**Executive Summary**

At the start of this project, we set out to discover relationships between Robinhood user behavior data and stock prices, hoping to either find correlations that would help us predict good investment practices for stockholders, or to implicate Robinhood trends as a bad predictor for investing. To do this, we chose two datasets that would provide a significant amount of information on the two areas of observation and analysis--Robinhood users and stock prices. We combined these datasets to create one large dataset representing the number of Robinhood users holding each stock on a given day and the corresponding market price.

We researched and tested a range of high-level tools for use in cleaning, analysis, and backtesting. During the final stages of the project, we compared the price predictions of different models that were trained with either price data itself, or with a series of different variables in order to see if robinhood user popularity was a good indicator for price prediction of stocks that had already been selected for having a high correlation between their price and the Robinhood user popularity (+/- 0.845).

There were four general stages to this project: preprocessing, correlation analysis, modeling, and interpretation of results. Various methods were researched and used for cleaning, which proved to be challenging. Time-series data requires a complicated vetting process, especially for use in graphing and modeling. Once preprocessing had been finished, we performed our correlation analysis using Pearson's correlation coefficient. We then used the Long Short Term Memory (LSTM) neural network and K-Nearest Neighbor Clustering to model behavior and price data.

The final results derived from the models showed us that Robinhood user data was a bad predictor of price, even when the stocks had shown a high historical correlation. The predictions of a multivariate LSTM that included Robinhood users holding data were far less accurate than those made by a univariate LSTM based solely on stock price. Training the model on past users holding data was quantitatively unhelpful in predicting stock prices, whereas simply training the model on price provided an extremely accurate prediction of future stock prices.

*Improvements after the last presentation:*

Most of our results were available before the last presentation. However, K-Nearest Neighbor Clustering had not been implemented yet, and we took some time to apply this model after the presentation. Similar to what occurred with the LSTMs, our results did not change after implementing K-Nearest-Neighbor, as price showed to be a better predictor of future price.

**Problem Statement | Data | Baselines**

As mentioned in earlier presentations, Robinhood users are notorious for following hyped stocks, and consequently, they are viewed as amatuer investors who consistently make ill-advised decisions with their money. Our goal in this project was to ascertain whether or not stock prices have any correlation (positive or negative) with Robinhood user behavior, and to then build models that would prove whether or not Robinhood behavior was a good predictor of

a stock's price. As discussed above, our approach was broken down into four stages. The primary stages for the problem itself, however, were correlation analysis and modeling.

Correlation analysis was used to vet the stocks for those with a high correlation. Our threshold was very high (+/- 0.845). In this way, we were able to single out 115 stocks that had a very good correlation between Robinhood user popularity and price--stocks that would be relevant for use in modeling.

Modeling was our main source for results. We used LSTM and K-Nearest Neighbor Clustering models with 80:20 train/test partitions to predict stock prices based on certain variables. In the case of LSTM, we compared two approaches to ascertain whether or not Robinhood user popularity played a useful role in predicting stock prices. The results helped us understand Robinhood's user behaviour significance in the stock market. These will be covered in a later section.

As described in the introduction, our dataset was made up of two separate datasets found on Kaggle:

- https://www.kaggle.com/cprimozi/robinhood-stock-popularity-history
- https://www.kaggle.com/tsaustin/us-historical-stock-prices-with-earnings-data

The first was a collection of hourly holding records for Robinhood users, grouped by stock. These records were concatenated into a single file holding all Robinhood user data, and later resampled daily. The second dataset was a large file containing stock price, volume, and opening and closing price for each stock, along with a few other unneeded features. We combined this and the Robinhood file to create one massive dataset with the proper formatting. The rest of our analysis was performed on this conglomerate file.

Our baseline was simple but effective: historical price data. Since we were trying to ascertain Robinhood's use as a price predictor, we compared the test data from our models to actual price data from those stocks. This can be observed in the **Results** section.

**Approaches | Algorithms | Tools**

Our initial strategy for tackling the project was to combine the datasets and then begin work on correlation analysis, moving averages, and other components of the next stage. Memory issues forced us to start on the analysis before the datasets were in one compact file. We split responsibilities--one of us working on data combination, and the other on trend analysis. Even after the data was combined, cleaning took up an exorbitant amount of time and resources, as time-series data requires some special treatment. Eventually, we were able to use the timestamps included in the datasets as indices and utilize the **datetime** library to aggregate hourly data (in the case of the Robinhood dataset) to daily. The final combined dataset used stock tickers as keys, included EMA (Exponential Moving Average of the popularity) calculated via Pandas, and showed all useful columns from both datasets.

