Report for CS7646 Project 6: Manual Strategy

Michael Tong (mtong31)

Part 1: Technical Indicators

For the first portion of this report, a discussion of five technical indicators will be made, namely, the simple moving average (SMA), Bollinger Bands (BB), exponential moving average (EMA), moving average convergence divergence (MACD), and the stoichastic oscillator (SOSC). Implementation of these indicators into a trading strategy will be discussed in the following sections.

The first technical indicator, simple moving average (SMA), is aptly named as it is simply a rolling mean across data points. A rolling mean consists of data and a window, and returns another dataset. The returned dataset points represent the average of values prior to that point, where the number of points to average over is the window (often in the notation of X-day SMA, where X is the window length). When this model is applied, it suppresses outliers in the dataset by diluting their overall impact. The SMA is commonly used as a means to apply other technical indicators and to return a smoother trend of data. There are some anecdotal trends that are identified through the SMA, such as the “death cross”, which represents a bearish signal, and occurs when the 50-day crosses the 200-day SMA. Implementation of the SMA is also characterized by its name, due to the pandas library providing a specific method for applying a SMA to a pandas dataframe. The pandas method used is pandas.DataFrame.rolling(window).mean(). Implementation of this method on the JPM stock data is shown below.

```
def simple_ma(df, window):
    return(df.rolling(window, center=False).mean())
```

***Code 1**: Pandas implementation of SMA*

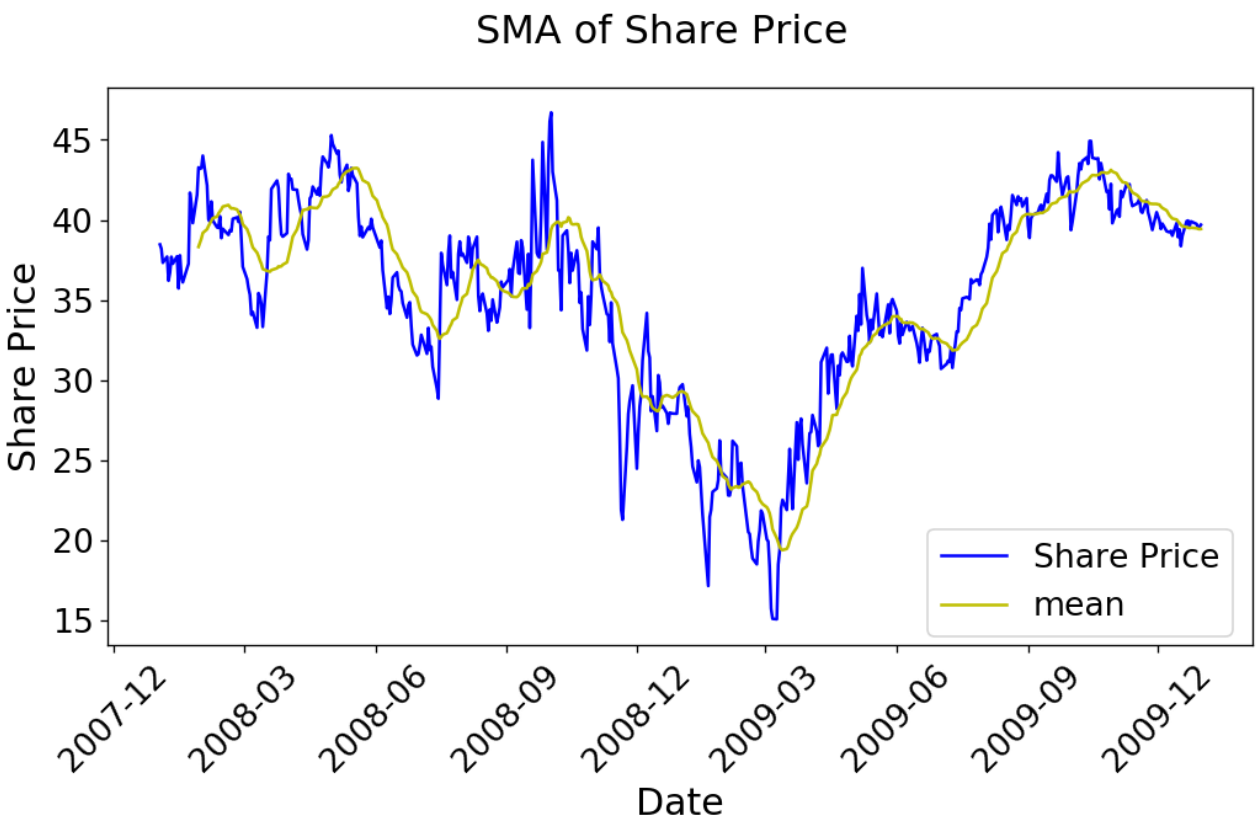

***Figure 1**: 19 Day SMA for JPM*

As an example of the SMA being a foundation for other indicators, the Bollinger Band® (BB) indicator is one such method. BBs are a commonly used indicator which generates two additional datasets to represent the SMA adjusted by the standard deviation. What manifests is two SMA trends above and below the SMA, where the separation from the SMA is a proportion of the rolling standard deviation. Commonly, the standard deviation used is two, both above and below. The rolling standard deviation is found similar to the rolling mean, using the pandas library shown below, with its associated plot. BBs historically serve two purposes, a visual of volatility (bands are highly separated) , and to determine "breakout" prices, where the price rises above the upper band, or drops below the lower band, signifying potential purchases or selling points.

```
def bollinger_bands(df, window, threshold):
    std = df.rolling(window, center=False).std()
    mean = simple_ma(df, window)

    upper_band = mean + threshold * std
    lower_band = mean - threshold * std
    return(mean, upper_band, lower_band)
```

