University of Illinois at Urbana-Champaign
Master of Science in Financial Engineering
Grainger School of Engineering, Gies College of Business

Practicum Research Project: Tower Research Capital
Spring 2023

Cross-exchanges Pricing and Anomaly Detection for Crypto Markets

Yu-Ching Liao, MS in Financial Engineering, UIUC
Saranpat Prasertthum, MS in Financial Engineering, UIUC
Hyoung-Woo Hahm, MS in Financial Engineering, UIUC
Xin-Chen Nie, MS in Financial Engineering, UIUC

Abstract

In this practicum research project, we are required to solve the real-world problems for the assigned companies, Tower Research Capital, the New York based High-Frequency Trading firm, on the topic of $Cross-exchanges\ Pricing\ Model\ for\ Crypto\ Marketes$.

The aim of this research project is to utilize different prices of the Crypto pairs in different exchanges to define the index prices of the pairs should have theoretically. If the price has deviated from the theoretical price, it is highly possible that it might cause the loss, as well it might show the arbitrage opportunity.

The main task of this project can be divided into three main steps: 1) Input pre-processing by Anomaly Detection, 2) Decision of the using model by Variance Check and 3) Getting the index prices. The first step is to figure out which prices might provide the "useful" information, and which might not be useful, or even worst, the "noise" in designing the pricing model; we exclude those critical anomalies, such as missing value or stale quote and use the rest as input. The second step is to decide the model we are using by the variance-check criteria: if the input passed the criteria, means that all of the inputs in this moment can be qualified as "High-quality prices", we then return median prices. If the inputs do not passed the variance check, we process the Clustering Pricing Algorithm as our pricing model for this moment. The third step is to utilize the model we decided in last step to get our index prices for this moment. With iterating the above steps, we then obtain the dynamic pricing model that can get the index price and exclude the anomalies for specific moment based on different situation of different moment.

We received 10 csv files, which are $1.csv$, $8.csv$, $9.csv$, $10.csv$, $12.csv$ $16.csv$, $19.csv$, $25.csv$, $28.csv$ and $30.csv$, from Tower Research. Each of those files represent the minute-interval order data of specific exchange, and we have been told that $1.csv$ is the exchange that is the most reliable whose prices are the most robust. As a result, we decided to use the data from other exchanges as the inputs of our model, and compare the output of our model with $1.csv$ to see if our model can provide insights that confirms the most reliable exchange.

As the inspection of the effectiveness of our model, we implemented this on the ask prices of LINK/BTC pairs. We eventually get the R^2 value of 99.85%, MSE of $6.1291 \times e^{-13}$ and MAPE of 0.00179. With this result, we can see that our model is effective in its accuracy of achieving the index price that a pair should have, and efficient in it only take prices of each moment as input, rather than relying on the static index such as liquidity.

As the overall result, after testing on the minute-interval price data of 120 pairs, there are 79% of the pairs whose R^2 reach the value above 99%, indicating our model is as well effective on most of the pairs that is being traded on the Crypto markets.

***Keywords*—** Anomaly Detection, Clustering, Variance Check, Cross-exchange Pricing, Crypto pairs

Table of Content

1 Introduction

The Cryptocurrency market has experienced rapid growth in recent years, with an increasing number of exchanges offering various trading pairs. This expansion has led to inconsistencies in pricing across different platforms, creating both risks and opportunities for traders and market participants. In this research paper, we aim to address the problem of cross-exchange pricing and anomaly detection for the Cryptocurrency market, with a focus on replicating a high-quality exchange data feed for each Crypto pair using various price feeds from multiple exchanges for profit and loss (PNL) purposes. This approach reduces the risk associated with relying on a single exchange, which may not always provide the most accurate or reliable pricing data.

The Cryptocurrency market's decentralized nature and the absence of a centralized authority to regulate prices contribute to the issue of price discrepancies across exchanges. These discrepancies can result in potential arbitrage opportunities for traders who can exploit the price differences between exchanges. However, these opportunities also come with inherent risks, as rapid price fluctuations and inaccurate pricing information can lead to significant losses. Therefore, it is crucial to develop a reliable pricing model that addresses these challenges and offers a more accurate representation of the market's true value.

The objective of our research project, in collaboration with Tower Research Capital, a New York-based High-Frequency Trading firm, is to develop a dynamic pricing model that leverages multiple exchange data sources to determine an accurate index price for each Cryptocurrency pair. By identifying deviations from the theoretical price, our model will be able to detect potential arbitrage opportunities and minimize losses caused by inaccurate pricing information.

Our proposed methodology comprises three main steps: 1) Anomaly detection for input pre-processing, 2) Model selection based on variance check, and 3) Index price calculation. The anomaly detection step aims to filter out "noisy" or unreliable data points by identifying critical anomalies such as missing values or stale quotes. By excluding these anomalies, we ensure that our model's input consists of only high-quality, relevant data points.

In the second step, our model selects the most suitable pricing model based on variance-check criteria. If the input data passes the criteria, indicating that all inputs can be qualified as "High-quality prices," our model returns the median prices. If the inputs do not pass the variance check, we process the Clustering Pricing Algorithm as our pricing model for that particular moment. This step ensures that our model adapts to the varying quality of input data and selects the most appropriate pricing model accordingly.

The third step involves utilizing the chosen pricing model to calculate the index price for each Cryptocurrency pair. By iterating the above steps, our dynamic pricing model can generate an index price that accurately reflects the true value of each trading pair while eliminating the impact of anomalies and adapting to different market conditions.

To validate the effectiveness of our model, we will test it on minute-interval order data for the LINK/BTC pair, obtained from ten different exchanges provided by Tower Research. By comparing the outputs of our model with the most reliable exchange's data ($1.csv$), we will assess the accuracy of our model in replicating a high-quality price feed. Our research also extends to testing the model on minute-interval price data of 120 different Cryptocurrency pairs.

2 Methods Review

2.1 Moving Average

Moving Average is a commonly used statistical method for analyzing time-series data. It is used to smooth out the fluctuations in the data and highlight any long-term trends. The Moving Average is calculated by taking the average of a specified number of data points from the past.

The formula for calculating the Moving Average of a time-series X_t with a window size of n is given by:

$$MA_t = \frac{1}{n} \sum_{i=t-n+1}^{t} X_i \tag{1}$$

where MA_t is the Moving Average at time t and X_i represents the value of the time-series at time i.

The Moving Average can be used to identify trends in the data and provide a smoother representation of the data over time. It can also be used to forecast future values by using the Moving Average as a predictor.

There are different types of Moving Averages, such as Simple Moving Average, Weighted Moving Average, and Exponential Moving Average. The choice of Moving Average type depends on the characteristics of the data and the purpose of the analysis.

In our experiment, we used the Simple Moving Average with a window size of 10 to smooth out the fluctuations in the Crypto pair prices data. The Moving Average was calculated for each Crypto pair at each time interval, and the resulting values were used as input for further analysis.‘‘‘

In this example, we provide a brief overview of Moving Average and its formula. We also mention different types of Moving Averages and explain how the choice of Moving Average type depends on the characteristics of the data and the purpose of the analysis. Finally, we describe how we used the Simple Moving Average in our experiment, including the choice of window size and the calculation of the Moving Average for each Crypto pair.

2.2 Variance

Variance is a measure of the dispersion of data points in a dataset. It is calculated as the average of the squared differences between each data point and the mean of the dataset. The formula for variance is given by:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 \tag{2}$$

where σ^2 is the variance, N is the number of data points, x_i is the i-th data point, and μ is the mean of the dataset.

2.3 Moving Variance

Moving variance is a method to compute the variance of a time series over a sliding window of fixed size. It is useful for understanding how the variability of the data changes over time. The moving variance can be calculated using the following formula:

$$\sigma_t^2 = \frac{1}{w} \sum_{i=t-w+1}^{t} (x_i - \mu_t)^2 \tag{3}$$

where σ_t^2 is the variance at time t, w is the window size, x_i is the i-th data point, and μ_t is the mean of the data points within the window.

2.4 K-Means Clustering

K-means clustering is an unsupervised learning algorithm used to partition a dataset into k clusters. The algorithm iteratively assigns each data point to one of the k clusters based on its distance to the cluster's centroid. The centroids are then updated as the mean of the data points in the cluster. The process is repeated until convergence. The objective is to minimize the within-cluster sum of squares:

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} ||x_i - \mu_j||^2 \tag{4}$$

where C_j is the j-th cluster, x_i is the i-th data point, and μ_j is the centroid of cluster C_j.

