DATA 228: Big Data Tech and App

Using the Kalman Filter-Based Pairs Trading Algorithm to Forecast the Cryptocurrency Market

Stephen Hanna

Chelton Kuang

Arul Palaniappan

**Acknowledgement**

This project would not have happened without the guidance and supervision of Professor Ming-Hwa Wang, as he directed us towards useful links and research papers. We would also like to thank the authors of the research papers we perused through to enhance and further understand our research topic. Finally, this project would not have been possible without our team connection and support.

**Table of Contents**

1. Acknowledgement 2

2. Table of Contents 3

3. List of Tables and Figures 6

4. Abstract 7

5. Introduction 8

a. Objective

b. What is the problem?

c. Why is this project related to this class?

d. Why do we think our approach is better?

e. Statement of the problem

f. Area or scope of investigation

6. Theoretical Bases and Literature Review 12

a. Definition of the problem

b. Related research

c. Advantage / disadvantage of those research

d. Our solution to solve this problem

e. How our solution differs from others

f. Why our solution is better

7. Hypothesis (or goals) 20

8. Methodology 20

a. How to generate / collect input data

b. How to solve the problem

i. Algorithm design

ii. Language used

iii. Tool used

c. How to generate output

d. How to test against hypotheses

e. How to proof correctness

9. Implementation 29

a. Code

b. Design document and flowchart

10. Data analysis and discussion 38

a. Output generation and analysis

b. Compare output against hypothesis

c. Discussion

11. Conclusions and recommendations 41

a. Summary and conclusions

b. Recommendations for future studies

12. Bibliography 43

13. Appendices 44

**List of Tables and Figures**

|  |  |  |
| --- | --- | --- |
| **Figure Number** | **Figure Name** | **Page Number** |
| Figure 1 | A Very Simplified View of the Kalman Filter | 10 |
| Figure 2 | Two correlated stocks that are diverging from each other | 12 |
| Figure 3 | Where to buy and sell stocks when comparing the prices of two cointegrated stocks | 12 |
| Figure 4 | Comparison of prices of bitcoin on two different exchanges | 13 |
| Figure 5 | Results of the tests performed by Fil & Kristoufek | 13 |
| Figure 6 | Statistics of the results of the tests performed by Fil & Kristoufek | 14 |
| Figure 7 | Dataset for the relationship between two cointegrated stocks | 15 |
| Figure 8 | Kalman Filtering to find best linear fit on the dataset | 16 |

**Abstract**

In our project proposal, we will use the Kalman filter algorithm alongside pair trading to increase the overall accuracy of forecasting the cryptocurrency market. It discusses the volatility of the cryptocurrency stocks and why forecasting is so important for investors. Currently, pair trading is used in all types of stocks, including cryptocurrency, but lacks overall performance in the crypto sector. Therefore, we plan on using the Kalman filter algorithm in conjunction as it can learn and update as the pairs are fed through. The Kalman outputs and non-Kalman outputs will be compared with the same data to prove which is better at forecasting. We have planned to implement the methodologies in Python and historical datasets gathered in the form of CSV files.

**Introduction**

**Objective**

The core objective is to develop an accurate forecasting model, which would help in forecasting the cryptocurrency stock prices. To achieve this objective, we will use the Kalman Filter-Based Pairs Trading algorithm through Python against normal pairs trading for comparison.

**Technical Objective**

The technical objectives will be implemented in Python. The system should be able to forecast the selected cryptocurrency stock prices based on the historical data taken from the Bitfinex API. The system should also work with both the Kalman algorithm and the pairs trading technique in Python.

**Experimental Objective**

This project will implement two prediction approaches, with the first being the Kalman filtering paired with the pairs trading technique and the other being only the pairs trading technique. We will compare each method’s forecasting accuracy by feeding the two approaches the same datasets. Once the results are clear, we can either approve or disapprove our expectations of the Kalman algorithm creating a more accurate forecast for cryptocurrency stock prices.

**What is The Problem?**

In the stock market, forecasting is always an important issue that is constantly being refined. Within the cryptocurrency market, there is no governing body that controls it, but rather, many algorithms control how much of their specific cryptocurrency is on the market and at what price. This is the main reason why crypto prices are so volatile, as there is no gold to base off of and there are no governing bodies that can support the system if things go badly wrong. In addition, cryptocurrency stocks have significant speculative value since much of their value comes from how they will be used in the future. As such, changes in traders’ sentiments based off of a variety of external factors, such as political atmosphere and current economic status, causes rapid changes in the crypto market. These changes become more pronounced when compared to regular stock markets as cryptocurrencies have smaller market sizes. Therefore, having accurate forecasting at any given moment can make or break a crypto trader.

Here are some techniques used for forecasting stock prices including: BART (time-series), SMA (average), LSTM (deep learning), SLR & MLR (statistical), and the Kalman filter (machine learning). In this project, we will use the Kalman filter alongside the pairs trading technique as pairs trading is extremely popular in crypto and quite suitable to the extremely volatile environment of the cryptocurrency stock market.

The problem with pairs trading is that while it is good in the long run, it falls flat when it comes to day-to-day profits, which is what a lot of day traders look towards and could potentially produce profitability if gotten right. In addition, pairs trading does not take into account external variables such as supply and demand, which has much larger repercussions when compared to regular stock markets. To improve accuracy for pairs trading, we will look towards using the Kalman filtering algorithm.

**How Our Project Relates to The Class**

Our project involves a machine learning algorithm in the Kalman filtering algorithm and a data-driven technique in pairs trading through Python. Both are designed to take big amounts of cryptocurrency price data, with the algorithm continuously learning and updating itself alongside the pairs trading technique.

**Why Other Approaches Are Not Good**

Other approaches use fuzzy logic, which will not produce accurate results most of the time. In addition, the fuzzy logic model can only work on simple problems.

**Why Our Approach is Better**

The Kalman filtering algorithm that we plan on using does not use fuzzy logic, since it is mainly time-based. In addition, using fuzzy logic will only slow down our processes and the algorithm will only get better the more data it is fed.

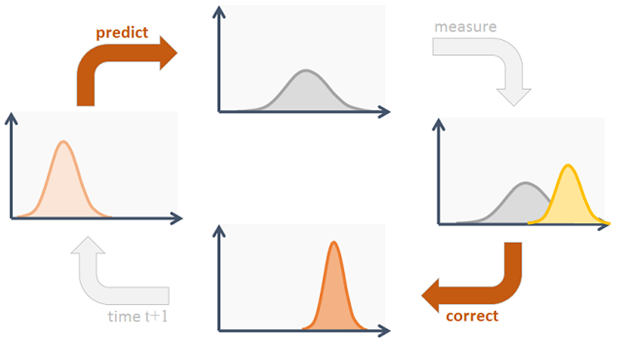


Figure 1: A Very Simplified View of the Kalman Filter

We believe our approach is better in that in addition to the pairs trading technique, which is a tried-and-true approach, the Kalman filter will filter out unrelated noise while flipping through each paired data and learning from it continuously. Therefore, this approach is better as the algorithm can make dynamic predictions, which means quick, up-to-date cryptocurrency price forecasting.

