NORTHWEST NAZARENE UNIVERSITY

Detecting Stock Market Patterns via Standard Query Language Data Analytics.

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

Braelyn Blessing Boerner
2021

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

Braelyn Blessing Boerner
2021

Detecting Stock Market Patterns via Standard Query Language Data Analytics Among a Stock Screener Application.

Author: Braelyn B. Boerner
Braelyn Blessing Boerner

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

Approved: [illegible]
Audra Butkus, College of Natural and Applied Sciences, Second Reader

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

ABSTRACT

Detecting Stock Market Patterns via Standard Query Language Data Analytics Among a Stock Screener Application.

BOERNER, BLESSING BRAELYN, DR. MCCARTY, KEVIN (Department of Mathematics and Computer Science)

Predictive analysis within the stock market has been a goal of many different banks and large organizations as well as individual traders, as there are substantial monetary gains to be had. The objective of this project is to ask the question: Can one use patterns developed within the stock market to predict the behavior and achieve positive financial margins? When beginning this research endeavor, learning the current patterns for trading algorithms is necessary. Traders often use technical analysis to predict future stocks moves. With this information and a sample database of around nineteen years of stock data, this hypothesis was tested on the Golden and Death Cross. Using specialized SQL queries these patterns were investigated through a series of tables and extensively explored to provide relevant data needed to achieve a prediction method for the trading algorithm. The results demonstrated that the opposite hypothesis, buying on the Death Cross and selling on the Golden Cross, occurred when employing these patterns implemented by this approach.

ACKNOWLEDGMENTS

This project was completed with the support of multiple individuals. I would like to thank the Computer Science Faculty for funding the research of this application and making all this possible. Thank you to Dr. McCarty for being the project creator, manager, and most of all my greatest mentor. I would like to thank my fellow lab administrators Andrew Welk and Ender Sandiage for the Systems Administration support. A special thanks to Tyler Shea for the assistance in the development and curation of the Golden Cross and Death cross queries. Thank you to Dr. Myers for being my academic advisor and guiding me through the advancement of my senior research paper.

Table of Contents

Table of Figures

PROJECT BACKGROUND

When walking around a grocery store there is a reason behind the placement of its products. For example, milk, one of the most bought items, is in the back of the store. The reason for this is to exhort the customer to walk around the entirety of the store while browsing through other products that they will most likely end up buying. The placement of the products is intentionally designed based on human tendencies. Human patterns are seen everywhere, the more you look the more you will find. Behavioral tendencies manifest in smaller time frames, recorded by predictable patterns. These designs are useful for a plethora of organizations other than grocery markets. For example, social media detects a human's interaction with the platform and uses predictive analytics for better intercommunication. For this project specifically, it is useful for the stock market. Within the stock market two common patterns are the Golden and Death Cross. These are the two this project will test and focus on specifically.

PROJECT OBJECTIVE

In the stock market, money is a measure of success or failure. Traders try to increase their odds of success using patterns. If the patterns were easy to track, the likelihood of success would be high, however, competition among traders tends to make patterns more difficult to track. This competition usually involves "front-running," which is a technique where a trader tries to adapt to a pattern by invoking a particular strategy just in front of other traders. An interesting thing to note is the different types of traders in the stock market. Retail traders more commonly abide by front-running strategies, whereas hedge funds look at stocks differently as they have policies about which stocks, they can invest in. This creates margins in the stock market and shows the variation of differing trading behaviors. The overarching goal of this project is to examine a pair of common patterns and test them. To complete this goal, we began by contextualizing data. Contextualizing data is viewing outcomes and seeing whether they are the result of an underlying human emotion. For instance, individuals may see price movement and be more inclined to buy or sell simply because the "chart" looks good or bad, which is often an emotional response. Data analyzing is a difficult concept especially when it comes to the stock market. Algorithms to detect patterns in the stock market have been around for years, however, due to its complexity, there is always something to further to discover. The stock market contains massive amounts of data. This project's aim is to sort through the noise and find something of value. The first step to get there is evaluating a common pair of existing patterns.

The first trend to discuss is the Golden Cross pattern. The Golden Cross is a bullish, or positive pattern where a short-term trendline crosses above a long-term trendline. It is a specialized chart pattern that is widely believed to result in an increasing price move, hence, the word "golden". For a further definition of the Golden Cross pattern, refer to appendix B. The second trend researched this summer is the opposite of a Golden Cross, a Death Cross. For a more completion explanation, refer to appendix D. The Death Cross is also a specialized chart trend that indicates a possibility of a selloff. Both patterns are believed to be reliable predictors in the stock market. Better visualizing these trends requires a series of complex SQL queries, designed to isolate both the trends themselves and then determine the crossover points and which type of crossover occurs. It is important to note that this thesis builds on previous work conducted in the summer of 2020. For a further background of the previous research, refer to appendix E.

IMPLEMENTATION

I. QUERIES

The curation of data is a whole other process in and of itself. The accumulation of stock market data has been ongoing for the past two year. It is approximately 500 gigabytes of data and contains tables of billions of rows, which presents a great research opportunity to those in this industry. The collection includes stock data all the way back from 2002. With a database this size, the potential for a data swamp is highly probable. For a further explanation of a data swamp, refer to appendix F. Due to the size of the database, the role of a database manager is essential. The beginning of this role primarily involved creating SSIS tasks (refer to appendix G), adding tasks to the SQL agent (refer to appendix H), scheduling jobs on the server, and spot-checking results. Later, we started automating the maintenance plans in the database, which consists of gathering, archiving, and backing up data. Automating processes is an efficient method to save time and allows for any database administrator to properly comprehend how to manage the data. Backing up the data requires an extensive amount of time and storage due to the size of the database (2 TB). Another important step in backups is filling any gaps of missing information in the data. The integrity of the data is crucial to achieve optimal functionality and verify the accuracy of the model.

As I joined the team and became more familiar with the queries necessary to comply with the stock server, we started creating specialized queries for the stock preselection. We spent hours curating specially designed queries over large timeframes to shape the database. For example, there exists a query to capture all daily highs, which is taking daily summary data and outputting the high trends, outputting to the daily summary table. Also, there are existing minute data that take snapshots of the market minute by minute. With these precedent queries, we started to grow the preselection and continue developing towards further trend specifications, i.e., Golden Cross and Death Cross.

II. METHODS

To provide context for the two patterns we tested, we performed the following steps:

1. Generate a list of highs and lows over various time periods
2. Determine what directions/trends stocks are going over those time periods
3. Catch stocks in "reversals" for analysis
4. Date each move
5. Calculate the type of cross that occurs
6. Aggregate the performance of the stocks of each type of cross over varying time periods

For step one, we created a query to process each stock and determine if a stock was up or down. This was calculated by examining a stocks close price and the buy price right afterwards. If there was a high after a low, then the stock is going up. If there was a low after a high, then the stock is going down. This sounds self-explanatory, but it is necessary to determine it on the analytical and technical level to see the direction of a trend.

Step two is the high/low step. We implement this step by developing queries with the said strategy of the Golden Cross and Death Cross such that the stocks that were up or down were defined. Thus, the Up&DownStocks.sql query was created. This query explains an overview of the stock data. An overview of the stock data was sampled and reviewed based on a variety of variables such as a 52-week high, a 52-week low, 26-week high and low, and the same for 13 weeks. The daily high query was used to verify the data and validate the max values. The 52, 26, and 13-week highs were found by finding the week low after the week high. These results were crucial for the development of the project as they are fundamental to further research trends.

After the Up&DownStocks.sql query was created, trends among daily quotes were found, thus creating the VReversalsDaily.sql query to find reversals in daily quotes. This process pulled from a group of stocks that were green (up) the day before, considering the green and red candlesticks seen within the stock market. Stocks that were in an uptrend were found based on if they were up the previous week. After this, stocks that were lower the previous three days were investigated to create a table called ‘gains’ that resulted in the profit data based on the open price, close price, quote date, and symbol, which is the company.

With the Up&DownStocks.sql and the VReversalsDaily.sql query, the Golden Cross and Death Cross queries had enough data to be created. So, the GoldenCross.sql query was created. The first step in this process was to use the date of the symbols as the primary key for each stock. The stocks were then organized by the moving average, which was defined previously. The moving average was outlined by 5 days and then 20 days. After these were calculated, the closing prices were determined resulting in the findings of the Golden Cross (GC) and Death Cross (DC). The GC is developed when the 5-day moving average is above the 20-day moving average. The DC is developed when the 5-day moving average is below the 20-day moving average.

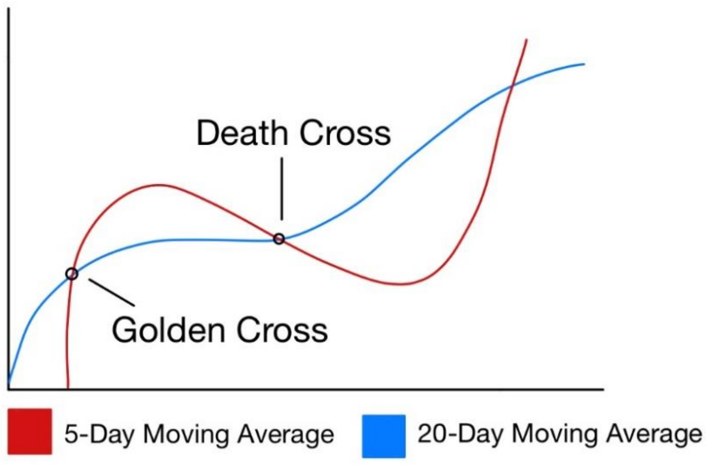


Figure 1 - Golden Cross and Death Cross Visual

The red line represents the 5-day moving average and the blue line represents the 20-day moving average.

