



## **A Bayesian Approach to Stock Trading**

Item Type	Masters Project
Authors	Bosco Dias, Keith
Download date	20/06/2024 01:25:08
Link to Item	<a href="http://hdl.handle.net/20.500.12648/8623">http://hdl.handle.net/20.500.12648/8623</a>

# **A Bayesian Approach to Stock Trading**

M.S. Project

**Keith Bosco Dias**

Supervised by:

**Dr. Bruno Andriamanalimanana**

December 2021



Master of Science in Computer and Information Sciences

Department of Computer Science

State University of New York Polytechnic Institute

## **A Bayesian Approach to Stock Trading**

A project by Keith Dias (U00331344), approved and recommended for acceptance as a final project in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences, SUNY Polytechnic Institute.

01/17/2022

\_\_\_\_\_  
Date

\_\_\_\_\_  
Dr. Bruno Andriamanalimana

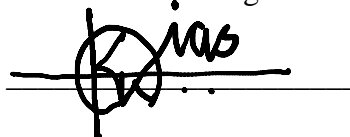
\_\_\_\_\_  
Dr. Jorge Novillo

\_\_\_\_\_  
Dr. Chen-Fu Chiang



### **STUDENT DECLARATION**

I declare that this project is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.



Keith Dias

## **Abstract**

This project focuses on using Probabilistic Programming and more specifically using the Bayesian approach to devise an effective strategy to trade. This project does so by implementing a novel model on co-integration for pairs-trading using probabilistic programming. As opposed to using the traditional and simpler frequentist approach for pair determination I have implemented a more sophisticated Bayesian approach for pair trading using probabilistic programming. Pair trading is a market neutral strategy that enables traders to profit from virtually any market conditions be it uptrend, downtrend, or sideways movement. It is characterized as statistical arbitrage and convergence trading strategy. Pair Trading combined with co-integration as criteria makes for a successful and reliable trading strategy. Unlike simpler frequentist cointegration tests, the Bayesian approach allows to monitor the relationship between a pair of equities over time, which further allows to follow pairs whose cointegration parameters change steadily or abruptly. Bayesian statistics also accounts for uncertainty in making predictions. It provides with mathematical tools to update beliefs about random events considering seeing new data or evidence about those events and it can do without having the need for a large dataset. It interprets probability as a measure of believability or confidence that an individual might possess about the occurrence of a particular event while including uncertainty in the equation. Along with a mean-reversion trading algorithm, this approach can be effectively used as a viable trading strategy, open for further evaluation and risk management.

**Keywords:** Bayesian analysis, Pair trading, cointegration, statistical arbitrage, mean reversion.

## **Acknowledgement**

Throughout the process of completing my capstone project I have received a great deal of support and assistance.

I would first like to thank my supervisor, Dr. Bruno Andriamanalimanana, whose expertise was invaluable in formulating the research questions and methodology. The subjects probabilistic programming and machine learning that you have instructed me in inspired me to do this work. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would like to acknowledge my colleagues from my internship Xpansiv Data Systems for their wonderful collaboration. I would particularly like to single out my manger Mr. Steve Leishman and Data Application Lead Gautham Ghorla at Xpansiv, for their input and introducing me to the technologies and techniques I have come to know about and use in my project and for their patient support and all the opportunities I was given to further my research.

I would also like to thank the members of my review panel, Dr. Jorge Novillo and Dr. Chen-Fu Chiang, for their valuable guidance throughout my master studies. You have guided me wonderfully and provided me with the tools that I needed to choose the right direction and successfully complete my project.

In addition, I would like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, I could not have completed this dissertation without the support of my friend, Nikhil Reddy, who provided stimulating discussions as well as happy distractions to rest my mind outside of my project.

I am grateful to God for helping me achieve this work and for helping me to learn and gain knowledge and wisdom throughout this journey.

Keith Bosco Dias

# Table Of Contents

<b>Chapter 1 Introduction .....</b>	<b>7</b>
1.1 Probabilistic Programming .....	7
1.2 Bayesian Learning .....	9
1.3 Algorithmic Trading .....	10
1.3.1 Pairs Trading .....	11
1.3.2 Co-integrated Stock Pairs:.....	13
<b>Chapter 2 Related study .....</b>	<b>15</b>
2.1 Motivation and Initial Research .....	15
2.2 Related Work .....	15
<b>Chapter 3 Design and Implementation .....</b>	<b>17</b>
3.1 Tools and Technology used .....	17
3.1.1 Python Libraries used: .....	17
3.1.2 Google Colab.....	19
3.2 Dataset .....	19
3.2.1 Supply Chain Dataset.....	19
3.2.2 Market Data.....	22
3.3 Bayesian Modelling .....	23
<b>Chapter 4 Long Short Trading Strategy .....</b>	<b>27</b>
<b>Chapter 5 Back Testing .....</b>	<b>30</b>
<b>Chapter 6 Conclusion.....</b>	<b>33</b>
6.1 Accomplishments and lessons learned .....	33
6.2 Future work.....	33
<b>References .....</b>	<b>34</b>
<b>Appendix .....</b>	<b>36</b>

# List Of Figures

1.1 Probabilistic programming description [1] .....	7
1.2 Probabilistic programming description [2] .....	8
1.3 Bayes Theorem.....	10
1.4 Algorithmic Trading.....	11
1.5 Market Close Data.....	21

# Chapter 1 Introduction

## 1.1 Probabilistic Programming

Probabilistic reasoning [1] has been used for a wide variety of tasks such as predicting stock prices, recommending movies, diagnosing computers, detecting cyber intrusions and image detection. However, until recently (partially due to limited computing power), probabilistic programming was limited in scope, and most inference algorithms had to be written manually for each task.

The idea behind Probabilistic programming is to bring the inference algorithms and theory from statistics combined with formal semantics, compilers, and other tools from programming languages to build efficient inference evaluators for models and applications from Machine Learning. In other words, probabilistic programming is a tool for statistical modeling. The idea is to borrow lessons from the world of programming languages and apply them to the problems of designing and using statistical models.

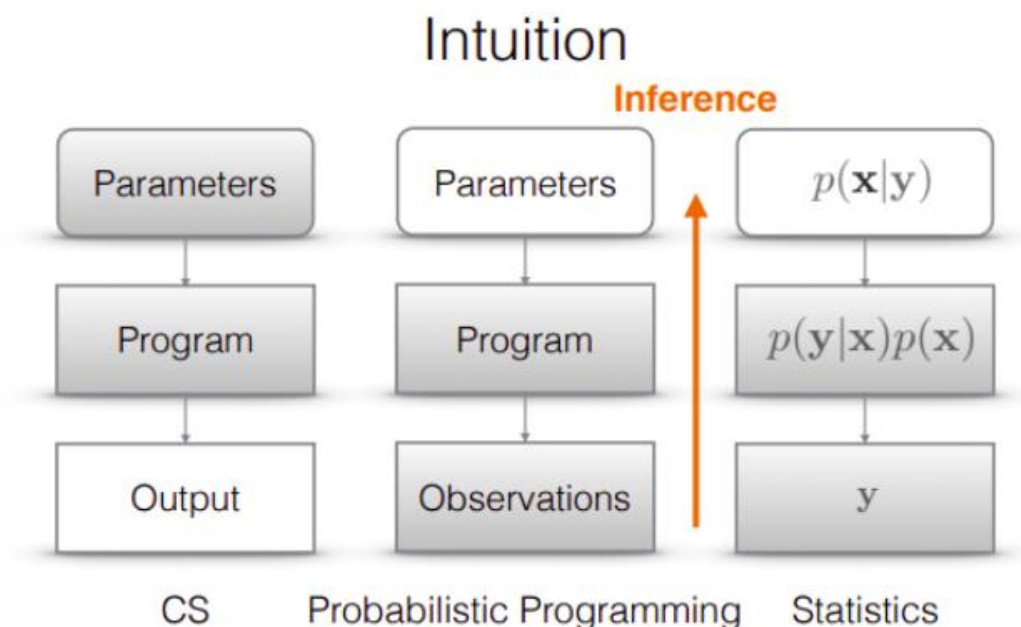
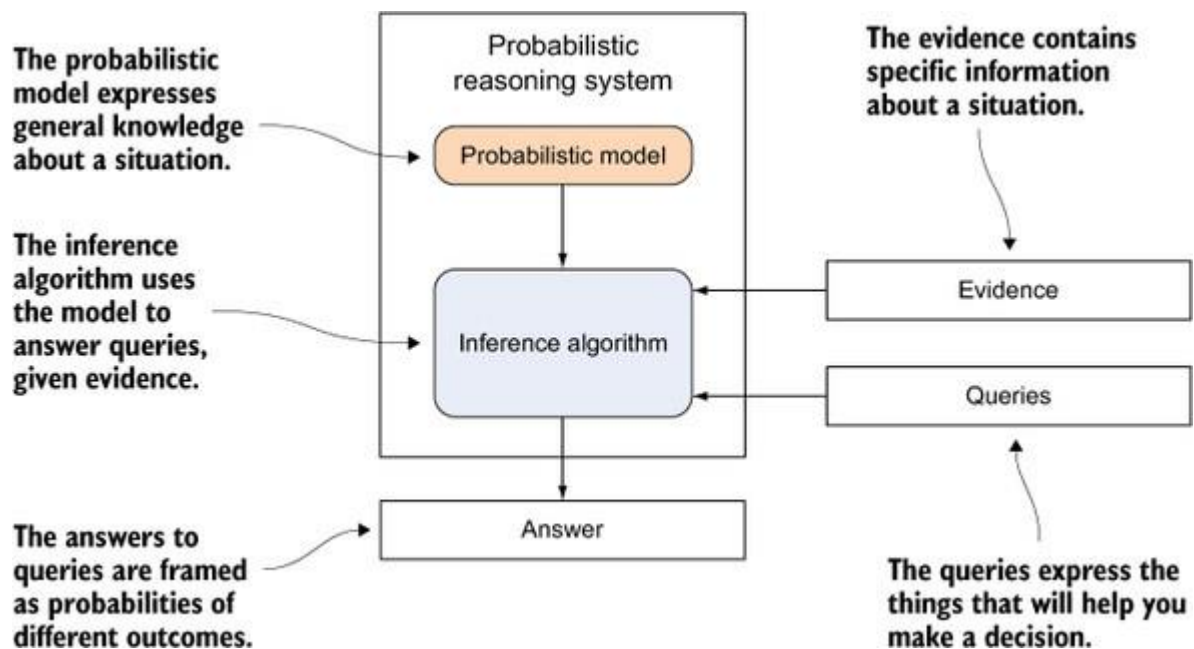


