2023-2024 FYP

# TRADING VOLUME PREDICTION

Wang Wenxin

DEPARTMENT OF INDUSTRIAL SYSTEMS

ENGINEERING AND MANAGEMENT NATIONAL

UNIVERSITY OF SINGAPORE

AY2023-2024

# TRADING VOLUME PREDICTION

Submitted by Wang Wenxin

Department of Industrial Systems Engineering and Management

In partial fulfillment of the requirements for the Degree of
Bachelor of Engineering National University of Singapore

AY2023-2024

# Summary

To meet the algorithm trading strategy's demand for QQQ ETF high-precision intraday trading volume, this paper introduces the time-feature optimization method and the CEEMDAN decomposition and reconstruction method, combined with the LSTM model. It helps to investigate the feasibility and optimization direction of QQQ intra-day trading volume prediction.

QQQ trading volume data has the features of high volatility, noise, and periodicity. In the direction of time feature optimization, the study uses Label Encoding, Cyclical Encoding, and Radial Basis Function to fit the intraday and intraweekly characteristics of trading volume data. In the direction of CEEMDAN, the research decomposes and refactors trading volume to produce a more simple volatility sub-sequence. After verification, the prediction model is reconstructed and the noise in the high-frequency sequence is filtered. Finally, the entire prediction value is calculated by integrating all of the sequences. This strategy extracts the sub-series complicated fluctuation patterns, improving the model's ability to predict extreme values.

The comparison results of multiple established models show that: Trading volume data is more impacted by extreme events and macro-factors, and the cyclical time feature optimization of QQQ trading volume forecast is not very helpful. But in the other direction, CEEMDAN-LSTM optimization performs better than the current modeling techniques with lower prediction error and more accurate extreme value prediction. CEEMDAN method has a lot of potential and viability in the QQQ trading volume

prediction. Despite the positive outcomes of the research, further work needs to be done on the CEEMDAN noise control, the addition of an attention mechanism, the choice of LSTM model parameters, and the growth of the research market range.

# Acknowledgment

I want to express profound appreciation to my mentors, Dr. Yap Chee Meng and Dr. Nelson Chui from the National University of Singapore, along with Dr. Wang Jianli from the Nanjing University of Aeronautics and Astronautics. Their steadfast support and expert guidance have been paramount in completing this research. Their meticulous supervision and insightful feedback have significantly enhanced my problem-solving capabilities and fostered my ability to conduct independent research. I wouldn't have grown so much without their frequent group meetings and invaluable advice.

I am grateful to my parents for their unwavering material and spiritual support. Their encouragement in my explorations of new fields has provided me with a stable foundation upon which to grow. They possess profound knowledge derived not only from books but also from rich life experiences, which have guided me through life's complexities. Their education to me, free from undue pressure, has always prioritized my happiness, for which I consider myself fortunate. I am deeply thankful for their consistent encouragement throughout my personal and academic development.

Special acknowledgment is also extended to my friends for their enduring support. I am particularly indebted to Liu Ze, whose insightful recommendations on my dissertation have profoundly deepened my understanding of the CEEMDAN method within financial market analysis and greatly enhanced the refinement of the model. I am grateful to my long-standing friend, Zuo Yue, with whom I have shared nearly a decade of mutual encouragement. Additionally, I express gratitude to my peers at the NUAA Musical

Instrument Association, and Swim Association, and participants of program 311. Together, we have navigated the challenges of an adverse economic environment, a competitive job market, and considerable academic pressures. I hope we can spread a reckless sail to the wind and ride the turbulent water in our lifetime.

# Contents

# List of Figures



# List of Tables

# 1 Introduction

## 1.1 Aim of the Research

Since S&P introduced depositary receipts in 1993, exchange-traded funds (ETFs), particularly the QQQ ETF, which tracks the NASDAQ-100 Index, have developed as important financial tools. QQQ comprises the top 100 market capitalization corporations, primarily focused on technology and non-financial fields, which can powerfully reflect the emerging sector, the real economy, and the forefront of market trends.ETFs offer benefits such as daily trading, low fee structures, and precise tracking of their underlying indices using a stock basket for redemption operations. As a result, ETFs play an essential role in investment and asset allocation by covering both the primary and secondary markets. This has cemented ETFs as one of the most important financial instruments of the twentieth century.

Therefore, accurate forecasting of ETF trading volume can help us make better use of this financial tool, particularly in the area of algorithmic trading, where daily transaction volumes are crucial for optimizing position sizing, detecting trading abnormalities, and maximizing profits. Berkowitz, Logue, and Noser (1988) created the volume-weighted average price (VWAP) technique, which is an excellent example of an algorithm that uses expected transaction volumes based on the day's trading activity structure that is still in use today.

Thus, reliable volume forecasting is required not only for the proper execution of such techniques, but also to improve the overall structural analysis of intraday trading, measure market volatility, manage liquidity concerns, and drive financial market innovation. It is a critical measure for analyzing market trends and making strategic trading decisions. Furthermore, reliable volume forecasting is critical for reducing intraday risk and improving the regulatory oversight of market activity.

Current forecasting models often suffer from significant limitations due to their heavy reliance on historical data and the intrinsic U-shaped pattern of intraday trading volumes. However, the advent of deep learning technologies, particularly the integration of Long Short-Term Memory (LSTM) networks with frequency decomposition methods or precise time features capture, presents a novel approach to high-frequency financial data predictions. However, this method has very few applications in the financial field, especially in the field of transaction volume prediction, highlighting a critical gap in financial market analytics.

## 1.2 Research Methodology

This study examines the NASDAQ 100 ETF (QQQ) by analyzing 24,786 pieces of 15-minute high-frequency trading volume and price data from January 2, 2020, to August 31, 2023.

The feature engineering stage mainly uses two methods. The study focused on the temporal features created by applying methods like Cyclical Encoding with Label encoding, Sine/Cosine, and Radial Basis Function to capture the cyclical characteristics

of trading volumes during various weekdays and hours. The second study utilized the Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) to decompose the original data into multiple features rather than analyzing the features based on the modeler's experience.

For prediction, the study employs an LSTM model enhanced by multiple time feature vectors and CEEMDAN features. Model parameters were optimized using a GridSearchCV, and performance was evaluated using Mean Absolute Error metrics. This study allows for the comparison of feature engineering techniques and the selection of an optimal model, aiming to improve the prediction of market volatility and enhance the performance of trading algorithms, thereby supporting investors and managers in making timely and accurate trading decisions.

# 2 Literature Review

Trading volume prediction is a critical domain within financial time series analysis. With the development of deep learning, Long Short-Term Memory (LSTM) networks are becoming more accurate in modeling high-frequency financial data with non-linear and irregular patterns. However, studies optimizing mixed model projections for ETF trading volumes -- especially for the QQQ -- remain scarce, despite the growing body of research focusing on stock price predictions.

Thus, this literature review will focus on two topics: the current methods for financial time series forecasting and the strategies for enhancing model optimization for ETF volume prediction.

## 2.1 Financial Time Series Forecasting Models

Predicting intra-day trading volume is the same as financial time series forecasting with clearly U-shaped patterns and time-varying structural fluctuations, which could use both linear and non-linear methodologies. Studies have shown that the predictive accuracy of non-linear algorithms surpasses that of linear algorithms (Behrendt, S. & Schmidt, A., 2021). As a result, non-linear techniques such as neural network models, regression trees, and support vector machines (SVM) (Huang, Nakamori, & Wang, 2005) are mostly used in this area in recent years.