**Statement of the Problem**

Cryptocurrency is known to be volatile and, therefore, difficult to forecast. The current use of the pairs trading technique in cryptocurrency forecasting results in zero day-to-day returns and daily results are typically more volatile when compared to long-term results. Improving day-to-day returns is a must in order to improve profitability.

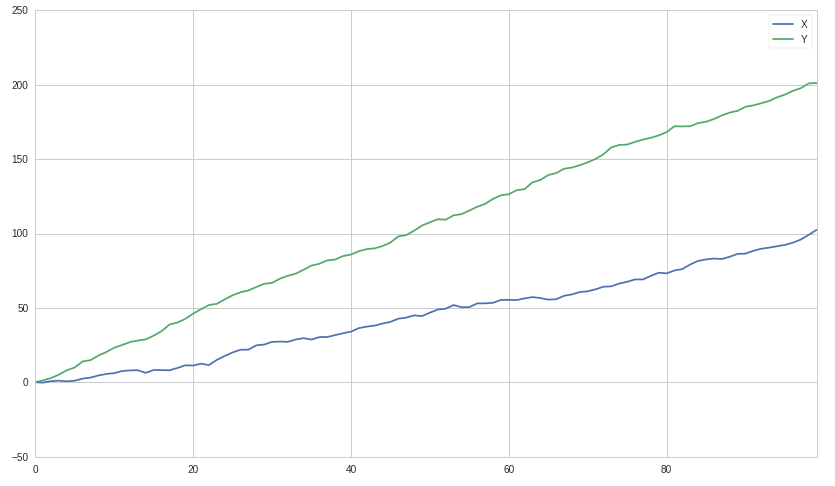
**Area or Scope of Investigation**

Our data will be historical cryptocurrency stock price data, which will be trained through the Kalman filtering algorithm, and estimates will be created using the paired trading technique.

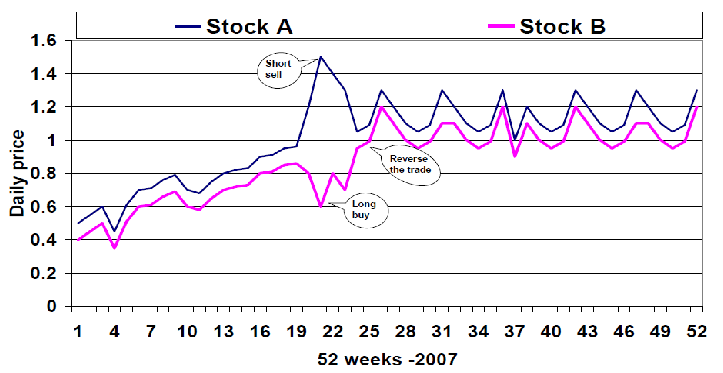
**Theoretical bases and literature review**

**Theoretical Background of the Problem**

Pairs trading is a trading strategy that enables traders to profit from any kind of market condition, such as uptrend, downtrend, or sideways movement. This strategy is a mean-reversion contrarian trading strategy that tries to find a long-run stable state between prices of two securities. [Fil & Kristoufek] Mean reversion is a strategy that involves betting that prices will revert back towards the mean. The main focus of pairs trading is comparing the means of the two prices and making sure that the difference in means does not substantially increase at any point. This is achieved by shorting the overperforming and longing the underperforming security in the hopes that its price reverts back to the mean, leading to a profit of the price spread in the long run. The two securities must be cointegrated because while their spread mean may occasionally diverge over time, it will eventually be pulled back together. If two securities are correlated but not cointegrated, then they move in the same general direction. There is a chance these securities diverge over time, causing the spread to diverge.

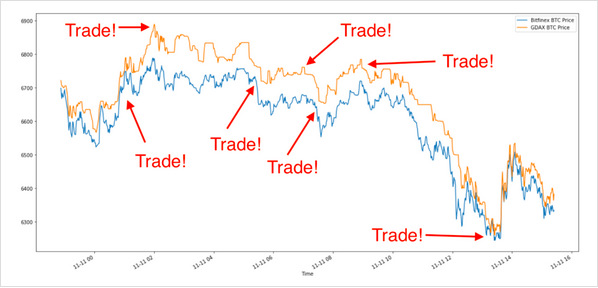


**Figure 2:** Two correlated stocks that are diverging from each other



**Figure 3:** Where to buy and sell stocks when comparing the prices of two cointegrated stocks

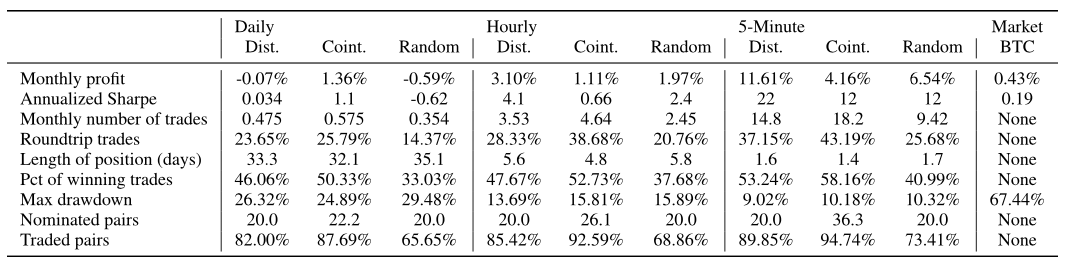
Cryptocurrencies are commonly perceived as inefficient, illiquid and suffering from frequent bear markets. [Fil & Kristoufek]. Pairs trading works well with forecasting cryptocurrency prices because of the unpredictable nature of the cryptocurrency market.



**Figure 4:** Comparison of prices of bitcoin on two different exchanges and shows where best to trade using the pairs-trading strategy

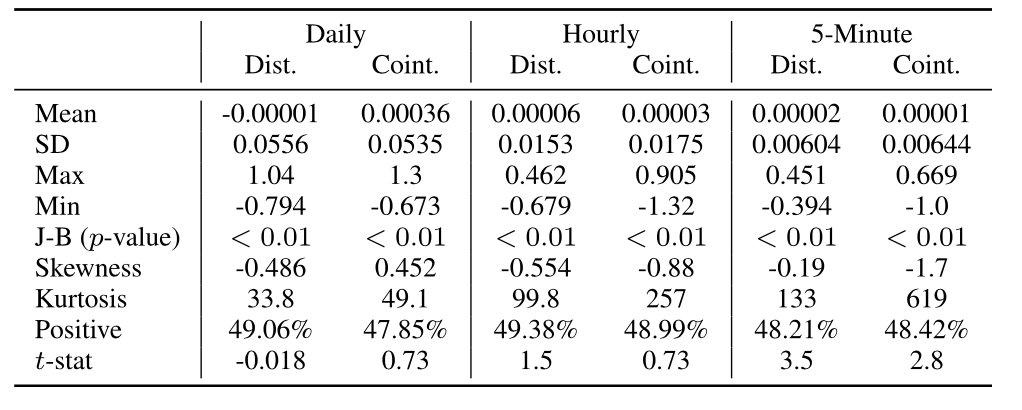
**Literature Review**

Fil & Kristoufek analyze the pairs trading algorithm to figure out if the profitability of the algorithm is better than using alternative methods. They applied the distance and cointegration methods to a basket of 26 liquid cryptocurrencies traded and used backtesting against daily, hourly, and 5-minute data to see the results.



