QuantFy: Predictive Market Behavior App using Machine Learning

Balagangadhar Bathula

March 15, 2017

Abstract

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. To maximize return of investment these firms use large amounts of historical data and apply machine learning algorithms to the process. QuantFy is an application (app) that would build predictive/optimized models using machine learning models such as linear regression, random forests and k-nearest neighbour.

Contents

This is description document for the Capstone project for Machine Learning Nanodegree program of Udacity.

1 Domain Background

Many financial, high frequency trading (HFT), and hedge fund companies are analyzing trading strategeis including algorithmic steps from information gathering to market orders using machine-learning approaches. This project proposes to apply these probabilistic techniques to make trading decisions. Using approaches such as linear regression, Q-learning, k-NN and regression trees and apply them to the actual trading situations.

This application software will be build using Python tools [Hil]. The book [RB] provides a comprehensive overview of domain knowledge that would be required to understand the financial market world. Also the speicifications given [Bal16] will be used as guidelines to build this application.

2 Problem Statement

Using the historical data for prices and performance statistics as features, we can get predicted future price of the stock. For instance we can set of performance statistics (such as Bollinger bands, price-to-earning (P/E) ratio) as features of the present date mapped to price (say one week forecast which is an input) of price. This creates a data set $< X, Y >$, where X is the set of features and Y as trading days. Features used are the measurable quantities that a particular stock could use in the predicting things such as change in price, market relative change in price or simply future stock price. Few questions that one would ask, before getting started are-

1. Breath and depth of data: How much historical data one would like to consider?
2. What ticker symbols are you going to use?

General machine learning models that can be used for this problem are:

1. Regression
2. k-nearest neighbors
3. Random Forests
4. Time-series analysis: This analysis will be used to estimate the commodity prices such as oil/gas etc. For instance we can observe long term trend in the time-series (such as exponential) and fit an exponential model to predict the future price. Using the concepts of drift and seasonality, build models for commodity stocks.

3 Project Design

Design phase of the QuantFy application consists of following phases:

- Time-series plots for the various stocks can be visualized at this link: Price plot [Bat17b]. This link also computes various other technical indicators such as the SMA, Bollinger bands, etc.
- Portfolio optimizer: Analyzing using stock price data, and using numerical optimization techniques, this has already been implemented in the app. Link to the app is **Portfolio optimizer** [Bat17a]
- Investment and Trading using machine learning: This was be built using the machine-learning stratagies such as (1) k-nearest neighbors (2) Random forests (3) Linear regression and (4) Ridge regression. One can compare the performance of any two algorithms in the application. For more details, visit the application at **ML models**

3.1 Software

1. Front-end: Python `flask` based web application
2. Back-end: Python libraries, such as `scikit-learn, pandas, numpy, scipy, bokeh`.
3. Hosting platform: Heroku

This application can be accessed at: **QuantFy** [Bat17c]. The current version of the source code for this application can be found at this GitHub link: **Source code** [Bat17d]

4 Datasets and Inputs

There are many open APIs to extract data:

1. Yahoo Finance:
   (a) Using the http://ichart.finance.yahoo.com/table.csv?s={YOUR_SYMBOL} we can query and get historical data.
   (b) We can also use yahoo_finance python library or,
   (c) We can also use pandas_datareader
2. Quandl: Using quandl python API we can get the historical data
3. Bloomberg API

Current version of QuantFy application proposes to uses Yahoo Finance source and Quandl to get historical data. One can extract more than 10 years of historical stock prices for many ticker symbols. This data is obtained using appropriate API calls to the sources (listed below).