***Code 2**: Bollinger Band Implementation*

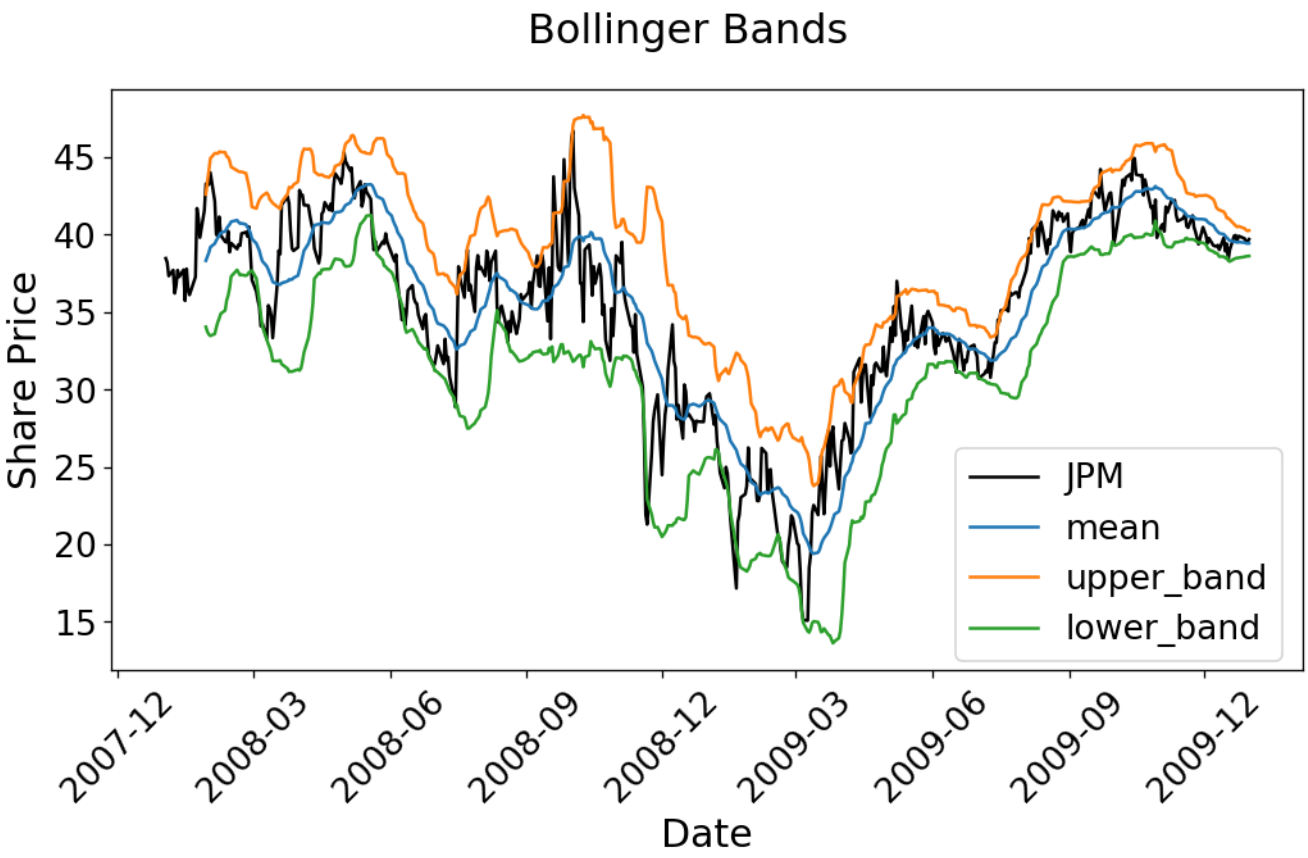

***Figure 2**: 19 Day Bollinger Bands and SMA For JPM (note the 1.45 std deviation as opposed to 2)*

SMA is not the only popular method to average datasets for technical indicators. The Exponential Moving Average (EMA) is another method of producing a “rolling mean” trend. EMA is similar to the SMA in the rolling regard, however, EMA biases recent data more heavily, meaning that recent price changes have a larger impact on the moving average. The EMA forms a similar plot trend to the SMA, and is used in similar regards. Typically the EMA notation is EMA-X, where X is the number of days, or the window period. The EMA is often used as a foundation for other technical indicators as well, which will be discussed in succeeding sections. Implementation of the EMA through Python is once again through the pandas library, shown below, with it’s respective plot for the JPM ticker.

```
def exp_ma(df, days):
    return(pd.ewma(df.copy(),com=((days-1)/2)))
```

