading today, as there might have been new innovations in the topic of algorithmic trading since this research was issued.

2.2 Alzaman, Chaher (03/2024), Deep Learning in Stock Portfolio Selection and Predictions[5]

Summary

The paper presents a novel approach to stock portfolio selection using deep learning models, by challenging the common concept that the LSTM (Long Short-Term Memory) model is the go to model when dealing with any type of sequential data and aims to show the capability and superiority of the Deep RankNet model for this specific problem. The authors employ genetic algorithms for hyperparameter optimization, which they claim improves the model's performance by 40% over grid search optimization. The paper focuses on the Toronto Stock Exchange (TSE) data.

Methodology

The authors use LSTM and Deep RankNet for prediction and compare the performance of these two models only. They also use genetic algorithms for hyperparameter optimization. The research questions include identifying the best hyperparameters, comparing LSTM with Deep RankNet, evaluating the impact of different portfolio sizes, and examining the influence of market profiles and trading period volatility on the Deep RankNet model performance.

Results

The authors analysed 100 stocks from the TSE using historical data from Yahoo Finance. They used four technical indicators: Moving Average (MA), Exponential Moving Average (EMA), Moving Average Convergence/Divergence (MACD), and Relative Strength Index (RSI). The genetic algorithm significantly improved the model's prediction accuracy by approximately 40% compared to the grid-random search. However, the study didn't explore the performance of other models than RankNet and LSTM.

Conclusion

The study aims to construct financial portfolios using deep learning, benefiting both academic research and practical applications in portfolio selection and risk analysis. The highlights of this study include showing the advantages of using the deep RankNet over LSTM, showing the advantages of using genetic based algorithm over grid search for hyperparameter optimization tuning, and a novel approach for synthesising multiple stocks and periods.

Strengths and Weaknesses

The paper provides a comprehensive methodology and addresses all the research questions it starts with. It also provides the parameters and results for each of the models, which adds to the transparency of the research. The study uses 100 different stocks to train the model, this diversity in stock selection is beneficial to generalise the model output and ensure the testing results are not the result of over-fitting.
The four technical indicators used here are MA, EMA, MACD, and RSI which are very common for this type of problem.

However the paper does not provide direct access to the data used, making replication of the work harder. The implementation of the models is not provided in a standard programming language for machine learning and algorithmic trading such as Python, which increases the difficulty of replicating the results of the research. The work doesn't measure the impact of the four different indicators used for technical analysis on the output of the models nor does it consider different technical indicators that are also used for this type of problems.
The paper might be overly optimistic about the performance of the RankNet model since it was trained on historical data and tested only during the Covid-19 lockdown, which had an enormous and unexpected impact on the financial markets all around the world making it unlikely for the model to be capable of predicting the impact of something it was never trained on. Therefore even if the model happens to perform really well within the covid-19 luckdown period, it needs to be tested for a longer period of time within more regular circumstances in order to confirm its accuracy.

2.3 B L, Shilpa (2023), Deep Learning Models for Stock Market Prediction using Optimization Approach[6]

Summary

This work presents a comprehensive approach to stock market prediction by combining historical stock prices data with news sentiment analysis. The authors employ the Self Improved Whale Optimization Algorithm (SIWOA) for weights optimization in the Deep Belief Network (DBN) model.

Methodology
The authors propose a stock price prediction model that integrates sentiment analysis of big data, combining stock data technical analysis with news sentiment data.
The technical analysis model utilises the moving average (MA), relative strength index (RSI) and moving average convergence divergence (MACD) indicators for feature selection.
The sentiment data is processed through pre-processing, keyword extraction, feature extraction, and classification. The SIWOA model is used for weight optimization in DBN, addressing local optima issues in standard Whale Optimization Algorithm (WOA).

Results

The proposed method for stock market prediction was evaluated using five different stocks, with learning rates varying from 60% to 90%. The method achieved low error metrics, demonstrating the effectiveness of the proposed approach.

Conclusion
The research developed a prediction framework that uses stock price data and news sentiment data. The Deep Belief Network (DBN) weights were tuned by the Self Improved Whale Optimization Algorithm (SIWOA) for precise sentiment prediction, achieving a Mean Absolute Error (MAE) of 0.92 at a 90% learning rate.

Strengths and Weaknesses

The paper presents a reliable approach to stock market prediction by combining stock historical prices data technical analysis with news text sentiment analysis. This approach is essential in real algorithmic trading systems as the stock market is known to be speculative and the value of a company stock can fluctuate depending on how a company is perceived by the news outlets and the public.
This work utilises MA, RSI, and MACD indicators for feature selection which are among the most common indicators for this type of problems.
The study also focuses on the DBN and the advantages of using the Self-Improved Whale Optimization Algorithm for hyperparameter optimization.

However, despite acknowledging the importance of sentiment analysis for stock prediction, the study only focuses on news outlets for the input data. A more comprehensive approach would be to also utilise people's sentiment of a company on social media networks such as Twitter or Reddit.
Even though this work utilises the MA, RSI, and MACD indicators for feature selection, it doesn't measure their impact on the model output nor does it try to include other technical indicators that may be useful for this type of problems.
The work does not provide the actual implementation of the model nor the actual preprocessing of the data, making it harder to replicate the results recorded in this work.
Additionally, the work does not provide a general introduction of the DBN and the Self-Improved Whale Optimization Algorithm for those who are not familiar with these algorithms.

2.4 IBRAHIM KARATAS (2023), Machine Learning for Stocks Trading[7]

Summary

The author begins by importing and processing company stock data from yfinance, an open-source tool that uses Yahoo Finance's publicly available API. They import the company's stock price data on a daily interval and then utilise pandas-ta, a technical indicators library compatible with pandas DataFrame, to add two technical indicators to the

stock prices data frames, which are RSI (Relative Strength Index) and CCI (Commodity Channel Index).

Following this, the author adds a label column indicating whether the stock's opening price is going up (1) or not (0). The data is then split into training and testing sets in a 70-30 ratio.

The author employs MLPClassifier from sci-kit learn as a model, training it on the training set and testing the model on the test set, yielding the following results:
precision: 0.50, recall: 0.73, f1-score: 0.59

The author then clarifies that the loss reduction stagnates after 20 epochs.

Conclusion
The author concluded that this performance is unacceptable to be of any use but yet expected as a consensus of experts in finance and AI don't think machine learning is capable of performing accurately in the financial market. There were no references included in this notebook to back this claim.
Finally the author concluded that there is no amount of data or training of any different model that can create a model capable of making acceptable and reliable predictions in the financial markets.

Strengths and Weaknesses

This study uses publicly available data and includes detailed data preprocessing steps, models, and accuracy metrics. This transparency is essential as it allows other researchers to access, replicate, and build upon these findings.

However, the conclusions were drawn from training a single model on the stock price history of one company. Only two technical indicators were used, with no attempt to include more or measure their impact on the model's output. Additionally, there was no data scaling in the preprocessing steps, nor was any effort made to optimise hyperparameters.

Finally, the author claims a consensus among finance and AI experts that machine learning is incapable of accurately performing in the financial market, but does not provide references to support this claim.

In conclusion, while the work provides a clear and replicable development pipeline, it demonstrates a limited understanding of machine learning and finance. Financial market price prediction is a complex problem requiring deep knowledge in both fields.

3. Project design

Project design section word count: 1829

/*

The most impactful change I would do to this section is to redo the development pipeline/workflow.

1. In the next iteration I would start with developing baseline deep learning models for both regression and classification approaches.
2. For each approach I need to create a baseline model for each model type from the predetermined selection (simpleRNN, RNN, LSTM, GRU, Transformers) if possible. If not possible I must clarify why.
3. Prepare the data to be trained on each of these models (we might apply different data preprocessing and reshaping to be suitable for each model type).
4. The model feature selection at this stage is still the same as the one from the prototype.
5. Use hyperparameter optimization to train each model. In the prototype I was thinking about using gridSearch to find the best parameters for each model, but this approach is computationally expensive, and according to the previous work, randomSearch is more efficient than gridSearch while providing similar results. So I think I will switch to randomSeach.
6. Use the accuracy metrics I already discussed in this section to pick the 2 best performing models so far.
7. At this stage we have already considered the most likely models we will be using so we can perform a proper feature selection on them to choose the final feature selection for this model. This part involves a lot of iterations.
8. Based on the previous step we can eliminate one of the models based on performance. This model is going to be our final model.
9. Now we try hyperparameter optimization on this model, with much more parameters to optimise compared to the last steps.
10. Then we try some manual optimization for the chosen model, this involves adding more layers, changing the number of neurons on each layer and so on. Even though we can do this part programmatically it might take us more time, and based on what I understood from the deep learning with python book, training models is more like of an art that require trial and error and not an exact science, so I should utilise my best intuition and judgement during this stage.
11. After we arrive at the best iteration of our selected model we can start using data at different intervals like month, day, hour, and so on.
12. Create a version of the model trained on each data interval.
13. Report the accuracy for each version of the model.
14. Wrap all these different interval model in one python class. This class will take a stock symbol string as input. This class should have a predict function that takes a time interval string as an argument (Month, week, …etc) and return the prediction for that specific interval (The stock price of X is expected to be Y the next Month/Week/Day)

Note: while this approach might be a little limited compared to the more thorough approach in the initial project design section, it’s much more manageable and realistic to achieve

within the time limit and within hardware limitations that I have. However it's still challenging and go beyond what I encountered in all of the previous works.

I need to make the text less dense for the reader by making the sentences shorter, and use bullet points when possible.

Use passive voice in moderation.
*/

3.1 Domain

The project explores the applicability and capability of deep learning within the financial markets, specifically the stock market. The financial sector benefits from continuous monitoring of data, news, and market shifts. This project utilises publicly available financial data to train deep learning models to identify patterns to make reasonably accurate predictions that normally would take a lot of time, knowledge, experience, and efforts for human analysts to make.
The project will focus primarily on the US stock market, however the analysis and results of this work can also be tested against other exchanges.

3.2 Target users

The primary aim of this project is to make financial market participation more accessible to a broader audience. This includes professionals such as teachers and nurses who may not typically engage in stock trading due to the complexities and costs involved. By providing unbiased, free financial advice through a trusted agent powered by deep learning, we can significantly lower the barriers to entry.

A reliable deep learning model has access to large amounts of data and possesses the capability to process this data, recognize trends, and make informed predictions. Unlike human advisors, the model operates without personal motives, ensuring that it delivers objective advice without charging fees or acting against the user's best interests.

This project can also benefit individual investors, financial analysts, and algorithmic traders by automating stock analysis and trend predictions. The automation reduces the complexity and cost traditionally associated with stock trading, hence democratising access to financial markets and allowing more people from various financial backgrounds to participate.

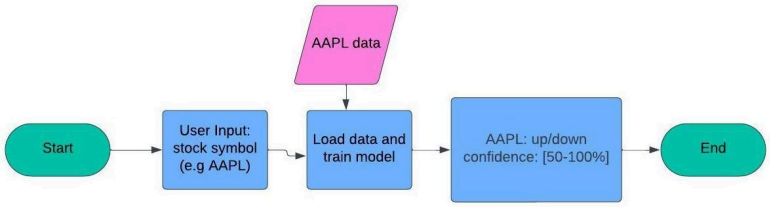
3.3 User Needs and Domain Requirements:

In the context of algorithmic trading, users need to have access to free and trusted financial advice that is based on thorough analysis, trend recognition and free of biases.