In the field of finance, one of the most popular neural network models is the Long Short-Term Memory (LSTM) model (Graves, 2013), which is an improvement on the Recurrent Neural Network (RNN). Numerous research could prove that LSTM is better than others. Fu, X. (2019) applied the LSTM model incorporating investor sentiment influence factors to study P2P market transaction volumes, demonstrating its superiority over baseline models SVM. Furthermore, Thomas et al. (2018) found that LSTM significantly performed better than random forest and logistic regression models in predicting non-linear financial time series. Despite these advancements, research focused on optimizing the LSTM model for predicting intra-day high-frequency trading volumes of ETFs remains sparse. This study, therefore, concentrates on the optimization of LSTM models for accurate forecasting of QQQ trading volumes, addressing a gap in the current financial time series forecasting literature.

## 2.2 Optimization Strategies for Financial Forecasting Models

In machine learning prediction models, improving the diversity of features during the feature engineering stage is a common approach to improve prediction precision. Factors influencing intra-day trading volume include historical data, stock prices, and other uncertainty factors like policies, and feature selection comes from the capture of these factor features. Traditional predictions of intraday trading volume have typically concentrated on the influence of prices or their trading volumes during nearby times. Expanding feature diversity can greatly increase the accuracy of existing forecasting methods. However, determining the appropriate traits is a very challenging issue.

The intra-day trading volumes in financial markets demonstrate a pronounced U-shaped pattern (Jain & Joh, 1988), also identified by McInish and Wood (1992), with peaks typically occurring at the market's open and close. This periodicity suggests a potential for enhancing model precision by incorporating cyclical trends of historical trading volume data as features. There are three methods for incorporating temporal features are prevalent: setting Label variables, cyclical encoding, and the Radial Basis Function (RBF) for coding (Kourentzes & Crone, 2010).

Furthermore, compared to a single model approach, building a hybrid model can greatly increase forecast accuracy. Integration methods have often yielded optimal outcomes in existing applications of LSTM models to forecast financial market data. Libman D et al. (2019) demonstrated that LSTM models, when integrated into AR hybrid models,

contribute to more accurate trading volume forecasts. Ghosh et al. (2022) observed that a combination of random forests and LSTM networks for predicting intra-day prices and stock selection enhances returns.

Among the various hybrid forecasting techniques, the decomposition-integration method has gained popularity in recent years. It might be regarded as a feature engineering method as well. This method involves decomposing the original time series into several components, forecasting each separately, and then integrating the results. Its effectiveness is evidenced by its wide application across diverse fields such as agriculture (Huang, et al.,2023), wind energy (Liu Y, et al.,2023), carbon pricing, and financial asset returns(Torres, et al.,2023). The decomposition-integration method is preferred over multi-model hybrid methods in different demands. due to its ability to construct predictive factors more straightforwardly, reduce computation time, and decreae noise influence which is suitable in financial markets.

Common time series decomposition techniques include Wavelet Transform (Daubechies I, 1992), Empirical Mode Decomposition (EMD) (N.E. Huang,1971), Ensemble Empirical Mode Decomposition (EEMD)(Wu & Huang, 2005), Multivariate Empirical Mode Decomposition (MEMD) (Rehman & Mandic, 2010), and Adaptive Noise Complete Ensemble Empirical Mode Decomposition (CEEMDAN) (M.E. Torres et al.,2011). HE Yi-yue et al. (2022) reduced prediction errors compared to SVR, MLP, and traditional LSTM models by enhancing the feature contribution of a MEMD-LSTM model using an attention mechanism combined with feature vector decomposition. Cao et al.(2019) found that combining the CEEMDAN signal decomposition algorithm with

the LSTM model to predict closing prices of major global stock indices outperformed fundamental models. Wang et al. established the CEEMDAN-SE-LSTM-RF model to predict carbon prices, showing better performance than four benchmark methods with smaller statistical errors. These studies highlight the advantages of the decomposition-integration method in financial market predictions and the effectiveness of the CEEMDAN method. However, there is still a deficiency in the literature about the application of the decomposition-integration approach in trading volume prediction, which makes my study exceptionally innovative.

## 2.3 Conclusion

The LSTM neural network is superior to other Nonlinear Forecasting Models in the precision of forecasting financial time series and dealing with long time series forecasting. In addition, there is a vacuum of research on the best way to use the LSTM model to predict the intraday high-frequency trading activity of ETFs. Hence, my study chooses the LSTM model to forecast the QQQ ETF trading volume and then optimizes the model on this information.

There are two approaches to optimizing the model: one is to increase the number of feature vectors, and the other is to use a hybrid model to increase the complexity of the model. In feature vector optimization, QQQ ETF has a special U-shaped structure, and the selection of the time feature vector has a great influence on the prediction result. Previous studies used time as an index directly, which made it difficult for the algorithm to learn. Since my research will investigate how to capture the time features more effectively.

However, adding the feature vector mainly through market and research experience makes it easier to ignore the important variables due to the complexity of selecting model characteristics. The EMD method can decompose the original variables effectively to get the features. The CEEMDAN model makes further work that can effectively solve the problem of white noise transfer from high frequency to low frequency, guaranteeing the accuracy and integrity of the reconstruction sequence and increasing efficiency. As one of the hybrid models, my research will combine the CEEMDAN method with the LSTM model to optimize the existing trading volume forecast.

# 3 Methodology

## 3.1 Encoding Time Information as Features

Identifying more meaningful features from the data can often make the most improvement to model accuracy for the least amount of effort. So, I will introduce three methods of assessment time features in this chapter.

### 3.1.1 Label Encoding

Label Encoding is a data preprocessing technique used in machine learning to transform categorical variables into numerical values. This method helps the algorithm classify non-numerical categories as numerical ones. Label Encoding assigns a unique integer to each category. The process is sequential and generally starts with zero: the first category is labeled as 0, the second as 1, and so forth. However, this method may add an ordinal dimension that the original categorical variables do not possess, leading to biased model outcomes.

### 3.1.2 Cyclical Encoding with Sine/Cosine Transformation

Cyclical encoding is used in data preprocessing to handle periodic features. This method is particularly relevant for variables such as time of day, and day of the week, where label encoding can not represent the relationship between the end and the beginning of the cycle.

$$\sin\left(\frac{2\pi h}{H}\right), \cos\left(\frac{2\pi h}{H}\right) \tag{3.1}$$

In cyclical encoding, each cyclical feature is split into two features using sine and cosine. This transformation maps the data onto a circle, ensuring the cyclical nature and shrinking the gap between the end and the start of the cycle.

### 3.1.3 Radial Basis Functions

The Radial Basis Function (RBF) is a nonlinear transformation method prevalent in neural networks. This function is used in time series analysis which could construct time variables interactions. It computes the Euclidean distance between an input variable and a designated central point and then maps this distance through the function to achieve feature transformation. Using RBF to encode time variables enables the extraction of latent patterns and complex periodicities within time series datasets.

$$\phi(x) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3.2}$$

$x$ is the time variable, $\mu$ and $\sigma$ are the center and width parameters of RBF.

Adopting time variable encoding methods can suit various analytical needs and scenarios. In practical applications, the strategic choice of an encoding method may

significantly improve both the performance and accuracy of time series predictive models.

## 3.2 CEEMDAN Decomposition and Refactoring

Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) is an improvement of the Ensemble Empirical Mode Decomposition (EEMD). This method enhances the decomposition of time series data by adaptively adjusting the dataset's noise level. CEEMDAN effectively reduces the problem of mode mixing, which occurs when different intrinsic mode functions are improperly combined or separated in the IMFs. This improves both the efficiency of the decomposition process and the stability of the decomposition.

