NORTHWEST NAZARENE UNIVERSITY

Stock Screener

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Kyle Lee Duncan
2021

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Kyle Lee Duncan
2021

Stock Screener

Author: Kyle Duncan
Kyle Duncan

Approved: Kevin S McCarty
Kevin McCarty, Ph. D., Assistant Professor of Computer Science,
Department of Mathematics and Computer Science, Faculty Advisor

Approved:
Jennifer DePew.
Second Reader

Approved: Barry Myers
Barry L. Myers, Ph.D., Chair,
Department of Mathematics & Computer Science

ABSTRACT

Creating a Stock Screener Application Utilizing Machine Learning for Ease of Scanning the Stock Market.

DUNCAN, KYLE (Department of Mathematics and Computer Science), MCCARTY, DR. KEVIN (Department of Mathematics and Computer Science).

The stock market is rapidly changing, most stock screeners or services allow for visualization of stock tickers using charts. These charts are used by stock traders to analyze tickers they are interested in; the issue is that a human can only adequately track a few charts at a time. The purpose of this application is to address this issue. Specifically, this project utilizes machine learning to aid in increasing a stock trader's ability to analyze the stock market. The machine learning model used in the application was trained using a random forest classifier that utilizes historical data. Using an application programming interface from Polygon, the stock screener receives real-time stock quotes which are then prepared and classified by the probability of increasing or decreasing in value. The classified data is then displayed on the user interface in a heatmap, and updates as soon as new data is classified. Displaying the data as a heatmap allows for an intuitive way to quickly analyze many stock tickers. The result of this application is a real-time stock screener tool that can be used to quickly visualize tickers and the probability of each ticker's movement.

Acknowledgments

First, I would like to thank my girlfriend, Anna, my brother, Brandon, and my mother for constantly listening to me try to talk out any problem or ideas I had regarding the development of the Stock Screener throughout the summer. I would also like to thank my grandparents and the DePew family for allowing me to work remotely on this project from their homes when my home Internet when was not working and there was nowhere else to go due to the Covid-19 pandemic. I would like to thank my professors, Dr. McCarty, and Dr. Hamilton. Without Dr. McCarty the stock screener would not exist as he provided all the necessary data to make the application possible and answered any questions I had very quickly. Dr. Hamilton was always available when I had any questions and was eager to give helpful feedback. Lastly, I would like to thank Jeffrey Fairbanks for helping me throughout with the development of the Stock Screener throughout the summer.

Table of Contents

List of Figures

Introduction

The primary goal of this project was to create a system within an application that enables users to easily view and analyze stock data utilizing the implementation of machine learning. The purpose of developing this system was to utilize a data pipeline to streamline the training process for the random forest machine learning model used in the application. The secondary goal was to obtain and display stock data as close to real-time as possible. This means that the application needed to be able to predict and display these predictions asynchronously to the user interface as quickly as possible to approach this goal and to minimize the amount of time the user can interact with the user interface. Overall, the objective was to create the application with these goals in mind and to build out a working application that serves as a prototype that can be expanded upon in the future.

Background

This project was part of a summer research team at Northwest Nazarene University (NNU) working on utilizing computer science with data from the stock market, whose goal was to explore the use of machine learning with this stock data in the form of a Widows based desktop application similar to a traditional stock screener. A stock screener is a tool used by traders and investors to separate stocks depending on the user-defined metrics or view stock data (Vineeth).

To effectively develop machine learning models, we needed a lot of data for a model to be reliable and realistic, as the stock market contains a massive amount of data. Thankfully, Dr. McCarty had provided a large database of stock data consisting of minute, hourly, daily, weekly, monthly, quarterly, and yearly stock quotes for over the last twenty years. The database was kept up to date by Dr. McCarty throughout development, ensuring that we always had the latest data

available to us.

Dr. McCarty also provided us with an API (application programming interface) called Polygon to obtain current stock data to use in our application. Polygon is a service that provides snapshots of the stock market in HTTP responses after sending an HTTP request to the API using proper credentials. A snapshot is a collection of a large number of stocks and their current status, including attributes such as the stock's open, high, low, close, and volume at the time in which the request was sent.

