KA$H: Knowledgeable Automated Stock Handler

14:332:452 Software Engineering - Group 7

4/14/2019: Third Report P1

Website URL
https://kash-stock.herokuapp.com/

TEAM LEADERS
Sunny Feng and Parker Fisher

TEAM MEMBERS
Sunny Feng, Parker Fisher, Nick Heah, Manish Kewalramani, Andrea Dumalagan, Varun Ravichandran, Amany Elgarf, Asmaa Hasan, Jon Tsai

Contribution Breakdowns

Sunny Feng (Team Leader and Technical Forecaster Team):	11%
Parker Fisher (Team Leader and Automated Trader Team):	11%
Nick Heah (Technical Forecaster Team):	11%
Manish Kewalramani (Technical Forecaster Team):	11%
Andrea Dumalagan (Sentiment Analyzer Team):	11%
Varun Ravichandran (Sentiment Analyzer Team):	11%
Amany Elgarf (Sentiment Analyzer Team):	11%
Asmaa Hasan (Automated Trader Team):	11%
Jon Tsai (Automated Trader Team):	11%

Table of Contents

0. Summary of Changes

Included Fast Fourier Transform information under Sections 2, 6, and 7
Replaced sequence, interaction and class diagrams with updated graphics
Updated requirements and use cases to better reflect our completed work
Updated domain analysis and association diagrams,
Updated all associated traceability matrix
Replaced hand-drawn images with updated graphics
Updated project size estimation
Updated interface specification to contain accurate implementations
Included OCL contracts section
Expanded upon design of tests
Updated Section 13 to reflect status after Demo 2

1. Customer Statement of Requirements

Customer Problem

The stock market is one of the best ways to earn money, but to the everyday person like me, it seems so inaccessible. As a customer with limited knowledge in stocks, I find it difficult to understand how exactly it is possible for anyone to use the stock market to make money and the complicated jargon surrounding stocks makes it really hard for a person like me with no financial background to make use of it. Consequently, I feel like I'm missing out on the opportunities the stock market offers to other people, more specifically the possibly massive amounts of money that can be made using it. I don't want to fall behind on putting my foot in the potentially great earnings from the economy, and it feels like the smart investors are getting ahead of me with their vast and trained experiences in this market.

A service that would help me navigate the stock market and make informed decisions about investing would be a valuable tool. If the service just did it for me, that would be even better. While investing in stocks is certainly risky for someone with no knowledge of the stock market, having an educated and experienced aide to minimize risk is already a possible solution for some people. However not everyone can afford hiring someone to do that, and I in particular cannot. For people like me with limited knowledge and time, the perceived risk of investing outweighs the potential benefits, as someone like me who knows absolutely nothing would be risking too much for too little gain. If, however, it was possible for me to find a tool that would help recommend stocks for me to buy or sell based on my budget and companies of interest, I could easily transform the prospect of investing in the stock market from certain financial doom to a plausible source of income. Even with simply knowing that company performance affect stocks, it requires a lot of time researching the internet, reading articles and keeping up with financial and economic newsletters, which is a tedious activity. A tool that could relate the news, the performance and the stock price history and put everything in one place makes it significantly easier to see the bigger picture and make better decisions.

Currently, I believe that the services that predict the future prices of stocks are questionable. Stock prices are difficult to be accurately predicted because we can only predict by looking at the stock history. Behavioral economists say that the imperfections are a combination of cognitive biases and other human errors. It seems like it is really hard to predict stocks since humans are so unpredictable and we react to bad news or stock price drops emotionally. Stock prices also don't follow a pattern really, which makes it hard to guess what will happen next by looking at previous stock prices.

I think that one of the reasons that we have trouble predicting future stock prices is because we tend to give more weight to current or easily recalled events instead of considering all relevant information. This means that we, sometimes, make wrong decisions based on inaccurate or lack of information. For me, to be able to invest with knowledge and confidence, I'd like to have something that can look at the existing data online and help me decide how to invest my money wisely. However, I have noticed some shortcomings the applications I have looked at.

Firstly, I have noticed that most models used by the stock forecasters I have seen were not tested based on performance, and it wasn't clear whether the forecasters could successfully predict stock prices. While the forecasters did go over potential methodologies to check against their models, most did not not implement these features to determine the accuracy of their predictions. I feel that these tests are essential to evaluating whether these prediction models would actually help me make financially sound investments. The lack of specific numbers and quantifications do not reassure me that they used tested and effective methods, and without this data, I don't think I would be confident in using any of these current stock forecasters.

Secondly, these existing models only take into account stock prices for their predictions. I believe that the sentiment of the company, both positive and negative, which I could be interested in, is really important as well, since company stock prices drop when bad news comes out, and may trend upwards when good financial news comes out. Even some big current events, such as the recent trade war and the decisions made by the Chairman of the Federal Reserve, can send waves of impact to stocks all across the exchange. Having a stock forecaster that would consider the current sentiment of the company and of the current economy, in addition to the previous stock prices, would give me a more accurate and reactive prediction to base my trading off of.

Lastly, buying and selling stocks are really difficult to learn on my own. I usually have to rely on my gut feelings and make choices that may seem good at the moment, but aren't actually in my best interest. Having an automated trader that would take the predictions and buy/sell/hold my stocks for me would be really helpful. I don't want to stay glued to my screen to keep track of all the changing and shifting stock prices, so with an automated trader that provides me with up to date and accurate predictions for trending stocks,, I will be able to let the automated trader make money for me and I can have stock trading as a supplementary source of income. With this ease of mind, I can dedicate myself to other endeavours such as taking steps in learning how to invest in the stock market on my own.

Proposed Solution to Problem

As a customer, I would love to have more accurate predictions made so that I can make the best trades, I also would like to have an automated system that trades for me so I don't have to stay glued to my screen and keep track of stock trends in my own time. I would like to stay away from stock brokers as they don't seem to always have my best interest in mind. They have a reputation of undercutting people like me and I would rather avoid them, and feel more comfortable with an automated option. I want to invest but I have little free time to do so and I'm a beginner who doesn't know what to actually do.. People who are low on time, who are investing for the first time, and skilled traders who are looking for better recommendations would all benefit from an automated trader that took in sentiment in its predictions. As a beginner, I feel clueless about how to trade stocks and current forecasters don't help me at all. I am also worried about the accuracy and the testing done for the existing forecasters.

A veteran trader would also benefit from more accurate predictions of future stocks and the sentiment analysis of the current stocks. By using sentiment analysis and general news to help in the technical predictions, I will be able to see more accurate predictions and choose to run the automated trader to buy/sell/hold for me. Using a product such as this will greatly increase my free time, and I will be able to use the resources available to personalize my portfolio to the most optimal extent.

2. Glossary of Terms

Accumulative Swing Index: A variation of Welles Wilder's swing index, comparing high, low, opening and closing prices in a given time.
Algorithm: A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
API: An application program interface; A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.
Artificial Intelligence (AI): The capability of a machine to imitate intelligent human behavior.
Ask price: The lowest price a prospective seller is willing to accept.
Automate: To make something operate automatically by using machines or computers.
Autoregressive Integrated Moving Average (ARIMA): A regressive model using random-walk and random-trend for stock prediction.
Auto-Trader: KA$H's automatic stock trading bot that utilizes deep learning powered by Tensorflow and the stock prediction/sentiment data gathered by Alphavantage/Google News API to trade stocks for investors in real-time.
Bid price: The highest price that a prospective buyer is willing to pay.
Backpropagation: Neural Network training algorithm to lower the rate of error across iterations
Closing price: The price of a stock share at the end of the day in a financial market.
Database: A structured set of data held in a computer, especially one that is accessible in various ways.
Dividend: A payment made by a corporation to its shareholders.
Dow Jones Industrial Average (DJIA): A price-weighted average of 30 significant stocks traded on the New York Stock Exchange (NYSE) and the Nasdaq.

Exchange: An organized market where commodities such as stock shares are sold and bought.
Fast Fourier Transform: An algorithm that transforms a signal to use for prediction
Forecaster: An entity that predicts or estimates a future event or trend.
Machine learning: An application of artificial intelligence (AI) that allows systems to automatically learn and improve from experience without being explicitly programmed.
Market Sector: A specific part of the economy
Moving Average Prediction Model: A technique to get an overall idea of the trends in a data set.
Negative sentiment: A negative attitude or opinion one expressed within a given post towards a specific subject.
Neural network: A computer system modeled on the human brain and nervous system.
Neuroevolution: Neural network training approach meant to mimic that of evolution of a species over time.
New York Stock Exchange: An American stock exchange located at 11 Wall Street, Lower Manhattan.
Portfolio: grouping of financial assets such as stocks held by investors
Positive sentiment: A positive attitude or opinion one expressed within a given post towards a specific subject.
Rally: A period of gain in price for a stock or index.
Rate of Change: The current prices ratios the prices that are n days away, with n ≈ 5.
Relative Strength Index (RSI): It is intended to chart the current and historical strength or weakness of a stock or market based on the closing prices of a recent trading period.
Risk Analysis: The Process of gauging the certainty/uncertainty associated with future events and factoring this information into predictions.
Sentiment analysis: Measurement of language statements with the intention of categorizing them as positive or negative.

Share: An ownership certificate for a specific company, can be interchangeable with “stock”.
Sharpe Index: A way to determine the performance of an investment by adjusting for risk; it decides the volatility of a stock based on its expected value and variance.
SQL: An abbreviation for structured query language; A standardized query language for requesting information from a database.
Stochastic Oscillator: It ratios the difference between the current and past lowest price by the difference between the past high and low price.
Stock: An ownership certificate for no company in particular.
Stock index: A weighted average of several stock prices used to track their overall performance.
Stock market: A collection of markets and exchanges where the regular activities of buying, selling and issuance of shares of publicly held companies take place.
Stock market prediction: An estimation of a future price of a particular stock or stock index.
Ticker symbol: An abbreviation used to uniquely identify shares of a particular stock on a particular stock market.
Trading day: The time between when a stock exchange opens and the next time it closes.
Quote: The most recent price at which a specific stock was agreed to be bought and sold.
User interface: The means by which a user interacts with a computer system.
Volatility: Measurement of how spread the prices are for a stock within a fixed time. The larger range over which a stock price varies in a shorter amount of time the more volatile a stock is.
Web Scraper: A tool used to collect information from across the Internet and accrue it in a local database.
Yield: A ratio of a stock’s yearly dividends divided by its price.

3. System Requirements

Functional Requirements

REQ-#	PW	User Stories
REQ-1	4	As a user, I will be able to look at accurate predictions from various statistical models to make decisions.
REQ-2	3	As a user, I will be able to see the best AI-weighted predictions with real stock examples to make decisions.
REQ-3	5	As a user, I can have up-to-date stock data available to me in a database to make more informed decisions.
REQ-4	5	As a user, I can see stock information displayed.
REQ-5	1	As a user, I will be able to see the recommendation of how much money/shares to spend.
REQ-6	2	As a user, I can use multiple stock indexes in the forecaster.
REQ-7	3	As a user, I can see the stock information and predictions on the webpage.
REQ-8	5	As a user, I will be able to decide whether to buy, sell, or hold a stock based on recent news about that stock.
REQ-9	5	As a user, I will be able to search for predictions about any stock that is being traded in the New York Stock Exchange.
REQ-10	5	As a user, I will be able to decide why I should buy, sell, or hold a stock based on the news headlines about that stock.
REQ-11	5	As a user, I will be able to search for a particular company and view its current stock value.
REQ-12	5	As a user, I will be able to view the auto-trader's recommendations.
REQ-13	4	As a user, I will be able to view a confidence value for each recommendation.

REQ-14	3	As a user, I will be able to view relevant news about a company.
REQ-15	5	As a user, I will be able to view a company's past stock history and performance.
REQ-16	5	As a user, I will be able to create a portfolio of all the stock tickers I am interested in as favorites.
REQ-17	5	As a user, I will be able to deposit money for the auto-trader to use.
REQ-18	3	As a user, I will be able to logout
REQ-19	5	As a user… I will be able to set up a new account
REQ-20	5	As a user, I will be able to withdraw money from my account.
REQ-21	3	As a user, I will be able to tell the auto-trader how many days to run for.
REQ-22	2	As a user, I will be able to view alerts when the auto-trader makes a decision.
REQ-23	4	As a user, I will be able to manually buy or sell, in addition to the auto-trader's decisions.

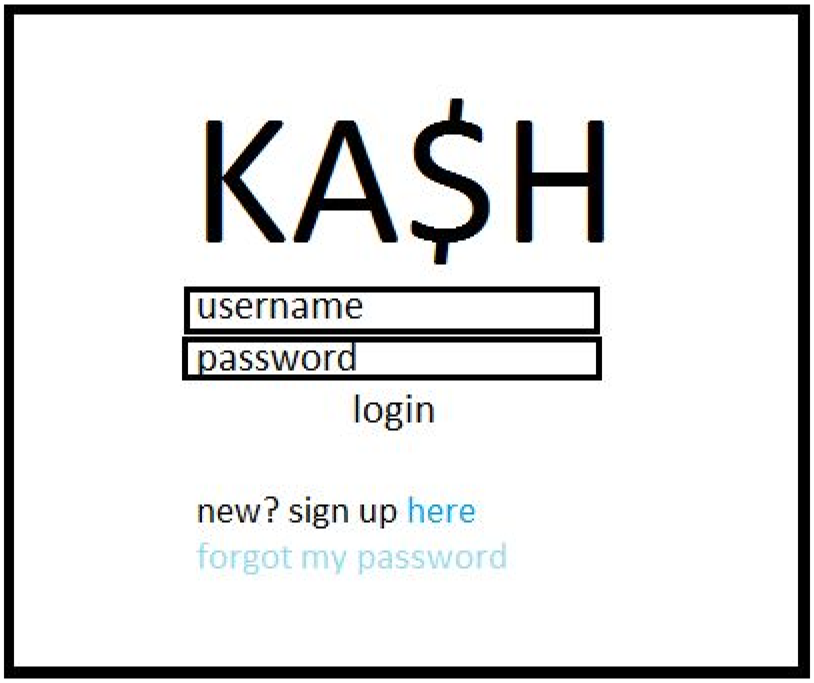
Non-Functional Requirements

REQ-#	PW	User Stories
REQ-24	3	As a user, I will be able to verify my login into my stock account.
REQ-25	3	As a user, I will be able to create new accounts.
REQ-26	1	As a user, I will be able to see relevant headlines in the news.
REQ-27	1	As a user, I will be able to see my past favorite stocks.
REQ-28	4	As a user, I will be able to view the history of my transactions.
REQ-29	3	As a user, I can view daily updated changes in stock value.

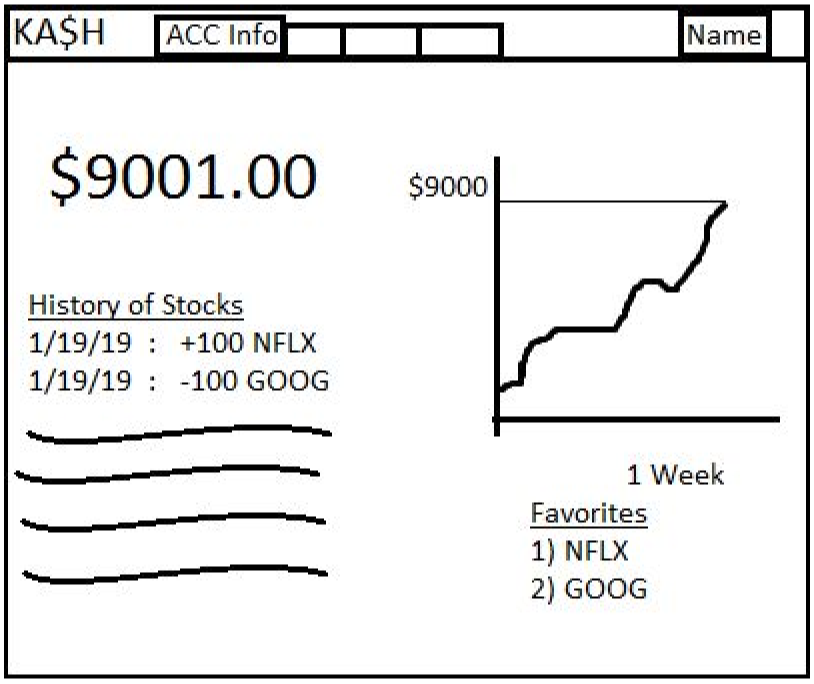
On-Screen Appearance Requirements

REQ-#	PW	User Stories
REQ-30	1	As a user, I will be able to see the current stock prices of stocks of my choice.
REQ-31	1	As a user, I will be able to see the prediction for the future stock price.
REQ-32	1	As a user, I will be able to see recommendations of whether to buy, sell, or hold on the webpage.
REQ-33	2	As a user, I will be able to manage and see my account information.
REQ-34	3	As a user, I will be able to select different services with tabs.
REQ-35	2	As a user, I will be able to see graphs with the previous stock prices and the predicted stock prices.

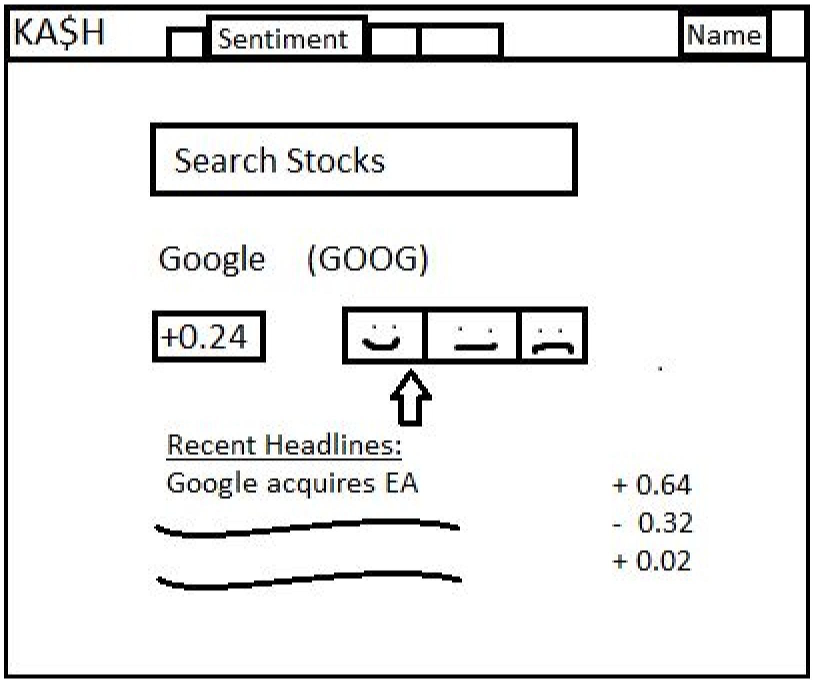
Login Screen



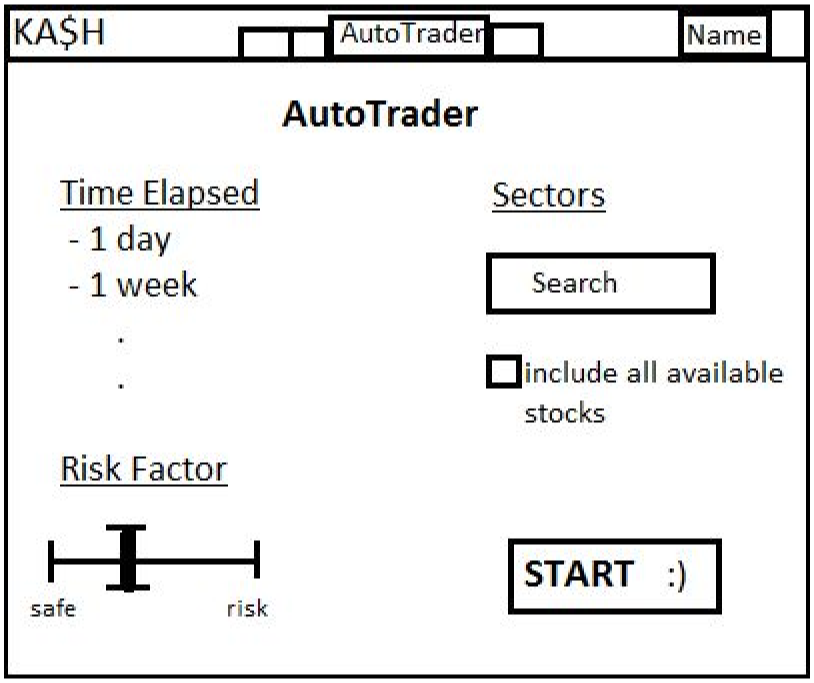
Account Info Tab



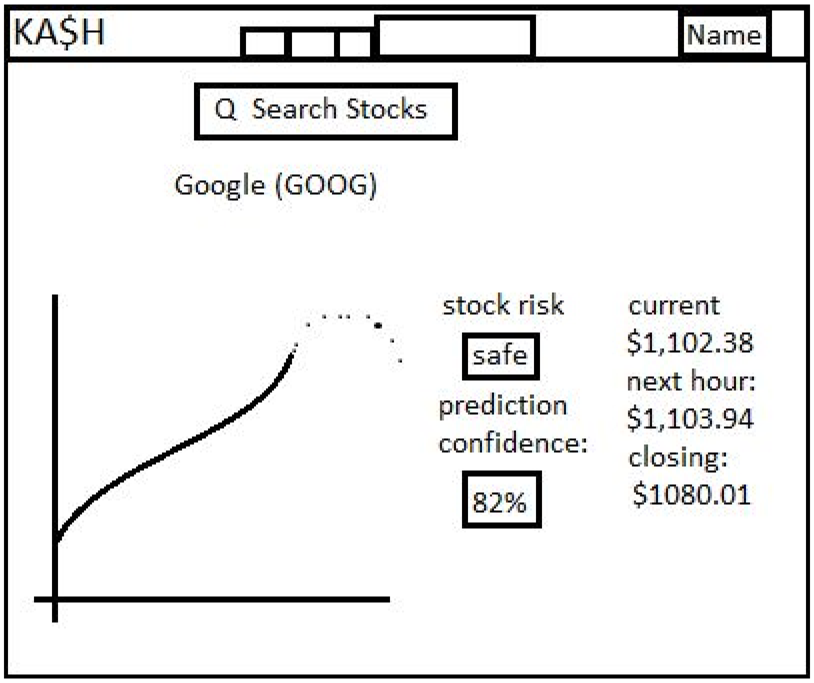
Sentiment Analysis Tab



Automated Trader Tab



Stock Forecaster Tab



4. Functional Requirements Specification

Stakeholders

Our primary stakeholders can be subdivided into two broad categories:

1. Investors, which can be further split into two subcategories
    a. Investors who wish to have the auto-trader do all of their trading for them. These investors must create an account and complete the appropriate settings and then deposit a certain amount of money to use for investments as the trader sees fit. The trader itself will make decisions based on sentiment and statistical analysis.
    b. Investors who want to make all trading decisions themselves but seek guidance. These investors do not have to create an account and simply search a stock to see the system's predictions for whether the stock price is expected to increase or decrease.
2. Economists/Researchers
    a. These stakeholders will use the system similar to investors who want to make trading decisions themselves. However, they do not actually invest in the stocks themselves and are simply looking at indicators for general economic health. These stakeholders can benefit from the graphs of stock prices, news headlines gathered, and sentiment analysis results displayed on each page of the system.

Actors and their Goals

1. Visitors - Visitors are users who have not created accounts with the system. Therefore, they cannot deposit any money or interact with the Auto-Trader in any way. Visitors simply wish to get information about various stocks in the S&P 500 and view the predictions made by the system based on news sentiment and/or old stock prices.
2. Registered Users - Registered users are users who have created accounts and can log in. They can do everything that visitors can and can also add/withdraw money from their account for the Auto-Trader to use for buying new stocks. This gives them increased flexibility when having the Auto-Trader behave as a proxy trader for them. The goal of Registered Users is to login to their account and search stocks, consume data and predictions about specific stocks, and tailor the Auto-Trader to their preferences before having it trade stocks for them.

