When coming up with an idea for my project my first idea to was to make a covered call screener. I have always been very interested in stocks and options, and this project was very easy to choose. My initial submission for my project proposal was to create a project as detailed above, but my plans soon changed. I realized that the bulk of the time in my program went towards pulling the option data, and it seemed a shame to only use that data for covered calls, so I decided to add the 5 other spreads. My project analyzes option spreads in order to determine the expected return from any given spread. A single stock will generally have call and put strikes ranging from a 0% of a stock's current price to about 300% of its current price. As used for my program, a $3 stock may 10 strikes for calls and puts, as well as 5 expirations. This means that a cheap stock may have 100 options available for purchase, and a more expensive stock may have 1000 or more strikes, as AAPL currently has over 1000 strikes available for trading. Calls have three spreads associated with them, covered calls, bull calls, and bear calls. Covered calls are the spread between the price of the stock, and the premium gained from selling a call on that stock. Bull calls are the spread between the purchase of a lower strike call and the sale of a higher priced call. Bear calls are the spread between the purchase of a higher priced call and the sale of a lower priced call. If we assume a stock with 10 call strikes, then there are 10 covered call spreads (one for each strike), 45 bull call strikes 9*(9+1)/2, because there is no spread associated with the highest value, i.e. at the highest strike there is no higher strike with which to create a bull spread, and 45 bear call strikes, using the same formula as before. Thus, even for a $3 stock there are 50 call spreads and an equal amount of put spreads for any given date, meaning that each date has this many spreads. By simple math we find that a very low price stock may have 500 spreads associated with it. Manual analysis of even 500 strikes is highly impractical, especially when you consider that options are actively traded, and their prices vary wildly over any course of time. Additionally, even computationally assisted analysis is limited, as simply searching for spreads with the highest current expected return, assuming constant price, will often lead choosing to spreads

with negative returns. In order to accurately evaluate option spreads we must have some sort of method to predict the stock price at a certain point in the future. Additionally, the difference between profit and loss can be small, and thus we must evaluate very single strike in order to search for favorable bids (sale price) and asks (purchase price). In this way the arbitrage becomes very important, as we must identify opportunities that are not priced into the market. On the whole people assume that the stock market is fairly priced, but if that were true we would not find any sort of unexpected returns based on technical analysis. My program (and all other technical analysis programs) prove this assumption false. The reason for this is twofold: in order for pricing to be as expected, everyone in the market must operate under the same model of expected return. For stock prices this is often the case, however for options with so many methods of return, each individual investor will evaluate profitability substantially differently; secondly in order for markets to be “fairly” priced each instrument must be actively traded. A stock without much volume may not have much trading going on for its derivates (options) and bids and asks may be outdated, allowing for trades that are more profitable than they “should” be. When we can check each individual spread for profit we can find expected profit very quickly. When option trading nearly all positions will lose money, in fact it is commonly cited that 90% of investors lose money options trading. This isn’t surprising given that using my model I predict that about 90% of spreads lose money, and spreads are less likely to lose money than naked options in general. With such dismal odds it becomes readily apparent why an accurate method of return prediction must be created.

I had a challenge building this project, in that it involved a very large number of lists that had to be taken apart and put together in different ways, as well as dealing with a painfully slow API that forced me to minimize my requests to the network. When structuring this program I initially built it to only compute covered calls, as I intended this program to generate data about

covered calls and then display them to a website. As I progressed I realized that most of the work had been in building the data structure to deal with all of the data and so I built out the other 5 spreads as well. When building out those 5 spreads I soon realized that it would take 1000 lines of code to properly analyze the spreads, and the website would add another layer of complexity to my project with a deadline looming, so I turned the project into a csv generator instead of a web based screener, since only one of the screens that I had planned to offer (the predicted return screen) really mattered anyway. My program generates every single spread possible and then stores them in a list ordered as [allstocks][symbol][expiration][spreads from given strike][individual spread]. This structure was created in order to allowing reasonable viewing and sorting from a web application. While creating a web application was no longer the goal, sorting the data in such a manner was still useful in its ease of using the map function, as well as producing a cohesive set of data to do operations on in case I ever decide to expand this project. This several layer list did prove to be challenging for sorting based off of predicted return, as it had to be flattened before it could be sorted, and since covered calls and covered puts had a different level of depth I developed a recursive list unpacker to deal with lists of any depth. This unpacker also had to return the an array of arrays, so it could not unpack all the way to where no arrays and added some complexity.