### 3.2.1 The CEEMDAN Principle

To decompose the $ith\ IMF$, the original signal $X(t)$ is added with an adaptive white noise $\alpha \cdot \sigma_i \cdot n_k(t)$, establishing the total number of white noise $K$ and the intensity $\alpha$ of white noise.

$$X_{i,k}(t) = X(t) + \alpha \cdot \sigma_i \cdot n_k(t), \quad k = 1,2, \dots, K \tag{3.3}$$

$\sigma_i$ is the standard deviation of the current residual signal and $n_k(t)$ is the normal distribution noise, the $Kth$ decomposition is performed. Each noisy signal $X_{i,k}(t)$is decomposed to get the integrated $\{IMF_{i,k}(t)\}_{i=1}^{K}$

$$\overline{IMF}_i(t) = \frac{1}{K}\sum_{k=1}^{K} IMF_{i,k}(t) \tag{3.4}$$

Subtract the IMF mean from the signal, and the updated residual signal $R_i(t)$is used for the next IMF extraction.

$$R_i(t) = X(t) - \overline{IMF}_i(t) \tag{3.5}$$

The final signal $X(t)$ is expressed as:

$$X(t) = \sum_{i=1}^{N} \overline{IMF}_i(t) + r(t) \tag{3.6}$$

CEEMDAN adjusts the noise level to ensure a more accurate and reliable decomposition, making it an essential tool in the quantitative analysis of complex financial systems.

## 3.2.2 The IMFS Refactoring Method

Reconstructing a time series signal S(t) involves taking the original IMFs and extracting the high-frequency, low-frequency, and trend-term components. There are three processes involved in this reconstruction:

Compute the mean $c_i(t)$ of each Intrinsic Mode Function (IMF):

$$c_i(t) = \frac{1}{T}\int_0^T c_i(t)\ dt, \quad i = 1,2,\dots,n \tag{3.7}$$

For each IMF $c_i(t)$ , do a test in sequence for the non-zero mean at a significance threshold of 0.05. Then, select the first test $x$ smaller than 0.05 as the divide for high frequency and low frequency.

$$t_{c_i} = \frac{\overline{c_i} - 0}{s/\sqrt{n}}, \quad i = 1,2,\dots,n \tag{3.8}$$

$c_i$ is the sample mean, $s$ is the sample standard deviation, and $n$ is the sample size.

Formation of sub-sequences:

$$\text{High frequency subsequence: } H(t) = \sum_{i=1}^{x-1} c_i(t) \tag{3.9}$$

$$\text{Low frequency subsequence: } \mathrm{L}(t) = \sum_{i=x}^{n} c_i(t) \tag{3.10}$$

$$\text{Trend component: } T(t) = r(t) \tag{3.11}$$

This structured method of signal reconstruction is critical for evaluating time series data since it can avoid model over-complexity, resulting in LSTM over-fitting.

## 3.3 LSTM Model

Long Short-Term Memory (LSTM) is an improved version of the Recurrent Neural Network (RNN) created to address the gradient vanishing and exploding issues when RNN training on long sequential data.

The LSTM cell is an essential part that consists of a memory cell state and three separate gates: the forget gate, the input gate, and the output gate. Each gate includes a sigmoidal neural network layer and point-wise multiplication operations, which gives good control over input flow within the network. And the cell state is a vector that changes over time steps, updated linearly in small steps, allowing valuable information to be preserved over lengthy periods. These gate mechanisms have greatly enhanced LSTM networks' capacity to recognize and understand long-range dependencies within sequential data. This increases their ability to manage challenging, long-term prediction problems. In previous research, LSTM has proved its usefulness in Financial Time Series forecasting by accurately predicting trends in stock prices, trade volumes, and other significant indicators.

### 3.3.1 LSTM Principle

Forget gate $f_t$ determines what information is kept or deleted:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \tag{3.12}$$

$\sigma$ is a sigmoid function that produces an output between 0 and 1 for gate control. $h_{t-1}$ is the hidden state of the previous time step. $W_f$ is the weight matrix. $b_f$ is the offset term, corresponding to the subscript of the weight matrix.

Input Gate $i_t$ controls the input of new information and determines what new information will be stored in the cell state:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.13}$$

$$\widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.14}$$

$W$ is the weight matrix, subscript $i$ is the forgetting gate, and subscript c is the cell candidate. Widetilde $\widetilde{C_t}$ is the cellular state, and tanh is the hyperbolic tangent activation function, which produces an output between -1 and 1 that is used to modulate nonlinearity in the network.

The new cell state $C_t$ is obtained by forgetting old information and adding new information:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{3.15}$$

$*$ represents the product by element (Hadamard product) .

Output Gate $o_t$ controls the flow of information from the cell state to the hidden state, determining what information will be output:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.16}$$

$$h_t = o_t * \tanh(C_t) \tag{3.17}$$

$W_o$ is the weight matrix, and subscript $o$ denotes the forgotten gate.

Figure 3.1 shows the basic structure of the LSTM neural network memory unit, replacing the state nodes in the RNN, which is the main difference from the RNN structure. By adopting gate units, LSTM can retain the nodes' properties for longer and better understand temporal dependence.

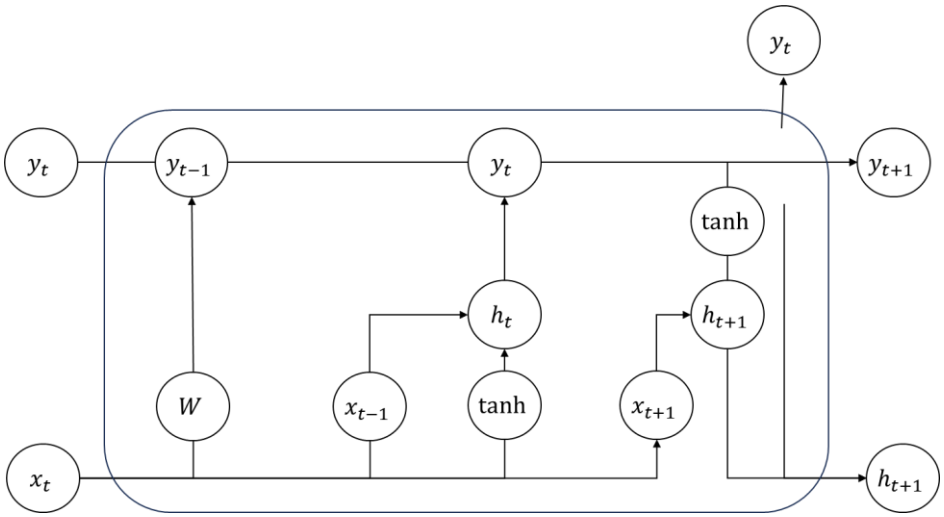



Figure 3.1 LSTM model predictive structure diagram

## 3.3.2 LSTM Evaluation Index

To comprehensively evaluate the performance of the LSTM model in financial time series forecasting, the study selects four model evaluation indicators to measure the difference between the predicted value and the actual value of the model: R-squared value ($R^2$), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Square Error (RMSE). These indexes could represent the model's capacity for generalization and prediction accuracy from various dimensions.

$R^2$:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.18}$$

$R^2$ can account for the proportion of variability in dependent variable y estimated by multiple regression equations. The closer the $R^2$ value is to 1, the more accurate the prediction is.

Mean absolute error (MAE) :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.19}$$

Mean Absolute Percentage Error (MAPE):

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\% \tag{3.20}$$

Root Mean Square Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.21}$$

Errors are represented by MAE, MAPE, and RMSE; the closer an error to zero, the more accurate the forecast.

# 4 Results and Discussion

## 4.1 QQQ Data Processing

### 4.1.1 Basic Processing

The study collected 24866 data points on the NASDAQ 100 ETF (QQQ)15-minute high-frequency trading volume and prices between January 2, 2020, and August 31, 2023. The data covers the hours of 9:30 a.m. to 4:00 p.m. Because 5 days in the data set could be a holiday or special activity data missing, this article chose to delete the 5 days of data, remaining 24786 data.