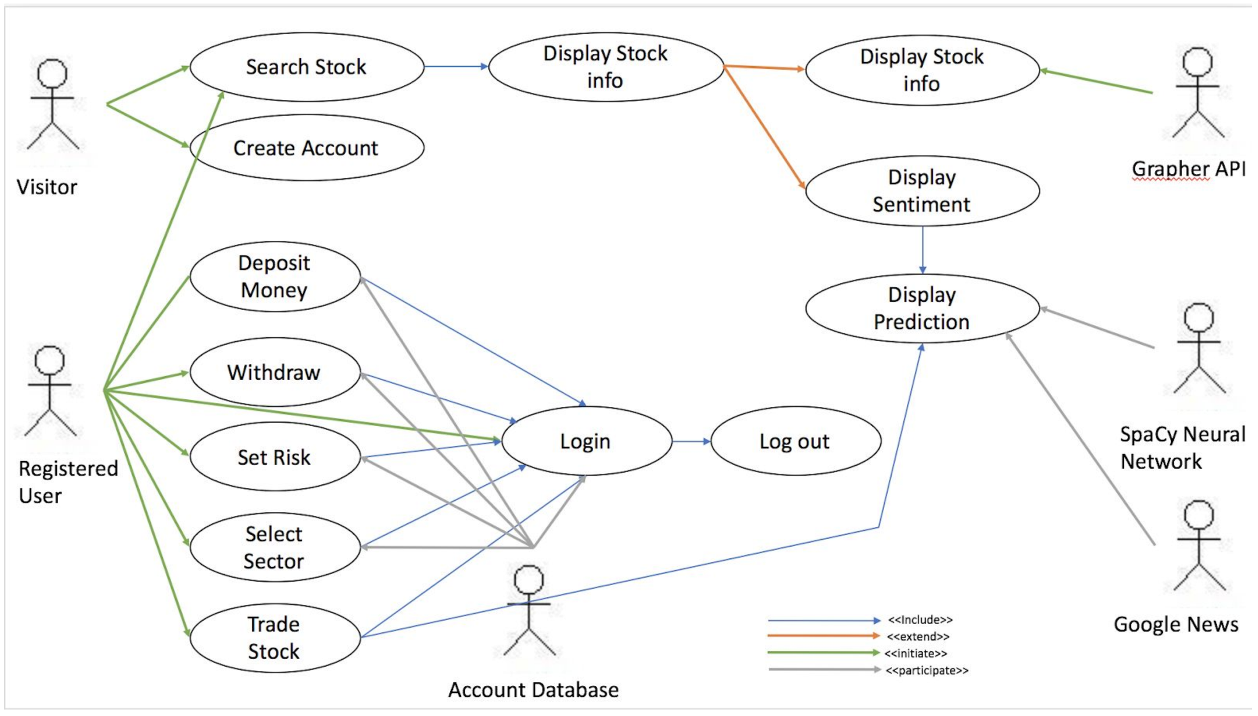
3. Database - The database is external to the system because it interacts with external APIs and participates in several use cases, ranging from Login to CreateAccount to storing data received from the external stocks and news APIs. The goal of the database is to cache any incoming data coming from the stock API for a certain period of time and more importantly, to store account information for all registered users. It will use this data to participate in any use cases that modify or manage accounts.
4. Grapher API - The goal of the Grapher API is simply to participate in the displayGraph use case and convert the stock price data points generated from Python packages into a graph that can be displayed by the system.

Casual Description of Use Cases

UC-Name	UC-#	Casual Description
Login	UC-1	To allow a REGISTERED USER to access a pre-existing account by entering their appropriate username and password. Related requirements: REQ-18,19
CreateAccount	UC-2	To allow a VISITOR to create a new account with a unique username and password that meets strength requirements (mix of upper and lowercase, numbers and alphabetic characters). REQ 22
AutoTrade	UC-3	To have the AutoTrader buy and sell stocks for a REGISTERED USER from the funds the REGISTERED USER has allocated to their account. The AutoTrader will make informed decisions to trade stock based on data from the sentiment and statistical future stock price predictions. REQ 2,3,6,11,17,21,23
DisplayStockInfo	UC-4	To see a stock's price per share plotted by a GRAPHER API and news articles pertaining to the stock gathered from the GOOGLE NEWS API. REQ 1,3,4,7,11,14
DisplayPrediction	UC-5	To view the predicted stock price patterns based on statistical analysis and AI-weighted sentiment analysis to make my OWN decisions whether to buy, sell, or hold my stocks. REQ 1,3,5,6,7,8,10,12,13,14
DisplayGraph	UC-6	To view a graph of the stock's price against time based on data

		points gathered from an external API. The creation of the graph is initiated by an external GRAPHER API. REQ 4,15
DisplaySentiment	UC-7	To determine the general sentiment towards a particular company/stock on a continuous scale from -1 to 1 (with -1 indicating most negative sentiment, 1 indicating most positive sentiment, and 0 indicating neutrality). REQ 1,10,14
DepositMoney	UC-8	To deposit money to the AutoTrader for it to use to invest in stocks based on personal preferences, statistical analysis, and sentiment analysis. Can only be done by REGISTERED USERS (Login is an included use case for this). REQ 20,22
WithdrawMoney	UC-9	To withdraw/limit the amount of money the AutoTrader has for it to use to invest in stocks based on personal preferences, statistical analysis, and sentiment analysis. Can only be done by REGISTERED USERS (Login is an included use case for this). REQ 17,19
SearchStock	UC-10	To look up any stock in the S&P at the NYSE and view its stock prices, ticker, recent news headlines, and statistical and/or sentiment analysis based on recent data about the stock. REQ 3,4,6,7,9
LogOut	UC-11	To log out of a previously logged in account, thus saving any preference changes made and locking it from access without performing login once more. Can only be done by REGISTERED USERS. REQ 18,19
ViewReport	UC-12	When the auto trading ends, the user gets a report with the start balance, end balance and the transactions made throughout. REQ 17,21,22

Use Case Diagram



Traceability Matrix

REQ-#	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12
REQ-1	4				X	X		X					
REQ-2	2			X		X							
REQ-3	5			X	X	X					X		
REQ-4	5				X		X				X		
REQ-5	1					X							
REQ-6	2			X							X		
REQ-7	3				X	X					X		
REQ-8	5					X							
REQ-9	5					X					X		
REQ-10	5					X		X					
REQ-11	5			X	X								

REQ-12	5												
REQ-13	4												
REQ-14	3				X			X					
REQ-15	5						X						
REQ-16	5												
REQ-17	5			X					X				X
REQ-18	3	X										X	
REQ-19	5	X	X						X	X		X	
REQ-20	5									X			
REQ-21	3			X									X
REQ-22	2												X
REQ-23	4			X									

Fully Dressed Descriptions for Main Use Cases

<u>**Use Case UC-3: Auto-Trade**</u>
Related Requirements: REQ 2,3,6,11,17,21,23
Initiating Actor: Registered User
Actor's Goal: To have the Auto Trader buy, sell or hold stocks from the funds the user allocated to their account. The Automatic Trader will make decisions to trade stocks based on data from the sentiment and statistical future price predictions of a certain stock.
Participating Actors: DisplaySentiment, DisplayPrediction, ViewReport
Preconditions:

- The user balance is not empty, the user enables the autotrader, the stock selected is valid input.

Postconditions:

- The user receives a transaction report of all transactions made by the auto trader, starting balance and end balance.

Success End Condition:

A. → The user selects sector of interest, inputs risk level and duration of trading.

B. ← The systems start auto trading. Include::DisplayPrediction (UC-5), DisplaySentiment (UC-7).

C. ← When done, the system sends the transactions report (UC-14).

Failed End Condition:

A. → The user starts with an empty balance.
B. ← Autotrader won't proceed with transactions and terminates early.

A. → User input invalid or "not found" sector from the list of available sectors.
B. ← The system displays "Please enter a valid input", "Sector not found".

--

<u>**Use Case UC-5: DisplayPredictions**</u>
Related Requirements: REQ 1,3,5,6,7,8,10,12,13,14
Initiating Actor: Any of: Registered User, Visitor
Actor's Goal: Receive AI prediction as an additional factor, removed from the automated trader, to making stock decisions.
Participating Actors: Some data source (database)
Preconditions:

- User needs to pick a valid stock that is present in the database

Success End Condition:

- Data is successfully retrieved from database for statistical analysis and from the web scraper for sentiment analysis; prediction displayed, and a recommendation is displayed accordingly

Failed End Condition:

- User does not enter a valid stock; stock entered not present in the database; database/API was unable to respond.

Flow of Events for Main Success Scenario:

A. → The user enters a stock name that's available in the database.
B. ← The system starts searching and returns predicted stock trends along with AI-generated predictions

Flow of Events for Extensions (Alternate Scenarios):

A. → The user enters a stock name that is unavailable in the database.
B. ← The system displays "Please enter a valid input", "Stock not found".

<u>**Use Case UC-10: SearchStock**</u>

Related Requirements: REQ 3,4,6,7,9
Initiating Actor: Registered User, Visitor
Actor's Goal: To look up any stock in the S&P at the NYSE and view its stock prices, ticker, recent news headlines, and statistical and/or sentiment analysis based on recent data about the stock.
Participating Actors: DisplayStockInfo, DisplayGraph,DisplaySentiment
Preconditions:

- User searches for a valid/existing stock

Postconditions:

- The user is able to see the stock portfolio which includes recent news headlines, sentiment score, and a graph showcasing the current stock price movements.

Success End Condition:

A. → The user types correct stock name
B. ← The system starts searching and returns the stock portfolio
  - Include::DisplayStockInfo, DisplayGraph,DisplaySentiment

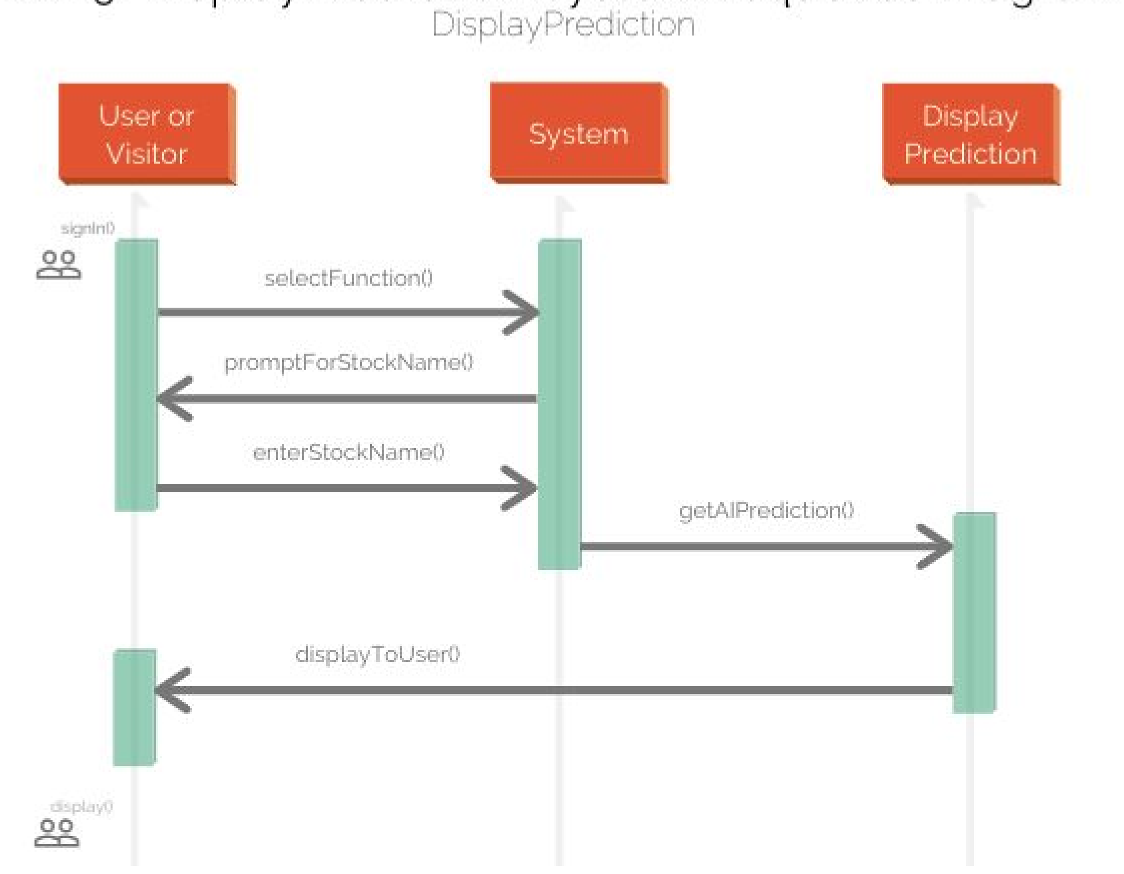
Failed End Condition:

A. → The user enters an invalid/incorrect stock name
B. ← The system displays "Please enter a valid input", "Stock not found".

System Sequence Diagrams

Use Case: DisplayPrediction

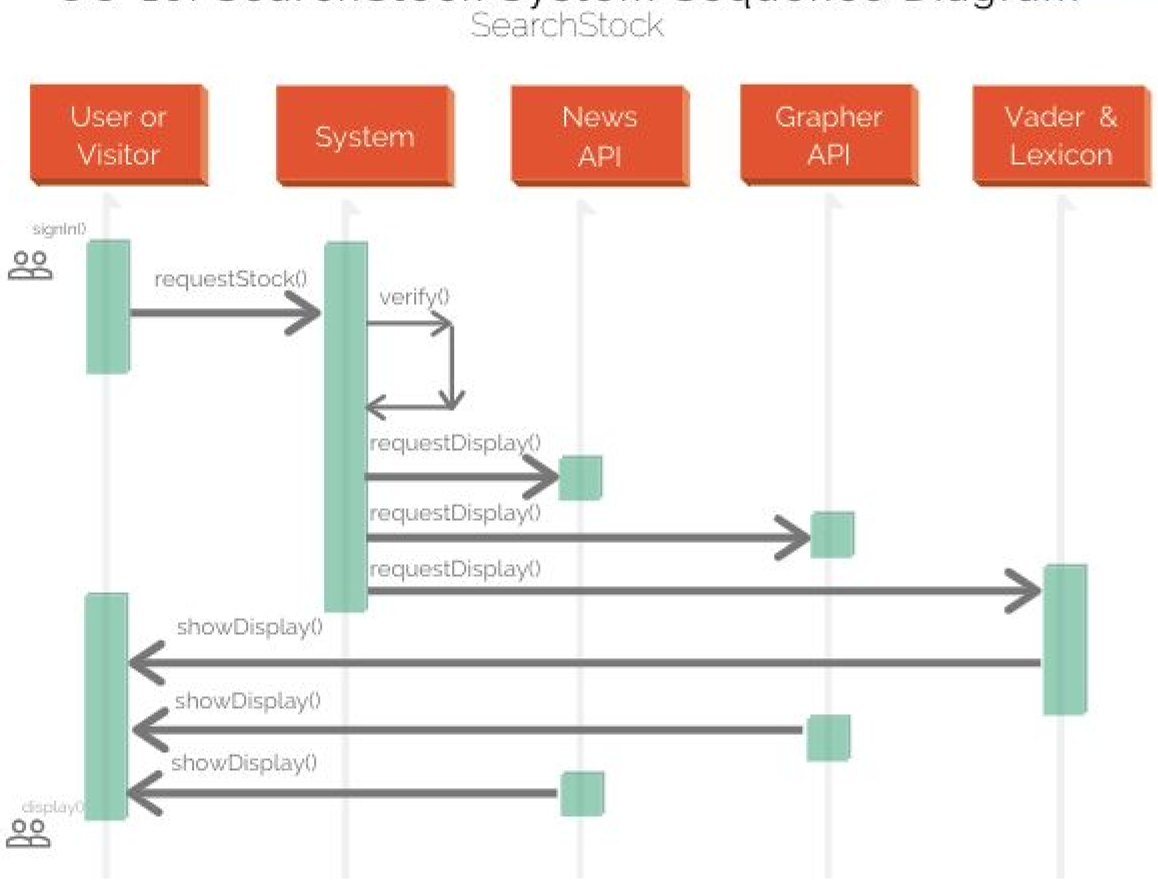
UC-5: DisplayPrediction System Sequence Diagram



Use Case: Search Stock

UC-10: SearchStock System Sequence Diagr

Copy page

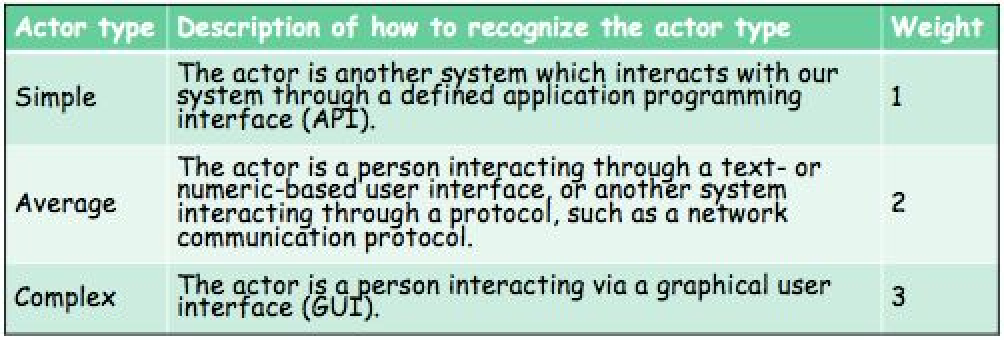


5. Effort Estimation using Use Case Points

Domain Model

1. Unadjusted Actor Weight (UAW)

Calculating basing off of:



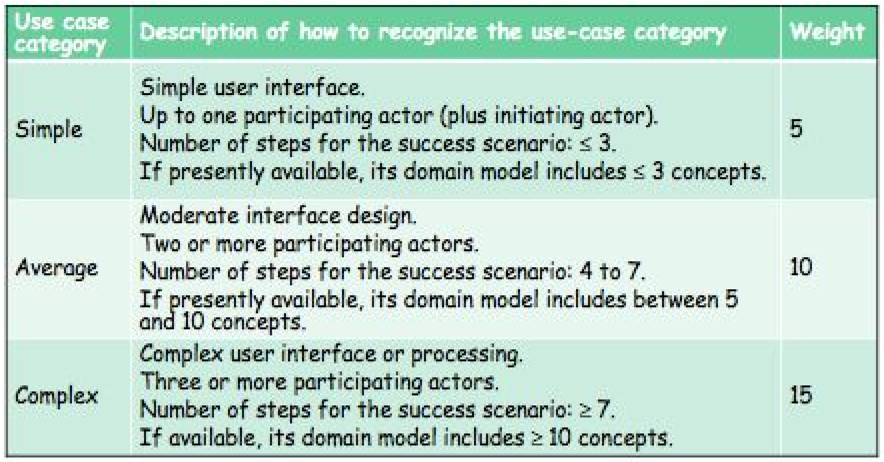
Actor type	Description of how to recognize the actor type	Weight
Simple	The actor is another system which interacts with our system through a defined application programming interface (API).	1
Average	The actor is a person interacting through a text- or numeric-based user interface, or another system interacting through a protocol, such as a network communication protocol.	2
Complex	The actor is a person interacting via a graphical user interface (GUI).	3

Actor	**Complexity**	**Weight**
Visitor	Complex	3
Registered User	Complex	3
GrapherAPI	Simple	1
Database	Average	2
Vader	Simple	1
Alphavantage API (getting stock prices)	Simple	1
Google News API	Simple	1

Unadjusted Actor Weight (UAW) = 4*Simple + 1*Average + 2*Complex = **12**

2. Unadjusted Use Case Weights (UUCW)

Calculated base off of:



Use case category	Description of how to recognize the use-case category	Weight
Simple	Simple user interface. Up to one participating actor (plus initiating actor). Number of steps for the success scenario: ≤ 3. If presently available, its domain model includes ≤ 3 concepts.	5
Average	Moderate interface design. Two or more participating actors. Number of steps for the success scenario: 4 to 7. If presently available, its domain model includes between 5 and 10 concepts.	10
Complex	Complex user interface or processing. Three or more participating actors. Number of steps for the success scenario: ≥ 7. If available, its domain model includes ≥ 10 concepts.	15

Use Case	UC-#	Description	Complexity	Weigh t
Login	UC-1	Simple user interface, 2+2 = 4 steps for main success, 3 participating actors (visitor, user, database)	Average	10
CreateAccount	UC-2	Moderate user interface, 2+7 = 9 steps for main success, 2 participating actors (vistor, database)	Average	10
AutoTrade	UC-3	Complex user interface, over 7 steps: selecting all specifications (time limits, risk, money, etc), 3 participating actors (user, database, neural network)	Complex	15
DisplayStockInf o	UC-4	Moderate user interface, 3 steps (search stock, select stock, explore graph), 3 participating actors (database, grapherAPI, user/visitor)	Average	10
DisplayPredicti on	UC-5	Simple user interface, 3 steps (search stock, select stock, display), 3 participating actors (user, neural network, database)	Simple	5
DisplayGraph	UC-6	Moderate user interface, 3 steps (search stock, select stock,	Average	10

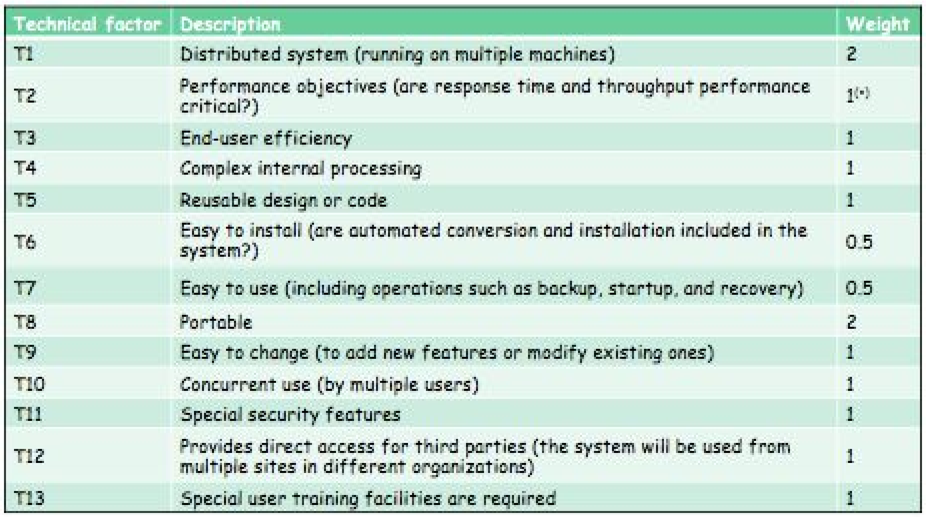
		explore graph), 3 participating actors (database, grapherAPI, user/visitor)		
DisplaySentiment	UC-7	Moderate user interface, 3 steps (select stock, display sentiment, display headlines), 3 participating actors (database, neural network, user/visitor)	Average	10
DepositMoney	UC-8	Simple user interface, 1 step (select button), 2 participating actors (user, database)	Simple	5
WithdrawMoney	UC-9	Simple user interface, 1 step (select button), 2 participating actors (user, database)	Simple	5
SearchStock	UC-10	Simple user interface, 2 steps (type stock and press enter), 2 participating actors (user, database)	Simple	5
LogOut	UC-11	Simple user interface, 1 step (log out), 2 participating actors (user, database)	Simple	5
ViewReport	UC-12	Complex user interface, 5 steps (view transactions, time, money, etc), 2 participating actors (user, database)	Complex	15

Unadjusted Use Case Weights (UUCW) = sum of weights = **105**

3. Technical Complexity Factors (TCFs)

From the wikipedia page: “TCF is determined by assigning a score between 0 (factor is irrelevant) and 5 (factor is essential) to each of the 13 technical factors listed in the table below. This score is then multiplied by the defined weighted value for each factor. The total of all calculated values is the technical factor (TF).”