The Stock Screener application utilizes both the Polygon API and database containing a history of previous stock quotes, both provided by Dr. McCarty. These provided resources served as the source of the Stock Screener's data. It is important to note that the United States stock market operates from Monday through Friday from 9:30am to 4:00pm eastern standard time.

Exploration

When starting the development of the project, Dr. McCarty mentioned that he wanted the application to be a Windows based desktop application created using the C# programming language as he had already worked on his own small prototype using C#. There were several frameworks to choose from when deciding how to build a Windows desktop app. The three frameworks we narrowed our options to was Windows Forms (WinForms), Windows Presentation Foundation (WPF), and Universal Windows Platform (UWP). There are advantages and disadvantages to each option, but after looking into each framework, we ultimately decided that it would be best to use WinForms. Dr. McCarty's prototype was already made in WinForms and he was familiar with the framework which made the development of the app in WinForms much easier than the development would have been in the other two frameworks.

The next major decision we had to make was which language we wanted to use for machine learning in the application. At the beginning stages of development Microsoft had just released a library for machine learning in .NET (C#) called ML.NET which we tested but decided not to use for our project due to the lack of documentation at the time we started the project. The next language we investigated implementing was the R programming language. The problem with R was how long it took to create and export models when we started using the large datasets necessary for the application. When using R, the app would take a full day to train and export models after making as many optimizations as we could, proving to be inefficient. Finally, we decided to explore using the Python programming language and the SciKit-Learn library. Python quickly proved to be a much more efficient option, building models and exporting them in only a few hours at first. The result yielded the entire system of obtaining the data from Dr. McCarty's database, creating the model, and serializing and exporting the model, taking less than an hour. Overall, Python proved to be the best fit for the application at the time that we began developing the machine learning system for the application.

Design

The main purpose of the Stock Screener was to focus on the backend systems that drive the application. Therefore, we decided to keep the user interface as simple as possible while still including all the necessary features supported by the backend. To keep the application simple, we decided to use a single form for the entirety of the user interface. Initially, we tried to separate components into multiple forms, however, this approach was unnecessary and resulted in performance issues. Due to the application needing to be able to handle working at near real-time the single form proved to be the best option for the performance requirements necessary.

Figure 1 – Startup State of Form

The initial state of the form on startup was designed to be in a "default" state, meaning that no options are selected on startup, allowing the user to select what they desire. The default state of the Stock Screener would run with real-time stock quotes and display them to the user in the live feed table. The purpose of designing the initial state of the form this way was to allow users that have used a stock screener before to open the application to a more familiar, standard stock screener, displaying a snapshot of the stock market every time the API is called.

