

Map Reduce for Algorithmic Trading

Akshaan Kakar, Alice Berard

Dept. of Computer Science

Columbia University

Email: ak3808@columbia.edu, alice.berard@columbia.edu

Abstract—Algorithmic Trading is an extremely competitive sector of financial markets. Developing trading algorithms involves the pivotal step of backtesting, where the performance of the algorithm is validated against large amounts of historical securities pricing data. These time series require large amounts of computational resources for storage, processing and visualization. In this paper, we describe the implementation of an algorithmic trading backtest engine that uses "Big Data" tools as a backbone to backtests for trading algorithms in an efficient and scalable fashion. We also show how this system can be extended to support live trading as well as a plethora of other features.

Keywords—Algorithms, Finance, Trading, Big Data, Hadoop, Spark, Backtesting

I. INTRODUCTION

With the computerization of order flows in financial markets, traders were afforded the ability to have computers place buy and sell orders on securities according to pre-defined strategies. The advent of such Algorithmic Trading strategies eventually developed into an extremely competitive sector of financial markets. The development of trading algorithms is now a formalized process that involves intricate research and mathematical models. Although a significant portion of this development effort is invested in the underlying mathematical theory, backtesting of algorithms is the most pivotal step in the process. Backtesting is the phase in which a trading algorithm is validated against large amounts of historical pricing data. Since trading algorithms may place orders extremely frequently and involve multiple securities in a single portfolio, large amounts of data must be stored, read and processed in order to run these tests. Moreover, the result must be visualized so that developers can gauge the performance of their algorithms quickly and conveniently.

There is an abundance of well developed and well maintained tools which have been built expressly to grapple with the large amounts of data that have now become commonplace. These tools leverage novel algorithmic and system level techniques to deal with the space and time bottlenecks that come with large datasets. Since the problem of backtesting algorithmic strategies is inherently a 'Big Data' shaped problem, we decided to use a combination of such tools in order to build an efficient and flexible system for the task.

II. RELATED WORKS

We modeled our implementation of the backtesting engine with some other tools and works in mind. One of the prominent platforms that we sought to emulate is Quantopian². The Quantopian platform allows in-browser implementation of trading algorithms in Python. It also allows the use of internal APIs as well as most external python modules for statistics, numerical algorithms etc. Although we have not extended our system to allow in-browser coding, we have also used python as the language of choice for trading algorithm implementation. Python is readable, flexible and popular, making it a good choice for algorithmic trading; a field where not everyone is well versed with computer programming. We also sought to emulate the charting and metric calculations performed by the Quantopian platform. The benchmark as well as the returns time series are plotted to provide a lucid understanding of the algorithm performance, without confusing the user with unnecessary detail. Also, important metrics such as the Sharpe Ratio, Maximum drawdown etc. are provided to the user for further insight.

III. SYSTEM OVERVIEW

The main objectives in our implementation were to keep the platform efficient and easily extensible. In order to achieve these goals, we partitioned the system into independently functioning subsystems which we then integrated to realize the final backtesting engine. For the purpose of demonstration, we utilized the free historical pricing data from QuantQuote³. The dataset as well as the subsystems are described in detail in the following subsections.

A. QuantQuote dataset

We decided to use the QuantQuote free historical stock price data for the purpose of demonstration. This data consists of daily stock tick data for the 500 symbols that are listed on the Standard and Poor's 500 Index from 1998 to present. Although we used the dataset throughout the design and testing process, we structured the backtesting engine such that it is dataset agnostic. More specifically, we designed our subsystems so that they do not impose any tight constraints on input data formatting. Further descriptions

of input data specifications are provided in the following subsection.

B. Hadoop Data Warehouse

Apache Hadoop is a distributed framework that is designed for the storage and processing of large datasets¹. Hadoop is especially well suited to read-only, batch accessing of large amounts of data. Our system only requires the reading of financial time series data, without frequent writes or editing, making Hadoop a perfect choice for the data warehouse subsystem. We stored the finance symbol pricing data in the Hadoop Distributed File System (HDFS), which can be configured to run on any number of machines without a single point of failure. We structured the file system so that the time series data for each symbol were stored in a separate file. This simple, flat structure makes accessing the data extremely simple, since for a particular symbol, only a single file must be accessed. Each file was named after the symbol, to facilitate programatic access using symbol names directly, without the need for any lookup or translation. In the case that the data repository needs to be updated with newer pricing data, only a single file need be written to. This simple organization does not compromise on efficiency since Hadoop is designed to work well with a relatively small number of very large files, as opposed to small fragments of a larger dataset. A directory listing showing the file structure is shown in Figure 1.