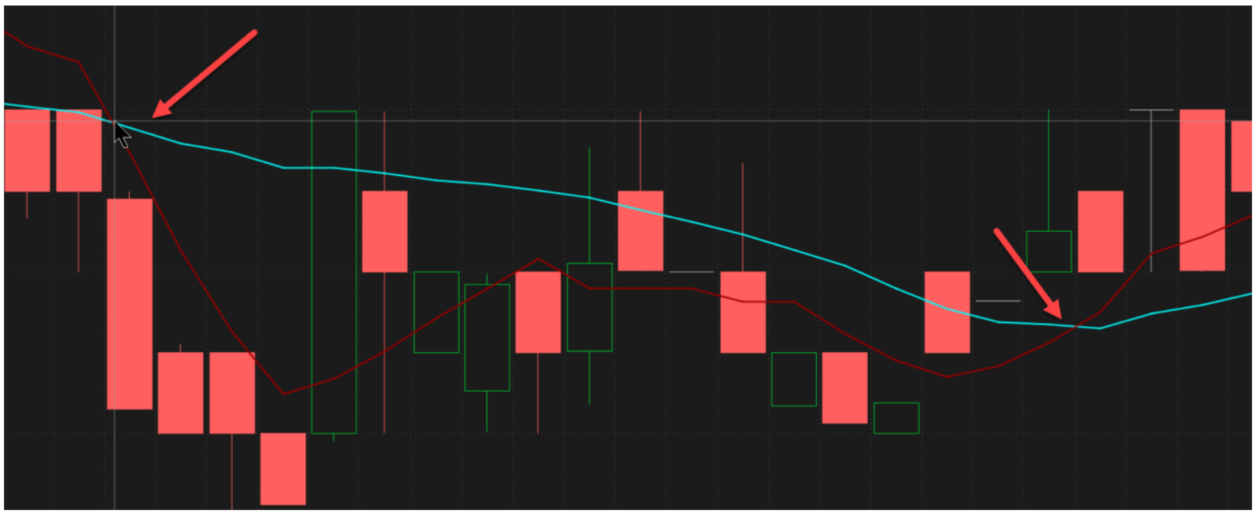


Figure 2 – Actual Death Cross and Golden Cross Visual in Stock Market

The image above illustrates an actual market Death Cross, seen by the first red arrow, showing the 5-day moving average diving below the 20-day moving average. The second red arrow illustrates a Golden Cross where the 5-day moving average rises above the 20-day moving average. Also note that the moving averages may vary by test, however, 5, 10, and 20- day moving averages were implemented for this research.

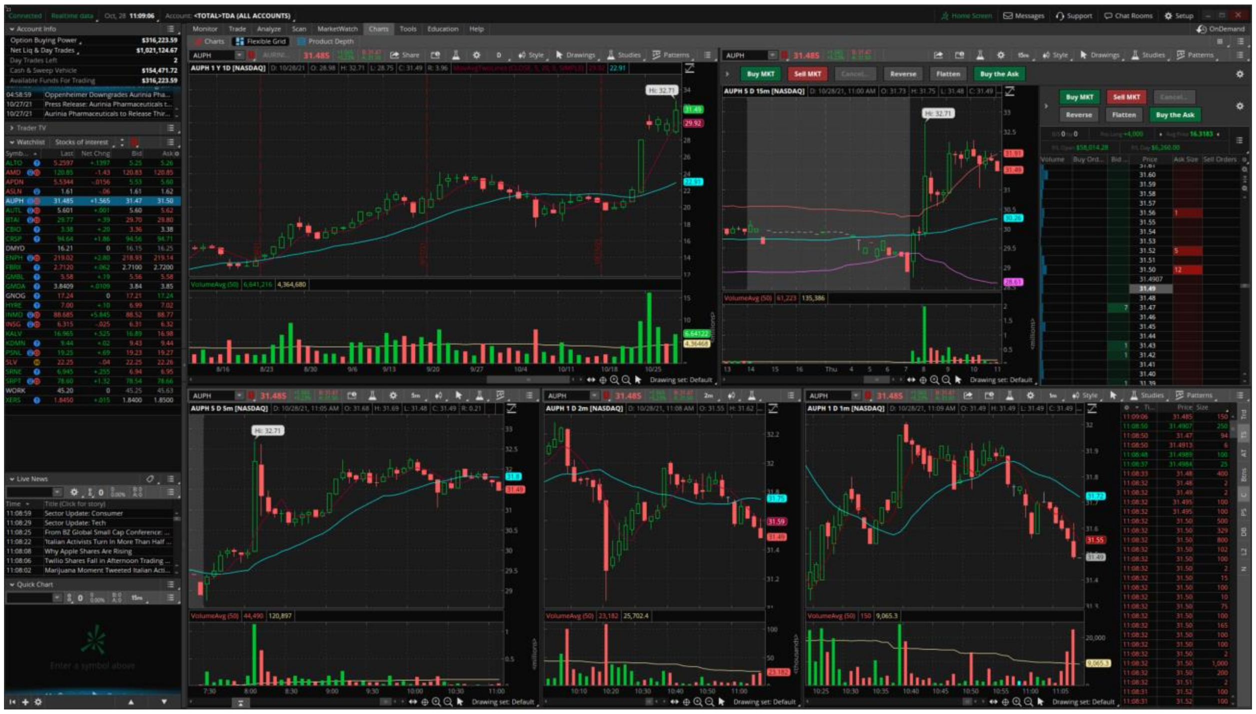


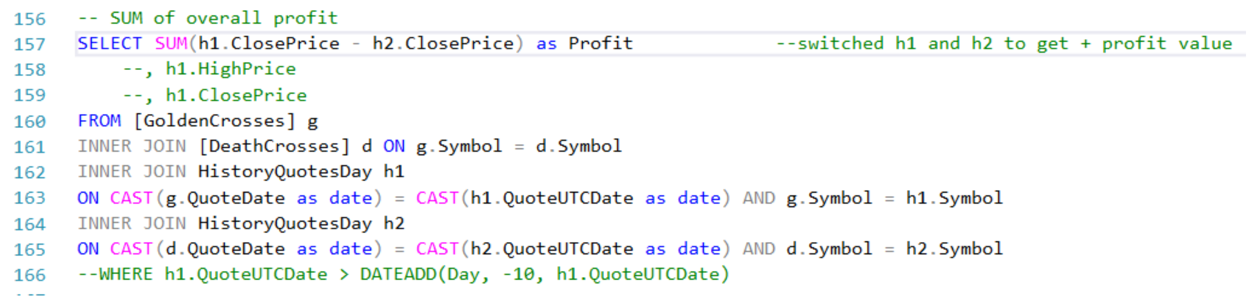
Figure 3 - Typical Trader Analyst's Platform

The image above is a screen scrape of what a typical trader's analyst platform looks like. This is using an application called Think or Swim.

With this information, finding where the Golden Cross occurred before the Death Cross was necessary. After this, the data was duplicated to ensure there were no gaps in the data. Validation of data is crucial in ensuring the accuracy of the results and trading algorithm, as mentioned above. After this, the buy point was found using the daily high data tables. Differing stocks were tested based on the close price. A new query was created called GoldesCrossUpdated.sql and was joined with the Up&DownStocks.sql table. It was joined on multiple primary keys to avoid duplicating unnecessary data. Duplicated data presented was an issue as there were two places that have a GC and DC afterward when it showed three points even though it should only be two. To fix this issue, the stocks/symbols were ordered by the most recent date to narrow the data down.

After the basics of the GC and DC were found, the tables were joined with the HistoryQuotesDay table, which contains a symbol/stock's Close Price, High Price, and Quote Date all of which are crucial in determining the profit evaluation. To retrieve the profit potential, the Golden Cross value was subtracted from the Death Cross value and summarized to find. In SQL terms, the h1ClosePrice was subtracted from the h2ClosePrice as seen below.

Figure 4 – Joining of the Golden Cross and Death Cross



```
-- SUM of overall profit
SELECT SUM(h1.ClosePrice - h2.ClosePrice) as Profit             --switched h1 and h2 to get + profit value
    --, h1.HighPrice
    --, h1.ClosePrice
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date) AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date) AND d.Symbol = h2.Symbol
--WHERE h1.QuoteUTCDate > DATEADD(Day, -10, h1.QuoteUTCDate)
```

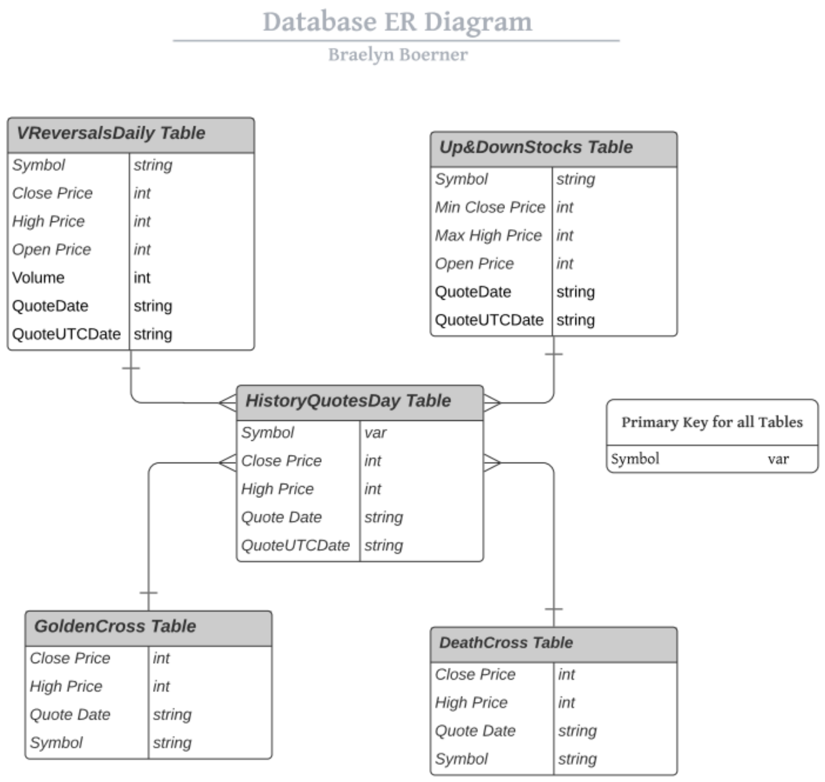


Figure 5 – Database Entity Relationship Diagram

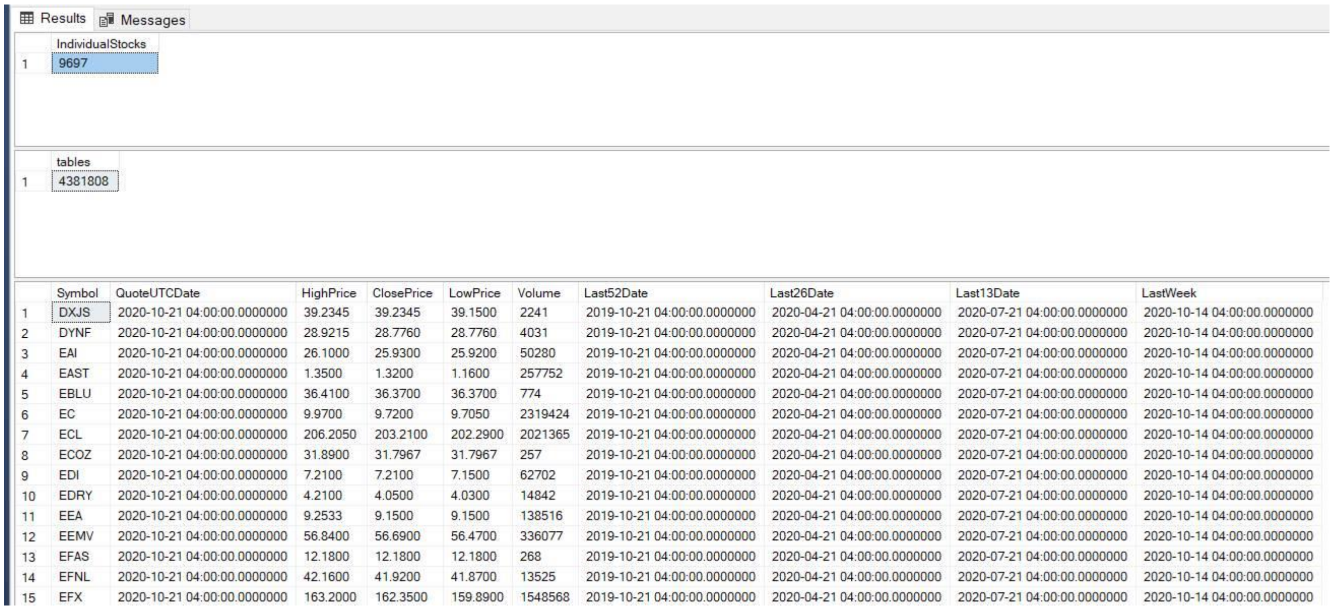
The diagram illustrates the connection between each of the SQL tables. All of the tables stem from the data in the History Quotes Day table and are connected through the symbol variable as the primary key.