Formula: TCF = 0.6 + (TF/100)



Technical factor	Description	Weight
T1	Distributed system (running on multiple machines)	2
T2	Performance objectives (are response time and throughput performance critical?)	1(*)
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install (are automated conversion and installation included in the system?)	0.5
T7	Easy to use (including operations such as backup, startup, and recovery)	0.5
T8	Portable	2
T9	Easy to change (to add new features or modify existing ones)	1
T10	Concurrent use (by multiple users)	1
T11	Special security features	1
T12	Provides direct access for third parties (the system will be used from multiple sites in different organizations)	1
T13	Special user training facilities are required	1

Technical Factor	**Weight**	**Assigned Value**	**Weight x Value**
T1	2	2	4
T2	1	2	2
T3	1	2	2
T4	1	2	2
T5	1	2	2
T6	0.5	2	1
T7	0.5	2	1
T8	2	2	4
T9	1	2	2
T10	1	2	2
T11	1	2	2
T12	1	2	2
T13	1	1	1
		TF	**27**

TCF = 0.6 + TF/100 = 0.6+27/100 = **0.87**

ECF = **1** (given)

4. Summary Table and Use Case Points

Type	**Weight**
UAW	12
UUCW	105
TCF	0.87
ECF	1

Use Case Points (UCP) = (UUCW + UAW) x TCF x ECF
= (105 + 12) x 0.87 x 1 = **101.79**

Project Size Estimation/Use Case Points = **101.79**

6. Domain Analysis

1. Concept Definitions

Responsibility	Type	Concept
Coordinating actions of concepts	D	Controller
Authenticating user login information	D	Authenticator
Storing account information	K	AccountTracker
Querying and updating database	D	DatabaseConnection
Producing graph of stock prices on website	D	GrapherConnection
Connect with Google API to get headlines	K	GoogleAPIConnection
Receive stock information	K	AlphavantageConnector
Displaying current state on website	D	Interface Page

Retrieving news headlines from Google News	K	Webcrawler
Using ARIMA to predict stock prices	D	ARIMAModel
Using Fast Fourier Transform to predict stock prices	D	FourierModel
Using Stochastic Oscillator to predict stock prices	D	StochasticOscillatorModel
Using Accumulative Swing Index to predict stock prices	D	AccumulativeSwingIndexModel
Using Rate of Change to predict stock prices	D	RateOfChangeModel
Use Fast Fourier Transform to predict prices	D	FourierModel
Averages the models to provide an accurate description	D	ModelAverager
Calculating sentiment from headline data	D	SentimentPredictor
Recommends transaction decisions (hold, buy, sell) based off the predictions information. Trader: trades automatically for the user	D	StockRecommender
Update the user's stock portfolio after each transaction	D	PortfolioUpdater
Sets the trading duration for the AutoTrader	D	TradingTimer
Reports what transactions have been made during the auto trading period.	D	TransactionReporter
Gets prediction data and sentiment to be incorporated in the auto trader decision-making process through feeding this data to the neural network.	K	ObtainPrediction (sentiment pred+Averager)
trades and makes actual transactions for the user and proceeds with the recommendations it makes, auto-pilot.	D	Autotrader

2. Association Definitions

Concept Pair	Association Description	Association Name
DatabaseConnection <-> Controller	Controller queries for the database information.	Query database
Controller <-> Authenticator	Controller sends login information to authenticator and Authenticator sends back the verification.	Authenticate
AccountTracker <-> Controller	AccountTracker gets account information from Controller	Get account info
ARIMAModel <-> ModelAverager	Model sends prediction to the ModelAverager	Send prediction
FourierModel <-> ModelAverager	Model sends prediction to the ModelAverager	Send prediction
StochasticOscillatorModelo <-> ModelAverager	Model sends prediction to the ModelAverager	Send prediction
AccumulativeSwingIndexModel <-> ModelAverager	Model sends prediction to the ModelAverager	Send prediction
RateOfChangeModel <-> ModelAverager	Model sends prediction to the ModelAverager	Send prediction
Controller <-> ModelAverager	Controller received the final prediction from ModelAverager.	Receive averaged prediction
Controller <-> Database	Controller sends the predictions to Database.	Send predictions
Controller <-> GrapherConnector	Controller sends predictions to GrapherConnector.	Send predictions
Database <-> GrapherConnector	GrapherConnector gets stock prices from database.	Query database
GrapherConnector <-> Interface Page	GrapherConnector sends the graph to the Interface	Send graph

	Page.	
StockRecommender <-> ObtainPrediction	the recommender obtains the prediction data to generate recommendations	generate recommendations
TradingTimer <>Autotrader	provides trading duration/interval for the autotrader	inputs time interval

3. Attribute Definitions

Concept	Attribute	Attribute Definition
Webcrawler	Stock Ticker	A ticker associated with a particular stock that the Webcrawler takes as input to search the Internet for news headlines.
	News API	The framework that the Webcrawler uses to access news sites and check for results connected to the search term.
Sentiment Calculator	Headlines	A list of titles of news articles that the calculator (powered by Vader and the Loughran-McDonald Financial Lexicon) needs to perform its sentiment analysis of a particular stock.
	Dictionary	A list of financial related concepts and definitions that is incorporated into Vader to provide the basis of the sentiment calculator's dataset. The calculator performs all sentiment calculations by

		looking for terms in the input headlines that appear in the dictionary.
Model Averager	Model predictions	Predictions from the four models that will be averaged for a more accurate prediction
	Past values	Past values of the stocks that are used in the models to make predictions
Controller	User’s identity	Used to determine credentials for the user’s account
	Account history	Past transactions and activity of the user
	Predictions	Predictions from the ModelAverager for stocks
Auto Trader	Stock recommender	recommends transactions based off sentiment and stock forecasting
	Predictions	used to generate recommendations
	sentiment	used to make generate recommendations
	risk level indicator	used to set risk level
	Trading timer	used to set trading interval
	sector selection	used to determine sector

4. Traceability Matrix

Use Case-#	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12
PW	8	5	32	25	33	10	14	10	10	20	8	10
Controller	X	X									X	
Authenticator	X	X										

AccountTracker	X	X						X	X		X	
DatabaseConnection	X	X		X	X	X	X	X	X	X		
GrapherConnection				X		X						
GoogleAPIConnection				X					X			
AlphavantageConnector			X	X						X		
InterfacePage				X	X	X	X					X
Webcrawler							X					
SentimentPredictor					X		X					
ARIMAModel					X							
StochasticOscillatorModel					X							
AccumulativeSwingIndexModel					X							
RateOfChangeModel					X							
FourierModel					X							
ModelAverager					X							
StockRecommender			X		X		X			X		
PortfolioUpdater			X					X	X			
TradingTimer			X									
TransactionReporter												X
ObtainPrediction			X		X		X					
AutoTrader			X					X	X			

System Operation Contracts

1) Search Stock Info

Responsibilities: The system must be able to find information on a stock when requested to do so by other parts of the system.

Cross References: User Case 10 (SearchStock)

Expectations: The system, when requested, will be able to find the instantaneous price of a stock at a specified time (current or otherwise), the opening or closing price of a stock on a certain day, a list of stocks included in a specific sector, or all of the prices of a stock over a specific time interval indexed over a smaller interval.

Preconditions: The stock must be valid, the requested information about the stock must either already be in the database or must be findable by the stock API

Postconditions: The specified stock information is returned to whomever requested it.

2) Update Forecast
Responsibilities: Makes a new prediction for a future values of a stock based off of the new data inside of the database. After forecasting a new value, it updates the database and the forecasting user interface.

Cross references: User cases 4 (DisplayStockInfo), 5 (DisplayPrediction), and 6 (DisplayGraph)

Expectations: The database should have up to date values for the stock in question, and the user should be able to see the newly displayed forecast and stock information. The prediction should also be accurate to some degree.

Preconditions: The user must have internet access, the database must be able to link to the forecaster algorithms successfully, the database must be able to collect live data

Postconditions: The new forecasting and stock information are made available to the user on the stock forecast tab.

3) Update Sentiment

Responsibilities: The user should be able to receive a recommendation to either buy/hold/sell a stock of their choosing.

Cross References: User Cases 4 (DisplayStockInfo), 7 (DisplaySentiment)

Expectations: The user should be able to receive a weighted sentiment analysis of any stock of their choosing, so long as the stock is present in the database. Based on the stock chosen by the user, a sentiment ranging from -1 to 1 (-1 indicating the most negative sentiment, 1 indicating the most positive sentiment, and 0 indicating neutrality) will be determined from the headlines parsed from financial websites. The sentiment will then not only correlate to a recommendation displayed to the user to either buy/hold/sell the particular stock in question but it will also display the headlines that have been parsed for sentiment analysis, thus validating the recommendation.

Preconditions: The user must have internet access; the stock in question must be present in the database prior to implementing the sentiment analysis; the sentiment analysis must perform with timely news headlines.

Postconditions: The new sentimental analysis, recommendation, and parsed headlines are made available to the user on the stock forecaster tab.

4) Automatic Buy/Sell

Responsibilities: The system will buy and sell stocks on behalf of the user

Cross References: UC 3 (Autotrade) and 12 (ViewReport)

Expectations: The trader proceeds with making transactions for the user and updates the balance of the user. The data is sent to the user in form of a report after trading time ends.

Preconditions: The autotrader know how long it should run before hand. The initial balance is not empty and the user is a registered not visitor only.

Postconditions: The transaction report is sent to user, user balance is updated based off the transactions made. The user can view these data on their profile.

Mathematical Model

1. Stochastic Oscillator

The stochastic oscillator is used to determine if a stock is about to change its "momentum", its downward or upward trajectory. Depending on what range the calculated index value falls in, the oscillator index can be interpreted as a signal to either buy, sell, or hold. The index is calculated by:

$$\%K = 100\frac{(CP-LN)}{(HN-LN)}$$

Where %K represents the oscillator index, a number between 0 and 100, which will indicate the likeliness of the stock to switch from increasing to decreasing or vice versa, CP represents the Current Price of the stock, LN represents the low price of the stock over the course of the last N trading periods, HN represents the high price of the stock over the course of the last N trading periods.

The oscillator index can be compared to a moving average of previous values in order to determine if the current momentum of the stock is switching from the past norm.

2. Rate of Change

The rate of change (ROC) of a stock is a simple calculation that will indicate if a stock is increasing or decreasing and also indicate how quickly it is increasing or decreasing. It can be calculated:

$$ROC = \frac{CP - CPn}{CPn}$$

Where CP is the current Closing Price of the stock and CPn is the closing price of the stock n time periods prior. The ROC can be any positive or negative number, a negative number being an indicator that the price is falling and a positive number indicating that the stock price is rising. A switch from positive to negative indicates a change from rising to falling and is an indicator to sell, and a switch from negative to positive is the opposite. It is important to use stock data from an appropriate amount of time prior, as long term trends may not be useful for short term trading and vice versa.

3. Accumulative Swing Index

Based on a stock's open, high, low and close prices, we can place up or down bars at discrete intervals daily. Using the current bar and the previous bar, we can calculate the accumulative swing index to plot a running total of the swing index value for each bar and see the long term trends the stock may undergo.

From tradingtechnologies.com, we have the formulas below:

ASIt=ASIt−1+SIt

where
SIt is the current bar's swing index calculated by the formula:

$$SI_t = 50 \times \left(\frac{Close_t - Close_{t-1} + 0.5 \times (Close_t - Open_t) + 0.25 \times (Close_{t-1} - Open_{t-1})}{R} \right) \times \frac{K}{T}$$

where

$$K = max\,(High_t - Close_{t-1}\ ,\ Close_{t-1} - Low_t)$$

T is a user defined value which means the maximum price change during a trading session.

R is a value calculated on the base of the relationship between current close price and previous high and low prices. The formula is:

$$R = TR - 0.5 \times ER + 0.25 \times SH$$

where

$$TR = max(High_t - Close_{t-1}, Close_{t-1} - Low_t, High_t - Low_t)$$

$$ER = \begin{cases} High_t - Close_{t-1} & ifClose_{t-1} > High_t \\ 0 & ifLow_t \leq Close_{t-1} \leq High_t \\ Close_{t-1} - Low_t & ifClose_{t-1} < Low_t \end{cases}$$

$$SH = Close_{t-1} - Open_{t-1}$$

4. Autoregressive Integrated Moving Average (ARIMA)

This model uses time series analysis to fit its future predictions for a value. It regresses the variable in question with some previous (lagging) and current values while calculating the regression error as a linear combination of the error terms of these past values. It also differences its observations to allow the model to fit as best as possible. This allows us to closely gauge the strength of the stock price in relation to its past and in turn to the other many changing variables it may depend upon.

From Wikipedia, how all of this is calculated:

Given a time series of data X_t where t is an integer index and the X_t are real numbers, an ARMA(p',q) model is given by

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q},$$

Or equivalently by

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

Where L is the lag operator, the α_i are the parameters of the autoregressive part of the model, the θ_i are the parameters of the moving average part and the ε_t are error terms.

An ARIMA(p,d,q) process expresses this polynomial factorisation property with p=p'−d, and is given by:

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

The above can be generalized as follows.

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d X_t = \delta + \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t.$$

This defines an ARIMA(p,d,q) process with drift $\frac{\delta}{1 - \sum \phi_i}$.

Luckily, we will not have to implement this from scratch; we will be using a python package to fit the data this way for us.

5. Fourier Extrapolation Model

This model uses extrapolation to predict a future stock price when given an array of past prices. It does this by mapping the received stock prices to a frequency domain complex valued function and then using that to calculate a predicted value. The discrete Fourier Transform is expressed as such:

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \\ &= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)], \end{aligned}$$

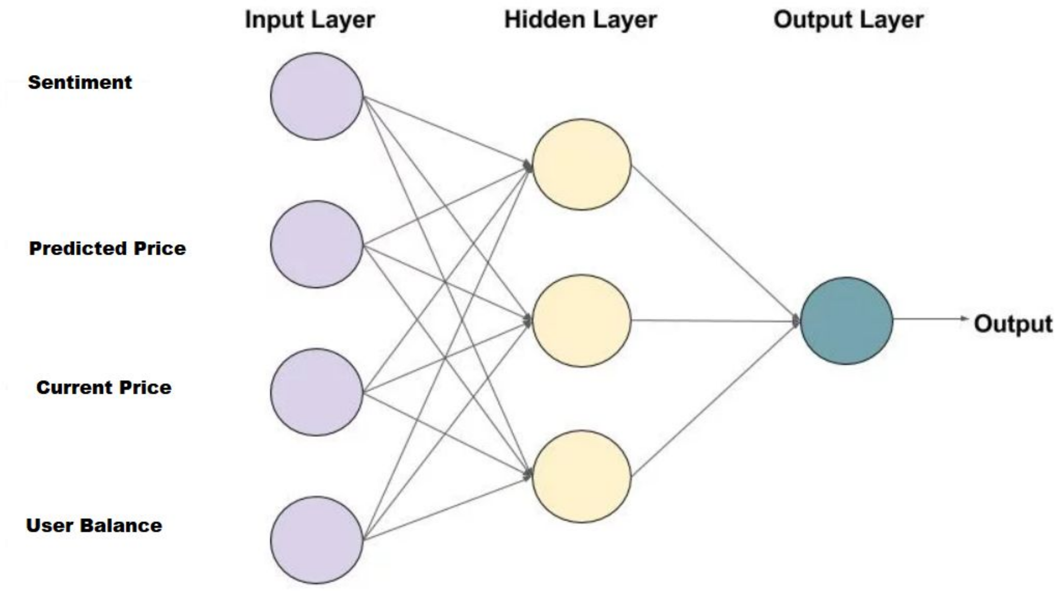
where Xk represents a sequence of points that map to the original sequence of values xn, k and n being the respective indices, and N is the number of points being mapped. The numpy library function polyfit is used to fit the complex values to a complex function in a way that minimizes the error calculation:

$$E = \sum_{j=0}^{k} |p(x_j) - y_j|^2$$

where p(xj) represents the prediction value and yj represents the actual value, k is the number of points being fitted, j is the summation index and E is the total error that is being minimized. Once the function is determined it is used to extrapolate a prediction on a future stock price.

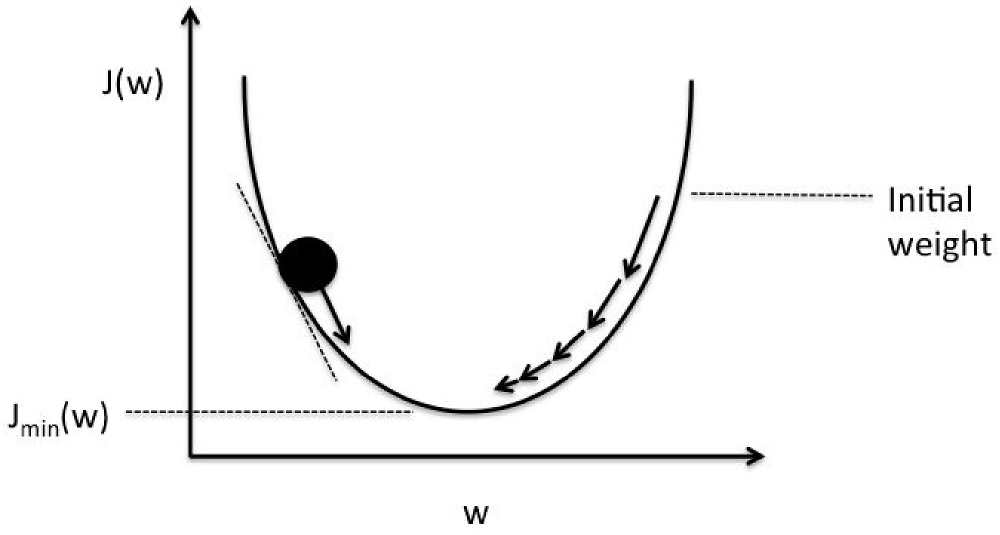
Neural Network Model

In order to allow the Auto-Trader present in our project to reach a competent level of accuracy and efficiency, a few neural network models and testing algorithms were considered for implementation, and the one that was decided was a Feed Forward Neural Network. This model was chosen over other models due to its relative lightweight operation and trainability compared to other possible choices.



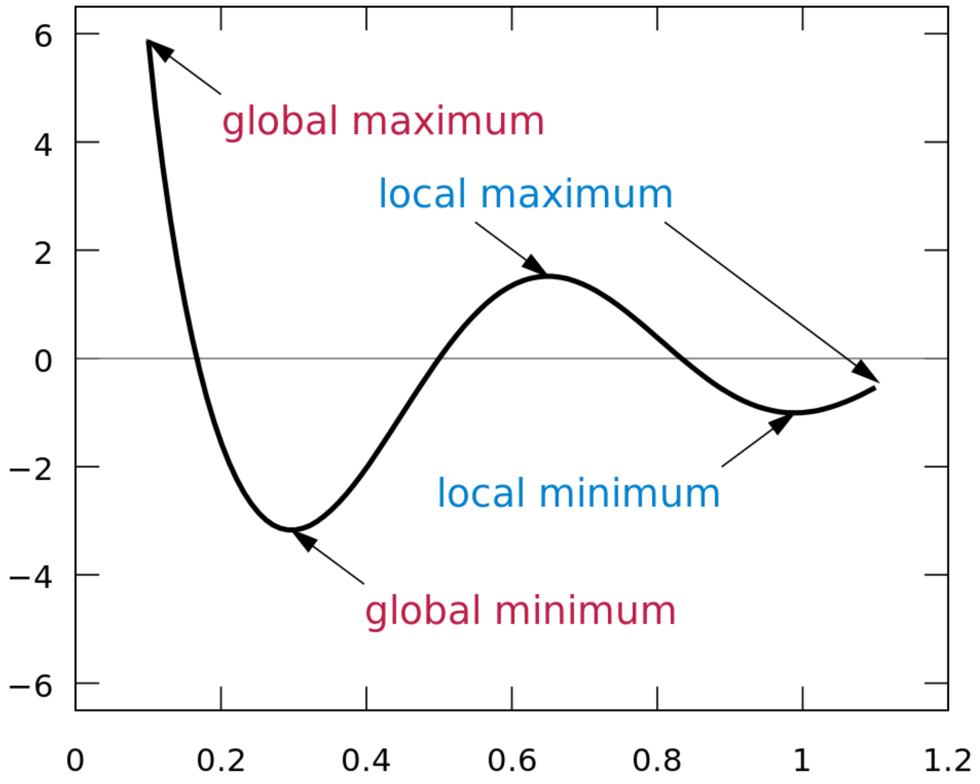
Operation Time is an important variable in this project. Stock prices are very volatile and change quickly, so fast performance is imperative while choosing neural network models. In addition, due to the large dataset that is to be analyzed when looking at stocks, sufficient training needs to be done in order to have the neural network function as intended. The time constraint that comes with the nature of a project that only lasts a semester limits the types of training methods that can be used, but the process that was decided on was using Neuroevolution with partial back propagation.

The most popular training method used in Forward Feed Neural Networks is backpropagation, a method that involves finding the error between the desired output and the calculated output, and taking the derivative to find the gradient descent. The values of the weights assigned to the hidden layers are adjusted based on if the gradient is positive or negative to find the minimum error.



Schematic of gradient descent.

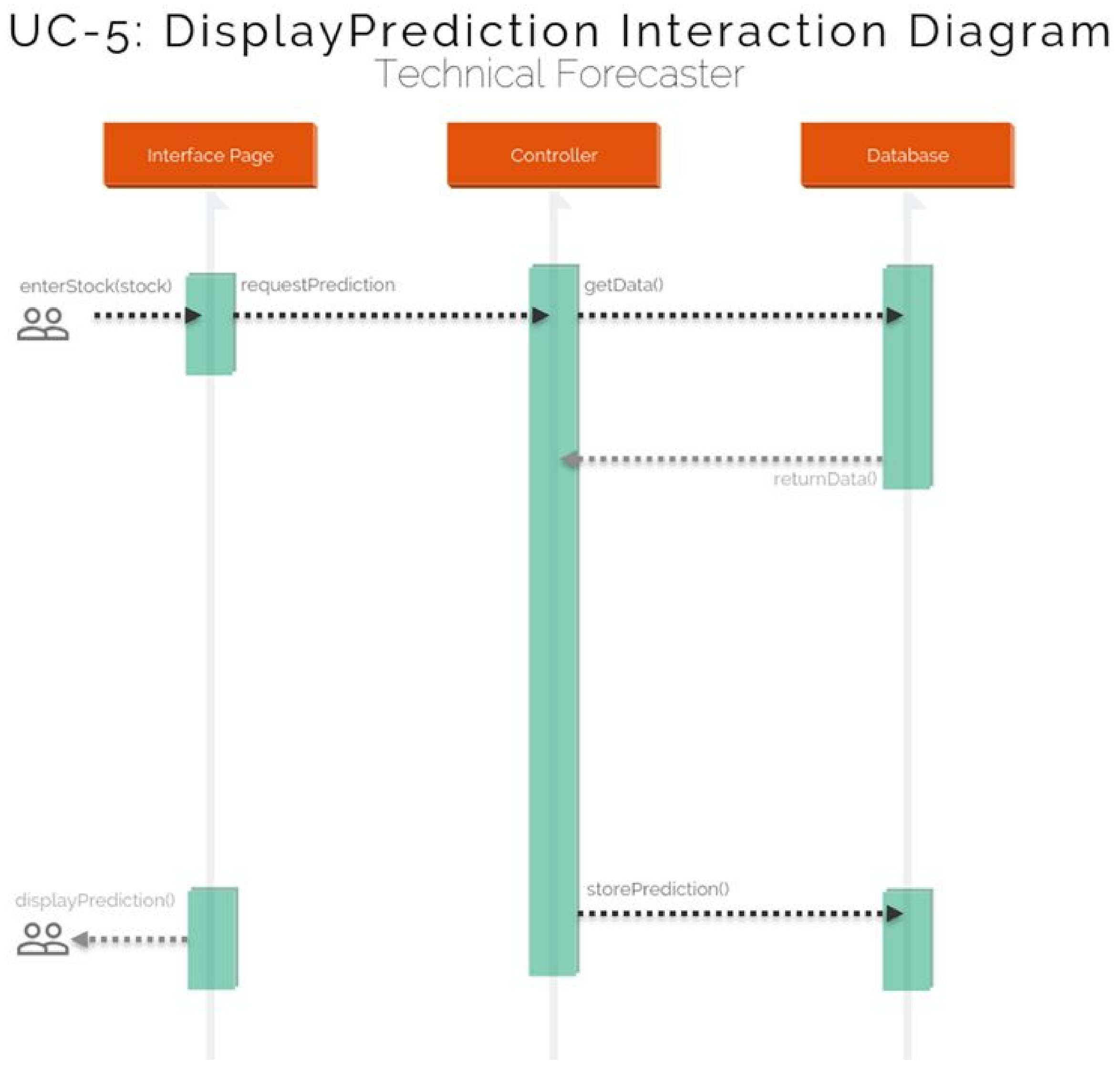
This method has downfalls, however. Backpropagation has limitations in that it is very proficient at finding a minimum, it is not necessarily the global minimum and the lowest possible error rate that can be done.



The training process of Neuroevolution starts with multiple instances of the network with random assortments of weights assigned to the hidden layers. This is known as Generation 1. The network's success is judged based on a fitness score, that is calculated based on the network's proximity to the actual price after the price interval being analyzed, along with the amount of net gain the user would get from the decision it makes. From this point, the instances with the best scores are saved and small variances are made to populate Generation 2, and then the process is repeated. To further improve performance, a partial implementation of backpropagation is used, in which one of the chosen instances are randomly picked to undergo the backpropagation process outlined above. This is a partial implementation because only one of these instances are back propagated per generation.