2.5 Fourier Transform and Inverse Fourier Transform

The Fourier Transform is a mathematical technique used to convert a time-domain signal into its frequency-domain representation. The Inverse Fourier Transform converts the frequency-domain representation back to the time-domain signal. The Fourier Transform $F(\omega)$ of a continuous time-domain signal $f(t)$ is given by:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{5}$$

The Inverse Fourier Transform $f(t)$ is given by:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \tag{6}$$

2.6 R^2

R^2, also known as the coefficient of determination, is a measure of how well a linear regression model fits the observed data. It is calculated as the proportion of the total variation in the dependent variable that is explained by the independent variable(s). The R^2 value ranges between 0 and 1, with 1 indicating a perfect fit. The formula for R^2 is:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{7}$$

where SS_{res} is the sum of squared residuals, and SS_{tot} is the total sum of squares. The sum of squared residuals is calculated as:

$$SS_{res} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{8}$$

and the total sum of squares is calculated as:

$$SS_{tot} = \sum_{i=1}^{N} (y_i - \bar{y})^2 \tag{9}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the observed values.

2.7 Mean-Square Error (MSE)

Mean-Square Error (MSE) is a measure of the average squared difference between the observed values and the predicted values of a model. The lower the MSE, the better the model's predictions are. The formula for MSE is given by:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{10}$$

where N is the number of observations, y_i is the observed value, and $\hat{y}_i$ is the predicted value.

2.8 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) is a measure of the average absolute percentage difference between the observed values and the predicted values of a model. The formula for MAPE is given by:

$$MAPE = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{11}$$

where N is the number of observations, y_i is the observed value, and $\hat{y}_i$ is the predicted value.

3 Experiment

In this chapter, we conduct numerical experiments to obtain the index price, which we can afterward compared the $1.csv$ prices to look into its performance. As shown in the flow chart below, we divide our numerical experiments into three parts.

First, for each of the new input, we process the Anomaly Detection first, to see if there is anomalies in our input, and if yes, we can then exclude those anomalies and thus have cleaner input.

Second, after Anomaly Detection, we then process Variance Filtering to see if we need to process the Clustering Pricing for this moment.

Third, we obtain the index price via the method we chose by Variance Filtering. If our model decided to process Clustering Pricing, we then get the index prices for this moment by the result of Clustering Pricing Model. If not, we then return median prices as our index prices.

We then repeat this to get the index prices for all moments based on difference situation of different moment.

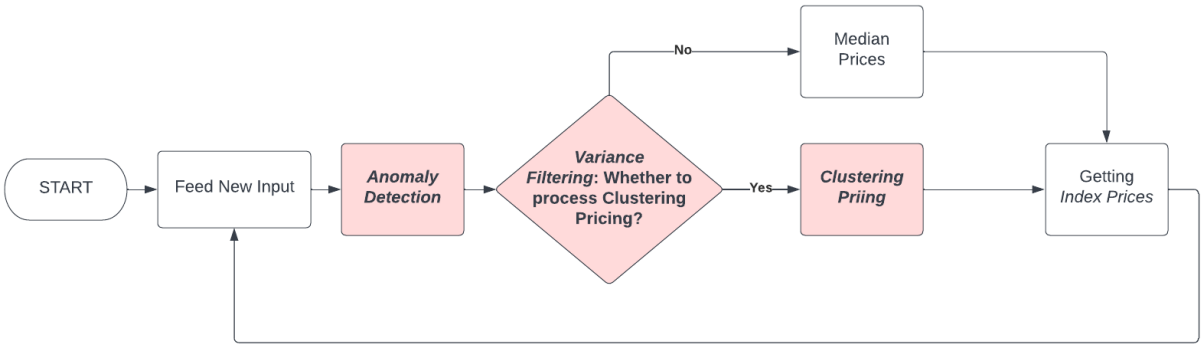


Figure 1: Flow Chart of Experiment

3.1 The environment of experiment

We construct our model by real data from Tower Research Capital and with programming language Python. The experiment run on a Macbook Pro, the equipment of the testing environment is as below:

CPU	Apple M2 Pro
Memory	32 GB
OS	macOS 13.3.1 (22E261)

3.2 The data of experiment

The experiment utilized minute-interval trading record data of different Crypto pairs in various exchanges ranged from November 7th to November 15th.

3.2.1 Description of data

Tower Research Capital provided ten CSV files containing the data for the experiment, namely $1.csv$, $8.csv$, $9.csv$, $10.csv$, $12.csv$, $16.csv$, $19.csv$, $25.csv$, $28.csv$, and $30.csv$. Each file represents the order of a specific exchange. The experiment used the data from $1.csv$ as the index since it was deemed the most reliable and robust. The data covers a period of several days and consists of various trading parameters, including the ask price, bid price, ask volume, bid volume, and traded Crypto pair. As shown below, we can see that the prices from different exchanges can be significantly different, and some of them even contain extreme anomalies that is highly inconsistent with other prices.

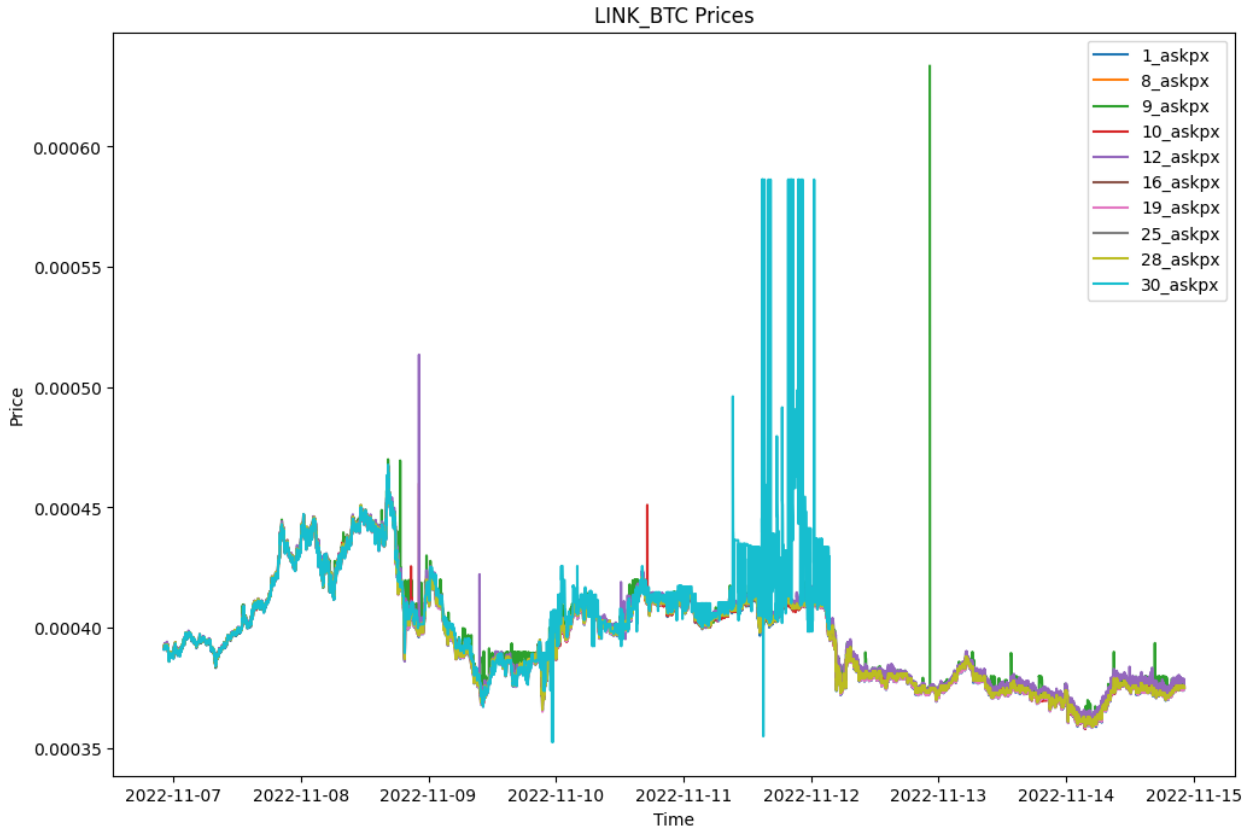


Figure 2: LINK/BTC Ask Prices on all exchanges

3.2.2 Data pre-processing

Since we are to use the ask price of different pair in this experiment, we first convert the ”exchange” data into ”Crypto ask price” data.