***Code 3**: Exponential Moving Average Implementation*

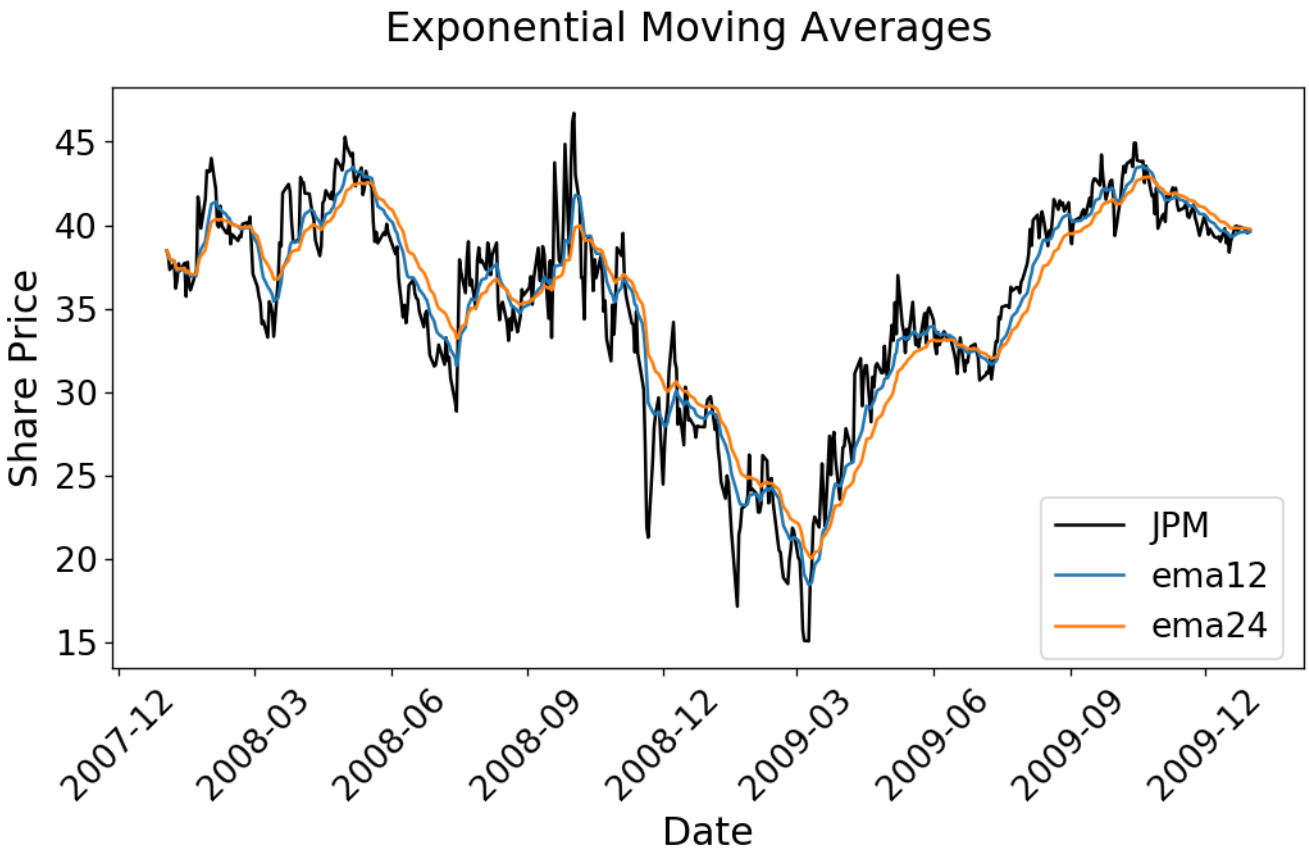

***Figure 3**: EMA-12 and EMA- 24 for JPM*

An application of the EMA can be found in the Moving Average Convergence Divergence (MACD), which is classified as a momentum indicator. As such, it seeks to determine how “strong” a price point is moving, through the comparison of two moving average points. For the MACD, an EMA is used to determine two components, the MACD and the MACD signal. The signal is simply a moving average of the MACD, which is utilized as a indicator of strong movements if crossed. The traditional MACD windows used are 26 and 12 day EMAs. Therefore the MACD is simply calculated as the difference of the two EMAs, and the MACD signal is determined as a moving average of those points. The Python implementation and plot is provided below.

```python
def MACD(df, ema1_days=12, ema2_days=26, macd_signal_days=9):
    ema1 = exp_ma(df, ema1_days)
    ema2 = exp_ma(df, ema2_days)

    macd = ema1-ema2
    macd_signal = exp_ma(macd, macd_signal_days)
    return(macd, macd_signal)
```