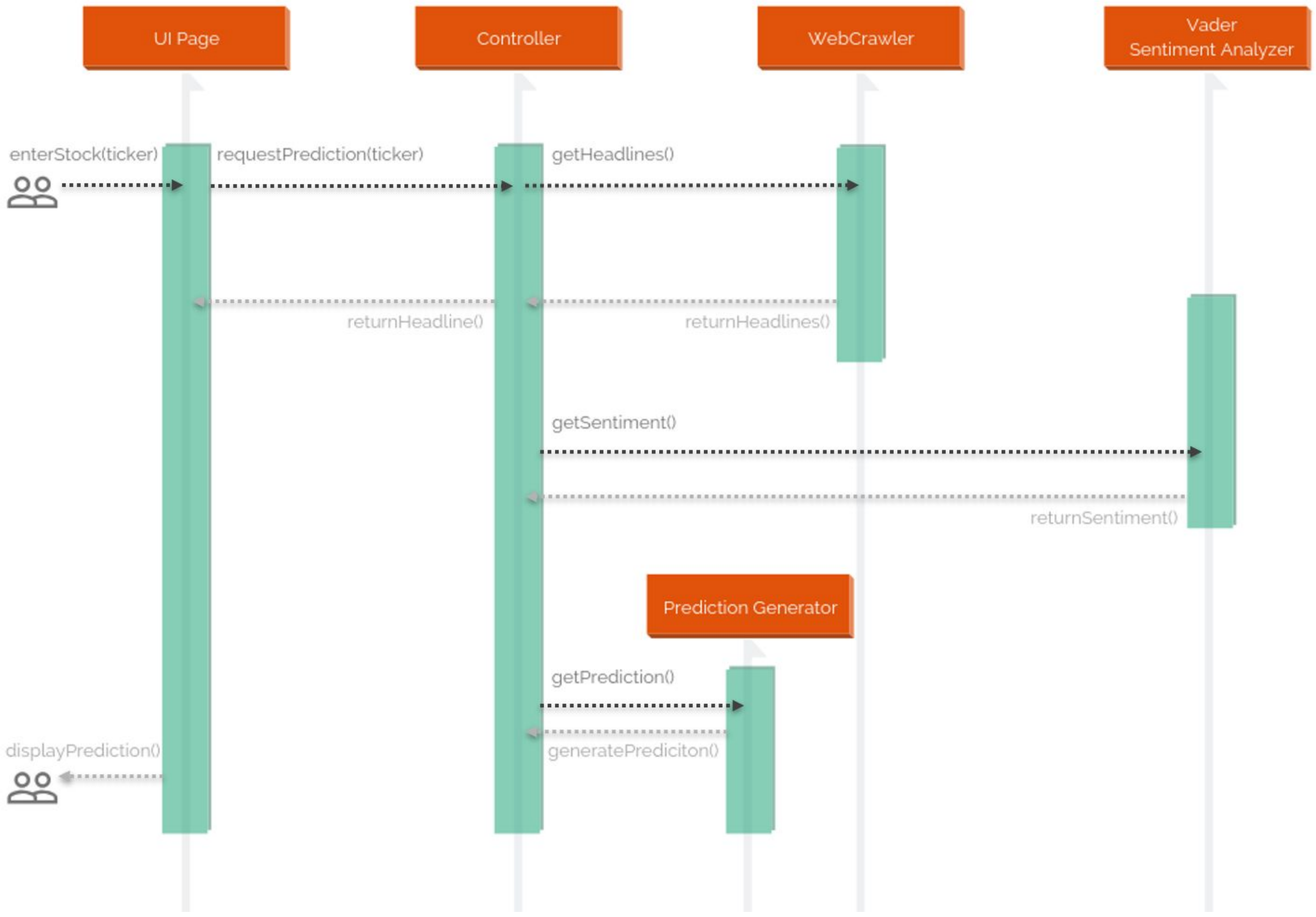
The above implementation was chosen due to the fact that neuroevolution is the most efficient way to find the version of the neural network with the lowest error percentage, with the extra bonus of using partial backpropagation to come to this conclusion faster than if it was not included.

7. Interaction Diagrams



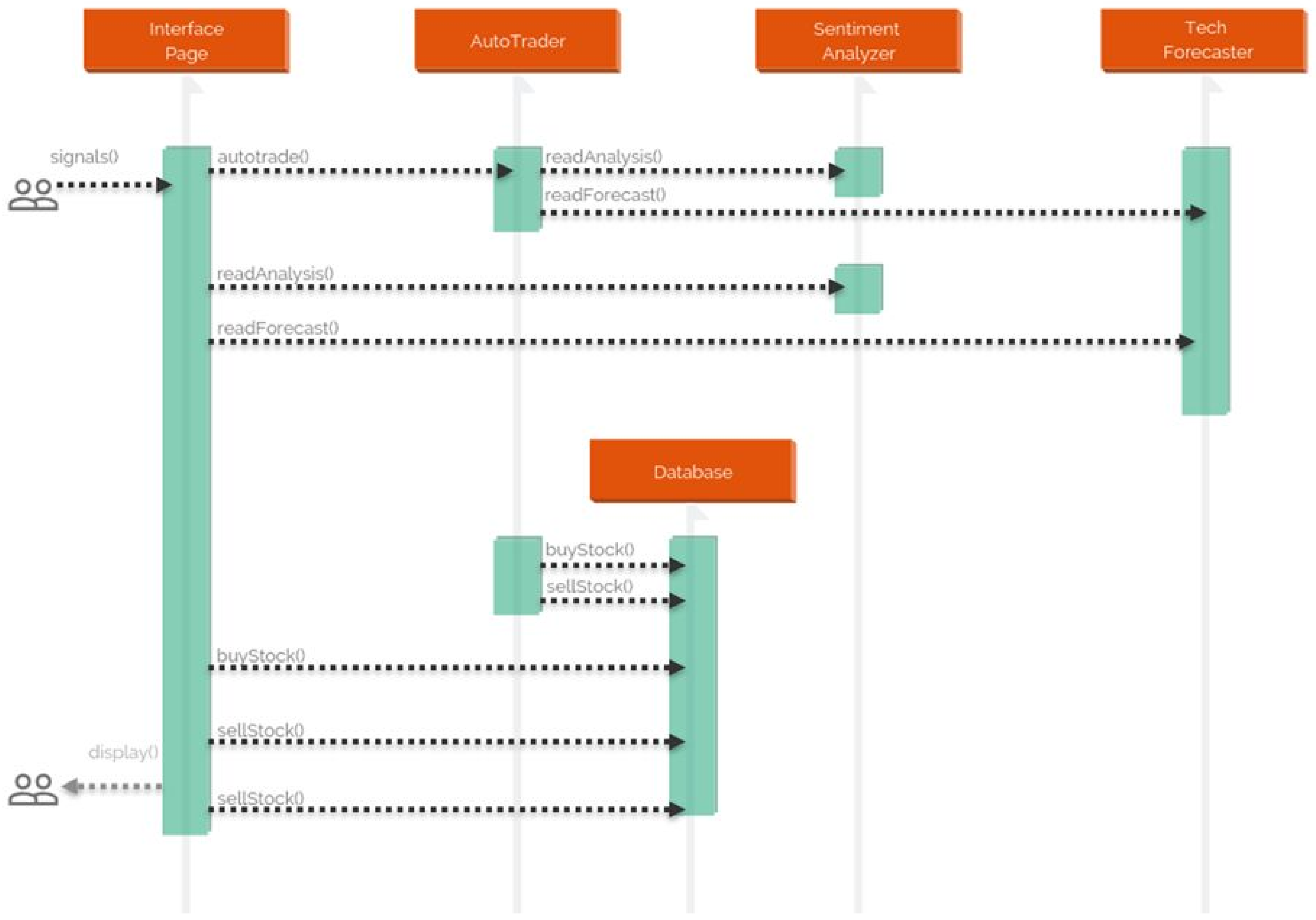
UC-5: DisplayPrediction Interaction Diagram

Sentiment Analyzer

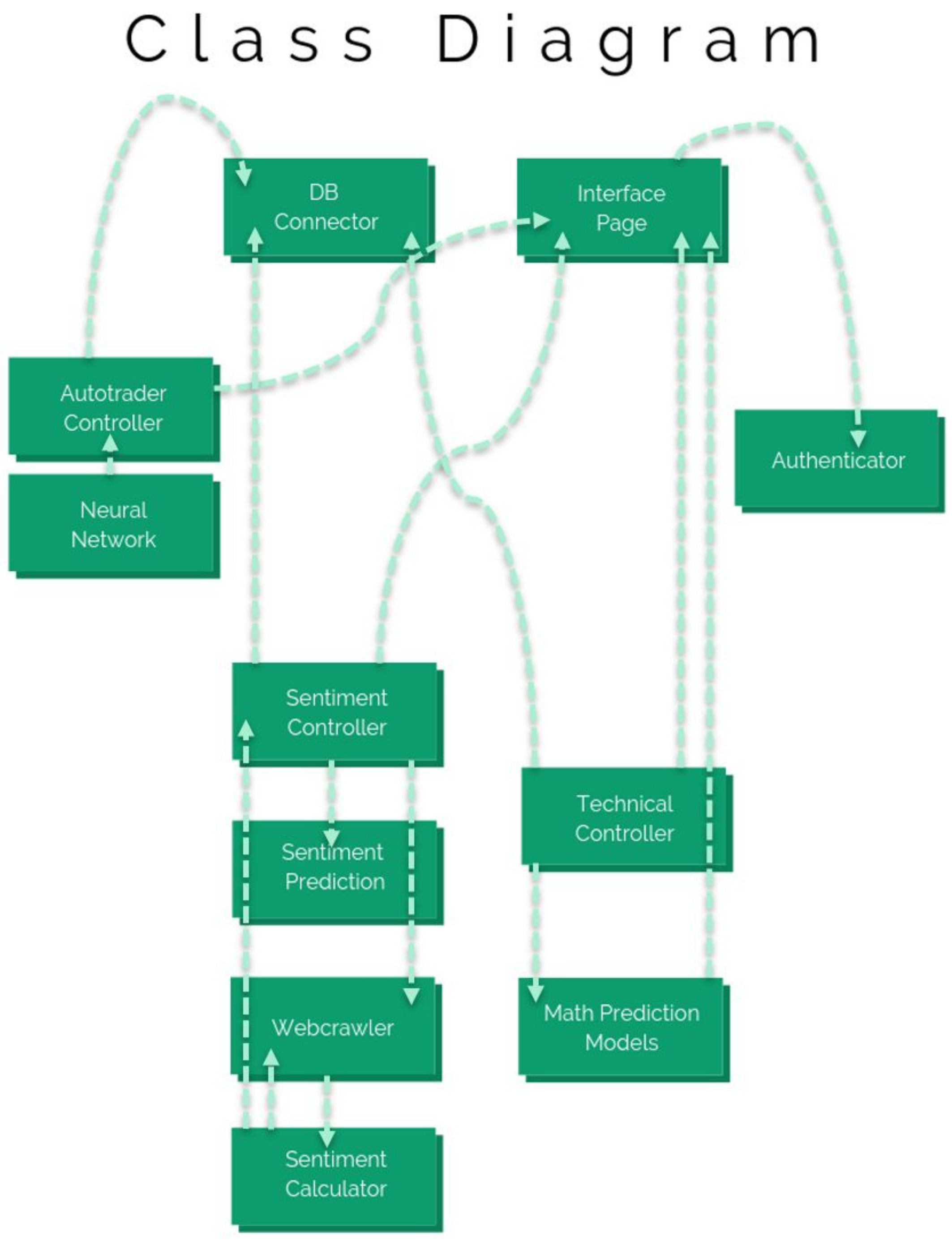


UC-5: DisplayPrediction Interaction Diagram

Automated Trader



8. Class Diagram and Interface Specification



Interface Specification

Interface Page (test copy.py)

Provides the user with the data needed for their reference, displays balance, report, browsing for stocks along with sentiment and price values for each stock. Interface also includes Login

+addUser(void):json_object

Updates the database and adds a user account to the list of users, then returns a string that confirms if the user was added successfully or unsuccessfully

+login(void):json_object

Checks if the user information matches a user in the database, then returns a string confirming whether it was successful

+deposit(void):json_object

Adds money to a user's account

+withdraw(void):json_object

Removes money from a user's account

+testFunc(void):json_array

Takes the user inputted stock ticker and checks for headlines related to it, then returns them in an array sorted by relevance. If at least five headlines cannot be found, returns an empty array.

+testFunc1(str):json_object

Helper function for testFunc, reformats data

+techFunc(void):json_object

Gets predictions for future prices of user inputted stock ticker and returns the various predictions in an array

+run_autotrader(void):json_object

Runs the autotrader for the amount of time specified by the parameters given by the user.

+loginn(void):json_object

Changes the website page when the user logs in

+loginb(void):json_object

Changes the website page when the user logs out

+logout(void):json_object

Returns user to index page of website after they log out

+lionel(void):json_object

Brings user to index page of website when they go to website

AutoTrader Controller

Aggregates data and send to proper locations and communicates with the rest of the subsystems such as the stock portfolio, Account balance, ledger...etc

+check_initial_deposit(void): void

Attempt to start up the AutoTrader and provide a negative initial deposit. Then, attempt to provide a valid deposit and relogin and check if the user is still prompted for an initial deposit because their initial deposit should already be recorded.

+initial_stock_selection(void): void

Enter a combination of valid and invalid stock tickers in quick succession as input to the function. Expected Results: The function should return an error message if the user attempts to choose a stock that is not in the database, otherwise, the stock should be added to the user's portfolio.

+get_stock_price(String stock_symbol): int

Enter various stock tickers both in and not in the database and compare the values returned to the actual prices as stored in the database.

+print_ portfolio(): String[]

This function prints the full portfolio, the number of companies in the portfolio, stocks currently owned and related information. Compare the printed output of this function with the actual portfolios, numbers of companies, and stocks currently owned of various users.

+purchase_shares_manually(): void

Take in user input for various stock tickers that they would like to purchase and check to see that they have the balance necessary to purchase the stock. If so, the number of stocks in their portfolio of this type will increase by as many as they bought and their balance will decrease by the amount of money they had to pay.

+sell_shares_manually(): void

Try entering different values as the user for the stock they would like to sell and the number of shares they would like to sell. The stock should already be in their portfolio for them to be able to sell it, while the number of shares they would like to sell must be greater than 0. Cover all corner cases with stocks not in the DB and negative/non-numerical values of shares.

+print_balance():void

Call the function repeatedly for different users with different account statuses to see whether or not their balance is properly printed for display. The function simply gets user account balance for display.

+request_trading_duration(int days, int hours, int minutes):void

The function prompt user to input time interval in days, hours, minutes ad runs the autotrade function accordingly.

+make_decision(): String decision

The function calls the correct transaction sell, buy or hold based of the results (prediction & sentiment).

- auto_trade():void

the function receives sentiment and analyzed prices and feeds these values to the neural network for an overall prediction. This is where the neural network is used for the decision on whether to buy, sell or hold. This function also prints to the portfolio and updates account

- withdraw_money(float value): void

  The function withdraws money successfully form the user balance and updates the portfolio.
- deposit_money(float value):void

Provide various float values of money that can be deposited in to a user's account. Make sure to cover negative and non-numeric values as error cases and for all other cases check balance to see that it has been appropriately updated in the database.

- compare_price(float currentPrice, float predictedPrice): int

  Provide various values of currentPrice and predictedPrice such that there are test cases for when currentPrice is greater than predictedPrice, vice versa, and when they are equal.
- getAccountBalance(): double[]

Gets company name from database

+ getTransactionHistory(stock_name: String): string[]
    Gets record of all transactions
+ accountBalance : double
    - Keeps track of the user account balance.
+ getCompanyName(stock_name: String): double[]
    Gets company name from database
+ getSharesOwned(stock_name: String): double[]
    Gets amount of shares owned for a given stock in stock portfolio
+ getTotalMoneyEarned(stock_name: String): double[]
    Gets total money earned from stock
+ stocks: string[stock_name]
    Keeps track of stocks in stock portfolio.

Neural Network

Where the neural network is held, initiates trading once all data needed for AutoTrader is present.

+ forward
This function is for forward propagation across the neural network
+ sigmoid
sigmoid is the activation function we used for the neural network
+ sigmoidPrime
This is the derivative of the sigmoid, used to optimize weights
+ backward
This function is for backward propagation across the neural network, calling the sigmoidPrime function in order to find the gradient descent of the error to optimize the weights
+ train
Trains data by running forward and backward functions.
+ saveWeights
Writes weights into a text file, w1 and w2

DBConnector (Database.py)

+addUser(string email,string password)

Function adds a user account to the database

+validateLogin(string email,string password)
Function validates user account login information

+deposit (string email, float amount):int
Enables user to deposit once user is logged into the database successfully and returns new balance (newbal)

+withdraw(string email,float amount): int
Enables user to withdraw once user is logged into the database successfully

+printStockCode(): void
Function prints requested stock name to the user

+getbal(string email): float
Functions returns balance amount in user account

Sentiment Controller
Listens to the Interface Page for requests for sentiment data. Upon receiving a request, it first transmits a message to the WebCrawler to get the headlines. Once that's completed, it communicates with the Sentiment Calculator to receive a score that it will store into the DBConnector.
+requestHeadlines(stockTicker: String): String []
    Requests the Webcrawler for the headlines corresponding to a particular ticker String
+requestPrediction(sentimentScore:double):String
    Requests the Sentiment Prediction class to return a prediction of "buy", "sell", or "hold" depending on the input integer "sentimentScore" sent in to the class
+calcAvgSentScore(headlines: String[]): double
    Calculates the average score of all the input headlines through numerous calls to the SentimentCalculator and averaging the results
+calculateSingleScore(headline: String): double
    Calculates a score for a single input headline, if needed, by calling the SentimentCalculator and using its calculate function.
+finalToUser(prediction: String, headlines: String[], avgScore: int): String[]

Sends the headlines, prediction, and average sentiment score to the interface to display those pieces of information to the user

+updateDB(avgScore: double): void

Sends the score to the DBConnector class so that the sentiment score assigned to that ticker can be updated in the database

+handleAutoTradeRequest(ticker: String): void

Calls requestHeadlines(stockTicker), calcAvgSentScore(headlines), and updateDB(avgScore) to calculate the average sentiment score for a specific Stock for the AutoTrader's use and the update the database accordingly.

-avgScore: double

Is returned by averageScore(totalScore, numScores). This is passed into requestPrediction() since it reflects the average sentiment from all the headlines gathered by the Webcrawler class

-totalScore: double

Is generated through multiple calls to requestSentiment(headline). The result of all the calls are summed up together to get totalScore, which is then passed as a parameter to averageScore(totalScore, numScores)

Sentiment Calculator

Goes over headlines of news and provides a sentiment result later used for generating stock predictions.

+__init__()

Simply initializes itself by setting up the lexicon. In order to this, it reads from the Loughran-McDonald dictionary and maps the sentiment values stored in there to the pre-existing Vader library to prepare the library for analysis.

+calculate(headline: String):double

Returns a sentiment score correlated to the stock required by the user.

-sentimentScore: double

Represents the score assigned to a particular headline

Sentiment Prediction

Generates a prediction to the user of whether to buy, sell or hold the stock.

+getPrediction(sentimentCalculator_data : double) : String

Calls the methods in the Sentiment Calculator to return a prediction to the user to whether buy, sell or hold the stock.

-prediction: String

Represents the actual string that is generated based on the score passed in

Webscraper

Grabs news headlines that correlates to stock specified by the user, used by the Sentiment Calculator in order to provide a numerated sentiment result.

-stockTicker : String

In order to implement the webcrawler, it first needs to know what stock the user is interested in order to grab headlines.

-newsAPIKey : String

Utilizes News API resources; needs an API key which correlates to the type of news which will be filtered and displayed to the user

-dateRange: int

Sets how many days back the newsapi should get news headlines from.

+getHeadlines(stockTicker : String): String []

Returns the headlines according to the stock ticker specified by the user

Technical Controller

Manages the passing of data and communication between classes related to the technical forecaster

+getStockData(stock_name : String) : double[][]

Retrieves stock information from the database. If the stock does not exist, returns an empty array

+graph_data(timeStamps: dateTime, stock_data: String[], stock_name: String): void

Graphs the data from the input onto the website.

+getPrediction(stock_name : double) : double[]

Calls the methods in the Math Prediction Model to get their calculated values and the overall prediction for a stock price,

Math Prediction Models

Makes multiple predictions from our various models.

+getRateOfChange(stock_data : double[][]) : double []

Returns measures of how dramatically a stock is expected to change

+getStochasticOscillator(stock_data : double[][]) : double []

Returns predictions of whether a stock will change trajectory

+getASI(stock_data : double[][]) : double[]

Returns predictions on whether the stock will increase or decrease in price

+getARIMA(stock_data : double[]) : double

Returns a prediction made by fitting the data to an ARIMA model

+getFourier(stock_data : double[]) : double

Returns a prediction made by fitting the data to a Fourier Series and taking the 4th coefficient

+aggregatePrediction(roc : double[], stoch_os : double[], asi : double[], arima_prediction: double, fourier_prediction : double) : double

Weights the various values and returns an aggregate prediction

Traceability Matrix

Concept	Classes	Reason
Controller	AutoTraderController, SentimentController, TechnicalController	The controllers coordinate the actions of the other classes.
Authenticator	AutoTraderController	This authenticates user login by checking the DB.
AccountTracker	AutoTraderController	Account information is stored in the database through the controller.
DatabaseConnection	DBConnector	It connects all of the controllers with the database with the querying and updating database.
GrapherConnection	SentimentController, TechnicalController	The graphers takes a graph of the stock prices and prediction and displays it in the InterfacePage
GoogleAPIConnection	Webcrawler	The webcrawler uses the Google API to get news headlines.
AlphavantageConnector	AutoTraderController	The controller gets the stock information and loads into the database.
Interface Page	InterfacePage	The interface page displays all the information on the website.
Webcrawler	Webscraper	The webscraper searches the internet for news headlines related to a certain stock entered by the user.
VaderSentimentAnalyzer	SentimentCalculator, SentimentPrediction	These calculate the sentiment from the headline data.
ARIMAModel	MathPredictionModels	The ARIMA model is calculated in MathPredictionModels.
StochasticOscillatorModel	MathPredictionModels	The StochasticOscillator model is calculated in MathPredictionModels.

AccumulativeSwingIndexModel	MathPredictionModels	The ASI model is calculated in MathPredictionModels.
RateOfChangeModel	MathPredictionModels	The ROC model is calculated in MathPredictionModels.
ModelAverager	MathPredictionModels	The ModelAverage in MathPredictionModels averages the model predictions for a final predicted price.
SectorSelector	AutoTraderController	The AT controller receivesthe sectorr of interest so the AutoTraders narrows down the trading to this sector.
StockRecommender	AutoTraderController	The AT controller prompts the Stock Recommender(which in turn receives the sentiment data and average model predicted values) for recommendations, sell,buy, and hold
PortfolioUpdater	AutoTraderController	The AutoTraderController updates the stock portfolio after each transaction.
TradingTimer	AutoTraderController	The AutoTraderController handles the trading duration for the AutoTrader.
RiskLevelIndicator	AutoTraderController, TechnicalController	Both the AutoTrader and Technical subgroups require risk level indication of stocks.
ObtainPrediction	SentimentCalculator, SentimentPrediction	The calculator calculates the sentiment from the headline and the SentimentPrediction return the sentiment in a number.
Autotrader	AutoTraderController	The autotrader handles the inputs and the outputs during the Auto trading period/mode. It sends recommendations to the stock portfolio to trade and makes actual transactions for the user, updates account balance and ledger as well.