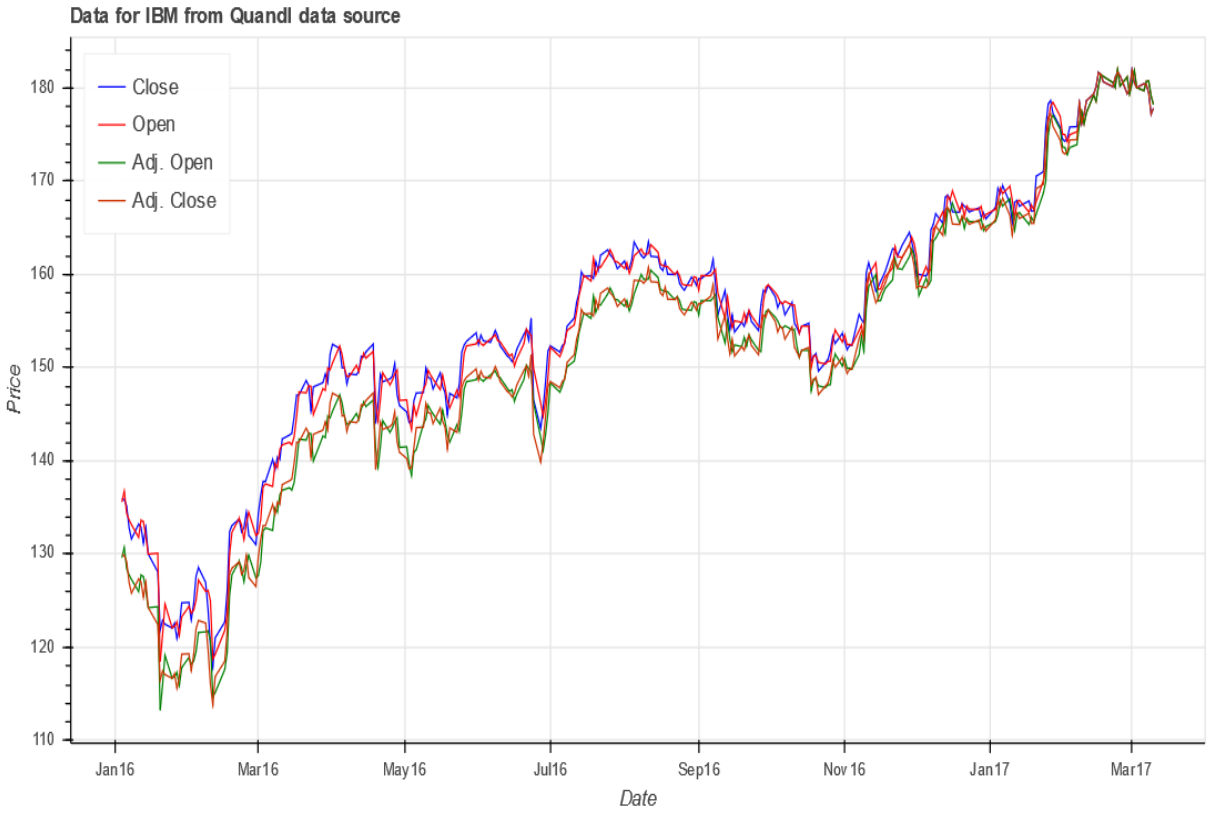

Figure 1: IBM prices from Jan 1 2016, from Quandl data source

For instance user can the date range for viewing the stock prices of ticker symbols. Then an request call will be made and acquired data will be converted into a `pandas` dataframe. This data will also be stored locally, using python libraries such as `ediblepickle`, so when the same request is made (no change in the parameters), data is retrieved locally. Since the data is generated online (rather than locally available data), we are not particularly concerned about the storing the data. At least from the past experience, the stock price data can be easily stored in the dataframe without any memory errors. We anticipate that the historical prices for the each ticker symbol will be in the order of less than 1 MB.

1. Close price and/or Adjusted close price
2. Start price and/or Adjusted start price,
3. Opening price and/or adjusted opening price

The data from these sources consists of features like:

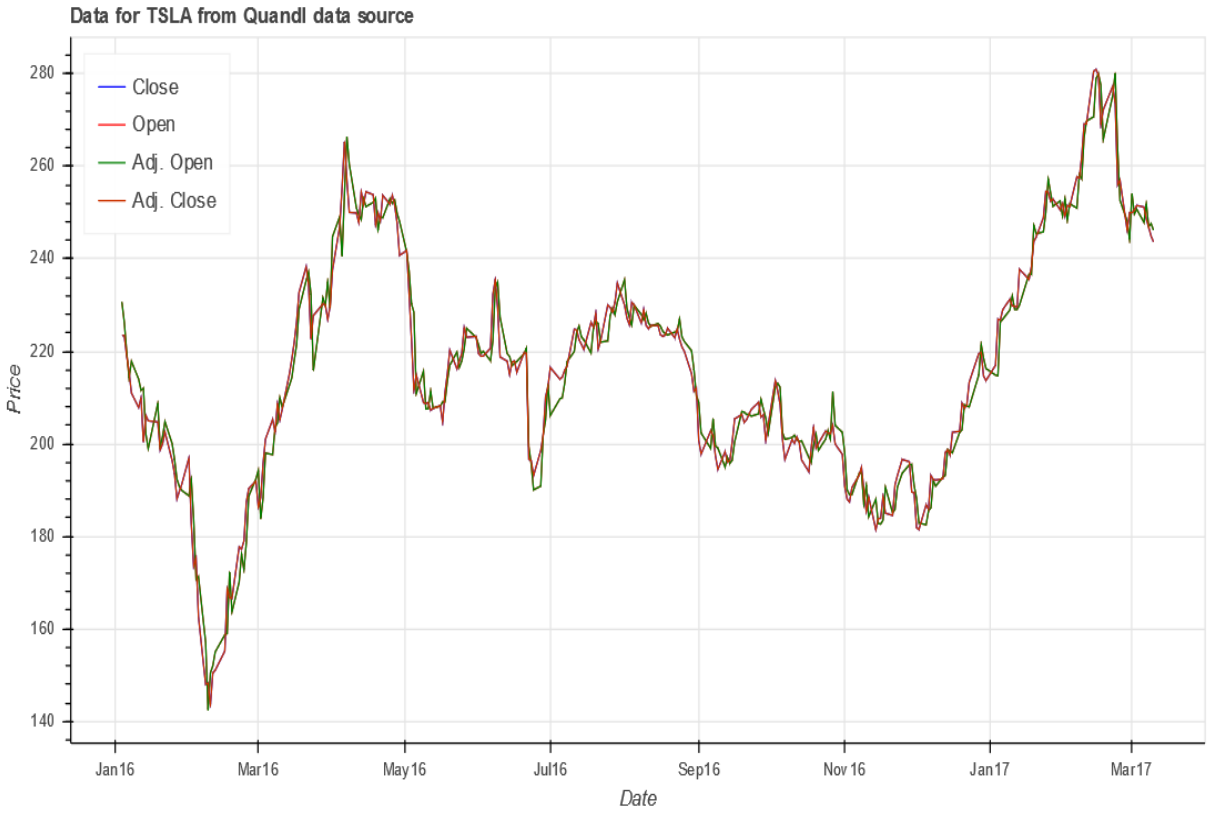

Figure 2: Tesla prices from Jan 1 2016, Quandl data source

Snapshot of the plot prices for IBM, TSLA (Tesla) and AAPL (Apple) are shown in Figure 1, 2, and 3 respectively. For more detailed refer to the application website **QuantFy** [Bat17c]. Table 1, describe the parameters for the data, such as count, mean, standard deviation etc. This can is achieved using the `describe()` option in pandas.

5 Analyzing the stock price data

When getting the stock price data for the ticker symbols, one need to clean the data to see if there are any un-available data points. For instance the stock has not traded a day, then it is mostly likely the exchange is closed. These `NaN` values have to be removed from the data-set. The most common way to do it this is to align the ticker symbol with the a benchmark symbol, say S&P500 company and drop any `NaN` values in the data set. Also, we drop dates at which the S&P500 has not traded. Sometimes the company's stock price might not be available, as it did not have an IPO (initial