TEST RESULTS

Result 1: Refer to UpandDownStocks.sql source code in Appendix I.

Figure 6 - Results of UpandDownStocks.sql Query

The results show the number of individual stocks, the Quote Date, the Low Price, the Volume, Last52Date, Last28Date, and LastWeek into a table called #MaxVals.



Results | Messages

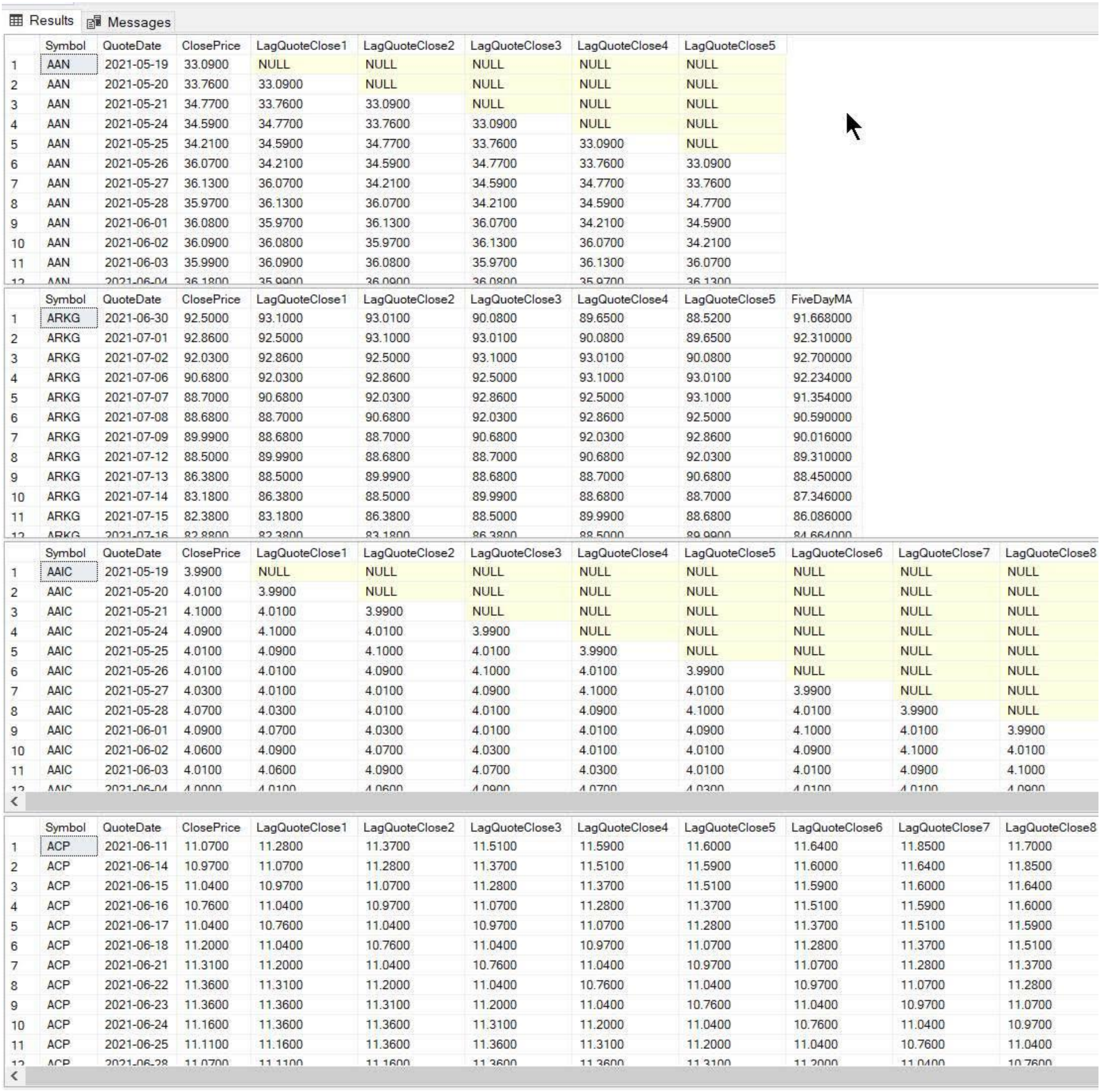
	IndividualStocks
1	9697

	tables
1	4381808

	Symbol	QuoteUTCDate	HighPrice	ClosePrice	LowPrice	Volume	Last52Date	Last26Date	Last13Date	LastWeek
1	DXJS	2020-10-21 04:00:00.0000000	39.2345	39.2345	39.1500	2241	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
2	DYNF	2020-10-21 04:00:00.0000000	28.9215	28.7760	28.7760	4031	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
3	EAI	2020-10-21 04:00:00.0000000	26.1000	25.9300	25.9200	50280	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
4	EAST	2020-10-21 04:00:00.0000000	1.3500	1.3200	1.1600	257752	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
5	EBLU	2020-10-21 04:00:00.0000000	36.4100	36.3700	36.3700	774	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
6	EC	2020-10-21 04:00:00.0000000	9.9700	9.7200	9.7050	2319424	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
7	ECL	2020-10-21 04:00:00.0000000	206.2050	203.2100	202.2900	2021365	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
8	ECOZ	2020-10-21 04:00:00.0000000	31.8900	31.7967	31.7967	257	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
9	EDI	2020-10-21 04:00:00.0000000	7.2100	7.2100	7.1500	62702	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
10	EDRY	2020-10-21 04:00:00.0000000	4.2100	4.0500	4.0300	14842	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
11	EEA	2020-10-21 04:00:00.0000000	9.2533	9.1500	9.1500	138516	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
12	EEMV	2020-10-21 04:00:00.0000000	56.8400	56.6900	56.4700	336077	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
13	EFAS	2020-10-21 04:00:00.0000000	12.1800	12.1800	12.1800	268	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
14	EFNL	2020-10-21 04:00:00.0000000	42.1600	41.9200	41.8700	13525	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000
15	EFX	2020-10-21 04:00:00.0000000	163.2000	162.3500	159.8900	1548568	2019-10-21 04:00:00.0000000	2020-04-21 04:00:00.0000000	2020-07-21 04:00:00.0000000	2020-10-14 04:00:00.0000000

Results 2: Refer to GoldenCross.sql source code in Appendix K.

Figure 7 - Results of GoldenCross.sql Query



Results Messages

	Symbol	QuoteDate	ClosePrice	LagQuoteClose1	LagQuoteClose2	LagQuoteClose3	LagQuoteClose4	LagQuoteClose5
1	AAN	2021-05-19	33.0900	NULL	NULL	NULL	NULL	NULL
2	AAN	2021-05-20	33.7600	33.0900	NULL	NULL	NULL	NULL
3	AAN	2021-05-21	34.7700	33.7600	33.0900	NULL	NULL	NULL
4	AAN	2021-05-24	34.5900	34.7700	33.7600	33.0900	NULL	NULL
5	AAN	2021-05-25	34.2100	34.5900	34.7700	33.7600	33.0900	NULL
6	AAN	2021-05-26	36.0700	34.2100	34.5900	34.7700	33.7600	33.0900
7	AAN	2021-05-27	36.1300	36.0700	34.2100	34.5900	34.7700	33.7600
8	AAN	2021-05-28	35.9700	36.1300	36.0700	34.2100	34.5900	34.7700
9	AAN	2021-06-01	36.0800	35.9700	36.1300	36.0700	34.2100	34.5900
10	AAN	2021-06-02	36.0900	36.0800	35.9700	36.1300	36.0700	34.2100
11	AAN	2021-06-03	35.9900	36.0900	36.0800	35.9700	36.1300	36.0700
12	AAN	2021-06-04	36.1800	35.9900	36.0900	36.0800	35.9700	36.1300