OCL Contracts

Sentiment Analyzer Classes

1. Webscraper
   a. context Webscraper inv:
      self.headlines.length > = 0
      self.response.length >= 0
      self.url.contains(APIKey)
      self.url.contains("https://newsapi.org/v2/")
      self.url.contains("q=[query]")
   b. context Webscraper:: getHeadlines()
      pre: self.headlines.length = 0
      pre: self.response.length = 0
      pre: self.url.contains(APIKey)
      pre: self.url.contains("https://newsapi.org/v2/")
      pre: self.url.contains("q=[query]")
      post: self.headlines.length >= 0
      post: self.response.length > 0
2. SentimentController
   a. context SentimentController inv:
      self.totalScore >= -1*self.headlines.length and
      self.totalScore<=1*self.headlines.length
      self.avgScore >= -1 and self.totalScore <= 1
   b. context SentimentController::handleAutoTraderRequest()
      pre: self.headlines.length = 0
      pre: db.contains(self.ticker) //database must contain ticker
      pre: self.avgScore = 0
      post: self.headlines.length > 0
      post: self.avgScore >= -1 and self.avgScore <=1
   c. context SentimentController::finalToUser()
      pre: self.avgScore = 0
      pre: self.parsedHeadlines.length > 0
      pre: self.k = ''

pre: self.l = ‘’

pre: self.array = []

post: self.avgScore >= -1 and self.avgScore <=1

post: self.k.contains(“Average score of all Headlines is ”) and self.k.contains(avgScore)

post: self.l.length > 0

post: self.array.length = 2

d. context SentimentController::requestHeadlines()

   pre: db.contains(self.ticker) //database must contain ticker

e. context SentimentController::calcAvgSentScore()

   pre: self.totalScore = 0

   pre: self.numHeadlines = 0

   pre: self.headlines.length > 0

   post: self.totalScore > 0

   post: self.numHeadlines > 0

   post: self.avgScore >=-1 and self.avgScore <= 1

f. context SentimentController::calculateSingleScore()

   pre: db.contains(self.ticker) //database must contain ticker

   post: self.score >= -1 and self.score <= 1

g. context SentimentController::requestPrediction()

   pre: self.averageSent >= -1 and self.averageSent <= 1

   post: self.prediction.length > 0

h. context SentimentController::updateDB()

   pre: db.contains(self.stockname) //database must contain stockname ticker

   pre: self.sentiment >= -1 and self.sentiment <= 1

   post: db.contains(self.stockname, self.sentiment) //database contains key-value pair represented by inputs to the function

3. <u>SentimentCalculator</u>

   a. context SentimentCalculator inv:

      self.financeLexicon.contains(word->forAll(w | FinanceLexicon.txt)) //financeLexicon must contain all words in FinanceLexicon.txt

      self.newsAnalyzer.lexicon.contains(word->forAll(w | financeLexicon))

    //newsAnalyzer must contain all words in financeLexicon

   b. context SentimentCalculator::calculate()

      pre: self.message.length > 0

      post:self.ss[“compound”] >= -1 and self.ss[“compound”] <= 1

4. <u>SentimentPredictor</u>
   a. context SentimentPredictor inv:

      self.prediction >= 0

   b. context SentimentPredictor::predict()

      pre: self.score >= -1 and (self.score <= 1 or self.score == 2)

      pre: self.prediction = ‘’

      post: self.prediction.length > 0

Technical Forecaster Classes

1. <u>TechnicalController</u>
   a. Context TechnicalController inv:

      Stock_name not in database

   b. Context TechincalController::getStockdata()

      Pre: db.contains(self.stock_name)

      Pre: time_stamps = []

      Pre: open_prices = []

      Pre: close_prices = []

      Pre: high_prices = []

      Pre: low_prices = []

      Post: time_stamps.length > 0

      Post: open_prices.length > 0

      Post: close_prices.length > 0

      Post: high_prices.length > 0

      Post: low_prices.length > 0

   c. Context TechnicalController::getPrediction()

      Pre: roc = []

      Pre: stoch_os = []

      Pre: asi = []

      Pre: curPrice = 0

      Pre: arima_prediction = 0

Pre: fourier_prediction = []
Pre: prediction = []
Post: roc.length > 0
Post: stoch_os.length > 0
Post: asi.length > 0
curPrice > 0
arima_prediction > 0
Post: fourier_prediction.length > 0
Post: prediction.length > 0

2. MathPredictionModel
    a. Context MathPredictionModel inv:
        Any stock data is negative
    b. Context MathPredictionModel::calculateROC()
        Pre: roc = []
        Post: roc.length > 0
    c. Context MathPredictionModel::calculateStoch_OS()
        Pre: stoch_OS = []
        Post: stoch_OS.length > 0
    d. Context MathPredictionModel::calculateASI()
        Pre: asi = []
        Post: asi.length > 0
    e. Context MathPredictionModel::calculateARIMA()
        Pre: arima_prediction = 0
        Post: arima_prediction > 0
    f. Context MathPredictionModel::calculateFourier()
        Pre: fourier_predictions = []
        Post: fourier_predictions.length > 0
3. GrapherConnector
    a. Context GrapherConnector inv:
        Stock_name not in database
    b. Context GrapherConnector::graph_data()
        Post: static/assets/img.contains(a.jpg)
        //the graph is stored in the img directory within the assets directory

AutoTrader Classes

1. ATController
    a. context ATController inv:
        self.balance > = 0
        self.withdrawal >= 0
        self.initial_deposit > 0
        self.cnx.cursor().fetchall().contains(stock_symbol)
    b. context ATController::check_initial_deposit():
        pre: self.balance = 0
        post: self.initial_deposit > 0
    c. context ATController::initial_stock_selection():
        pre: stock_symbol.length > 0
        pre: db.contains(stock_symbol) //db must contain stock_symbol
        post: self.stock_portfolio = []
    d. context ATController::search_stock_symbol():
        pre: self.cnx.cursor().fetchall().contains(stock_symbol)
        pre: db.contains(stock_symbol)
        post: self.stock_portfolio.length > 0
    e. context ATController::get_stock_price():
        pre: stock_symbol.length > 0
        pre: db.contains(stock_symbol)
        post: stock_price > 0
    f. context ATController::print_portfolio_prices():
        pre: self.stock_portfolio.length > 0
        pre: self.stock_portfolio.keys > 0
        pre: self.stock_portfolio.values > 0
        post: stock_price > 0
    g. context ATController::print_portfolio():
        pre: self.stock_portfolio.length > 0
        pre: self.stock_portfolio.keys > 0
        pre: self.stock_portfolio.values > 0
    h. context ATController::purchase_shares_manually():
        pre: self.cnx.cursor().fetchall().contains(stock_symbol)

pre: db.contains(stock_symbol)
pre: self.balance > 0
post: self.portfolio(stock_symbol) > 0

i. context ATController::sell_shares_manually():
    pre: self.cnx.cursor().fetchall().contains(stock_symbol)
    pre: db.contains(stock_symbol)
    pre: self.portfolio(stock_symbol) > 0
    post: self.balance > 0
j. context ATController::print_balance():
    pre: self.balance >= 0
k. context ATController::deposit_money():
    pre: self.balance >= 0
    post: self.new_balance > self.balance
l. context ATController::withdraw_money():
    pre: self.withdrawal >= 0
    post: self.balance >= 0
m. context ATController::compare_price():
    pre: self.currprice > 0
    pre: self.predictedprice > 0
    post: return_value == 1 or return_value == 2
n. context ATController::make_decision():
    pre: sentiment > -1 and sentiment < 1
    pre: price > 0
    pre: analyzed_price > 0
    pre: stock_symbol.length > 0
    pre: db.contains(stock_symbol)
    post: self.balance >= 0
    post: self.portfolio(stock_symbol) >= 0
o. context ATController::auto_trade():
    pre: stock_symbol.length > 0
    pre: db.contains(stock_symbol)
    pre: trading_duration > 0
    post: self.balance >= 0
    post: self.portfolio(stock_symbol) >= 0

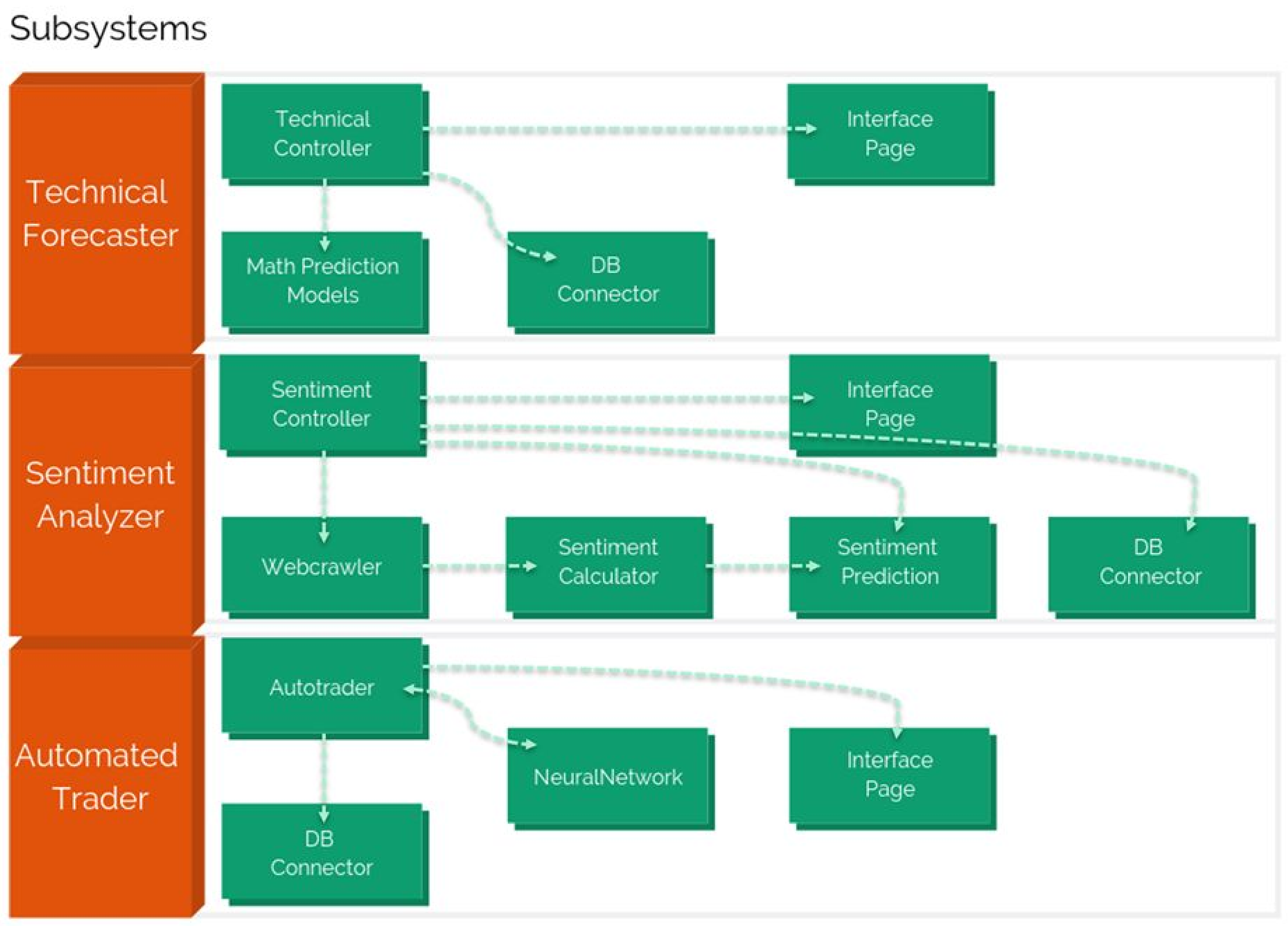
2. <u>NeuralNetwork</u>
   For our NeuralNetwork class, there are no constraints on the variables, and so an OCL Contract is not applicable.

9. System Architecture and System Design

Architectural Styles

1. <u>Event-driven Architecture</u>
   a. The whole process is triggered by a search request of a stock name. When the user requests information of a certain stock in any of the three subprojects, they begin processing and getting the predictions of that particular stock.
2. <u>Client-Server Architecture</u>
   a. The user interacts with the Interface Page and makes requests through that. The server is the controllers of the prediction models, sentiment analysis, and the automated trader. The server provides the information to the Interface Page at the user's request.
3. <u>Data-centric</u>
   a. The predictions and trading all revolve around the database. Without the database information, the predictions and trading could not happen. The database stores the user account information, the past stock prices, and the predictions for the autotrader to make decisions with.

UML Diagram of Subsystems



The three subsystems are the three subprojects, which are TechnicalForecaster, SentimentAnalyzer, and Automated Trader. The TechnicalForecaster contains its controller and the predictions of the models. The SentimentAnalyzer contains its controller and the sentiment predictions. The AutomatedTrader contains the necessary classes to make trades with the predictions from the database connector.

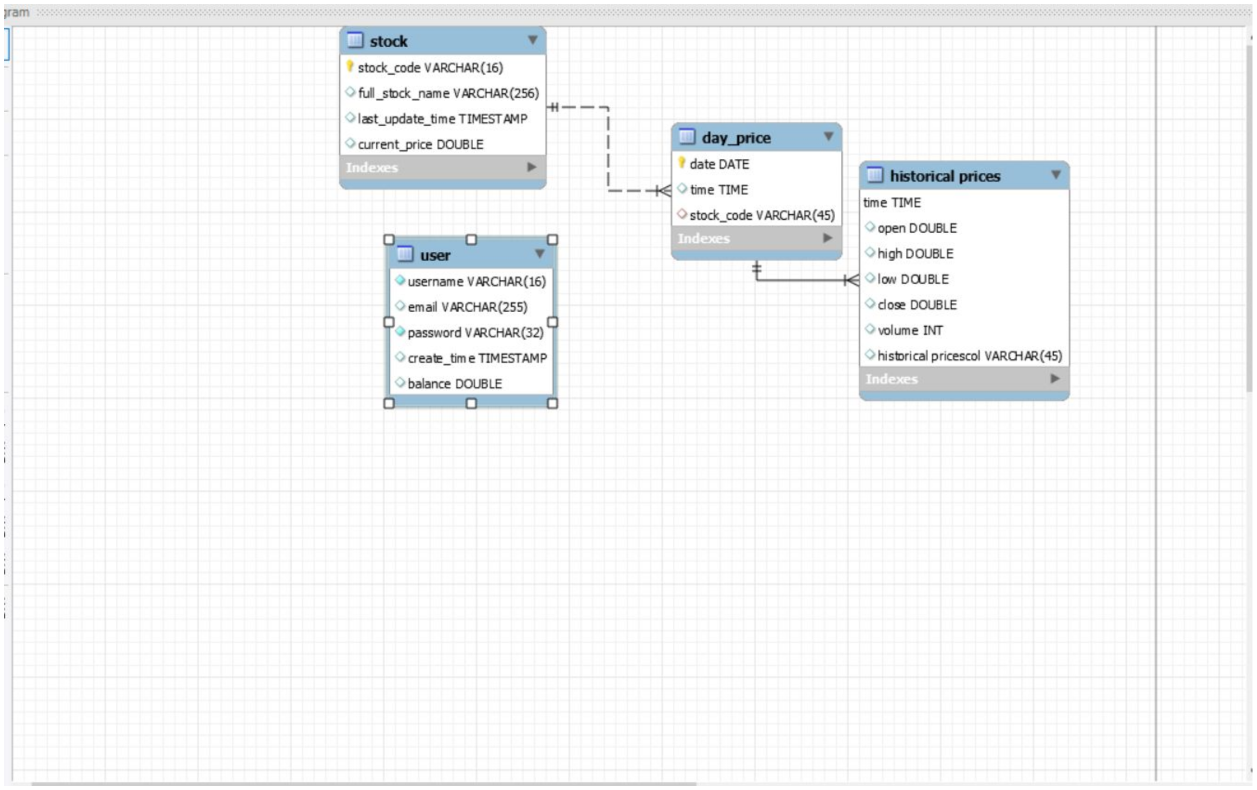
Mapping Subsystems to Hardware

In theory, we could have the system run on one computer, where the user can connect to the database and access our services using the same computer, in effect having the client and server share the hardware. However, we will have our system run on multiple computers, as users should be able to access our website through network connections from their own personal computers.

The users will act as clients, using their own machines to interact with our website through a web browser to send various requests regarding many stocks. The web browser will send data back and forth with our server, which is hosted on a separate computer. The server is connected to our database and updates it with the results of our analyses, as well as sending the information back to the web browser for the client to see.

Persistent Data Storage

Persistent Data Storage is required for this project, and the current database model is a relational database created in SQL.



Persistent Objects:

User: object that holds each user, with their username, email, password, and account balance

Stock: holds each valid stock along with its price, and points to the object day_price

Day_price: tied to each stock, holds the price history for a given day and time by pointing to historical_price

Historical_price: Holds the important information for a given day/time for a stock, holding things such as the open price, close price, high, and low price for a 5 min period.

Network Protocol

With a web service like ours, the messages are typically sent from a client to a server using the HTTP protocol. HTTP, which stands for HyperText Transfer Protocol, is the protocol used to request/receive Web pages from a Web server to a browser. We also plan to use REST, REpresentational State Transfer, as our messaging protocol vs SOAP,Simple Object Access Protocol, because our approach is to mainly locate a resource on a server and update it which REST would allow us to, where SOAP we would have to create mini services that would travel through the server and perform the required task when called by user, which would be cumbersome and take a lot of time to implement.

Because our back end is coded purely in Python, we used Flask as our framework to connect our code to the front end. The web server is a pure Python web server called Waitress, and the website is being hosted on Heroku, while the database is being hosted by Amazon Web Services.

Global Control Flow

Execution Orderliness:

The system is event driven as it waits for user input of what stock to get a prediction of. After the event is triggered, the procedures for making the predictions are linear. The user can query stock names in any order.

Time Dependency:

There are timers for the API connections in case of no response to prevent freezing/crashing. The Automated Trader has a trading timer that allows the user to set how long the automated trading should last. The system is of event-response type as it sets off when a user requests a stock name. The trading timer is based on real time and the user sets the time limit.

Concurrency:

The system is not concurrent during the events for a single user, but there will be threads for multiple users to perform requests at the same time. The server will need to handle requests and autotrade for multiple users. The shared data will need to be protected and synchronized, so the database will need to be protected since multiple parts will be updating and querying from the database at the same time.

Hardware Requirements

Screen size:

Our website will be accessible through any web-enabled device. Because screen sizes are always changing, we intend to employ responsive web design so that the layout of our website will change based on the size and capabilities of the device accessing it. For example, on a phone users might see content shown in a single column view, whereas a tablet might show the same content in two columns.

Server hardware:

Our server computer has a 24 core processor and 100GB of available hard drive space. It also must be able to access the internet.

Network communication:

Minimum network bandwidth of 56 Kbps is required.

10. Algorithms and Data Structures

Sentiment Analyzer

Algorithms

1. VaderSentimentAnalyzer Algorithm
   The algorithm returns a sentiment score in the range -1 to 1, from most negative to most positive. The sentiment score of a sentence is calculated by summing up the sentiment scores of each VADER-dictionary-listed word in the sentence.
   The individual words have a sentiment score between -4 to 4, but the returned sentiment score of a sentence is between -1 to 1. The algorithms applies an equation to the word’s score to map it to a value between -1 to 1.

2. Averaging Algorithm
   The algorithm sums up all the headlines sentiment scores correlate to a certain stock and divides the sum by the number of the headlines to give an average sentiment score to the user.

<u>Data Structures</u>

1. Arrays
   An array to maintain a list of headlines correlate to a specific stock
2. Hash Tables
   A hash table is used to store all the words added to the Vader lexicon and their equivalent scores. The following data types are used to store the information in the table:
   - Word: string
   - Score: int

<u>Technical Forecaster Averager</u>

This averager uses the mathematical models mentioned in Report 1, the ARIMA model, the rate of change, the accumulative swing index, the stochastic oscillator, and the fourier series analyser to produce an accurate prediction for a specific stock. Both the ARIMA model and the fourier series analyzer use past stock data to produce a specific estimate for the future price of a stock.

Initially, our plan was to look at how much these two predictions differ. If they were very similar, we could simply average the two estimates for a price prediction, but if they are relatively far apart, we would take into effect the other three indices, the rate of change, swing index, and oscillator and use them to reach a prediction. The swing index and rate of change both serve as indicators of the volatility, while the stochastic oscillator serves as an indicator of whether a stock is going to “change directions”.

We were going to use these factors to skew our predictions, but after some testing, we discovered that this testing resulted in poor accuracy for our predictions, and we tried scaling our weighting differently and retesting our new results. Afterwards, we decided to test on our plain ARIMA and fourier predictions, and after thorough testing, found that these calculations were very accurate, and decided with keeping them our predictions.

Math models:

Rate of Change Algorithm:

The algorithm for ROC is by calculating the rate of change at a set interval. The algorithm for calculating ROC is the change in closing price in the period divided by the previous closing price. The data structures used are Arraylists to store the ROC values and the stock prices.

Stochastic Oscillator Algorithm:

The algorithm uses the current price, the highest price in a set period of time, and the lowest price. The Stochastic Oscillator percentage is found by dividing the difference between the current and lowest price with the difference between the highest and the lowest prices. The data structures used are Arraylists to store the ROC values and the stock prices.

Accumulative Swing Index Algorithm:

The algorithm uses the low and high prices for a given day as well as the closing price for that day to determine how much the price of a stock changed over the course of the day, then does the same for previous days and compares them to see how much the price of a particular stock is expected to change over the course of a day.

ARIMA Algorithm:

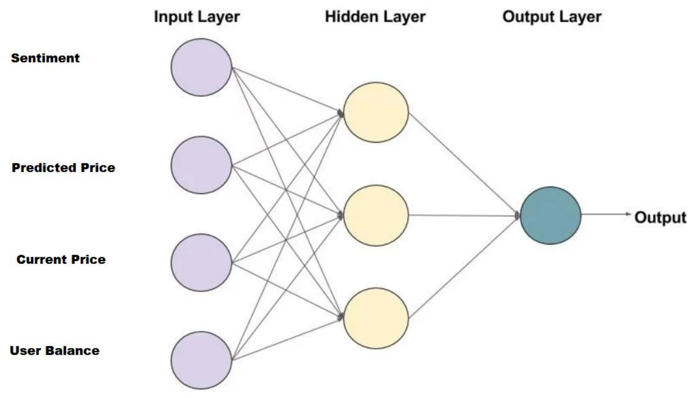
We imported a library called to use its ARIMA function for our forecaster. The library function uses the equations listed in the Mathematical Models section, and takes an array of stock data to fit the data to an ARIMA model. We can then ask the model to predict a future price of the stock in question and send this prediction back to the averager.

Fast Fourier Transform Algorithm:

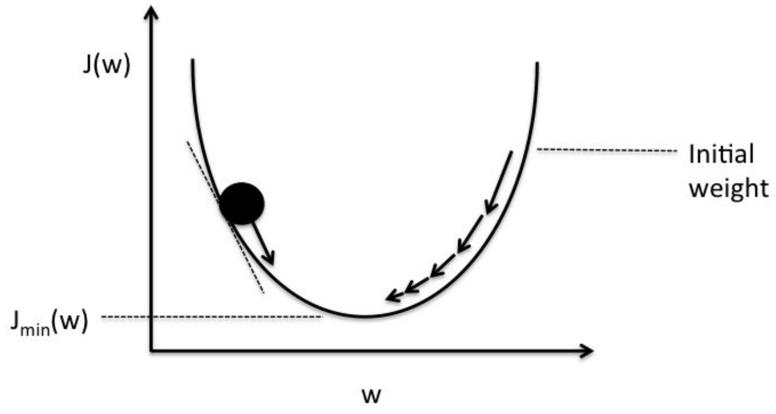
The fast fourier transform is an algorithm that can be used to map finite length sequences to analytical functions. Once there is a function representing the sequence, it becomes trivial to calculate a future value of the sequence. This algorithm hopes to map the given array of stock prices to a fitted function in order to get an accurate prediction on the future price of a stock.

Automated Trader

In order to allow the Auto-Trader present in our project to reach a competent level of accuracy and efficiency, a few neural network models and testing algorithms were considered for implementation, and the one that was decided was a Feed Forward Neural Network. This model was chosen over other models due to its relative lightweight operation and trainability compared to other possible choices.



The most popular training method used in Forward Feed Neural Networks is backpropagation, a method that involves finding the error between the desired output and the calculated output, and taking the derivative to find the gradient descent. The values of the weights assigned to the hidden layers are adjusted based on if the gradient is positive or negative to find the minimum error.



Schematic of gradient descent.

This method has downfalls, however. Backpropagation has limitations in that it is very proficient at finding a minimum, it is not necessarily the global minimum and the lowest possible error rate that can be done.

The training process of Neuroevolution starts with multiple instances of the network with random assortments of weights assigned to the hidden layers. This is known as Generation 1. The network's success is judged based on a fitness score, that is calculated based on the network's proximity to the actual price after the price interval being analyzed, along with the amount of net gain the user would get from the decision it makes. From this point, the instances with the best scores are saved and small variances are made to populate Generation 2, and then the process is repeated. To further improve performance, a partial implementation of backpropagation is used, in which one of the chosen instances are randomly picked to undergo the backpropagation process outlined above. This is a partial implementation because only one of these instances are back propagated per generation.

The above implementation was chosen due to the fact that neuroevolution is the most efficient way to find the version of the neural network with the lowest error percentage, with the extra bonus of using partial backpropagation to come to this conclusion faster than if it was not included.