While we researched and developed moving average and exponential moving average methods, we found the built-in Pandas libraries more convenient for our purposes. We also considered various techniques for implementing correlation analysis and decided upon Pearson's correlation coefficient, also a Pandas feature, to help us proon the dataset.

Once we could calculate the correlation between various stocks and their corresponding Robinhood user data, we wanted to narrow our window of inspection to only highly correlated stocks. As discussed earlier, we chose a high threshold of +/- 0.845 to eliminate all stocks that did not have a clear connection to Robinhood user behavior. The result was a group of 115 stocks we could perform detailed analysis on.

Various models were inspected for use in analyzing the correlated stocks, including Linear Regression, Auto ARIMA, and SVM. Our final selections were LSTM and K-Nearest Neighbor Clustering. K-Nearest Neighbor Clustering is a fairly simplistic but useful way to analyze trends in non-stationary data, and LSTM neural networks are perhaps the most accurate stock price predictors in the arena. LSTM is a complicated model to implement and requires intensive formatting and data shaping. Because of this, we researched various implementations, and those seen in the source code are Singh's (univariate) and Brownlee's (multivariate). We adapted these to our dataset through experimentation and testing.

Singh's univariate approach:
https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/

Brownlee's multivariate approach:
https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

*Data Pipeline*

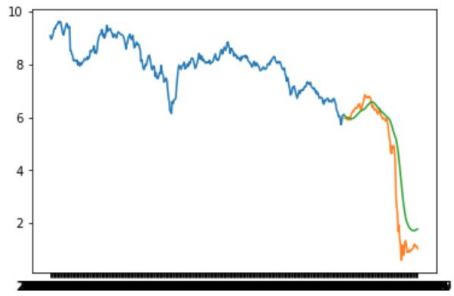
1. Read popularity files, resample them, and concatenate them to a single file
2. Load both price and popularity data
3. Merge the datasets
4. Group by symbol and measure the historical correlation of each
5. Filter the dataframe for stocks with a high correlation (+/- 0.845), which was chosen with the intention of building our own index
6. Feed the processed data to the models

Tools may have caused us the most grief in this project. Inexperienced in using AWS, we were forced to spend much of our time testing various tools and trying to give our instances the memory we needed to work with a large dataset. As we had to perform some large-scale computations without parallelization, we needed tools that could handle keeping at least 5 gigs of data in memory constantly. We spun up an EMR instance to provide for this, but we also used SageMaker Lab and Notebook for developing the source code. Jupyter was useful for testing small chunks of code or training models on single-stock samples of the larger dataset. We considered using Zipline and Quantopian for backtesting, but we chose to create an index from our 115 stocks and compare against our baselines instead.

**Results**

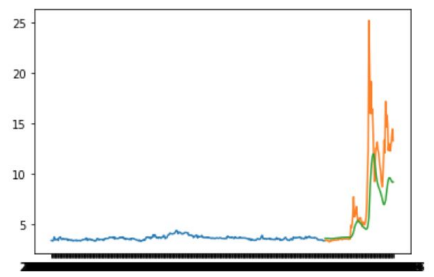
Our results were easy to visualize by plotting model performance. They are presented below:

RMSE = 0.9593156724329953



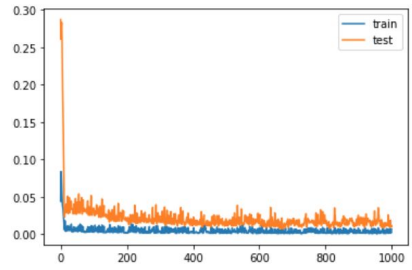
Stock: CEN | Univariate LSTM

RMSE = 3.7566068880840042



Stock: APT | Univariate LSTM

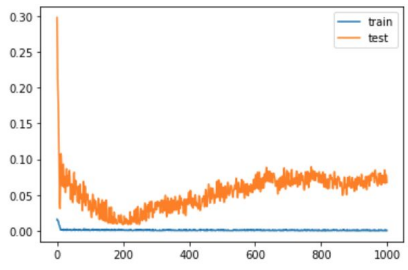
Epoch 1000/1000
8/8 - 0s - loss: 0.0066 - val_loss: 0.0108