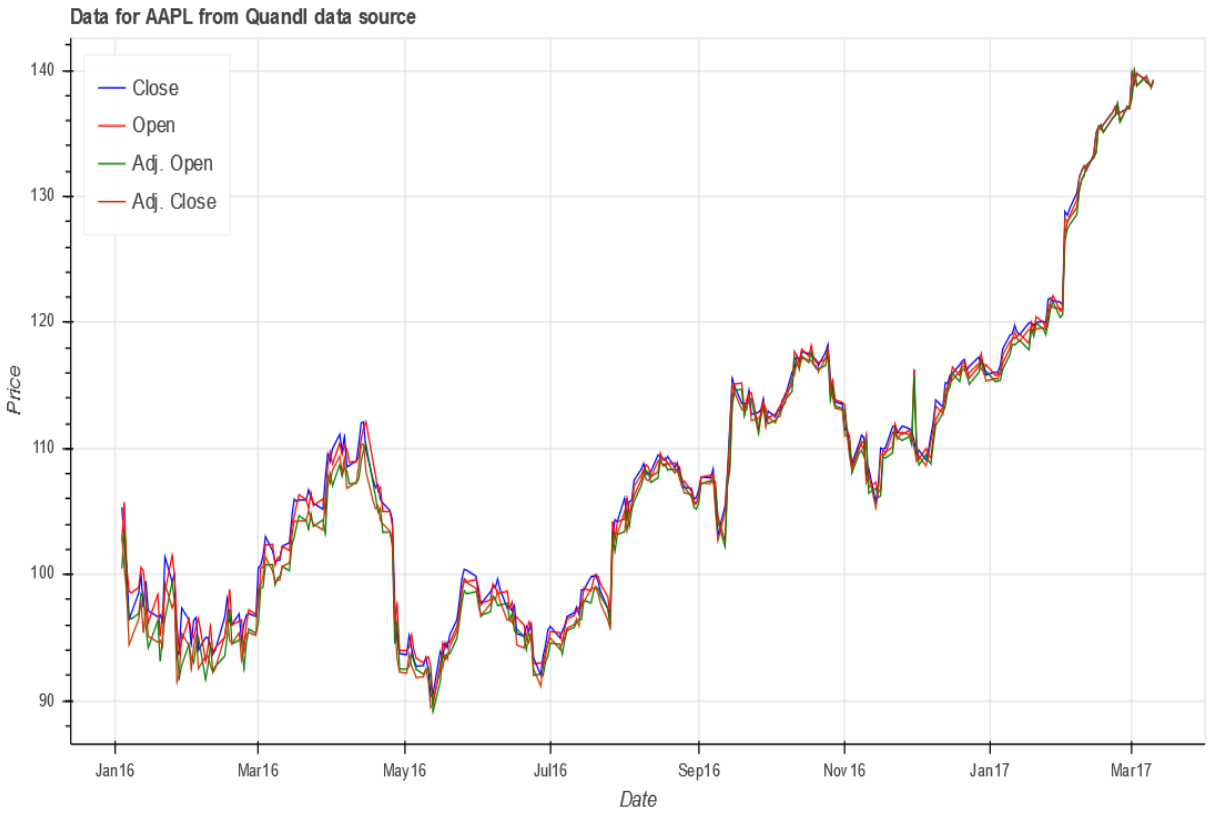

Figure 3: Apple prices from Jan 1 2016, Quandl data source

Metric	IBM	TSLA	AAPL
Count	300	300	300
Mean	151.4008	216.3058	107.5923
Standard deviation	15.7737	25.3987	12.2136
Minimum	113.8011	143.67	89.47
25% percentile	143.5718	199	96.9126
50% percentile	151.7834	215.205	107.1624
75% percentile	151.7834	215.205	107.1624
Maximum	151.7834	215.205	107.1624

Table 1: Description of the data for the stock symbols IBM, TSLA, AAPL

public offering) for the date. So we need to remove such data points as well. For example if we get the stock price of the `GOOG` from year 2010, then there will be `NaN` until year 2014. In `pandas` we can accomplish this with `df=df.dropna(axis=0)`, where `df` is the dataframe, and `dropna()` drops any values with `NaN` along the rows (`axis=0`).

We use adjusted close price is what we use for the rest of the discussion. Along with these features, one could compute other features like:

1. Daily returns (DR): How much the price will go up or down on a particular instance of time:
$$r(t) = \frac{p(t)}{p(t-1)} - 1,$$
where $p(t)$ is the price at time t.
2. Average daily return (ADR): This is simply the average value of the the daily returns.
$$adr = \frac{\Sigma_{t=1}^{N} r(t)}{N}$$
3. Cumulative daily return (CDR): Defined as the ratio of the final value of the the daily return to the initial value of the daily return
$$cdr = \frac{r(N)}{r(0)},$$
where N is the last sample point in the series.
4. Standard deviation of DR: It is simply the standard deviation of the daily return ($r(t)$), which is the square-root of the variance.
5. Cumulative return is given by,
$$cr(t) = \frac{p(t)}{p(0)} - 1$$
6. Simple moving average (SMA): Also know as moving average
```
rolling_df=pd.DataFrame(index=df.index)
rolling_df =pd.Series.rolling(df,window=window).mean().to_frame()
rolling_df.ix[:window, :] = 0
```
Table 2 shows the computed parameters for the data. Also, the other parameters are plotted in Fig. 4, 5 and 6 respectively.
7. Bollinger bands:
$$U_{band} = m_r + 2 \times \sigma_r,$$
$$L_{band} = m_r - 2 \times \sigma_r$$
where m_r is the rolling mean and σ_r is the rolling standard-deviation of the adjusted close price.

Metric	IBM	TSLA	AAPL
Cummulative daily return	0.368	0.091	0.348
Average daily return	0.0011	0.00056	0.0011
Standard deviation of daily return	0.012	0.023	0.014
Sharpe ratio	1.49	0.38	1.23