+getPrediction(double sentiment ,double forecaster)

returns combined result for autotrader to use

+sell_stock(string stock_name)

no return, balance gets updated
+buy_stock(string stock_name)
no return, balance gets updated
+hold_stock(string stock_name)
no return, balance gets updated

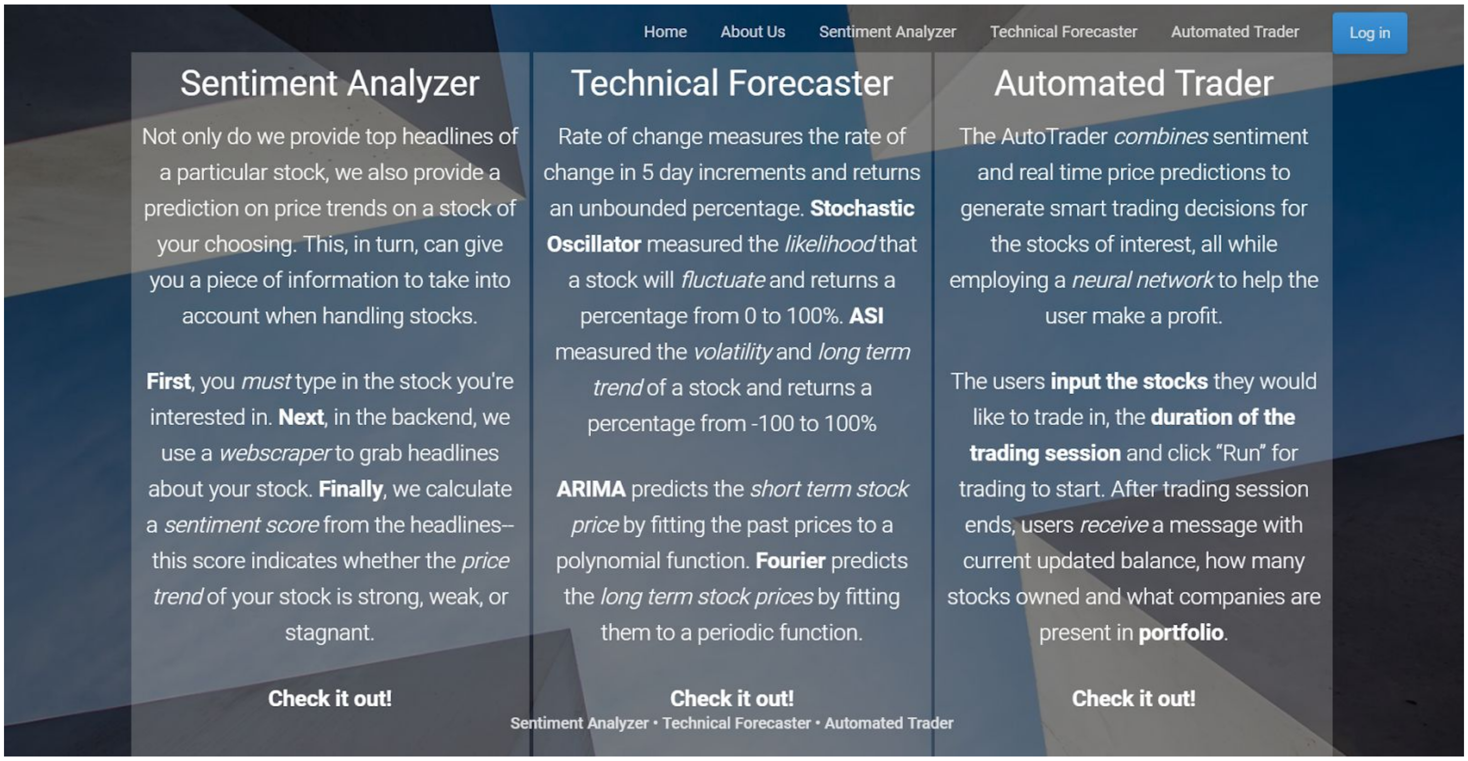
11. User Interface Design and Implementation

About page

Steps to use About Us UI:

1. The user navigates to the About Us tab.
2. The user scrolls down to a description to each of the features that KA$H web application provides, and 3 columns display to the user (left to right): Sentiment Analyzer, Technical Foreaster, Automated Trader.
    a. Sentiment Analyzer: describes what this feature does and the steps the user must go through to navigate the feature.
    b. Technical Forecaster: defines statistical models used to calculate prediction and describes the functionality of this feature.
    c. Automated Trader: describes what this feature does and the steps the user must go through to navigate the feature.

    Each column, at the end of each description, has a link to that the user can click on to direct them to the respective feature of the web application.



Sentiment Analyzer

Not only do we provide top headlines of a particular stock, we also provide a prediction on price trends on a stock of your choosing. This, in turn, can give you a piece of information to take into account when handling stocks.

First, you *must* type in the stock you're interested in. **Next**, in the backend, we use a *webscraper* to grab headlines about your stock. **Finally**, we calculate a *sentiment score* from the headlines-- this score indicates whether the *price trend* of your stock is strong, weak, or stagnant.

Check it out!

Technical Forecaster

Rate of change measures the rate of change in 5 day increments and returns an unbounded percentage. **Stochastic Oscillator** measured the *likelihood* that a stock will *fluctuate* and returns a percentage from 0 to 100%. **ASI** measured the *volatility* and *long term trend* of a stock and returns a percentage from -100 to 100%

ARIMA predicts the *short term stock price* by fitting the past prices to a polynomial function. **Fourier** predicts the *long term stock prices* by fitting them to a periodic function.

Check it out!

Automated Trader

The AutoTrader *combines* sentiment and real time price predictions to generate smart trading decisions for the stocks of interest, all while employing a *neural network* to help the user make a profit.

The users **input the stocks** they would like to trade in, the **duration of the trading session** and click "Run" for trading to start. After trading session ends, users *receive* a message with current updated balance, how many stocks owned and what companies are present in **portfolio**.

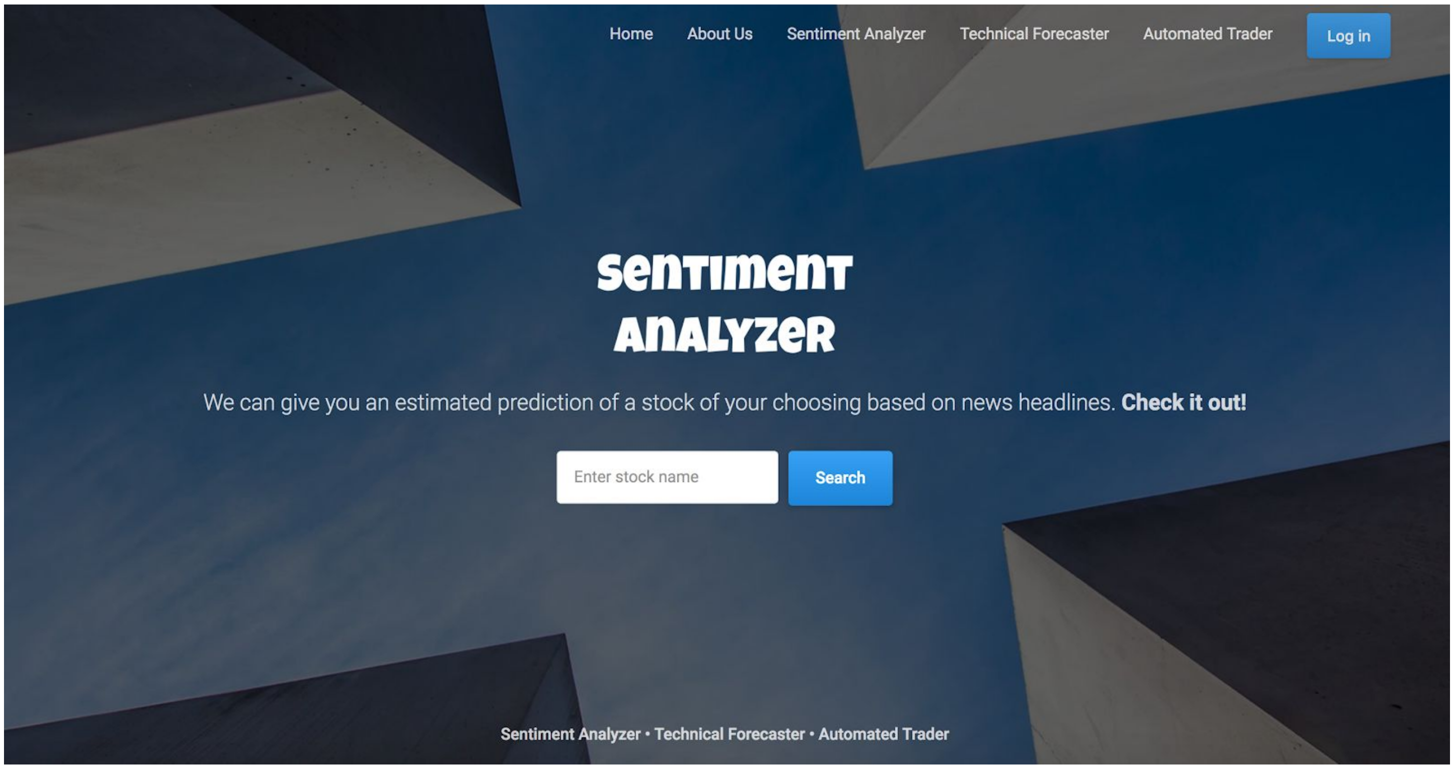
Check it out!

Sentiment Analyzer • Technical Forecaster • Automated Trader

Sentiment Analyzer

Steps to use the Sentiment Analyzer UI:

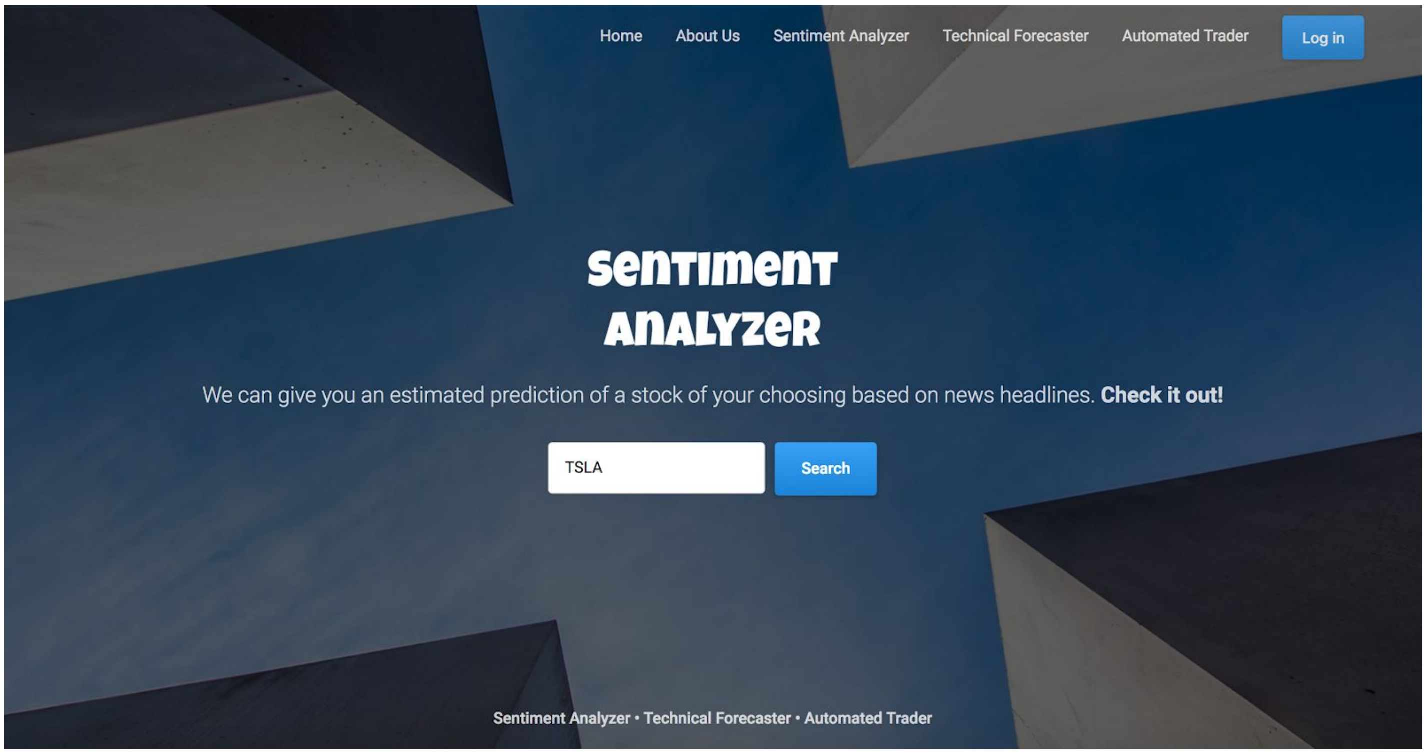
1. The user navigates to the Sentiment Analyzer tab.
2. The user inputs a stock ticker on the "Enter stock name" text box and hit the search button.
3. Four things show up to the user:
    a. Top five news headlines related to the stock.
    b. Average sentiment score of all the news headlines correlate with the stock.
    c. Prediction to whether buy, sell or hold the stock based on Sentiment.
    d. Graph that represents the Sentiment trend of the stock in the past 30 days.



Sentiment Analyzer

We can give you an estimated prediction of a stock of your choosing based on news headlines. **Check it out!**

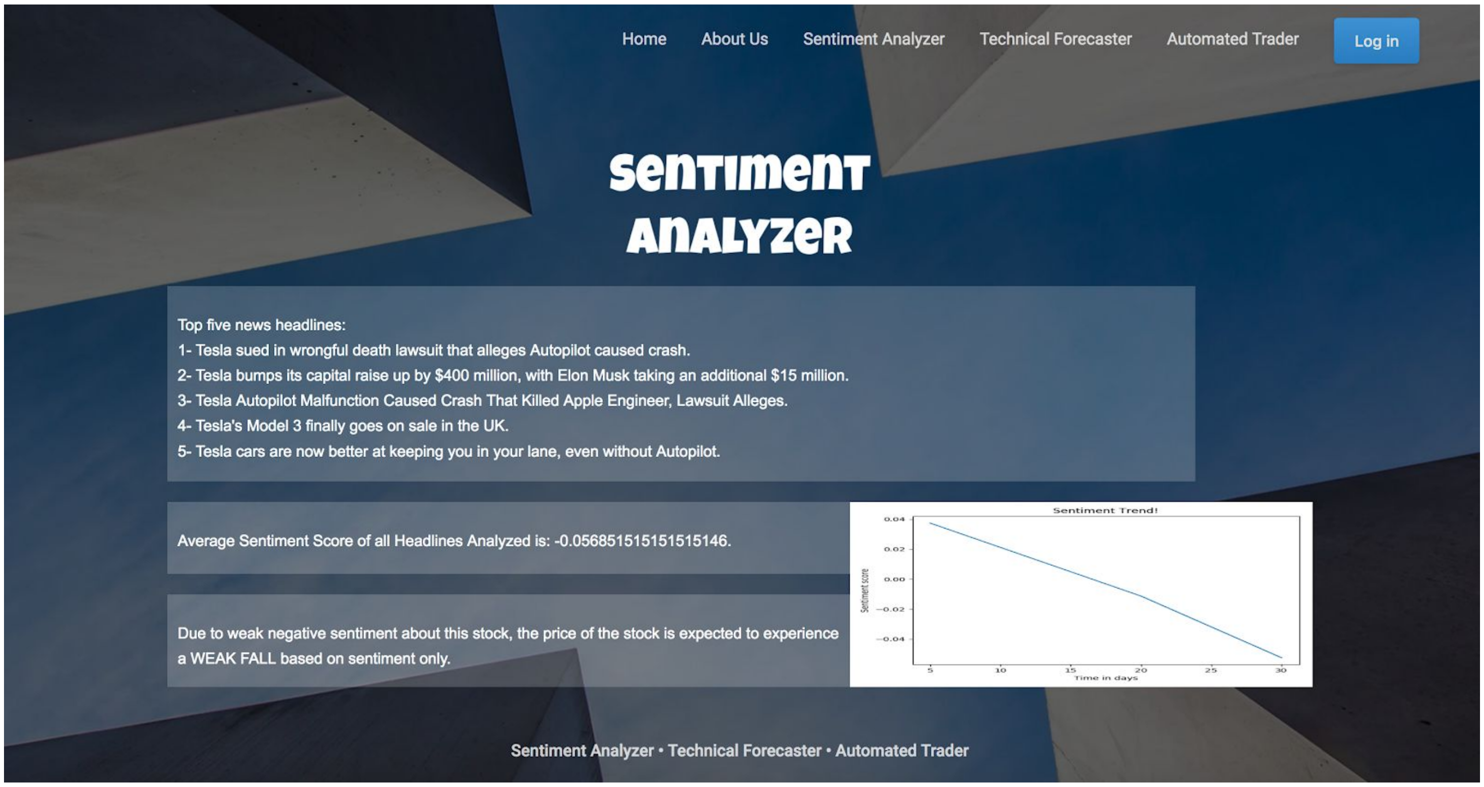
Enter stock name
Search
Sentiment Analyzer • Technical Forecaster • Automated Trader



Sentiment Analyzer

We can give you an estimated prediction of a stock of your choosing based on news headlines. **Check it out!**

TSLA
Search
Sentiment Analyzer • Technical Forecaster • Automated Trader

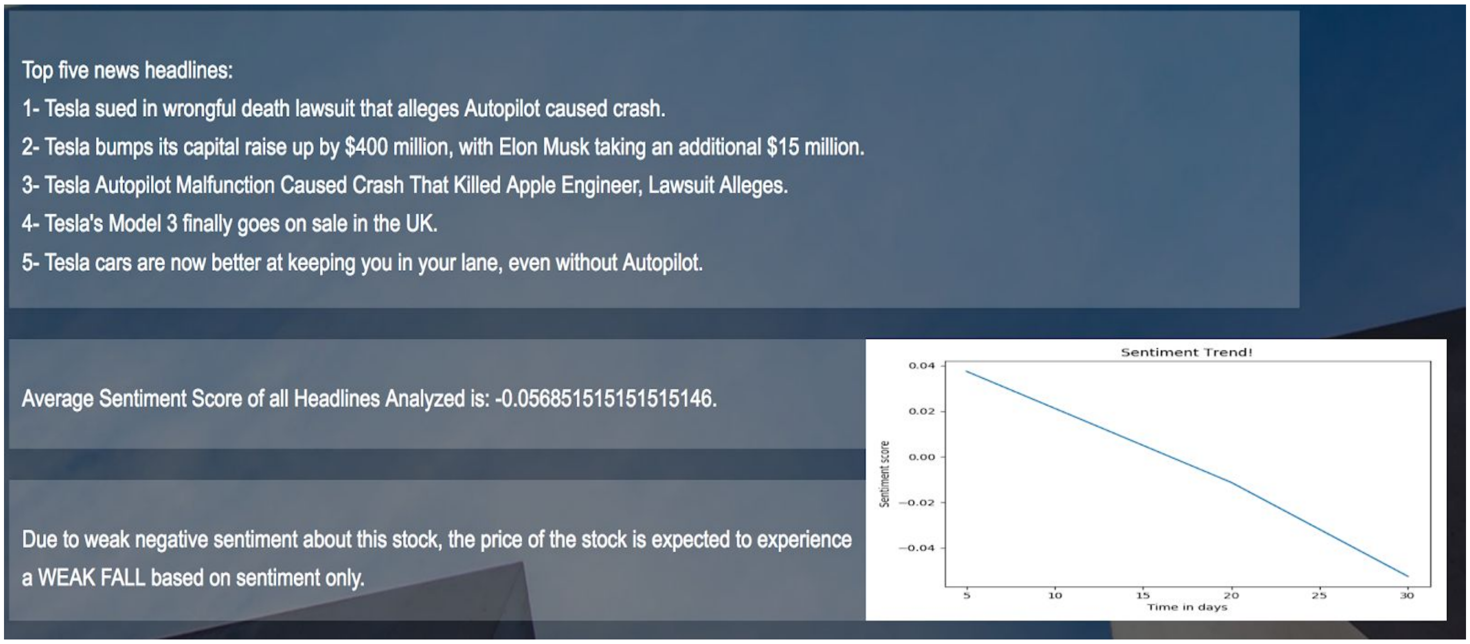


Sentiment Analyzer

Top five news headlines:
1- Tesla sued in wrongful death lawsuit that alleges Autopilot caused crash.
2- Tesla bumps its capital raise up by $400 million, with Elon Musk taking an additional $15 million.
3- Tesla Autopilot Malfunction Caused Crash That Killed Apple Engineer, Lawsuit Alleges.
4- Tesla's Model 3 finally goes on sale in the UK.
5- Tesla cars are now better at keeping you in your lane, even without Autopilot.

Average Sentiment Score of all Headlines Analyzed is: -0.056851515151515146.

Due to weak negative sentiment about this stock, the price of the stock is expected to experience a WEAK FALL based on sentiment only.


Top five news headlines:
1- Tesla sued in wrongful death lawsuit that alleges Autopilot caused crash.
2- Tesla bumps its capital raise up by $400 million, with Elon Musk taking an additional $15 million.
3- Tesla Autopilot Malfunction Caused Crash That Killed Apple Engineer, Lawsuit Alleges.
4- Tesla's Model 3 finally goes on sale in the UK.
5- Tesla cars are now better at keeping you in your lane, even without Autopilot.

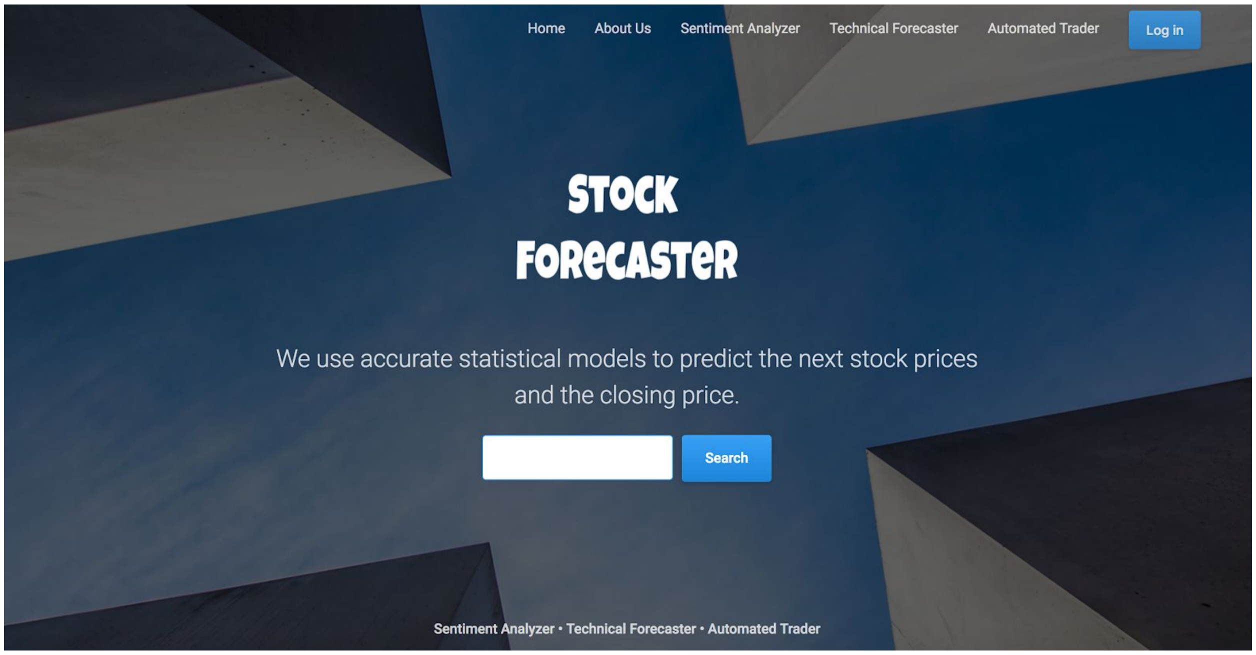
Average Sentiment Score of all Headlines Analyzed is: -0.056851515151515146.

Due to weak negative sentiment about this stock, the price of the stock is expected to experience a WEAK FALL based on sentiment only.