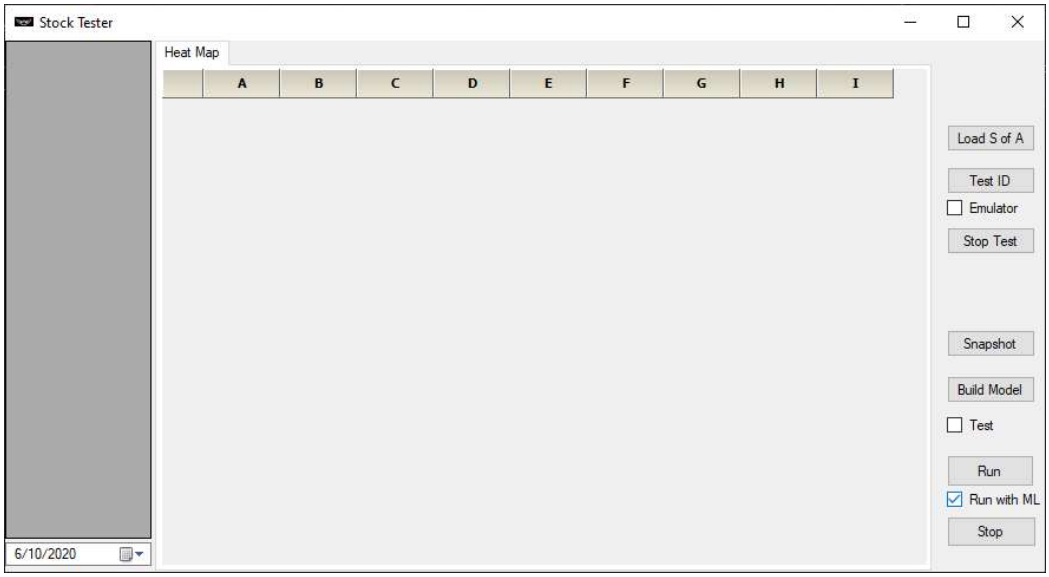
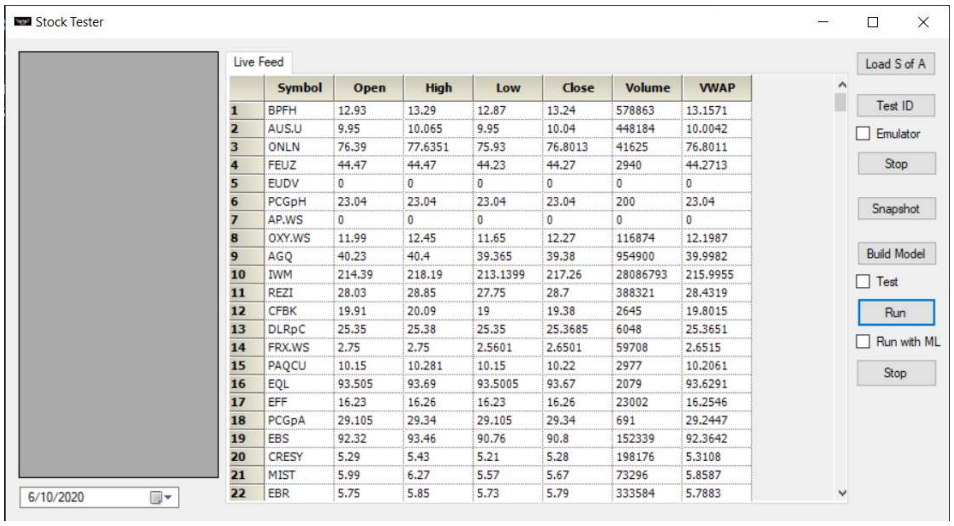


Figure 2 – Machine Learning Form State

The alternate state from the default state of the form is the machine learning state. The machine learning state is activated when the user checks the “Run with ML” checkbox on the user interface. The purpose of the machine learning state is to allow the user to run predictions on snapshots of the stock market and view the resulting predictions in a form of a heatmap.

Implementation

Being that the main goal of the project was to explore the use of machine learning with stock data, implementing a way to obtain the data was our first priority. To obtain real-time stock data we decided to use the Polygon API, which allowed us to capture data from almost 9000 different stocks every time an HTTP request was sent to the API. The API would send back a JSON (JavaScript Object Notation) response that we then deserialized into a list of objects and displayed to the live feed table.