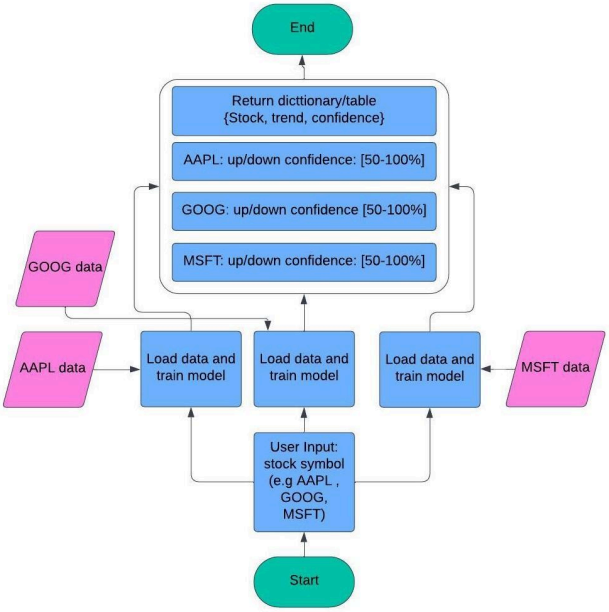
The minimum viable product for this project allows users to input one or more stock symbols (e.g. AAPL for Apple). The model will then:

1. Load data for the specified stocks.
2. Train itself on the data.
3. Output predictions in the format: (stock symbol: up/down, confidence: [50-100%]).

The "confidence" metric indicates the model's certainty in its predictions for each stock which we can do if our final model was a classification model.



To be able to deliver that we need a continuous access to real market data, we need to process that data in an understandable way for both humans and machine, we need to perform technical analysis on that data and recognize any patterns that would help the deep learning model to make inferences, we need to train the model iteratively, adjust the model variables, record observations between each iteration, and test the accuracy of the model against unseen real market data.
Our goal is to develop a stock prediction system capable of delivering accurate predictions for any stock. To achieve this, we will design a horizontal model architecture where the user input generates multiple model instances. Each instance will correspond to a specific stock from the user input and will be trained on its individual data. The final output will be a dictionary (or table) listing all the stock symbols along with their corresponding predictions and confidence levels.



It's important to note that we will need to train the model from scratch every time we need to make a prediction (so if we use it once every month we must train it once every month on the latest data). We cannot simply train the model on the latest data that has emerged since the last training session. This approach, known as continuous training, is still an active area of research and it's outside the scope of this project.

3.4 Research Questions and Methods

3.4.1 Questions:

1) What are the optimal data intervals and periods?
Investigate various data intervals (e.g., weekly, daily) and periods (e.g., last 10 years) to determine their impact on model accuracy.

2) What is the optimal feature combination?
Evaluate a range of technical indicators, adding them incrementally to identify features that enhance model performance.

3) What is the optimal model and model architectures?
Test different models, mainly RNN, LSTM, GRU, and Transformers and optimise them using GridSearchCV and RandomSearch. Parameter optimization includes neuron count, layer depth, timesteps, optimizer function, loss functions, epochs, and batch size.

4) Should we approach this problem from a regression perspective? or classification one?
We need to decide whether to use a regression approach (predicting next closing price) or a classification approach (predicting up/down trends) for stock prediction.

The regression approach provides detailed insights, which is useful for portfolio optimization. However usually it requires cooperation with different analysis based models such as fundamental, quantitative, sentiment analysis models, this is too complex for the scoop of the project. We will see how accurate the best regression model we can build based solely on the technical analysis approach.

The classification approach is simpler, provides clear trading signals (easy to interpret output) and is already used in our prototype, which is why we will consider it as a reliable safety net if regression proves unattainable.

3.4.2 Methodology:

1. Load historical stock data using the yfinance API.
    - The model will be trained on a random selection of stocks from the S&P 500 list, which comprises the most valuable companies in the US. To ensure generalizability, we will avoid the top 10 most valuable companies due to their unique stock price histories. Instead, we will select companies from various ranks within the S&P 500, such as those ranked 50th-60th and 150th-160th.

This approach ensures a diverse representation of average market movements.
- Consider different time periods and intervals. The yfinance API allows us to specify the data period (e.g., the last 10 years) and intervals (e.g., quarterly, weekly, daily, hourly, and minute basis).
- Analyse the impact of different data periods on the model's performance. Additionally, we will build multiple model instances for each interval to cater to different trading styles, such as quarterly traders and daily traders.

2. Add prediction targets.
    - Create the next closing price column. This will be the target for regression models.
    - Create the price trend column. 1 if the next closing price is greater than the current closing price and 0 otherwise. This will be the target for classification models

3. Add technical indicators. This is an iterative step as we incrementally add relevant indicators and see their impact on the model performance.

4. Construct the models. This is an iterative step as we construct the potential model based on the selected architectures (RNN, LSTM, GRU, Transformers). We also test out various construction steps for each model such as layers count, type and neurons count.

5. Prepare the data for training. This involves scaling and reshaping the data to be compatible with the desired model, and splitting the data into training and testing sets using a 70-30 ratio. This 70-30 split, in contrast with the 80-20 ratio seen in some previous work, is preferred as it reduces the likelihood of model overfitting.

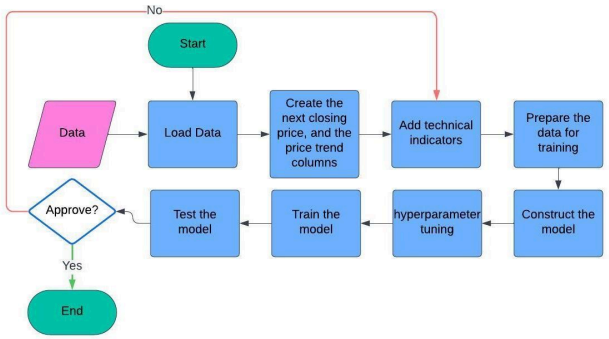
6. Perform hyperparameter optimization tuning for each model using GridSearchCV and RandomSearch to find the optimal ones.

7. Train the best-performing model using its optimal hyperparameters.

8. Test the model on unseen real market data and evaluate the results.
   For classification models we will optimise for the highest precision then accuracy, as precision is more important for the model to be correct when predicting a positive (trend up), if the model predicted up but in reality the price went down, users might lose a lot of money.
   For the regression models we will optimise for the highest R2 and lowest RMSE.

9. Approve the model or go back to 3.

For the development process we will follow chapters 6 and 7 from François Chollet book "Deep Learning with Python"(2017)[8].



3.5 Technologies and Methods

3.5.1 Technologies:

Data Source: Yahoo finance stock prices historical data through the yfinance API.

We will use Python as our development language and we will process the data and build the models in jupyter notebook.

pandas, numpy will be used for data processing and training preparation.
pandas-ta will be our library for technical analysis as using it will save us the time and effort of writing the technical analysis formulas ourselves. pandas-ta also is compatible with pandas dataframes.

matplotlib for visualisation.

All the deep learning models will be built with TensorFlow, but we will still use scikit-learn for some tasks such as data pre-processing, evaluation, and hyperparameter optimization.

Finally to build a simple user interface to enable users to interact with the model using a webpage we will use the Flask framework.

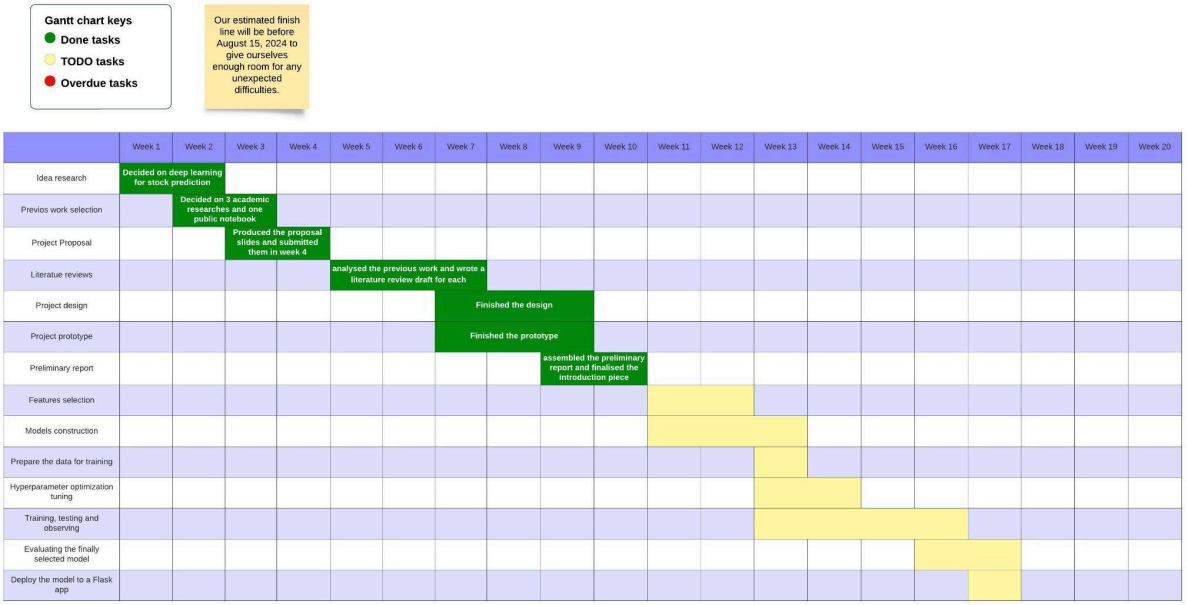
3.5.2 Methods and development pipeline:

1. Collect financial data using yfinance.
2. Process the data using pandas.
3. Implement technical indicators with pandas-ta.
4. Visualise data and results using matplotlib.
5. Prepare the data for training using numpy.
6. Scale the data using MinMaxScaler from scikit-learn
7. Optimise model hyperparameters with GridSearchCV from scikit-learn and RandomSearch (whichever gives the better results).

8. Develop and train deep learning models using TensorFlow.
9. Evaluate the model performance using precision_recall_fscore_support and confusion_matrix from sci-kit learn.
10. The final deliverable model will be a class in python.
11. Deploy the model with a simple Flask web interface.

3.6 Work Plan

This is the gantt chart for this project tasks. We are aiming to finish all the major development by August 15, 2024. Which is about 2 weeks before the submission deadline, this is to give ourselves room to deal with any unexpected difficulties or delays on the associated tasks.



3.7 Testing and Evaluation

For the classification models we will create a table comparing each model type and its parameters alongside the results of testing the model on the testing set which account for 30% of the whole dataset. For each model iteration we will record our observations of the accuracy, precision, recall, and f1-score, however the metrics we want to maximise the most are the **precision (**to minimise false positives) and accuracy.

This is just an example of what the table would look like.

Model	seed	depth	Layers	neurons count	Neurons	optimizer	metrics	Loss function	loss	Activation function	validation_split	epochs	batch_size	Parameter x	Accuracy	Precision	Recall	F1-score
SimpleRNN	x	3	[SimpleRNN, SimpleRNN ,Dense]	102	[50, 50, 2]													

Confusion Matrix will also be utilised in order to understand the distribution of prediction errors.

For the regression models we would do a similar table to the classification models but instead of precision we would target highest R Squared and lowest RMSE.

Finally after training the best model and testing it on the test set, it must be also tested over an extended period (no less than six months) to evaluate its real-world performance. This period will allow us to observe the model's behaviour in a live environment and confirm its accuracy and reliability. Despite our thorough evaluation, observing the model's performance over time is essential for validating its effectiveness in practical applications. This is of course outside the scope of this project however it must be stated for ethical consideration that any model we produce through our project is for study and research purposes only and should not be used to make any financial or investment decisions.

4. Project prototype (implementation)

Important note: Whenever I include code other than my own I always include the soruce of the code right above it. If a code snippet doesn't include a reference to a source, you should assume it's done by me.