Table 2: Parameters computed for the stock symbols IBM, TSLA, AAPL

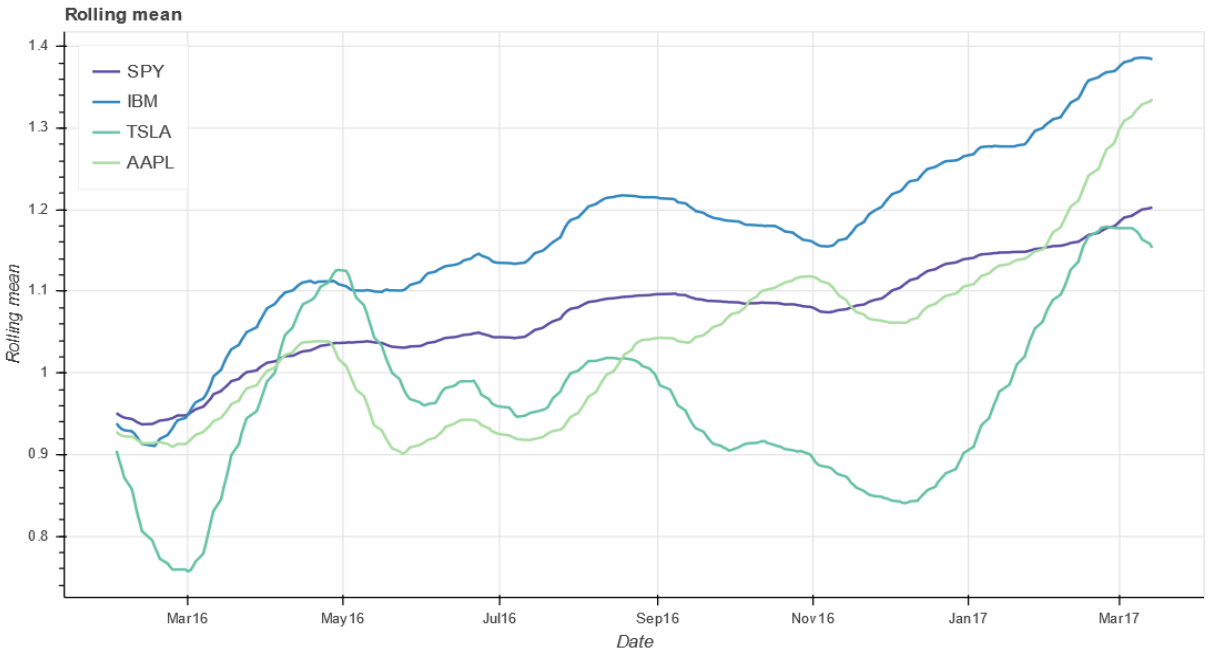

Figure 4: Rolling mean for the symbols IBM, TSLA and AAPL

8. Normalized Bollinger bands: Value varies between -1.0 and 1.0.

$$bb_{norm} = \frac{price[t] - SMA[t]}{2 \times \sigma[t]}$$

where σ is the rolling standard deviation and SMA is the simple moving average.

9. Sharpe ratio This metric adjusts the return for risk, also know as risk adjusted reward.

$$sr = \frac{R_p - R_f}{\sigma_p},$$

where R_p is the portfolio return, R_f is risk-free return and σ_p is the standard deviation of portfolio return.

$$sr = \frac{E[R_p - R_f]}{\sigma[R_p - R_f]}$$

Sharpe ratio varies depending on how frequently you sample the data. $sr_f = k \times sr$, where $k = \sqrt{\#\text{ samples/year}}$

10. Volatility is nothing but the standard deviation of the daily returns.

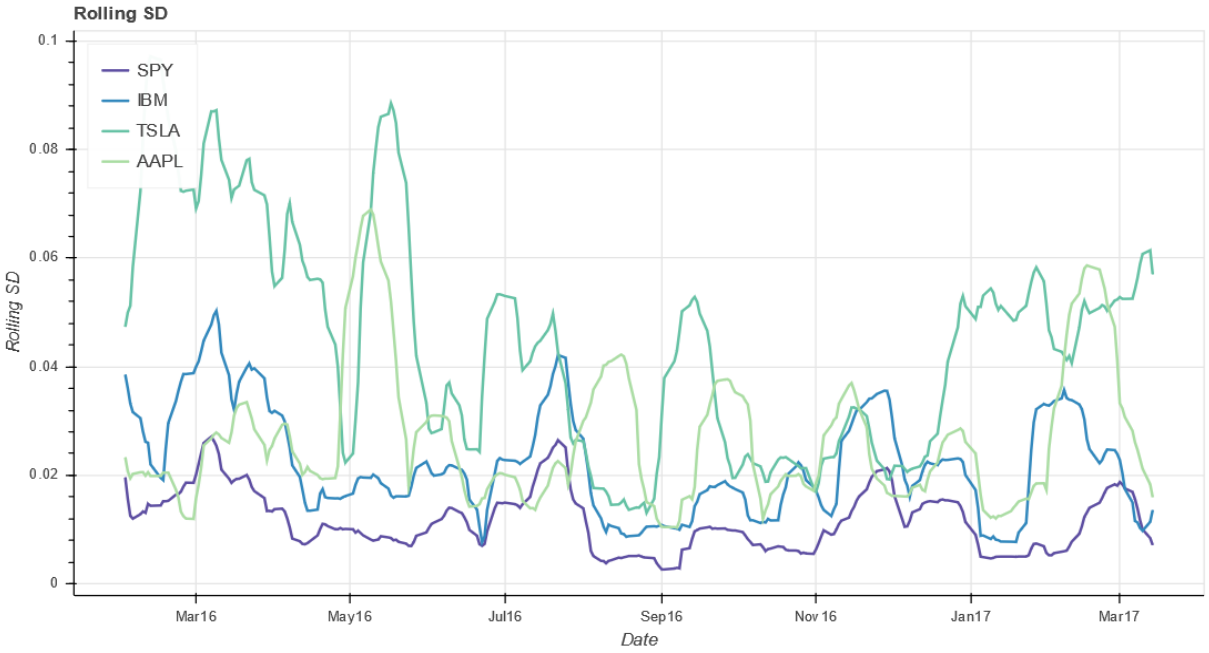

Figure 5: Rolling standard deviation (σ) for the symbols IBM, TSLA and AAPL

11. Momemtum is defined as:

$$r(t) = \frac{p(t)}{p(t-N)} - 1$$

where N is the size of the window.

Plots for rolling mean, rolling standard deviation and bollinger-bands are show in Fig. 4, 5, and 6 for IBM, TSLA and AAPL. For more details, refer to the application website [Bat17b]