Fig1 Probabilistic programming description [1]



In the above figure we see a typical computer science programming pipeline: Write a program, specify the values of its arguments then evaluate the program to produce an output. The right-hand side illustrates the approach taken to modeling in statistics: Start with the output, the observations or data  $Y$ , then specify an abstract generative model  $p(X,Y)$ , often denoted mathematically, and finally use algebra and inference techniques to characterize the posterior distribution,  $p(X | Y)$ , of the unknown quantities in the model given the observed quantities. Whereas in Probabilistic programming: a programming language for model definitions and statistical inference algorithms for computing the conditional distribution of the program inputs that could have given rise to the observed program output.



*Fig2 Probabilistic programming description [2]*

In probabilistic reasoning, a model is created that captures all the relevant general knowledge of a specific domain in quantitative, probabilistic terms. Then, for a particular situation, the model is applied to any specific information available to draw conclusions. This specific information is called the evidence. The conclusions drawn can help make decisions. The relationship between the model, the information provided, and the answers to queries is well defined mathematically by the laws of probability. The process of using the model to answer queries based on the evidence is called probabilistic inference.

## 1.2 Bayesian Learning

Machine learning [2] models are usually developed from data as deterministic machines that map input to output using a point estimate of parameter weights calculated by maximum-likelihood methods. However, there is a lot of statistical flukes going on in the background. For instance, a dataset itself is a finite random set of points of arbitrary size from an unknown distribution superimposed by additive noise, and for such a particular collection of points, different models (i.e. different parameter combinations) might be reasonable. Hence, there is some uncertainty about the parameters and predictions being made. Bayesian statistics provides a framework to deal with the so-called aleatoric and epistemic uncertainty.

Time series data includes many kinds of real experimental data taken from various domains such as finance, medicine, scientific research (e.g., global warming, speech analysis, earthquakes), etc. Time series forecasting has many real applications in various areas such as forecasting of business (e.g., sales, stock), weather, disease, and others [2]. Statistical modeling and inference (e.g., ARIMA model) [1][2] is one of the popular methods for time series analysis and forecasting.

There are two statistical inference methods:

- Bayesian inference
- Frequentist inference

The philosophy of Bayesian inference is to consider probability as a measure of believability in an event and use Bayes' theorem to update the probability as more evidence or information becomes available, while the philosophy of frequentist inference considers probability as the long-run frequency of events.

probability a hypothesis is true given the evidence

probability a hypothesis is true (before any evidence is present)

probability of seeing the evidence if the hypothesis is true

probability of observing the evidence

$$P(H/E) = \frac{P(H) P(E/H)}{P(E)}$$

Fig 3 Bayes Theorem

Frequentist inference is usually used when a large number of data samples are available. In contrast, the Bayesian inference can be applied to both large and small datasets.

There are two major types of uncertainty we can model in Bayesian modeling, which are aleatoric uncertainty and epistemic uncertainty. Aleatoric *uncertainty*, which is also called data-inherent uncertainty, denotes the intrinsic noise in the observations. *Epistemic uncertainty* is related to model parameters, and it arises when the model is not suitably trained due to the lack of training data.

### 1.3 Algorithmic Trading

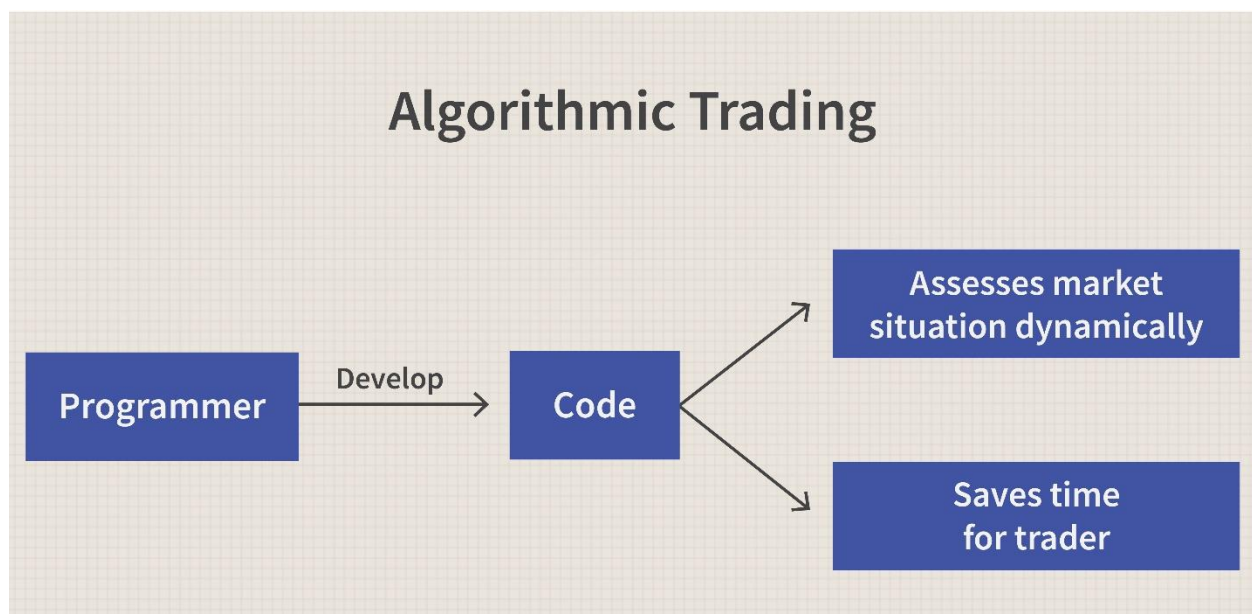
Trading decisions often depend on the trader's subjective belief of the distribution of the asset price on a given future date. For example, if a trader anticipates a big price movement for a company stock after its earnings announcement, then perhaps a long straddle position makes sense. There are many instances like this.

And as time progresses, the trader will learn more about the price distribution by observing price fluctuations, which in turn will inform trading decisions.

Mathematically, the trader's market view can be described by a prior distribution of the asset price on some future date. The price process reveals itself over time, but there is also uninformative noise embedded in the asset price dynamics. This leads to the question: how to build a model that appropriately reflects the trader's market view dynamically over time?

For this we use Algorithmic trading which is a process for executing orders utilizing automated and pre-programmed trading instructions to account for variables such as price, timing and volume.

Algorithmic trading [3] makes use of complex formulas, combined with mathematical models and human oversight, to make decisions to buy or sell financial securities on an exchange. Algorithmic traders often make use of high-frequency trading technology, which can enable a firm to make tens of thousands of trades per second. Algorithmic trading can be used in a wide variety of situations including order execution, arbitrage, and trend trading strategies.



*Fig 4 Algorithmic Trading*

### **1.3.1 Pairs Trading**

A pairs trade is a trading strategy that involves matching a long position with a short position in two stocks with a high correlation. Pairs trading [4] is a classic approach to statistical arbitrage that has a long-standing track record. It is a market neutral trading strategy enabling traders to profit from virtually any market conditions: uptrend, downtrend, or sideways movement. This strategy is categorized as a statistical arbitrage and convergence trading strategy. Pair trading was pioneered by Gerry Bamberger and later led by Nunzio Tartaglia's quantitative group at Morgan Stanley in the 1980s.