Install Dependencies and import libraries

```
# pip install pandas numpy yfinance pandas-ta scikit-learn
tensorflow

# https://pypi.org/project/yfinance/ (""" it's an open-source tool
that uses Yahoo's publicly available APIs, and is intended for
research and educational purposes. """)
# import yfinance, our data source
import yfinance as yf

# https://pypi.org/project/pandas-ta/ ("""An easy to use Python 3
Pandas Extension with 130+ Technical Analysis Indicators. Can be
called from a Pandas DataFrame or standalone""")
# import pandas-ta
import pandas_ta as ta

# import pandas and numpy
import pandas as pd
import numpy as np

# import matplotlib for data visualisation
import matplotlib.pyplot as plt

# import from scikit-learn
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import precision_recall_fscore_support,
confusion_matrix, ConfusionMatrixDisplay

# import from tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, LSTM, Input,
GRU
from tensorflow.keras.utils import to_categorical
```

Select and download the stocks histroical prices data

For this prototype we will consider the following stocks from the S&P500 index. we will take 5 random stocks from the S&P500 index list. Ranks are as of June 6, 2024 from https://www.slickcharts.com/sp500[9]

- Rank 48 Pfizer Inc. 'PFE'
- Rank 150 Roper Technologies, Inc. 'ROP'
- Rank 251 Xylem Inc 'XYL'
- Rank 350 Corpay, Inc. 'CPAY'

- Rank 450 Incyte Genomics Inc 'INCY'

```
# insert the stock symbols into a list
symbols_list = ['PFE', 'ROP', 'XYL', 'CPAY', 'INCY']
```

For now we will download the selected stocks weekly price data for the last 10 years

```
# we will take the weekly data for the last 10 years
# data_weekly = yf.download(symbols_list, period='10y',
interval='1wk')
```

Format the data and save it as a CSV file

Reshape the dataframe so it has the ticker as a second index

```
# source of inspiration:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.stack.
html[10]
# Return a reshaped DataFrame having a multi-level inde
# stacked_data_weekly = data_weekly.stack()
# stacked_data_weekly
```

What are we looking at?

Date: one of the indexes and is the date in which the information on the reset of the columns takes place. Ticker: an abbreviation of the company's share listed on the exchanges. Adjusted close: is the closing price after adjustments for all applicable splits and dividend distributions. Close: is the closing price of the stock, on that timeframe, (here it's a week). High: is the highest point a stock has reached in the given timeframe. Low: is the lowest point a stock has reached in the given timeframe. Open: the opening price of the stock on that timeframe. Volume: the volume of stocks were traded on that timeframe.

Usually for this type of models we would only keep either Adjusted close or close but we will leave theme both for now.

Save the data to a CSV so we don't have to make any extra unnecessary requests to the API every time we reload the notebook

```
# save the dataframe to a csv file
# stacked_data_weekly.to_csv('stacked_data_weekly_1.csv',
index=True)

# load the the dataframe from the csv file
df = pd.read_csv('stacked_data_weekly_1.csv').set_index(["Date",
"Ticker"])

# df.head(5)
```

Perform simple exploritory data analysis

```
# how many null values in each column
df.isnull().sum()
```

```
Adj Close    0
Close        0
High         0
Low          0
Open         0
Volume       0
dtype: int64
```

```
# the data shape
df.shape
```

```
(2615, 6)
```

```
# data basic stats
df.describe()
```

```
         Adj Close        Close         High          Low
Open  \
count  2615.000000  2615.000000  2615.000000  2615.000000
2615.000000
mean    144.165301   147.598917   151.117521   143.524492
147.284862
std     125.341165   125.626836   127.977518   122.748919
125.314164
min      18.176517    25.400000    26.170000    25.200001
25.580000
25%      47.719940    51.580000    52.844999    50.285000
51.695000
50%      94.846069    95.919998    99.419998    92.790001
96.489998
75%     208.165001   209.534996   215.550003   203.775002
209.764999
max     570.140015   570.140015   574.289978   556.640015
567.109985

             Volume
count  2.615000e+03
mean   3.118655e+07
std    5.983749e+07
min    3.771000e+05
25%    2.756700e+06
50%    4.932200e+06
75%    9.492550e+06
max    6.333997e+08
```

Devide the data into five dataframes, one for each stock

```
# source of inspiration
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.xs.htm
```

```
l [11]
# select specific stock data at the 'Ticker' level of this multi
index dataframe
df1 = df.xs('PFE', axis=0, level='Ticker', drop_level=True)
df2 = df.xs('ROP', axis=0, level='Ticker', drop_level=True)
df3 = df.xs('XYL', axis=0, level='Ticker', drop_level=True)
df4 = df.xs('CPAY', axis=0, level='Ticker', drop_level=True)
df5 = df.xs('INCY', axis=0, level='Ticker', drop_level=True)

# disply the first dataframe
df1.head(5)
```

	Adj Close	Close	High	Low	Open	Volume
Date						
2014-07-14	19.895063	29.155598	29.174574	28.785580	28.965843	25720973
2014-07-21	19.545464	28.643265	29.136621	28.538898	28.870968	96927527
2014-07-28	18.684397	27.381405	28.927895	27.220114	28.263758	178002365
2014-08-04	18.509615	26.888046	27.476280	26.442125	27.466793	143172198
2014-08-11	18.705553	27.172676	27.419355	26.593927	27.068312	107166188

```
# show the new df shape
df1.shape
```

```
(523, 6)
```

Create the target of the model

Now after this point in the prototype we will only work with df1 which includes the historical stock prices data for the last 10 years for Pfizer Inc. or 'PFE'. This is so we don't make the this prototype too lengthy. However you can easily tell that it's going to be the exact same steps we can follow to train the model on any of the other 4 stocks dataframes. For the final project report we will include at least 5 different stocks data to make sure that our model generalises well.

We are going to try and predict the stock trend based on its past data, that is we will try to predict if the stock price will go up or down, if we know the stock price will go up in the future, we can buy it now and sell it for a profit if the model estimation were correct. also if we know the stock will go down in the future, we can sell it (or short it) now, so our assets won't depreciate in value.

To do that we will create 2 new columns:

- 'Next' column where the values for this column will be equal to the next closing price. This is will be the target for the regression model.
- 'trend' column where the values for this column will be either 1 or 0 depending on whether the 'Next' closing price will be higher than the closing price. This will be the target for the classification model, which what we will be using in this prototype.

```
# copy the dateframe before modification so we don't get a warning
from jupyter notebook
df1 = df1.copy()

# create the 'Next' column to be equal to the next closing price
# this can be accomplished easily by shifting the close column
backward by 1
df1["Next"] = df1['Close'].shift(-1)

# create a function that returns 1 if the the next closing price is
higher than current closing price and 0 otherwise.
def assign_trend(row):
    if row['Next'] > row['Close']:
        return 1
    elif row['Next'] < row['Close']:
        return 0
    else: # if the next value is missing then return NaN
        return np.nan


# create the 'Trend' column to be equal to the output of the
'assign_trend' function
df1['Trend'] = df1.apply(assign_trend, axis=1)

# check out the results
df1.head(5)
```

	Adj Close	Close	High	Low	Open	Volume
Date						
2014-07-14	19.895063	29.155598	29.174574	28.785580	28.965843	25720973
2014-07-21	19.545464	28.643265	29.136621	28.538898	28.870968	96927527
2014-07-28	18.684397	27.381405	28.927895	27.220114	28.263758	178002365
2014-08-04	18.509615	26.888046	27.476280	26.442125	27.466793	143172198
2014-08-11	18.705553	27.172676	27.419355	26.593927	27.068312	107166188

	Next	Trend
Date		
2014-07-14	28.643265	0.0
2014-07-21	27.381405	0.0
2014-07-28	26.888046	0.0
2014-08-04	27.172676	1.0
2014-08-11	27.438330	1.0

Check if the data is balanced

```
# let's check the occurance of each value in the Trend column
df1['Trend'].value_counts()
```

```
Trend
1.0    269
0.0    249
Name: count, dtype: int64
```

```
df1['Trend'].value_counts()[1]
```

```
269
```

```
# percentage of 'trend up' to the whole column
df1['Trend'].value_counts()[1]/df1.shape[0]
```

```
0.51434034416826
```

We notice that the data is very balanced in this dataframe. This is good because it will mean the model will be less prone to overfitting.

Create a common sense baseline

It's essential to start our model development by setting a common sense baseline, this is important because we need to make sure the model we will create, will atleast match this logical baseline, otherwise we will be better off not relying on the model at all. For this problem, stock prediction, a reasonable assumption is to assume the average person will predict the future trend of the stock to be equal to its current trend. In other words if the stock is going up, this person will think it will go up again and vise versa. This is actually a very solid baseline as it's very reminiscent of the real life. This is has something to do with the topic of behaviorl finance[12] but it's outside the scope of this project.

```
# this can be accomplished easily by shifting the close column
forward by 1
common_sense = df1['Trend'].shift(1)

# measure the average of when the common sense (naive) prediction
will match the actual 'Trend'
(common_sense == df1['Trend']).mean()
```

```
0.5239005736137667
```

This means if a person make a prediction in this method, they will be 52% of the time correct!

Include the technical indicators

Adding indictors to the dataframe is an essential steps in improving the model performce, as these will be consiered among the model features list and can have a massive benefits on the model performance. There are two ways to include technoical indicators, we can do it manually, which will take a lost of time, and reseach, as well as it's more prone to mistakes. We can also utlize an existing library for this very purpose. That's why we will be utilizing the pandas-ta library which not only includes most of the nessarry technical indicators we need to calculate, but it's also designed specifically to work on pandas dataframs which is ideal in our case.

We will use pandas-ta to add all the technical indicators we want to the dataframe

```
#  we can easily check the available indicators in the pandas-ta
library
# help(df1.ta.indicators())
```

We notice there are a large list of indicators that we can add easily to our dataframe. This is important for the final report because all the indicators we might need are already available, so we can systematically study each one of them, add them to our features list, and test their impact on the model performance.

```
#  we can also learn about any specific indicator like this
# help(ta.macd)
```

For this prototype we will only consider few of the most common technical indicators that are used by analysts and ML models to make an estimation of the stock trend direction. These are MACD, RSI, SMA, and EMA. the ta.macd function also returns Signal, and Histogram values. so we will add those too.

```
# for the time being let's create a function that add all the
technical indicators we want to a df
def assign_TIs(_df):
    # apply macd on the Close column in a df and add it to the
dataframe
    mcda = ta.macd(_df["Close"])
    # The MACD (Moving Average Convergence/Divergence) is a popular
indicator to that is used to identify a trend
    _df.insert(6, "MACD", mcda["MACD_12_26_9"])
    # Signal is an EMA (exponential moving average) of MACD
    _df.insert(7, "Signal", mcda["MACD_12_26_9"])
    # Histogram is the difference of MACD and Signal
    _df.insert(8, "Histogram", mcda["MACD_12_26_9"])

    # apply RSI on the Close column in a df and add it to the
dataframe
    # RSI (Relative Strength Index) is popular momentum oscillator.
Measures velocity and magnitude a trend
    rsi = ta.rsi(_df["Close"])
    _df.insert(9, "RSI", rsi)

    # apply SMA on the Close column in a df and add it to the
dataframe
    # SMA (Simple Moving Average) is the classic moving average that
is the equally weighted average over n periods.
    sma = ta.sma(_df["Close"])
    _df.insert(10, "SMA", sma)

    # apply EMA on the Close column in a df and add it to the
dataframe
    # EMA (Exponential Moving Average). The weights are determined
by alpha which is proportional to it's length.
    ema = ta.ema(_df["Close"])
    _df.insert(11, "EMA", ema)
```

```
    return _df

# apply the function to the dataframe
df1 = assign_TIs(df1)

# drop the NaN values
df1.dropna(inplace=True)

# fix the 'Trend' data type to be int
df1 = df1.astype({'Trend': int})

# check the dataframe
df1.head(5)
```