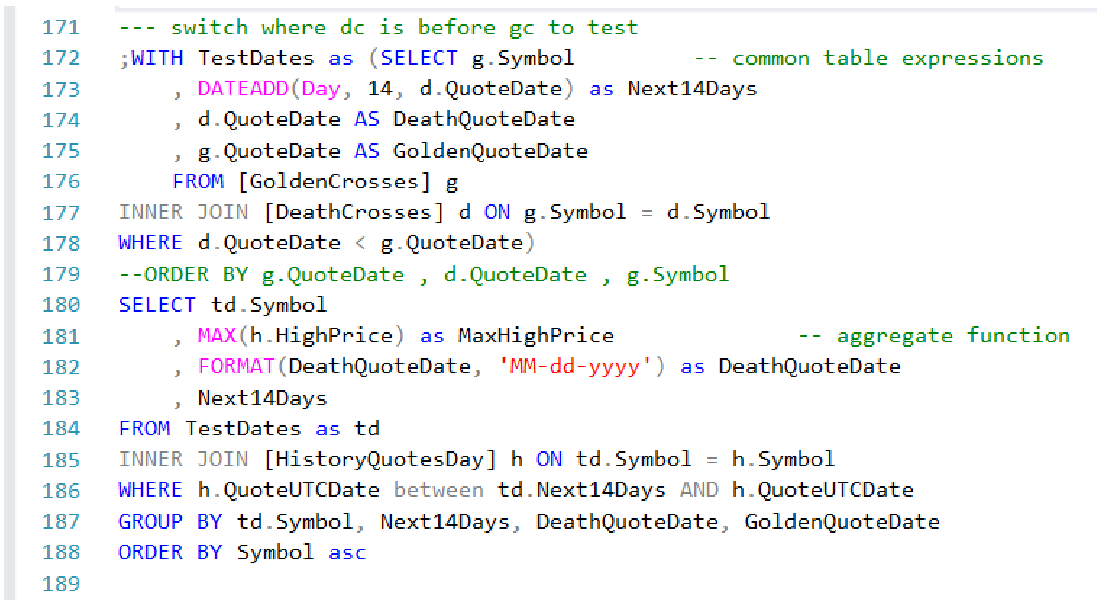
	Symbol	QuoteDate	ClosePrice	LagQuoteClose1	LagQuoteClose2	LagQuoteClose3	LagQuoteClose4	LagQuoteClose5	FiveDayMA
1	ARKG	2021-06-30	92.5000	93.1000	93.0100	90.0800	89.6500	88.5200	91.668000
2	ARKG	2021-07-01	92.8600	92.5000	93.1000	93.0100	90.0800	89.6500	92.310000
3	ARKG	2021-07-02	92.0300	92.8600	92.5000	93.1000	93.0100	90.0800	92.700000
4	ARKG	2021-07-06	90.6800	92.0300	92.8600	92.5000	93.1000	93.0100	92.234000
5	ARKG	2021-07-07	88.7000	90.6800	92.0300	92.8600	92.5000	93.1000	91.354000
6	ARKG	2021-07-08	88.6800	88.7000	90.6800	92.0300	92.8600	92.5000	90.590000
7	ARKG	2021-07-09	89.9900	88.6800	88.7000	90.6800	92.0300	92.8600	90.016000
8	ARKG	2021-07-12	88.5000	89.9900	88.6800	88.7000	90.6800	92.0300	89.310000
9	ARKG	2021-07-13	86.3800	88.5000	89.9900	88.6800	88.7000	90.6800	88.450000
10	ARKG	2021-07-14	83.1800	86.3800	88.5000	89.9900	88.6800	88.7000	87.346000
11	ARKG	2021-07-15	82.3800	83.1800	86.3800	88.5000	89.9900	88.6800	86.086000
12	ARKG	2021-07-16	82.8800	82.3800	83.1800	86.3800	88.5000	89.9900	84.664000

	Symbol	QuoteDate	ClosePrice	LagQuoteClose1	LagQuoteClose2	LagQuoteClose3	LagQuoteClose4	LagQuoteClose5	LagQuoteClose6	LagQuoteClose7	LagQuoteClose8
1	AAIC	2021-05-19	3.9900	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
2	AAIC	2021-05-20	4.0100	3.9900	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	AAIC	2021-05-21	4.1000	4.0100	3.9900	NULL	NULL	NULL	NULL	NULL	NULL
4	AAIC	2021-05-24	4.0900	4.1000	4.0100	3.9900	NULL	NULL	NULL	NULL	NULL
5	AAIC	2021-05-25	4.0100	4.0900	4.1000	4.0100	3.9900	NULL	NULL	NULL	NULL
6	AAIC	2021-05-26	4.0100	4.0100	4.0900	4.1000	4.0100	3.9900	NULL	NULL	NULL
7	AAIC	2021-05-27	4.0300	4.0100	4.0100	4.0900	4.1000	4.0100	3.9900	NULL	NULL
8	AAIC	2021-05-28	4.0700	4.0300	4.0100	4.0100	4.0900	4.1000	4.0100	3.9900	NULL
9	AAIC	2021-06-01	4.0900	4.0700	4.0300	4.0100	4.0100	4.0900	4.1000	4.0100	3.9900
10	AAIC	2021-06-02	4.0600	4.0900	4.0700	4.0300	4.0100	4.0100	4.0900	4.1000	4.0100
11	AAIC	2021-06-03	4.0100	4.0600	4.0900	4.0700	4.0300	4.0100	4.0100	4.0900	4.1000
12	AAIC	2021-06-04	4.0000	4.0100	4.0600	4.0900	4.0700	4.0300	4.0100	4.0100	4.0900

	Symbol	QuoteDate	ClosePrice	LagQuoteClose1	LagQuoteClose2	LagQuoteClose3	LagQuoteClose4	LagQuoteClose5	LagQuoteClose6	LagQuoteClose7	LagQuoteClose8
1	ACP	2021-06-11	11.0700	11.2800	11.3700	11.5100	11.5900	11.6000	11.6400	11.8500	11.7000
2	ACP	2021-06-14	10.9700	11.0700	11.2800	11.3700	11.5100	11.5900	11.6000	11.6400	11.8500
3	ACP	2021-06-15	11.0400	10.9700	11.0700	11.2800	11.3700	11.5100	11.5900	11.6000	11.6400
4	ACP	2021-06-16	10.7600	11.0400	10.9700	11.0700	11.2800	11.3700	11.5100	11.5900	11.6000
5	ACP	2021-06-17	11.0400	10.7600	11.0400	10.9700	11.0700	11.2800	11.3700	11.5100	11.5900
6	ACP	2021-06-18	11.2000	11.0400	10.7600	11.0400	10.9700	11.0700	11.2800	11.3700	11.5100
7	ACP	2021-06-21	11.3100	11.2000	11.0400	10.7600	11.0400	10.9700	11.0700	11.2800	11.3700
8	ACP	2021-06-22	11.3600	11.3100	11.2000	11.0400	10.7600	11.0400	10.9700	11.0700	11.2800
9	ACP	2021-06-23	11.3600	11.3600	11.3100	11.2000	11.0400	10.7600	11.0400	10.9700	11.0700
10	ACP	2021-06-24	11.1600	11.3600	11.3600	11.3100	11.2000	11.0400	10.7600	11.0400	10.9700
11	ACP	2021-06-25	11.1100	11.1600	11.3600	11.3600	11.3100	11.2000	11.0400	10.7600	11.0400
12	ACP	2021-06-28	11.0700	11.1100	11.1600	11.3600	11.3600	11.3100	11.2000	11.0400	10.7600

To find the overall summary of the profit and see the total gain the prices for the GC and DC were joined with the Daily Summary table and History Quotes Day table. Within the span of a few months, the results showed a profit of -$5226.0994. After these results, the hypothesis, GC before DC, was proven incorrect. The method was switched to finding quotes where the DC appears before the GC. So, buy on the DC and sell on the GC. The h1 and h2 values of the Close Price within the were switched to ensure optimal profit value, which resulted in +$5226.0994. So, this means that the optimal method for the highest profit is buying on the Death Cross and selling on the Golden Cross as seen below using a common table expression.

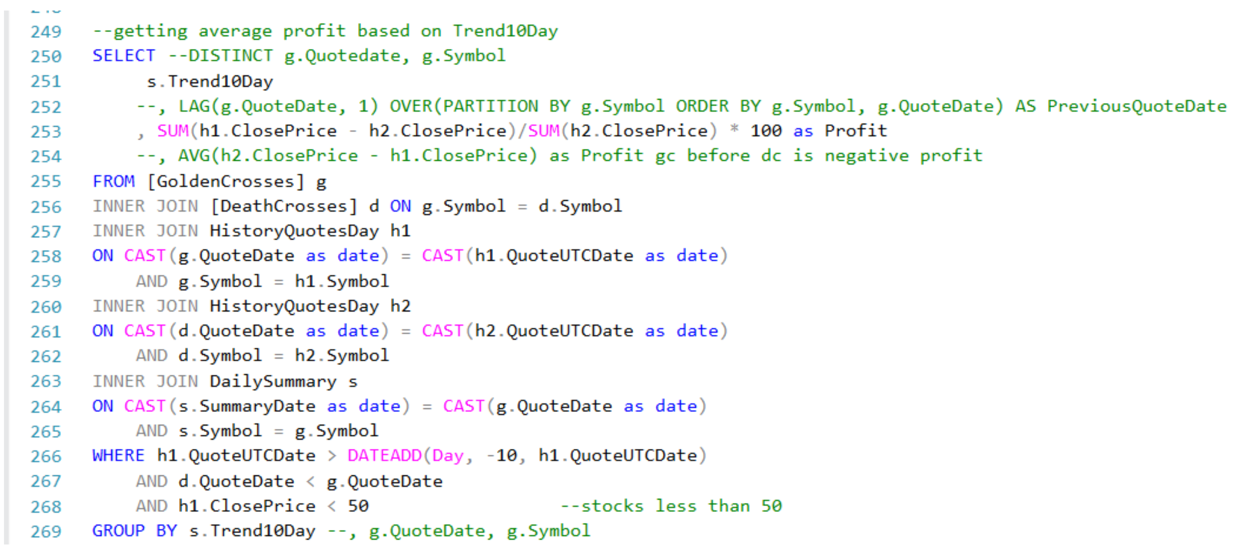
Figure 8 – GoldenCrossUpdated.sql Query Source Code



```sql
--- switch where dc is before gc to test
;WITH TestDates as (SELECT g.Symbol          -- common table expressions
    , DATEADD(Day, 14, d.QuoteDate) as Next14Days
    , d.QuoteDate AS DeathQuoteDate
    , g.QuoteDate AS GoldenQuoteDate
    FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
WHERE d.QuoteDate < g.QuoteDate)
--ORDER BY g.QuoteDate , d.QuoteDate , g.Symbol
SELECT td.Symbol
    , MAX(h.HighPrice) as MaxHighPrice              -- aggregate function
    , FORMAT(DeathQuoteDate, 'MM-dd-yyyy') as DeathQuoteDate
    , Next14Days
FROM TestDates as td
INNER JOIN [HistoryQuotesDay] h ON td.Symbol = h.Symbol
WHERE h.QuoteUTCDate between td.Next14Days AND h.QuoteUTCDate
GROUP BY td.Symbol, Next14Days, DeathQuoteDate, GoldenQuoteDate
ORDER BY Symbol asc
```

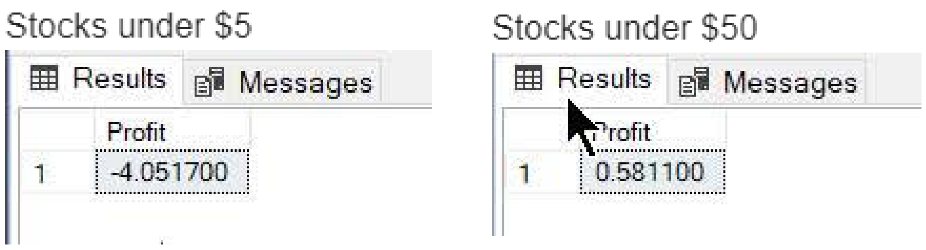
After this, more tests were performed to validate that the original hypothesis was incorrect. This was done by looking at the dates within the stock market trading software. The average profit based on Trend 10Day was calculated as seen below.