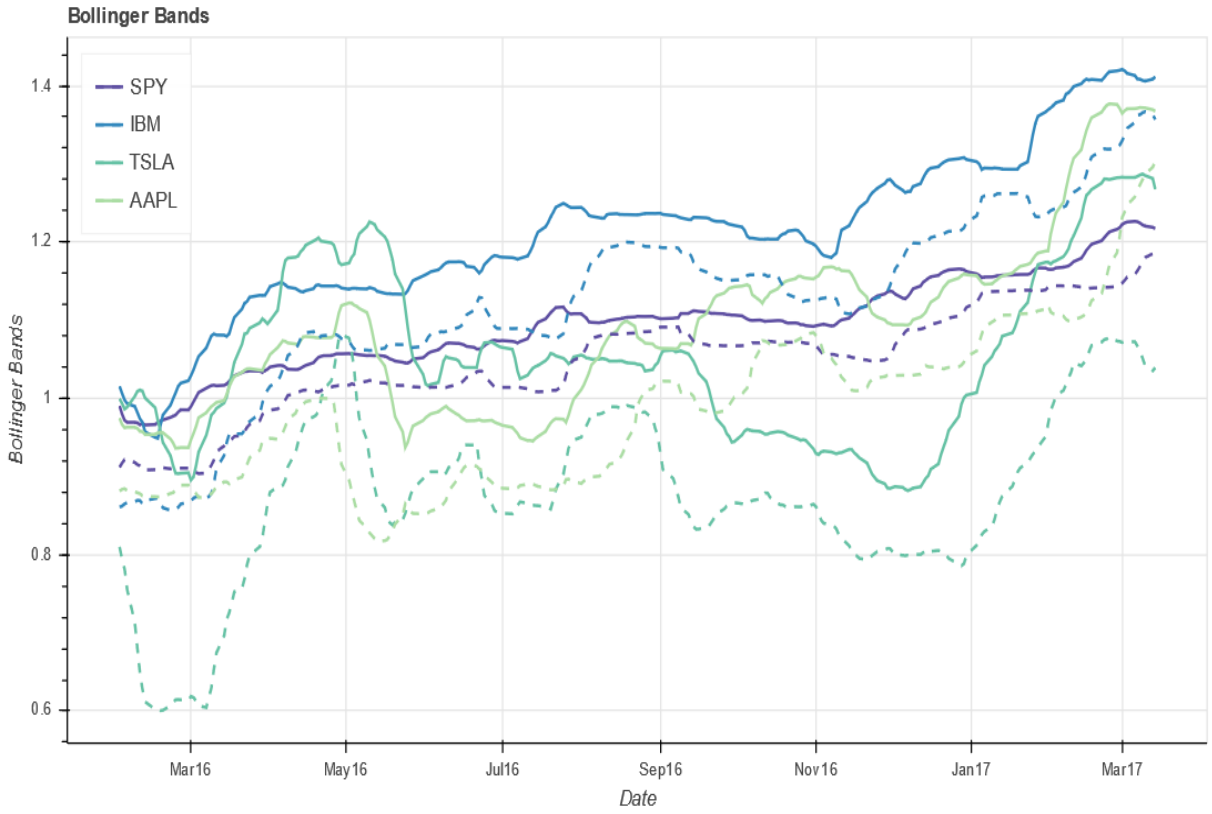

Figure 6: Bollinger bands IBM, TSLA and AAPL; upper are shown in solid lines and lower bands are shown in dotted lines

6 Algorithms used in the project

This section provides brief description of the algorithms used in this project.

6.1 Convex Optimization

A real-valued function defined on an interval is said to be convex if the line-segment between two points on the graph of the function lies above or on the graph. More formally, let $\mathbf{X}$ be a convex set in a real vector space and let $f : \mathbf{X} \to \mathbb{R}$ be a function, then f is called **convex** if:

$$\forall x_1, x_2 \in \mathbf{X}, \forall t \in [0,1] : f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

In finance and economics, *convex optimization* plays an important role. Convex optimization algorithms are easy to solve, since they have one local minimum, which is also a global minimum. They can be solved using gradient-descent or other numerical optimizations as Sequential Least SQuare Programming (SLSQP). Fig. 7 shows an example of the convex function.

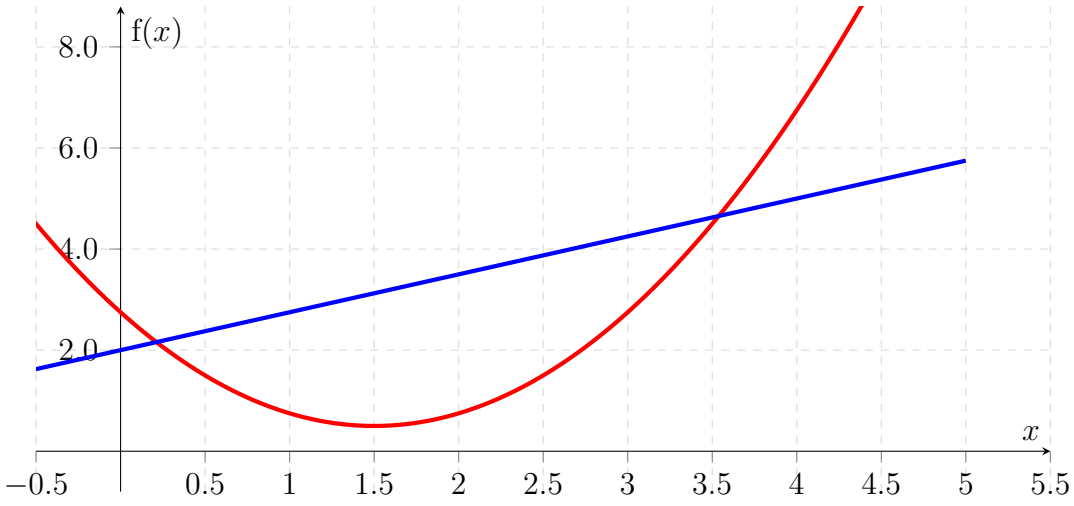

Figure 7: Example of the convex function $f(x) = (x - 1.5)^2 + 0.5$.

- Given set of assets and time-period, find allocations of funds to assets that maximizes performance.
- Performance metrics can be:
  - $\max cr(t)$,
  - $\min \sigma(t)$, i.e., minimizing volatility.
  - $\max sr$
- Framing the problem:
  1. Provide a function $f(X) = cX + b$ to minimize.
  2. Provide an initial guess for allocations.
  3. Call the optimizer.
- Constraints: We know that allocations for each stock symbol in the portfolio should be ≤ 1, i.e.,

  $$\Sigma_{i=0}^{k} |X_i| = 1.0,$$

  where X_i's are allocations for each stock.