	Adj Close	Close	High	Low	Open	Volume
Date						
2015-01-05	21.509281	30.977230	31.024668	29.421251	29.743834	149756432
2015-01-12	21.608099	31.119545	31.527514	30.474382	30.996204	145315401
2015-01-19	21.377520	30.787476	31.783682	30.407970	31.280834	138244854
2015-01-26	20.586981	29.648956	31.641365	29.601519	30.806452	180924897
2015-02-02	21.851845	31.470589	31.707781	29.430740	29.829222	197528769

	MACD	Signal	Histogram	RSI	SMA	EMA
Date						
2015-01-05	0.987145	0.987145	0.987145	64.138291	29.636622	29.674931
2015-01-12	1.041722	1.041722	1.041722	64.864548	29.909867	29.937588
2015-01-19	1.046121	1.046121	1.046121	61.723513	30.110057	30.092113
2015-01-26	0.946823	0.946823	0.946823	52.361419	30.185958	30.011539
2015-02-02	1.003551	1.003551	1.003551	62.232156	30.377609	30.276821

	Next	Trend
Date		
2015-01-05	31.119545	1
2015-01-12	30.787476	0
2015-01-19	29.648956	0
2015-01-26	31.470589	1
2015-02-02	32.865276	1

```
# the shape of the data now
df1.shape
```

```
(493, 14)
```

Prepare the data for training

We start by removing the unnecessary columns, only 'Date' for now.

```
# reset the index
df1.reset_index(inplace = True)

# drop the Date column as it's not necessary for now
df1.drop(['Date'], axis=1, inplace=True)

# df1.head(5)
```

Create the features list, for now we will use every column except the last two.

```
# The features list
X1 = df1.iloc[:, :-2]

X1.head(2)
```

```
   Adj Close      Close       High        Low       Open     Volume
MACD  \
0  21.509281  30.977230  31.024668  29.421251  29.743834  149756432
0.987145
1  21.608099  31.119545  31.527514  30.474382  30.996204  145315401
1.041722

     Signal  Histogram        RSI        SMA        EMA
0  0.987145   0.987145  64.138291  29.636622  29.674931
1  1.041722   1.041722  64.864548  29.909867  29.937588
```

Create the target, which is the 'Trend' column for now.

```
# The Target (Trend for now)
y1 = df1.iloc[:, -1]
```

Scaling numerical data is an important step before training machine learning models as it reduces the impact of outliers on the output of the model, giving us more accurate predictions.

Since the features are all numercial data, we can scale them easily to a value in the range of 0 to 1 using scikit-learn minmax scaler.

```
# initialize a MinMaxScaler instance for a range between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))

# pass the features to the scaler
scaled_X1 = scaler.fit_transform(X1)

# scaled_X1
```

Now How are we going to actually predict the trend of a stock for the next week? For now we are going to look back at the last 6 weeks and make a prediction based on that. This is how we are going to train the model therefore we need to reshape the data to be in a sequence of 6 timesteps, (6 rows of data). This is a necessary step for RNN models as the input data need to have the shape of (samples, timesteps, number of features)

```
# source of isnpiration:
https://stackoverflow.com/questions/47945512/how-to-reshape-input-
for-keras-lstm?rq=4 [13]
# create a function to reshape X and y into sequences of x timesteps
def create_seqs(features, target, num_rows):
    # create 2 empty lists to store the newly shaped features and
target lists
    X, y = [], []

    # iterate over the features
    for i in range(len(features) - num_rows):
        # create indexes of the start and end of each sequence
        seq_s = i
        seq_e = i + num_rows

        # the ith sequence will be a slice of the features between
the indexes, create it and add it to X
        xi = features[seq_s : seq_e]
        X.append(xi)

        # do the same for the target and add it to y
        yi = target[seq_e]
        y.append(yi)

    # return the X and y as numpy arraies
    return np.array(X), np.array(y)

# Create sequences
timesteps = 6
X_seq1, y_seq1 = create_seqs(scaled_X1, y1, timesteps)

# check the new shapes for the features and labels sets
X_seq1.shape, y_seq1.shape
```

```
((487, 6, 12), (487,))
```

```
# source:
https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categor
ical [14]
# use to_categorical from tf to converts the target (Trend) to
binary class matrix
y_seq1 = to_categorical(y_seq1)
```

Devide the data into a training set and a test set in 70-30 ratio

```
#  sets the training test ratio to be 70-30
training_ratio = int(len(X_seq1) * 0.7)
```

```
# # split the data into training and test
X1_train, X1_test = X_seq1[:training_ratio], X_seq1[training_ratio:]
y1_train, y1_test = y_seq1[:training_ratio], y_seq1[training_ratio:]

X1_train.shape, X1_test.shape
```

```
((340, 6, 12), (147, 6, 12))
```

Create and train the baseline classification model

The base line model will be SimpleRNN (simple Recurrent Neural Network) classification model with 3 layers. Input layer, hidden layer both with 64 neurons, and output layer. The output layer activation function will be 'softmax', and the model optimizer will be 'adam' for now.

```
# source of inspiration: François Chollet (11, 2017), “Deep Learning
with Python” chapter 6 [8]
# construct the model
def create_model():
    # initialize a sequential model
    model = Sequential()

    # add the model layers
    model.add(SimpleRNN(64, input_shape=(timesteps,
X1_train.shape[2]), return_sequences=True))
    model.add(SimpleRNN(64, return_sequences=False))
    model.add(Dense(2, activation='softmax'))

    # compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

    return model

# initialize the model
model1 = create_model()
```

```
C:\Users\ammaroAsus\anaconda3\Lib\site-packages\keras\src\layers\
rnn\rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```

```
# get the model weights before training
# model1.get_weights()

# train the model
history = model1.fit(X1_train, y1_train, validation_split=0.2,
epochs=50, batch_size=32, verbose=0)
```

Model evaluation and prototype conclusion

```
# test the model accuracy
model1.evaluate(X1_test, y1_test, verbose=2)
```

```
5/5 - 0s - 5ms/step - accuracy: 0.5170 - loss: 0.8761

[0.8761386871337891, 0.5170068144798279]
```

Training this model takes only few seconds, and the accuracy after initialising and training the model few times we sometimes get an accuracy score that matches our common sense base line or slightly beat it, we can tell it's already higher than IBRAHIM KARATAS(2023)[7] work.

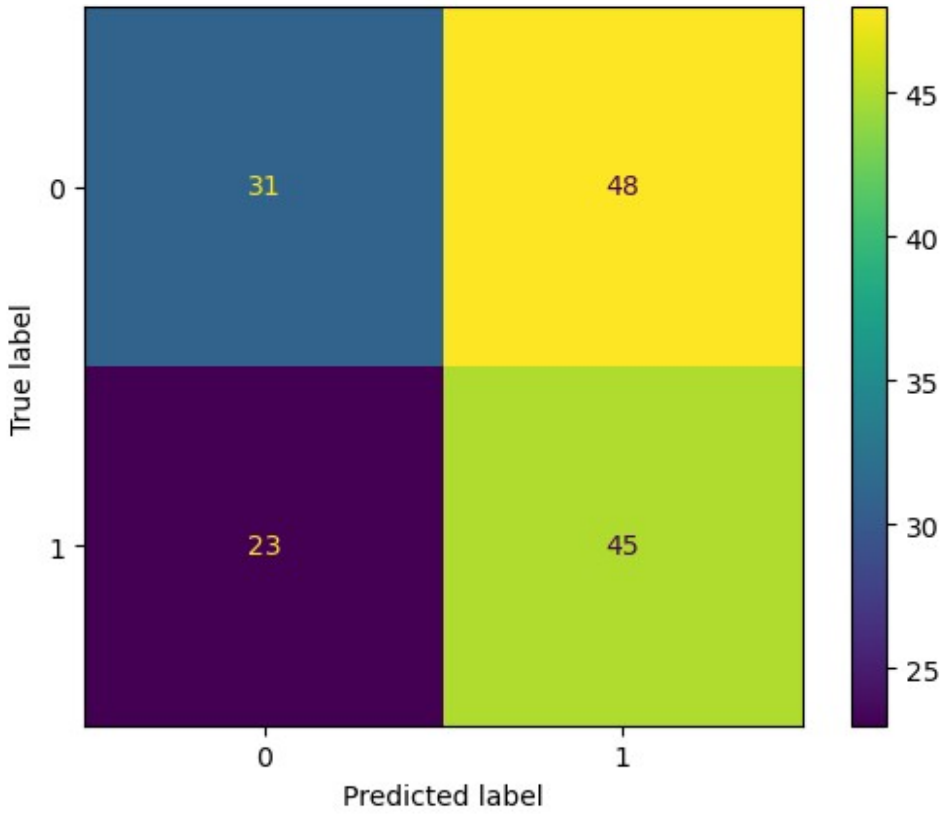
We can also confirm this by getting the precision and f1-score.

```
# get predictions from the model given the test set
y1_pred = model1.predict(X1_test)

# source of inspiration:
https://stackoverflow.com/questions/48987959/classification-metrics-
cant-handle-a-mix-of-continuous-multioutput-and-multi-la [15]
# convert the predictions and test set to be in the shape of a
vector of labels
y1_pred_labels = np.argmax(y1_pred, axis=1)
y1_test_labels = np.argmax(y1_test, axis=1)

# get precision, recall, and fscore
precision, recall, fscore, support =
precision_recall_fscore_support(y1_test_labels, y1_pred_labels,
average='weighted')
print("Precision:", precision)
print("Recall:", recall)
print("F-score:", fscore)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Precision: 0.5323474670632888
Recall: 0.5170068027210885
F-score: 0.5091121770708936
```

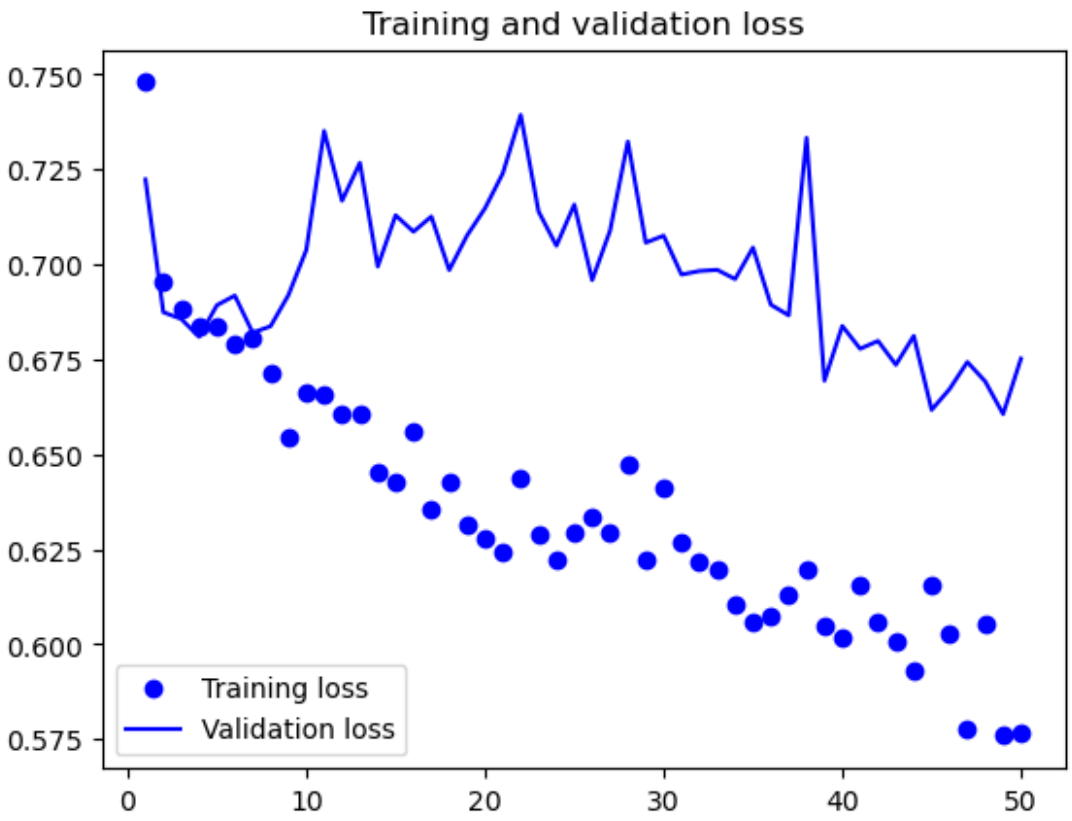
Display the confusion matrix.