RMSE = 1712.8453

Multivariate LSTM

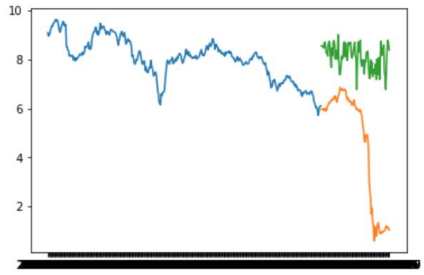
Epoch 1000/1000
8/8 - 0s - loss: 3.8514e-04 - val_loss: 0.0676




RMSE = 35630.48

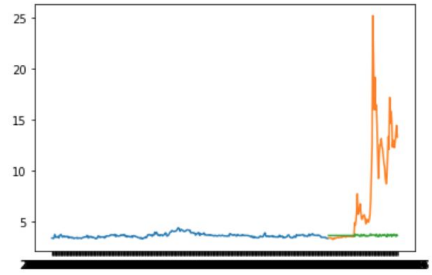
Multivariate LSTM

RMSE = 4.101499186279391



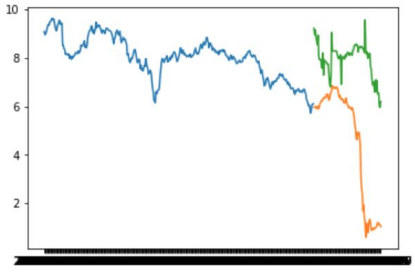
K-Nearest Neighbor with Price

RMSE = 6.471844199668853



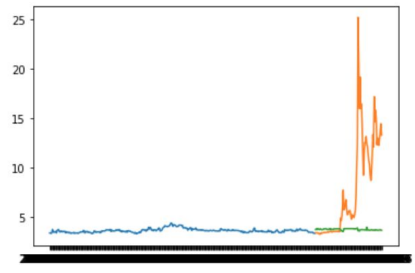
K-Nearest Neighbor with Price

RMSE = 3.9289020676965856



K-Nearest Neighbor with Robinhood Data

RMSE = 6.4469574741427



K-Nearest Neighbor with Robinhood Data

Two stocks (CEN and APT) are shown here, with univariate LSTM; multivariate LSTM; K-Nearest Neighbors using price and volume; and K-Nearest Neighbors using price and the number of Robinhood users holding modeled respectively. Except in the case of the multivariate LSTMs, which depict loss, the blue lines show train data, the orange show the price baseline, and the green show model predictions. The multivariate LSTMs are trained using Robinhood data, as are the secondary K-Nearest Neighbors models. Trying to train LSTMs for price predictions on Robinhood user data is clearly inadvisable, as the RMSE is very high. For K-Nearest Neighbors, the RMSE is barely lower when simply using price and volume as predictors. In short, modeling with these variables shows that Robinhood user data is not a useful predictor for stock prices.

**Conclusions**

One of the primary challenges in completing this project was the type of data we worked with. Time-series data is notoriously challenging to clean, analyze, and model. Discrepancies can be hard to pin down and even harder to deal with, and a large amount of formatting is often required to perform the simplest analysis. Plotting time-series data is something of a nightmare in Pandas libraries, and we spent much time solving problems related to this. Every algorithm or model we implemented demanded more than the usual amount of research because not only did we have to understand how to write the algorithm or model, but we had to find a resource that explained how to do it specifically with time-series data.

Another challenge presented itself when we tried to merge the datasets. We were forced to pivot between a high-power EMR instance and Jupyter Notebooks constantly to deal with memory issues. The notebook kernel itself could not handle certain computations unless MapReduce methods were used. This and other memory issues cost us much time in experimentation and research.

Overall, the project was a success, and we learned useful skills. In future we would prefer to take on a smaller project without so many inherent complications. Sometimes there is no simple way of solving a problem, and this was certainly true of ours. However, we were able to familiarize ourselves with AWS, SageMaker Notebook and Lab, and various Pandas and Sklearn libraries. We also gained valuable experience in cleaning and working with big data. Implementing complicated models like LSTM gave us experience with neural networks and visualizing modeled data. While our lack of experience was not ideal, we overcame the setbacks and roadblocks to learn new skills and achieve meaningful results.