Within each file, the time series were stored in a comma separated value (CSV) format, with time stamps and price values as the fields in each row. The CSV format is simple and widely used, making our system largely data agnostic. Any data in CSV format with the appropriate fields can be used with our data warehouse system since our engine does not impose and other restrictions on data formatting.

C. Spark Algorithm Processing Engine

Apache Spark is a tool that was developed for processing general large-scale data efficiently⁴. Spark come with support for multiple languages and platforms and an also integrate seamlessly with the Hadoop Distributed File System. We chose Spark to be the workhorse for the main trading algorithm processing. In order to leverage the features offered by Spark, we used pySpark, which is the Spark API in the Python scripting language. Our algorithm processing engine is responsible for detecting all the trading symbols that are involved in a user-defined trading strategy, and to retrieve the corresponding data from the Hadoop data store. After retrieving the required time series, the engine then runs the rules specified in the strategy against these series and computes the basic returns for the overall portfolio. Once the returns have been computed for the entire data, further portfolio performance metrics are computed. These include the mean return, standard deviation, maximum drawdown and Sharpe Ratio. The final returns time series

and the metrics are then written to temporary output files on disk for use further downstream in the processing pipeline.

D. Javascript Server Back End

We sought to build a simple web app which could concisely present the results of algorithm backtesting to the user. In order to do this, we implemented a very basic server using the node.js JavaScript runtime.

E. Visualization

In order to visualize the performance of user defined trading algorithms viz. the benchmark, we built a JavaScript front end for our web application. In order to produce detailed and informative plots, we utilized the Highcharts and Highstock charting libraries. The schematic in Figure 2. shows an outline of the system design.

IV. ALGORITHMS

A. Map Reduce

The algorithmic paradigm we leveraged was the map reduce paradigm. We use a map reduce job to retrieve the required time slices of price time series from the Hadoop data warehouse. The data is stored in a distributed fashion across multiple machines hosting the Hadoop warehouse. The Map-Reduce Job simply accesses a file with a particular name from the Hadoop Distributed File System. In later versions, we might require only particular slices or windows of the temporal data and this can easily be achieved by augmenting the current Map Reduce job logic.

B. Mapping a function across a Spark RDD

We used the functional paradigm of mapping a function over an iterable object. Spark provides this functionality as a lazily evaluated map function that can be used with Redundant Distributed Datasets (RDDs). This map function was used for purposes such as normalizing the stock prices, processing the calculated returns for each strategy etc.

C. For-each sequential processing of an RDD

Finally, we used the foreach sequential processing of the stock price RDD in order to apply the actual trading strategy rules to the prices. This sequential processing is similar to a reduce operation, in that it must be the final action performed on an RDD and is sequential. Unlike the reduce operation, the foreach function does not return a single value but instead performs some action on each element.

V. SOFTWARE PACKAGE DESCRIPTION

The software package that we have open sourced is written in Python. It contains scripts corresponding to each of the modules shown in Figure 2. The scripts are named in a self explanatory fashion. Our web application has an interface for charting the returns and the benchmark. A