```
# source of inspiration:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.Co
nfusionMatrixDisplay.html [16]
conf_mat = confusion_matrix(y1_test_labels, y1_pred_labels)
disp = ConfusionMatrixDisplay(conf_mat)
disp.plot()
plt.show()
```



Looking at the following plot, we can see that we are hitting the vanishing returns point before we reach the 15th epoch, this means increasing the epochs will not help us at this point.

```
# source of the code snippet[17]
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```




```
# get the model summary
# model1.summary()

# get the model compile configurations
# model1.get_compile_config()

# get the model configurations after training
# model1.get_config()

# get the model weights after training
# model1.get_weights()
```

Get a prediction from the model given the last 6 weeks. This is to simulate how a user would get a prediction from the model. The input will be the last entry in the test set.

```
# reshape the input so it have the shape (1, 6, 12) which what the
model expect as input
_input = X1_test[-1].reshape(1, X1_test.shape[1], X1_test.shape[2])
pred = model1.predict(_input)
print(f"Trend: {np.argmax(pred)}, ",f"Confidence: {np.max(pred)}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Trend: 1,  Confidence: 0.7864111065864563
```

From here, we can start iteratively building more accurate models. The final product should look like the following snippet, where users interact with the model through a dictionary of dictionaries. Each dictionary includes the associated models and their functionalities for a given stock symbol.

```
# this is just an example for iillustrative purposes
# {
#     'PFE':
#     {
#         'classification_model': {'model': model1, 'interval':
'week', 'predict_function':model1_predict},
#         'regression_model': {'model': None, 'interval': 'week',
'predict_function':None}
#     },
# }
```

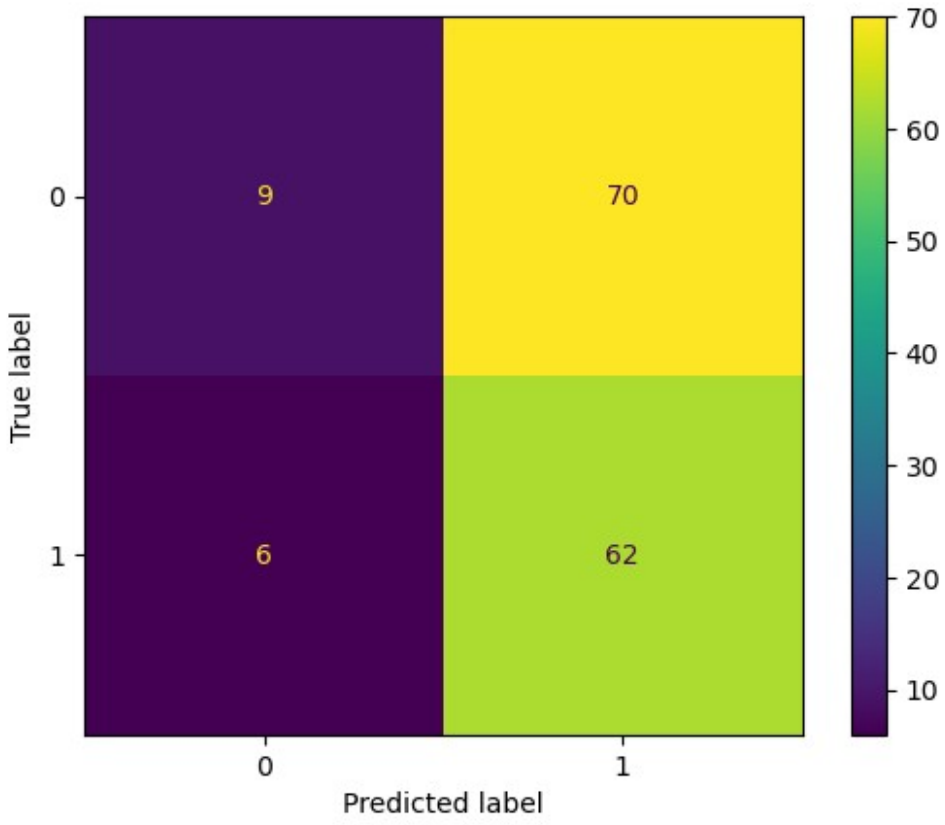
Now that we created our first model, let's create some other models so we can compare their performance against each others. For now we will only add an LSTM model (Long short-term memory recurrent neural network) and GRU model (Gated recurrent unit).

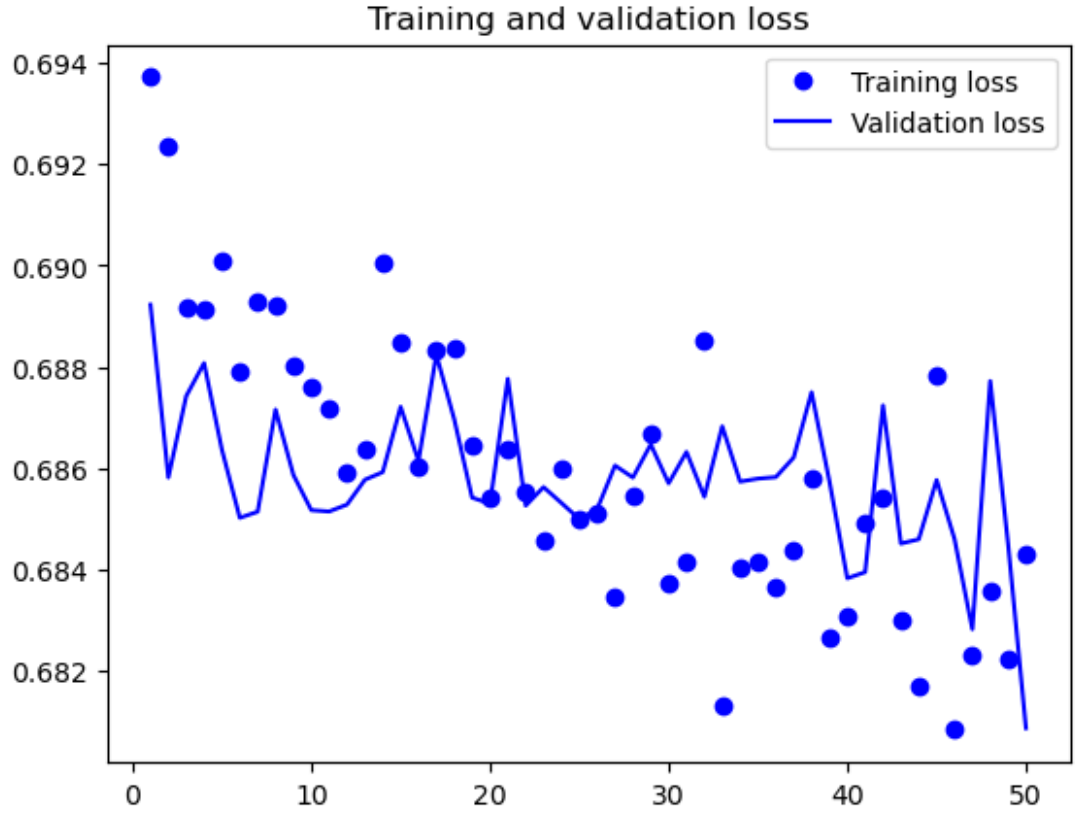
LSTM

```
# construct LSTM the model
def create_LSTM_model():
    # initialize a sequential model
    model = Sequential()

    # add the model layers
    # input layer
    model.add(Input(shape=(timesteps, X1_train.shape[2])))

    # first dense layer
    model.add(LSTM(64, return_sequences=True))

    # second dense layer
    model.add(LSTM(64, return_sequences=False))

    # output layer
    model.add(Dense(2, activation='softmax'))

    # compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

    return model

# initialize the model
model1 = create_LSTM_model()

# train the model
history = model1.fit(X1_train, y1_train, validation_split=0.2,
epochs=50, batch_size=32, verbose=0)

# test the model accuracy
model1.evaluate(X1_test, y1_test, verbose=2)

# get predictions from the model given the test set
y1_pred = model1.predict(X1_test)
```

```
# source of inspiration:
https://stackoverflow.com/questions/48987959/classification-metrics-cant-handle-a-mix-of-continuous-multioutput-and-multi-la [15]
# convert the predictions and test set to be in the shape of a vector of labels
y1_pred_labels = np.argmax(y1_pred, axis=1)
y1_test_labels = np.argmax(y1_test, axis=1)

# get precision, recall, and fscore
precision, recall, fscore, support =
precision_recall_fscore_support(y1_test_labels, y1_pred_labels,
average='weighted')
print("Precision:", precision)
print("Recall:", recall)
print("F-score:", fscore)

# source of inspiration:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html [16]
conf_mat = confusion_matrix(y1_test_labels, y1_pred_labels)
disp = ConfusionMatrixDisplay(conf_mat)
disp.plot()
plt.show()

# source of the code snippet[17]
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
5/5 - 0s - 5ms/step - accuracy: 0.4830 - loss: 0.7560
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
Precision: 0.5397237682951969
Recall: 0.48299319727891155
F-score: 0.3897119698943407
```


9
70
6
62
0
1
True label
0
1
Predicted label
70
60
50
40
30
20
10


Training and validation loss
Training loss
Validation loss
0.694
0.692
0.690
0.688
0.686
0.684
0.682
0
10
20
30
40
50