Figure 9 – Switch for Optimal Profit Source Code



```
--getting average profit based on Trend10Day
SELECT --DISTINCT g.Quotedate, g.Symbol
       s.Trend10Day
     --, LAG(g.QuoteDate, 1) OVER(PARTITION BY g.Symbol ORDER BY g.Symbol, g.QuoteDate) AS PreviousQuoteDate
     , SUM(h1.ClosePrice - h2.ClosePrice)/SUM(h2.ClosePrice) * 100 as Profit
     --, AVG(h2.ClosePrice - h1.ClosePrice) as Profit gc before dc is negative profit
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date)
     AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date)
     AND d.Symbol = h2.Symbol
INNER JOIN DailySummary s
ON CAST(s.SummaryDate as date) = CAST(g.QuoteDate as date)
     AND s.Symbol = g.Symbol
WHERE h1.QuoteUTCDate > DATEADD(Day, -10, h1.QuoteUTCDate)
     AND d.QuoteDate < g.QuoteDate
     AND h1.ClosePrice < 50                    --stocks less than 50
GROUP BY s.Trend10Day --, g.QuoteDate, g.Symbol
```

The up or downtrend of the stock was calculated based on the close price of the stock. The results are seen below. Also, the percentages of the profit were found by dividing the sum of the profit by the sum of the close price times 100.



Stocks under $5

Results | Messages

	Profit
1	-4.051700

Stocks under $50

Results | Messages

	Profit
1	0.581100

Figure 10 – Profit Results Based on Stock Price

One of the most interesting pieces about the results depends on the price of the stock. For example, stocks less than $5 or less than $50 output different results on not only price but also the order of the cross pattern. As you can see in the images above, stocks less than $5 produce negative profit and stocks less than $50 produce positive profit when buying first on the DC and selling on the GC. Of course, these are very small scale, however, once invested in a multitude of stocks, the profit instantly increases.

CONCLUSION

The objective of this project was to ask the question: Can one use patterns developed within the stock market to predict the behavior and achieve positive financial margins? With this information and a sample database of around nineteen years of stock data, this hypothesis was tested on the Golden and Death Cross. Using specialized SQL queries these patterns were investigated through a series of tables and extensively explored to provide relevant data needed to achieve a prediction method for the trading algorithm. The data demonstrated that the opposite hypothesis, buying on the Death Cross and selling on the Golden Cross, occurred when employing these patterns implemented by this approach. The original hypothesis for the optimal collaboration of patterns regarding Golden Cross and Death Cross performance was to buy on the Golden Cross and to sell on the Death Cross. However, the development of this data proved otherwise. The original hypothesis was 180 degrees incorrect. The data showed the highest profit occurred to buy on the Death Cross and sell on the Golden Cross. Although the hypothesis was proved to be incorrect, the results are still profitable and successful. The result of this work points to why it is necessary to test trends and create software to do so. Researchers and stock traders do not want to question the possibility of major success at the expense of capital gain/money. So, ensuring any possible source for success, regardless how it occurred, concludes with positive results.

I think the biggest reason for the opposing hypothesis, buy on the DC and sell on the GC, is a statistical variation due to the types of traders and what types of stocks they buy. For example, retail investors who buy stocks less than five dollars are the ones who follow the front-running ideology of buying on GC and selling on the DC. Hedge funds on the other hand look at stocks differently as they have policies to not invest in stocks lower than five dollars. A reason for this may be because cheaper stocks tend to be new and less substantial. This creates margins in the stock market as it shows the variation of differing traders. So, lower retail traders who ride lower stock prices follow the front running patterns of GC before DC as it is the most “intuitively correct”. However, the data points to a new type of trading pattern for a higher profit. Buying on the Death Cross and selling on the Golden Cross often results in higher profit due to the institutional investors who deal with higher priced stock and use technical indicators less. Even without the clear profit results, there is a change in the pattern. Of course, higher stocks will have a higher profit, however, when the profit occurs there is an interesting change in the pattern order, buying on DC first or GC first. Although it is difficult to say the exact reasons the hypothesis was completely different from the concluding results, the data still presents itself as supporting information for the stock screener application.

FUTURE WORK

Because the results went against conventional wisdom, it behooves us to determine the exact cause. Other future work of this project is to incorporate artificial intelligence (AI) and machine learning (ML) into the algorithm. The research presented in this thesis serves as a great addition to the stock screener application. The use of trend detection using SQL analysis aids to the application as it narrows in on what data to focus on and how to better improve the model. In addition to the AI/ML incorporation, adding new filters on the existing queries would be useful in further testing and modifications.

WORKS CITED

Chen, James. “What Is a Death Cross?” *Investopedia*, Investopedia, 9 Nov. 2021, https://www.investopedia.com/terms/d/deathcross.asp.

Chugugrace. “SQL Server Integration Services - SQL Server Integration Services (SSIS).” *SQL Server Integration Services (SSIS) | Microsoft Docs*, https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-ver15.

“Data Lakes and Data Swamps.” *IBM Developer*, https://developer.ibm.com/articles/ba-data-becomes-knowledge-2/.

Fernando, Jason. “Moving Average (MA) Definition.” *Investopedia*, Investopedia, 9 Nov. 2021, https://www.investopedia.com/terms/m/movingaverage.asp.

Hayes, Adam. “What Is a Golden Cross?” *Investopedia*, Investopedia, 9 Nov. 2021, https://www.investopedia.com/terms/g/goldencross.asp.

APPENDICIES

A. **Investopedia Definition -** A financial site that originated in New York. This website was used for multiple definitions such as, Golden Cross, Death Cross, and moving average.

B. **Golden Cross Definition Expanded -** A Golden Cross is defined by, "a chart pattern in which a relatively short-term moving average crosses above a long-term moving average" (Investopedia Golden Cross). It is a "breakout pattern formed from a crossover involving a security's short-term moving average (such as the 15-day moving average) breaking above its long-term moving average (such as the 50-day moving average) or resistance level. As long-term indicators carry more weight, the Golden Cross indicates a bull market on the horizon and is reinforced by high trading volumes" (Investopedia Golden Cross).

C. **Moving Average Definition -** A moving average is defined by, "...a calculation used to analyze data points by creating a series of averages of different subsets of the full data set" (Investopedia Moving Average).

D. **Death Cross Definition -** Investopedia provides a definition as it writes, "[t]he Death Cross is a technical chart pattern indicating the potential for a major sell-off. The Death Cross appears on a chart when a stock's short-term moving average crosses below its long-term moving average. Typically, the most common moving averages used in this pattern are the 50-day and 200-day moving averages" (Investopedia Death Cross).

E. **Summer 2020 Research on the Stock Bot -** The genesis of the app began in the summer of 2020. The team members included Kyle Duncan, Jeffrey Fairbanks, Andrew Welk, Dr. McCarty, and I. Each of us had different roles in the project. Kyle and Jeff worked on the machine learning of the application, my role was and is the database developer, Andrew was the systems architect, and Dr. McCarty was the project creator and manager. Throughout the duration of this project, Orion Trotter took on the role of the machine learning developer and Ender Sandiage joined the team as another systems architect. Also, a special thanks to Tyler Shea for the assistance in developing the Golden Cross and Death Cross queries. The goal of the project was to create an application that can automatically invest and sell in the stock market. Stock screener applications similar to this are currently being used all around the world. However, we wanted to create an application that integrated more. Most applications use high-tech algorithms to adhere to such tasks and eliminate human interaction. Dr. McCarty wanted to add new elements to the mix such as fuzzy logic using exterior platforms such as social media to integrate into specialized trading algorithms. The primary motive to create such an application is for the advancement of the Computer Science Department at NNU. We want to produce a means of financial gain for a greater purpose that serves the school well and administers greater opportunities for the students

F. **Data Swamp Definition -** IMB shares a definition of a data swamp as it writes, “A data swamp is a badly designed, inadequately documented, or poorly maintained data lake. These deficiencies compromise the ability to retrieve data, and users are unable to analyze and exploit the data efficiently” (IBM Developer). Data lakes and warehouses include processes that manage massive amounts of data that are intended to perform specialized queries and data analysis (IBM Developer).

G. **SSIS Tasks Definition -** SQL Service Integration Service. Microsoft provides a definition of SSIS as it writes, “SQL Server Integration Services is a platform for building enterprise-level data integration and data transformations solutions. Use

Integration Services to solve complex business problems by copying or downloading files, loading data warehouses, cleansing and mining data, and managing SQL Server objects and data" (Microsoft Ignite, SSIS Tasks).

H. **SQL Agent Definition -** Microsoft provides a definition of the SQL Agent as it writes, "SQL Server Agent is a Microsoft Windows service that executes scheduled administrative tasks, which are called *jobs* in SQL Server" (Microsoft Ignite, SQL Agent).