**Figure 5:** Results of the tests performed by Fil & Kristoufek

The results of their tests showed that the distance method fails to perform on daily data, delivering zero returns. The performance of the distance method is better for the higher frequency data, delivering a 3.1% profit for the hourly frequency and an 11.6% profit for the 5-minute data. The cointegration method performs much better in the daily setting, with a 1.36% monthly profit and a Sharpe ratio of 1.1. However, this method doesn’t improve in higher frequencies as the performance worsens for the hourly data to 1.11% profit, but increases to 4.16% profit at the 5-minute data. Overall, the distance method improves profitability as the frequency of trading data increases. But the profits of the cointegration method stays relatively stable no matter the frequency of trading data.

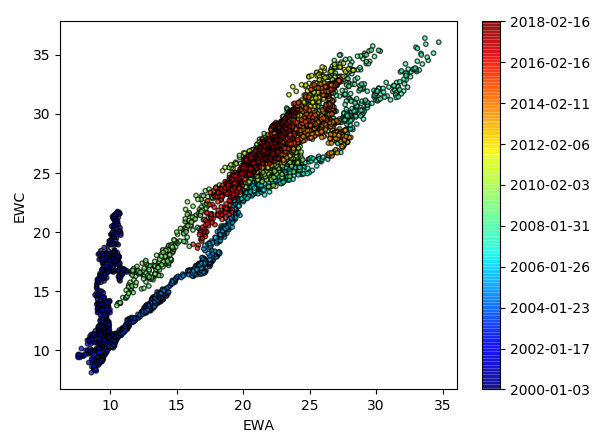


**Figure 6:** Statistics of the results of the tests performed by Fil & Kristoufek

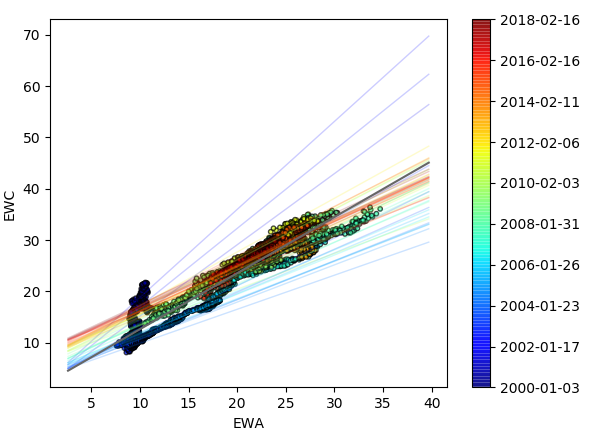
As shown in the table above, Fil & Kristoufek find that the returns found in their results are very volatile. The mean is the same across all frequencies. The range is huge because pairs can lose or gain greater than 100% within a period. The kurtosis is also very large, especially for higher frequency data, which indicates that there are a lot of outliers. They also found that exogenous variables, such as transaction costs and execution windows, are extremely important and their exact determination is vital.

**Our Solution to the Problem**

Our solution to address the problem of improving the overall performance of pair trading is to use the Kalman filtering alongside pair trading. In this scenario, Kalman filter is used to dynamically track the hedging ratio between the two in order to keep the spread between the pair data stationary and continuing mean reversion as the data gets fed into the algorithm. The estimated state from the previous time step and the current measurement are the only information needed to compute the estimate for the current state. This adds more weight to more recent data so that older data does not unnecessarily skew the model.



**Figure 7:** Dataset for the relationship between two cointegrated stocks



**Figure 8:** Kalman Filtering to find best linear fit on the dataset

The above figures show how the Kalman filter algorithm works when trying to find the hedge ratio of the relationship between two stocks. The first figure shows a dataset of the relationship between two stocks. As time goes on, the relationship changes, which means that the hedge ratio also changes. The second filter shows the Kalman filter based linear regression model on the dataset to find the optimal linear fit on the relationship.

**Kalman Filter Algorithm**

The process model defines the transformation of the state from time *k*−1 to time *k* with the following algorithm.The Kalman filter model assumes the true state at time *k* is evolved from the state at (*k* − 1) according to:



* **F***k* is the state transition model (transition matrix) which is applied to the previous state **x***k*−1
* **B***k* is the control-input model (observation matrix) which is applied to the control vector **u***k*
* **w***k* is the process noise, which is assumed to be drawn from a normal distribution *Q:* ***w****k*−1 ∼*𝒩*(0,*Q*)

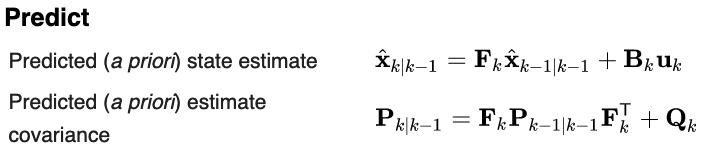
The transition matrix tells us how the system evolves from one state to another and the observation matrix which tells us the next measurement we should expect given the predicted next state.

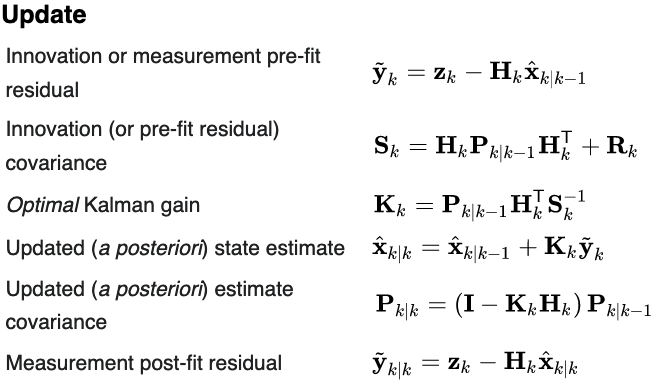
At time *k* an observation (**z***k*) of the true state (**x***k*) is made according to:



* ***z****k* is the measurement vector
* ***H****k* is the measurement matrix, which maps the true state space into the observed space
* ***ν****k* is the measurement noise vector that is assumed to be drawn from a normal distribution *R:* ***ν****k*∼*𝒩*(0,*R*)

The following steps are used to predict and update the states:





In the above equations, the variables with the hat operator (^) are the estimates of those variables*.* The *a priori* estimate (prediction) is the state estimate that is calculated before the state’s measurements are taken and the *a posteriori* estimate (update) is the state estimate that is calculated after the state’s measurements are taken.