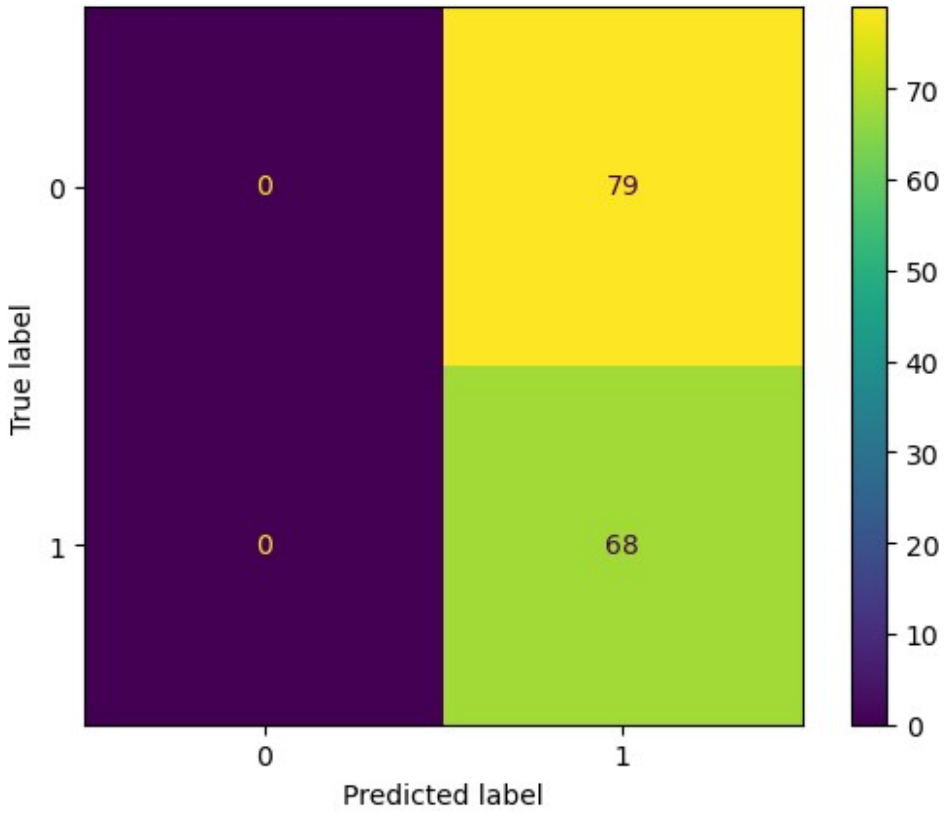
GRU

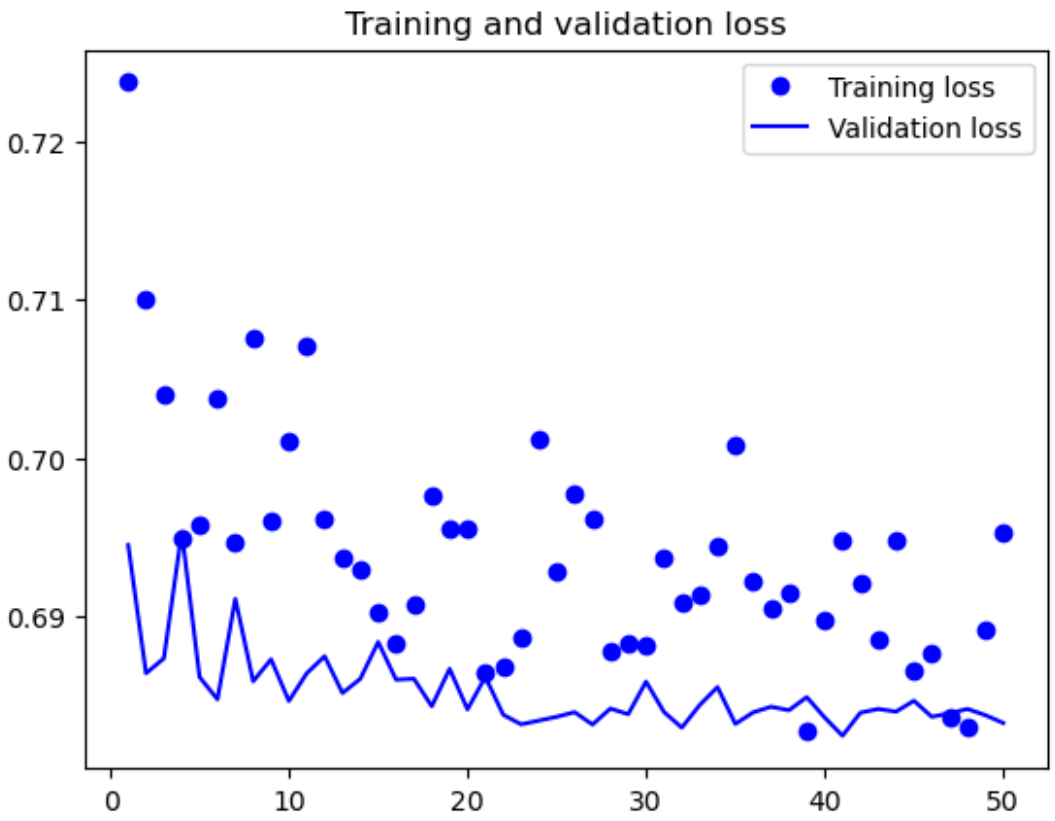
```
# source of inspiration: François Chollet (11, 2017), “Deep Learning
with Python” chapter 6 [8]
# construct the model
def create_GRU_model():
    # initialize a sequential model
    model = Sequential()

    # add the model layers
    # input layer
    model.add(Input(shape=(timesteps, X1_train.shape[2])))

    # first dense layer
    model.add(GRU(64,
                  dropout=0.1,
                  recurrent_dropout=0.5,
                  return_sequences=True))

    # second dense layer
    model.add(GRU(64,
                  dropout=0.1,
                  recurrent_dropout=0.5))

    # output layer
    model.add(Dense(2, activation='softmax'))

    # compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

    return model

# initialize the model
model1 = create_GRU_model()

# train the model
history = model1.fit(X1_train, y1_train, validation_split=0.2,
epochs=50, batch_size=32, verbose=0)

# test the model accuracy
model1.evaluate(X1_test, y1_test, verbose=2)

# get predictions from the model given the test set
y1_pred = model1.predict(X1_test)

# source of inspiration:
https://stackoverflow.com/questions/48987959/classification-metrics-
cant-handle-a-mix-of-continuous-multioutput-and-multi-la [15]
# convert the predictions and test set to be in the shape of a
vector of labels
y1_pred_labels = np.argmax(y1_pred, axis=1)
y1_test_labels = np.argmax(y1_test, axis=1)

# get precision, recall, and fscore
```

```
precision, recall, fscore, support =
precision_recall_fscore_support(y1_test_labels, y1_pred_labels,
average='weighted')
print("Precision:", precision)
print("Recall:", recall)
print("F-score:", fscore)

# source of inspiration:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.Co
nfusionMatrixDisplay.html [16]
conf_mat = confusion_matrix(y1_test_labels, y1_pred_labels)
disp = ConfusionMatrixDisplay(conf_mat)
disp.plot()
plt.show()

# source of the code snippet[17]
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
5/5 - 0s - 5ms/step - accuracy: 0.4626 - loss: 0.7142
WARNING:tensorflow:5 out of the last 12 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_di
stributed at 0x0000024F2FF06340> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the
loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
1/5 ━━━━━━━━━━━━━━━━━━━━ 1s 281ms/stepWARNING:tensorflow:5 out of
the last 12 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_di
stributed at 0x0000024F2FF06340> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the
loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
```

```
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step
Precision: 0.2139849136933685
Recall: 0.46258503401360546
F-score: 0.2926119284923272

C:\Users\ammaroAsus\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```





Notice how the GRU model seems to be incapable of outputting anything but 1. This is an issue that we must address before we actually compare this model with the other ones.

Now le'ts create a regression version of all the 3 models we have so far. First let's adjust the target column of the model, for refression the column we are trying to predict is the Next colum.

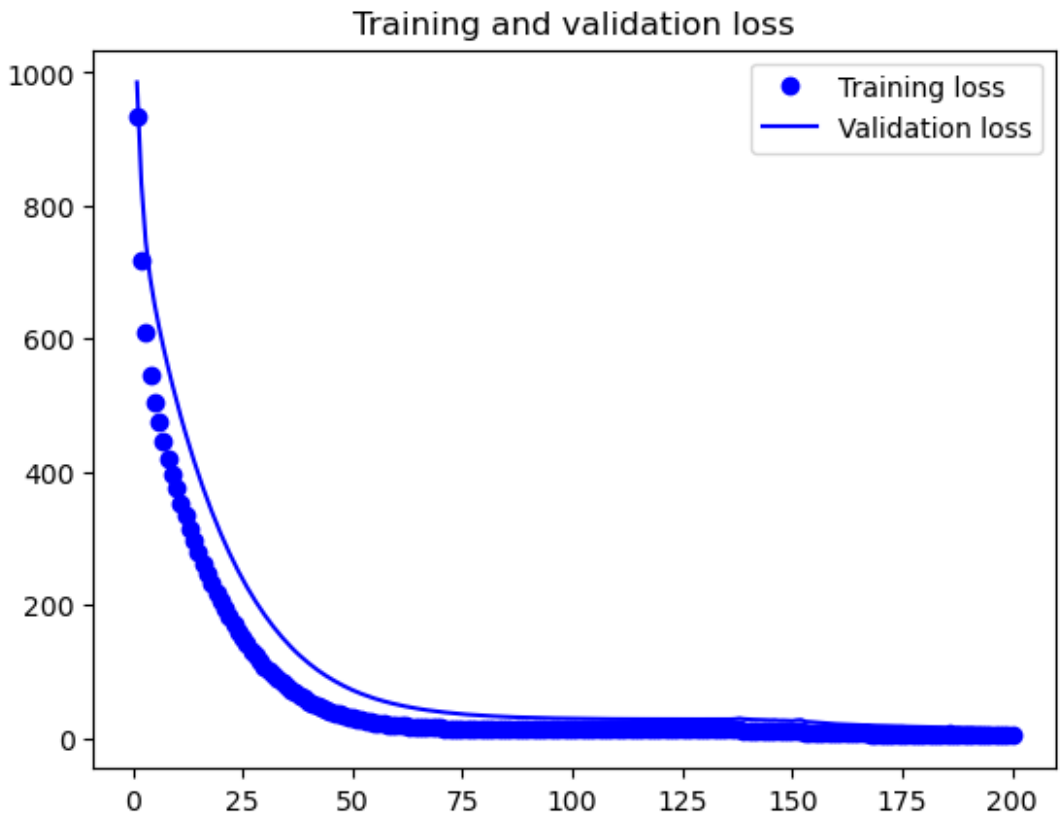
```
# The Target (Next for now)
y1_reg = df1.iloc[:, -2]

# Create sequences
timesteps = 6
X_seq1_reg, y_seq1_reg = create_seqs(scaled_X1, y1_reg, timesteps)

# split the data into training and test 70-30 ratio
X1_train_reg, X1_test_reg = X_seq1_reg[:training_ratio],
X_seq1_reg[training_ratio:]
y1_train_reg, y1_test_reg = y_seq1_reg[:training_ratio],
y_seq1_reg[training_ratio:]
```

SimpleRNN regression model

```
# construct the model
def create_SimpleRNN_Reg_model():
    # initialize a sequential model
    model = Sequential()

    # add the model layers
    model.add(Input(shape=(timesteps, X1_train.shape[2])))
```

```
    model.add(SimpleRNN(64, return_sequences=True))
    model.add(SimpleRNN(64, return_sequences=False))
    model.add(Dense(1))

    # compile the model
    model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae'])

    return model

from sklearn.metrics import r2_score

# initialize the model
model1 = create_SimpleRNN_Reg_model()

# train the model
history = model1.fit(X1_train_reg, y1_train_reg,
validation_split=0.2, epochs=200, batch_size=32, verbose=0)

# test the model accuracy
model1.evaluate(X1_test_reg, y1_test_reg, verbose=2)

# get predictions from the model given the test set
y1_pred = model1.predict(X1_test_reg)

# get the R2 of the model
r2 = r2_score(y1_test_reg, y1_pred)
print(f"R2 score is: {r2}")


# source of the code snippet[17]
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

```
5/5 - 0s - 4ms/step - loss: 98.3125 - mae: 8.0684
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
R2 score is: -0.04270364949012895
```