***Code 4**: MACD Implementation*

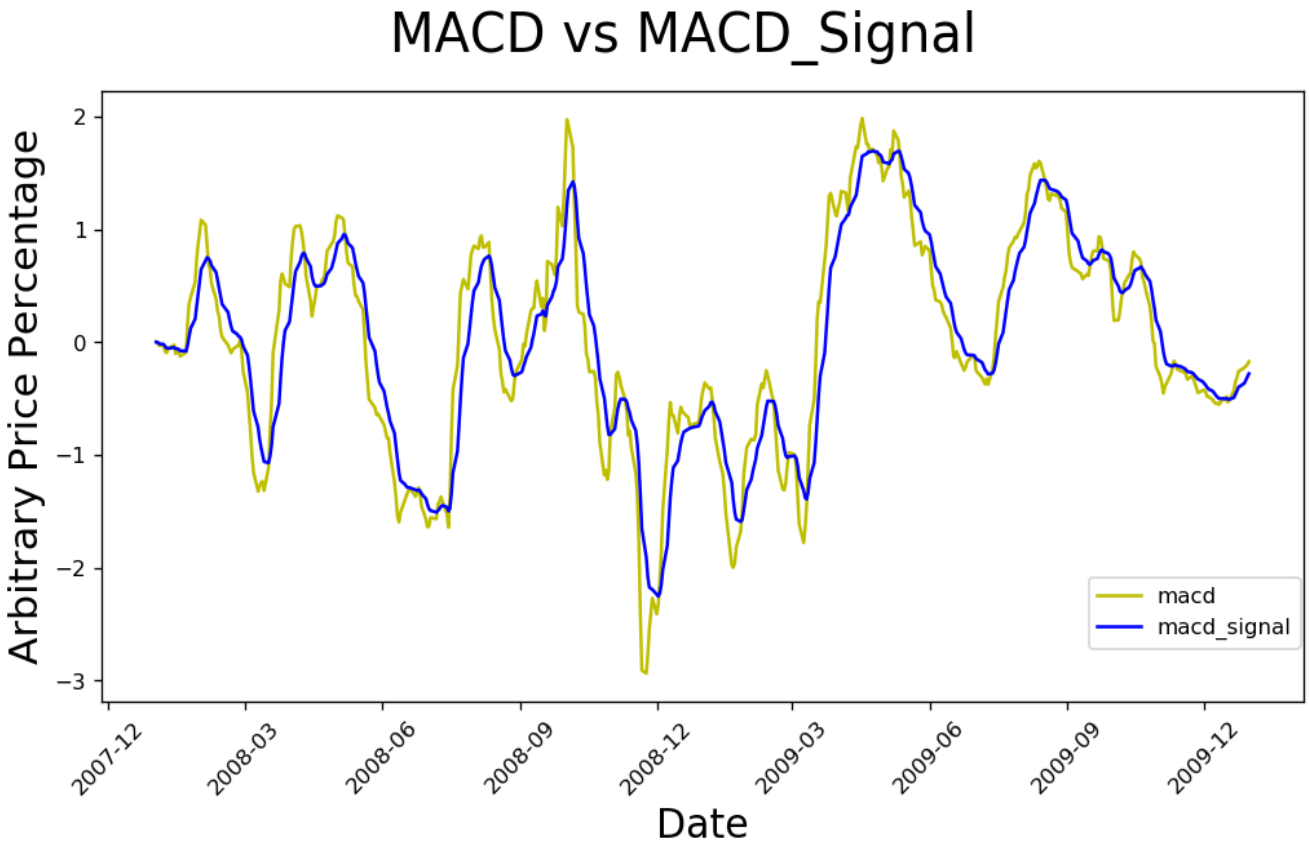

***Figure 4**: 12/26 day MACD Plot for JPM with a 9 Day Signal*

Another momentum indicator implemented for this strategy is the Stochastic Oscillator (SOSC). Being an oscillator, this analytic method returns normalized values scaled between 0 to 100. Rather than returning specific price points, this tool is a relative measure of where a point lies with respect to the rolling window's high and low prices. This value is captured commonly as the variable "K", and analysts typically further average the "K" value over a short period to dilute any extreme values (eg. the min or max of the window), which is represented as "D". The traditional moving average window for D is three days, whereas the min and max window is 14 days. Often, the SOSC is used to determine when a particular price point is oversold or overbought, which is traditionally represented as K or D values below 20 and above 80 respectively. To find the SOSC values at specific data points, a rolling min and max is first calculated, and each point within the window is compared to those values using the formal expression shown below:

$$K = 100 * \frac{(\text{Adj\_close} - \text{Low}_{\text{window}})}{(\text{High}_{\text{window}} - \text{Low}_{\text{window}})}$$

Implementation in Python surprising utilizes the Pandas library once again, where the rolling min and max functions are calculated.

```
def stoc_osc(df, k_window=14, d_window=3):
    high = pd.rolling_max(df, window=k_window)
    low = pd.rolling_min(df, window=k_window)
    K = 100*(df-low)/(high-low)
    D = pd.rolling_mean(K,window=d_window)
    return(K,D)
```

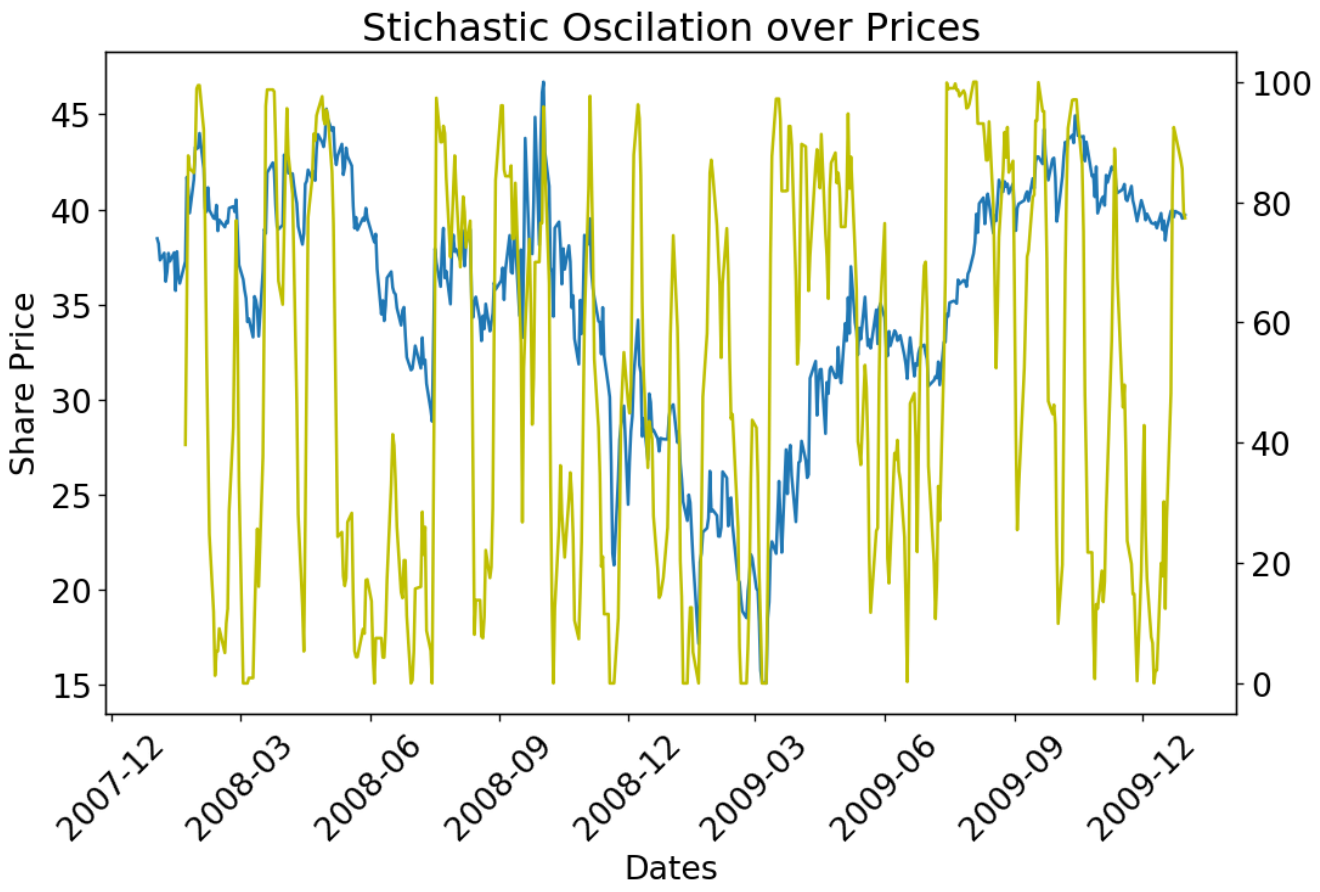

***Figure 5**: Stochastic Oscillator "D" Value plotted over the JPM Share Price*

Part 2: Best Possible Strategy (BPS)