**Why our Solution is Better**

Kalman filtering is an algorithm that provides estimates of the true underlying state given the measurements of the variables observed over a period of time in a data stream. It works best when there is a model to predict what the state should be at a certain time. At each time step, the algorithm makes a prediction, takes in a measurement, and updates itself based on the error between prediction and measurement. An execution window is not needed for this algorithm because it only needs the results of the previous iteration to update the states. The algorithm also takes into account data that is noisy. By doing the adjustment by itself, the algorithm generates a predictive learning model which can learn and update itself through the data that is fed to it. This ultimately leads to a more accurate measure of the mean being observed between two cryptocurrencies.

**Hypotheses**

**Single Hypothesis**

Our hypothesis is that using a Kalman Filter Pairs Trade will perform a better Sharpe ratio than a standalone Pairs trading method. This essentially means we’re expecting that our profit margins will be higher with lower implicit risk exposure than those reported in the underlying research.

**Positive Hypothesis**

This approach will result in positive hypothesis testing due to the evidence containing the property of interest if correct.

**Methodology**

**Collecting Input Data**

The method by which we will collect our historical cryptocurrency data is through Bitfinex exchange API. We plan to collect the data from Jan 2018 to Sept 2019 using the below client:

>>> pip install bitfinex-tencars

It provides an easy data retrieval mechanism that will be installed to our Python distribution. Next, it is as simple as creating an instance of the API client which will provide us access to the public endpoints.

>>> import bitfinex

>>> api\_v2 = bitfinex.bitfinex\_v2.api\_v2()

We will then set the parameters and tailor the program to our needs.

*Parameters*:

· *symbol*: currency pair,default: *BTCUSD*

· *interval*: temporal resolution, e.g. *1m* for 1 minute of OHLC data

· *limit*: number of returned data points, default: *1000*

· *start*: start time of interval in milliseconds since 1970

· *end*: end time of interval in milliseconds since 1970

>>> import datetime

>>> import time

>>> # Define query parameters

>>> pair = 'btcusd' # Currency pair of interest

>>> bin\_size = '1m' # This will return minute data

>>> limit = 1000 # We want the maximum of 1000 data points >>> # Define the start date

>>> t\_start = datetime.datetime(2018, 1, 1, 0, 0)

>>> t\_start = time.mktime(t\_start.timetuple()) \* 1000

>>> # Define the end date

>>> t\_stop = datetime.datetime(2019, 9, 2, 0, 0)

>>> t\_stop = time.mktime(t\_stop.timetuple()) \* 1000

Unfortunately, the API only allows for 1000 minutes (data points) to be retrieved in one iteration. Thus, we can create a loop to request the data until all the information we need is captured. Here is where another problem arises, the Bitfinex API only allows for 60 calls per minute, or 1 call per second. Taking this into account we can force the program to pause before running each iteration as so:

>>> def fetch\_data(start, stop, symbol, interval, tick\_limit, step):

>>> # Create api instance

>>> api\_v2 = bitfinex.bitfinex\_v2.api\_v2()

>>> data = []

>>> start = start - step

>>> while start < stop:

>>> start = start + step

>>> end = start + step

>>> res = api\_v2.candles(symbol=symbol, interval=interval, limit=tick\_limit, start=start,

>>> end=end)

>>> data.extend(res)

>>> time.sleep(2)

>>> return data

We can now run the query to retrieve all of the data with the following syntax:

>>> # Set step size

>>> time\_step = 60000000

>>> # Define the start date

>>> t\_start = datetime.datetime(2018, 4, 1, 0, 0)

>>> t\_start = time.mktime(t\_start.timetuple()) \* 1000

>>> # Define the end date

>>> t\_stop = datetime.datetime(2018, 5, 1, 0, 0)

>>> t\_stop = time.mktime(t\_stop.timetuple()) \* 1000

>>> pair\_data = fetch\_data(start=t\_start, stop=t\_stop, symbol=pair,

>>> interval=bin\_size, tick\_limit=limit,

>>> step=time\_step)

Lastly, format the data and remove duplicates:

>>> import pandas as pd

>>>

>>> # Create pandas data frame and clean/format data

>>> names = ['time', 'open', 'close', 'high', 'low', 'volume']

>>> df = pd.DataFrame(pair\_data, columns=names)

>>> df.drop\_duplicates(inplace=True)

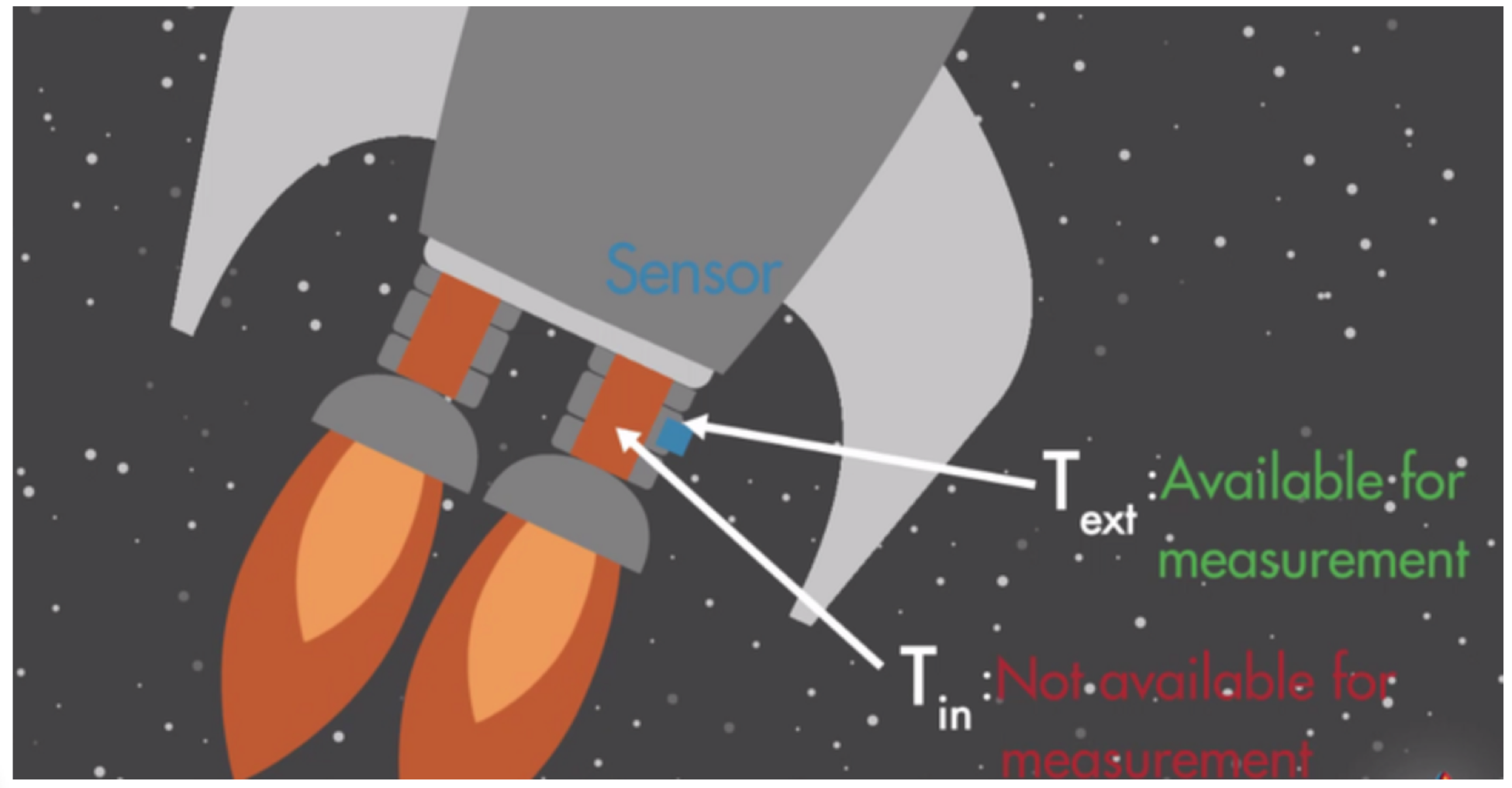
>>> df['time'] = pd.to\_datetime(df['time'], unit='ms')