To be more specific, the original data was divided into 10 csv data, which each of them is representing the trading record of specific exchange. We then convert them into numbers of csv files that is divided by the traded pair. For example, original data, as shown below, we can see that they are representing the order record of EOS/ETH and LINK/BTC in Exchange 1, 10 and 19 on 22:17 and 22:18 on 2022/11/06.

Table 1: Original data-set overview.

Exchange	Symbol	time	Bid Price	Bid Qty	Ask Price	Ask Qty
Exchange 1	EOS/ETH	2022/11/06 22:17	0.000704	25310000	0.000705	45390000
Exchange 1	LINK/BTC	2022/11/06 22:17	0.0003908	20648000	0.0003909	880000
Exchange 1	EOS/ETH	2022/11/06 22:18	0.000704	47490000	0.000705	23450000
Exchange 1	LINK/BTC	2022/11/06 22:18	0.0003913	10914000	0.0003914	1100000
Exchange 10	EOS/ETH	2022/11/06 22:17	0.00005321	23431692	0.00005336	27368908
Exchange 10	LINK/BTC	2022/11/06 22:17	0.00039032	600000	0.00039129	4435879
Exchange 10	EOS/ETH	2022/11/06 22:18	0.0000532	5600000	0.00005333	22241080
Exchange 10	LINK/BTC	2022/11/06 22:18	0.0003907	1700724	0.00039171	600000
Exchange 19	EOS/ETH	2022/11/06 22:17	0.00071285	21000000	0.00071337	21000000
Exchange 19	LINK/BTC	2022/11/06 22:17	0.00043207	300000	0.00043257	2232000
Exchange 19	EOS/ETH	2022/11/06 22:18	0.00071289	21000000	0.00071361	21000000
Exchange 19	LINK/BTC	2022/11/06 22:18	0.00043236	1300000	0.00043263	188000

We then convert this order record into the file with the format that is divided by different symbol of pairs as shown below:

Table 2: Converted data-set overview.

Symbol	time	Exchange 1	Exchange 10	Exchange 19
EOS/ETH	2022/11/06 22:17	0.000705	0.00005336	0.00071337
EOS/ETH	2022/11/06 22:18	0.000705	0.00005333	0.00071361
LINK/BTC	2022/11/06 22:17	0.0003909	0.00039129	0.00043257
LINK/BTC	2022/11/06 22:18	0.0003914	0.00039171	0.00043263

With this format of data, we can then call the data of each of the pairs we have individually, rather than storing all the record in the memory and thus slow down the program. We can also noticed that their are the differences among prices, where our model are aimed to obtain the index price from these differences.

3.3 Design of experiments

We will show how we design the experiments in this section. As above, the main process of this experiment can be divided into three parts: Anomaly Detection for Critical Anomalies Exclusion, Variance Check for Model Selection and Getting Index Price. We will, as well, show the required packages for this experiment and how are we going to mimic that we are feeding the data in real-time.

3.3.1 Importing the packages

We adopt certain packages which are needed to complete this experiment. The packages we utilize are shown below:

Package	Utilization
Pandas	Data manipulation and analysis
NumPy	Numerical computations
Scikit-learn	Machine learning and data mining

Table 3: Packages used overview.

For a more detailed representation, we list the specific functions and classes imported from each package and their corresponding utilization:

Package	Function/Class	Utilization
`Pandas`	`pandas`	Data manipulation and analysis
`NumPy`	`numpy`	Numerical computations
`Scikit-learn`	`euclidean_distances`	Pairwise distance computation
`Scikit-learn`	`Normalizer`	Data normalization
`Scikit-learn`	`KMeans`	Clustering algorithm
`Scikit-learn`	`make_pipeline`	Pipeline construction
`Scikit-learn`	`mean_squared_error` (MSE)	Regression evaluation metric
`Scikit-learn`	`r2_score` (R2)	Regression evaluation metric
`Scikit-learn`	`mean_absolute_percentage_error` (MAPE)	Regression evaluation metric

Table 4: Packages used details.

We import those packages via the following Python code:

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.preprocessing import Normalizer
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score as R2
from sklearn.metrics import mean_absolute_percentage_error as MAPE
```

3.3.2 Importing data in real-time

The experiment used a real-time data streaming process to continuously receive the trading data and feed it into the model. To achieve so, we first read the price data via `read_csv` function of `pandas` package. Using LINK/BTC as an example, we define `exchanges_data_df` and `time_list` via following code:

```
exchanges_data_df = pd.read_csv("LINK/BTC.csv")
time_list = exchanges_data_df.index
```

Where `exchanges_data_df` is data frame of LINK/BTC ask prices and `time_list` is the timestamp that LINK/BTC being traded. We then fetch the data by each minute via following code:

```
for moment_indx in range(len(time_list)):
    curr_moment = time_list[moment_indx]
    prices_this_moment = exchanges_data_df[curr_moment]
```

By the Python code above, we can then mimic the real-time inputting of prices since we will now feed the prices as the points unevenly distributed in single dimensional space as shown below.

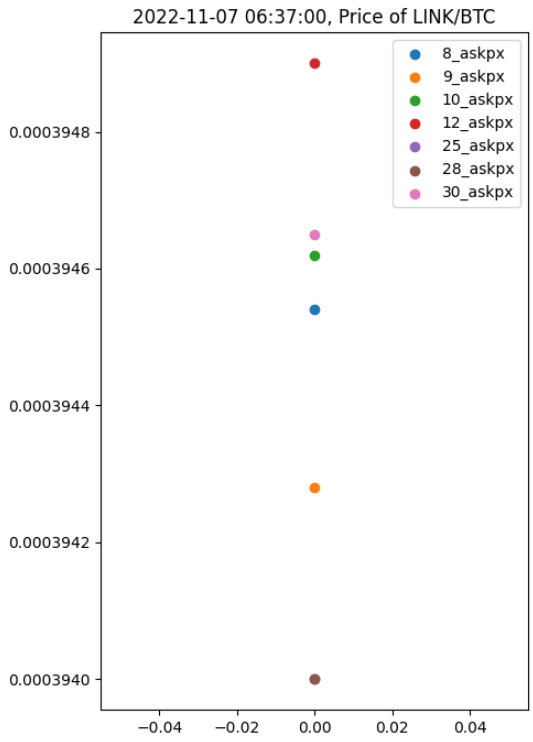


Figure 3: Prices of LINK/BTC

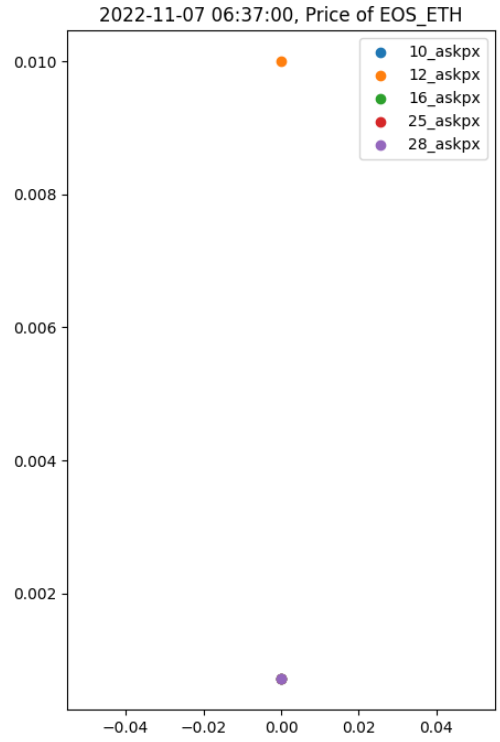


Figure 4: Prices of EOS/ETH

Note that we are not to include the price from $1.csv$ here since we are using it as a target rather than features. We can notice that, even if there are 5 exchanges trade on EOS/ETH on 06:37 on 2022/11/07, the

plot shows only two prices. This is due to the extraordinary price of Exchange 12 was so distant from others that making others seemed to be clustered together. As a result, in such cases, we are to deal with anomalies that have abnormal performance as Exchange 12 so that we can exclude those useless or even harmful input and thus have more accurate and reliable index prices.

For the next part, we will show how are we going to exclude the anomalies via Anomaly Detection model.

3.3.3 Anomaly Detection for Critical Anomaly Exclusion

In this section of the research paper, we will discuss the importance of anomaly detection and its potential to enhance the accuracy of certain pairs in index price calculations, as outlined in Section 3.3.5. In the aforementioned section, it was observed that price data clustered around the median were deemed high quality and subsequently utilized for index price calculation. However, following the application of this method, we identified pronounced spikes in the index prices for less popular pairs, such as EOS/ETH and IOTA/BTC. Further investigation revealed that these inconsistencies were caused by significant data loss, as illustrated in Figure 5 (below). It was determined that the inaccuracies in index price calculations were due to erroneous input data from the provided CSV file. To address this issue, we sought to identify and exclude anomalies occurring on each exchange for less popular Cryptocurrency pairs. Based on our observations, the following types of anomalies were detected:

1. Stale Quote: Exchange quotes have ceased updating for a considerable duration.
2. Missing Quote: Exchange quote data is absent.
3. Frequent Missing Quote: Exchange quotes are consistently missing.
4. Negative Bid/Ask Spread: Exchange exhibits a negative spread.
5. Negative Volume: The sum of bid and ask volume for the exchange is less than zero.

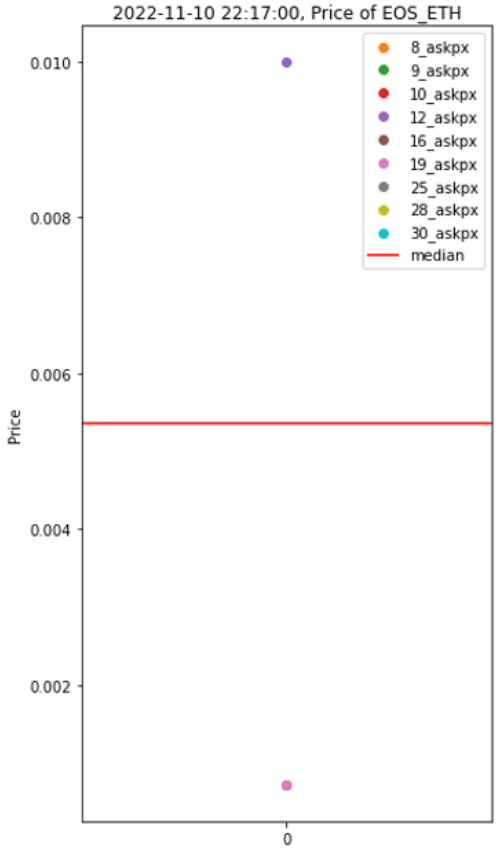


Figure 5: Quote of EOS/ETH

Subsequently, we categorized each anomaly's severity according to its observed impact on the reliability of the exchange quote: Normal, Non-Critical, and Critical. This categorization serves as a crucial aspect of

anomaly detection. Non-Critical severity serves as a warning to users that an anomaly has occurred, while Critical severity not only warns the user but also provides them the option to remove the critically severe anomalies from the index price calculation methods by employing the ”Force Use Normal Price” feature.

Our analysis of the optimal setting from the dataset provided revealed that all identified anomalies should be classified as Critical, with the exception of negative volume, as there was no observed correlation between exchanges exhibiting negative volume and the provision of unreliable prices. Upon enabling the aforementioned feature, the resulting data exhibited fewer spikes and increased accuracy, as depicted in Figures 6 and 7 below. This anomaly detection underscores the significance of input data quality for less popular pairs. However, for more popular pairs, utilizing the ”Force Use Normal Price” feature may yield inaccurate results, as it may remove excessive input data from index price calculations.

It is in the user's best interest to examine the behavior of each pair and the index price result as described in Section 3.3.5. This examination should include the identification of spikes and any potential anomalies that may be contributing to these irregularities.

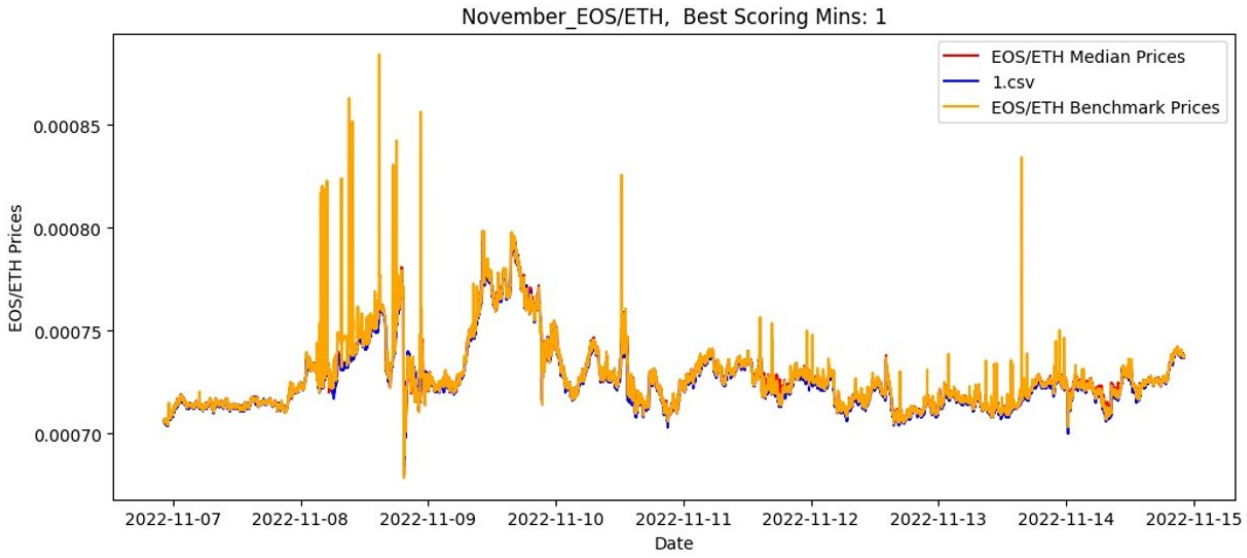


Figure 6: EOS/ETH Result without ”Force Use Normal Price”

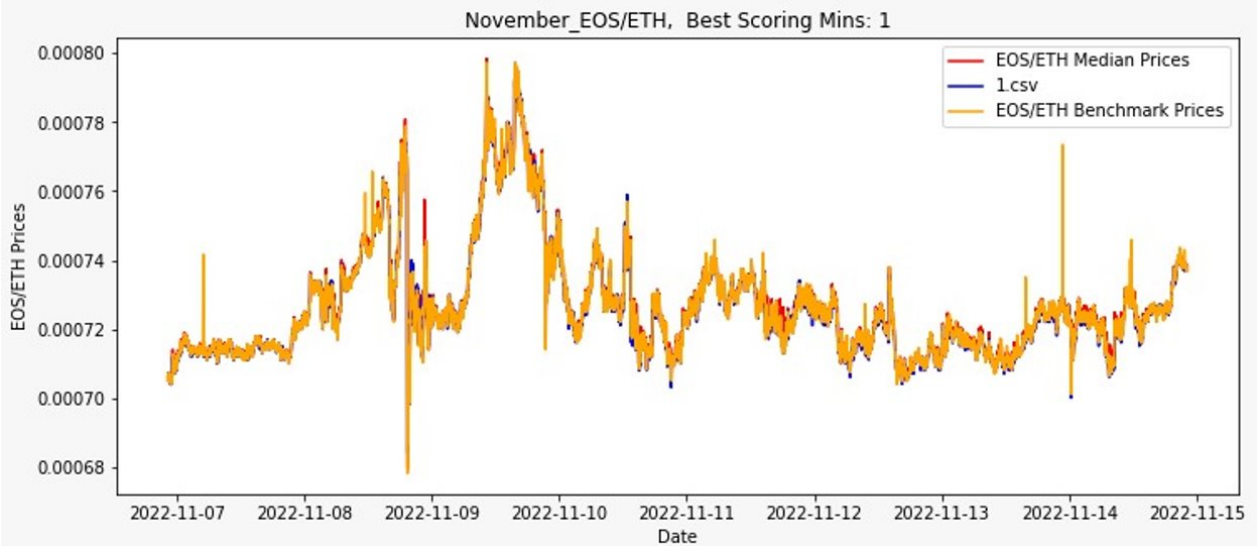


Figure 7: EOS/ETH Result with ”Force Use Normal Price”

The following code snippet demonstrates how we detect stale quotes in our model:

```
def detect_stale_quote(exchange_name,price_type, price):
    if np.isnan(price): return False
    if exchange_name not in prev_qoutes:

        #IF THE EXCHANGE IS NOT IN THE DICTIONARY, ADD IT WITH THE CURRENT
            QUOTE USING DEQUE WILL DISCARD THE OLDEST ONE
        prev_qoutes[exchange_name] = {}
        prev_qoutes[exchange_name][price_type] = deque([price],maxlen=
            stale_qoute_window)

    else:
        if price_type not in prev_qoutes[exchange_name]: prev_qoutes[
            exchange_name][price_type] = deque(maxlen=stale_qoute_window)
        prev_qoutes[exchange_name][price_type].append(price)

        # CHECK IF THE CURRENT QUOTE IS THE SAME AS THE PREVIOUS N-1 QUITES
        if len(prev_qoutes[exchange_name][price_type]) == stale_qoute_window
            and all(quote == prev_qoutes[exchange_name][price_type][-1] for
            quote in prev_qoutes[exchange_name][price_type]):
             return True  # STALE QUOTE DETECTED
    return False  # QUOTE IS NOT STABLE
```

The ‘detect_stale_quote‘ function commences by evaluating if the given price is NaN (not a number), as establishing the staleness of a quote is unattainable under such circumstances. If the price is not NaN, the function proceeds to verify whether the historical data for the specified exchange is present within the ‘prev_qoutes‘ dictionary. In the absence of historical data, a new entry is added and a deque for the price type is initialized, with a maximum length determined by the pre-established window.

In contrast, if the historical data for the exchange already exists in the dictionary, the function iterates through all previous quotes to ascertain their staleness. Upon completing this analysis, the history is updated with the most recent quote. Subsequently, the function proceeds to evaluate the incoming pricing data for all exchanges in a consistent manner.

3.3.4 Variance Check for Model Selection

The price of the Crypto currency fluctuates every second, so we had to test our model with data that had much shorter intervals. This resulted in a significant increase in data size, which led to computation time issues. To address this problem, we implemented a variance filtering (VF) function. The VF function calculates the standard deviation of the prices for each minute and filters them using both 12 hour long-term moving average(LTMA) and 1 hour short-term moving average(STMA).

As you can see from the figure 5, standard deviation of the prices fluctuates a lot. Using this noise as an input we wanted to filter out the feeds with high standard deviation so that our clustering model will be used when it can bring meaningful outcome. Consequently, based on the standard deviation of the prices, the VF function decides when to cluster the exchanges using LTMA and STMA as a criteria. If the standard deviation of that moment exceeds either of LTMA or STMA, the function will decide to use the clustering model. However when the standard deviation of the prices is lower than the both criteria, it will return the median of the prices rather than using the clustering model.

By implementing VF, we were able to decrease computation time significantly. We observed an 175-fold increase in iteration speed, from 18.87 iterations/s to 3290.93 iterations/s. Although there was a slight decrease in the R^2 value due to the trade-off relationship, the decrease was not significant. Overall, the implementation of the VF function proved to be an effective solution to address the computation time issue we faced when dealing with data that had much shorter intervals.

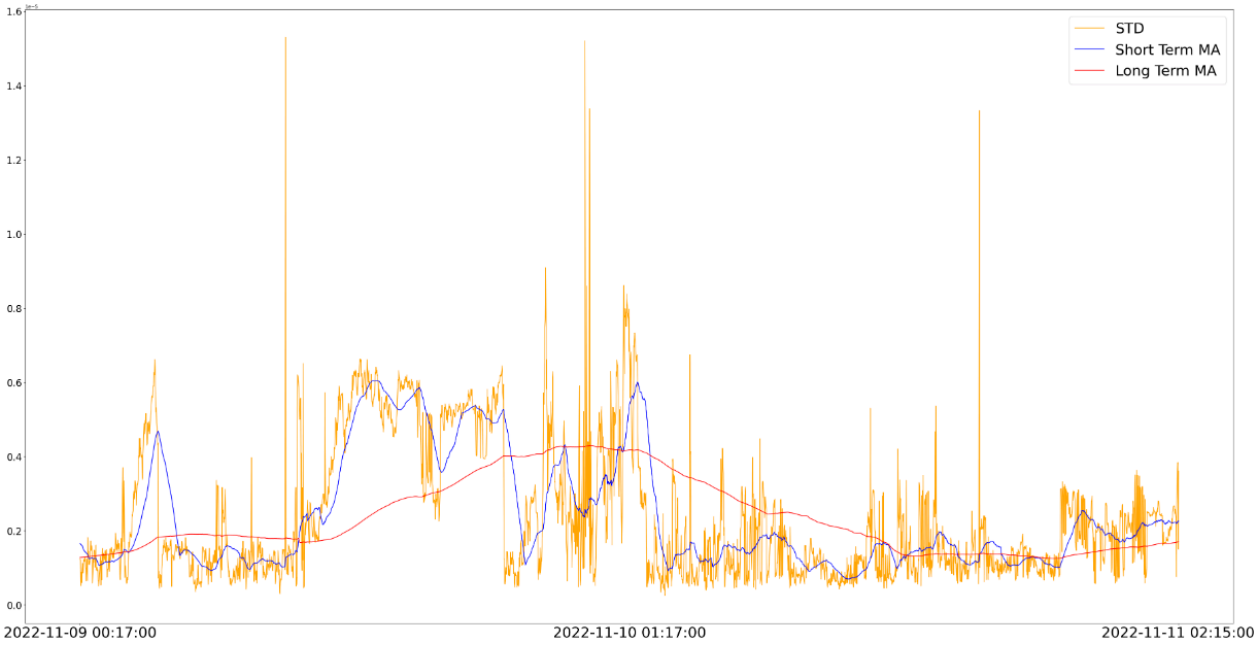


Figure 8: LINK/BTC Standard Deviation Plot

```python
#Vaninace check for model selection
def var_checking(use_varcheck,curr_prices):

    # CHECK IF THE VAR_CHECKING FUNTION SHOULD BE ENABLES OR NOT
    if not use_varcheck: return True

    #FUNCTION TO UPDATE THE MOVING AVERAGE OF THE STANDARD DEVIATION
    def update_std_moving_window(curr_std):
        std_ma_window_l.append(curr_std)
        std_ma_window_s.append(curr_std)
    curr_std = np.std(curr_prices)

    # COMPARE THE STD WITH THERSHHOLDS AND THEN RETURN THE RESULT
    if curr_std > np.mean(std_ma_window_s) or curr_std >= std_ma_window_l:
        update_std_moving_window(curr_std)
        return True
    return False
```

3.3.5 Getting Index Prices

After the Variance Filtering, we will decide whether to process the Clustering Pricing to order to get the Index Price for this moment. If not, we will simply return the median of all inputs of this moment. However, if yes, we then process the Clustering Pricing to obtain the Index Price of this moment. In other words, we can say that this is the core calculation procedure of our model.

To develop a robust price index, it is essential to differentiate high-quality from low-quality prices. In this study, a clustering model in machine learning was employed to categorize the prices into high-quality and low-quality groups based on their distances to the median of all prices at a given moment. This approach facilitates the identification and exclusion of outliers, which might be detrimental to the calculation of the index price.

As a result, for the first step, we calculate the distances to median of the prices for all the prices as shown in the left-hand side below. After obtained all distances of all prices to median, we can then obtain the result as shown in the right-hand side below. We can achieve so via the Python code below.