For our future manual strategy discussion, a performance comparison is generated by creating a function which returns a list of trades that maximizes returns for a given share over a period of time. Historic adjusted close prices are used, and the concept implemented is that a share is longed prior to a rise, and shorted prior to a descent. To generate these trades, the difference between successive days was first determined, which represents if a share increase or decreased in value in the following day. The Python implementation used to determine this is a difference between the share price data and a shift of the share price data by one day. Once we have the price difference between each successive day, the actual difference is not relevant, so for my implementation, I had normalized the values to either 1 or -1, representing increases and decreases respectively. Since there may exist successive dates where the price does not change, the returned value is zero, which in my implementation returns a nan value due to the normalization process used. To avoid future issues, these nan values must be accounted for. Since the eventual goal of this process is to determine inflection points in the price data, that is, when the price difference changes from positive to negative or vice-vera, a backfill for the price difference can be used. A forward fill method was not employed due to it potentially creating a signal on the last day, if it were to be a zero difference day, thus creating information.
With our price difference dataframe, the goal is to trade on instances where the values change sign; the inflection points. This is performed by comparing the a date's signage with its previous date. If the sign changes, then a order is made. This is performed in the Python API through initializing the first date to be either a long or short, keeping track of what that position is, then looping through the rest of the dates and performing the opposite action when an inflection point is found. Since the project constraints are a maximum position of 1000 shares, the initial long/short is a 1000 share trade, and all future trades are either to long/short 2000 shares. Once all dates are iterated through, the final position is forced to maintain its position since the following date movement is unknown.

Once this strategy is implemented on a particular stock symbol, it returns a dataframe of the optimal trades to maximize the return for each price movement. The dataframe is a single column for this project, which contains an index of all the dates of a given range, and a column which represents purchases and sells as a positive or negative value respectively. When this trading dataframe is implemented into our market simulator (previous project, modified slightly to accept a different orders dataframe), which returns the portfolio's value over time, the expectation should be that it only increases. For our design criteria, the plot below displays this trend for JPM from 2008 to the end of 2009, where our holdings are constrained to positions of -1000 or 1000. This strategy is plotted against a benchmark criteria, which represents buying and holding 1000 shares of JPM throughout the sample period.

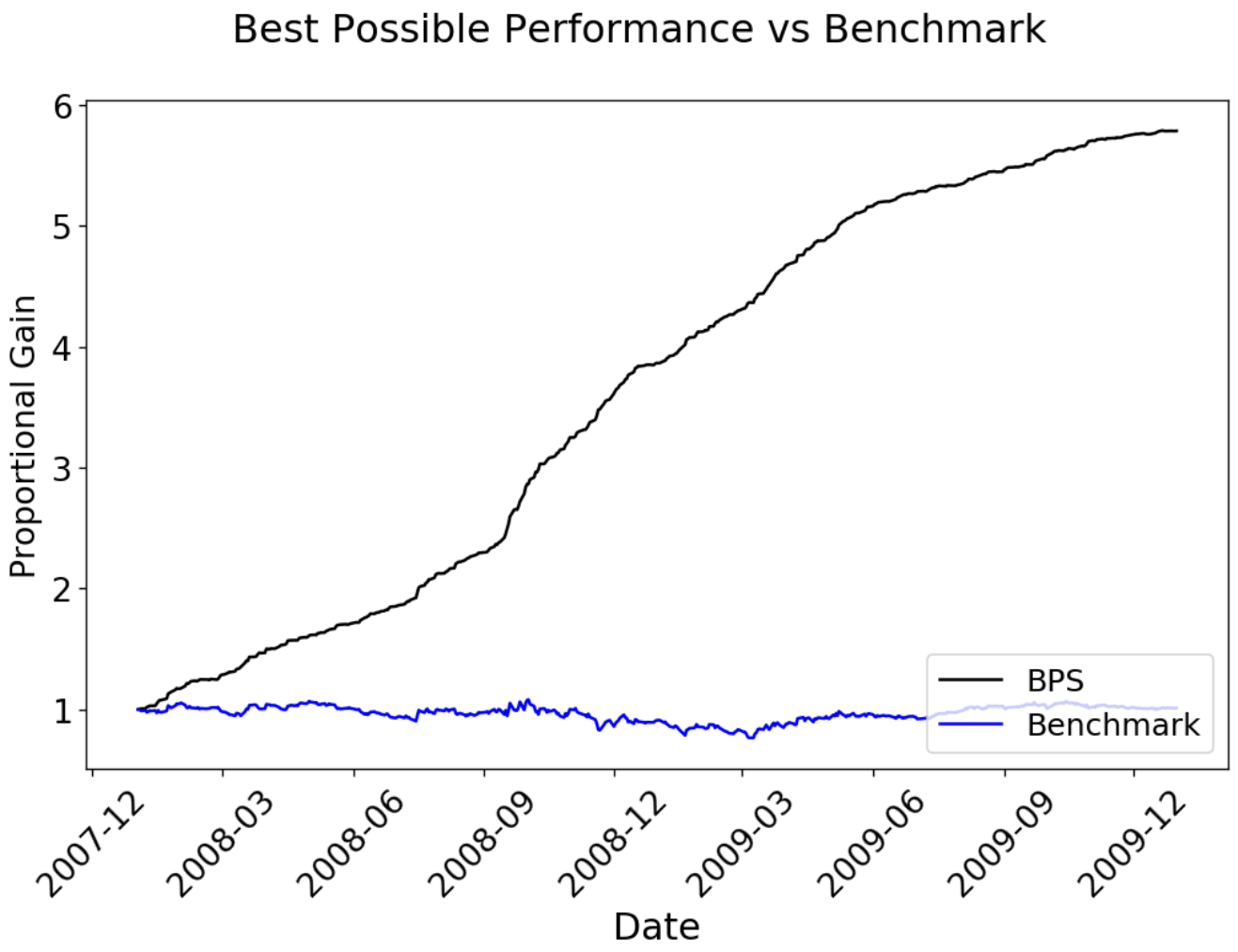

***Figure 6**: Portfolio Value for JPM vs its benchmark using the Best Possible Strategy*

As shown in the figure above, the best possible trading strategy successfully created a trading scheme that maximizes the returns as there are no days where the portfolio returns are negative (moves downwards). The portfolio metrics for these two positions are shown below.