The strategy monitors performance of two historically correlated securities. When the correlation between the two securities temporarily weakens, i.e. one stock moves up while the other moves down, the pairs trade would be to short the outperforming stock and to long the underperforming one, betting that the "spread" between the two would eventually converge.[5] The divergence within a pair can be caused by temporary supply/demand changes, large buy/sell orders for one security, reaction for important news about one of the companies, and so on. Pairs trading strategy demands good position sizing, market timing, and decision-making skill. Although the strategy does not have much downside risk, there is a scarcity of opportunities, and, for profiting, the trader must be one of the first to capitalize on the opportunity.

Traditionally pairs for pairs trading have been determined using correlation. However, correlation is not a strong statistically property as it is not consistent. Instead, this project uses cointegration as a criterion for pairs determination. This has been done usually by frequentist statistical cointegration tests, such as the Engle-Granger two-step test. However, in this project a more sophisticated Bayesian approach to pairs trading using probabilistic programming has been implemented. Unlike simpler frequentist cointegration tests, Bayesian approach allows to monitor the relationship between a pair of equities over time, which allows to follow pairs whose cointegration parameters change steadily or abruptly. Combining with a simple mean-reversion trading algorithm, this demonstrates to be a viable theoretical trading strategy, ready for further evaluation and risk management.

### Benefits of Pairs Trading:

- The pairs trade helps to hedge sector- and market-risk. For example, if the whole market crashes, and the two stocks plummet along with it, the trade should result in a gain on the short position and a negating loss on the long position, leaving the profit close to zero in spite of the large move.
- Pairs trade is a mean-reverting strategy, betting that the prices will eventually revert to their historical trends.

- Pairs trade is a substantially self-funding strategy since the short sale proceeds may be used to create the long position.

### **1.3.2 Co-integrated Stock Pairs:**

Cointegration is also criteria for a pairs trade, and cointegration is oftentimes the more reliable strategy for successful pairs trading. Cointegration describes the distance between the two assets in price over time, whereas correlation describes the tendency to move in similar directions.

Cointegration [6] is a statistical property of a collection (eg:  $X_1, X_2, \dots, X_k$ ) of time series variables. First, a series must be integrated of order  $d$ . Order of integration is defined as the number of times one must take the derivative of a signal to get a stationary signal. When a signal is stationary, none of its statistical moments change over time. Most notably, the mean of the signal is therefore fixed, and any fluctuations in the stock must happen around the mean. Aka, if the signal deviates from the mean, it must revert to the mean. Next, if a linear combination of this collection is integrated of order less than  $d$ , then the collection is said to be co-integrated. Formally, if  $(X, Y, Z)$  are each integrated of order  $d$ , and there exist coefficients  $a, b, c$  such that  $aX + bY + cZ$  is integrated of order less than  $d$ , then  $X, Y$ , and  $Z$  are cointegrated. Cointegration is an important property in contemporary time series analysis. Time series often have trends, they could be either deterministic or stochastic.

In the case of Pairs Trading, it is preferred that a portfolio of two stocks repeatedly revert to a fixed value. A consistent reversion of the portfolio gives the confidence to short the portfolio when it is above the reversion value and to go long when below this reversion value. The mathematical notion of reverting to a fixed value is cointegration and it is stronger and more preferred statistical property as compared to the commonly known property of correlation.

Cointegration is effectively a stronger form of the more common statistical concept of correlation. When stocks are correlated, they tend to move similarly relative to one another (either in the same direction, or in opposition).

1. Correlation has no well-defined relationship with cointegration. Cointegrated series might have low correlation, and highly correlated series might not be cointegrated at all.
2. Correlation describes a short-term relationship between the returns.

3. Cointegration describes a long-term relationship between the prices.

An example of an obvious cointegrated pairs trade would be Pepsi (PEP) and Coca-Cola (KO); Assuming one is unsure of the direction of both stocks, but one is confident that no matter the direction, Coca-Cola will outperform. With this view, one could enter a market neutral position with a long position for Coca-Cola and short position for Pepsi. Even if both stocks went down in price, as long as Pepsi goes down more than Coca-Cola, it will be a profitable trade (provided they were equally weighted positions when the trade was opened).

## **Chapter 2 Related study**

### **2.1 Motivation and Initial Research**

In my second semester of my master's degree, I took a course on Probabilistic Programming and Data science in which I did a project on 'Bayesian Time Series Forecasting for Energy Management'. This work that I did, combined with my experience at Xpansiv Data Systems which is a Trading Firm dealing with ESG commodities led me to pursue this topic. Bayesian probabilistic programming is an amazing tool for analyzing financial data and providing strategies to trade. The technical knowledge gained from this course couple with the business and trading knowledge that I learned from market analysts and traders at Xpansiv helped me to better understand and research this topic.

### **2.2 Related Work**

Gatev, Goetzmann and Rouwenhorts [7] show the benefits of a simple pair trading technique to generate profits over a long period, using distance model. However, Do and Faff [8] document that after 2002 this strategy based upon the distance model was largely unprofitable once trading costs are taken into account and also the distance model is immensely time-varying. Pairs Trading strategy can work on the correlation or co-integration. Cointegration is often times the more reliable strategy for pairs trading. Rad et al. [9] compare various methods such as Distance, co-integration and copula methods and come to the conclusion that pairs trading is the most superior of all pairs trading strategies and especially during turbulent market conditions. In [10] different methods such as using an LST based Deep learning Neural Network based on financial news has proposed, however the accuracy of the model is just 0.635% and thus cannot be relied upon entirely. [11] discusses various Neural Network architectures such as Convolution, however these aren't highly accurate as well. Gatarek et al. [12] discusses various statistical arbitrages based on patterns observed in historical data that are expected to be repeatable. It gives the mathematical proof and approach for using Bayesian analysis for pair selection. It introduces a simulation-based Bayesian estimation procedure that allows us to combine estimation and model uncertainty in a natural way with decision uncertainty associated with a decision process like a trading strategy. [13] discusses using P/E ratio as the basis for an economic prior for the Bayesian Pair Training



Model however in my implementation I have used Suppliers and Customers Data set as a basis for selecting the co-integrated pairs. Economic prior could range from anything such as ESG rating, P/E ratio or Competitors or Similar companies.

## Chapter 3 Design and Implementation

In this project I have implemented python based scientific computing technologies and made use of an equities back testing engine along with the RapidAPI Supply Chain Dataset and S&P 500 Market Close data. A simulation-based Bayesian procedure is used for developing a sound long short trading strategy. Using this model and the proposed inferential technique, it is able to connect estimation and model uncertainty with risk and return of stock trading.

### 3.1 Tools and Technology used

Statistical Analysis is a field that is computationally intensive, and choice of machine greatly affects the speed and efficiency of the model. I had initially used Jupyter Notebook to perform and run the analysis. However this takes up a significant amount of time (approximately more than an hour) on my current MacBook machine. So instead, I decided to use Google colab for executing the code thus utilizing the power of google cloud servers instead of my own machine. This gave a serious bump in performance while also sorting out library versioning issues.

#### 3.1.1 Python Libraries used:

There are multiple Python libraries that can be used to program Bayesian analysis and inference. Such type of programming is called probabilistic programming. For implementing the Bayesian model, I have implemented using PyMC3 since the documentation is more concise and easily implementable.

**PyMC3:** PyMC3 [14] is a new open-source Probabilistic Programming framework written in Python. Providing recent advances in Markov chain Monte Carlo (MCMC) sampling, PyMC3 allows inference on increasingly complex models. PyMC3 features next-generation Markov chain Monte Carlo (MCMC) sampling algorithms such as the No-U-Turn Sampler, a self-tuning variant of Hamiltonian Monte Carlo . This class of samplers works well on high dimensional and complex posterior distributions and allows many complex models to be fit without specialized knowledge about fitting algorithms. HMC and NUTS take advantage of gradient information from the likelihood to achieve much faster convergence than traditional sampling methods, especially for larger models. NUTS also has several self-tuning strategies for adaptively setting the tunable parameters of Hamiltonian Monte Carlo, which means you usually don't need to have specialized knowledge about how the algorithms work.

PyMC3 aims for intuitive and readable, yet powerful syntax that reflects how statisticians describe models. The modeling process generally follows these three steps: 1) Encode a probability model by defining: i) The prior distributions that quantify knowledge and uncertainty about latent variables ii) The likelihood function that conditions the parameters on observed data 2) Analyze the posterior using one of the options described in the previous section: a) Obtain a point estimate using MAP inference b) Sample from the posterior using MCMC methods c) Approximate the posterior using variational Bayes 3) Check your model using various diagnostic tools 4) Generate predictions.