	Symbol	Open	High	Low	Close	Volume	VWAP
1	BPFH	12.93	13.29	12.87	13.24	578863	13.1571
2	AUS.U	9.95	10.065	9.95	10.04	448184	10.0042
3	ONLN	76.39	77.6351	75.93	76.8013	41625	76.8011
4	FEUZ	44.47	44.47	44.23	44.27	2940	44.2713
5	EUDV	0	0	0	0	0	0
6	PCGpH	23.04	23.04	23.04	23.04	200	23.04
7	AP.WS	0	0	0	0	0	0
8	OXY.WS	11.99	12.45	11.65	12.27	116874	12.1987
9	AGQ	40.23	40.4	39.365	39.38	954900	39.9982
10	IWM	214.39	218.19	213.1399	217.26	28086793	215.9955
11	REZI	28.03	28.85	27.75	28.7	388321	28.4319
12	CFBK	19.91	20.09	19	19.38	2645	19.8015
13	DLRpC	25.35	25.38	25.35	25.3685	6048	25.3651
14	FRX.WS	2.75	2.75	2.5601	2.6501	59708	2.6515
15	PAQCU	10.15	10.281	10.15	10.22	2977	10.2061
16	EQL	93.505	93.69	93.5005	93.67	2079	93.6291
17	EFF	16.23	16.26	16.23	16.26	23002	16.2546
18	PCGpA	29.105	29.34	29.105	29.34	691	29.2447
19	EBS	92.32	93.46	90.76	90.8	152339	92.3642
20	CRESY	5.29	5.43	5.21	5.28	198176	5.3108
21	MIST	5.99	6.27	5.57	5.67	73296	5.8587
22	EBR	5.75	5.85	5.73	5.79	333584	5.7883

Figure 3 – Live Feed Display

After successfully implementing a system to obtain and display stock data, we moved on to implementing machine learning into the application. To start we needed a way to obtain data to build our model. After discussing with Dr. McCarty, we decided it would be best to use only a sample of the database minute data, using 4% of the previous month’s minute data, to avoid

overtraining the model. While 4% may not sound like much, 4% of the last month's minute data is over 1.6 million rows of data. Before sampling 4% of the data, we used a query on the previous month's data that returned discrete values and a label for us to be able to use the data in a model. Our label we selected was a Boolean value indicating whether a stock increased or decreased in value in the next minute. The other attributes we used were if the stock's volume and value increased or decreased in value from the previous minute, as well as if the volume of the stock is currently above the stock's 50-day average volume. This was in no means meant to be a viable trading strategy, but instead to serve as a starting point for machine learning with stock data as determined by Dr. McCarty.

Results Messages

	Symbol	QuoteUTCDate	UpOrDownMinute	UpOrDownVolume	PrevMinClose	NextMinClose	VolumeCompare	UpOrDownNextMinute
1	ARR	2020-09-14 19:13:00.0000000	1	0	9.8950	9.9200	0	1
2	ARR	2020-09-14 19:17:00.0000000	1	0	9.9000	9.8900	0	0
3	ARR	2020-09-14 19:21:00.0000000	0	0	9.8801	9.8800	0	0
4	ARR	2020-09-14 19:25:00.0000000	1	1	9.8800	9.8900	1	1
5	ARR	2020-09-14 19:30:00.0000000	0	1	9.9050	9.8950	1	0
6	ARR	2020-09-14 19:34:00.0000000	1	1	9.8891	9.9000	1	0
7	ARR	2020-09-14 19:38:00.0000000	0	1	9.9050	9.8950	1	0
8	ARR	2020-09-14 19:42:00.0000000	1	1	9.8950	9.9000	0	0
9	ARR	2020-09-14 19:46:00.0000000	1	0	9.9050	9.9100	0	0
10	ARR	2020-09-14 19:50:00.0000000	1	0	9.9198	9.9150	0	0
11	ARR	2020-09-14 19:54:00.0000000	0	0	9.9150	9.9150	0	1
12	ARR	2020-09-14 19:58:00.0000000	0	1	9.9150	9.9000	1	0
13	ARR	2020-09-15 13:25:00.0000000	1	1	9.9300	9.9500	0	0
14	ARR	2020-09-15 13:32:00.0000000	1	0	9.9388	9.9428	0	0
15	ARR	2020-09-15 13:36:00.0000000	1	1	9.9511	9.9500	1	0

Query executed successfully. WIN-QE19868HQ0M (15.0 RTM) Stock Analysis Users (58) StockAnalysis 00:00:13 1,625,228 rows

Figure 4 – Model Building Query Results

After obtaining the data to use with our machine learning model, we needed to implement a Python script to train a model using this data. We chose to write a Python scripts so that the script could be executed from within the C# application by running the script in an Anaconda environment. To train a model, we used Sci-Kit Learn's random forest classifier. A random forest classifier is an ensemble that contains multiple classifiers, in this case decision tree classifiers, and is generated using a random selection of attributes at each node to determine the split. During classification, each tree votes and the most popular class is returned (Russell & Norvig). After creating a model, at the end of the script the model was serialized and exported as

a SAV file.