Best Possible Strategy performance metrics:
Cumulative Return of Fund: 4.783181022184462
Standard Deviation of Fund: 0.004899109263677386
Average Daily Return of Fund: 0.003499965429401603

Benchmark performance metrics:
Sharpe Ratio of Fund: 0.15720496488905314
Cumulative Return of Fund: 0.01232493334014717
Standard Deviation of Fund: 0.01704124706817433
Average Daily Return of Fund: 0.0001687591621464017

The production of the Best Possible Strategy did incorporate some assumptions that needs to be addressed. A significant assumption made is that the starting value of the portfolio was not considered, which impacts the formulation of the trading scheme. As mentioned earlier, trades are constrained to holdings of 1000 or -1000, which led to an initial long/short of 1000 shares, and it was assumed that the starting value is sufficient to cover such (as well as shifts of 2000 shares). If this API is used in future efforts, this fallacy should be accounted for. The other assumption made is that this API only applies to trading strategies for a single stock. It will not optimize the trading strategy for multiple shares.

Part 3: Manual Strategy

This last, and primary potion of this project involved the creation of the manual strategy learner, which will leverage the indicators discussed in Part 1 to produce a trading dataframe. This trading dataframe will then also be run through the market simulator and the resulting portfolio values will be assessed in comparison with the best possible strategy. To being, all five indicators discussed have been implemented into the indicators.py library, and called into the ManualStrategy.testPolicy.py function. From here, I had decided to create a dataframe containing all of the resulting values from these indicators as well as the price data (df2 in the script). This was performed by concatenating the resulting dataframes from the indicators to the prices dataframe. From this dashboard of information, rules were set for what would indicate a long/short position for each indicator, based initially on the traditional criteria discussed. These rules were implemented and placed into two separate dataframes, df_buy and df_sell. With these rules, the two dataframes were populated with either a 1 or 0, which would represent that the signal was met or not met respectively, for instance, if the price on a day exceeded the Bollinger Band upper band, it would trigger a 1 in the df_buy for that particular day. Each indicator was separated in these two dataframes as different columns. After fully populating the two dataframes, it became evident that the Bollinger Bands was the most prominent feature in that they were triggered the least in comparison with the other four indicators. The EMA crossings appeared to be the least significant as they triggered nearly every time the MACD or SOSC was triggered. As such, I decided to remove the EMA crossings from the strategy. With the two momentum indicators, I decided to group them as an OR function since they fundamentally similar, meaning that either can be triggered to represent an action. To ensure that actions were not made as a result of spikes in the K value due to outliers, I had decided to only trade when the three day D value was triggered. This led to the trading logic shown below:

Trigger = BB and D and (MACD or Stochastic Oscillator)

To make things interesting, I had decided to iteratively optimize the characteristics of these indicators for the in sample test date range. This was performed very crudely with for loops and iterating though each value to determine the best return parameters (I was not aware that sklearn was allowed until after the fact). Logically, I started with the BB, which found that a change in the window from 20 days and a standard deviation of 2 to 19 days and a standard deviation of 1.45 increased returns by over 40%! Similar procedures were done on the MACD And Stochastic Oscillator, which increased returns by 5% as a conglomerate. The final parameters determined for the in sample test range is as shown:

- Bollinger Bands: 19 day window, 1.45102 standard deviation
- MACD: EMA12 and EMA24 with a 9 day signal
- Stochastic Oscillator: 12 day window with a 3 day average
- D oversold/overbought limits: 30 and 70 respectively.

With a fundamental understanding of technical indicators, I determined that this would be a effective trading strategy as it monitors both price divergence (BB) and the strength of price actions (momentum indicators). The goal is to trade when price action appears to be at peaks or valleys, and momentum appears to be shifting to the opposite direction. Below is a plot of when trades are made, where green represents longs and red represents shorts, and a comparison between the manual trading strategy and the benchmark performance.