>>> df.set\_index('time', inplace=True)

>>> df.sort\_index(inplace=True)

**How to solve the problem**

Before we seek to understand the method by which we will attempt to solve the problem in our study, let’s first dive into why this method was initially proposed. To begin, a Kalman filter is an optional estimation algorithm that seeks to estimate a system state when it cannot be measured directly. This concept has very valuable and useful applications in many areas. For example, rockets are propelled by liquid hydrogen into space creating an extremely hot combustion chamber. The problem is that if the chamber gets too hot then it can create mechanical issues within the rocket. Thus, NASA must monitor the temperature of the combustion chamber at all times in order to guarantee its safety. With that being said, they cannot place the sensor inside the chamber, where it will have the highest precision, because the sensor will quickly melt. Instead, they place the sensor directly outside the chamber, where it is cooler, and measure the external temperature.



They then use an intelligent mathematical algorithm, called a Kalman filter, to find the best estimate of the internal temperature using an indirect measurement. In conclusion, we would like to use this same methodology to extract information from what we can’t measure with what we can with the use of a Kalman filter in conjunction with pairs trading.

**Algorithm Design**

In order for this algorithm to work we will need cryptocurrencies that are cointegrated so let’s begin here. We will use the coin() method taken from the statsmodel api as shown:

import statsmodels.api as sm

result = sm.tsa.stattools.coint(stock1, stock2) # get conintegration

Next we will define our Kalman Filter algorithm with the help of the pykalman module. The first function generates a rolling mean among the observed values of the price:

from pykalman import KalmanFilter

def KalmanFilterAverage(x):

# Construct a Kalman filter

kf = KalmanFilter(transition\_matrices = [1],

observation\_matrices = [1],

initial\_state\_mean = 0,

initial\_state\_covariance = 1,

observation\_covariance=1,

transition\_covariance=.01) #create KalmanFilter object

# Use the observed values of the price to get a rolling mean

state\_means, \_ = kf.filter(x.values)

state\_means = pd.Series(state\_means.flatten(), index=x.index)

return state\_means

We will now define a regression function to constantly monitor the state parameters:

def KalmanFilterRegression(x,y):

kf = KalmanFilter(n\_dim\_obs=1, n\_dim\_state=2,

initial\_state\_mean=[0,0],

initial\_state\_covariance=np.ones((2, 2)),

transition\_matrices=np.eye(2),

observation\_matrices=obs\_mat,

observation\_covariance=2,

transition\_covariance=trans\_cov) #create KalmanFilter object

state\_means, state\_covs = kf.filter(y.values)

return state\_means

Then, we will create the engine that constantly monitors our hedging strategy and calculates the weight, or number of units, of each pairwise, cointegrated, cryptocurrency we need to short or long:

def backtest(df,s1, s2):

# run regression (including Kalman Filter) to find hedge ratio and then create spread series

# calculate z-score with window = half life period

meanSpread = df1.spread.rolling(window=halflife).mean()

stdSpread = df1.spread.rolling(window=halflife).std()

df1['zScore'] = (df1.spread-meanSpread)/stdSpread

entryZscore = 2

exitZscore = 0

#set up num units long

df1['long entry'] = ((df1.zScore < - entryZscore) & ( df1.zScore.shift(1) > - entryZscore))

df1['long exit'] = ((df1.zScore > - exitZscore) & (df1.zScore.shift(1) < - exitZscore))

#set up num units short

df1['short entry'] = ((df1.zScore > entryZscore) & ( df1.zScore.shift(1) < entryZscore))

df1['short exit'] = ((df1.zScore < exitZscore) & (df1.zScore.shift(1) > exitZscore))

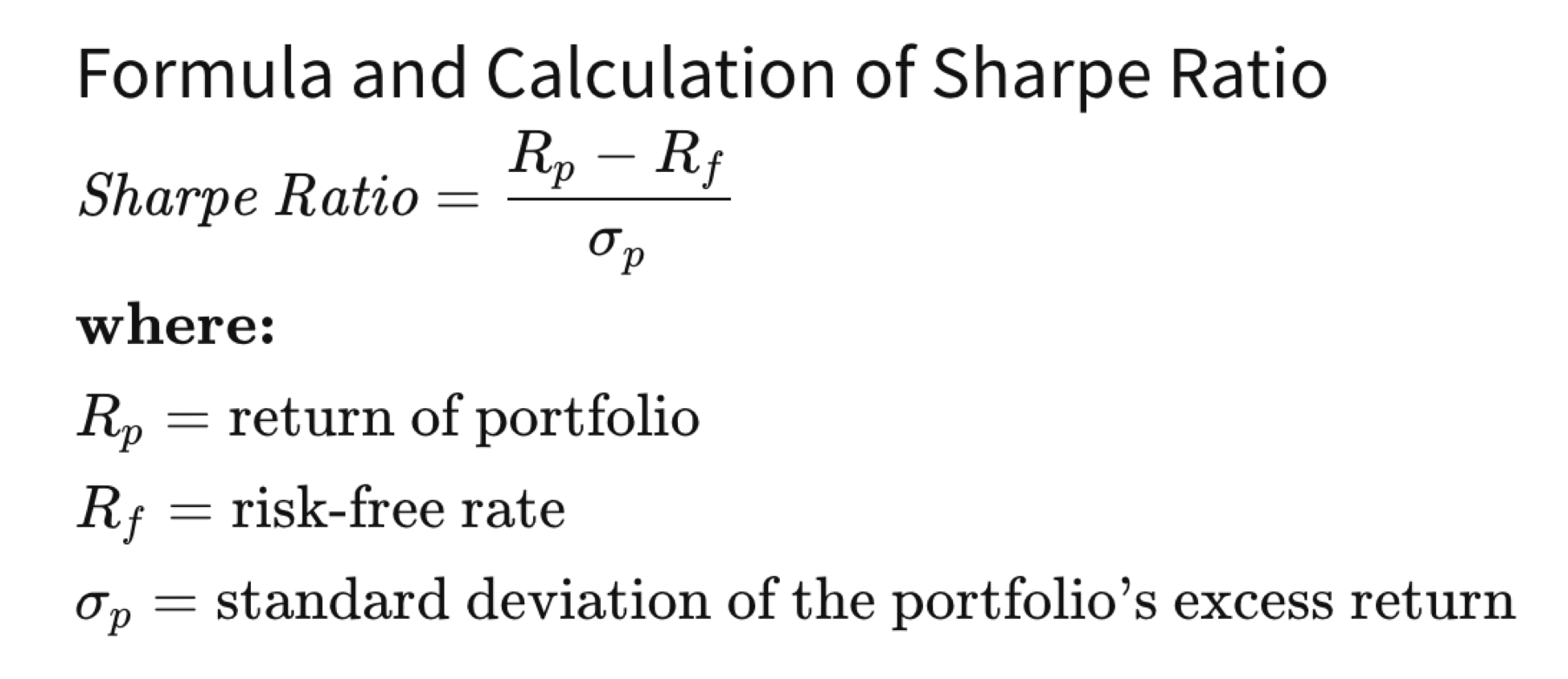
df1['numUnits'] = df1['num units long'] + df1['num units short']