The final piece of the machine learning system was to implement a script that loads the saved serialized model, deserializes it, and then can be used to make predictions with unlabeled stock data. The stock data received by the script comes from the main application. As discussed above, the main application uses HTTP request to receive a snapshot of the stock market containing data for almost 9000 different stocks. This data is then prepared for use with our model using a function that evaluates the snapshot, using the same metrics as the SQL query, and outputs a CSV (comma separated values) file in the correct format for the prediction script to ingest and run predictions. When the prediction script makes predictions, the predictions consist of predicting the probability that a stock will increase in value in the next minute. The output of the prediction script is a CSV file containing tuples. These tuples consist of the stock symbol, increase probability, and decrease probability. This CSV file is then consumed by the main application, which converts the predictions into a list of objects.

After implementing the prediction script, it was time to implement a way to display the predictions to the user interface. Utilizing a table with two tabs, a tab for the stocks with the highest probability of increasing in value and another for the stocks with the highest probability of decreasing in value, the predictions were displayed to the user interface as a heatmap. Table cells indicating the probability were then colored using a gradient to represent a simple heatmap visual. The green gradient ranging from light green to dark green was used in the increasing probability table, where the darker the green the higher the probability of a stock increasing in value; the same concept was implemented for the decrease table using a red gradient.

While this system was implemented using multiple programming languages, our final choice, Python, proved to be much more efficient. The previous iterations of the machine

learning system were slow, which is why we chose to use Python. The initial implementation of the system took only three hours to train a model, a significant improvement and one that we deemed acceptable, as the models only needed to be trained once a week and could be run during the weekend when the stock market was inactive. However, with some modifications, we were able to achieve a training time from anywhere between 30-40 minutes, a massive improvement from the initial implementation. Using this model to make predictions, the results were displayed on the frontend as a heatmap in which the 100 stocks with the highest probability of increasing in value were displayed and the 100 stocks with the lowest probability of increasing in value were displayed.


Heat Map

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	CKPT	PFGC	CANF	STPK	HLIO	SBS	EXPR	RADA	OEC	XLB	KVLE	AFGE	GGTpG	CAH	THBRW	TRX
2	MTB	PNC	CHIL	DVA	ORGO	HWKN	NYCB	ZEUS	CVGI	COW	IRS	DSKE	SIGIP	RALS	CFIVW	SELB
3	ATOS	ADT	TUSK	ALGS	FSMO	PRFX	FLXS	MBAC.U	IPOE.U	PG	KO	LANDO	CDLX	RL	ICCC	VCNX
4	TZOO	ETN	EYLD	CAE	AXGN	ROLL	DBD	KRTX	URG	AIN	MIY	AIT	BENEW	GFX	MBRX	SRTS
5	CVV	HGSH	SEE	GUTpA	HUIZ	IYZ	RCHG	COVAW	HPQ	HDSN	MVP	VYNT	ATNX	MTEM	GMBL	ATHX
6	MDLX	REVG	CREG	ICHR	VKTX	ACND.U	ODP	BKSE	SPCX	GABpH	UBFO	VVPR	IIN	GDYN	ZYNE	EEIQ
7	PDFS	QLI	TMP	PLYMpA	FRX	VMI	RUTH	PYN	BA	JDST	FTXN	CNEY	LYFT	SNOA	WPF.WS	AXL
8	ENRpA	RFpC	VGAC	NVS	BIPC	GCACU	MAGA	PRIFpB	FTI	CEI	VIXM	DCTH	VPCBU	CMLS	PDEX	FLACW
9	NWL	TNPpD	JOFFW	MOHO	CTBI	ALK	XFLTpA	FDRR	CHPT	DS	MYOV	TDW.WS.A	AVIR	SNGXW	BDRY	BJRI
10	WSTG	NRC	CRNX	PFFL	GLAQU	LAWS	SCVX	EDAP	KZIA	QFIN	KNSA	REPL	CLW	GILT	VTGN	BPTH
11	SBGI	RFP	LCAP	KKRpC	MNOV	AGC	SHBI	GRSVU	XOP	SRNGU	PFMT	CNBS	JAKK	WLFC	APVO	TOUR
12	GDEN	LEGOW	MCR	BLTSU	WMT	VGAC.WS	NGACW	SSPK	LOOP	AESE	NAT	USWS	RIDE	CTXRW	PVAC	SPGS.WS
13	BANF	CRSAW	ETON	PLTM	DFNS.WS	PHX	TWND.WS	CHEF	SRGA	SLCA	STRR	ANVS	CNSP	RZLT	NNOX	ASC
14	GLDD	LBAY	VPV	APDN	MAXN	LODE	RCON	RKDA	FUBO	SMID	ACIC.WS	LRMR	USEG	PVH	PXI	UA
15	PBSM	ZION	MHpD	PRU	CLDR	ASPN	VMACW	BPRN	OII	BLMN	SPI	EM	SJI	FDX	OLB	PCYG
16	DLN	APSG.WS	DGNU	AHPI	EWBC	FRAK	CRT	SGU	TRQ	CFRX	INMB	CWEB	ENSG	OMER	REI	EMPW.WS