table_a.csv	table_aon.csv	table_bwa.csv	table_col.csv	table_dps.csv	table_fhn.csv	table_hd.csv	table_joy.csv	table_mas.csv	table_nflx.csv	table_pfg.csv	table_rsg.csv	table_te.csv	table_vtr.csv
table_aa.csv	table_apa.csv	table_bxp.csv	table_cop.csv	table_dri.csv	table_fis.csv	table_hes.csv	table_jpm.csv	table_mat.csv	table_nfx.csv	table_pg.csv	table_rtn.csv	table_teg.csv	table_vz.csv
table_abpl.csv	table_apc.csv	table_c.csv	table_cost.csv	table_dte.csv	table_fisv.csv	table_hig.csv	table_jwn.csv	table_mcd.csv	table_ni.csv	table_pgr.csv	table_s.csv	table_tel.csv	table_wag.csv
table_abbv.csv	table_apd.csv	table_ca.csv	table_cov.csv	table_dtv.csv	table_fitb.csv	table_hnz.csv	table_k.csv	table_mchp.csv	table_nke.csv	table_ph.csv	table_sai.csv	table_ter.csv	table_wat.csv
table_abc.csv	table_aph.csv	table_cag.csv	table_cpb.csv	table_duk.csv	table_flir.csv	table_hog.csv	table_key.csv	table_mck.csv	table_noc.csv	table_phm.csv	table_sbx.csv	table_tgt.csv	table_wdc.csv
table_abt.csv	table_apol.csv	table_cah.csv	table_crm.csv	table_dva.csv	table_flr.csv	table_hon.csv	table_kim.csv	table_mco.csv	table_nov.csv	table_pki.csv	table_scg.csv	table_thc.csv	table_wec.csv
table_ace.csv	table_arg.csv	table_cam.csv	table_csc.csv	table_dvn.csv	table_fls.csv	table_hot.csv	table_klac.csv	table_mdz.csv	table_nrg.csv	table_pld.csv	table_schw.csv	table_tif.csv	table_wfc.csv
table_acn.csv	table_ati.csv	table_cat.csv	table_cscs.csv	table_ea.csv	table_fmc.csv	table_hp.csv	table_kmb.csv	table_mdt.csv	table_nsc.csv	table_pll.csv	table_se.csv	table_tjx.csv	table_wfm.csv
table_act.csv	table_avb.csv	table_cb.csv	table_cscx.csv	table_ebay.csv	table_fosl.csv	table_hpg.csv	table_kmi.csv	table_met.csv	table_ntap.csv	table_pm.csv	table_see.csv	table_tmk.csv	table_whr.csv
table_adbe.csv	table_ayp.csv	table_cbg.csv	table_ctas.csv	table_ecl.csv	table_frx.csv	table_hrb.csv	table_kmx.csv	table_mhfi.csv	table_ntrs.csv	table_pnc.csv	table_shw.csv	table_tmo.csv	table_wip.csv
table_adi.csv	table_ayy.csv	table_cbs.csv	table_ctl.csv	table_ed.csv	table_fslr.csv	table_hrl.csv	table_ks.csv	table_mj.csv	table_nu.csv	table_pnr.csv	table_sjal.csv	table_trip.csv	table_wlp.csv
table_adm.csv	table_ayp.csv	table_cce.csv	table_ctsh.csv	table_efx.csv	table_fti.csv	table_hrs.csv	table_kr.csv	table_mkc.csv	table_nue.csv	table_pnw.csv	table_sjm.csv	table_trow.csv	table_wmb.csv
table_adp.csv	table_azo.csv	table_cci.csv	table_ctxs.csv	table_eix.csv	table_ftr.csv	table_hsp.csv	table_krft.csv	table_mmc.csv	table_nvda.csv	table_pom.csv	table_sib.csv	table_trv.csv	table_wmt.csv
table_adsk.csv	table_ba.csv	table_ccl.csv	table_cvc.csv	table_el.csv	table_gas.csv	table_hst.csv	table_kss.csv	table_mmm.csv	table_nml.csv	table_ppg.csv	table_slm.csv	table_tsn.csv	table_wmt.csv
table_adt.csv	table_bac.csv	table_celg.csv	table_cvh.csv	table_emc.csv	table_gci.csv	table_hsy.csv	table_l.csv	table_mnst.csv	table_nwso.csv	table_ppl.csv	table_sna.csv	table_tso.csv	table_wpo.csv
table_aee.csv	table_bax.csv	table_cern.csv	table_cvs.csv	table_enn.csv	table_gd.csv	table_hum.csv	table_leg.csv	table_mo.csv	table_nyx.csv	table_prgo.csv	table_sndk.csv	table_tss.csv	table_wpx.csv
table_aep.csv	table_bbb.csv	table_cfm.csv	table_cvx.csv	table_enn.csv	table_ge.csv	table_ibm.csv	table_len.csv	table_molx.csv	table_oi.csv	table_pru.csv	table_sni.csv	table_twc.csv	table_wu.csv
table_aes.csv	table_bbt.csv	table_cfn.csv	table_d.csv	table_eog.csv	table_gild.csv	table_ice.csv	table_lh.csv	table_mon.csv	table_nke.csv	table_psa.csv	table_so.csv	table_twx.csv	table_wy.csv