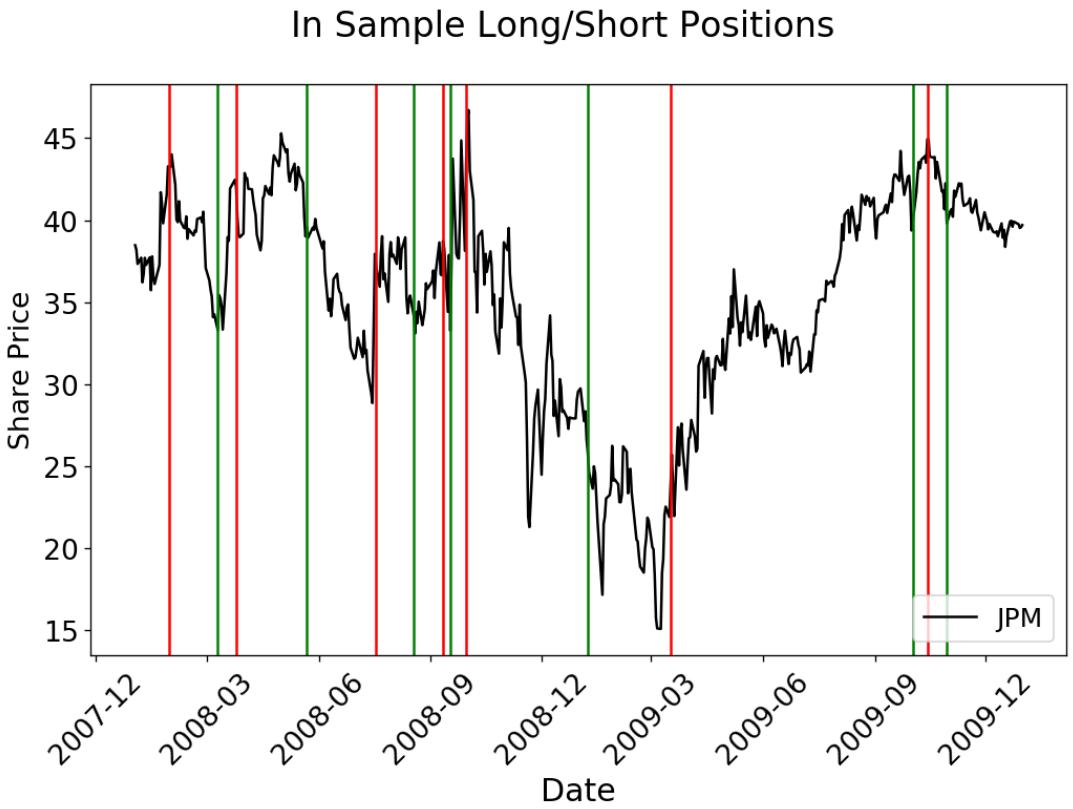

***Figure 7**: In sample manual strategy trades*

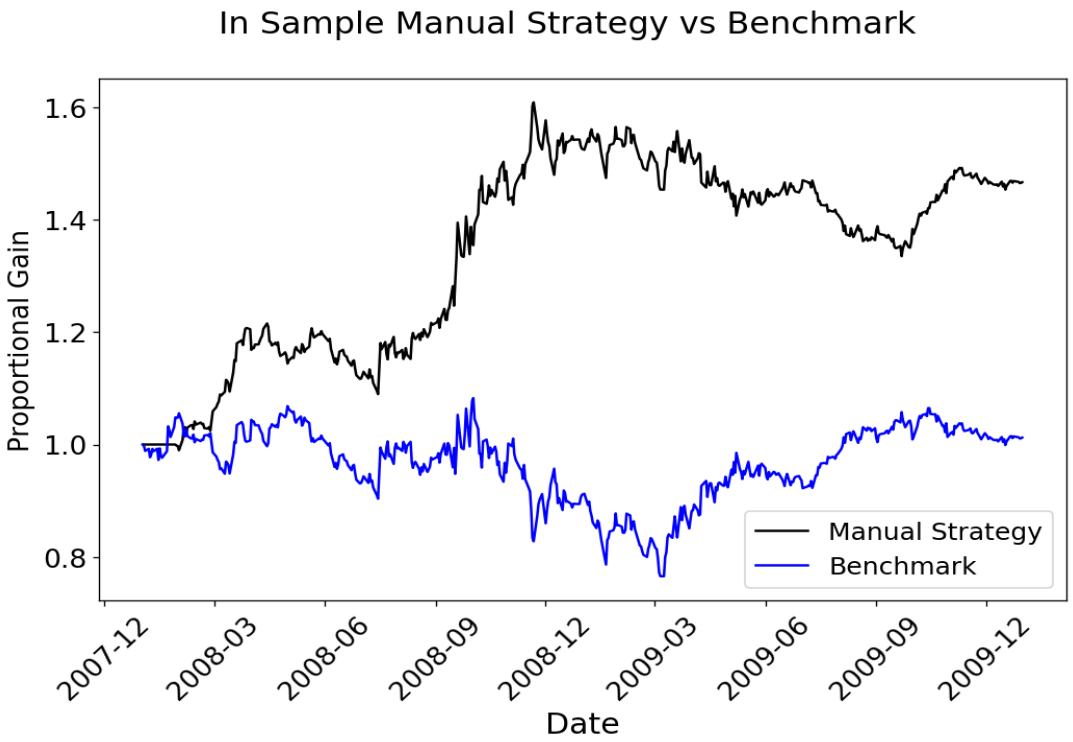

***Figure 8**: Manual Strategy vs Benchmark performance*

Part 4: Comparative Analysis

With the optimized parameters, the following performance criteria was returned for the in sample data, using the assess_learners function produced in project 1.

JPM Manual Strategy Performance for In Sample Data

Sharpe Ratio of Fund: 1.1383419277443256
Cumulative Return of Fund: 0.46654549999999984
Standard Deviation of Fund: 0.011520549890214475
Average Daily Return of Fund: 0.0008261248210704854

Final Portfolio Value: 146654.55

For a reasonable comparison, below is the performance of our benchmark, which represents 1000 shares of JPM held from day one until the end of the in sample period

Benchmark Performance for In Sample Data

Sharpe Ratio of Fund: 0.15720496488905314
Cumulative Return of Fund: 0.01232493334014717
Standard Deviation of Fund: 0.01704124706817433
Average Daily Return of Fund: 0.0001687591621464017

Final Portfolio Value: 101027.7

For the comparison between using the manual strategy on JPM and its benchmark, it did phenomenally better! The manual strategy returned 46.7% over the in sample period, while the benchmark only increased by 1.0%! This should be somewhat expected, as the manual strategy was partially optimized. Now for the comparison with the best possible strategy.

Best Possible Strategy for In Sample Data

Sharpe Ratio of Fund: 11.340883771066459
Cumulative Return of Fund: 4.783181022184462
Standard Deviation of Fund: 0.004899109263677386
Average Daily Return of Fund: 0.003499965429401603

Final Portfolio Value: 577263.2499999998

Well for the best possible strategy really puts things into perspective! Sadly, our manual trader did not beat the best possible strategy. To present these results graphically, below is a plot with the Y axis representing the proportion of returns from the starting value of the portfolio.

Now, since it appears that the manual strategy does work with in sample data, and has been optimized to maximize the returns for this period, we apply the same criteria across the out of sample data. The same performance metrics as before is displayed below.

JPM Manual Strategy Performance for Out of Sample Data

Sharpe Ratio of Fund: -0.3493907618732351
Cumulative Return of Fund: -0.10554600000000003
Standard Deviation of Fund: 0.008453974344227252
Average Daily Return of Fund: -0.00018606816424467245

Final Portfolio Value: 89445.4

Benchmark Performance for Out of Sample Data

Sharpe Ratio of Fund: -0.25665656051989255
Cumulative Return of Fund: -0.08357911003280027
Standard Deviation of Fund: 0.008500158322332455
Average Daily Return of Fund: -0.00013742923038916516

Final Portfolio Value: 91445.7

Best Possible Strategy for Out of Sample Data

Sharpe Ratio of Fund: 10.960193084108203
Cumulative Return of Fund: 2.077809245212489
Standard Deviation of Fund: 0.003248362903127673
Average Daily Return of Fund: 0.0022427583220339187