Figure 5 – Heatmap

Now that all the core components were implemented, there remained one major issue. The user interface was unresponsive when running either machine learning or displaying data to the user interface. To fix this, the current system needed to be implemented to use asynchronous programming. With the use of asynchronous programming, the user can still interact with the application while the user interface is being updated. This was one of the goals of the project, allowing the user to interact with the data with minimal to no interference.

When receiving a snapshot, the deserialized data is now displayed by triggering an event

associated to a delegate. A delegate is a type that represents references to methods with a particular parameter list and return type (Wagner). Delegates were used to display the live feed and machine learning predictions to the user interface and automatically continuously update these tables. By changing the application to be asynchronous, this also allowed the user to now stop the application whenever they desire, whereas before the user would have to wait for the user interface to update or the Python script to finish running.

Future Work

There are many directions that can be taken to improve this project in the future. The first step that I would recommend would be to improve the user interface. Since the initial goal of the project was to make a Windows application, I think it would be beneficial to convert the project to the WPF framework. Another option for the user interface would be to completely start over and develop the Stock Screener as a web-based application. Below is a prototype of a web-based application I worked on using React Javascript.

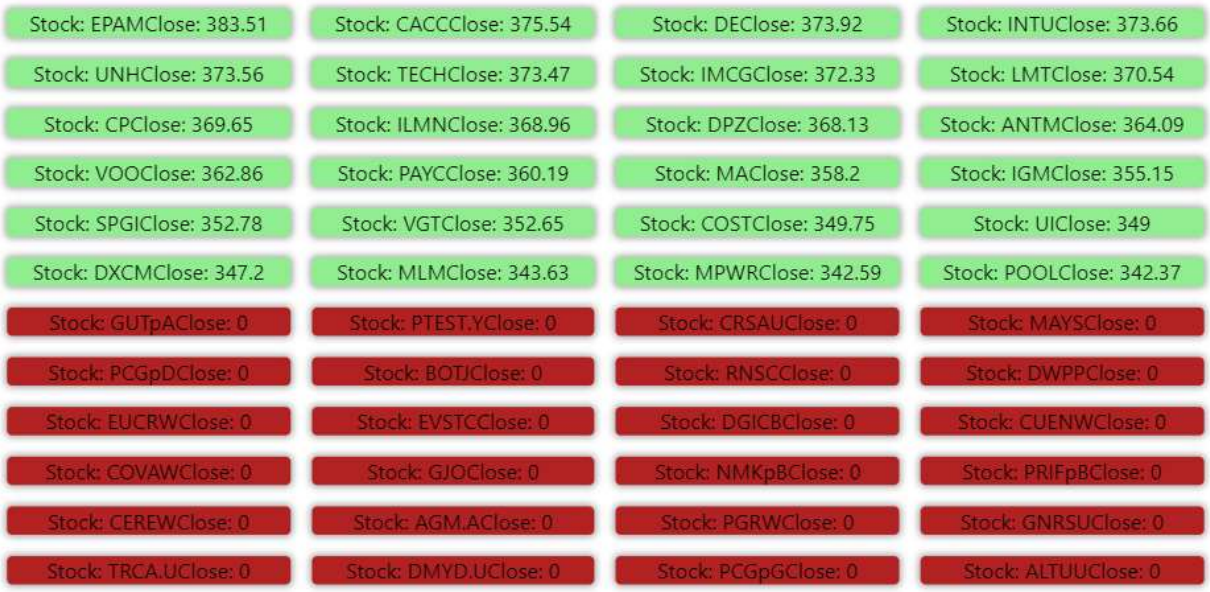


Figure 6 – Web App Prototype: Heatmap

Besides the many possibilities for the Stock Screener's user interface improvements, an

area that I would like to make further improvements to is machine learning. By using more complex trading strategies and incorporating techniques such as testing the statistical significance of the predictions, a more robust model could be trained and utilized. A final suggestion I have for future work would be to improve the heatmap and live feed tables.

Conclusion

Overall, this project was fun and very difficult. When I started this project, I had no experience with the C# programming language, WinForms, and the R programming language, and minimal experience with Python. While I was not new to the concepts of machine learning or artificial intelligence upon starting the project, I had very little knowledge about the stock market, let alone applying machine learning to stock data. This presented