table_aet.csv	table_bby.csv	table_chk.csv	table_dd.csv	table_eqr.csv	table_gis.csv	table_fff.csv	table_life.csv	table_mon.csv	table_nke.csv	table_psa.csv	table_so.csv	table_twx.csv	table_wy.csv
table_afl.csv	table_bcr.csv	table_chw.csv	table_de.csv	table_eqt.csv	table_glw.csv	table_tgt.csv	table_lil.csv	table_mpc.csv	table_orcl.csv	table_pvh.csv	table_spl.csv	table_txn.csv	table_wyn.csv
table_agm.csv	table_bdx.csv	table_ct.csv	table_dell.csv	table_esrx.csv	table_gme.csv	table_intc.csv	table_litc.csv	table_mrk.csv	table_orly.csv	table_pwr.csv	table_spl.csv	table_txn.csv	table_wyn.csv
table_aig.csv	table_beam.csv	table_cinf.csv	table_dfs.csv	table_esv.csv	table_gm.csv	table_intu.csv	table_lly.csv	table_mro.csv	table_oxy.csv	table_pax.csv	table_sre.csv	table_unh.csv	table_xel.csv
table_aiv.csv	table_ben.csv	table_cl.csv	table_dfs.csv	table_etfc.csv	table_gm.csv	table_intu.csv	table_lly.csv	table_mro.csv	table_oxy.csv	table_pax.csv	table_sre.csv	table_unh.csv	table_xel.csv
table_aiz.csv	table_bfb.csv	table_clf.csv	table_dg.csv	table_etn.csv	table_gpc.csv	table_ipg.csv	table_lnc.csv	table_msi.csv	table_payx.csv	table_pbd.csv	table_qcom.csv	table_stj.csv	table_unp.csv
table_akm.csv	table_bhi.csv	table_clx.csv	table_dgx.csv	table_etr.csv	table_gps.csv	table_ir.csv	table_lnc.csv	table_msi.csv	table_payx.csv	table_pbd.csv	table_qcom.csv	table_stj.csv	table_unp.csv
table_all.csv	table_bib.csv	table_cma.csv	table_dhi.csv	table_ew.csv	table_gpm.csv	table_irm.csv	table_lo.csv	table_mtb.csv	table_pbi.csv	table_pcar.csv	table_r.csv	table_stx.csv	table_urbn.csv
table_altr.csv	table_bk.csv	table_cmca.csv	table_dhr.csv	table_exc.csv	table_gs.csv	table_isrg.csv	table_low.csv	table_mtb.csv	table_pbi.csv	table_pcar.csv	table_r.csv	table_stx.csv	table_urbn.csv
table_alxn.csv	table_bll.csv	table_cme.csv	table_dis.csv	table_expe.csv	table_gw.csv	table_itw.csv	table_lrcx.csv	table_mur.csv	table_pcl.csv	table_pcln.csv	table_rdc.csv	table_swk.csv	table_utm.csv
table_amat.csv	table_bll.csv	table_cmg.csv	table_disca.csv	table_expe.csv	table_gw.csv	table_itw.csv	table_lrcx.csv	table_mur.csv	table_pcl.csv	table_pcln.csv	table_rdc.csv	table_swk.csv	table_utm.csv
table_ama.csv	table_bmc.csv	table_cmi.csv	table_diph.csv	table_fast.csv	table_har.csv	table_jci.csv	table_ltd.csv	table_myl.csv	table_pcp.csv	table_rhi.csv	table_swy.csv	table_var.csv	table_yum.csv
table_amgn.csv	table_bms.csv	table_cms.csv	table_dltr.csv	table_fox.csv	table_has.csv	table_jcp.csv	table_luv.csv	table_nbl.csv	table_pcs.csv	table_rht.csv	table_syk.csv	table_vfc.csv	table_zion.csv
table_amp.csv	table_bmy.csv	table_cnp.csv	table_dnb.csv	table_fdx.csv	table_hban.csv	table_jdsu.csv	table_lyb.csv	table_nbr.csv	table_pdc.csv	table_rk.csv	table_sym.csv	table_vio.csv	table_zmh.csv
table_amt.csv	table_brcm.csv	table_cnx.csv	table_dnr.csv	table_fdx.csv	table_hban.csv	table_jdsu.csv	table_lyb.csv	table_nbr.csv	table_pdc.csv	table_rk.csv	table_sym.csv	table_vio.csv	table_zmh.csv
table_amz.csv	table_brb.csv	table_cof.csv	table_dov.csv	table_fdx.csv	table_hban.csv	table_jdsu.csv	table_lyb.csv	table_nbr.csv	table_pdc.csv	table_rk.csv	table_sym.csv	table_vio.csv	table_zmh.csv
table_an.csv	table_bsx.csv	table_cog.csv	table_dov.csv	table_fdx.csv	table_hban.csv	table_jdsu.csv	table_lyb.csv	table_nbr.csv	table_pdc.csv	table_rk.csv	table_sym.csv	table_vio.csv	table_zmh.csv
table_anf.csv	table_btu.csv	table_coh.csv	table_dow.csv	table_ffiv.csv	table_hcp.csv	table_jnpr.csv	table_mar.csv	table_nem.csv	table_pfe.csv	table_rrc.csv	table_tdc.csv	table_vrsn.csv	