I experienced moderate difficulty with this project. The most irritating of the things I encountered was the difficultly of setting up numPy and sciPy to run on my machine. Typical python implements numPy and sciPy using math functions from the operating system, but apparently since I was using an AMD windows laptop those functions were not available, as I got an error relating to lack of BLAS and LAPACK. There was no option to download the required libraries for my machine, and so I had the choice between building the libraries or using WinPython. I chose to use WinPython, and I then had to deal with setting all of my paths which

turned out to be more annoying than anticipated, but in the end I finally got my python set up, and I was ready to begin my project. Aside from technical considerations there are also organizational considerations when producing such a large project, and I found myself having to restructure large portions of my program partway through, when I changed my goals from creating a website to interact with simple data, to creating complex data structures without a website. While I had initially planned on structing my program as several files of scripts that all passed data to a web server, I realized that it would be more efficient to construct all of my data structures within one script. Debugging was also an issue, since there were generally no benchmarks to test my code against, and I was forced to create several tests to make sure that my mathematical models produced the desired results. Even a small error will lead to useless results, so it is critical that all the math be correct. To the best of my knowledge the math is currently correct, but honestly a flaw of my program is that the mathematical functions are a bit of a black box. Aside from that I had some annoying typecasting errors, and I inefficiently copied my lists which contained all of my data into numPy arrays so that I could access some useful helper functions. Honestly this decision was out of necessity rather than consideration for what is the “best” programming practices, since I was running out of time to complete my program, and I had to get it to work. There are some other instances of “sloppy” coding in my program, but on the whole I tried to keep it as pythonic as possible. Finally, I don’t know if my network is bad, or if the API detected that I was making thousands of requests a day, but my connection to the API has been getting forcibly closed lately, and I actually haven’t been able to pull real time data in a while, so I am relying on old data to test it.

In completing this project I had to learn the python programming language. I had only used python very minimally before this project (maybe 10 lines of code for a small assignment, some sort of hello world). Python is rather different than any language that I have worked with

in depth before, such as C++ or Java, due to its dynamic typing, its lack of compilation, and its high level of abstraction. Python relies heavily on built in methods to deal with iterables, such as map and reduce, which is rather different than loops. Python functions can act somewhat as Classes when they are lambas or closures, and that presents a new way of thinking about programming. Technically C++ has closures, but I've never used them in the way that you are expected to in python. Syntactically python also has significant differences from other languages that I have worked with, given that type declarations are not needed, and that semicolons are not used, but rather indenting is highly important. Learning this new language was helped by my understanding of data structures and algorithms, but it still presented a challenge in learning a new way of thinking about programming. I would assert that it is easier to learn how to poorly code in python than in c plus plus, but I also believe that the strict form of c plus plus gives guidance on proper coding practice in a way that python does not. This is really to say that since python is very expansive and without constraints, and with a high level of abstraction, it can be hard to know if you are doing something in a good way or a bad one. This is not say that I did not enjoy learning python, and in fact I would say that coding python filled me with a much lower amount of hatred then coding c plus plus, in which compilation errors due to type casting make up the majority of my debugging. While you might assume that dynamic typing would lead to confusion and require additional code it is really quite convenient and makes nearly all type declarations irrelevant. I suspect that I may not have followed best practices at all times, since I read somewhere that scripting languages generally require tests in order to ensure that they function properly, but I only tested things when I thought they were broken, and did not systemically generate any tests for my program.

If I had to do this program over again I would change a lot. A large portion of the code in my project (about 50%) is close to copied versions of other functions with small portions of it

changed, mainly for math purposes. I believe that this close to duplicate code could be decreased by making functions that return other functions, such that I could define new helper methods with arguments, rather than redefining the entire method. If I were to do this project again I also believe that I could actually fulfill the scope of my original project by creating a website capable of screening stocks, options, and spreads. I deliberately added complexity to my program in order to make it easier to parse later, but I never got the chance because I did not anticipate the difficulty. Honestly, I think that to implement the front end would probably take another several hundred lines of code, and it would be much less straightforward then the backend, but I just don't think I fully understood the difficulty of making a program with so many moving parts when I first started.