**Theano:** PyMC3 uses Theano as its computational backend for dynamic C compilation and automatic differentiation. Theano is a matrix-focused and GPU-enabled optimization library developed at Yoshua Bengio's Montreal Institute for Learning Algorithms (MILA) that inspired TensorFlow. MILA recently ceased to further develop Theano due to the success of newer deep learning libraries (see chapter 16 for details). PyMC4, planned for 2019, will use TensorFlow instead, with presumably limited impact on the API.

**NumPy:** library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

**Pandas:** software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series

**Seaborn:** data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data

**Sklearn:** library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

**Matplotlib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK

### **3.1.2 Google Colab**

Colaboratory or Colab [15] for short, is a Google Research product. Google Colab is built on top of vanilla Jupyter Notebook, which is built on top of Python that runs entirely in the cloud and allows developers to write and execute Python code through their browser. Google Colab is an excellent tool for deep learning and probability programming tasks that gives free access to Google computing resources such as GPUs and TPUs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook. Anaconda distribution of Jupyter Notebook comes shipped with several pre-installed data libraries, such as Pandas, NumPy, Matplotlib, which is awesome. However, Google Colab, on the other hand, provides even more pre-installed machine learning libraries such as Keras, TensorFlow, and PyTorch. All Google Colab notebooks are saved under your Google Drive account. GPU and TPU acceleration make a huge difference even for some small projects. This is one of the main reasons for me to code all my educational projects on Google Colab.

## **3.2 Dataset**

To perform the Bayesian Modelling that will help us model the pairs trading strategy I have used two datasets as follows:

### **3.2.1 Supply Chain Dataset**

In order to find good pairs trading candidates a potential good source of cointegrated stocks is a supply chain dataset. Oftentimes finding a company's actual suppliers and customers is a very difficult task since many companies do not want to reveal information to competitors. Fortunately, I was lucky to source this dataset from an API marketplace RapidAPI that leverages hybrid data sources to deliver a complete picture of company supply chain ecosystem. This Supply Chain dataset [16] provides a list of suppliers and customers extracted from millions of unstructured documents for a given company ticker from the Russell 3000 Index. It exposes supplier and customer relationships allowing investors to identify hidden exposures. The supply chain includes business associated with a particular product, including companies that assemble

and deliver parts. Thus, it enables investors to construct hedges and build statistical arbitrage strategies for a given corporate supply chain. Below is an example of how to be invoking the API using python requests.

```
url = "https://corporate-supply-chain.p.rapidapi.com/api/v1/resources/supplychain"

querystring = {"ticker": "IBM"}

headers = {
    'x-rapidapi-host': "corporate-supply-chain.p.rapidapi.com",
    'x-rapidapi-key': "e159bd1a20mshac51099c55d67p16ae77jsndcd63e339daa"
}

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)
```

```
[{"Ticker": "CMPGF", "Full Name": "Compass Group PLC", "Relationship": "Suppliers"}, {"Ticker": "HTHIF", "Full Name": "Hitachi, Ltd.", "Relationship": "Suppliers"}]
```

Below Diagram shows the list of suppliers and customers in a pandas data frame. This list could be further filtered using fundamental data, technical indicators, or other sources of alternative data to get a list of good pairs trading candidates. To demonstrate pair trading using co-integration I have chosen IBM as a supplier and AmerisourceBergen Corporation which is a customer of IBM.

	Ticker	Full Name	Relationship
0	CMPGF	Compass Group PLC	Suppliers
1	HTHIF	Hitachi, Ltd.	Suppliers
2	FJTSF	Fujitsu Limited	Suppliers
3	CAJFF	Canon Inc.	Suppliers
4	SKSBF	Skanska AB (publ)	Suppliers
5	RICOF	Ricoh Company, Ltd.	Suppliers
6	SDXOF	Sodexo S.A.	Suppliers
7	SSNLF	Samsung Electronics Co., Ltd.	Suppliers
8	6701.T	NEC Corporation	Suppliers
9	WPPGF	WPP plc	Suppliers
10	VLKPF	Volkswagen AG	Customers
11	PEUGF	Peugeot S.A.	Customers
12	ABC	AmerisourceBergen Corporation	Customers
13	HTHIF	Hitachi, Ltd.	Customers
14	FURCF	Faurecia S.E.	Customers
15	LNVGf	Lenovo Group Limited	Customers
16	WBA	Walgreens Boots Alliance, Inc.	Customers
17	TTRAF	Telstra Corporation Limited	Customers
18	KNYJF	KONE Oyj	Customers
19	MNBEF	MinebeaMitsumi Inc.	Customers
20	INTC	Intel Corporation	Suppliers
21	NVDA	NVIDIA Corporation	Suppliers
22	CSCO	Cisco Systems, Inc.	Customers
23	WIPRO.BO	Wipro Limited	Customers
24	CAPMF	Capgemini SE	Customers
25	BR	Broadridge Financial Solutions, Inc.	Customers
26	RMBS	Rambus Inc.	Suppliers
27	QMQO	Quantum Corporation	Suppliers
28	CDNS	Cadence Design Systems, Inc.	Suppliers
29	JNPR	Juniper Networks, Inc.	Suppliers
30	AKAM	Akamai Technologies, Inc.	Suppliers
31	FFIV	F5 Networks, Inc.	Suppliers
32	CLS	Celestica Inc.	Suppliers
33	KEM	KEMET Corporation	Suppliers
34	AVT	Avnet, Inc.	Customers
35	EFX	Equifax Inc.	Customers
36	JKHY	Jack Henry & Associates, Inc.	Customers
37	CNXN	PC Connection, Inc.	Customers
38	NSIT	Insight Enterprises, Inc.	Customers
39	JNPR	Juniper Networks, Inc.	Customers
40	MRCY	Mercury Systems, Inc.	Customers
41	CSPI	CSP Inc.	Customers
42	SCSC	ScanSource, Inc.	Customers

### 3.2.2 Market Data

In order to get S&P 500 market close data I have pulled in 5 years' worth of daily close data from IEX Cloud, a financial data vendor. IEX Cloud is a platform that makes financial data and services accessible to everyone. This Cloud API is based on REST, has resource-oriented URLs and returns JSON-encoded responses, and standard HTTP response codes. Below is an invocation of the API using python http requests.

```
IEX_API_Key = 'pk_76f28d2269784cc3940e69659f86c033'

tickers = [ 'IBM', 'ABC', ]

#Create an empty string called 'ticker_string' that we'll add tickers and commas to
ticker_string = ''

#Loop through every element of 'tickers' and add them and a comma to ticker_string
for ticker in tickers:
    ticker_string += ticker
    ticker_string += ','

#Drop the last comma from 'ticker_string'
ticker_string = ticker_string[:-1]

#Endpoint and years strings
endpoints = 'chart'
years = '5'

#Interpolating endpoint into the HTTP_request string
HTTP_request = f'https://cloud.iexapis.com/stable/stock/market/batch?symbols={ticker_string}&types={endpoints}&range={years}&y&cache=true&token={IEX_API_Key}'

#Send HTTP request to IEX Cloud API and store the response in pandas DataFrame
stock_data = pd.read_json(HTTP_request)
```

IEX Cloud provides both real-time stock prices and 15-minute delayed stock prices during market trading hours for U.S. stocks and ETFs. Real-time prices are based on all trades that occur on the Investors Exchange (IEX). 15-minute delayed stock prices reflect activity from all U.S. exchanges and is provided by the Securities Information Processor / Consolidate Tape Association (SIP).

While real-time prices for Nasdaq-listed stocks are available to all users, 15-minute delayed price data for Nasdaq-listed securities requires UTP authorization.

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146.0	10062.83
1	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515.0	7407.06
2	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786.0	3815.79
3	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590.0	3960.27
4	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749.0	3486.05

*Fig 5 Market Closing Data*

Further this data frame is filtered for the for IBM and AmerisourceBergen Corporation and for their closing price over the last 5 years. A date column is added and set as the index and then both companies are plotted against a graph of date vs price using matplotlib as shown below.



*Potentially Cointegrated Stocks*

### 3.3 Bayesian Modelling

The Bayesian model [18] of stocks does not initially make assumptions about the order of integration of the two stocks  $S_1$  and  $S_2$ . So, in order to simply, the assumption is made that some linear combination of a cointegrated pair of stocks will, itself, be stationary, and normally distributed, which is defined as the distribution  $p$ :

$$p = \alpha_1 S_1 + \alpha_2 S_2$$

$$p \sim N(\mu_p, \sigma_p^2)$$



Under classical frequentist definitions of cointegration, the two coefficients  $\alpha_1$  and  $\alpha_2$  are constants over all time. Subsequently, this could be a dangerous assumption to make, as stocks may go in and out of cointegration as internal business conditions or external market conditions change over time.