Final Portfolio Value: 307121.34999999986

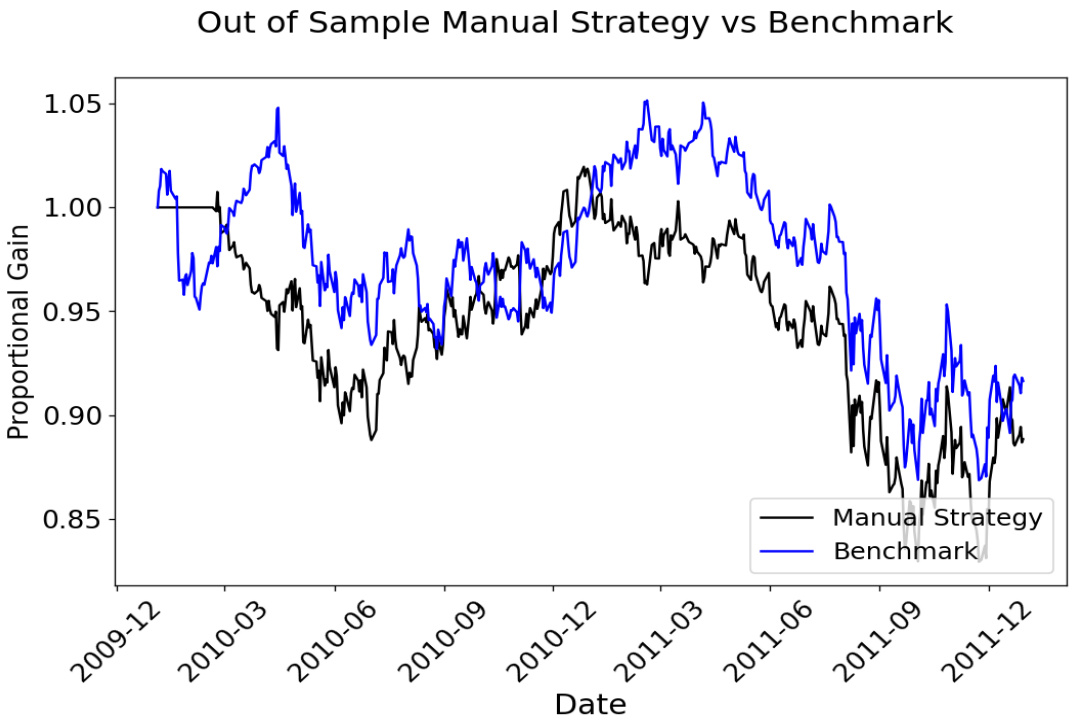

***Figure 9**: Out of sample performance comparison*

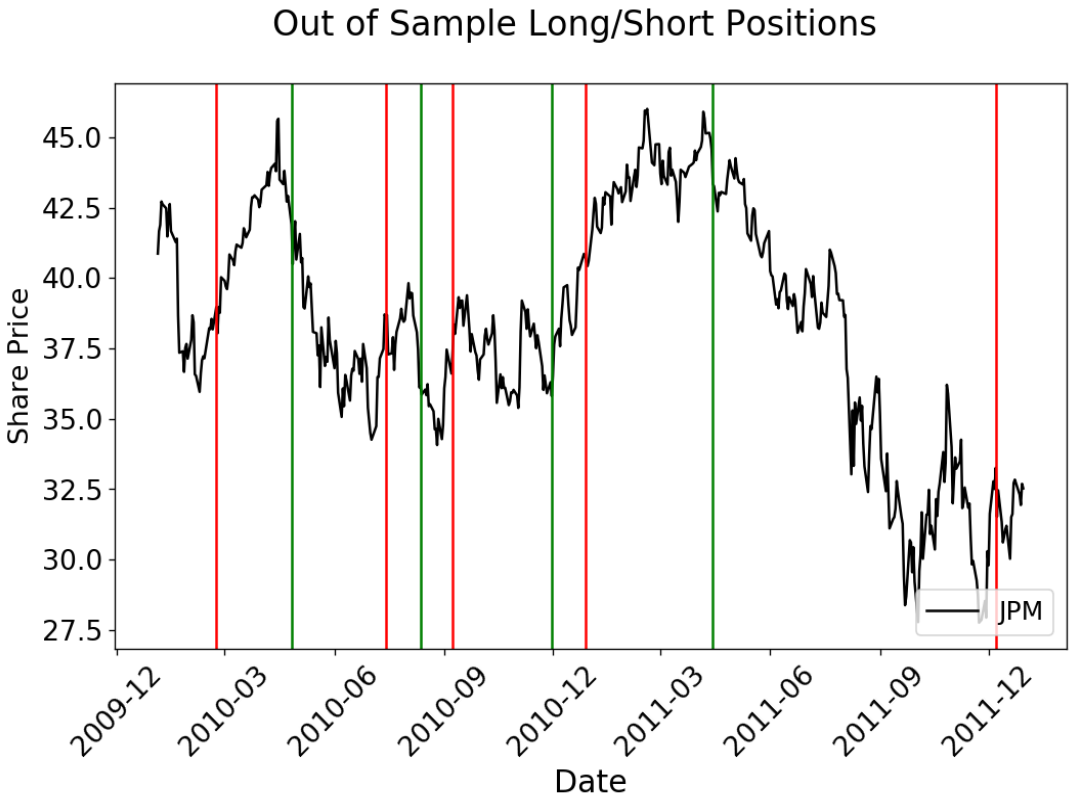

***Figure 10**: Out of sample manual strategy trades*

Ouch! Unfortunately the manual strategy for the out of sample data returned a deficit of 10.5%, whereas the benchmark only had a deficit of 8.6%.

While our manual strategy did, to an extent, fail on our out of sample data, it is to note that in additional to the benchmark returning a deficit, even the best possible strategy saw significantly reduced returns, from 477% to 207%. This means that JPM overall during this period performed very poorly relative to the in sample period. With this in mind, it is understandable that the manual strategy fails for the out of sample criteria, the best return that could have been made is a little over 200%. Additionally, while the manual strategy did not net positive returns over this period, the deficit was comparable to the benchmark during the out sample period, only losing about 2% more, whereas it outperformed the benchmark by over 45% for the in sample period. It is difficult to conclude the success of our manual strategy trader with such limited information, but we were successful in meeting our goal of creating a manual trader using some fundamental indicators.

From this project, we had investigated five technical indicators, and implemented a trading strategy that ultimately utilized all of them. From this, we identified that the strategy may work under certain situations, but does indeed have limitations. As an added bonus, we were exposed to the potential gains of trading through the development of the best possible strategy trader.