df1['spread pct ch'] = (df1['spread'] - df1['spread'].shift(1)) / ((df1['x'] \* abs(df1['hr'])) + df1['y'])

Finally, we will test how our hedging strategy compared to the research paper using the Sharpe ratio:

sharpe = (final\_res.pct\_change().mean() / final\_res.pct\_change().std()) \* (sqrt(252))

The Sharpe ratio is the preferable measure of success for our results since it calculates intrinsic risk in its assessment. Only comparing returns does not take risk into account and thus provides an inaccurate comparison of results.



*https://www.investopedia.com/terms/s/sharperatio.asp*

**Language Used**

Python

**Tools Used**

· KalmanFilter class in pykalman module

· coint() method in the statsmodels api

· Bitfinex exchange API

### **Implementation**

**Code**

import bitfinex

import datetime

import time

import pandas as pd

from operator import itemgetter

from functools import reduce

import copy

from pykalman import KalmanFilter

from math import sqrt

import statsmodels.api as sm

import findspark

findspark.init()

import pyspark

sc = pyspark.SparkContext(appName="hw")

import pyspark.sql.types as typ

from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()

from pyspark.sql.types import \*

import pyspark.sql.functions as fn

import pyspark.mllib.stat as st

import numpy as np

import pyspark.mllib.linalg as ln

import numpy as np

import statsmodels.api as sm

from pyspark.sql.window import Window

from pyspark.sql.functions import when

# In[2]:

def fetch\_data(start, stop, symbol, interval, tick\_limit, step):

api\_v2 = bitfinex.bitfinex\_v2.api\_v2()

data = []

start = start - step

while start < stop:

start = start + step

end = start + step

res = api\_v2.candles(symbol=symbol, interval=interval,limit=tick\_limit, start=start, end=end)

for x in res:

x[0] = datetime.datetime.utcfromtimestamp(x[0]/1000).strftime('%Y-%m-%d %H:%M:%S')

x[1:6] = [float(x) for x in x[1:6]]

data.extend(res)

time.sleep(2)

rdd = spark.sparkContext.parallelize(data)

return rdd

pairs = ['ETHUSD', 'LTCUSD', 'XMRUSD', 'NEOUSD', 'XRPUSD', 'ZECUSD', 'BTCUSD']

bin\_size = '5m'

limit = 1000

t\_start = datetime.datetime(2019, 9, 1, 0, 0)

t\_start = time.mktime(t\_start.timetuple()) \* 1000

t\_stop = datetime.datetime(2019, 9, 30, 0, 0)

t\_stop = time.mktime(t\_stop.timetuple()) \* 1000

names = ['time', 'open', 'close', 'high', 'low', 'volume']

time\_step = 300000 \* limit

crypto\_prices = [spark.createDataFrame(fetch\_data(start=t\_start, stop=t\_stop, symbol=pair, interval=bin\_size, tick\_limit=limit, step=time\_step), schema = names) for pair in pairs]

crypto\_copy = [x.select("\*") for x in crypto\_prices]

clean\_df\_list = []

def find\_cointegrated\_pairs(dataframe):

n = len(dataframe.columns)

keys = dataframe.columns

pairs = []

for i in range(n):

for j in range(i+1, n):

stock1 = dataframe.select(keys[i]).toPandas()[keys[i]]

stock2 = dataframe.select(keys[j]).toPandas()[keys[j]]

result = sm.tsa.stattools.coint(stock1, stock2)

pvalue = result[1]

pairs.append((keys[i], keys[j], pvalue))

return pairs

# In[3]:

for pair, df in zip(pairs, crypto\_copy):

df = df.sort("time")

df = df.select("time", "close")

df = df.withColumnRenamed("close", pair)

clean\_df\_list.append(df)

combined\_cryptos = reduce(lambda df1,df2: df1.join(df2,on='time'), clean\_df\_list)

combined\_cryptos = combined\_cryptos.sort('time')

combined\_cryptos.show()

# In[4]:

split = int(combined\_cryptos.count() \* .4)

no\_date\_values = combined\_cryptos.drop(fn.col('time'))

pairs = find\_cointegrated\_pairs(no\_date\_values.limit(split))

for pair in pairs:

print("Stock {} and stock {} has a co-integration score of {}".format(pair[0],pair[1],round(pair[2],4)))

# In[6]:

critical\_level = 0.05

for pair in pairs:

if pair[2] < critical\_level:

print("Stock {} and stock {} has a co-integration score of {}".format(pair[0],pair[1],round(pair[2],4)))