I. **UpandDownStocks.sql Source Code –**

```
USE StockAnalysis
GO
-- #1 the number of individual stocks in our sample
SELECT COUNT(distinct symbol) FROM DailySummary -- = 7938
-- the number of tables
SELECT COUNT(*) as [tables] FROM dbo.DailySummary -- = 3530391
-- For 52: Stocks that are up (52-week high after 52-week low)
SELECT Symbol, Max (HighPrice) as MaxValue52
INTO #MaxValue52
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -12, QuoteUTCDate)
GROUP BY Symbol -- 7938
-- select * from #MaxValue52
--For 52: Stocks that are down (52-week low after 52-week high)
-- low values
USE StockAnalysis
GO
SELECT
        Symbol
        , Min (LowPrice) as MinValue52
INTO #MinValue52
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -12, QuoteUTCDate)
GROUP BY Symbol
-- select * from #MinValue52
-- For 26: Stocks that are up (26-week high after 26-week low)
DROP TABLE IF EXISTS #MaxValue26
SELECT
        Symbol
        , Max (HighPrice) as MaxValue26
INTO #MaxValue26
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -6, QuoteUTCDate)
GROUP BY Symbol
GO
-- select * from #MaxValue26
-- 26 week low
```

```
SELECT
            Symbol
   , Min (LowPrice) as MinValue26
INTO #MinValue26
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -6, QuoteUTCDate)
GROUP BY Symbol
GO
-- select * from #MinValue26
-- For 13: Stocks that are up (13-week high after 13-week low)
SELECT
            Symbol
   , Max (HighPrice) as MaxValue13
INTO #MaxValue13
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -3, QuoteUTCDate)
GROUP BY Symbol
GO
-- select * from #MaxValue13
-- 13 week low
SELECT
            Symbol
   , Min (LowPrice) as MinValue13
INTO #MinValue13
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -3, QuoteUTCDate)
GROUP BY Symbol
GO
-- select * from #MinValue13
-- from daigly highs -- MaxValues
-- Current high's for each date
DROP TABLE IF EXISTS #MaxValues
SELECT --DISTINCT
            Symbol
   , QuoteUTCDate
            , HighPrice
            , ClosePrice
            , LowPrice
            , Volume
            , DATEADD(Month, -12, QuoteUTCDate) AS Last52Date
            , DATEADD(Month, -6, QuoteUTCDate) AS Last26Date
            , DATEADD(Month, -3, QuoteUTCDate) AS Last13Date
            , DATEADD(Day, -7, QuoteUTCDate) AS LastWeek
INTO #MaxValues
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(YEAR, -2, GetDate())
AND ClosePrice > 1
AND UPPER(Symbol) = Symbol COLLATE Latin1_General_CS_AS
select * from #MaxValues
-- SELECT * FROM #MaxValues;
-- SELECT COUNT(distinct symbol) FROM #MaxValues -- = 7947 (why are there 9 more values than in the
DailySummary table?)
-- 52 week high join -- I did v for value and q for quote
DROP TABLE IF EXISTS #maxHigh52Date;
SELECT
            v.Symbol
```

```
	, v.QuoteUTCDate
	, Max(q.QuoteUTCDate) AS MaxHigh52Date
INTO #maxHigh52Date
FROM #MaxValues v
INNER JOIN #MaxValues q -- I do not think the max values should be the same...
	ON v.Symbol = q.Symbol
	AND q.QuoteUTCDate BETWEEN v.Last52Date AND v.QuoteUTCDate
INNER JOIN #MaxValue52 mh52
	ON v.Symbol = mh52.Symbol
	AND mh52.MaxValue52 = q.HighPrice
GROUP BY v.Symbol, v.QuoteUTCDate;
-- SELECT * FROM #MaxHigh52Date;
DROP TABLE IF EXISTS #maxHigh26Date;
SELECT
	v.Symbol
	, v.QuoteUTCDate
	, Max(q.QuoteUTCDate) AS MaxHigh26Date
INTO #maxHigh26Date
FROM #MaxValues v
INNER JOIN #MaxValues q
	ON v.Symbol = q.Symbol
	AND q.QuoteUTCDate BETWEEN v.Last26Date AND v.QuoteUTCDate
INNER JOIN #MaxValue26 mh26
	ON v.Symbol = mh26.Symbol
	AND mh26.MaxValue26 = q.HighPrice
GROUP BY v.Symbol, v.QuoteUTCDate;
-- SELECT * FROM #maxHigh26Date
-- 13 week high join
DROP TABLE IF EXISTS #maxHigh13Date;
SELECT
	v.Symbol
	, v.QuoteUTCDate
	, Max(q.QuoteUTCDate) AS MaxHigh13Date
INTO #maxHigh13Date
FROM #MaxValues v
INNER JOIN #MaxValues q
	ON v.Symbol = q.Symbol
	AND q.QuoteUTCDate BETWEEN v.Last13Date AND v.QuoteUTCDate
INNER JOIN #MaxValue13 mh13
	ON v.Symbol = mh13.Symbol
	AND mh13.MaxValue13 = q.HighPrice
GROUP BY v.Symbol, v.QuoteUTCDate;
-- SELECT * FROM #maxHigh13Date;
-- 52 max high week
DROP TABLE IF EXISTS #maxHighWeek;
SELECT
	v.Symbol
	, v.QuoteUTCDate
	, Max(q.HighPrice) AS HighWeek
INTO #maxHighWeek
FROM #MaxValues v
INNER JOIN #MaxValues q
	ON v.Symbol = q.Symbol
	AND q.QuoteUTCDate BETWEEN v.LastWeek AND v.QuoteUTCDate
GROUP BY v.Symbol, v.QuoteUTCDate
-- select * from #maxHighWeek
```

J. VReversalsDaily.sql Source Code –

```
-- what about reversals in daily quotes, can they work
-- first we want stocks where yesterday's bar was green
DECLARE @testDateCurrent DATE = '5/14/2021';
DROP TABLE IF EXISTS #green_today;
-- disregard the no-interest stocks too
SELECT Symbol, ClosePrice, HighPrice, OpenPrice, Volume
    , MAX(CAST(QuoteUTCDate AS DATE)) As LastDate, LAG(QuoteUTCDate, 1) OVER(PARTITION
   BY Symbol ORDER BY QuoteUTCDate) AS LagQuoteDate1, LEAD(QuoteUTCDate, 1)
OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS LeadQuoteDate1
, LAG(QuoteUTCDate, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteDate2
, LEAD(QuoteUTCDate, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LeadQuoteDate2
, LAG(QuoteUTCDate, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteDate3
, LEAD(QuoteUTCDate, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LeadQuoteDate3
, LAG(QuoteUTCDate, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteDate4
, LEAD(QuoteUTCDate, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LeadQuoteDate4
, LAG(QuoteUTCDate, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteDate5
, LEAD(QuoteUTCDate, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LeadQuoteDate5
INTO #green_today
FROM dbo.HistoryQuotesDay
WHERE UPPER(Symbol) = Symbol COLLATE Latin1_General_CS_AS          -- some symbols are bad
GROUP BY Symbol, ClosePrice, HighPrice, OpenPrice, Volume, QuoteUTCDate
HAVING MAX(CAST(QuoteUTCDate AS DATE)) >= DATEADD( Day, -7, @testDateCurrent)
;
DELETE #green_today
SELECT * FROM #green_today
FROM #green_today
WHERE ClosePrice <= OpenPrice
OR ClosePrice NOT BETWEEN 2 AND 200
OR Volume < 50000
OR LastDate <> @testDateCurrent
--SELECT * FROM #green_today
--ORDER BY Symbol, LastDate;
/*
SELECT count(*)
FROM dbo.HistoryQuotesDay
WHERE ClosePrice > OpenPrice
AND ClosePrice BETWEEN 2 AND 200
AND Volume > 50000
AND UPPER(Symbol) = Symbol COLLATE Latin1_General_CS_AS  -- some symbols are bad
AND Symbol = 'GNCA'
;
*/
--DECLARE @testDateCurrent1 DATE = '5/14/2021';
-- pull from that group the stocks that were green the day before
DROP TABLE IF EXISTS #green_both_days;
```

```
SELECT d.Symbol
INTO #green_both_days
FROM dbo.HistoryQuotesDay d
INNER JOIN #green_today gt
            ON d.Symbol = gt.Symbol
            AND CAST(d.QuoteUTCDate AS DATE) = CAST(gt.LagQuoteDate1 AS DATE)
WHERE d.ClosePrice > d.OpenPrice
AND d.ClosePrice < gt.ClosePrice
AND d.HighPrice < gt.HighPrice                -- needs to break previous day high too
;
--SELECT * FROM #green_both_days d
--INNER JOIN #green_today gt
--          ON d.Symbol = gt.Symbol;
DROP TABLE IF EXISTS #in_uptrend;
-- what about stocks in an uptrend
-- defined as higher than previous week
SELECT d.Symbol, gt.ClosePrice AS CurrentClose, d.ClosePrice AS LastWeekClose
INTO #in_uptrend
FROM dbo.HistoryQuotesDay d
INNER JOIN #green_today gt
            ON d.Symbol = gt.Symbol
INNER JOIN #green_both_days gbd
            ON d.Symbol = gbd.Symbol
WHERE CAST(d.QuoteUTCDate AS DATE) = DATEADD( Day, -14, @testDateCurrent)
AND gt.ClosePrice > d.ClosePrice
;
--SELECT * FROM #in_uptrend;
DROP TABLE IF EXISTS #Gains
-- lower the previous 3 days?
SELECT gt.ClosePrice, gt.Symbol
            , d1.ClosePrice AS PrevDay1Close
            , d2.ClosePrice AS PrevDay2Close
            , d3.ClosePrice AS PrevDay3Close
            , L1.ClosePrice AS PostDay1Close
            , L2.ClosePrice AS PostDay2Close
            , L3.ClosePrice AS PostDay3Close
            , L4.ClosePrice AS PostDay4Close
            , L1.ClosePrice - gt.ClosePrice AS GainForDay1
            , L2.ClosePrice - gt.ClosePrice AS GainForDay2
            , L3.ClosePrice - gt.ClosePrice AS GainForDay3
            , L4.ClosePrice - gt.ClosePrice AS GainForDay4
INTO #Gains
FROM dbo.HistoryQuotesDay d1
INNER JOIN #green_today gt
            ON d1.Symbol = gt.Symbol
            AND d1.QuoteUTCDate = gt.LagQuoteDate2
INNER JOIN #green_both_days gbd
            ON d1.Symbol = gbd.Symbol
INNER JOIN #in_uptrend i
            ON i.Symbol = d1.Symbol
INNER JOIN dbo.HistoryQuotesDay d2
            ON d2.Symbol = gt.Symbol
            AND d2.QuoteUTCDate = gt.LagQuoteDate3
INNER JOIN dbo.HistoryQuotesDay d3
            ON d3.Symbol = gt.Symbol
            AND d3.QuoteUTCDate = gt.LagQuoteDate4
```

```
INNER JOIN dbo.HistoryQuotesDay L1
            ON L1.Symbol = gt.Symbol
            AND L1.QuoteUTCDate = gt.LeadQuoteDate1
INNER JOIN dbo.HistoryQuotesDay L2
            ON L2.Symbol = gt.Symbol
            AND L2.QuoteUTCDate = gt.LeadQuoteDate2
INNER JOIN dbo.HistoryQuotesDay L3
            ON L3.Symbol = gt.Symbol
            AND L3.QuoteUTCDate = gt.LeadQuoteDate3
INNER JOIN dbo.HistoryQuotesDay L4
            ON L4.Symbol = gt.Symbol
            AND L4.QuoteUTCDate = gt.LeadQuoteDate4
WHERE d1.ClosePrice < d1.OpenPrice
AND d2.ClosePrice < d2.OpenPrice
AND d3.ClosePrice < d3.OpenPrice
SELECT SUM(GainForDay1) AS GainForDay1
            , SUM(GainForDay2) AS GainForDay2
            , SUM(GainforDay3) AS GainForDay3
            , SUM(GainforDay4) AS GainForDay4
FROM #Gains
SELECT * FROM #Gains
```