Figure 1. HDFS directory listing showing the flat directory structure for the symbol price data from the QuantQuote dataset.

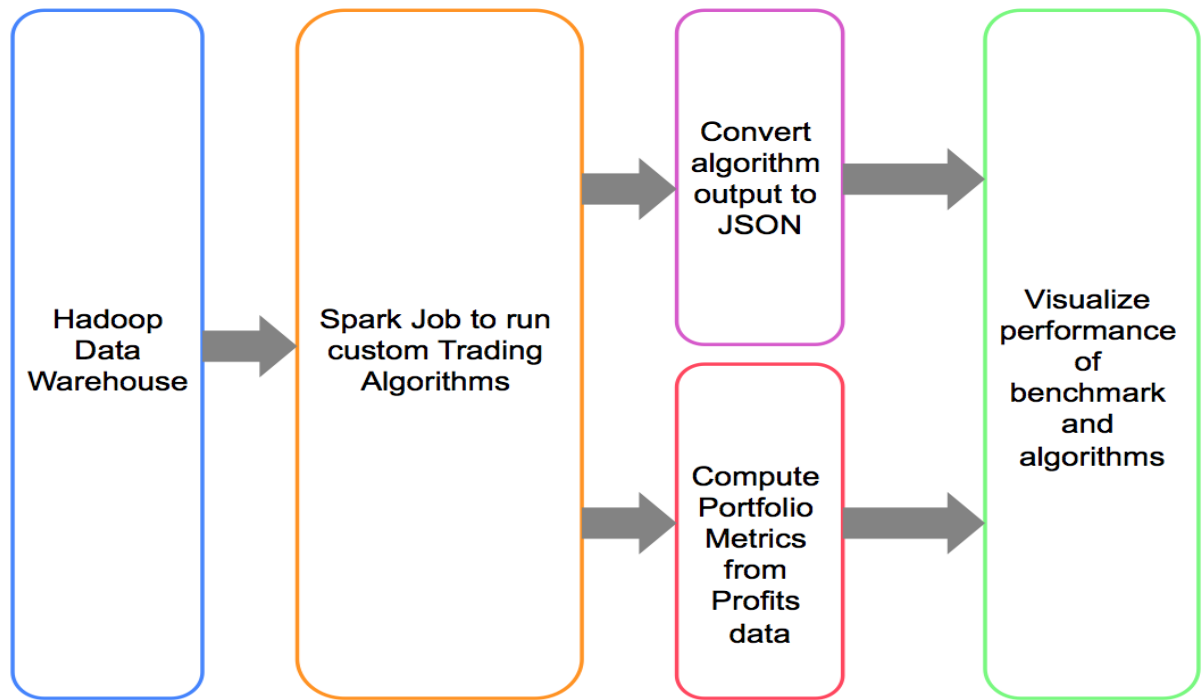


Figure 2. Schematic depicting the overall system organization

SP500 Project

View SP500 quote

WIN (WINDSTREAM HOLDINGS INC)