Below is the code snippet of the optimization in `scipy`.

```
import scipy.optimize as spo
initial_alloc=[(1.0/len(sym))]*len(sym)
bounds=((0,1),)*len(sym)
opt_allocs=spo.minimize(sharpe_function*-1, initial_alloc,
        args=df, method='SLSQP',bounds=bounds,
        constraints=({'type':'eq','fun':lambda
        opt_allocs: 1-np.sum(np.abs(opt_allocs))}
        )
        );
optimal_allocs=opt_allocs.x
```

In this project, we are trying to predict the stock-prices for the

6.2 Linear regression

6.3 Random forest regression

6.4 k-nearest neighbors

6.5 Ridge regression

6.6 Benchmark Model

As a benchmark model, one could use a simple mean-model. For instance, a particular stock price can be predicted using mean of the adjusted close price. Also, algorithms such as k-nearest neighbor etc., can be applied using default parameters in `scikit-learn` and then using `GridsearchCV` one could obtain optimal parameter setting. In other words, benchmark models can be based on simple parameter settings. Performance of advanced algorithms such as ensemble approaches can be tested against these benchmark models.

7 Evaluation Metrics

Here we choose two evaluation metrics:

Evaluation metrics can include root-mean-square error (RMSE), correlation coefficient.

- **RMSE**: Mean Absolute Error (MAE), is calculated as the ratio of sum of absolute values of difference between actual value and the predicted value to number of

samples. Clearly this metric indicates the average deviation of a predicted value from its actual value. This is a linear function and all errors are weighted equally.

On the other hand mean-square-error (MSE), is ratio of sum of squared deviations to the number of samples, i.e., it is the mean of the squared deviation. This is a quadratic function, and if the errors are large, they are weighted more while the small errors are weighted less (say if absolute error for a sample point is less than 1, then square of the number is much smaller). Thus in a MSE based estimator, we can clearly see how the predicted values compared to the true values. MSE would be preferred performance metric over MAE especially if you want to have a model, where large errors are particularly undesirable. Also by taking the square-root of the MSE value we get the Standard deviation, which is again helps to predict the variability of the data.

For this problem, minimizing the metric RMSE makes the stock price prediction accurate.

– **Correlation**: This metric can be used to evaluate ML algorithm is to look at the relationship between predicted value vs. actual value. Let say we have a test data, and we run our model again test data to get $Y_{predict}$, which is the stock price in this case. We can now compare the Y_{true} against $Y_{predict}$. By looking at the relationship between Y_{true} and $Y_{predict}$ we say if the model is correctly predicting the values. This metric is the correction function ρ and it ranges between $-1 \leq \rho \leq +1$. If $\rho = 1$, then they are strongly correlated; if $\rho = -1$ they are negatively correlated and if $\rho = 0$, they are not correlated.

8 Methodology

9 Results

9.1 Portfolio optimization

Portfolios in comparison with SP&500 for the symbols, GOOG, AAPL, MSFT and IBM for duration between Jan 1 2013 to March 14 2017, for both unoptimized and optimized are shown in Fig. 7 and 8 respectively.

9.2 Market Simulator

Based on the orders file, the portfolio value can be analyzed. For instance if BUY or SELL order comes, depending on the stock price value of the order, the portfolio value changes. Using the leverage threshold, orders can be executed or fail to execute if the leverage value exceeds the threshold.

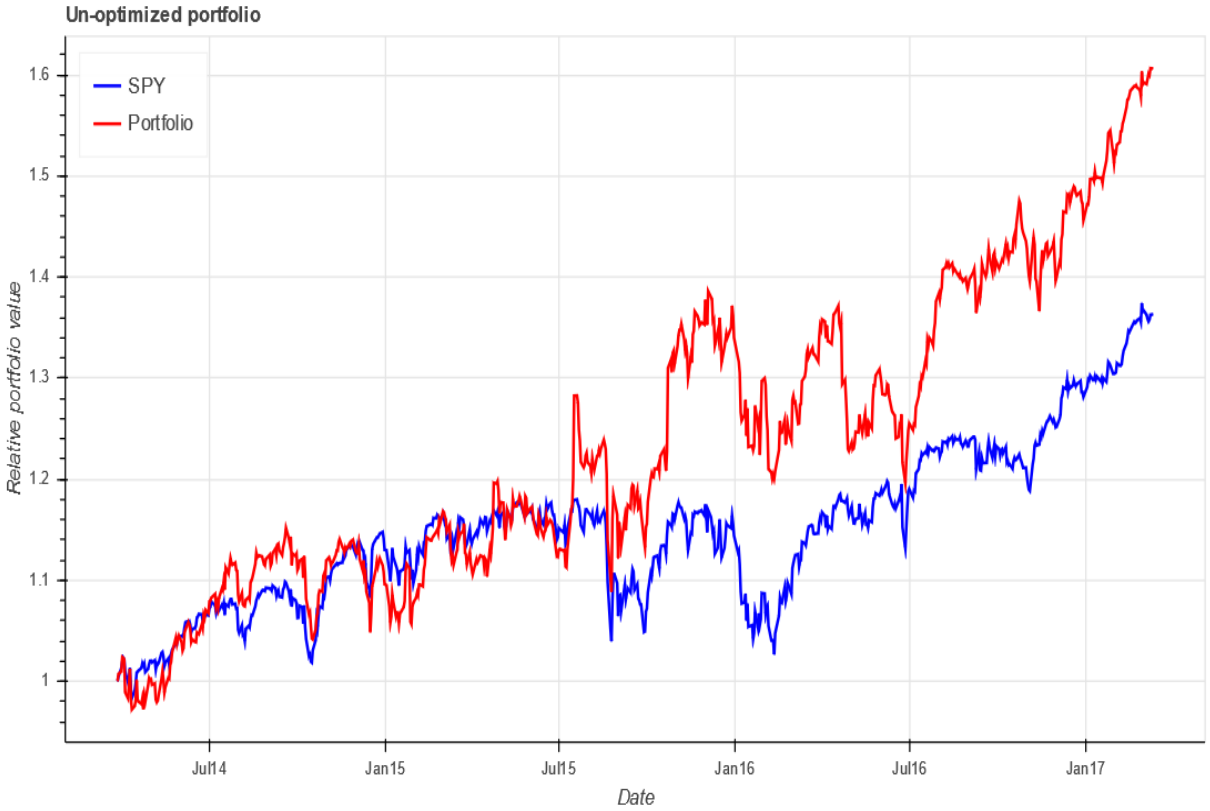

Figure 8: Unoptimized portfolio for GOOG, AAPL, MSFT and IBM in comparison with SP&500, with a start value of $10000

