(Revised) Product Backlog:

1. Real-Time Tweet Scraping: Tweets regarding a pre-programmed list of companies will be scraped from Twitter in real time using the Tweepy Stream API functionality.
   - Complexity: Complex. While there is an API to help ease the process of scraping the tweets, the process is made more difficult by the following factors: finding appropriate filters for the tweets, discovering what functionality is offered to a free developer account on Twitter (the API is rather hazy in that regard) and the general hassle of obtaining a Twitter Developer License and using it made this task less than easy.
   - Estimated Time Complextity: 2 weeks. The backend team (Jonathan, Drake) made this their main priority for the project, as the "sentiment" of the Sentimental Stocks title depended on the tweets scraped from Twitter. As such, it was completed faster than it would otherwise.
2. Sentiment Analysis of Tweet Text: The scraped tweets, after acquired and stored, are analyzed for sentiment (positive, negative, neutral) by VADER sentiment analysis and then smoothed.
   - Complexity:Mildly Complex. The analysis takes place via a single API call that takes in the text and returns a float sentiment (-1 to 1, where -1 is negative sentiment, 0 is neutral, 1 is positive sentiment). However, to help streamline the data, we compressed 100 tweet sentiments into 1 data point by extracting the sentiments from the database and applying a custom smoothing algorithm to them.
   - Estimated Time Complexity: 1 week. This was done after the twitter scraping had been completed, the database had been established, and the server was set up. The actual implementation was simple, and the design of the smoothing algorithm was to take the average of the past hour of tweets.
3. Stock Value Scraping: Stock prices were scraped from a pre-programmed list of companies to compare to the sentiment values obtained from tweets.
   - Complexity: Mildly complex. The API for the stock prices needed to be accessed to obtain the values. We used Alphavantage, which had the advantage of clear documentation to use. Thus, the problem was made much easier than it could have been.
   - Estimated Time Complexity: 3 days. The API of Alphavantage was very cooperative, leading us to assign less time to this than the Twitter scraping.
4. Database Storage for Tweet Sentiments and Stock Values: The sentiments of the tweets for each company and day are stored in a PostgreSQL database, and used for calculations by the front end.
   - Complexity: Complex. The team needed to develop a database, API for interacting with the database, and then reference that API after: a) VADER analyzed a tweet and b) a stock price was supplied by the stock value scraper. This was made more complex by the sheer amount of tweets taken in every day (according to measurements, around 500,000), and means were taken to lessen the time needed for database operations. In addition, the database stored on a web server, so the calls to the database consisted of POST requests to the server. Finally, the database is regularly cleaned so as not to clutter up storage space and increase load times for the server and front end.
   - Estimated Time Complexity: 1 week. This task was the initial main focus of the database team (Sandy), as the database was needed to then reference for the front-end. The amount

of features that needed to be added to the database increased the man-hours required, and the time complexity turned out to be more than estimated.

5. Display of Tweet Sentiments and Stock Prices: The sentiments and stock prices obtained in earlier features and stored in a database were then plotted on a nginx hosted website with url https://sentocks.sandyuraz.com.
    - Complexity: Complex. The team needed a website that would host the plots of the graphs obtained from the database mentioned previously, and continually update itself as new data was taken in. The web server was set up by hand and all of the coding was done in Vue.
    - Estimated Time Complexity: 3 weeks. This was the task the front-end (Dylan, Ross) was mainly focused on, as it was how we would communicate our data to the world and served as the main purpose of our project: to analyze whether there was a correlation between stock prices and tweet sentiments about the stock on twitter.
6. POTUS sentiment analyzer: As a fun aside, we used the same methodology to scrape sentiments about the POTUS party from twitter.
    - Complexity: Simple. With the underlying structure already in place for analysis of tweets given keywords, all that was needed was a new list of keywords to begin the sentiment analysis.
    - Estimated Time Complexity: 1 week. While this aspect of the project was simple, it also had low priority and was mainly meant as a fun side project that built off of the stock analysis. Thus, it was only worked on when no other assignments were given.