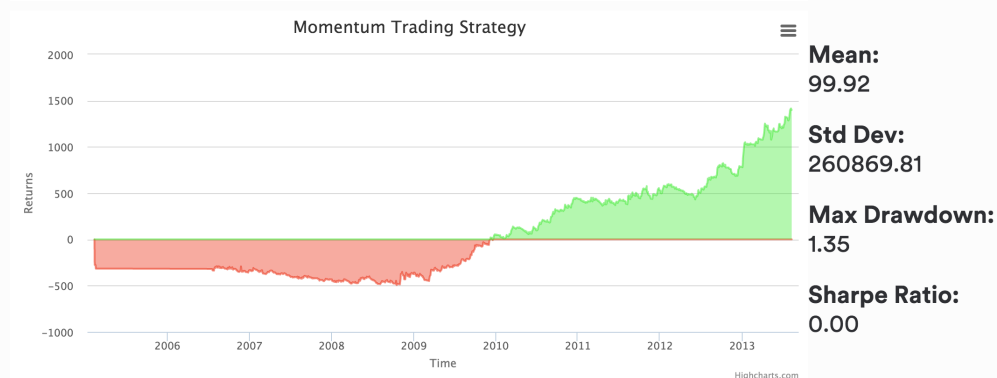
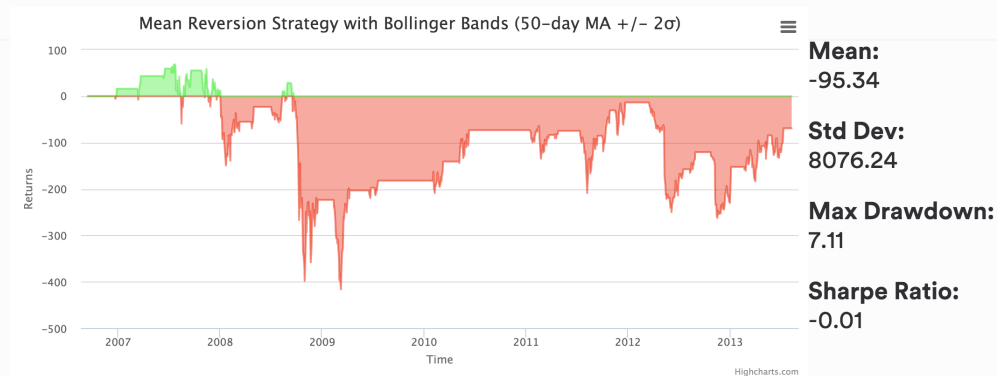


Figure 3. Image of the entire User Interface

screenshot of the overall UI for our app is shown in Figure 3.

The dropdown box at the top of the page enables the user to select a single stock symbol that is to be traded using the user defined strategies. The first chart at the top is the graph depicting the stock price time series. Below this first chart are the charts for all the user defined strategies loaded onto the engine. We used two specific strategies for testing our engine. These strategies are shown and described in more detail in the next section.

VI. EXPERIMENT RESULTS

We used two distinct trading strategies to test our trading engine. These are described below:

A. *Mean Reversion with Bollinger Bands*

The mean reversion strategy assumes that a stock price can be modeled as a mean reverting stochastic process. Every day, a moving average and standard deviation is calculated over the prices from the last five days. The bands two standard deviations from the mean are used to generate buy and sell signals depending on whether the stock prices crosses these thresholds. This strategy worked well on our engine and was easy to implement within our framework. A screenshot showing the returns for this algorithm are shown in Figure 4.