Metric	Un-optimized	Optimized
Cumulative daily return	0.6065	0.8507
Average daily return	0.0007	0.0009
Standard deviation of daily return	0.0116	0.0126
Sharpe ratio	0.9624	1.1424
End value	$160651.061	$185068.0823

Table 3: Parameters computed for un-optimized and optimized portfolio

Metric	GOOG	AAPL	MSFT	IBM
Random allocations	0.4682	0.1894	0.2449	0.0975
Optimized allocations	0	0.539	0.461	0

Table 4: Parameters computed for un-optimized and optimized portfolio

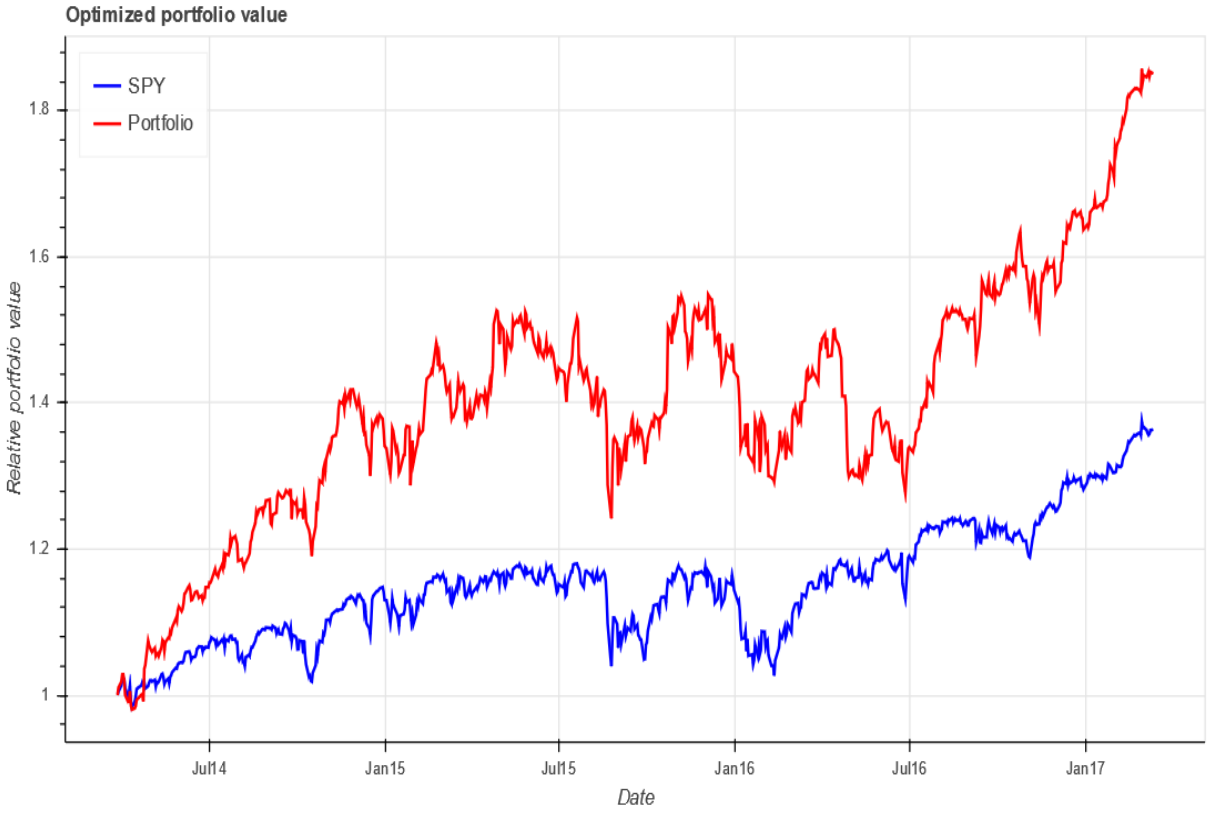

Figure 9: Optimized portfolio for GOOG, AAPL, MSFT and IBM in comparison with SP&500

Leverage value is defined as:

$$leverage = \frac{\Sigma|sp|}{(\Sigma(sp) + cash)}$$

,

where sp is defined as all-stock positions [Bal16].

Figure 10: Market simulator for a order file. The green vertical lines indicate a BUY order and RED vertical lines indicate a SELL order

The Fig. 9 is generated using orders file located at **Orders-demo**

10 Solution Statement

Predicts the stock prices and also measures the accuracy of these forecasts. Using back-testing a method where we roll back time and measure the accuracy of these forecasts. Slice the sample data from (historical) training set and apply the machine learning models to get the forecast. By comparing the predictive results of the model against

the historical results, back-testing can determine whether the model has predictive value.