```
# GETTING THE PRICES FOR THIS MOMENT AND EXCLUDE THE NAN VALUE
prices_this_moment = exchanges_data_df[curr_date]
prices_this_moment = prices_this_moment.dropna()

#GETTING THE MEDIAN PRICE
med_price = np.median(prices_this_moment)

#CALCULATING THE DISTANCE TO MEDIAN OF ALL PRICES
distances_from_mean = abs(prices_this_moment - med_price)
```

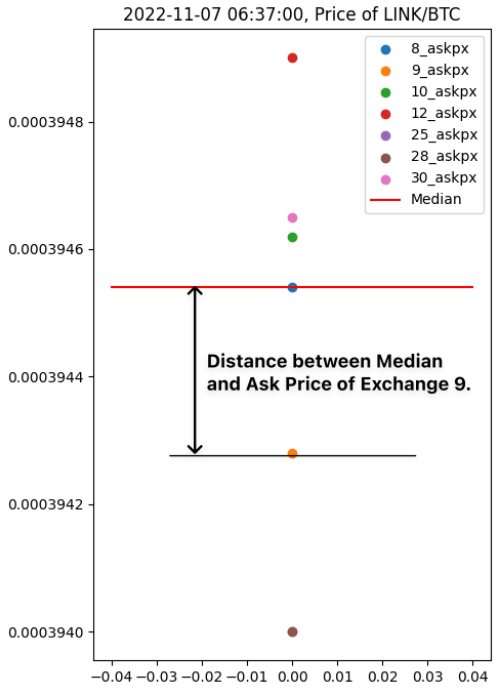


Figure 9: Price and Median of LINK/BTC

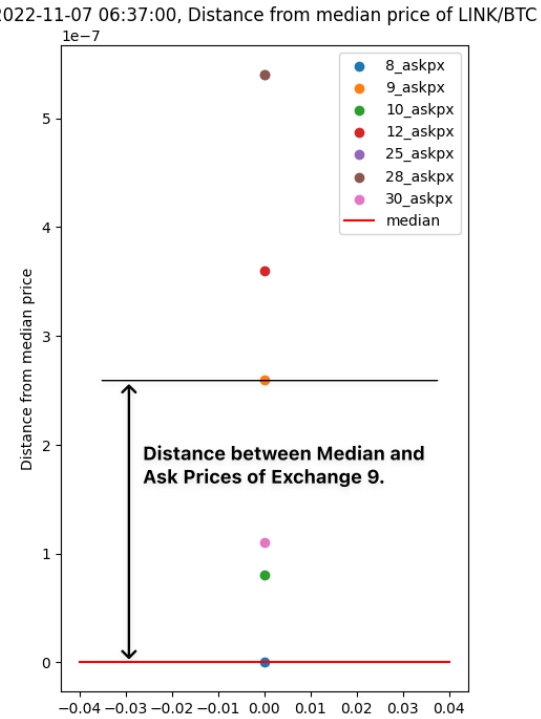


Figure 10: Distances to Median of all prices

From the result in the right-hand side above, we can see that some of the prices are closer to 0, while some of them are not. We then process to implement K-Means Clustering Model on those distances. As shown in the left-hand side below, we can see that those distant from the median will be clustered in the same cluster, while those close to median will also clustered in the same cluster. We then afterward convert them back to the prices, thus we can get the result as shown in the right-hand side below. The following Python code is to implementing this. Note that is the section of `## DEFINING WHICH LABEL IS HIGH - QUALITY AND WHICH IS LOW - QUALITY BY MEAN OF DISTANCE`, since the K-Means clustering model will not directly inform us which are the high-quality prices and which are low-, we have to define the high- and low- quality by whose ”Mean of Distance to Median” is smaller. Since that ”the mean of distance to median is small” indicates that these prices are more closer to median, we then define the cluster as high-quality prices if its mean of distance to median is smaller. And we also define the high- and low-quality exchanges of this moment for the following steps.

```
# RESHAPE DISTANCES INTO 2D ARRAY TO FIT THE K-MEANS MODEL
distances_from_mean_reshaped = distances_from_mean.values.reshape(-1, 1)

#IMPLEMENT K-MEANS MODEL WITH 2 CLUSTERS
kmeans = KMeans(n_clusters=2)
kmeans.fit(distances_from_mean_reshaped)

#GETTING THE LABEL OF EACH PRICES
```

```
clustering_label = np.array(kmeans.predict(distances_from_mean_reshaped))

#DEFINING WHICH LABEL IS HIGH-QUALITY AND WHICH IS LOW-QUALITY BY MEAN OF
    DISTANCE
if distances_from_mean[clustering_label == 0].mean() > distances_from_mean[
    clustering_label == 1].mean():
    hq_prices = prices_this_moment[clustering_label == 1]
    lq_prices = prices_this_moment[clustering_label == 0]
    hq_exchanges = prices_this_moment.index[clustering_label == 1]
    lq_exchanges = prices_this_moment.index[clustering_label == 0]
else:
    hq_prices = prices_this_moment[clustering_label == 0]
    lq_prices = prices_this_moment[clustering_label == 1]
    hq_exchanges = prices_this_moment.index[clustering_label == 0]
    lq_exchanges = prices_this_moment.index[clustering_label == 1]
```

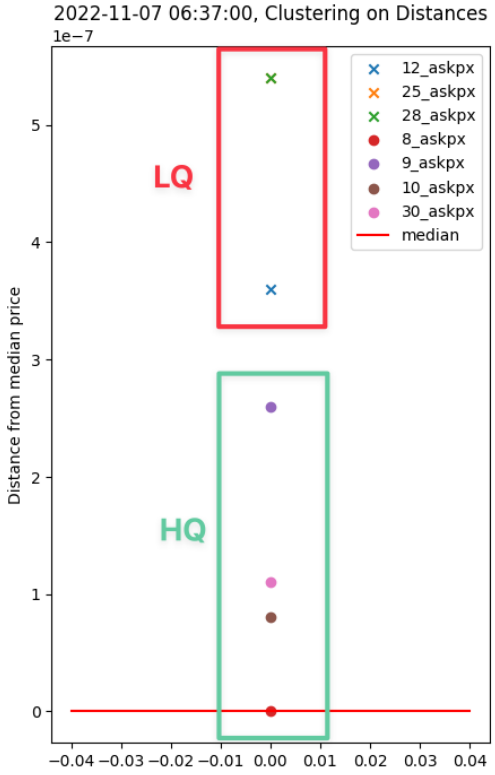


Figure 11: Clustering on Distances

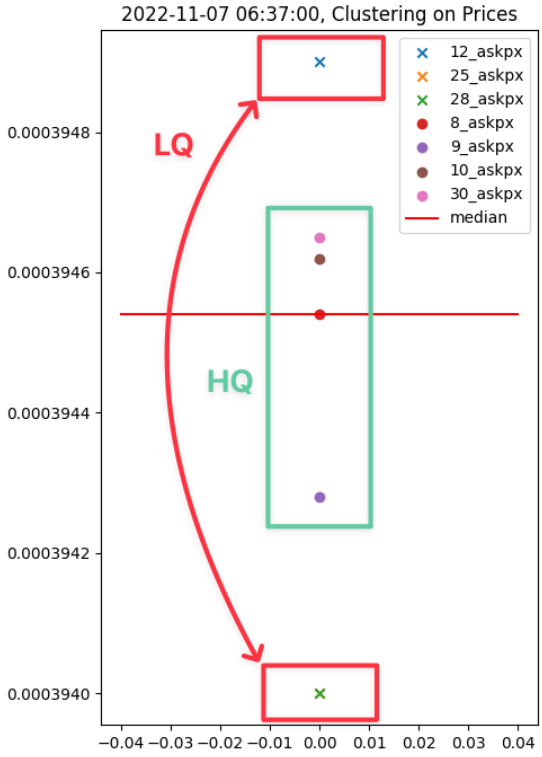


Figure 12: Clustering on Prices

By applying this clustering model to new inputs at each moment, high-quality and low-quality groups can be identified based on the price distribution at that moment, rather than relying on static criteria. The proportion of time that a specific exchange is clustered as high-quality can then be used to calculate the weights for each exchange.

For instance, as shown below, if minute-interval data is used with a scoring window of 5 minutes, and Exchange 19 clustered as high-quality 5 times would have a weight of 1, while another Exchange 8 clustered as high-quality 3 times would have a weight of 0.6. Note taht different Cryptocurrencies have different nature, thus have different scoring windows, which were determined through Hyperparameter tuning.

	HQ count	LQ count	Weights
Exchange 8	3	2	$3/(3+2) = 0.6$
Exchange 19	5	0	$5/(5+0) = 1.0$
Exchange 25	1	4	$1/(1+4) = 0.2$

Table 5: High-quality (HQ) and low-quality (LQ) count and weights for each exchange.

This can be achieve by the following Python code:

```
#DEFINE THE SCORING WINDOW (WE ASSUME IT IS 8000 MINS HERE)
exchanges_list = exchanges_data_df.columns
n_scoring_window = 8000
scoring_window = pd.DataFrame(0, columns=["HQ Count", "LQ Count", "Weights"],
    index=exchanges_list)

#DEFINE THE HIGH- AND LOW- QUALITY EXCHANGE RECORD
hq_exchanges_records = []
lq_exchanges_records = []

# CHECK IF THE SCORING WINDOW HAVE OVERFLOW
#IF YES, KILL THE FIRST RECORD
while moment_indx + 1 > n_scoring_window:
    scoring_window.loc[hq_exchanges_records]["HQ Count"] -= 1
    scoring_window.loc[lq_exchanges_records]["LQ Count"] -= 1

# UPDATE THE SCORING WINDOW
scoring_window.loc[hq_exchanges]["HQ Count"] += 1
scoring_window.loc[lq_exchanges]["LQ Count"] += 1

# UPDATE THE WEIGHTS
scoring_window.["Weights"] += scoring_window["HQ Count"] /
            (scoring_window["HQ Count"] + scoring_window["LQ Count"])

# RECORED THE PERFORMANCE
hq_exchanges_records.append(hq_exchanges)
lq_exchanges_records.append(lq_exchanges)
```

During the research, it was found that the difference between weights 0 and 1 was not significant enough for exchanges that were consistently clustered as high-quality. Therefore, a Fourier Amplifier was introduced to enhance the difference among weights, allowing high-quality exchanges to have more significant weights.

The Fourier Amplifier first transforms the weights from the time domain to the frequency domain, multiplies the weights in the frequency domain, and then converts them back to the time domain. After obtaining the amplified weights and high-quality prices for each moment, the weighted average can be calculated to determine the index price for that moment. This can be achieve by the following Python code:

```
def fourier_weight_amplifier(weights):
    # TRANSFORM THE WEIGHTS INTO SIGNALS
    fourier_coeffs = np.fft.fft(weights)

    # AMPLIFY THE SIGNAL BY MULTIPLYING BY A CONSTANT GREATER THAN 1
    amplification_factor = np.pi
    amplified_fourier_coeffs = fourier_coeffs * amplification_factor

    # INVERSE TRANSFORM THE SIGNALS INTO WEIGHTS
    amplified_signal = np.fft.ifft(amplified_fourier_coeffs)

    return amplified_signal.real
```

By so doing, we can amplified the difference between the weights and thus make those exchanges which have constantly being clustered as high-quality enjoy more weights when obtaining the index price as shown below in the Figure 13.

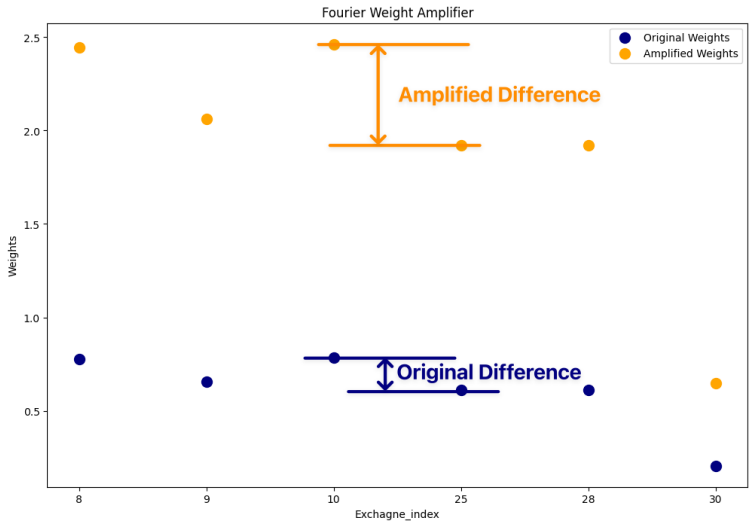


Figure 13: Fourier Amplifier Demonstration

After we get the amplified weights and the high-quality prices of this moment, we can then return the weighted average as the index price of this moment.

$$Index\ Price = \frac{High - Quality\ Prices \times Amplifed\ Weights}{\sum Amplifed\ Weights} \tag{12}$$

Following Python code return the index price of this moment by taking high-quality prices and amplified weights into account:

```
curr_index_prices = np.dot(hq_prices,
    scoring_window .loc[ hq_exchanges ]["Weights"])
    / np.sum(scoring_window .loc[ hq_exchanges ]["Weights"])
```

With implementing such calculation on all of the inputs, we can then achieve the index price path as shown below in the Figure 14.

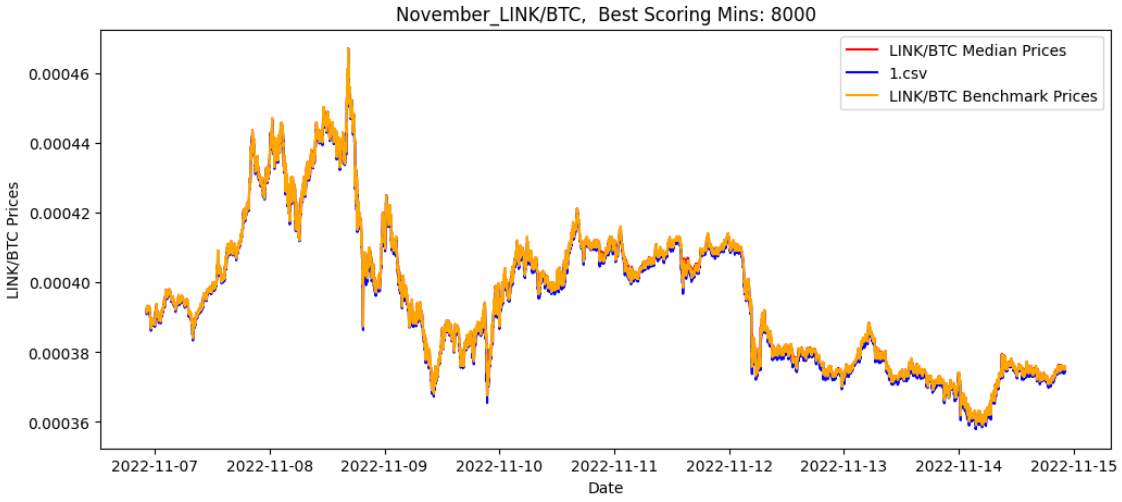


Figure 14: LINK/BTC Index Price Path

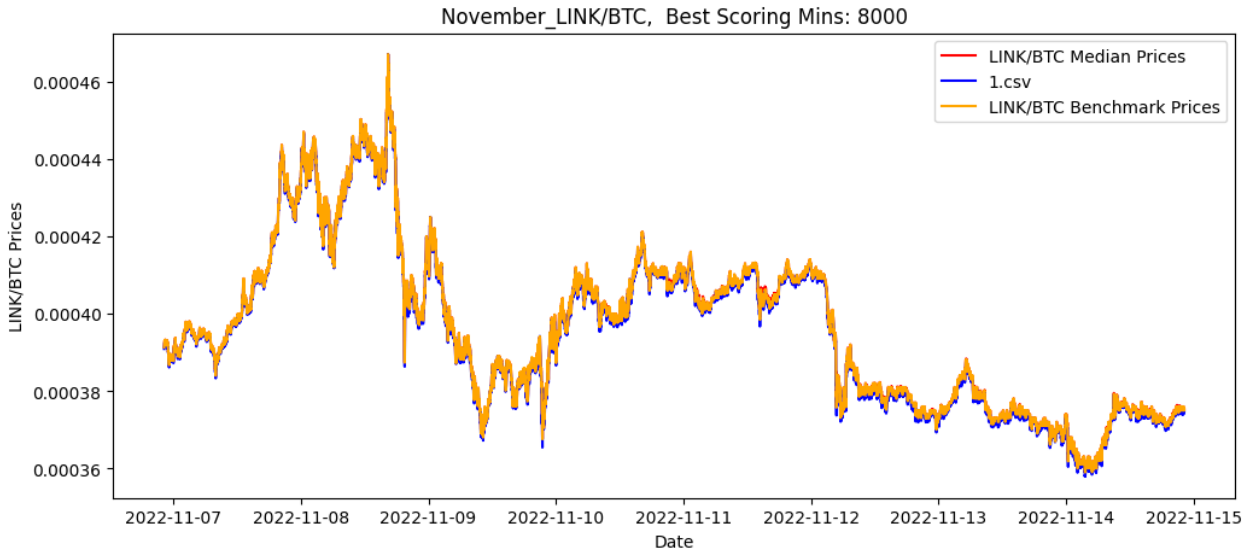
4 Numerical Results

In this section, we will display the numerical results of our experiment for validating its effectiveness. We will first describe the individual performance of specific Crypto pairs, and eventually move on to the inspection of overall performance of our model.

4.1 Individual Performance

For demonstrating the individual performance, we are to test our model on LINK/BTC pair. The time horizon of our data set will be from November 7th to November 15th of last year, and we are to test on minute-interval data of this period of time.

Using $1.csv$ as the target since it is the most reliable exchange, our model can generate the index price which can provide the 99.85% of R^2, which is better than simply taking the median of the all prices from various exchanges.



Index	Result	Result of Median
R square	0.9985967131772127	0.9984200086515317
MSE	6.129161649726366e-13	6.893296477282621e-13
MAPE	0.0017944405855733457	0.001898861380051704

Figure 15: LINK/BTC Index Price

We can as well look into other features, such as MSE or MAPE. We can obtain the $6.12916 \times e^{-13}$ on MSE and 0.00179 on MAPE, which all of those have better performance compared to simply taking the median prices from various exchanges.

From this individual result, we can see that our model is capable in detecting the anomalies in order to have a harmless input, and generate the index price that can approximate the theoretical price that a Crypto pair should have.

4.2 Overall Performance

We totally have the price data of 120 pairs with the time horizon from November 7th to November 15th of last year to test the performance of our model. And after we tested on all the pairs we have, our model can have better performance compared to median prices on over 83% of all the Crypto pairs. Besides, our model can generate the R^2 over 99% on over 79% of all the Crypto pairs.

To make sure that out model is not over-fitting to the November data, we have as well tested on the price data which ranged from October 7th to October 15th last year, to see if our model has over-fitting to

November data. In here, our model can have better performance compared to median prices on over 80% of all the Crypto pairs, and over 79% of pairs have R^2 over 99%.

From these results above, we can see that, in most of the cases, not only our model can provide even accurate result compared to simple taking the median of all prices from different exchanges, but as well our model can generate the index price that have R^2 over 99% on most of the pairs. We can as well notice that this model will not over-fitting to specific period of time, but can generate the accurate results on any time we implement on.

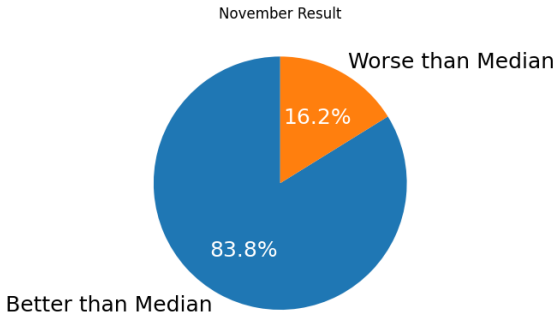


Figure 16: Performance of Index Price winning Median on R^2 on November Prices

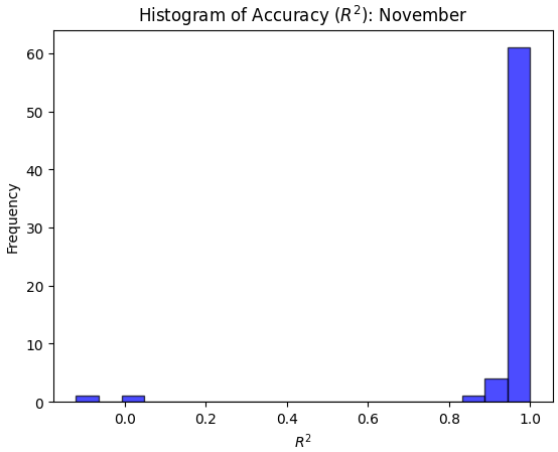


Figure 17: R^2 of November Index Price

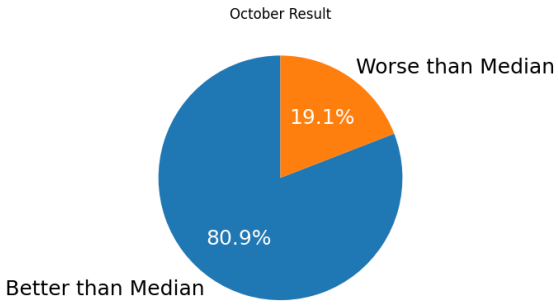


Figure 18: Performance of Index Price winning Median on R^2 on October Prices

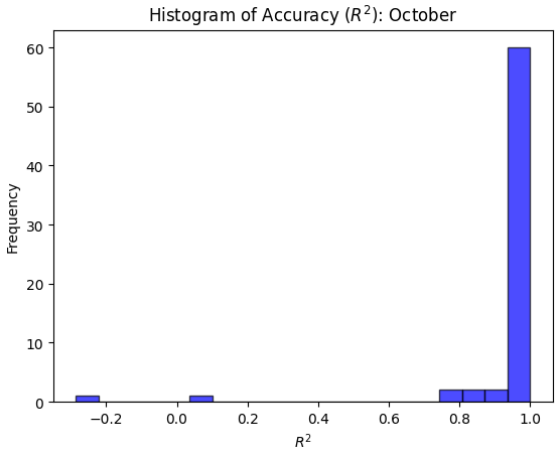


Figure 19: R^2 of October Index Price

5 Discussion and conclusion

In conclusion, this research project has successfully developed a cross-exchange dynamic pricing model for the Cryptocurrency market. Our model effectively addresses the problem of cross-exchange pricing and anomaly detection, providing an accurate index price for each Cryptocurrency pair by leveraging multiple exchange data sources. The dynamic nature of our model allows it to adapt to varying market conditions and input data quality, ensuring that the generated index price accurately reflects the true value of each trading pair.

Our proposed methodology consists of three main steps: 1) Anomaly detection for input pre-processing, 2) Model selection based on variance check, and 3) Index price calculation. By identifying and excluding critical anomalies such as missing values or stale quotes, our model ensures that only high-quality, relevant data points are used as input. The model selection step dynamically chooses the most suitable pricing model based on variance-check criteria, allowing our model to adapt to different market situations. Finally, the index price calculation step generates an accurate index price for each Cryptocurrency pair using the chosen pricing model.

To validate our model's effectiveness, we tested it on minute-interval order data for the LINK/BTC pair obtained from ten different exchanges provided by Tower Research. Our model achieved an R^2 value of 99.85%, a MSE of $6.1291 \times e^{-13}$, and a MAPE of 0.00179 on the data ranging from November 7^{th} to November 15^{th}, demonstrating its ability to replicate a high-quality exchange data feed with high accuracy. Furthermore, our research extended to testing the model on minute-interval price data of 120 different Cryptocurrency pairs, with 79% of the pairs exhibiting R^2 values above 99%. These results indicate that our model is effective for the majority of trading pairs in the Cryptocurrency market.

To further assess the robustness of our model and ensure it does not overfit the data, we tested it on October data, ranging from October 7^{th} to October 15^{th} of the previous year. In this evaluation, our model outperformed the median prices in over 80% of all cryptocurrency pairs, and over 79% of pairs achieved an R^2 value greater than 99%.

Our dynamic pricing model offers several advantages over static models. Firstly, it adapts to different market conditions and input data quality, generating more accurate results compared to simply taking the median of all prices. Secondly, our model effectively detects anomalies, ensuring that only useful and relevant data are used as input. This approach minimizes the impact of noisy or unreliable data points on the generated index price. Lastly, our model reduces time consumption by dynamically deciding whether to process the Clustering Pricing Algorithm based on the situation of each moment.

In summary, our research has successfully developed a dynamic, adaptive pricing model for the Cryptocurrency market that addresses the challenges of cross-exchange pricing and anomaly detection. By leveraging multiple exchange data sources and applying a robust methodology, our model provides accurate index prices for various Cryptocurrency pairs, thereby reducing the risks and maximizing the opportunities for traders and market participants. With the potential for further refinement and application to other financial markets, this research contributes to the ongoing development of innovative pricing models and strategies in the rapidly evolving world of finance.