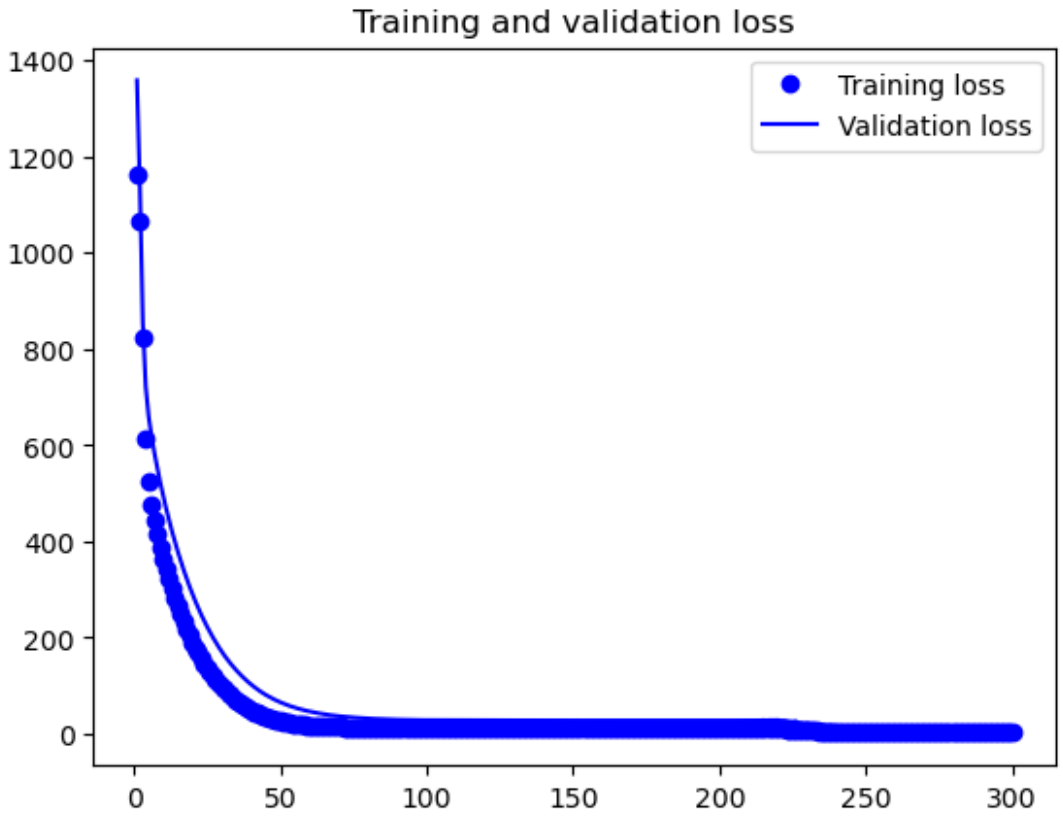


LSTM regression model

```
# construct the model
def create_LSTM_Reg_model():
    # initialize a sequential model
    model = Sequential()

    # add the model layers
    model.add(Input(shape=(timesteps, X1_train.shape[2])))
    model.add(LSTM(64, return_sequences=True))
    model.add(LSTM(64, return_sequences=False))
    model.add(Dense(1))

    # compile the model
    model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae'])

    return model

# initialize the model
model1 = create_LSTM_Reg_model()

# train the model
history = model1.fit(X1_train_reg, y1_train_reg,
validation_split=0.2, epochs=300, batch_size=32, verbose=0)

# test the model accuracy
model1.evaluate(X1_test_reg, y1_test_reg, verbose=2)
```

```python
# get predictions from the model given the test set
y1_pred = model1.predict(X1_test_reg)

# get the R2 of the model
r2 = r2_score(y1_test_reg, y1_pred)
print(f"R2 score is: {r2}")

# source of the code snippet[17]
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

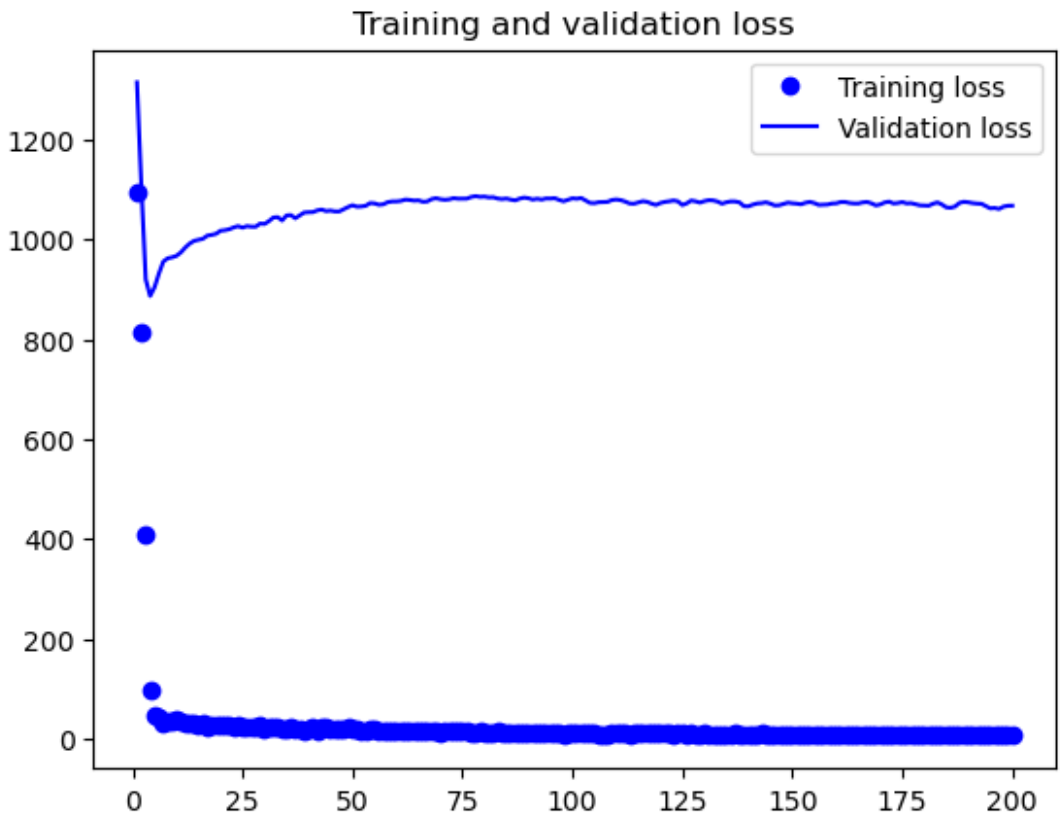
```
5/5 - 0s - 5ms/step - loss: 59.0259 - mae: 6.0010
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
R2 score is: 0.3739699642515384
```



GRU regression model

```
# construct the model
def create_GRU_Reg_model():
    # initialize a sequential model
    model = Sequential()

    # add the model layers
    model.add(Input(shape=(timesteps, X1_train.shape[2])))
    model.add(GRU(64, dropout=0.1, recurrent_dropout=0.5,
return_sequences=True))
    model.add(GRU(64, dropout=0.1, recurrent_dropout=0.5))
    model.add(Dense(1))

    # compile the model
    model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae'])

    return model

# initialize the model
model1 = create_GRU_Reg_model()

# train the model
history = model1.fit(X1_train_reg, y1_train_reg,
validation_split=0.2, epochs=200, batch_size=32, verbose=0)

# test the model accuracy
model1.evaluate(X1_test_reg, y1_test_reg, verbose=2)

# get predictions from the model given the test set
y1_pred = model1.predict(X1_test_reg)

# get the R2 of the model
r2 = r2_score(y1_test_reg, y1_pred)
print(f"R2 score is: {r2}")

# source of the code snippet[17]
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
5/5 - 0s - 5ms/step - loss: 1341.1838 - mae: 35.2960
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step
R2 score is: -13.224618380586763
```



We notice that the GRU model also did perform the worst here, which increase the need to perform further investigation. For the other models we got a low R2 score as well as MAE of 6.4-6.9 which doesn't mean a lot until we check the average value of the Next colum which we are trying to predict. this MAE is relatively high and make the regression models not useful for real world usage.

```
# the average value for the Next closing price to judge weather the
mae is acceptable or not
df1['Next'].mean()
```

```
36.691718606387866
```

Hyperparameters optimization

We will use RandomizedSearchCV from sikit-learn to perform hyperparameter optimization on the SimpleRNN classification model for now, at this stage we only want to confirm that this approach works, at a later stage we will perform more in depth experiments with different models as well as different Hyperparameters optimization algorithms. To perform such operation we need to convert our tensorflow model to be compatible with the scikit-learn library. To do so we will utilize the scikeras to warp the model first, then perform Hyperparameters optimization tunning on that warpped version of the model.

```
# !pip install scikeras

# helper function to measure how long a process would take
from datetime import datetime
```

```
def get_time():
    return datetime.now()

from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier, KerasRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import warnings


# get the time before starting the process
start = get_time()

# source of inspiration:
https://stackoverflow.com/questions/72392579/scikeras-
randomizedsearchcv-for-best-hyper-parameters [19]
# construct the model
def create_model(n_hidden = 1, n_neurons = 30, learning_rate=3e-3):
    # initialize a sequential model
    model = Sequential()

    # create an input layer
    model.add(Input(shape=(timesteps, X1_train.shape[2])))

    # add a static first deep layer
    model.add(SimpleRNN(n_neurons, return_sequences=True))

    # add the other model deep layers dynamically
    for layer in range(1, n_hidden):
        model.add(SimpleRNN(n_neurons))

    # output layer
    model.add(Dense(2, activation='softmax'))

    # create an Adam optimizer with a variable learning rate
    opt = Adam(learning_rate=learning_rate)

    # compile the model
    model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

    return model


# define possible parameters (this is just a test for now and we
will add more parameters for the final report)
# we need to distingiush between the model building function input
and the RandomizedSearchCV input, to do that we prefix the model
input with model__
# source:
https://adriangb.com/scikeras/stable/notebooks/Basic_Usage.html (7.1
Special prefixes) [20]
params = {
    "model__n_hidden": [0, 1, 2, 3, 4, 5],
    "model__n_neurons": [int(x) for x in np.arange(1, 128)],
```

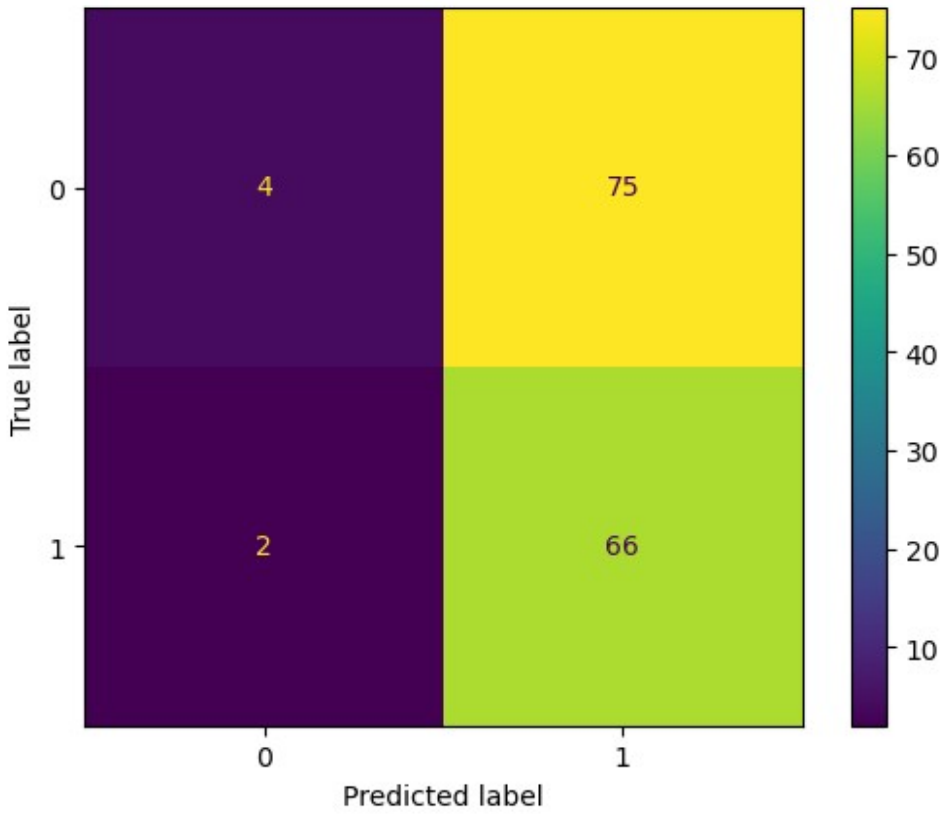
```
    "model__learning_rate": [1e-2, 1e-3, 1e-4],
    'epochs': [10, 20, 30],
    'batch_size': [16, 32, 64]
}