# In[7]:

def KalmanFilterAverage(x):

kf = KalmanFilter(transition\_matrices = [1],

observation\_matrices = [1],

initial\_state\_mean = 0,

initial\_state\_covariance = 1,

observation\_covariance=1,

transition\_covariance=.01)

state\_means, \_ = kf.filter(x.values)

state\_means = pd.Series(state\_means.flatten(), index=x.index)

return state\_means

def KalmanFilterRegression(x,y):

delta = 1e-3

trans\_cov = delta / (1 - delta) \* np.eye(2)

obs\_mat = np.expand\_dims(np.vstack([[x], [np.ones(len(x))]]).T, axis=1)

kf = KalmanFilter(n\_dim\_obs=1, n\_dim\_state=2,

initial\_state\_mean=[0,0],

initial\_state\_covariance=np.ones((2, 2)),

transition\_matrices=np.eye(2),

observation\_matrices=obs\_mat,

observation\_covariance=2,

transition\_covariance=trans\_cov)

state\_means, state\_covs = kf.filter(y.values)

return state\_means

# In[8]:

def backtest(df,s1, s2):

x = df.select(pair[0]).toPandas()[pair[0]]

y = df.select(pair[1]).toPandas()[pair[1]]

state\_means = KalmanFilterRegression(KalmanFilterAverage(x),KalmanFilterAverage(y))

temp\_rdd = state\_means.tolist()

temp\_df = sc.parallelize(temp\_rdd).toDF(["remove", "hr"])

df = df.withColumn("id", fn.monotonically\_increasing\_id())

temp\_df = temp\_df.withColumn("id", fn.monotonically\_increasing\_id())

temp\_df = temp\_df.withColumn("id", fn.row\_number().over(Window.orderBy(fn.col("id"))))

df\_joined = df.join(temp\_df, on=['id'], how="inner").drop("id")

df\_joined = df\_joined.withColumn("hr", -1\*fn.col("hr"))

df\_joined = df\_joined.select(["time",s1,s2,"hr"])

df\_joined = df\_joined.withColumn("spread", fn.col(s2) + (fn.col(s1)\*fn.col("hr")))

df\_copy = df\_joined.select("\*")

windowSpec = Window().orderBy('time').rowsBetween(-5,0)

df\_copy = df\_copy.withColumn("meanSpread",fn.mean("spread").over(windowSpec))

df\_copy = df\_copy.withColumn("stdSpread",fn.stddev("spread").over(windowSpec))

df\_copy = df\_copy.withColumn('zScore', (fn.col("spread") - fn.col("meanSpread"))/fn.col("stdSpread"))

entryZscore = 1

exitZscore = 0

w = Window().partitionBy().orderBy(fn.col("time"))

df\_copy = df\_copy.withColumn("shiftedZ", fn.lag(df\_copy.zScore).over(w))

df\_copy = df\_copy.withColumn('long entry', (df\_copy.zScore < - entryZscore) & (df\_copy.shiftedZ > - entryZscore))

df\_copy = df\_copy.withColumn('long exit', (df\_copy.zScore > - exitZscore) & (df\_copy.shiftedZ < - exitZscore))

long\_entry\_exit = when(fn.col("long exit") == "false", fn.lit(0)).when(fn.col("long entry") == "true", fn.lit(1)).otherwise(fn.lit(0))

df\_copy = df\_copy.withColumn('num units long', long\_entry\_exit)

df\_copy = df\_copy.withColumn('short entry', (df\_copy.zScore > entryZscore) & (df\_copy.shiftedZ < entryZscore))

df\_copy = df\_copy.withColumn('short exit', (df\_copy.zScore < exitZscore) & (df\_copy.shiftedZ > exitZscore))

short\_entry\_exit = when(fn.col("short exit") == "false", fn.lit(0)).when(fn.col("short entry") == "true", fn.lit(-1)).otherwise(fn.lit(0))

df\_copy = df\_copy.withColumn('num units short', short\_entry\_exit)

w = Window().partitionBy().orderBy(fn.col("time"))

df\_copy = df\_copy.withColumn('numUnits', fn.col('num units long') + fn.col('num units short'))

df\_copy = df\_copy.withColumn("shiftedSpread", fn.lag(df\_copy.spread).over(w))

df\_copy = df\_copy.withColumn('spread pct ch', (fn.col('spread') - fn.col('shiftedSpread')) / ((fn.col(s1) \* fn.abs(df\_copy.hr)) + fn.col(s2)))

df\_copy = df\_copy.withColumn("numUnitsShifted", fn.lag(df\_copy.numUnits).over(w))

df\_copy = df\_copy.withColumn('port rets', fn.col('spread pct ch') \* df\_copy.numUnitsShifted)

try:

sharpe = df\_copy.select(fn.avg('port rets') / fn.stddev('port rets') \* sqrt(252)).collect()[0][0]

if not sharpe:

sharpe = 0.0

except ZeroDivisionError:

sharpe = 0.0

return s1+" "+s2, sharpe

# In[9]:

results = []

split = int(combined\_cryptos.count() \* .6)

for pair in pairs:

if pair[2] < critical\_level:

df\_subset = combined\_cryptos.withColumn("index", fn.monotonically\_increasing\_id())

df\_subset = df\_subset.orderBy(fn.desc("index")).drop("index")

df = df\_subset.limit(split)

df\_reverse = df.withColumn("index", fn.monotonically\_increasing\_id())

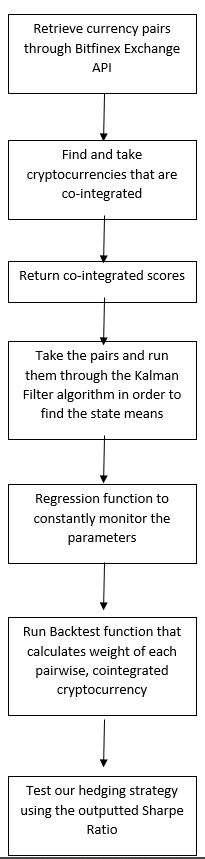
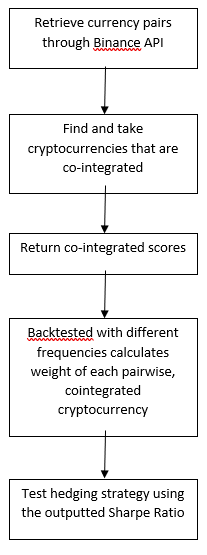
df = df\_reverse.orderBy(fn.desc("index")).drop("index")

rets, sharpe = backtest(df\_subset.limit(split),pair[0],pair[1])

results.append((rets, sharpe))

print("The pair {} and {} produced a Sharpe Ratio of {}".format(pair[0],pair[1],round(sharpe,2)))