To address this, a novel model of cointegration using probabilistic programming is proposed. Instead of assuming  $\alpha_1$  and  $\alpha_2$  are constants, it is presumed that at least one of them is a time-varying random variables, where the time step will be defined as subscript  $t$ . To further simplify the above equation for  $p$ , it is assumed  $\alpha_2=1$  over all time, and  $\alpha_1$  is renamed to the time varying variable  $\beta_t$ , to get the following final model for our cointegrated pairs:

$$p = \beta_t S_1 + S_2$$

$$p \sim N(\mu_p, \sigma_p^2)$$

For  $\beta_t$  to be a time varying random variable, it is preferable to choose a non-stationary distribution, so it can move gradually over time. For this Brownian motion is chosen, given its simplicity of implementation. Though  $\beta_t$  between two stocks could be a more complex stochastic time-series process, it is like an autoregressive signal. Since the plan is to model  $\beta_t$  as Brownian noise, it gives the following recursively defined statistical distribution:

$$\beta_t \sim N(\beta_{t-1}, \sigma_{2\alpha}^2)$$

Finally, to help with convergence, one last simplifying assumption is made and  $\mu_p=0$  and that  $\sigma_p^2$  is sufficiently small that we can approximate  $p$  to be simply zero. This leaves us with a highly simplified equation:  $S_2 = -\beta_t S_1$

```

] with pm.Model() as model:

    # inject external stock data
    stock1 = th.shared(data1)
    stock2 = th.shared(data2)

    # define our cointegration variables
    beta_sigma = pm.Exponential('beta_sigma', 50.)
    beta = pm.GaussianRandomWalk('beta', sd=beta_sigma,
                                shape=data1.shape[0])

    # with our assumptions, cointegration can be reframed as a regression problem
    stock2_regression = beta * stock1

    # Assume prices are Normally distributed, the mean comes from the regression.
    sd = pm.HalfNormal('sd', sd=.1)
    likelihood = pm.Normal('y',
                           mu=stock2_regression,
                           sd=sd,
                           observed=stock2)

] with model:
    stock1.set_value(data1)
    stock2.set_value(data2)
    trace = pm.sample(2000,tune=1000,cores=4)

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [sd, beta, beta_sigma]
████████████████████ 100.00% [12000/12000 02:31<00:00 Sampling 4 chains, 0 divergences]
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draws total) took 152 seconds.
The acceptance probability does not match the target. It is 0.8926699767273322, but should be close to 0.8. Try to increase the number of tuning steps.
The number of effective samples is smaller than 25% for some parameters.

```

## *Bayesian Modelling using PyMC3*

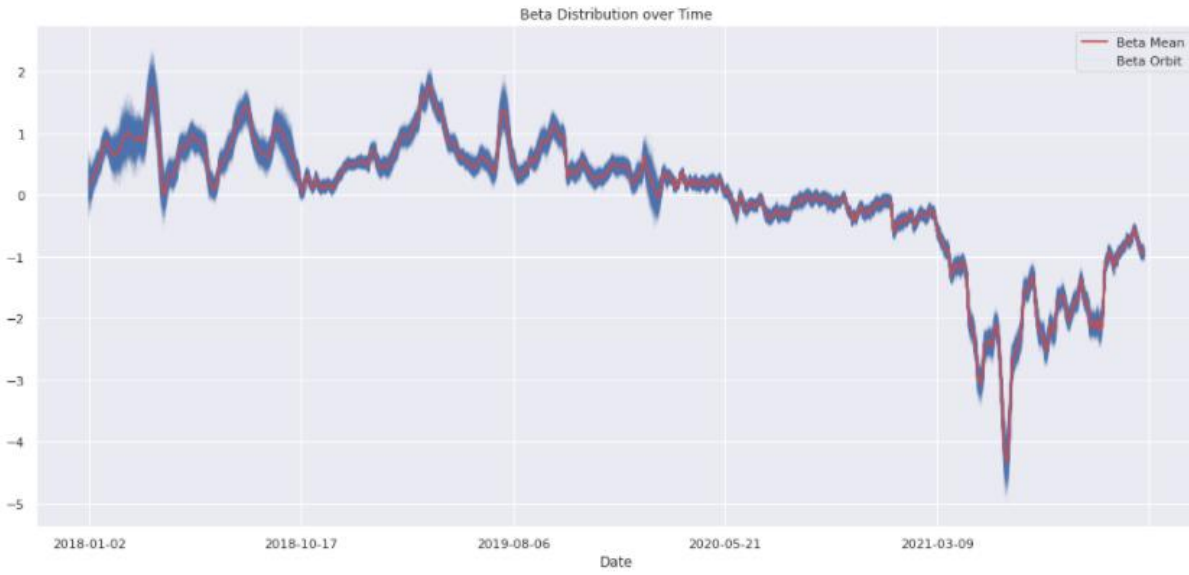
```

rolling_beta = trace[beta].T.mean(axis=1)

plt.figure(figsize = (18,8))
ax = plt.gca()
plt.title("Beta Distribution over Time")
pd.Series(rolling_beta,index=orig_data.index).plot(ax=ax,color='r',zorder=1e6,linewidth=2)
for orbit in trace[beta][:500]:
    pd.Series(orbit,index=orig_data.index).plot(ax=ax,color=sns.color_palette()[0],alpha=0.05)
plt.legend(['Beta Mean','Beta Orbit'])

plt.show()

```



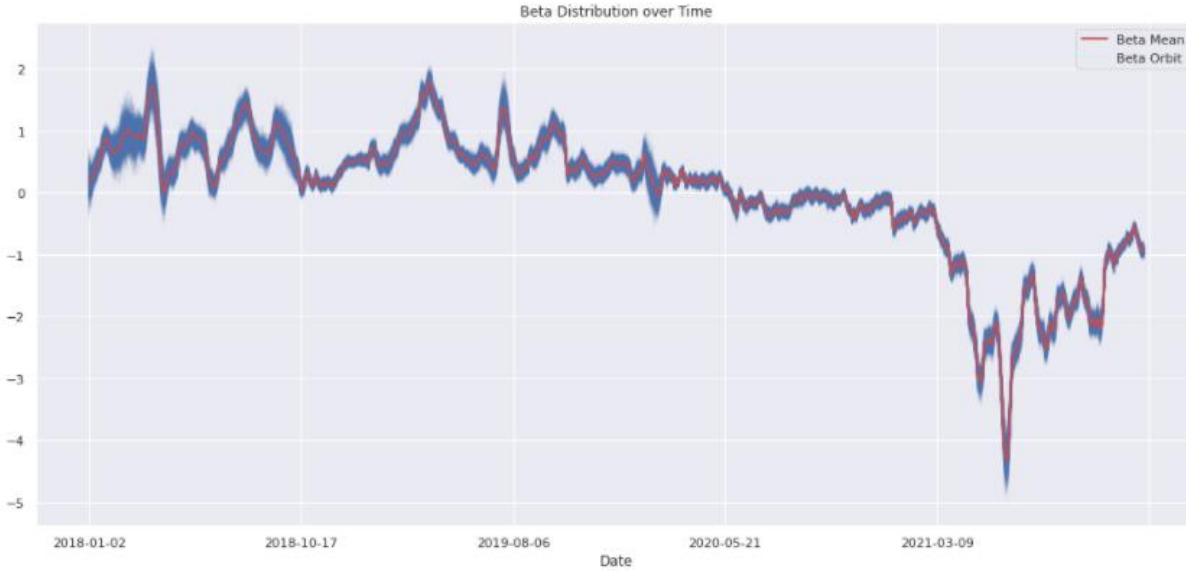
*Beta Distribution  $\beta$  (the relation between two stocks) over time*

## Chapter 4 Long Short Trading Strategy

S&P futures is one of the most liquid futures markets in the world. S&P futures and their options are traded in several financial markets, such as the Chicago Mercantile Exchange (CME) and the CME electronic GLOBEX platform. Below are the various actions we can do with a stock or an option.

- **Long:** a long position is to buy or own a underlying entity (e.g., asset, index, or interest rate futures).
- **Short:** a short position is to sell or owe.
- **Put:** a put option gives the owner of the put, the right, but not the obligation, to sell an asset (the underlying) at a specific price (the strike), by a pre-determined date (the expiration or maturity date) to a given party (the seller of the put). Put options are most used in the stock market to protect against the decline of a stock price below a specific price.
- **Call:** a call option gives the buyer of the call, the right, but not the obligation, to buy an agreed quantity of the underlying from the seller (or “writer”) of the call option before a certain time (the expiration date) at a certain strike price.