# create a keras classification model wrappers from the scikeras
library which allow us to utilize the hyperparameter tunning
functions from the scikit-learn library
kerasWarp = KerasClassifier(model=create_model, verbose=0)

# RandomizedSearchCV
random_simpleRnn_clas = RandomizedSearchCV(estimator=kerasWarp,
param_distributions=params, n_iter=10, cv=None, random_state=101)

# source of inspiration:
https://stackoverflow.com/questions/40105796/turn-warning-off-in-a-
cell-jupyter-notebook [21]
# prevent FitFailedWarning
with warnings.catch_warnings():
    warnings.simplefilter('ignore')

    # fit the RandomizedSearchCV models
    random_simpleRnn_clas.fit(X1_train, y1_train,
validation_split=0.2)

# print results
print(f"RandomizedSearchCV best parameters:
{random_simpleRnn_clas.best_params_}")
print(f"RandomizedSearchCV best score:
{random_simpleRnn_clas.best_score_}")


# get the time after finishing the process
end = get_time()

# print the duration
print(f"process finished in: {end - start}")
```

```
RandomizedSearchCV best parameters: {'model__n_neurons': 108,
'model__n_hidden': 2, 'model__learning_rate': 0.001, 'epochs': 30,
'batch_size': 16}
RandomizedSearchCV best score: 0.5529411764705883
process finished in: 0:00:46.003049
```

```
# retrieve the best model and refit on the training data to get the
history
best_model = random_simpleRnn_clas.best_estimator_.model_

# evaluate the best model on the test data
best_model.evaluate(X1_test, y1_test, verbose=1)

# get predictions from the model given the test set
y1_pred = best_model.predict(X1_test)

# convert the predictions and test set to be in the shape of a
```

```
vector of labels
y1_pred_labels = np.argmax(y1_pred, axis=1)
y1_test_labels = np.argmax(y1_test, axis=1)

# get precision, recall, and fscore
precision, recall, fscore, support =
precision_recall_fscore_support(y1_test_labels, y1_pred_labels,
average='weighted')
print("Precision:", precision)
print("Recall:", recall)
print("F-score:", fscore)

# Confusion Matrix
conf_mat = confusion_matrix(y1_test_labels, y1_pred_labels)
disp = ConfusionMatrixDisplay(conf_mat)
disp.plot()
plt.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4478 - loss:
0.8583
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Precision: 0.574805808848362
Recall: 0.47619047619047616
F-score: 0.3427392009435353
```




```
# get a summary of the best model
best_model.summary()
```

```
Model: "sequential_158"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ simple_rnn_308 (SimpleRNN)           │ (None, 6, 108)              │          13,068 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ simple_rnn_309 (SimpleRNN)           │ (None, 108)                 │          23,436 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_98 (Dense)                     │ (None, 2)                   │             218 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 110,168 (430.35 KB)
 Trainable params: 36,722 (143.45 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 73,446 (286.90 KB)
```

we see we have relatively better results than the base line SimpleRNN classification model when it come to the confussion matrix, which means we are on the right direction.

I also experimented with gridSearchCV from scikit learn as well as Hyperband from keras tuner by following the instructions in this notebook from tensorflow official documentation https://www.tensorflow.org/tutorials/keras/keras_tuner However I will not include that in this draft submission as I ran out of time to add them in a presentable way.

5. Evaluation

Model	Configurations	Results
baseline SimpleRNN classification model	{layers: (layer1(units=64, return_sequences=True), layer2(units=64, return_sequences=False), output_layer(units=2, activation='softmax')), optimizer='adam', loss='categorical_crossentropy', epochs=50, batch_size=32}	{accuracy: 0.5102, loss: 0.8259, Precision: 0.5227, Recall: 0.5102, F-score: 0.5047}
baseline LSTM classification model	{layers: (layer1(units=64, return_sequences=True),	{accuracy: 0.4626, loss: 0.7830, Precision: 0.4802,

Model	Configurations	Results
	layer2(units=64, return_sequences=False), output_layer(units=2, activation='softmax')), optimizer='adam', loss='categorical_crossentropy', epochs=50, batch_size=32}	Recall: 0.4625, F-score: 0.3768}
baseline GRU classification model	{layers: (layer1(units=64, dropout=0.1, recurrent_dropout=0.5, return_sequences=True), layer2(units=64, dropout=0.1, recurrent_dropout=0.5), output_layer(units=2, activation='softmax')), optimizer='adam', loss='categorical_crossentropy', epochs=50, batch_size=32}	{accuracy: 0.4626, loss: 0.7179, Precision: 0.2139, Recall: 0.4625, F-score: 0.2926}
SimpleRNN regression model	{layers: (layer1(units=64, return_sequences=True), layer2(units=64, return_sequences=False), output_layer(units=1)), optimizer='adam', loss='mean_squared_error', metrics=mae, epochs=200, batch_size=32}	{loss: 74.4604, mae: 6.9609, R2 score: 0.2103}
LSTM regression model	{layers: (layer1(units=64, return_sequences=True), layer2(units=64, return_sequences=False), output_layer(units=1)), optimizer='adam', loss='mean_squared_error', metrics=mae, epochs=300, batch_size=32}	{loss: 63.3322, mae: 6.4408, R2 score: 0.3283}
GRU regression model	{layers: (layer1(units=64, dropout=0.1, recurrent_dropout=0.5, return_sequences=True), layer2(units=64, dropout=0.1, recurrent_dropout=0.5), output_layer(units=1)), optimizer='adam', loss='mean_squared_error', metrics=mae, epochs=200,	{loss: 1275.4667, mae: 34.2446, R2 score: -12.5277}

Model	Configurations	Results
	batch_size=32}	
SimpleRNN classification model with RandomizedSearchCV Hyperparameters optimization	{'model__n_neurons': 108, 'model__n_hidden': 2, 'model__learning_rate': 0.001, 'epochs': 30, 'batch_size': 16}	{accuracy: 0.4997, loss: 2.6462, Precision: 0.5389, Recall: 0.5238, F-score: 0.5168 }

6. Conclusion

So far we have managed to create a total of 6 baseline models between regression and classification models, we also managed to perform hyperparameter optimization which enhanced the performance of the model we applied. So far all the results are relatively low and barely outperform our established common-sense baseline, however, we must remember that we haven't started doing any feature selection yet, so the training dataset is extremely limited at this stage but we want to establish a robust training pipeline before we start working on our feature selection. This will make our process more organized and yield the best possible results.

I know this is not a complete project but this is what I could come up with since I only had 4 days to work on this draft, in the past few weeks I didn't work on the final project as I was busy with my other modules midterm, therefore this is not a representation of the final report but just a very early draft.

7. References

Note: we used the ACM citation style for the referencing.

[1] Arielle O'Shea (2023). "What is a stock?". Nerd wallet. (Mar 31, 2023). Retrieved Jun 14, 2024 from https://www.nerdwallet.com/article/investing/what-is-a-stock#:~:text=A%20stock%20is%20a%20security,increases%20in%20value%20as%20well

[2] "Algorithmic Trading Market - Growth, Trends, COVID-19 Impact, and Forecasts (2022 - 2027)". Yahoo finance. (Mar 18, 2022). Retrieved Jun 14, 2024 from https://finance.yahoo.com/news/algorithmic-trading-market-growth-trends-112400870.html

[3] ADAM HAYES, Gordon Scott, DAVID RUBIN (2024). investopedia. "Position Definition—Short and Long Positions in Financial Markets". (May 31, 2024). Retrieved Jun 14, 2024 from https://www.investopedia.com/terms/p/position.asp#:~:text=our%20editorial%20policies-,What%20Is%20a%20Position%3F,short%20securities%20with%20bearish%20intent.

[4] Nuti, Giuseppe (11/2011). "Algorithmic Trading". Computer (Long Beach, Calif.) (0018-9162), 44 (11), p. 61, Retrieved April 21,2024 from https://ieeexplore.ieee.org/document/5696713

[5] Alzaman, Chaher (03/2024). "Deep learning in stock portfolio selection and predictions". Expert systems with applications (0957-4174), 237 , p. 121404, Retrieved April 21,2024 from https://www.sciencedirect.com/science/article/pii/S0957417423019061

[6] B L, Shilpa (2023). "Deep Learning Models for Stock Market Prediction Using Optimization Approach" in 2023 International Conference on Network, Multimedia and Information Technology (NMITCON) (979-83-503-0083-3), (p. 1), Retrieved April 21,2024 from https://ieeexplore.ieee.org/document/10275882

[7] IBRAHIM KARATAS (2023). "Machine Learning for Stocks Trading". Kaggle. (2023). Retrieved April 21,2024 from https://www.kaggle.com/code/ibrahimkaratas/machine-learning-for-stocks-trading/notebook

[8] François Chollet (11, 2017), "Deep Learning with Python", chapters (6, 7), Manning Publications Co, Retrieved May 14, 2024 from https://www.manning.com/books/deep-learning-with-python

[9] S&P 500 ETF Components. Slickcharts. (2024). Retrieved June 6, 2024 from https://www.slickcharts.com/sp500

[10] pandas.DataFrame.stack. Pandas. (2024). Retrieved May 16, 2024 from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.stack.html

[11] pandas.DataFrame.xs. Pandas. (2024). Retrieved May 16, 2024 from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.xs.html

[12] ADAM HAYES, ERIC ESTEVEZ, Yarilet Perez (2023). "Behavioral Finance". "What Is Behavioral Finance?". investopedia. (Dec 19, 2023). Retrieved Jun 14, 2024 from https://www.investopedia.com/terms/b/behavioralfinance.asp

[13] mpariente (2017). "How to reshape input for keras LSTM?". stack overflow. (Dec 22, 2017). Retrieved May 20, 2024 from https://stackoverflow.com/questions/47945512/how-to-reshape-input-for-keras-lstm?rq=4

[14] tf.keras.utils.to_categorical. TensorFlow. (2024). Retrieved May 26, 2024 from https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical

[15] ericheindl (2021), "classification metrics can't handle a mix of continuous-multioutput and multi-label-indicator targets". stack overflow. (May 11, 2021). Retrieved Jun 15, 2024 from https://stackoverflow.com/questions/48987959/classification-metrics-cant-handle-a-mix-of-continuous-multioutput-and-multi-la

[16] sklearn.metrics.ConfusionMatrixDisplay. Sci-kit learn. (2024). Retrieved Jun 15, 2024 from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

[17] François Chollet (11, 2017), "Deep Learning with Python", chapters (6), Listing 6.38. Plotting results, Manning Publications Co, Retrieved Jun 15, 2024 from https://www.manning.com/books/deep-learning-with-python

[18] Dataset obtained using yfinance: https://pypi.org/project/yfinance/ (""" it's an open-source tool that uses Yahoo's publicly available APIs, and is intended for research and educational purposes. """)

[19] ecatanzani (2022), "SciKeras - RandomizedSearchCV for best hyper-parameters". stack overflow. (May 26, 2022). Retrieved Jul 20, 2024 from https://stackoverflow.com/questions/72392579/scikeras-randomizedsearchcv-for-best-hyper-parameters

[20] Scikeras tutorials, Basic usage. "7.1 Special prefixes". SciKeras's documentation (2024). Retrieved jul 21, 2024 from https://adriangb.com/scikeras/stable/notebooks/Basic_Usage.html

[21] mins(2021), pixelistik (2023), "turn warning off in a cell jupyter notebook". stack overflow. (Jan 20, 2021). Retrieved Jul 22, 2024 from https://stackoverflow.com/questions/40105796/turn-warning-off-in-a-cell-jupyter-notebook