K. GoldenCross.sql Source Code –

```
-- Summary Table
-- date primary key
-- symbol
-- 5 MA
-- 20 MA
-- closing price
-- SAMPLE
SELECT Symbol
, CAST(QuoteUTCDate As DATE) AS QuoteDate
      , ClosePrice
      , LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
      , LAG(ClosePrice, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose2
      , LAG(ClosePrice, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose3
      , LAG(ClosePrice, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose4
      , LAG(ClosePrice, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose5
INTO #temp1
FROM dbo.vHistoryQuotesDay
WHERE QuoteUTCDate > '5/07/2021'
AND Symbol = 'AAPL'
-- Temp Table for 5 day MA
DROP TABLE IF EXISTS #Temp5MA
SELECT Symbol
                        , CAST(QuoteUTCDate AS DATE) AS QuoteDate
                        , ClosePrice
```

```
                    , LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
                    , LAG(ClosePrice, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose2
                    , LAG(ClosePrice, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose3
                    , LAG(ClosePrice, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose4
                    , LAG(ClosePrice, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose5
INTO #Temp5MA
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > '05/19/2021'
SELECT * FROM #Temp5MA
-- 5 Day Moving average calculation SUM
DROP TABLE IF EXISTS #Sum5DayMA
SELECT *, (ClosePrice + LagQuoteClose1 + LagQuoteClose2 + LagQuoteClose3 + LagQuoteClose4) / 5
As FiveDayMA
INTO #Sum5DayMA
FROM #Temp5MA
SELECT * FROM #Sum5DayMA
-- 20 Day Moving Average TEMP table
DROP TABLE IF EXISTS #Temp20MA
SELECT Symbol
                    , CAST(QuoteUTCDate AS DATE) AS QuoteDate
                    , ClosePrice
                    , LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
                    , LAG(ClosePrice, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose2
                    , LAG(ClosePrice, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose3
                    , LAG(ClosePrice, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose4
                    , LAG(ClosePrice, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose5
                    , LAG(ClosePrice, 6) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose6
                    , LAG(ClosePrice, 7) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose7
                    , LAG(ClosePrice, 8) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose8
                    , LAG(ClosePrice, 9) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose9
                    , LAG(ClosePrice, 10) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose10
                    , LAG(ClosePrice, 11) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose11
                    , LAG(ClosePrice, 12) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose12
                    , LAG(ClosePrice, 13) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose13
                    , LAG(ClosePrice, 14) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose14
                    , LAG(ClosePrice, 15) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose15
```

```
                    , LAG(ClosePrice, 16) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose16
                    , LAG(ClosePrice, 17) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose17
                    , LAG(ClosePrice, 18) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose18
                    , LAG(ClosePrice, 19) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose19
                    , LAG(ClosePrice, 20) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose20
INTO #Temp20MA
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > '05/19/2021'
SELECT * FROM #Temp20MA
-- 20 Day Moving average calculation
DROP TABLE IF EXISTS #Sum20DayMA
SELECT *, (ClosePrice + LagQuoteClose1 + LagQuoteClose2 + LagQuoteClose3 + LagQuoteClose4 +
LagQuoteClose5 +
        LagQuoteClose6 + LagQuoteClose7 + LagQuoteClose8 + LagQuoteClose9 + LagQuoteClose10 +
LagQuoteClose11 + LagQuoteClose12 +
        LagQuoteClose13 + LagQuoteClose14 + LagQuoteClose15 + LagQuoteClose16 +
LagQuoteClose17 + LagQuoteClose18 + LagQuoteClose19) / 20 AS SumDay20MA
INTO #Sum20DayMA
FROM #Temp20MA
SELECT * FROM #Sum20DayMA
/*
-- Join of the two tables
SELECT d.Symbol
        , t.QuoteDate
        , d.ClosePrice
        , FiveDayMA
        , SumDay20MA
FROM #Sum5DayMA d
INNER JOIN #Sum20DayMA t ON d.Symbol = t.Symbol
AND d.QuoteDate = t.QuoteDate
WHERE FiveDayMA < SumDay20MA -- this is where to detect golden crossing
*/
-- 5 day is below 20 MA
--creating a new temp table to hold the data necessary to determine golden/death crosses
DROP TABLE IF EXISTS #SumGoldenCross
SELECT d.Symbol
        , t.QuoteDate
        , d.ClosePrice
        , LAG(FiveDayMA, 1) OVER(PARTITION BY d.Symbol ORDER BY t.QuoteDate) AS
PrevFiveDayMA
        , FiveDayMA
        , LAG(SumDay20MA, 1) OVER(PARTITION BY d.Symbol ORDER BY t.QuoteDate) AS
PrevSumDay20MA
        , SumDay20MA
INTO #SumGoldenCross
FROM #Sum5DayMA d
INNER JOIN #Sum20DayMA t ON d.Symbol = t.Symbol
AND d.QuoteDate = t.QuoteDate;
--Finding golden crosses/creating golden crosses temp table
DROP TABLE IF EXISTS #GoldenCrosses
SELECT *
```

```
INTO #GoldenCrosses
FROM #SumGoldenCross
WHERE PrevFiveDayMA < PrevSumDay20MA
AND FiveDayMA > SumDay20MA
;
SELECT * FROM #GoldenCrosses;
--Finding death crosses/creating death crosses temp table
DROP TABLE IF EXISTS #DeathCrosses
SELECT *
INTO #DeathCrosses
FROM #SumGoldenCross
WHERE PrevFiveDayMA > PrevSumDay20MA
AND FiveDayMA < SumDay20MA
;
SELECT * FROM #DeathCrosses;
--finding scenarios where a golden cross occurred before a death cross
SELECT g.Symbol
		, g.QuoteDate AS GoldenQuoteDate
		--, g.ClosePrice AS GoldenClosePrice
		--, g.PrevFiveDayMA AS GoldenPrevFiveDayMA
		--, g.FiveDayMA AS GoldenFiveDayMA
		--, g.PrevSumDay20MA AS GoldenPrevSumDay20MA
		--, g.SumDay20MA AS GoldenSumDay20MA
		, d.QuoteDate AS DeathQuoteDate
		--, d.ClosePrice AS DeathClosePrice
		--, d.PrevFiveDayMA AS DeathPrevFiveDayMA
		--, d.FiveDayMA AS DeathFiveDayMA
		--, d.PrevSumDay20MA AS DeathPrevSumDay20MA
		--, d.SumDay20MA AS DeathSumDay20MA
FROM #GoldenCrosses g
INNER JOIN #DeathCrosses d ON g.Symbol = d.Symbol
WHERE g.QuoteDate < d.QuoteDate
ORDER BY g.Symbol;
```