To implement a long short trading strategy a simple mean-reversion style trading algorithm, which capitalizes on the assumed mean-reverting behavior of a cointegrated portfolio of stocks is implemented. Non-cointegrated stocks are discarded. Trading occurs only when the portfolio moves back toward its mean value. When the algorithm is not trading,  $\beta$  and its other parameters are dynamically updated to adapt to potentially changing cointegration conditions. Once a trade begins, It is needed to trade the two stocks at a fixed rate, and so  $\beta$  becomes locked for the duration of the trade. The algorithm's exact implementation is as follows:



*Beta Distribution over time*

### Algorithm

1. First a "**Signal**" is defined which should mean-revert to zero if  $\beta$  remains relatively stationary.
2. Then a "**Smoothed signal**" is defined which is a 15-day moving average of the "**Signal**".
3. Based on what the algorithm is doing there are three different scenarios:

Not Trading	Long Trade	Short Trade
<ul style="list-style-type: none"> <li>▪ Update <math>\beta</math> so that it does not remain fixed while we aren't trading.</li> <li>▪ If the smoothed signal is above zero and moving</li> </ul>	<ul style="list-style-type: none"> <li>▪ If the smoothed signal goes below its start value, close the trade; we may be diverging from the mean.</li> <li>▪ If the smoothed signal rises through the zero</li> </ul>	<ul style="list-style-type: none"> <li>▪ If the smoothed signal goes above its start value, close the trade; we may be diverging from the mean.</li> <li>▪ If the smoothed signal falls through the zero</li> </ul>

<p>downward, short our portfolio.</p> <ul style="list-style-type: none"> <li>▪ If the smoothed signal is below zero and moving upward, go long on our portfolio.</li> </ul>	<p>line, we've reached the mean. Close the trade.</p>	<p>line, we've reached the mean. Close the trade.</p>
---	---	---

## Chapter 5 Back Testing

Back testing [19] is a general method for seeing how well a strategy or model would have done. Backtesting assesses the viability of a trading strategy by discovering how it would play out using historical data. If backtesting works, traders and analysts may have the confidence to employ it going forward.

The underlying theory is that any strategy that worked well in the past is likely to work well in the future, and conversely, any strategy that performed poorly in the past is likely to perform poorly in the future. When testing an idea on historical data, it is beneficial to reserve a time period of historical data for testing purposes. If it is successful, testing it on alternate time periods or out-of-sample data can help confirm its potential viability.

For a long short algorithm the expectation is that it would perform strongly during a market drop. The Back testing pricing [18] is based upon some given set of leverage. Leverage works as such that it is calculated overnight. Thus, leverage calculated for today is applied to the previous day's closing price. Below diagram depicts the code to perform Backtesting.

## ▼ Backtesting

```
[ ] portfolioWeights = getTradeStrategyPortfolioWeights(rolling_beta,stock1_name, stock2_name,data).fillna(0)
```

```
def backtest(pricingDF,leverageDF,start_cash):

    pricing = pricingDF.values
    leverage = leverageDF.values

    shares = np.zeros_like(pricing)
    cash = np.zeros(pricing.shape[0])
    cash[0] = start_cash
    curr_price = np.zeros(pricing.shape[1])
    curr_price_div = np.zeros(pricing.shape[1])

    for t in range(1,pricing.shape[0]):

        if np.any(leverage[t]!=leverage[t-1]):

            # handle non-existent stock values
            curr_price[:] = pricing[t-1] # you can multiply with this one
            curr_price[np.isnan(curr_price)] = 0
            trading_allowed = (curr_price!=0)
            curr_price_div[:] = curr_price # you can divide with this one
            curr_price_div[~trading_allowed] = 1

            # determine new positions (warning: leverage to non-trading_allowed stocks is just lost)
            portfolio_value = (shares[t-1]*curr_price).sum()+cash[t-1]
            target_shares = trading_allowed * (portfolio_value*leverage[t]) // curr_price_div

            # rebalance
            shares[t] = target_shares
            cash[t] = cash[t-1] - ((shares[t]-shares[t-1])*curr_price).sum()

        else:

            # maintain positions
            shares[t] = shares[t-1]
            cash[t] = cash[t-1]

    returns = (shares*np.nan_to_num(pricing)).sum(axis=1)+cash
    pct_returns = (returns-start_cash)/start_cash
    return (
        pd.DataFrame( shares, index=pricingDF.index, columns=pricingDF.columns ),
        pd.Series( cash, index=pricingDF.index ),
        pd.Series( pct_returns, index=pricingDF.index)
    )
```

*Back testing Code*

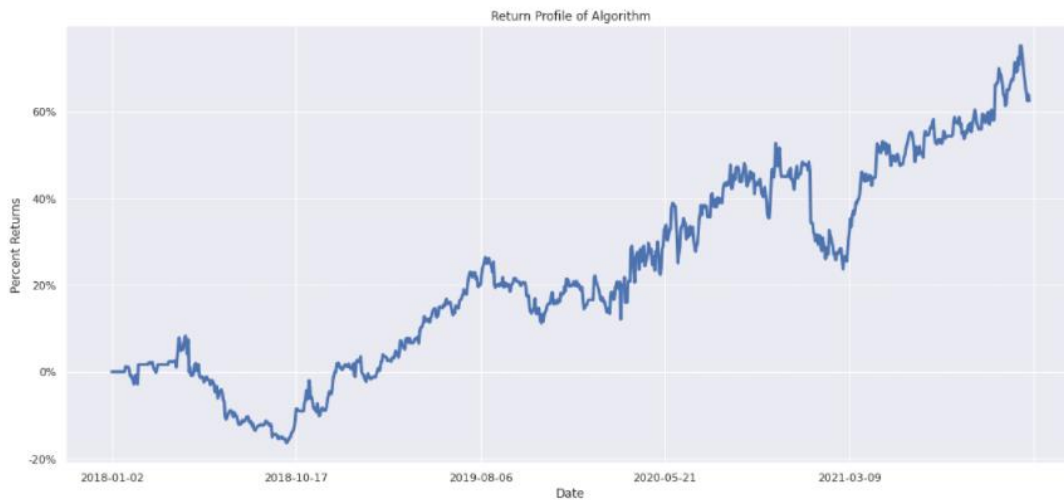


```

shares, cash, returns = backtest( orig_data, portfolioWeights, 1e6 )

plt.figure(figsize = (18,8))
ax = plt.gca()
plt.title("Return Profile of Algorithm")
plt.ylabel("Percent Returns")
returns.plot(ax=ax,linewidth=3)
vals = ax.get_yticks()
ax.set_yticklabels(['{:,.0%}'.format(x) for x in vals])
plt.show()

```



### *Return Profile of the Algorithm*

As expected, performance through market drops is strong. The returns are slightly outsized due to the fact the portfolio only has two stocks.

## **Chapter 6 Conclusion**

### **6.1 Accomplishments and lessons learned**

In this project an effective trading strategy based on pair trading using Bayesian probabilistic programming was designed. The proposed implementation using Bayesian methodology has an upper hand over previously implemented Frequentist methodology[19]. Better and openly available Financial Datasets provided by RapidAPI and IEXCloud helped identify correct stock pairs and helped develop a more robust model. Python framework PyMC3 helped develop an excellent Bayesian Model that could effectively give the right inputs for the long-short trading algorithm. This combined with the highly computational unit TPU allowed for better performance and fine tuning the parameters.

### **6.2 Future work**

I feel the need to research more where the algorithm and approach could be improved, including expanding the portfolio, creating criteria for when  $\beta$  is suitable to trade over, back testing over more periods, using a Bayesian model with fewer simplifying assumptions, and investigating potential nonlinear relationships between stocks. To make the approach more robust more rigorous back testing would be required possibly over stocks in the Russel 3000 index and not just the S&P500 and Cointegrated Groupings could also be extended to more than 2 stocks. Additional functionality to scale up the approach with newer python frameworks needs to be explored. Automated probabilistic programming for a changing and uncertain world needs to be researched to make this approach even more user friendly.

## References

- [1] Wikipedia. Probabilistic Programming Wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/Probabilistic\\_programming](https://en.wikipedia.org/wiki/Probabilistic_programming)
- [2] Bayesian Statistics: A Beginner's Guide <https://www.quantstart.com/articles/Bayesian-Statistics-A-Beginners-Guide/> abstract
- [3] Investopedia. Algorithmic Trading <https://www.investopedia.com/terms/a/algorithmictrading.asp>
- [4] Investopedia. Pair Trading <https://www.investopedia.com/terms/p/pairtrade.asp>
- [5] An introduction to cointegration for pairs trading <https://hudsonthames.org/an-introduction-to-cointegration/>
- [6] Creating and implementing a pairs trading strategy from scratch <https://medium.datadriveninvestor.com/creating-and-implementing-a-pairs-trading-strategy-from-scratch-658267bab249>
- [7] Pairs Trading: Performance of a Relative Value Arbitrage Rule Evan Gatev Assistant Professor Boston College William N. Goetzmann Edwin J. Beinecke Professor of Finance and Management Studies Yale University K. Geert Rouwenhorst Professor Yale University First draft: June 1998 This version: February 2006 [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=141615](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=141615)
- [8] Are Pairs Trading Profits Robust to Trading Costs? Binh Do Department of Accounting and Finance, Monash University Robert Faff UQ Business School, University of Queensland [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=1707125](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1707125)
- [9] The Profitability of Pairs Trading Strategies: Distance, Cointegration, and Copula Methods Rad, Hossein, Low, Rand Kwong Yew and Faff, Robert W., The Profitability of Pairs Trading Strategies: Distance, Cointegration, and Copula Methods, Quantitative Finance, DOI: [org/10.1080/14697688.2016.1164337](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2614233) [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=2614233](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2614233)