Then the study does the descriptive statistics of data. The dataset's distribution is 2.59, indicate its right-skewed. The mean trading volume is 1,449,008.0, which is much greater than the median of 1,100,175.5, showing the distribution's asymmetry. The dataset ranges from 5,000 to 16.39 million, demonstrating significant volatility in trading volumes. Furthermore, the coefficient of variation is 0.81, showing significant dispersion about the volume. The Skewness rating of 2.5948 indicates that the tail deviates to the right. The Kurtosis value of 10.7386 is significantly greater than 3, suggesting that the distribution of transaction volume data is steeper than the usual distribution. The QQQ volume dataset contains numerous exceptionally big values.

In this study, I predicted the intra-day trading volume dataset with a 15-minute interval, which is split into 6:2:2 samples as a training set, validate set, and test set. The data of the validation set is used to compare the prediction results of the model in training to

obtain the state of the model and avoid over-fitting so that the hyperparameters can be adjusted in time according to the training results of the validation set. The test set is used to show the ability of the final model prediction.

Based on the data, I plotted the daily trading volume trends between January 2, 2020, to August 31, 2023, demonstrating the volatility of market activity over four years. The figure shows that QQQ transaction volume is a dynamic, non-linear, non-stable, noisy, and chaotic system, corresponding to the disorder of the financial market and is suitable for forecasting by a neural network model.

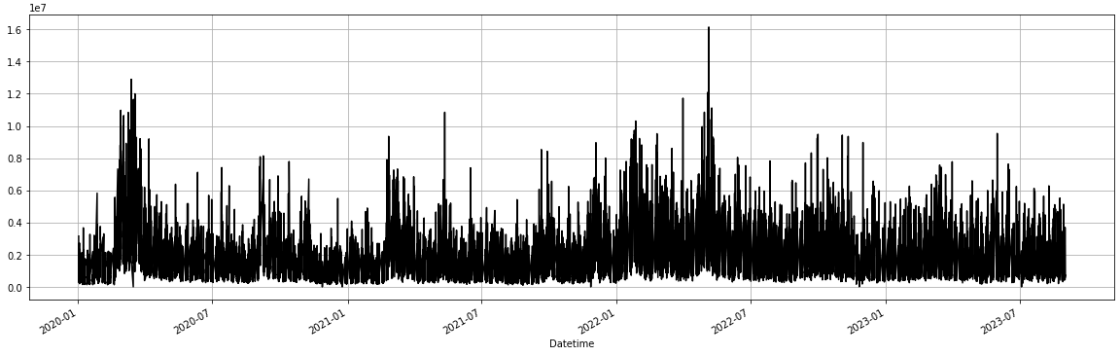



Figure 4.1 15 - min interval volume changes of QQQ

## 4.1.2 Intraday Trend in Trading Volume

In conducting the analysis, I averaged the minute-by-minute trading volumes within each day to construct a daily trading volume trend graph. This graphical revealed the classic U-shaped intraday pattern that trading volumes were seen to be very high at the opening and closure of the market, with a significant decrease in midday.

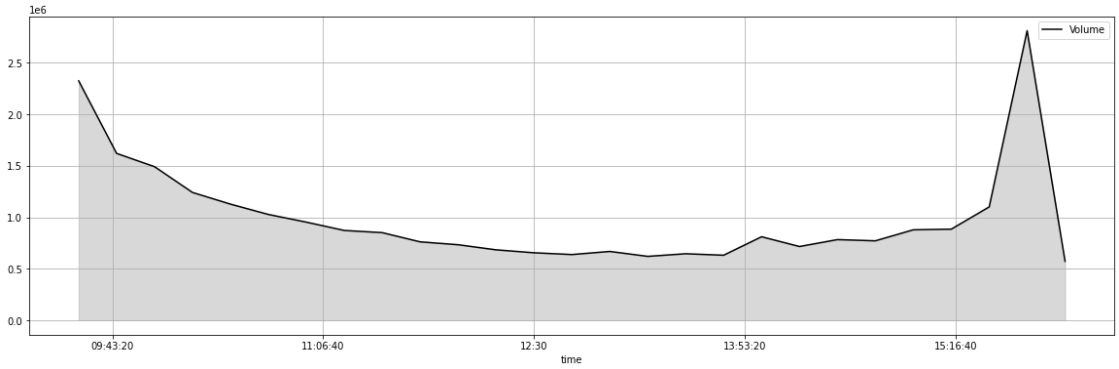



Figure 4.2 Intraday volume changes of QQQ

The overnight build-up of information and the resulting trading demand are responsible for the increased trade volume at the market opening. As the time approaches midday, a slow drop in volume is usually observed. Volumes then gradually rise, and near the conclusion of the day, a noticeable increase is observed which is associated with investors adjusting their positions to mitigate potential risks. This character shows the time feature should be attention during the feature engineering stage.

## 4.1.3 Weekday Trend in Trading Volume

In financial markets, trading volumes exhibit unique patterns on weekdays and may also be observed in QQQ ETFs. So I plot its average weekday volume.

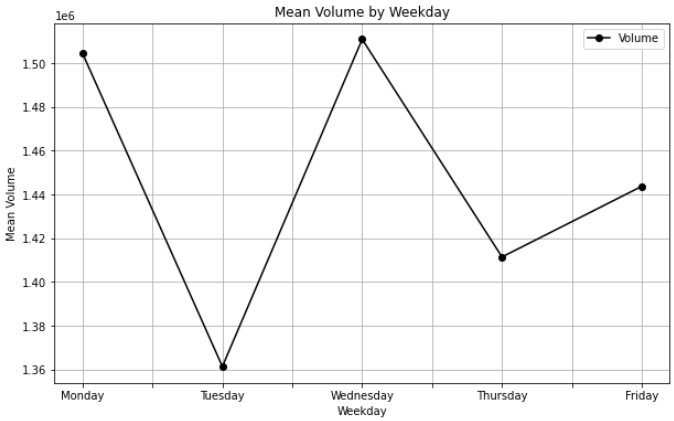



Figure 4.3 Weekday volume changes of QQQ

As Figure 4.3 shows, elevated trading volumes on Mondays, Wednesdays, and Fridays can be attributed to systemic and behavioral factors. Trading volumes rise on Monday due to the accumulation of news and data over the weekend and traders assimilate and act upon it. And Wednesday sees high trading volumes often driven by the release of important economic indicators, central bank announcements, and Federal Open Market Committee minutes. The information prompts traders to adjust their positions based on the new data, leading to increased trading activity. Additionally, Friday trading volumes are higher which is similar to the conclusion of the day as traders adjust their positions to avoid holding risks at close times. These features should also be considered in feature engineering.

### 4.1.4 Return Rate of QQQ

Price fluctuations are directly correlated with changes in trading volume. Based on the open and close prices every 15 minutes, I calculate the logarithmic yield, which reflects the level of gains or losses investors have made over the holding period.

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) \tag{4.1}$$

$r_t$ is the return rate at time $t$

## 4.2 Encoding Time Features of QQQ

In the case of the QQQ ETF, encoding is employed to quantify temporal patterns within the data by capturing the properties of the five weekdays and the 15-minute time (27

intervals). Thus to help the LSTM model learn the intraweek dynamics and the "U-shaped" intraday trend better.

Table 4.1 summarizes the evaluation results of various predictive modeling methods applied to the QQQ ETF data. The table compares the outcomes based on the number of columns increased and the features captured by each model.