Sprint Backlog:

1. Tweet Scraping: Tweets regarding a pre-programmed list of companies will be scraped from Twitter using the Tweepy Search API functionality.
    - Complexity: Mildly Complex. This was the initial project of the back-end (Drake, Jonathan). At the start of program run-time, tweets would be scraped from Twitter using tweepy's search API. The tweets would be found from the past week, and would look at the sentiment of the tweet if one of the pre-programmed companies was found in the text.
    - Estimated Time Complexity: 1 week. The tweepy API was helpful in obtaining the needed tweets, but Twitter's rate limit made testing the program's validity difficult.
2. Database Storage:
    - Complexity: Mildly Complex. This was the project of the database team (Sandy) and dealt with storing the tweet text and sentiment analyzed by VADER. This database was stored on a web server, so POST requests were required to access the data.
    - Estimated Time Complexity: 1 week. Due to Sandy's experience in developing databases, the estimated time on this aspect of the project was short.
3. Sentiment Analysis of Tweet Text: The scraped tweets, after acquired and stored, are analyzed for sentiment (positive, negative, neutral) by VADER sentiment analysis.
    - Complexity: Simple. The analysis takes place via a single API call that takes in the text and returns a float sentiment (-1 to 1, where -1 is negative sentiment, 0 is neutral, 1 is positive sentiment).

    - Estimated Time Complexity: 1 week. This was done alongside the tweet scraping, and was only a few extra lines of code added to the tweet scraper.
4. Display of Tweet Sentiments: The sentiments obtained from Twitter and stored in a database were then plotted on a nginx hosted website that was accessed through IP address.
    - Complexity: Complex. The team needed a website that would host the plots of the graphs obtained from the database mentioned previously, and continually update itself when commanded. The web server was set up by hand and all of the coding was done in Vue.
    - Estimated Time Complexity: 1 week. This was the task the front-end (Dylan, Ross) was mainly focused on, as it was how we would communicate our data to the world. It was only supposed to be a simple website that would display a graph and nothing more.

Meeting Log:

NOTE: All meetings were held on either a custom Slack Channel (20-3-25 to 20-4-8) or a Discord chat (20-4-8 onward)

- 20-3-25: Initial Project Meeting. The team met and discussed what project they would like to work on. In addition, meeting times were set for 11:00 Monday, Wednesday, and Fridays of every week.
- 20-3-27: Project Topic Discussion. The team went over a potential idea for a project: An application that would access Spotify and Apple Music's API to determine the "compatibility" of a playlist - If someone were to move from Spotify to Apple Music and vice versa, how many previous songs would they have available to them?
- 20-3-30: Project Topic Discussion. Problems were found with the original idea, and several new ideas were proposed. The team took a vote, and the Sentocks idea won out.
- 20-4-1: Prototype Planning. The prototype features of the project (seen in the Sprint Backlog) were fleshed out, and the initial tasks were assigned. These tasks were assigned a week-long completion date.
- 20-4-3: Prototype Planning. Any issues encountered with the prototype development were discussed, and questions geared towards other sub-teams were asked, like 'how do you want the data to send to the database formatted? What API calls should I make to the front-end?' and so on.
- 20-4-6: Prototype Planning. Not much occured in this meeting, as several people failed to show. Meeting was ended early.
- 20-4-8: Reports coming back from the initial assignments were given. The initial assignments were completed, and the prototype was finished. Speculation on what the next features of the project should be was undertaken. Meetings were moved to Discord as a method of further reminders (Slack was reminding us too little).
- 20-4-10: Few people showed up to this meeting. A possible rescheduling to 4:00 MWF was suggested, and was taken up next meeting.
- 20-4-13: New ideas for the project were given, including a real-time aspect. In addition, the database had grown much slower due to the massive amounts of data obtained from the project. Restructuring of the database and a shift to real-time data analysis was assigned.
- 20-4-15: Few showed to this meeting. Two members of the team discussed math homework for another class, as no new information or questions were presented.

- 20-4-17: This meeting was mainly a discussion between the database team and the front-end team about specific problems arising from Docker and the REST api of the web server. We discovered the screen sharing feature in Discord.
- 20-4-20: This meeting was predominantly questions about the real-time analysis and demonstrations of the real-time analysis of tweets. Various names were given to features, and one member of the team assigned himself the task of making the code more modular (as per the architecture design posted on the Github repository).
- 20-4-22: This meeting had few individuals, and math homework was again discussed. A suggestion was given to have less meetings.
- 20-4-24: This meeting was predominantly between the database and front-end teams, who tried to solve a problem occuring with the web server interactions with Docker.
- 20-4-27: We only talked about the documentation at this point, and where to place it.
- 20-4-29: We congratulated ourselves for a job well done, then proceeded to discuss what we would do for project 3. We made a release for the completed project, and discussed creating a presentation for our project.
- 20-5-1: We discussed the presentation and made a rehearsal. Later that night, we recorded the entire presentation.
- 20-5-4: We discussed the remainder of the items we have to complete for the project, which included the user manual.
- 20-5-6: We added the documentation to our master branch.