many challenges throughout the development of the Stock Screener application. Learning these languages and learning about the stock market was time consuming and frustrating at times. The scope of this project changed a lot throughout development. The original plan for the project was to use reinforcement learning to create a bot to trade stocks; however, given that there was no existing application to even acquire stock data or a system to train and utilize machine learning or artificial intelligence, the scope of the project quickly shifted to accomplish that issue. There were many times where I went down paths that led to dead ends and slowed development but aided in the learning process.

Many of the courses I have taken at Northwest Nazarene University were applied to this project, from machine learning to operating systems, databases, and computer networking. All these courses aided in my understanding of different components of this project as I moved along. Understanding how an API works, connecting and querying a remote database, and threading were some of the most important concepts I learned and implemented into this project.

Dr. McCarty was very helpful throughout the development of the Stock Screener. I am thankful for his help and for providing me with this project as I learned many useful things from developing an application. Being able to work with a team, meet deadlines, and attend meetings was a great experience that has carried over into my current internship, as have the skills I learned while developing this application.

While the Stock Screener is far from complete, it serves as a starting point for anyone to pick up and continue. There are many possibilities for further modifications or development of the Stock Screener, which is what motivated me and excited me through the entire development of the application. I am excited to see what future students can accomplish if they choose to expand on this project.

References

Russell, S. and Norvig, P. (2010). Artificial intelligence: A modern approach. Prentice Hall, Upper Saddle River, NJ.

SciKit (2021). Syncfusion Essential Windows Forms Documentation. Retrieved March 31, 2021, from https://help.syncfusion.com/windowsforms/overview.

Syncfusion (2021). Syncfusion Essential Windows Forms Documentation. Retrieved March 31, 2021, from https://help.syncfusion.com/windowsforms/overview.

Vineeth. (2021). Stock Screener. Retrieved March 31, 2021, from https://cleartax.in/g/terms/stock-screener#:~:text=Stock%20screeners%20are%20a%20tool,popular%20trading%20platforms%20and%20websites.

Wagner, B. (2021). C# documentation. Retrieved March 31, 2021, from https://docs.microsoft.com/en-us/dotnet/csharp/.