**Design Document and Flowchart**

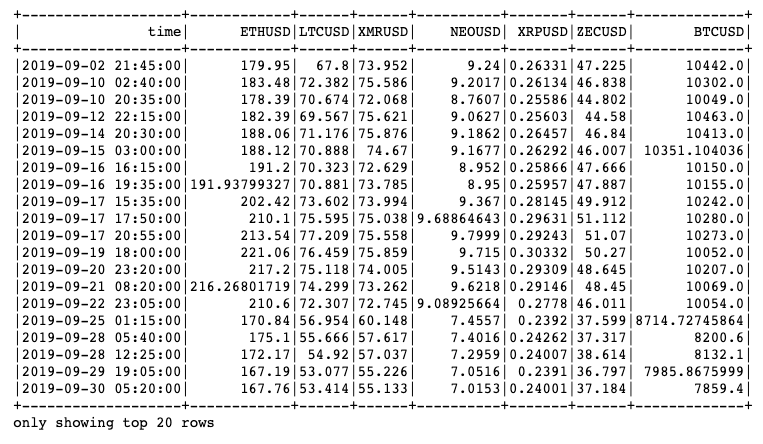
****

Kalman Filter Flowchart Standard Pairs Trading Flowchart

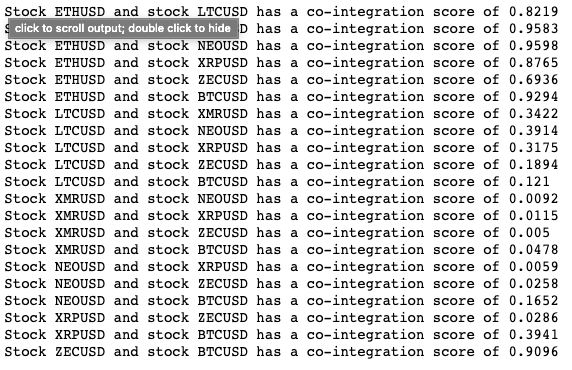
### 

### **Data Analysis and Discussion**

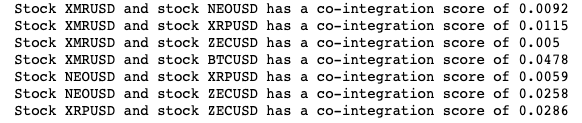
**Output Generation and Analysis**

Our program read in inputs of cryptocurrencies from the bitfinex exchange API. Prices of seven popular cryptocurrencies stored into PySpark dataframes. The collected data was cleaned and the pairs were merged and sorted based on common times that all of the currencies were traded.

From the table above, the currencies that were chosen were traded in various ranges. The largest chosen currency was traded between $7,000 to $10,000 per unit, and the smallest chosen currency traded between a range of a few cents per unit.

Each pair of cryptocurrencies were co-integrated against each other and a score was determined based on the relationships between the two means. The cointegration score ranges from 0 to 1. A lower co-integration score indicates that the means of both currencies will not deviate much from each other as time passes. The following are the co-integration scores of each pair of currencies.

The figure above indicates that there are a lot of pairs that are not co-integrated. Pairs trading should be done with pairs that have a low cointegration score. A critical threshold score of 0.05 was selected to analyze only the most co-integrated pairs. The following pairs met the critical threshold score.



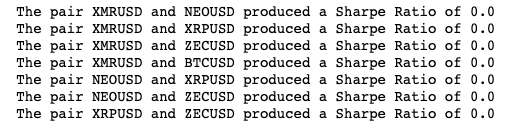
**Compare Output Against Hypothesis**

Overall, we failed to reject the null hypothesis. This meant that we could not ultimately prove that adding the Kalman filter results in better Sharpe ratios. This may be explained by several limitations we ran into over the course of the project. The first of which was that the predefined functions, such as the Kalman filter, required a pandas dataframe to run through. In addition, it took a lot of time to switch from a pyspark dataframe to pandas. This can be seen in the cointegrated pairs function as the runtime took much longer than a preset pairs function that did not do the conversion.

Another limitation we had was the amount of cryptocurrencies we could use. In Fil’s and Kristoufek’s research paper, they used 181 cryptocurrencies over a much longer duration of time. Therefore, we cannot accurately compare our results to their research paper, since we used only 7 cryptocurrencies and the processing was already too much for our machines to handle.

If these limitations were removed, then it would be likely that our observations would line up with our hypothesis, since the Kalman filter should provide us with better and more accurate results.

**Discussion**

We pulled seven popular cryptocurrencies including Bitcoin using the Bitfinex API and cleaned their data from September-2019 to prepare to find if any currencies were cointegrated. The cointegrated pairs were then run through the Kalman Filter, which helped reduce noise, and then produced Sharpe Ratios. The cointegrated pairs that matched the threshold had the following Sharpe ratios.

The cointegrated pairs that were tested resulted in a Sharpe ratio of 0. A Sharpe ratio value above 1 denotes that the returns of investing in the pair of stocks are better than the risk-free rate and that their excess returns are above their excess risks. This means that it is a good idea to invest in the pair of stocks. A Sharpe ratio value between 0 and 1 denotes that the returns derived from investing in the stocks are better than the risk-free rate, but their excess risks exceed their excess returns. This means that it is a good idea to invest in the pair of data within the given time frame. However, the volatility of the pair of stocks would be very high, which means it might not be a good idea to invest in the pair outside of the time frame. A Sharpe ratio value of 0 means the excess return is zero, which is when the return on the portfolio is exactly equal to the risk-free rate. It is not recommended to invest in pairs that have a Sharpe ratio of 0 because there is no gain in investing in these pairs of currencies.

### **Conclusions and Recommendations**

**Summary and Conclusions**

Our hypothesis is that using a Kalman Filter Pairs Trade will perform a better Sharpe ratio than a standalone Pairs trading method. In the end, our project could not support our hypothesis due to the limitations stated before. With more cryptocurrencies and better runtimes, we should be able to determine if the Kalman filter provides us similar or different results when compared to normal pairs trading strategies. The results can be determined based on the Sharpe Ratio - the higher the ratio, the better. A higher score means we’re expecting that our profit margins will be higher with lower implicit risk exposure than those reported in the underlying research.

**Recommendations for Future Studies**

For further studies and recommendations, we can select more cryptocurrencies in order to fully compare our results with the research paper’s results. With a larger dataset, we would be able to find more cointegrated pairs and determine which pairs are worth it to trade based on their Sharpe Ratios. We can also try pulling from stocks and see if there are any differences from crypto since stocks are typically much more stable compared to cryptocurrencies. In addition, work on streamlining the conversion from pandas to pyspark can be continued and could potentially be faster if more documentation appears.

### 

### **Bibliography**

M. Fil and L. Kristoufek, "Pairs Trading in Cryptocurrency Markets," in IEEE Access, vol. 8, pp. 172644-172651, 2020, doi: 10.1109/ACCESS.2020.3024619.

Kim, Youngjoo, and Hyochoong Bang. “Introduction to Kalman Filter and Its Applications.” *IntechOpen*, IntechOpen, 5 Nov. 2018, www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications.

Tinsley, Martin. “Using Cointegration for a Pairs Trading Strategy.” *Trade Like A Machine*, 12 June 2018, www.tradelikeamachine.com/blog/cointegration-pairs-trading/part-1-using-cointegration-for-a-pairs-trading-strategy.

Wang, Letian. “Kalman Filter and Pairs Trading.” *Quantitative Trading and Systematic Investing*, 4 Sept. 2020, letianzj.github.io/kalman-filter-pairs-trading.html.

J., Stuart. “Mean Reversion Pairs Trading With Inclusion of a Kalman Filter.” *Python For Finance*, 25 July 2019, pythonforfinance.net/2018/07/04/mean-reversion-pairs-trading-with-inclusion-of-a-kalman-filter/.

Klein, Carsten. “How to Get Historical Cryptocurrency Data.” *Medium*, Coinmonks, 14 Oct. 2020, medium.com/coinmonks/how-to-get-historical-crypto-currency-data-954062d40d2d.

### 

### **Appendices**

**Graph of cryptocurrency prices over September, 2019 (without Bitcoin)**