Table 4.1 Encoding Time Features Summary

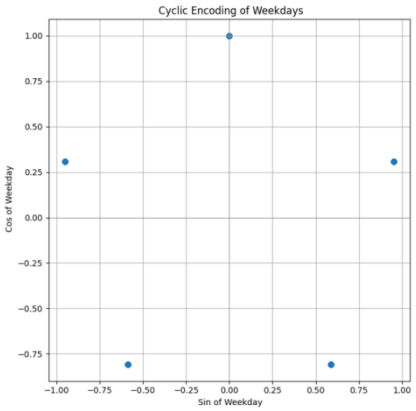
| Model | Columns Increased | Features Capture |
|---|---|---|
| Label Encoding | 2 | |
| Cyclical Encoding | 4 | Intra-day & Intra-week |
| Radial Basis Function | 5 | |

### 4.2.1 Label Encoding

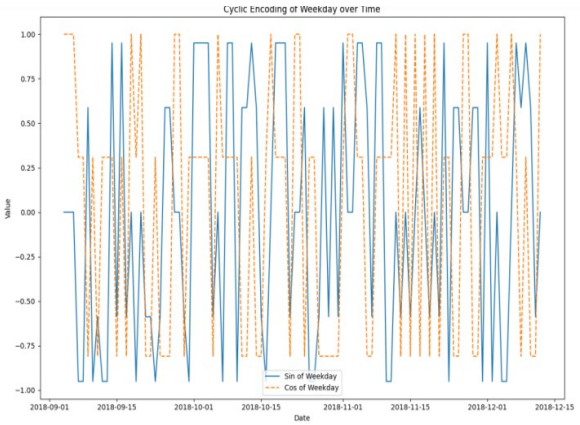
The study assigns label feature codes to each weekday (ranging from 1 to 5) and to every time interval within a day (ranging from 1 to 27).

### 4.2.2 Cyclical Encoding

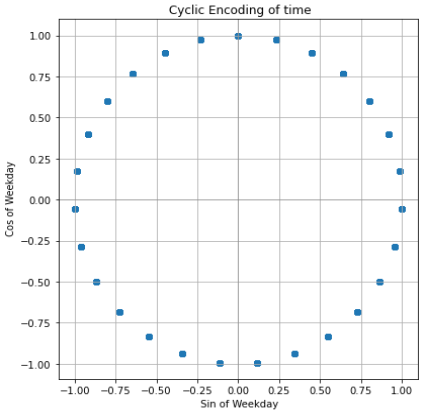
Figure 4.4 shows the cyclical encoding of weekdays and time. The left panel illustrates the day-to-day and time-to-time cyclical encoding patterns; the right panel plots the cyclically encoded values against the date. The sinusoidal function is depicted by blue solid lines, while the cosine function is indicated by orange dotted lines.



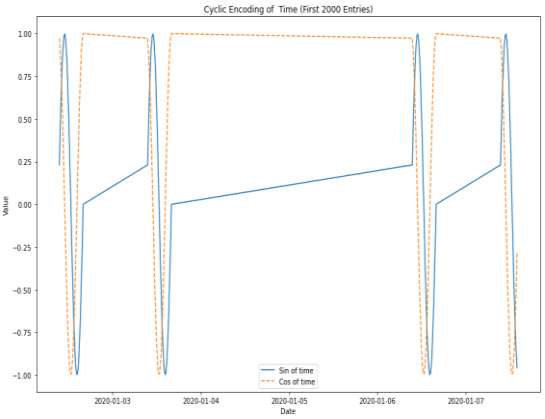

（a）Cyclical Encoding Weekday in circle




（b）Cyclical Encoding Weekday in time series




（c）Cyclical Encoding time in circle

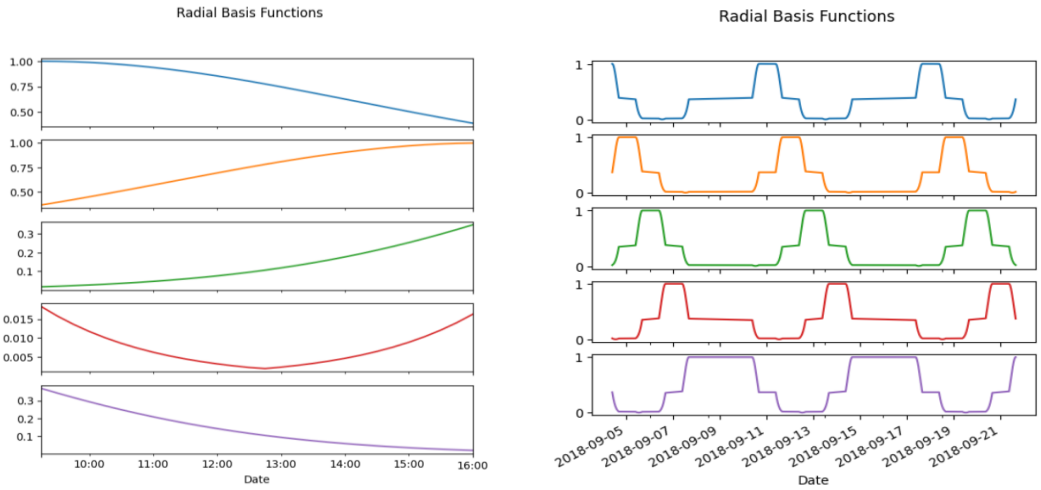



（d）Cyclical Encoding time in time series

Figure 4.4 Cyclical Encoding of Weekday and Time

## 4.2.3 Radial Basis Function Encoding

In Figure 4.5, the left panel depicts the RBF output corresponding to various intraday time points; the right panel illustrates the vibration of the RBF encoded features over a sequence of days. This method captures temporal trading dynamics for more details.




（a）Radial Basis Function of a single day （b）Radial Basis Function based on time

Figure 4.5 Radial Basis Function

## 4.3 CEEMDAN of QQQ

This chapter's study primarily focuses on the decomposition and refactoring of the CEEMDAN approach, which has a significant impact on the accuracy of the LSTM hybrid model.

### 4.3.1 CEEMDAN Decomposition of QQQ

Figure 4.6 depicts the CEEMDAN algorithm decomposing the data into 16 sequences arranged from top to bottom. The intrinsic mode components (IMFs), which comprise the first 15 of these sequences, are simpler than the original sequence with gradually decreasing fluctuation frequencies.