- [10] DP-LSTM: Differential Privacy-inspired LSTM for Stock Prediction Using Financial News  
Xinyi Li, Yinchuan Li, Hongyang Yang, Liuqing Yang, Xiao-Yang Liu  
<https://arxiv.org/abs/1912.10806>
- [11] A comprehensive survey on deep neural networks for stock market: The need, challenges, and future directions Ankit Thakkar Kinjal Chaudhari  
<https://www.sciencedirect.com/science/article/abs/pii/S0957417421002414>
- [12] Return and Risk of Pairs Trading using a Simulation-based Bayesian Procedure for Predicting Stable Ratios of Stock Prices Lukasz Gatarek<sup>1</sup> Lennart F. Hoogerheide<sup>2</sup> Herman K. van Dijk, <https://papers.tinbergen.nl/14039.pdf>
- [13] Stock Trading Using PE ratio: A Dynamic Bayesian Network Modeling on Behavioral Finance and Fundamental Investment Haizhen Wang, Ratthachat Chatpatanasiri, Pairote Sattayatham <https://arxiv.org/abs/1706.02985>
- [14] Getting started with PyMC3 Authors: John Salvatier, Thomas V. Wiecki, Christopher Fonnesbeck [https://docs.pymc.io/en/v3/pymc-examples/examples/getting\\_started.html](https://docs.pymc.io/en/v3/pymc-examples/examples/getting_started.html)
- [15] Google Colab Welcome To Colaboratory  
[https://colab.research.google.com/github/barbodkh/python-programing/blob/master/Welcome\\_To\\_Colaboratory.ipynb](https://colab.research.google.com/github/barbodkh/python-programing/blob/master/Welcome_To_Colaboratory.ipynb)
- [16] RapiAPI Financial Supply chain API <https://rapidapi.com/alphavantage/api/alpha-vantage/>
- [17] IEXCloud Stock APIs <https://iexcloud.io/docs/api/#stocks-equities>
- [18] Bayesian Pairs Trading using Corporate Supply Chain Data <https://medium.com/analytics-vidhya/machine-learning-for-stock-trading-unsupervised-learning-techniques-2be85e553361>
- [19] Investopedia. Backtesting  
<https://www.investopedia.com/terms/b/backtesting.asp#:~:text=Backtesting%20is%20the%20ge,neral%20method,to%20employ%20it%20going%20forward.>

## Appendix

### *Bayesian Approach to Pairs Trading.pynb*

#### **Imports**

```
import json
import requests
import pymc3 as pm
import numpy as np
import pandas as pd
import theano as th
import seaborn as sns
import sklearn.decomposition
import matplotlib.pyplot as plt
%matplotlib inline
sns.set()

import warnings
warnings.filterwarnings('ignore')
```

#### **Stock Data for Pair Selection**

```
url = "https://corporate-supply-
chain.p.rapidapi.com/api/v1/resources/supplychain"

querystring = {"ticker":"IBM"}

headers = {
    'x-rapidapi-host': "corporate-supply-chain.p.rapidapi.com",
    'x-rapidapi-key': "e159bd1a20mshacec51099c55d67p16ae77jsndcd63e339daa"
}

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text)
```

```
[{"Ticker":"CMPGF","Full Name":"Compass Group PLC","Relationship":"Suppliers"},{"Ticker":"HTHIF","Full Name":"Hitachi, Ltd.","Relationship":"Suppliers"},{
```

```
# Create DataFrame
df = pd.DataFrame.from_dict(response.json())
df
```

	Ticker	Full Name	Relationship
0	CMPGF	Compass Group PLC	Suppliers
1	HTHIF	Hitachi, Ltd.	Suppliers
2	FJTSF	Fujitsu Limited	Suppliers
3	CAJFF	Canon Inc.	Suppliers
4	SKSBF	Skanska AB (publ)	Suppliers
5	RICOF	Ricoh Company, Ltd.	Suppliers
6	SDXOF	Sodexo S.A.	Suppliers
7	SSNLF	Samsung Electronics Co., Ltd.	Suppliers
8	6701.T	NEC Corporation	Suppliers
9	WPPGF	WPP plc	Suppliers

## **Market Data**

// 5 years worth of daily close market data

```
IEX_API_Key = 'pk_76f28d2269784cc3940e69659f86c033'
```

```
tickers = [ 'IBM', 'ABC' ]
```

```
#Create an empty string called `ticker_string` that we'll add tickers and  
commas to
```

```
ticker_string = ''
```

```
#Loop through every element of `tickers` and add them and a comma to ticke  
r_string
```

```
for ticker in tickers:  
    ticker_string += ticker  
    ticker_string += ','
```

```
#Drop the last comma from `ticker_string`
```

```
ticker_string = ticker_string[:-1]
```

```
#Endpoint and years strings
```

```
endpoints = 'chart'
```

```
years = '5'
```

```

HTTP_request = f'https://cloud.iexapis.com/stable/stock/market/batch?symbols={ticker_string}&types={endpoints}&range={years}y&cache=true&token={IEX_API_Key}'

stock_data = pd.read_json(HTTP_request)

series_list = []

#Loop through each tickers and parse a pandas Series of their closing prices over the last 5 years
for ticker in tickers:
    series_list.append(pd.DataFrame(stock_data[ticker]['chart'])['close'])

#Add in a column of dates
series_list.append(pd.DataFrame(stock_data['IBM']['chart'])['date'])

#Copy the 'tickers' list from earlier in the script and add a new element called 'Date'.
#These elements will be the column names of our pandas DataFrame later on.
column_names = tickers.copy()
column_names.append('Date')

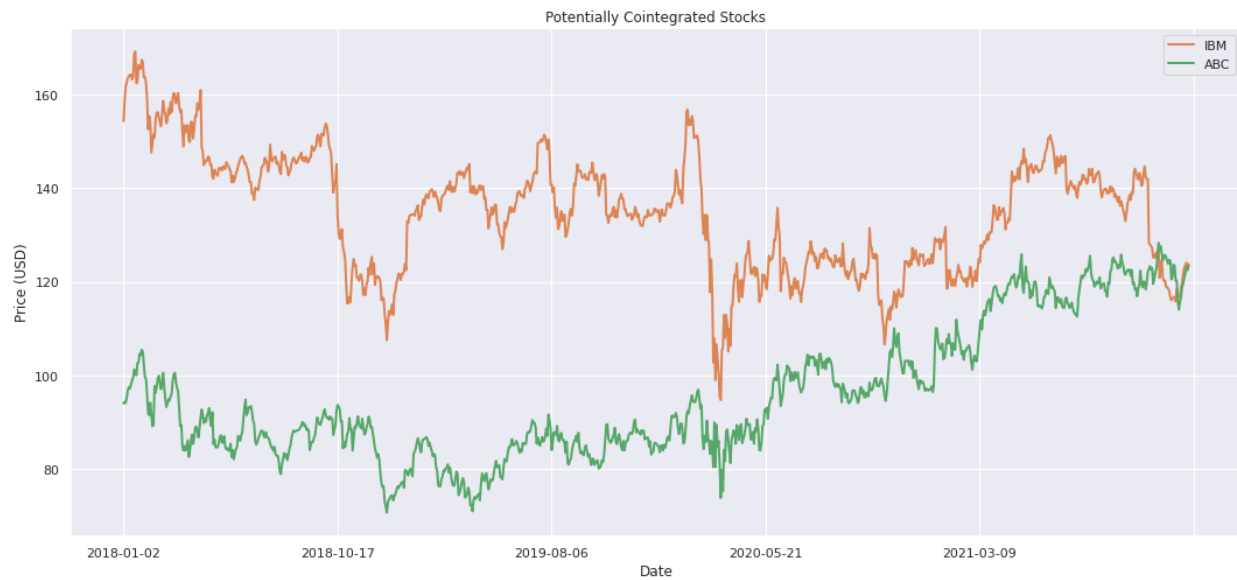
#Concatenate pandas Series together into a single DataFrame
stock_data = pd.concat(series_list, axis=1)

#Name columns of the DataFrame and set the 'Date' column as the index
stock_data.columns = column_names
stock_data.set_index('Date', inplace = True)

stock1_name, stock2_name = 'IBM', 'ABC'
orig_data = stock_data.loc['2018-01-01':,]
data = orig_data.diff().cumsum()
data1 = data[stock1_name].ffill().fillna(0).values
data2 = data[stock2_name].ffill().fillna(0).values

plt.figure(figsize = (18,8))
ax = plt.gca()
plt.title("Potentially Cointegrated Stocks")
orig_data[stock1_name].plot(ax=ax,color=sns.color_palette()[1],linewidth=2)
orig_data[stock2_name].plot(ax=ax,color=sns.color_palette()[2],linewidth=2)
plt.ylabel("Price (USD)")
plt.legend()
plt.show()