L. GoldenCrossUpdate.sql source code –

```
-- Summary Table
-- date primary key
-- symbol
-- 5 MA
-- 20 MA
-- closing price
-- SAMPLE
SELECT Symbol
	, CAST(QuoteUTCDate As DATE) AS QuoteDate
	, ClosePrice
		, LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
	, LAG(ClosePrice, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose2
	, LAG(ClosePrice, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose3
	, LAG(ClosePrice, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose4
```

```
        , LAG(ClosePrice, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose5
INTO #temp1
FROM dbo.vHistoryQuotesDay
WHERE QuoteUTCDate > '5/07/2021'
AND Symbol = 'AAPL'
-- Temp Table for 5 day MA
DROP TABLE IF EXISTS [5dayMA]
SELECT Symbol
             , CAST(QuoteUTCDate AS DATE) AS QuoteDate
             , ClosePrice
             , LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
             , LAG(ClosePrice, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose2
             , LAG(ClosePrice, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose3
             , LAG(ClosePrice, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose4
             , LAG(ClosePrice, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose5
INTO [5dayMA]
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > '05/19/2021'
SELECT * FROM [5dayMA]
-- 5 Day Moving average calculation SUM
DROP TABLE IF EXISTS [Sum5DayMA]
SELECT *, (ClosePrice + LagQuoteClose1 + LagQuoteClose2 + LagQuoteClose3 + LagQuoteClose4) / 5
As FiveDayMA
INTO [Sum5DayMA]
FROM [5DayMA]
SELECT * FROM [Sum5DayMA]
-- 20 Day Moving Average TEMP table
DROP TABLE IF EXISTS [Temp20MA]
SELECT Symbol
             , CAST(QuoteUTCDate AS DATE) AS QuoteDate
             , ClosePrice
             , LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
             , LAG(ClosePrice, 2) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose2
             , LAG(ClosePrice, 3) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose3
             , LAG(ClosePrice, 4) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose4
             , LAG(ClosePrice, 5) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose5
             , LAG(ClosePrice, 6) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose6
             , LAG(ClosePrice, 7) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose7
             , LAG(ClosePrice, 8) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose8
             , LAG(ClosePrice, 9) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose9
```

```
                    , LAG(ClosePrice, 10) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose10
                    , LAG(ClosePrice, 11) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose11
                    , LAG(ClosePrice, 12) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose12
                    , LAG(ClosePrice, 13) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose13
                    , LAG(ClosePrice, 14) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose14
                    , LAG(ClosePrice, 15) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose15
                    , LAG(ClosePrice, 16) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose16
                    , LAG(ClosePrice, 17) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose17
                    , LAG(ClosePrice, 18) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose18
                    , LAG(ClosePrice, 19) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose19
                    , LAG(ClosePrice, 20) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose20
INTO [Temp20MA]
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > '05/19/2021'
SELECT * FROM [Temp20MA]
-- 20 Day Moving average calculation
DROP TABLE IF EXISTS [Sum20DayMA]
SELECT *, (ClosePrice + LagQuoteClose1 + LagQuoteClose2 + LagQuoteClose3 + LagQuoteClose4 +
LagQuoteClose5 +
          LagQuoteClose6 + LagQuoteClose7 + LagQuoteClose8 + LagQuoteClose9 + LagQuoteClose10 +
LagQuoteClose11 + LagQuoteClose12 +
          LagQuoteClose13 + LagQuoteClose14 + LagQuoteClose15 + LagQuoteClose16 +
LagQuoteClose17 + LagQuoteClose18 + LagQuoteClose19) / 20 AS SumDay20MA
INTO [Sum20DayMA]
FROM [Temp20MA]
SELECT * FROM [Sum20DayMA]
--creating a new table to hold the data necessary to determine golden/death crosses
DROP TABLE IF EXISTS [SumGoldenCross]
SELECT d.Symbol
          , t.QuoteDate
          , d.ClosePrice
          , LAG(FiveDayMA, 1) OVER(PARTITION BY d.Symbol ORDER BY t.QuoteDate) AS
PrevFiveDayMA
          , FiveDayMA
          , LAG(SumDay20MA, 1) OVER(PARTITION BY d.Symbol ORDER BY t.QuoteDate) AS
PrevSumDay20MA
          , SumDay20MA
INTO [SumGoldenCross]
FROM [Sum5DayMA] d
INNER JOIN [Sum20DayMA] t ON d.Symbol = t.Symbol
AND d.QuoteDate = t.QuoteDate;
--Finding golden crosses/creating golden crosses temp table
DROP TABLE IF EXISTS [GoldenCrosses]
SELECT *
INTO [GoldenCrosses]
```

```
FROM [SumGoldenCross]
WHERE PrevFiveDayMA < PrevSumDay20MA
AND FiveDayMA > SumDay20MA
;
SELECT * FROM [GoldenCrosses]
WHERE Symbol = 'AA'
UNION SELECT * FROM DeathCrosses
WHERE Symbol = 'AA'
ORDER BY QuoteDate
SELECT * FROM DeathCrosses
WHERE Symbol = 'AA'
--Finding death crosses/creating death crosses temp table
DROP TABLE IF EXISTS [DeathCrosses]
SELECT *
INTO [DeathCrosses]
FROM [SumGoldenCross]
WHERE PrevFiveDayMA > PrevSumDay20MA
AND FiveDayMA < SumDay20MA
;
SELECT * FROM [DeathCrosses]
-- when golden cross occurs before death cross, ORDER BY their dates
SELECT g.Symbol
        , g.QuoteDate AS GoldenQuoteDate
        , d.QuoteDate AS DeathQuoteDate
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
WHERE g.QuoteDate < d.QuoteDate
ORDER BY g.QuoteDate , d.QuoteDate , g.Symbol
;WITH AllTypes as (SELECT Symbol, QuoteDate as GoldenQuoteDate        , 'Gold' as DateType
FROM GoldenCrosses
UNION
SELECT Symbol, QuoteDate as DeathQuoteDate , 'Death' as DateType
FROM DeathCrosses)
SELECT * FROM AllTypes
ORDER BY Symbol, DateType, GoldenQuoteDate
-- SUM of overall profit
SELECT SUM(h1.ClosePrice - h2.ClosePrice) as Profit      --switched h1 and h2 to get + profit value
        --, h1.HighPrice
        --, h1.ClosePrice
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date) AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date) AND d.Symbol = h2.Symbol
--WHERE h1.QuoteUTCDate > DATEADD(Day, -10, h1.QuoteUTCDate)
-- 5226.0994
--- switch where dc is before gc to test
;WITH TestDates as (SELECT g.Symbol                   -- common table expressions
        , DATEADD(Day, 14, d.QuoteDate) as Next14Days
        , d.QuoteDate AS DeathQuoteDate
        , g.QuoteDate AS GoldenQuoteDate
        FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
WHERE d.QuoteDate < g.QuoteDate)
--ORDER BY g.QuoteDate , d.QuoteDate , g.Symbol
```

```
SELECT td.Symbol
            , MAX(h.HighPrice) as MaxHighPrice                                          -- aggregate function
            , FORMAT(DeathQuoteDate, 'MM-dd-yyyy') as DeathQuoteDate
            , Next14Days
FROM TestDates as td
INNER JOIN [HistoryQuotesDay] h ON td.Symbol = h.Symbol
WHERE h.QuoteUTCDate between td.Next14Days AND h.QuoteUTCDate
GROUP BY td.Symbol, Next14Days, DeathQuoteDate, GoldenQuoteDate
ORDER BY Symbol asc
-- test, and it works
SELECT * FROM [HistoryQuotesDay]
WHERE Symbol = 'AAIC'
AND QuoteUTCDate between '6/4/2021' and '6/18/2021'                    -- test if this is june 30th, to match dc
before gc
SELECT * FROM DailySummary
WHERE Symbol = 'TEL'
AND SummaryDate > '6/17/2021'
-- see if date is higher than previous 10 days
-- have date and trend and join with gc and dc
---tests from daily highs -- x + 10
-- For 13: Stocks that are up (13-week high after 13-week low)
DROP TABLE IF EXISTS [MaxValue13]
SELECT Symbol
    , Max (HighPrice) as MaxValue13
INTO [MaxValue13]
FROM dbo.HistoryQuotesDay
WHERE QuoteUTCDate > DATEADD(Month, -3, QuoteUTCDate)
GROUP BY Symbol
GO
-- look at the dates
-- death: where red goes below the blue it is the short trend
-- next steps: document what went wrong, see dates where you get profit and how far between it is
-- Query to find profit of dc before gc
SELECT g.Symbol
            , d.QuoteDate AS DeathQuoteDate
            , g.QuoteDate AS GoldenQuoteDate
            , h1.HighPrice
            , h1.ClosePrice
            , s.Trend10Day
            , s.AvgVolume10
    -- , LAG(g.QuoteDate, 10) OVER(PARTITION BY g.Symbol ORDER BY h1.QuoteUTCDate) -- trying
to integrate -10 days
            , SUM(h1.ClosePrice - h2.ClosePrice) as Profit
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date)
            AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date)
            AND d.Symbol = h2.Symbol
INNER JOIN DailySummary s
ON CAST(s.SummaryDate as date) = CAST(g.QuoteDate as date)
            AND s.Symbol = g.Symbol
--WHERE h1.QuoteUTCDate > DATEADD(Day, -10, h1.QuoteUTCDate)
            AND d.QuoteDate            < g.QuoteDate
            AND h1.ClosePrice > 50
GROUP BY g.Symbol, g.QuoteDate, d.QuoteDate, h1.ClosePrice, h1.HighPrice, s.Trend10Day,
s.AvgVolume10
ORDER BY Symbol, g.QuoteDate, d.QuoteDate
SELECT * FROM DailySummary
                        --, LAG(ClosePrice, 1) OVER(PARTITION BY Symbol ORDER BY QuoteUTCDate) AS
LagQuoteClose1
--getting average profit based on Trend10Day
SELECT --DISTINCT g.Quotedate, g.Symbol
             s.Trend10Day
            --, LAG(g.QuoteDate, 1) OVER(PARTITION BY g.Symbol ORDER BY g.Symbol, g.QuoteDate)
AS PreviousQuoteDate
            , SUM(h1.ClosePrice - h2.ClosePrice)/SUM(h2.ClosePrice) * 100 as Profit
            --, AVG(h2.ClosePrice - h1.ClosePrice) as Profit gc before dc is negative profit
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date)
            AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date)
            AND d.Symbol = h2.Symbol
INNER JOIN DailySummary s
ON CAST(s.SummaryDate as date) = CAST(g.QuoteDate as date)
            AND s.Symbol = g.Symbol
WHERE h1.QuoteUTCDate > DATEADD(Day, -10, h1.QuoteUTCDate)
            AND d.QuoteDate           < g.QuoteDate
            AND h1.ClosePrice < 50                                          --stocks less than 50
GROUP BY s.Trend10Day --, g.QuoteDate, g.Symbol
---when close price is > $50
-- D 2.349544
-- U 3.869387
---when close price is < $50
-- D 0.342190
-- U 0.760847
 --divide the sum of the profit by the sum of the close price times 100 to get percentages

;WITH NearestDates AS (
SELECT
d.Symbol
, d.QuoteDate As DeathDate
, MIN(g.QuoteDate) As NearestGoldenDate
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date) -- golden cross
value
       AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date)  -- death cross
value
       AND d.Symbol = h2.Symbol
INNER JOIN DailySummary s
ON CAST(s.SummaryDate as date) = CAST(g.QuoteDate as date)
       AND s.Symbol = g.Symbol
```

```
WHERE h1.QuoteUTCDate > DATEADD(month, -3, h1.QuoteUTCDate)
       AND d.QuoteDate     < g.QuoteDate      -- death cross before golden
cross
       AND h1.ClosePrice < 50                           --stocks less than 50
GROUP BY
d.Symbol
, d.QuoteDate
)

SELECT
nd.Symbol,
h2.ClosePrice As DeathPrice,
h1.ClosePrice AS GoldenPrice,
h2.ClosePrice - h1.ClosePrice As Profit
, nd.DeathDate
, nd.NearestGoldenDate
FROM
NearestDates nd
INNER JOIN HistoryQuotesDay h1
ON CAST(nd.NearestGoldenDate AS date) = CAST(h1.QuoteUTCDate as date) -- golden
cross value
       AND nd.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(nd.DeathDate as date) = CAST(h2.QuoteUTCDate as date) -- death cross
value
       AND nd.Symbol = h2.Symbol
WHERE h1.QuoteUTCDate > DATEADD(month, -3, h1.QuoteUTCDate)
       AND h1.ClosePrice < 50                            --stocks less than 50
order by nd.Symbol, nd.DeathDate


 ;WITH NearestDates AS (
SELECT
d.Symbol
, d.QuoteDate As DeathDate
, MIN(g.QuoteDate) As NearestGoldenDate
FROM [GoldenCrosses] g
INNER JOIN [DeathCrosses] d ON g.Symbol = d.Symbol
INNER JOIN HistoryQuotesDay h1
ON CAST(g.QuoteDate as date) = CAST(h1.QuoteUTCDate as date) -- golden cross
value
       AND g.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(d.QuoteDate as date) = CAST(h2.QuoteUTCDate as date)  -- death cross
value
       AND d.Symbol = h2.Symbol
INNER JOIN DailySummary s
ON CAST(s.SummaryDate as date) = CAST(g.QuoteDate as date)
       AND s.Symbol = g.Symbol
WHERE h1.QuoteUTCDate > DATEADD(month, -3, h1.QuoteUTCDate)
       AND d.QuoteDate     < g.QuoteDate              -- death cross before
golden cross
       AND h1.ClosePrice < 50                            --stocks less than 50
GROUP BY
d.Symbol
```

```
, d.QuoteDate
)


SELECT
SUM(h1.ClosePrice - h2.ClosePrice)/SUM(h2.ClosePrice) * 100 as Profit
FROM
NearestDates nd
INNER JOIN HistoryQuotesDay h1
ON CAST(nd.NearestGoldenDate AS date) = CAST(h1.QuoteUTCDate as date) -- golden
cross value
       AND nd.Symbol = h1.Symbol
INNER JOIN HistoryQuotesDay h2
ON CAST(nd.DeathDate as date) = CAST(h2.QuoteUTCDate as date) -- death cross
value
       AND nd.Symbol = h2.Symbol
WHERE h1.QuoteUTCDate > DATEADD(month, -3, h1.QuoteUTCDate)
       AND h1.ClosePrice < 50                             --stocks less than 50
```