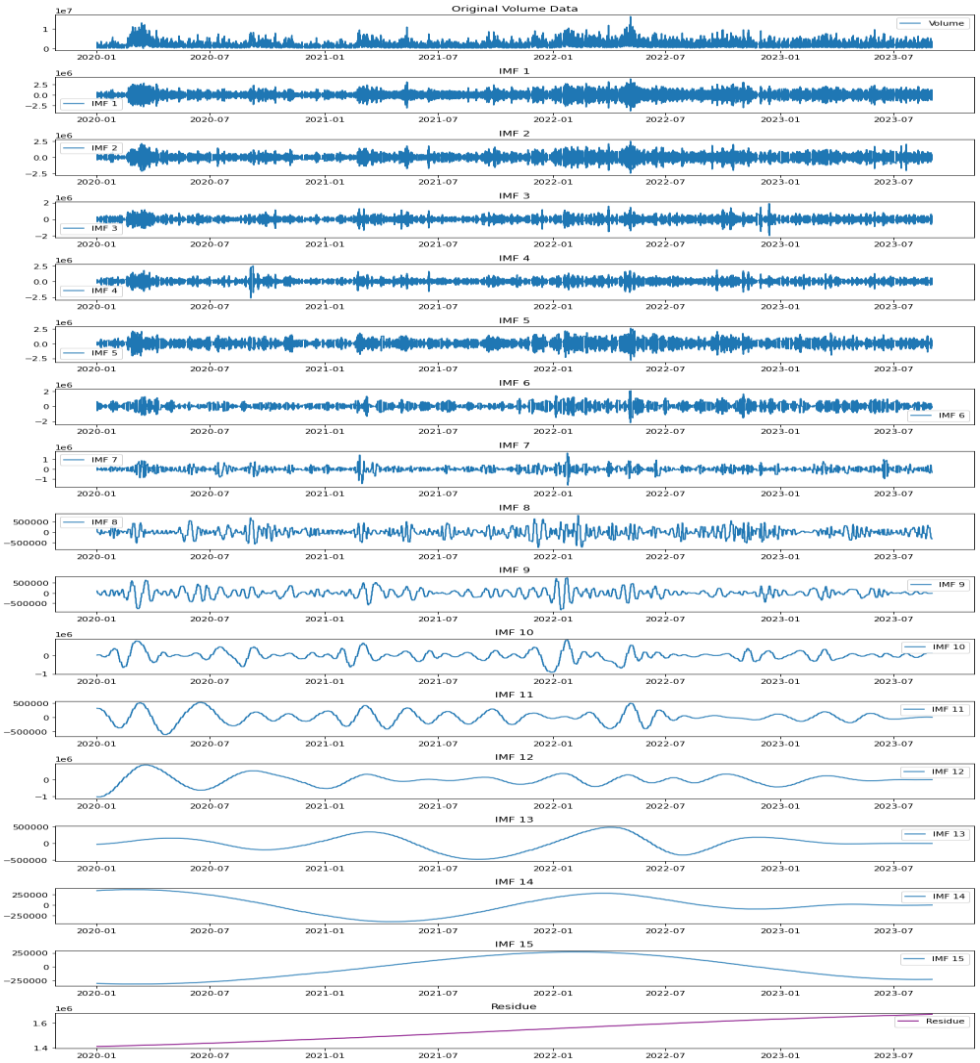



Figure 4.6 CEEMDAN Decomposition

High-frequency IMFs capture short-term swings or noise in data, which may better reflect market reactions to real-time information or intraday trading volatility. It is also the most difficult aspect of understanding the law and predicting the volume. Low-frequency IMFs swing slowly, reflecting long-term trends or cyclical fluctuations in trade activity. These components could be linked to business cycles, seasonal influences, or macroeconomic policy changes. The residue can show the basic trend of trading volume, which is a steady upward curve, reflecting the steady development of QQQ as a portfolio of technology pioneer shares.

Since too much decomposition in this stage could influence the LSTM model's efficacy and learning state, we must recreate IMFs.

### 4.3.2 CEEMDAN Refactoring of QQQ

Following the application of the CEEMDAN refactoring method to IMFs, Figure 4.6 represents the results of the decomposition process, each of which captures different frequency components and trends of the data. The variation patterns of the sub-sequences are relatively simple and regular, which makes it easy to further extracting the fluctuation features of each sub-sequence

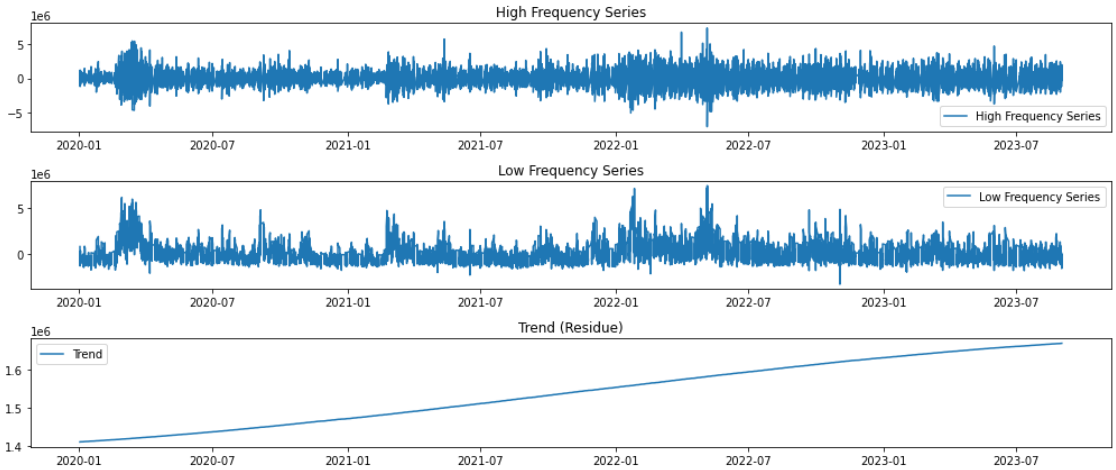



Figure 4.7 CEEMDAN Refactoring

Collectively, the procedure makes it easier to comprehend the financial time series' inherent oscillatory characteristics and hierarchical structure, which supports subsequent analysis and predictive modeling initiatives.

## 4.4 LSTM prediction of QQQ

In this chapter, the corresponding LSTM prediction models are constructed for three trading volume data sets and three component sub-sequences formed by CEEMDAN

decomposition and recombination. Then, the model is used to make rolling predictions for each sequence and data set in the prediction interval. The predictive performance of each model is evaluated and compared using four metrics: $R^2$, MAE, MAPE, and RMSE.

The model is structured as follows. The rolling prediction model used in the study has a window size of 56. The original 24,786 time series data were split into 6:2:2, and represented by the DataFrame object in the Python Pandas library. The training set has 14,816 rows (24,786 × 0.6 - 56 + 1) and 56 × feature number columns, while the verification and test sets have 4902 rows (24786 × 0.2 - 56 + 1) and 56 × feature number columns. Next, create the relevant LSTM prediction model in turn.

Some adjustments are applied to the neural network so as to optimize it, considering into account the intricate features of the financial time series and the computing efficiency of the model. During the modeling process, the data were normalized using the Min-Max Scaler to eliminate dimensional effects. The parameter range is normally adjusted by Optuna Grid search, and it is further fine-tuned using GridSearchCV. There are two options for batch size: 16 or 32; and for epochs: 50 or 100. The stacked LSTM has two hidden layers and the dropout layer has a 0.2 drop rate. Adam (Adaptive Moment Estimation) is chosen as the optimizer, and the learning rate is set to 0.001 to find the global minimum of the loss function.

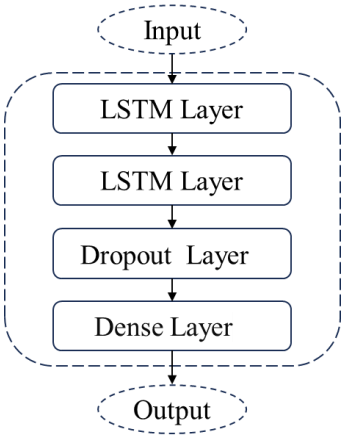


Figure 4.8 Simplified LSTM Neural Network Architecture

### 4.4.1 LSTM with Time Features

Table 4.2 displays the results of the multi-criteria evaluation of the LSTM prediction model under various combinations of temporal features. The optimal combination of LSTM is the label encoding. Label-LSTM is slightly better than Basic-LSTM and the others are worse, suggesting that the label information has not significantly improved to increase forecast accuracy. The Label-LSTM model can only explain about 70.99% of QQQ transaction volume changes and RMSE and Mae values are quite high, showing an extensive average error between predicted and real values, which demonstrates the model's limited capacity to capture the volatility of trading volume.

Table 4.2 Evaluation results of different time features

| | Model | $R^2$ | RMSE | MAE | MAPE |
|---|---|---|---|---|---|
| ★★ | Basic-LSTM | 0.7048 | 535,549.78 | 315,784.59 | 21.79% |
| ★ | Label-LSTM | 0.7099 | 530,887.44 | 308,894.37 | 21.32% |
| | Cyclical-LSTM | 0.7026 | 537,528.06 | 311,943.99 | 21.53% |
| | Radial-LSTM | 0.6801 | 557,502.94 | 320,309.95 | 22.11% |

Overall, since the trading volume used the time index in the learning process, the three types of time coding only provide cyclical qualities, which are insufficient. This shows that although the trading volume shows an obvious cyclical signal, it is not the primary factor. Macro factors on QQQ have a greater impact, such as market mood, macroeconomic data, and emergencies, thus we should pay more attention to them. Figure 4.8 depicts the results of various models compared to the true value. The Blue Line shows the basic model without a time signal, which estimates the extremum better than other models. Instead, periodic signals introduce noise that disrupts the LSTM model's ability to learn about extremes.