B. *Momentum Trading*

The momentum trading strategy attempts to ride trends in the stock price. If a decreasing trend is noticed, the stock is deemed cold and sold. In the opposite case, the stock is deemed hot and is bought. This strategy shows staggeringly large returns on most stock symbols. This strategy was also extremely easy to implement in our framework and worked in near real time. Figure 5. shows the return chart for this strategy.

VII. CONCLUSION

We were able to implement a complete trading platform based on Apache Hadoop and Apache Spark. We were able to test popular real world trading algorithms and achieve visual as well as numerical insight into their performance on any of 500 stock symbols. We plan to extend the features of our system in the future so as to build a more flexible and universal securities trading engine. Some ideas we have are as follows

A. *Multiple security types*

There are many different sorts of financial instruments that can be traded algorithmically, such as options, bonds etc and we wish to enable trading of these on our engine.

B. *Live testing*

We would like to enable the testing of algorithms in live, real-time stock tick data

C. *Custom visualizations*

We would like to allow uses to plot custom indicators and series on the charts and also to extend charting to portfolios with more than a single stock.

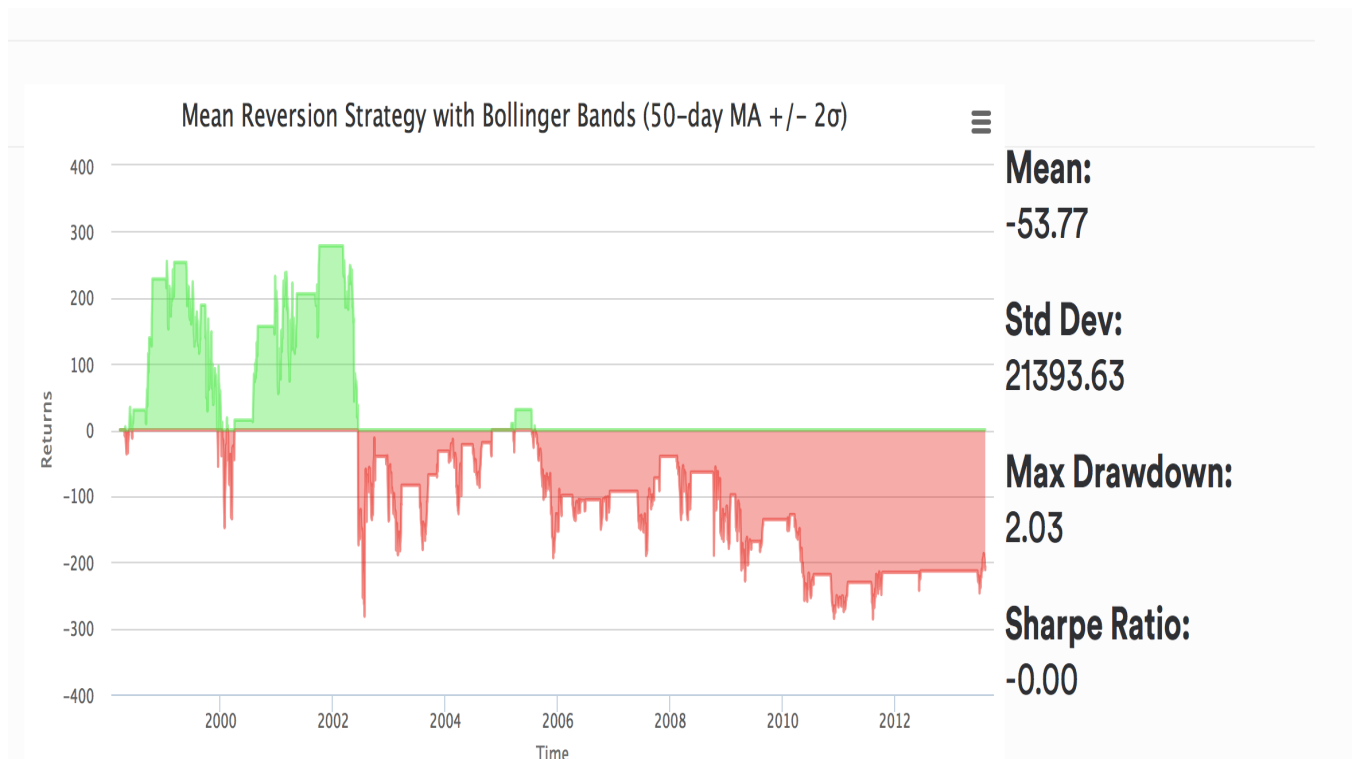


Figure 4. Returns from the Mean Reversion Trading Algorithm

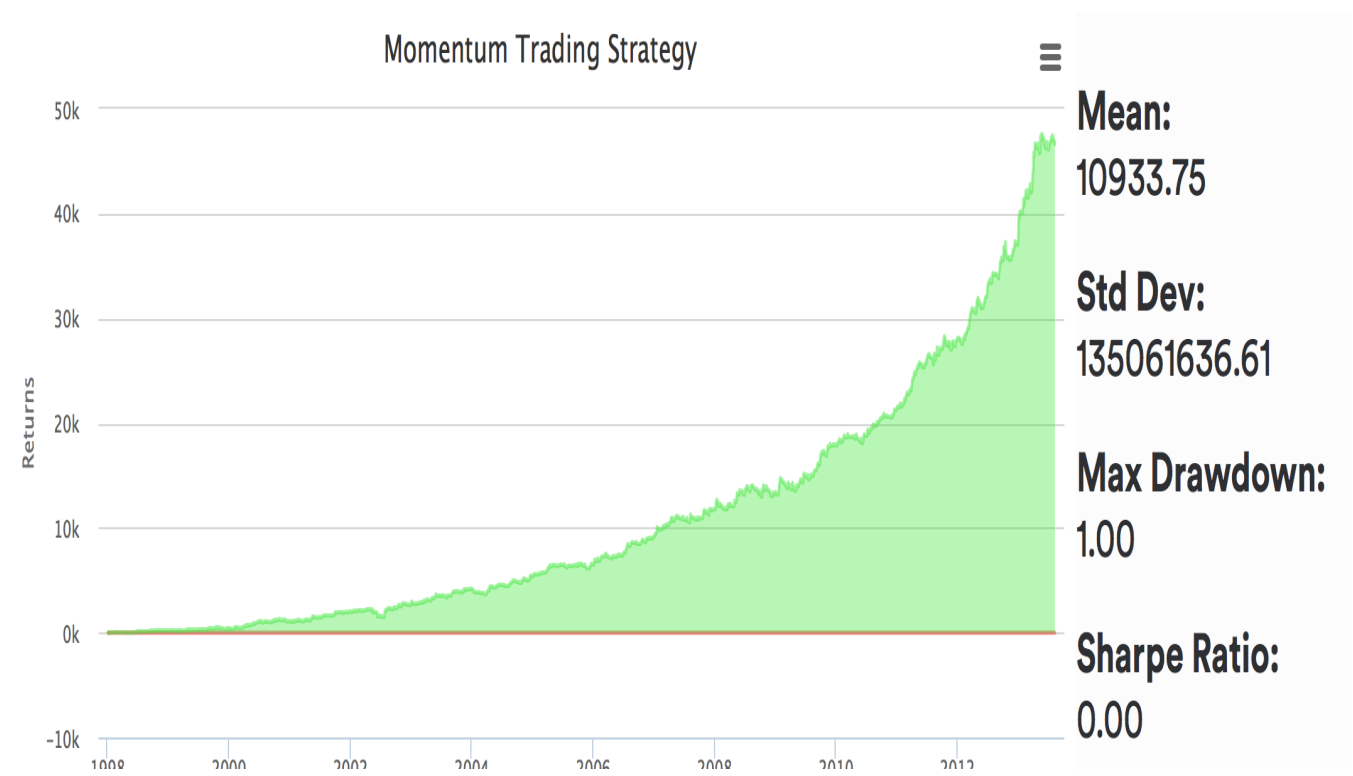


Figure 5. Returns from the Momentum Trading Algorithm

APPENDIX
REFERENCES

- [1] Apache Hadoop (<http://hadoop.apache.org>)
- [2] Quantopian (<https://www.quantopian.com>)
- [3] QuantQuote (<https://quantquote.com>)
- [4] Apache Spark (<http://spark.apache.org>)