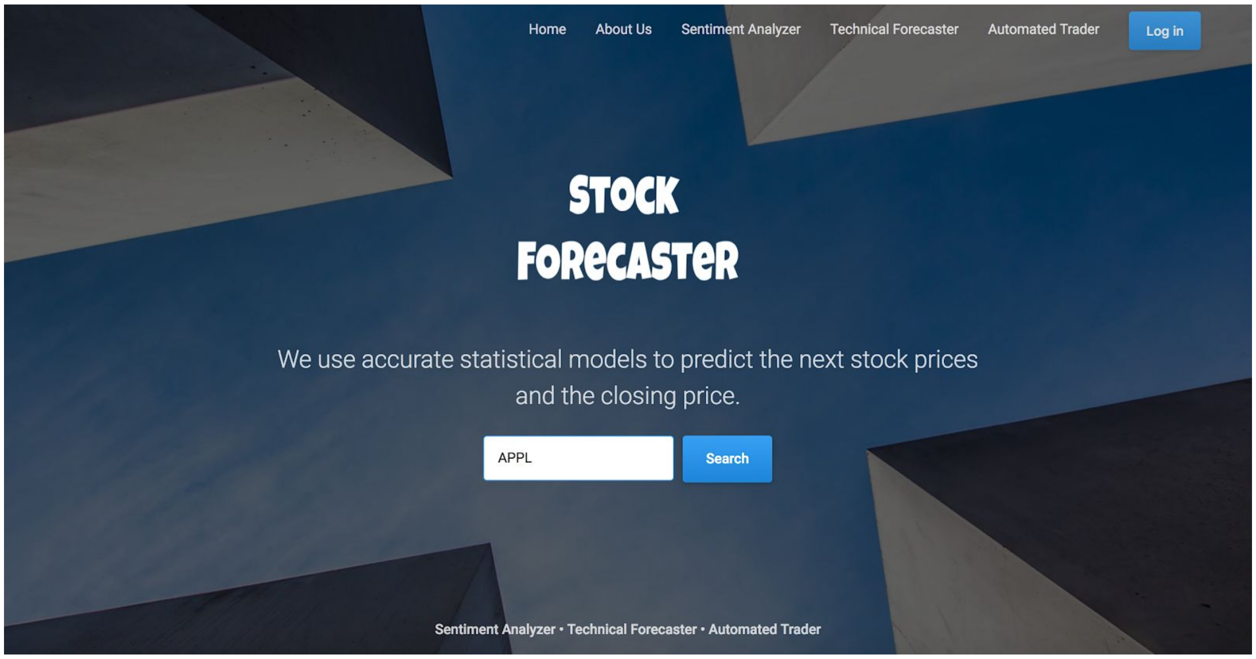
Technical Forecaster

Steps to use the Technical Forecaster UI:

1. The user navigates to the Technical Forecaster tab.
2. The user inputs a stock ticker on the search box and hit the search button.
3. Results page displays to user:
    a. The current price and the short and long term predictions
    b. The rate of change, stochastic oscillator, and the accumulated swing index values
    c. A graph displaying the past prices and the predicted price (green if higher, red is lower)



We use accurate statistical models to predict the next stock prices and the closing price.

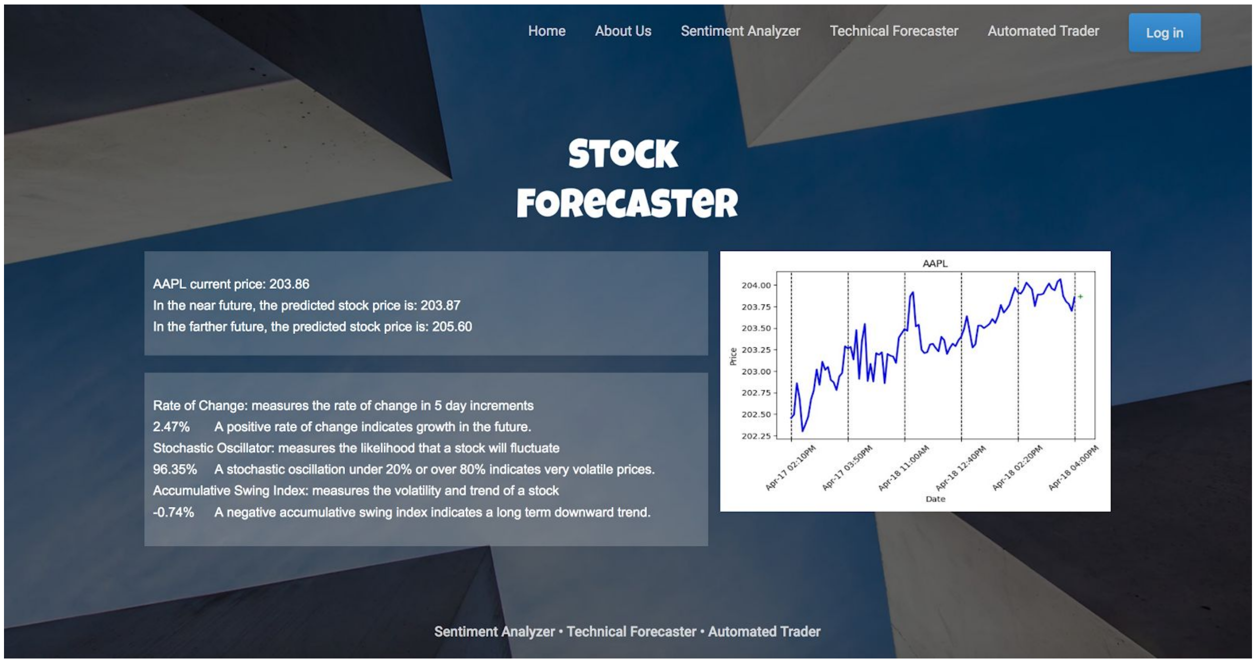


STOCK FORECASTER

We use accurate statistical models to predict the next stock prices and the closing price.

APPL

Sentiment Analyzer • Technical Forecaster • Automated Trader



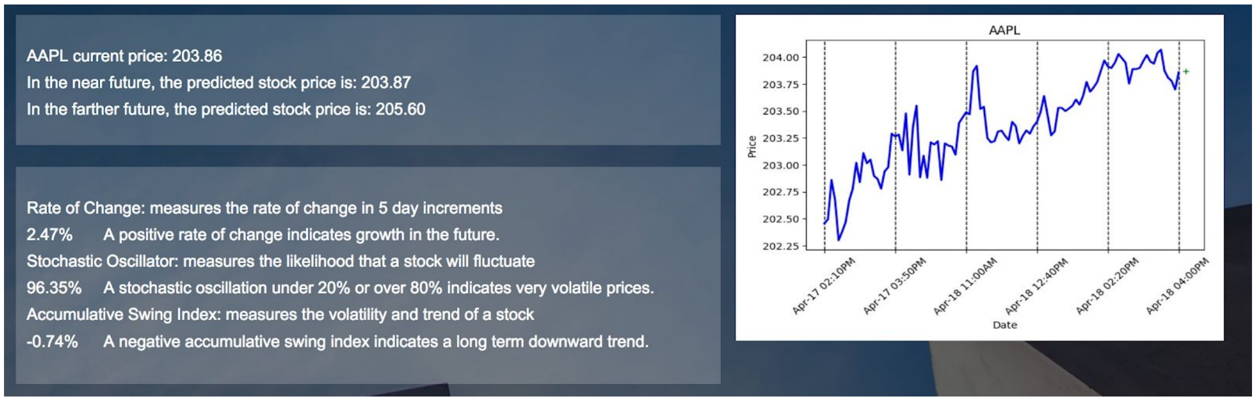
STOCK FORECASTER

AAPL current price: 203.86
In the near future, the predicted stock price is: 203.87
In the farther future, the predicted stock price is: 205.60

```
Rate of Change: measures the rate of change in 5 day increments
2.47%      A positive rate of change indicates growth in the future.
Stochastic Oscillator: measures the likelihood that a stock will fluctuate
96.35%     A stochastic oscillation under 20% or over 80% indicates very volatile prices.
Accumulative Swing Index: measures the volatility and trend of a stock
-0.74%     A negative accumulative swing index indicates a long term downward trend.
```

Sentiment Analyzer • Technical Forecaster • Automated Trader

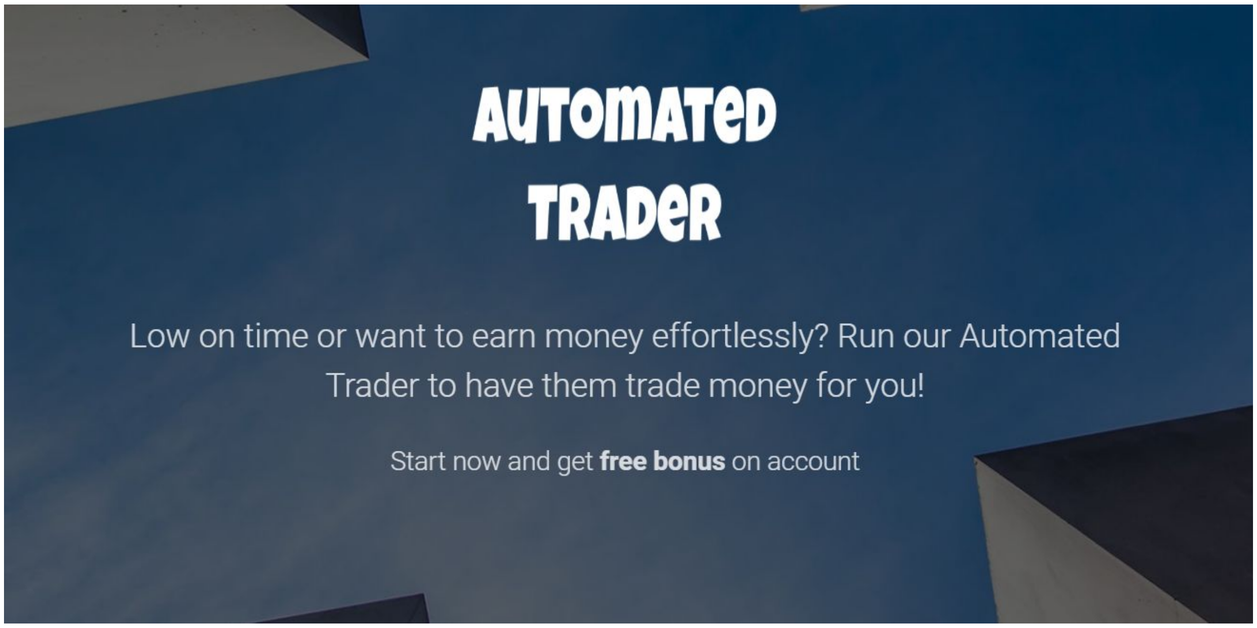
A closer look at the results:



AAPL current price: 203.86
In the near future, the predicted stock price is: 203.87
In the farther future, the predicted stock price is: 205.60

Rate of Change: measures the rate of change in 5 day increments
2.47% A positive rate of change indicates growth in the future.
Stochastic Oscillator: measures the likelihood that a stock will fluctuate
96.35% A stochastic oscillation under 20% or over 80% indicates very volatile prices.
Accumulative Swing Index: measures the volatility and trend of a stock
-0.74% A negative accumulative swing index indicates a long term downward trend.

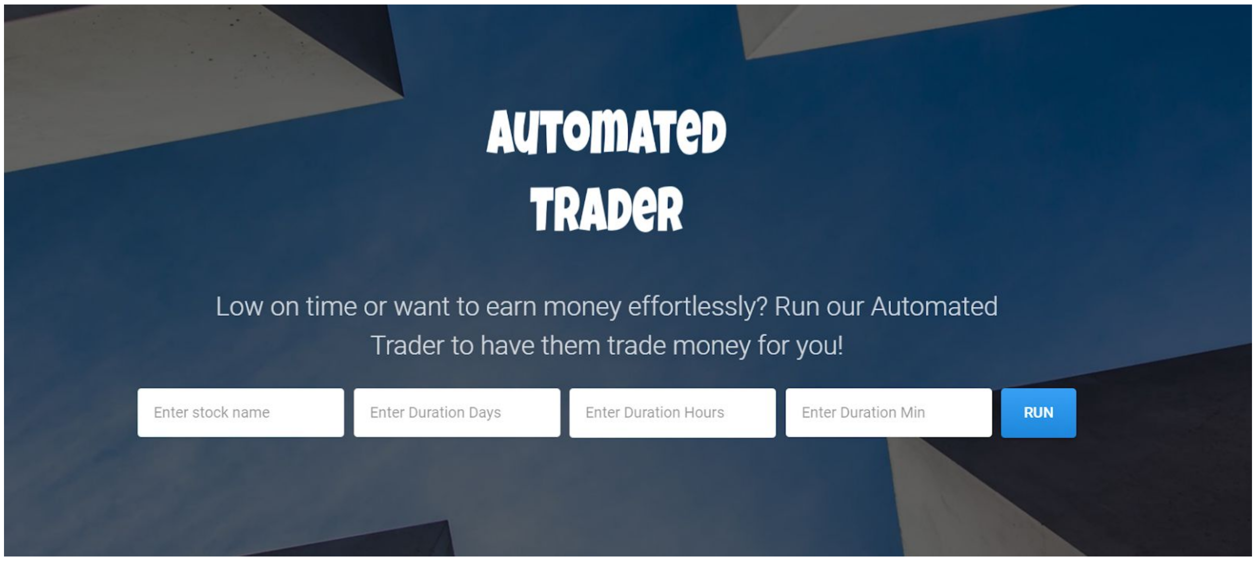
Automated Trader



AUTOMATED TRADER

Low on time or want to earn money effortlessly? Run our Automated Trader to have them trade money for you!

Start now and get **free bonus** on account


AUTOMATED
TRADER
Low on time or want to earn money effortlessly? Run our Automated Trader to have them trade money for you!
Enter stock name
Enter Duration Days
Enter Duration Hours
Enter Duration Min
RUN

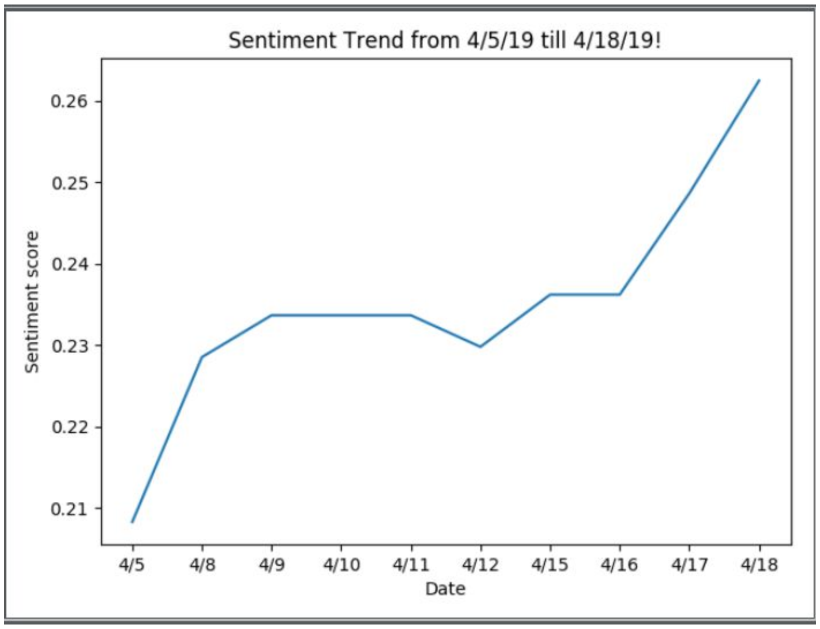
12. Design of Tests

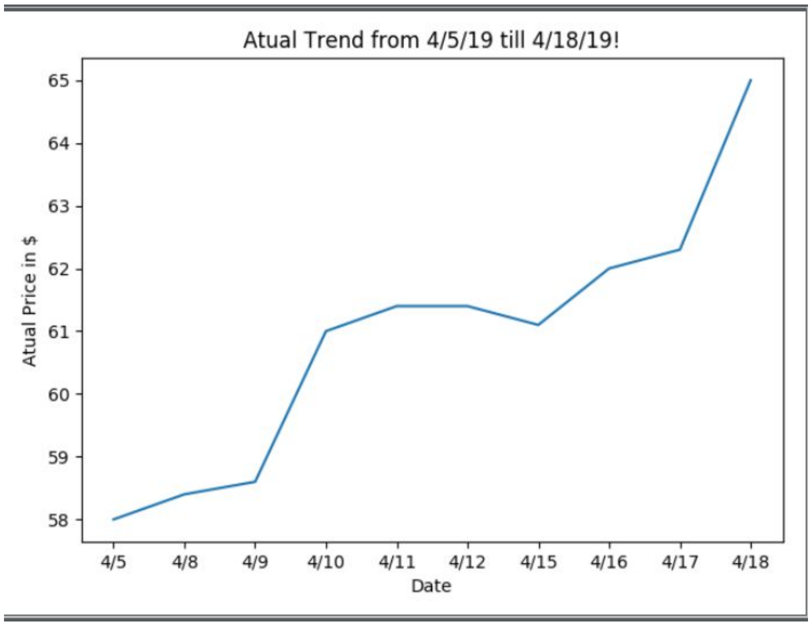
Sentiment Analyzer

Acceptance/Accuracy Test

List Of Stock Tickers tested = ["AAPL","AA", "A", "AAL", "AABA", "AAC", "AAOI", "AAN", "AAON", "AAP", "AAT", "AAU", "AAWW", "AAXJ", "AAXN", "AB", "ABB", "ABBV", "ABC", "ABCB", "ABDC", "ABEO", "ABG", "ABIL", "ABM", "ABMD", "ABR", "AADR",
"ACST", "ACRX", "ACRS", "ACRE", "ACP", "NFLX",
"ACOR", "ACNB", "ACN", "ACMR", "ACM", "ACLS","ACIW", "ACIU", "ACIM",
"ACIA", "ACHV", "ACHN", "ACHC", "ACH", "ACGLO", "ACGL", "ACET",
"ACER", "ACCO", "ACC", "ACBI", "ACB"]

We wrote code to generate two graphs: one based on the actual prices of the stock and the other one based on the sentiment scores generated by our sentiment analyzer.
The actual scores are saved in the database for up to one month and the news api we are using to get the headlines can go up to 30 days in the past for free.
We tested 57 stock tickers from the period of 4/5/2019 to 4/28/2019. Our Sentiment trend graph looked so similar to the actual trend graph for all the tickers we tested. The code used for generating the two graphs for all stock tickers we tested is included in the Project electronic archive.
Below are the graphs for the actual trend and the sentiment prediction trend for "AAXN" stock ticker





Unit Tests

<u>Sentiment Controller</u>

+trend(stockTicker: String): int [][]

Description of test: Given a valid String ticker, this method should return a graph that represents the sentiment trend in the past 30 days

Success criteria/expected results: returns a graph that represents the sentiment trend in the past 30 days

Failed criteria: a default image should pop up to the user instead of the actual graph

Coverage: Giving this method various String tickers, from tickers that don't exist to popular tickers (with many news articles) to unpopular tickers (with few tickers), to see how the response changes

+requestHeadlines(stockTicker: String): String []

Description of test: Given a valid String ticker, this method should request the top headlines in a properly formatted String array from the Webcrawler

Success criteria/expected results: An array of Strings containing popular news headlines on the Web about a particular stock from the Webcrawler

Failed Criteria: An error message explaining why the Webcrawler was not able to get the requested headlines.

Coverage: Giving this method various String tickers, from tickers that don't exist to popular tickers (with many news articles) to unpopular tickers (with few tickers), to see how the response changes.

+requestSentiment(headline:String): double

Description of test: Given a String headline text, this method should request the SentimentCalculator to calculate a score and get a double (floating point number) in response.

Success criteria/expected results: A double between -1.0 and 1.0 that represents the compound sentiment of the input headline is received.

Failed Criteria: A message indicating that the calculator did not have enough

words in its lexicon to accurate gauge the sentiment of the headline.

Coverage: Varying the input headlines text from strongly positive headlines, strongly negative headlines, finance-heavy headlines, and words that are not in the Vader lexicon to get the best possible picture of how well the calculator performs.

+requestPrediction(sentimentScore:double):String

Description of test: Given a floating point value between -1.0 and 1.0, request the SentimentPredictor for a price movement prediction represented as a String to send to the Interface Page for display purposes.

Success criteria/expected results: Expected to receive "strong rise" if the input value was between 0.65 and 1.0, "weak rise" if the input value was between 0.3 and 0.65, "stagnant" if the input value was between -0.3 and 0.3, "weak fall" if the input value was between -0.65 and -0.3, and "strong fall" if the value was between -1.0 and -0.65.

Failed Criteria: If the input value was not within that predefined range, the request will fail with an error message that will be displayed.

Coverage: The method will be tested with values in all ranges to test for all potential strings that it can receive.

+averageScore(total: double, numHeadlines: int): double

Description of test: The function will be tested simply by giving various combinations of total and numHeadlines to see whether the value returned matches with total/numHeadlines calculated on a calculator.

Success criteria/expected results: Upon success, the function will correctly compute total/numHeadlines.

Failed Criteria: If the inputs are not representable by floating point values, it will return an error. Alternatively, if the function's return value does not

Match with the quotient of total and numHeadlines, the test will have failed.

Coverage: Different combinations of input values will be selected to account
for the different cases; i.e. total > numHeadlines, total = numHeadlines, total < numHeadlines, both numbers are floats, one is an integer, both are integers, at least one is neither a float nor an int, etc.

+sendToInterface(prediction: String, headlines: String[]): void

Description of test: The function will be provided with a set of headlines and a prediction String based on what the controller received from its request functions. It is expected that this will return the headlines array and the prediction String.

Success criteria/expected results: The successful outcome of this function will result in the Interface class having access to the prediction and headlines fields to display.

Failed Criteria: The function will have failed if the Interface Class cannot access the headlines or the prediction strings OR if any of the data is malformed.

Coverage: The test cases will largely revolve around changing the size of the

input string and array to ensure that the size of the data needed to go to the Interface class is not an issue.

sendToDB(avgScore: double): void

Description of test: Given a floating point number representing a calculated sentiment score, this function will send that value to be used in the database

Success criteria/expected results: On success, the function is expected to retrieve the sentiment score variable and return it so that the DBController can use it to update the database.

Failed Criteria: The function will have failed the test if the DBController cannot access or receives malformed data.

Coverage: The tests will primarily be concerned with different values of numbers that the function is expected to return. This will ensure that there is no overflow if the floating point numbers become large and are properly truncated if necessary.

Sentiment Calculator
+getStockInfo(stock_name : String) : double[]

Description of test: Given a string, the method should retrieve stock information from the database.

Success criteria/expected results: When given a valid stock name, the method will return the correct data from that stock.

Failed Criteria: Given an empty or invalid name (stock is not in database), the method returns an empty array.

Coverage: We will test this method with multiple stock names in our database, in addition to a few invalid names.

+getSentiment(headline: String):double
Description of test: Given a string, the method returns a sentiment score correlated to the given headline.

Success criteria/expected results: The method is expected to return a number(sentiment score) between -1.0 and 1.0 that correlate to the input headline.

Failed Criteria: Given an empty headline or a headline that doesn't correlate to the required stock.

Coverage: We will test this method with multiple stock names in our database, in addition to a few invalid names.

<u>Sentiment Prediction</u>

+getPrediction(sentimentCalculator_data : double) : String

Description of Test: The function will be given various values of floating point numbers ranging from -1.0 to 1.0 and we will ensure that the output text prediction is consistent with the input value provided.

Success criteria/expected results: Expected to return “strong rise” if the input value was between 0.65 and 1.0, “weak rise” if the input value was between 0.3 and 0.65, “stagnant” if the input value was between -0.3 and 0.3, “weak fall” if the input value was between -0.65 and -0.3, and “strong fall” if the value was between -1.0 and -0.65.

Failed Criteria: The test will have failed if it receives an invalid input value OR if it outputs a string that does not match the corresponding input value.

Coverage: We will conduct multiple tests with input data from every possible range to generate all possible string outputs. We will also test data outside of that range to ensure that it handles the error input correctly.

<u>Webcrawler</u>

+getHeadlines(stockTicker : String): String []

Description of test: Given a string, specifically, a string that contains the the stock chosen by the user, the method returns headlines pertaining to the stock name.

Success criteria/expected results: When given a string from the user, it will

Failed criteria: When given an empty string, the result should return back that the string input is a stock not located in the database, or that headlines could not be retrieved for the string input.

Coverage: Giving this method multiple string inputs--inputs that do not correlate to a stock name in the database and inputs that do correlate to a stock name existing in the database

System Tests: Use Case Acceptance Tests

Use Case Tested: DisplaySentiment
Test -case Identifier: TC-1 Test covers: Graphical User Interface Assumption: A valid sentiment value has been calculated by the SentimentCalculator and sent to the GUI by the SentimentController. Steps: ● User requests sentiment score related to a particular stock Expected: A sentiment score will be displayed to the user Fails if: The sentiment calculator does not receive any words in its lexicon and cannot calculate a sentiment score. A message indicating this will be sent to the user showing that headlines are not indicative of reliable sentiment data.

Use Case Tested: SearchStock

Test -case Identifier: TC-2 Test covers: Graphical User Interface Assumption: The user inputs a valid stock. Steps: ● User requires information related to a particular stock [its stock prices, ticker, recent news headlines, and statistical and/or sentiment analysis based on recent data about the stock] Expected: The required information will be displayed to the user. Fails if: The user did not input a valid stock. In this case the user should see an error message clarifying that they did not input a valid stock.

Use Case Tested: DisplaySentimentPrediction
Test -case Identifier: TC-3 Test covers: Graphical User Interface Assumption: A valid prediction has been generated by the SentimentPredictor and has been sent to the UI from the SentimentController. Steps: ● User requests sentiment score related to a particular stock and it extrapolates a prediction from the score Expected: A sentiment prediction will be displayed to the user. Fails if: The sentiment calculator does not receive any words in its lexicon and cannot calculate a sentiment score. A message indicating this will be sent to the user showing that headlines are not indicative of reliable sentiment data.