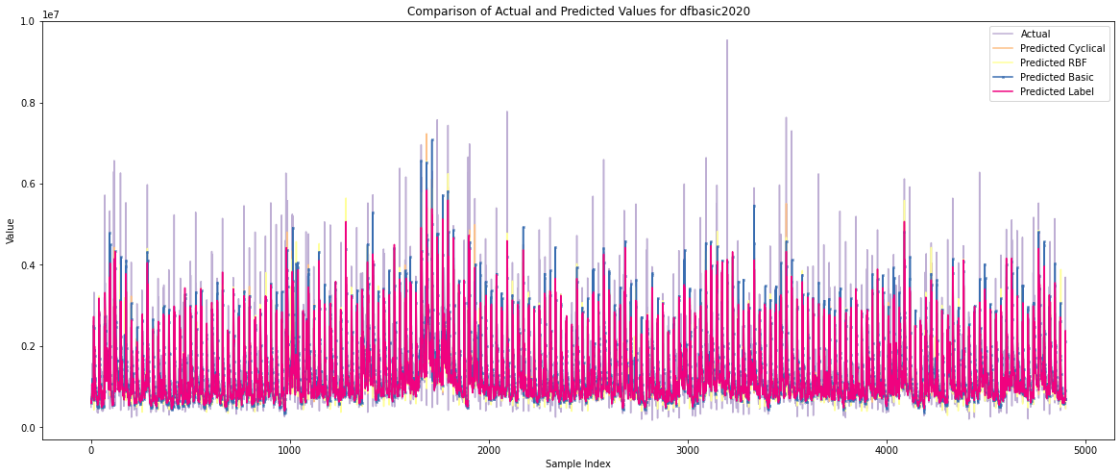



Figure 4.9 Comparison of Three Time Features LSTM

Since there is little difference between the models, we prefer to choose a simpler model, thus we still use the basic model as our optimal model in the stage of introducing the periodic time feature.

## 4.4.2 CEEMDAN-LSTM of QQQ

The high-frequency, low-frequency, and trend subsequences in the prediction interval were predicted separately using an established LSTM prediction model. Table 4.3

displays the evaluation results for each subsequence prediction model: The low-frequency and trend prediction models have R2 values greater than 0.9, indicating excellent forecasting. However, the high-frequency prediction model has an R2 of only 0.2816, meaning insufficient explanation and large prediction error, requiring further improvement.

Table 4.3 Evaluation results of three IMFs

| Model | R2 | RMSE | MAE |
|---|---|---|---|
| High-frequency | 0.2816 | 574,364.06 | 368,323.66 |
| Low-frequency | 0.9957 | 50,178.82 | 38,113.79 |
| Trend | 0.9198 | 3,465.87 | 3,350.37 |

The IMF1-IMF3, components of high-frequency sequences, contain a lot of noise and fluctuation-specific information. To increase the model's accuracy, we need to recombine the high-frequency to produce new wave characteristics and noise proportions.

Table 4.4 Evaluation results of different IMF combinations

| IMFs Composition | R2 | RMSE | MAE |
|---|---|---|---|
| 1+2+3 | 0.2816 | 574,364.06 | 368,323.66 |
| 1+2 | 0.6669 | 375,648.08 | 245,938.74 |
| 2+3 | 0.7173 | 283,421.54 | 197,569.25 |
| 1+3 | 0.5252 | 450,945.19 | 319,977.70 |

Table 4.4 shows that recomposing IMFs in the high-frequency sequence can significantly minimize the inaccuracy. Based on this, we compute the overall predicted values for comparison, seeking for the best combination.

Table 4.5 QQQ integrated forecasting results evaluation

| | Model | $R^2$ | RMSE | MAE | MAPE |
|---|---|---|---|---|---|
| ★ | Basic-LSTM | 0.7048 | 535,549.78 | 315,784.59 | 21.79% |
| | IMFs123-LSTM | 0.6574 | 576,950.42 | 374,550.52 | 25.85% |
| ★★ | IMFs12-LSTM | 0.8277 | 409,169.58 | 279,264.97 | 19.27% |
| | IMFs23-LSTM | 0.5518 | 659,904.18 | 472,744.52 | 32.63% |
| | IMFs13-LSTM | 0.7235 | 518,308.03 | 362,315.60 | 25.00% |

Table 4.5 shows the results improved significantly, with an $R^2$ of 0.8277. It indicates that the CEEMDAN-LSTM model can explain the variability of most data and has good prediction performance. Compared with the original model, the CEEMDAN-LSTM model improves R 2 by 17.44%, RMSE by 23.60%, MAE and MAPE by more than 10%, indicating that the optimization is successful.

To show the actual meaning of the error, I used the inverse normalization of the data to calculate the error, due to the large range of the original data and volatility, the error is particularly large. The relative value of MAPE was 19.27%, but the absolute value of RMSE was 409,169.58, and MAE was 279,264.97, both of which were higher than the mean value of 1,449,008. The inaccuracy suggests that we should continue to look into models that are more resistant to outliers or can better model extremes.

A comparison of the ideal CEEMDAN model and the Basic model with the actual values is presented in Figure 4.9. Overall, the CEEMDAN factorization factor learns the extremes better and extracts the majority of the metrics. CEEMDAN decomposition and reorganizing can greatly increase modeling accuracy. It demonstrates the viability and efficiency of this concept.

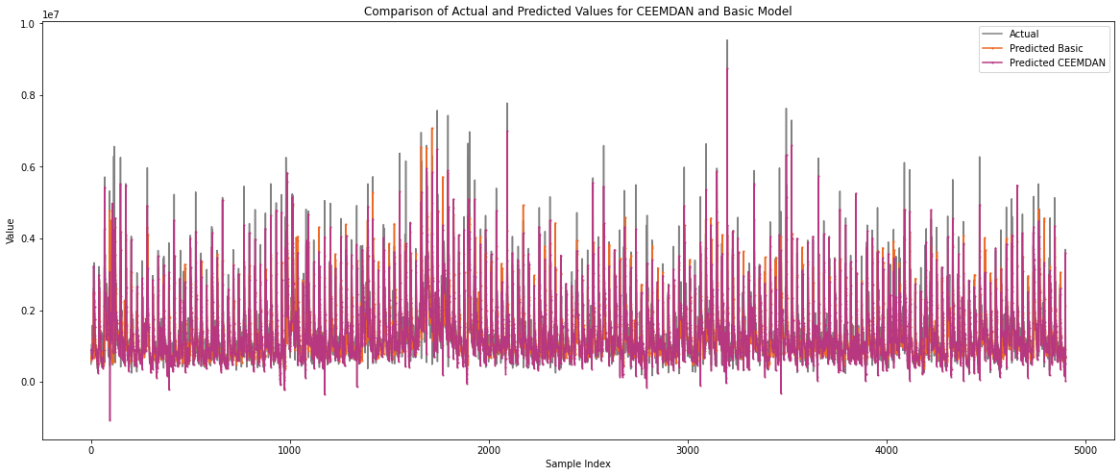



Figure 4.10 Comparison of CEEMDAN and Basic Model

# 5 Conclusion and Recommendation

According to the characteristics of QQQ intra-day transaction volume, my study adopts the time feature optimization method and CEEMDAN decomposition and refactoring method, to investigate the feasibility and optimization direction of QQQ intra-day trading volume series forecast

In the direction of time feature optimization, due to the notable intra-day and intra-week periodicity of trading volume series, my study fits this Cyclical feature using three methods: Label Encoding, Cyclical Encoding, and Radial Basis Function. Surprisingly, these three methods did not greatly increase prediction accuracy and even decreased the model's performance. This also leads me to conclude that cyclical signals are not the

main factors affecting trading volume prediction. Poor extreme value fitting indicates that macroeconomic issues have a greater impact on QQQ.It is commonly believed that adding cyclical characteristics to financial indicators with periodicity can increase forecasting accuracy. However, research on QQQ has shown that this does not apply to all markets, particularly in markets where the macro index has a significant influence. Therefore, the method should be used with caution. The comparison of the three feature optimization methods also reveals that due to the high noise level in trading volume data, simple models may be better to capture relevant information and generalize to unknown data. Just like Occam's razor principle, when the performance of two models is comparable, the simpler one should be chosen.