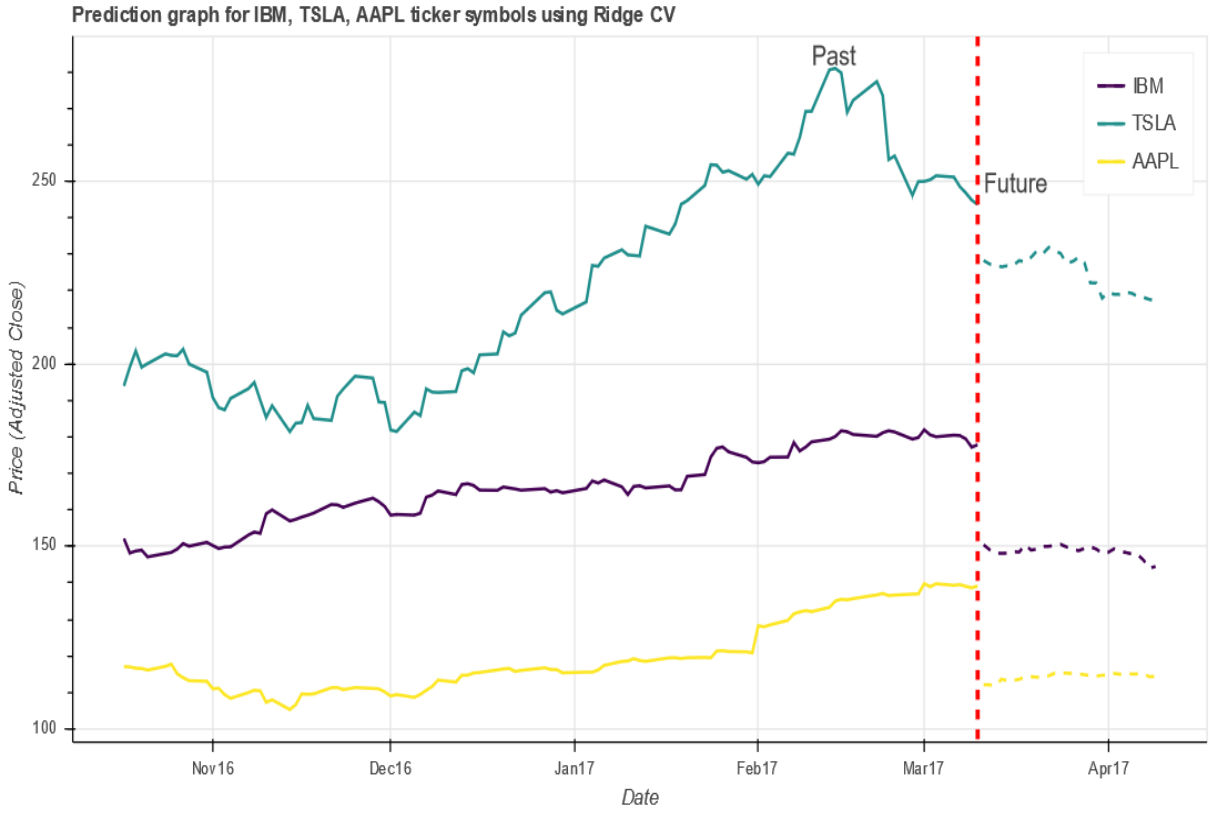

Figure 11: Prediction values for 30 days for IBM, TSLA and AAPL using Ridge regression

Left side of the RED dotted line indicates the past values, while the right side are the predicted values of the stock.

11 Conclusion

Commodity stocks (as future work) , Reinforcement Learning (future work)

References

[Bal16] Tucker Balch. Machine learning for trading. Udacity course, 2016.

[Bat17a] Balagangadhar Bathula. Portfolio optimizer: QuantFy. `http://quantfy.herokuapp.com/portfolio`, 2017. [Online].

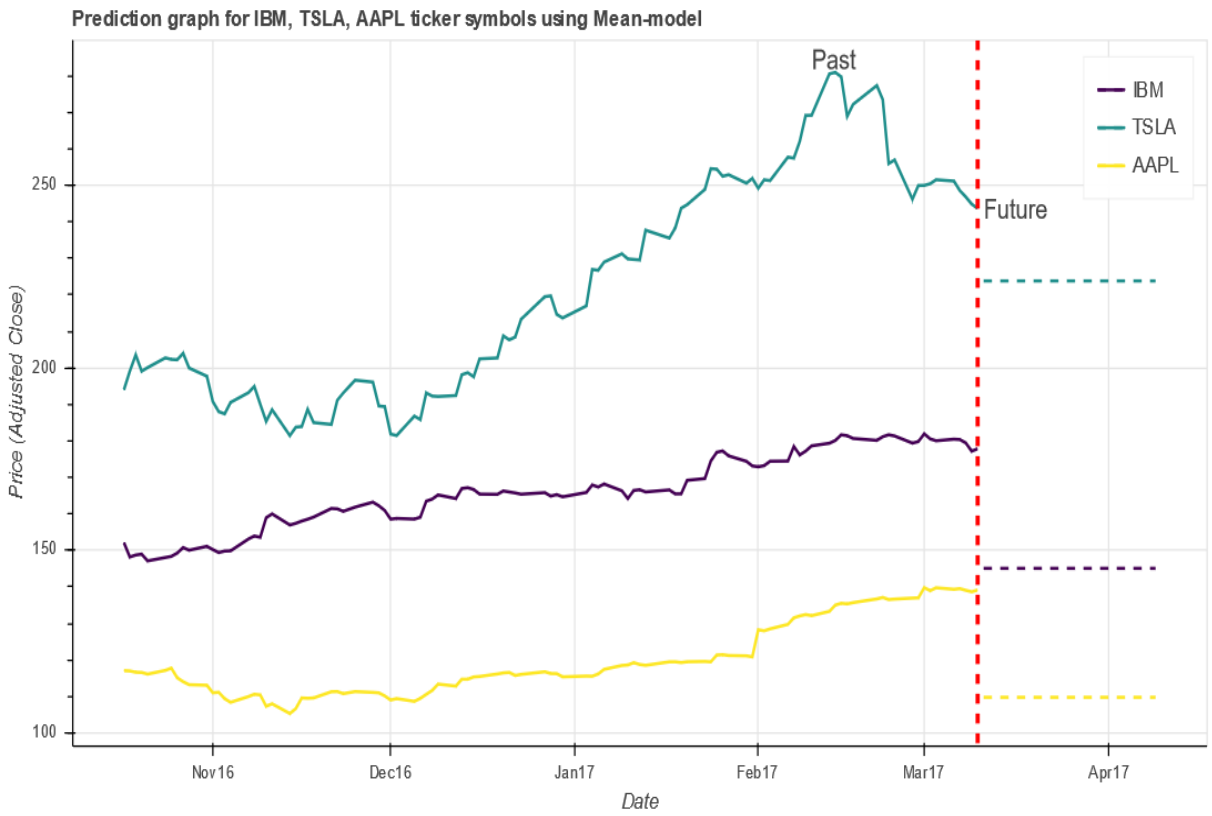

Figure 12: Prediction values for 30 days for IBM, TSLA and AAPL using the benchmark Mean model.

[Bat17b] Balagangadhar Bathula. Price Plot: QuantFy. http://quantfy.herokuapp.com/price_plot, 2017. [Online].

[Bat17c] Balagangadhar Bathula. QuantFy. http://quantfy.herokuapp.com/, 2017. [Online].

[Bat17d] Balagangadhar Bathula. Source code: QuantFy. https://github.com/beegeesquare/QuantFy, 2017. [Online].

[Hil] Yves Hilpisch. *Python for Finance*. O'Reilly.

[RB] Philip J. Romero and Tucker Balch. *What Hedge Funds Really Do: An Introduction to Portfolio Management*. Business Expert Press.