Technical Forecaster

Unit Tests

Technical Controller

+getStockData(stock_name : String) : double[][]

Description of test: Given a string, the method should return the array of previous prices and timestamps for the named stock

Success criteria/expected results: When given a valid stock name, the method will return the correct data from that stock. Given an empty or invalid name (stock is not in database), the method returns an empty array.

Coverage: We will test this method with every stock name in our database, in addition to a few invalid names.

+getPrediction(stock_name : String) : double[]

Description of test: Given an name of a stock, the method should return a prediction for the next number in the array

Success criteria/expected results: When given a non-empty array, the method should return a reasonable number. Given an empty array, the method should return an empty array.

Coverage: We will test this method with most stock data arrays in our database in addition to empty arrays.

Math Prediction Models

+getRateOfChange(stock_data : double[]) : double[]

Description of test: Given an array of prices, verify that it returns a double that indicates the rate of change of the stock.

Success criteria/expected results: When given a valid price array the rate of change should be positive for progressively increasing prices and negative for decreasing ones. The magnitude can be verified by looking at a plot.

Coverage: We will test this method with a sample of generated data for which the calculation is verifiable by another method. Then we will verify it with historic stock data.

+getStochasticOscillator(stock_data : double[]) : double

Description of test: Given a string, the method should return a double between 0 and 1 that indicates the likelihood of the stock to change direction.

Success criteria/expected results: When given a valid stock array, the method will return the correct stochastic index value for that stock.

Coverage: We will test this method with generated stock data to verify the output against an alternatively calculated method. Then we will verify it using historical stock data.

+getASI(stock_data : double[]) : double

Description of test: Given a string, the method should return a double between 0 and 1 indicating the largest prior change in the stock price as a fraction of the day end price of the stock.

Success criteria/expected results: When given a valid price array, the method will return the accumulative swing index of the stock.

Coverage: We will test this with generated data for which the calculation is verifiable by another means. Afterwards we will test it on historic stock data to verify its accuracy.

+getARIMA(stock_data : double[]) : double

Description of test: Given an array of stock data, the method should return the fitted ARIMA model prediction for the next stock price

Success criteria/expected results: When given a non-empty array, the method should return a reasonable number as calculated by ARIMA. Given an empty array, the method should return -1.

Coverage: We will test this method with most stock data arrays in our database in addition to empty arrays.

+getFourier(stock_data : double[]) : double

Description of test: Given an array of stock data, the method should return the Fourier analysis model prediction for the next stock price

Success criteria/expected results: When given a non-empty array, the method should return a reasonable number as calculated by Fourier Analysis. Given an empty array, the method should return -1.

Coverage: We will test this method with most stock data arrays in our database in addition to empty arrays.

+aggregatePrediction(roc : double, stoch_os : double, asi : double, arima_prediction: double, fourier_prediction : double) : double

Description of test: Given these parameters, the method should return our final prediction for the next stock price.

Success criteria/expected results: When given all of these parameters, the method should return a reasonable number through weighted averages. Aside from rate of change, if any other parameters are negative, the method should return -1.

Coverage: To test whether our weight averager is working correctly, we can load in predetermined parameters and compare the results with pre-calculated averages. We will also have to test the results in the presence of negative parameters (other than rate of change).

<u>Grapher Connector</u>

+graph_data(stock_data : double[], prediction : double) : jpg

Description of test: Given an array of stock prices and a prediction, the method should return a graph of the stock prices along with that prediction

Success criteria/expected results: A graph will all of the points listed in the array and the prediction should be created.

Coverage: There is not much testing we can do in terms of varying the inputs, as even empty arrays would only result in no points being graphed. We can only test by inputting various arrays and predictions and ensuring the graph is correct.

System Tests: Use Case Acceptance Tests

Use Case Tested: DisplayPrediction
Test -case Identifier: TC-4 Test covers: Graphical User Interface Assumption: A valid prediction has been calculated by the MathPredictionModel and sent to the UI by the TechnicalController Steps: ● User requests a prediction for a stock price in the near future Expected: The prediction will show up on the UI for the user to see Fails if: The user did not input a valid stock. In this case the user should see an error message clarifying that they did not input a valid stock.

Use Case Tested: DisplayGraph
Test -case Identifier: TC-5 Test covers: Graphical User Interface Assumption: Steps: ● User searches for a stock using a search bar ● User requests a prediction for a stock Expected: A graph of the stock, along with its predicted price if requested, will show up on the US Fails if: The user did not input a valid stock. In this case the user should see an error message clarifying that they did not input a valid stock.

<u>Technical Forecaster Subsystem test (testing prediction algorithms):</u>

In order to test the accuracy of the predictions given by the technical forecaster, we set up a python program to go through old stock data that was stored in our database and compare its predictions to existing "future" data. For example, the program would look at the prices starting with yesterday's closing price and going back 1 month in order to get a prediction on today's opening price. Then the program calculates the percent error between its prediction and the actual price. Using roughly forty stocks and going back roughly one month we found that our 5 minute predictions were within 99% of the actual price and that our hourly predictions were within 95% of the actual price.The file that contains the code used to test the prediction algorithms is found in the project archive as filename testingAccuracy.py

Automated Trader

<u>Unit Tests</u>
<u>AutoTraderController</u>
+check_initial_deposit(void): void

Description of Test: Attempt to start up the AutoTrader and provide a negative initial deposit. Then, attempt to provide a valid deposit and relogin and check if the user is still prompted for an initial deposit because their initial deposit should already be recorded.

Expected Results: The function should return an error message if the user attempts to deposit a negative value or skip the prompt with non-numeric input. Otherwise, it should update the database with the user input's amount of money entered into their account and not prompt them ever again.

Failure Criteria: If the function permits the user to enter negative/non-numeric values or continues asking them for initial deposits even after they have done so.

+initial_stock_selection(void): void

Description of Test: Enter a combination of valid and invalid stock tickers in quick succession as input to the function.

Expected Results: The function should return an error message if the user attempts to choose a stock that is not in the database, otherwise, the stock should be added to the user's portfolio.

Failure Criteria: if the stocks specified are in the database and not added to the user's portfolio OR the stocks are not in the database and added to the portfolio anyway.

+get_stock_price(String stock_symbol): int

Description of Test: Enter various stock tickers both in and not in the database and compare the values returned to the actual prices as stored in the database.

Success Criteria: The function also connects to the database, since stock prices are not constant, the function needs to fetch updated prices every time it's called. That's when alpha_vantage time series comes to action, it's set to work in 5 min intervals.

Failure Criteria: The function returns values inconsistent with those in the database or simply does not print error messages for input tickers not in the DB.

+print_ portfolio(): String[]

Description of Tests: This function prints the full portfolio, the number of companies in the portfolio, stocks currently owned and related information. Compare the printed output of this function with the actual portfolios, numbers of companies, and stocks currently owned of various users.

Success Criteria: The function properly prints out the portfolio that the user has with all information matching that stored in the database.

Failure Criteria: At least some of the information from the portfolio is not accurate with that of the database.

+purchase_shares_manually(): void

Description of Tests: Take in user input for various stock tickers that they would

like to purchase and check to see that they have the balance necessary to purchase the stock. If so, the number of stocks in their portfolio of this type will increase by as many as they bought and their balance will decrease by the amount of money they had to pay.
Success criteria/expected results: This function enables users to purchase on their own from stocks present in their portfolio, it receives the stock symbol of interest, the number of shares to purchase. The function notifies the user if the balance is not enough or if the input is not valid.
Failed Criteria: The portfolio doesn't update accordingly, fails to update balance

+sell_shares_manually(): void

Description of Test: Try entering different values as the user for the stock they would like to sell and the number of shares they would like to sell. The stock should already be in their portfolio for them to be able to sell it, while the number of shares they would like to sell must be greater than 0. Cover all corner cases with stocks not in the DB and negative/non-numerical values of shares.
Success criteria: The function enables users to sell on their own from stocks present in their portfolio, it receives a stock symbol of interest, number of shares to purchase. The function notifies the user if the balance is not enough of if the input is not valid.
Failed criteria: notifications are not delivered to user, selling results in incorrect balance values or doesn't update correctly.

Coverage: we conducted multiple test too observe the function behaves as expected with different amount of money and other edge cases such as negative values and when insufficient funds.

+print_balance():void

Description of Test: Call the function repeatedly for different users with different account statuses to see whether or not their balance is properly printed for display.
Success criteria: The function simply gets user account balance for display.
Failed criteria: display not found or old balance gets printed.
Coverage: we conducted multiple test too observe the function working and if it connects successfully to frontend.

+request_trading_duration(int days, int hours, int minutes):void

Success criteria:

The function prompt user to input time interval in days, hours, minutes ad runs the autotrade function accordingly.

Failure criteria: user puts invalid input, negative value, non-integer value. An error from some other function occurs and he function fails to get the data it needs to continue the trading interval.

Coverage: Outputs of the trading sessions were timed manually and checked whether changes happened and were passes to other functions.

+make_decision(): String decision

Success criteria: the function calls the correct transaction sell, buy or hold based of the results (prediction & sentiment).

Failure criteria: the function calls an incorrect transaction or calls multiple transactions at the same time.

Coverage: Output of auto trade and portfolio are used to trace back which transaction the make_decision function called and if it aligns with the results of the neural network

+auto_trade():void

Success criteria: the function receives sentiment and analyzed prices and feeds these values to the neural network for an overall prediction. This is where the neural network is used for the decision on whether to buy, sell or hold. This function also prints to the portfolio and updates account
Failed criteria: function fails to collect input, function runs on low balance, run doesn't execute.
Coverage: The trader was run multiple times and compare results of neural network. All punctualities of auto trade were tested with edge cases such as negative values and when insufficient funds.

+ withdraw_money(float value): void

Success Criteria: The function withdraws money successfully form the user balance and updates the portfolio.
Failure criteria: The function does not update balance, the functions withdraw money that doesn't exit in balance, the function does not reflect on the portfolio.

Coverage: we conducted multiple test too observe the function behaves as expected with different amount of money and other edge cases such as negative values and when insufficient funds.

+deposit_money(float value):void

Description of Test: Provide various float values of money that can be deposited in to a user's account. Make sure to cover negative and non-numeric values as error cases and for all other cases check balance to see that it has been appropriately updated in the database.

Success Criteria: The function returns error if the input is less than zero or is not numeric. Otherwise, the database for that particular user's balance has been incremented by the correct value as the user deposited.

Failure Criteria: Either the database's balance was decremented or altered in a way when the user's input was invalid or the database's balance was not altered at all or modified incorrectly when a valid input was provided.

+compare_price(float currentPrice, float predictedPrice): int

Description of Test: Provide various values of currentPrice and predictedPrice such

that there are test cases for when currentPrice is greater than predictedPrice, vice versa, and when they are equal.

Success Criteria: The function returns 2 when the currentPrice is greater than the predictedPrice and 1 for the opposite case (less than or equal to).

Failure Criteria: The function returns a value besides 2 when the currentPrice is greater than the predictedPrice and a value besides 1 when the currentPrice is less than or equal to the predictedPrice.

Integration Testing Strategy

We used horizontal integration testing to ensure our project works, starting from the smaller pieces of each subgroup and slowly moving to testing entire systems of each subgroups, and then the entire project from the front end, connecting the systems every subgroups together. Each group started with unit testing on their methods to make sure that they were receiving and producing data correctly before testing the communication of these methods with their controller and the database. Once these links were thoroughly tested and known to be working, we tested each of the overall systems separately, and then integrated them with the front end. Again, we tested the passing of information, this time between backend and frontend, and lastly we moved on to test the overall site functionality.

Neural Network Integration Test

After the structure of the neural network was decided on, preliminary testing was done to prepare for demo 1. Due to the nature of the neural network's function to forecast stock prices, a supervised testing methodology was used. Past stock prices were used as the 'test' as to whether or not the neural network was correct. We would take data from previous points in time to act as a proxy for a stock's current price, its forecasted prediction, and its sentiment. these values, along with the stocks actual price over our trading time interval were compiled into data sets that the network would use to train from. This first iteration used only back propagation to train, but this was expanded upon for the next demonstration.

These datasets were grouped up and ran through the network 1000s of times for each iteration. The neural network initially had randomly assigned weights, but using the process outlined above was able to manipulate these weights with the goal to find the lowest error margin between its prediction and the expected output. To prepare for the second demo, the network was retrained with larger datasets, as more data was available to us with the passage of time. The network also had the added improvement of emulating the evolution of a species by tweaking the weights in an effort to emulate DNA mutations that occur in nature. The network would be tested and iterations made until the error rate reached a cap in which it wouldn’t go any lower.

13. History of Work, Current Status, and Future Work

Breakdown of Responsibilities

We broke this project down into three parts: the autotrader, the sentiment analyzer, and the technical forecaster. Because we have nine people in our group, we broke evenly into three subgroups of three people, each taking care of one part of our project. The breakdown for developing and testing is as follows:

Autotrader Team: Parker Fisher, Asmaa Hasan, Jon Tsai
Classes: AutoTraderController (ATController.py), NeuralNetwork(NeuralNetwork.py), DBConnector (Database.py)

Sentiment Analyzer Team: Andrea Dumalagan, Amany Elgarf, Varun Ravichandran
Classes: SentimentController, WebCrawler, SentimentCalculator, SentimentPrediction

Technical Forecaster Team: Sunny Feng, Nicholas Heah, Manish Kewalramani
Classes: TechnicalController, MathPredictionModel, GrapherConnector

Classes for all to develop and test: DBConnector, Interface Page, Authenticator

After completing all of the testing for the classes, we plan to all work together on integrating our parts with the front end and the database, and then work on integration testing together to ensure that all bases are covered.

Contributions from Individual Team Members

1. Sunny, Nick, and Manish - There were issues with connecting our code written in Python/Java to the Google sites website. We worked on learning javascript and figuring out how to display the values on the website.

2. Sunny, Nick, and Manish - We were and still are trying to decide what language to write the code in. At first, Sunny used Java but then Nick realized that the ARIMA model would be best done in python, so all code was migrated there. We then realized that it would be hard to run python code on a website so we thought about writing the code in Java. We will continue to explore the best language for this mini-project.

3. Amany, Andrea, Varun - All of us had agreed to implement our mini-project in Python. Amany and Varun took on the responsibility of deciding the structure of the sentiment analyzer--regarding which libraries and algorithm to utilize and implement to accurately calculate a weighted prediction per stock ticker. Some issues regarding this area of responsibility was choosing an algorithm, whether to go along with SpaCy--a neural network based method--or Vader; other issues include accounting for words in a headline that are not located in the library, and whether or not this will affect a stock's true neutrality score. Andrea's

responsibility was creating a webcrawler utilizing an API--NewsAPI--that retrieves headlines according to the stock ticker chosen by the user.

4. Amany, Andrea, Varun - Our next plan of action is to connect the webcrawler to the sentiment analyzer and train the data in order to reach a goal of 50% prediction accuracy. Additionally, moving forward, our goal on top of training the data is to display the sentiment calculations to the websites alongside with the headlines associated with the predicted score.

5. Parker, Asmaa, and Jonathan - A majority of the issues experienced came from implementing a workable AI that would be able to conclude an output given different inputs. Asmaa was able to implement a rudimentary AI while Parker and Jon researched training algorithms. Some of the issues that came about while coming up with a neural network structure was finding a balance between simplicity and accuracy, as we want it to perform as efficiently as possible. It was concluded that the FF network was the most optimal for our purposes so after that was decided we were able to implement a prototype version of the AI

6. Parker, Asmaa, and Jonathan - Next we plan to further flesh out the AI so that it can use the datatypes we need for this project and train it using the methods described above. The training methods have been solidified and have been implemented to be able to be utilized once the AI has been fully completed.

Project Coordination and Progress Report
Because all of the use cases require a fully developed front end, we have not fully implemented them. However, we have all made progress on the back end, successfully writing much of the required code to run our algorithms, mathematical models, and analyses that we will pass on to the front end UI. As of Demo 1, we have successfully deployed our database so that all member may access it, so our current focus is ensuring that our code can successfully communicate with the database to receive data and update values for the correct operation and functionality of our project.

To ensure that everyone is making progress, we have held one subgroup meeting every week to work on our classes and code, as well as one full group meeting weekly to update each other on our progress, decide on goals to meet in the near future, and work on the weekly deliverables. We have shared all of our resources on Google Drive and Github, and use a group chat to make sure everyone gets help and stays up to date, and that all important messages can be noticed and handled, as soon as possible if necessary.

History of Work

Technical Forecaster

Gantt Chart for Technical Forecaster (PDF):
https://drive.google.com/file/d/1zsKbuDl7krHPcCs4jWwu4ShfO8QeX7nx/view?usp=sharing

After finishing up the core of our mathematical models, namely the ARIMAModel, ASIModel, and ROCModel, the Technical Forecaster team decided to include a Fast Fourier Transform Model into their timeline. Its analysis would provide further accuracy for our prediction and a longer term projection for a particular stock's price. This meant that the ModelAverager had to be on hold until the parameters for the Fourier analysis could be fine tuned. After writing initial code for the ModelAverager, they went to test its accuracy, so some groups members worked on improving and modifying the averaging algorithm, while others moved on to work on drawing a graph that was accurately labelled and provided relevant information for a user.

With these parts wrapped up, after further discussion with the entire group, the Technical Forecaster dropped its development of the database updater and the SendToTrader functions, as they were deemed vestigial and no longer needed. The rest of the time was dedicated to forming the connection from the backend to the front end in order to update the website with the calculations and graphs generated by the Technical Forecaster. This took the majority of the time after the first demonstration.

Key accomplishments to note by Technical Forecaster:

- Created a price forecaster that is accurate within 1% in a five minute interval
- Created a price forecaster that is accurate within 5% in a three hour span of time

- Successfully generating a graph that is labelled with stock price and intervalled time periods and a color-coded indicator for our predicted stock price

Sentiment Analyzer

Gantt Chart for Sentiment Analyzer (PDF):
https://drive.google.com/file/d/1gvc4-NZOu_StWRKf0eKCAAHp-BFAvZbX/view?usp=sharing

Compared to our previous progress chart, Sentiment Analyzer had to revisit *Research* on more than one occasion during the duration of this course, and the reason lies in why *TrainingData* and *SentimentNeuralNetwork* are of null duration and 0% completion.

Initially, Sentiment Analyzer was going to use an API called spaCy--a free, open-source library for advanced Natural Language Processing (NLP) in Python-- in order to calculate sentiment scores of headlines respective to a stock ticker. In order to correctly utilize spaCy, we needed not only training data but also evaluation data; and, for KA$H, Sentiment Analyzer would need to train a model from scratch, our group would need to provide at least a few hundred examples for both training and evaluation. Considering resources and the time constraint, utilizing spaCy would not have been feasible. Instead, Sentiment Analyzer opted for utilizing Vader to calculate sentiment scores grabbed by News API in real-time; this change was reflected in Demo 1 and Demo 2.

Key accomplishments to note by Sentiment Analyzer:
- Utilizing News API to grab headlines of a user-specified stock in real-time; search headlines up to 30 days old from the time search function is called.
- Calculating sentiment scores in real-time of a user-specified stock and correlating the score to a stock price trend prediction.
- Displaying top 5 relevant headlines to a user-specified stock.
- Displaying the stock price trend prediction within a month's time on a graph.

Automated Trader

Gantt Chart for Automated Trader (PDF):
https://drive.google.com/file/d/1UPCC_tl7lFeqSTqqrTz-7UUpTb5oGQRl/view?usp=sharing

Based on our Gantt Chart plans, as shown above, we had split our work into several parts based on the various models that needed to be implemented and integrated along with the other portions of the application. The first priority was to complete the TensorFlowModel as that was only 25% complete and formed the basis of the predictive model that would be used to make trades. We spent the bulk of the last phase of the project working on the code for the TensorFlowModel and then testing its performance against actual data to determine how accurate the AutoTrader was.

Before we could actually implement the TensorFlowModel, however, we had to work on the Training Data that the model utilized to prepare itself for the actual data that would be fed into it as soon as the Sentiment Analyzer and Technical Forecaster teams had completed working on their code and had numbers updated into the database. This part of the project was entirely completed prior to the first demo.

Once we had generated a training dataset and had appropriately trained the TensorFlow model on that data, we began working on the actual functionality of the AutoTrader. This entailed the major functions that it can offer the user: creating a portfolio of stocks, creating accounts, logging in, depositing money, withdrawing money, and maintaining a profile. Initially, we had also hoped to separate the stocks by sector to provide additional options for users, but we were forced to back down on that goal due to a lack of time. This phase took up most of the latter half of March and early April and involved the brunt of the project itself.

The weeks that preceded the second demo were dedicated to integrating the AutoTrader with the other two parts of the project to produce one cohesive product. Those goals are described via the UserTechnicalForecaster, UserSentimentAnalyzer, WebsiteUpdate, and DatabaseUpdate goals. At this stage, we focused on running the TensorFlowModel and the various AutoTrader functions with actual data generated by the Sentiment Analyzer and the Technical Forecasters rather than the sample data that we had been relying on up to that point. In addition, the first demo had been a showcase of the backend capabilities of the Auto Trader, so we focused on developing the frontend of the website so that it could have its own page and interface via which users can actually utilize the product.

Ultimately, all of these goals were completed just in time for the second demo, leaving us with just the final goal. Error calculation was our way of testing the trader so that we could see exactly how well the trader was performing relative to the market and so we could have a baseline for our future work. By carrying out these goals in a systematic format, as seen in the Gantt Chart, the AutoTrader grew from a mere machine learning concept to an actual product with most of the features that we had set out to implement.

Key Accomplishments to Note by Automated Trader

- Creating a user interface comprised of login pages, account creation, and personalized profiles for users to manage their portfolios and money
- Creating a training dataset for the neural network to train on prior to the arrival of actual data from the completed Sentiment Analyzer and Technical Forecasters and then training the neural network on that data
- Developing a database schema to store user account information, hashed password information, portfolios, stock prices, and calculated sentiment data for quick and easy update and look-up while running the trader

Current Status and Future Work

As of now, we have all of our code working and connected to the front end, and we have deployed our software onto a website. Future goals for the technical forecaster include improving upon our prediction algorithm to provide even better accuracy and long term predictions over for spans of several months. For the sentiment analyzer, we would like to gather data and introduce a neural network into our analyzer for a sentiment prediction that is more curtailed for our project. The Autotrader team would like to further develop their current neural network to make a better trading decisions, so that users will feel more confident with putting their money into the autotrader. Overall, we would like to give users more options to customize what they want to see in regards to the headlines displayed, as well as putting more work into developing a more streamlined website that would allow features such as comparing a stock’s price to its sentiment score over a time period. Lastly, we would like to look into making the site mobile friendly, or develop an app that runs alongside our current product.

14. References

Sentiment Analyzer References

1. Comparison of Top 6 Python NLP Libraries
   https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb

2. TextBlob GitHub
   https://github.com/sloria/TextBlob

3. SpaCy Documentation
   https://spacy.io/models/en

4. Algorithmic Trading using Sentiment Analysis on News Articles

https://towardsdatascience.com/https-towardsdatascience-com-algorithmic-trading-using-sentiment-analysis-on-news-articles-83db77966704

5. Master Dictionary of financial related words and concepts

   https://sraf.nd.edu/textual-analysis/resources/#Master%20Dictionary

6. Apify
   https://www.apify.com/docs

7. Scrapy
   http://docs.scrapy.org/en/latest/
8. News API
   https://newsapi.org/docs

Automated Trader References

9. A simple deep learning model for stock price prediction using TensorFlow
   https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877

10. Evolving Neural Networks through Augmenting Topologies
   http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

11. Gradient Descent versus neurovolution
   https://towardsdatascience.com/gradient-descent-vs-neuroevolution-f907dace010f

12. An evolutionary algorithm for feed-forward neural networks optimization
   https://ieeexplore.ieee.org/document/7060901

Technical Stock Forecaster References

13. Autoregressive integrated moving average
   https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

14. Accumulative swing index
https://library.tradingtechnologies.com/trade/chrt-ti-accumulative-swing-index.html

15. Stochastic oscillator
https://en.wikipedia.org/wiki/Stochastic_oscillator

16. Stochastic oscillator definition
https://www.investopedia.com/terms/s/stochasticoscillator.asp

17. Price rate of change indicator - ROC definition and uses
https://www.investopedia.com/terms/p/pricerateofchange.asp.

18. Fourier Analysis
https://web.wpi.edu/Pubs/E-project/Available/E-project-122214-115000/unrestricted/AndroMQP.pdf

19. High-Frequency Trading
https://en.wikipedia.org/wiki/High-frequency_trading