```



## **Bayesian Modelling**

```
with pm.Model() as model:

    # inject external stock data
    stock1 = th.shared(data1)
    stock2 = th.shared(data2)

    # define our cointegration variables
    beta_sigma = pm.Exponential('beta_sigma', 50.)
    beta = pm.GaussianRandomWalk('beta', sd=beta_sigma,
                                  shape=data1.shape[0])

    # with our assumptions, cointegration can be reframed as a regression
    # problem
    stock2_regression = beta * stock1

    # Assume prices are Normally distributed, the mean comes from the regression.
    sd = pm.HalfNormal('sd', sd=.1)
    likelihood = pm.Normal('y',
                            mu=stock2_regression,
                            sd=sd,
                            observed=stock2)

with model:
    stock1.set_value(data1)
    stock2.set_value(data2)
```



```

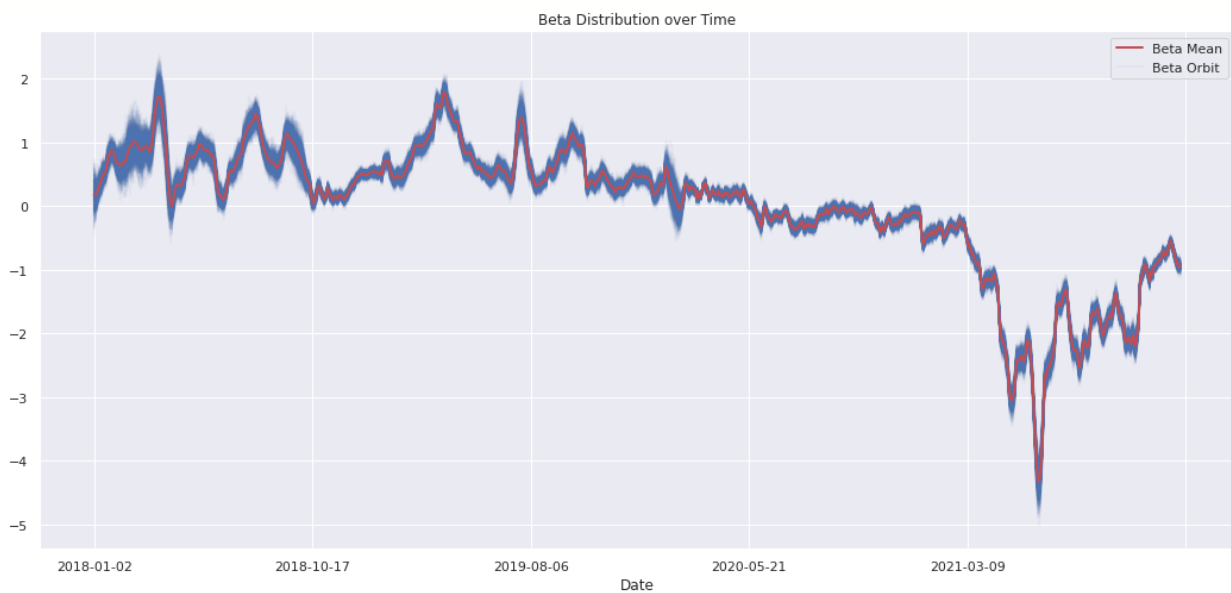
trace = pm.sample(2000,tune=1000,cores=4)

rolling_beta = trace[beta].T.mean(axis=1)

plt.figure(figsize = (18,8))
ax = plt.gca()
plt.title("Beta Distribution over Time")
pd.Series(rolling_beta,index=orig_data.index).plot(ax=ax,color='r',zorder=
1e6,linewidth=2)
for orbit in trace[beta][:500]:
    pd.Series(orbit,index=orig_data.index).plot(ax=ax,color=sns.color_pale
tte()[0],alpha=0.05)
plt.legend(['Beta Mean','Beta Orbit'])
plt.savefig("beta distrib.png")

plt.show()

```



## Trading Strategy

```

def getTradeStrategyPortfolioWeights(rolling_beta, stock_name1, stock_name2,
data, smoothing_window=15):

    data1 = data[stock_name1].ffill().fillna(0).values
    data2 = data[stock_name2].ffill().fillna(0).values

    # initial signal rebalance
    fixed_beta = rolling_beta[smoothing_window]
    signal = fixed_beta*data1 - data2
    smoothed_signal = pd.Series(signal).rolling(smoothing_window).mean()
    d_smoothed_signal = smoothed_signal.diff()
    trading = "not"
    trading_start = 0

    leverage = 0*data.copy()
    for i in range(smoothing_window, data1.shape[0]):
        leverage.iloc[i,:] = leverage.iloc[i-1,:]

        if trading=="not":

            # dynamically rebalance signal when not trading
            fixed_beta = rolling_beta[i]
            signal = fixed_beta*data1 - data2
            smoothed_signal = pd.Series(signal).rolling(smoothing_window).
mean()
            d_smoothed_signal = smoothed_signal.diff()

            if smoothed_signal[i]>0 and d_smoothed_signal[i]<0:

                leverage.iloc[i,0] = -fixed_beta / (abs(fixed_beta)+1)
                leverage.iloc[i,1] = 1 / (abs(fixed_beta)+1)

                trading = "short"
                trading_start = smoothed_signal[i]

            elif smoothed_signal[i]<0 and d_smoothed_signal[i]>0:

                fixed_beta = rolling_beta[i]
                leverage.iloc[i,0] = fixed_beta / (abs(fixed_beta)+1)
                leverage.iloc[i,1] = -1 / (abs(fixed_beta)+1)

                trading = "long"
                trading_start = smoothed_signal[i]

            else:

```

```

        leverage.iloc[i,0] = 0
        leverage.iloc[i,1] = 0

    elif trading=="long":

        # a failed trade
        if smoothed_signal[i] < trading_start:
            leverage.iloc[i,0] = 0
            leverage.iloc[i,1] = 0
            trading = "not"

        # a successful trade
        if smoothed_signal[i]>0:
            leverage.iloc[i,0] = 0
            leverage.iloc[i,1] = 0
            trading = "not"

    elif trading=="short":

        # a failed trade
        if smoothed_signal[i] > trading_start:
            leverage.iloc[i,0] = 0
            leverage.iloc[i,1] = 0
            trading = "not"

        # a successful trade
        if smoothed_signal[i]<0:
            leverage.iloc[i,0] = 0
            leverage.iloc[i,1] = 0
            trading = "not"

    return leverage

```

## **Backtesting**

```

portfolioWeights = getTradeStrategyPortfolioWeights(rolling_beta,stock1_name, stock2_name,data).fillna(0)

```

```

def backtest(pricingDF, leverageDF, start_cash):

    pricing = pricingDF.values
    leverage = leverageDF.values

    shares = np.zeros_like(pricing)
    cash = np.zeros(pricing.shape[0])
    cash[0] = start_cash
    curr_price = np.zeros(pricing.shape[1])
    curr_price_div = np.zeros(pricing.shape[1])

    for t in range(1, pricing.shape[0]):

        if np.any(leverage[t] != leverage[t-1]):

            # handle non-existent stock values
            curr_price[:] = pricing[t-
1] # you can multiply with this one
            curr_price[np.isnan(curr_price)] = 0
            trading_allowed = (curr_price != 0)
            curr_price_div[:] = curr_price # you can divide with this o
ne
            curr_price_div[~trading_allowed] = 1

            # determine new positions
            portfolio_value = (shares[t-1]*curr_price).sum()+cash[t-1]
            target_shares = trading_allowed * (portfolio_value*leverage[t]
) // curr_price_div

            # rebalance
            shares[t] = target_shares
            cash[t] = cash[t-1] - ((shares[t]-shares[t-
1])*curr_price).sum()

        else:

            # maintain positions
            shares[t] = shares[t-1]
            cash[t] = cash[t-1]

    returns = (shares*np.nan_to_num(pricing)).sum(axis=1)+cash
    pct_returns = (returns-start_cash)/start_cash
    return (
        pd.DataFrame( shares, index=pricingDF.index, columns=pricingDF.col
umns ),

```

```

        pd.Series( cash, index=pricingDF.index ),
        pd.Series( pct_returns, index=pricingDF.index)
    )

import matplotlib.pyplot as plt
%matplotlib inline
shares, cash, returns = backtest( orig_data, portfolioWeights, 1e6 )

plt.figure(figsize = (18,8))
ax = plt.gca()
plt.title("Return Profile of Algorithm")
plt.ylabel("Percent Returns")
returns.plot(ax=ax,linewidth=3)
vals = ax.get_yticks()
ax.set_yticklabels(['{:,.0%}'.format(x) for x in vals])
plt.show()

```