The results of time feature optimization show that the majority of normal trading days can be predicted by using time-indexed QQQ trading volume and yield. However, we still need to improve the prediction of market sentiment, macroeconomic indicators, and unexpected economic events. These indices are difficult to measure and frequently demand in-depth financial market analysis. Based on this, the Empirical Mode Decomposition approach is taken into consideration in my study. This method can extract the features from the original trade volume signals by separating the high-frequency, low-frequency, and trend factors.

During the EMD step, my study uses the CEEMDAN method to decompose and restructure to produce simple fluctuating subsequences. Then the study filtered the noise in the high-frequency sequence and continued reconstructing the sub-sequence prediction model to ensure that all of the sub-sequence's complicated fluctuation patterns

were extracted. The final results certify the CEEMDAN's feasibility and great potential in the QQQ ETF transaction volume forecast.

However, due to computing power constraints, the selection of model layers, optimizer, and learning rate is still subjective in the CEEMDAN-LSTM modeling process. The two-layer LSTM may be unable to fully exploit the deep-level change patterns contained in the complex and large-fluctuation time series and needs to be optimized. And the LSTM model can also add an attention mechanism. These strategies should be able to increase the model's prediction accuracy even more.

While employing the CEEMDAN method, we could also consider improving noise management. Various noise addition methodologies, such as dynamic noise modification based on data properties, may enhance the accuracy and stability of decomposition. Furthermore, this experimental method has only been used to evaluate the QQQ ETF in this study. Considering there is little study on volume prediction, it is vital to widen the research market and examine its universality.

# List of References

Behrendt, S., & Schmidt, A. (2021). Nonlinearity matters : The stock price–trading volume relation revisited. Economic Modelling, 98, 371-385.

Berkowitz, S. A., Logue, D. E., & Noser Jr, E. A. (1988). The total cost of transactions on the NYSE. The Journal of Finance, 43(1), 97-112.

Cao, J., Li, Z., & Li, J. (2019). Financial time series forecasting model based on CEEMDAN and LSTM. Physica A: Statistical mechanics and its applications, 519, 127-139.

Daubechies I. Ten lectures on wavelets. Society for industrial and applied mathematics; 1992.

Fu, X., Zhang, S., Chen, J., Ouyang, T., & Wu, J. (2019). A Sentiment-Aware Trading Volume Prediction Model for P2P Market Using LSTM. IEEE Access, 7, 81934-81944.

Ghosh, P., Neufeld, A., & Sahoo, J. K. (2022). Forecasting directional movements of stock prices for intraday trading using LSTM and random forests. Finance Research Letters, 46, 102280.

Graves A. Generating sequences with recurrent neural networks[J/OL]. 2013, arXiv: 1308.0850

Huang, S., Yu, L., Luo, W., Pan, H., Li, Y., Zou, Z., ... & Chen, J. (2023). Runoff Prediction of Irrigated Paddy Areas in Southern China Based on EEMD-LSTM Model. Water, 15(9), 1704.

Jain, Prem C. and Joh, Gun-Ho, (1988), The Dependence between Hourly Prices and Trading Volume, Journal of Financial and Quantitative Analysis, 23, issue 3, p. 269-283, https://EconPapers.repec.org/RePEc:cup:jfinqa:v:23:y:1988:i:03:p:269-283_01.

Kourentzes, N., & Crone, S. F. (2010). Frequency independent automatic input variable selection for neural networks for forecasting. In Pro-ceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. https://doi.org/10.1109/IJCNN.2010.5596637

Libman D, Haber S, Schaps M. Volume Prediction With Neural Networks. Front Artif Intell. 2019 Oct 9;2:21. doi: 10.3389/frai.2019.00021. PMID: 33733110; PMCID: PMC7861222.

Liu, Y., He, J., Wang, Y., Liu, Z., He, L., & Wang, Y. (2023). Short-term wind power prediction based on CEEMDAN-SE and bidirectional LSTM neural network with Markov chain. Energies, 16(14), 5476.

M.E. Torres, M.A. Colominas, G. Schlotthauer, P. Flandrin, A complete ensemble empirical mode decomposition with adaptive noise, in: IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 125, 2011, pp. 4144-4147.

McInish, T. H., & Wood, R. A. (1992). An analysis of intraday patterns in bid/ask spreads for NYSE stocks. the Journal of Finance, 47(2), 753-764.

N.E. Huang, Z. Shen, S.R. Long, M.C. Wu, H.H. Shih, Q. Zheng, et al., The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis, Proc. Math. Phys. Eng. Sci. 454 (1971) (1998) 903–995.

Rehman N, Mandic DP. Multivariate empirical mode decomposition［J］. Proceedings of the Royal Society， 2010,466( 2117) : 1291-1302．

Thomas F, Christopher K． Deep learning with long short term memory networks for financial market predictions ［J］． European Journal of Operational Research，2018，270( 2): 654-669．

Torres M E, Colominas M A, Schlotthauer G, et al. A complete ensemble empirical mode decomposition with adaptive noise[C]//2011 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2011: 4144-4147.

Wang J, Sun X, Cheng Q, Cui Q. An innovative random forest-based non-linear ensemble paradigm of improved feature extraction and deep learning for carbon price forecasting. Sci Total Environ 2021;762:143099.

Wei Huang, Yoshiteru Nakamori, Shou-Yang Wang, (2005), Forecasting stock market movement direction with support vector machine, Computers & Operations Research, Volume 32, Issue 10, Pages 2513-2522, ISSN 0305-

Wu Z H, Huang Norden E., Ensemble empirical mode decomposition: A noise-assisted data analysis method, Adv. Adapt. Data Anal. 1 (01)(2005) 1–41.

HE Yi-yue, LIU Lei, GAO Ni. Research on Forecasting Intraday Trading Volume of Stock Index Based on M-LSTM[J]. Operations Research and Management Science, 2022, 31(10): 196-203.

# Appendices

# Appendix A Code of Data Process

# Appendix A.1 Code of Data Basic Process

```
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pandas as pd
import math
import numpy as np
from datetime import datetime, timedelta, date
import time
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
%matplotlib inline

# Data processing
file_path = 'QQQ.xlsx'
df = pd.read_excel(file_path)
print(df)

## 1. 补缺失值/保留 9:30-16:00/看了日内特征和周内特征
## 1. Add missing value/retain 9:30 -16:00/look at intra-day and intra-week features

# Convert the 'Date' column to datetime format
df['Datetime'] = pd.to_datetime(df['Date'], format='%Y-%m-%d %H:%M:%S')

 # Set 'Datetime' as the index and drop unnecessary columns
df.set_index('Datetime', inplace=True)
df.drop(columns=['Date', 'Time'], inplace=True)

# 如果 Volume 不是数值类型，则转换
df['Volume'] = pd.to_numeric(df['Volume'], errors='coerce')

# Remove any NaN values
df.dropna(subset=['Volume'], inplace=True)

# 为缺失的分钟补 0 Add 0 for vacant
df = df.resample('T').asfreq().fillna(0)
```

```
38.
39.  df_resampled = df.resample('15T').sum()
40.  df_resampled.index = pd.to_datetime(df_resampled.index)
41.  df_resampled.index = df_resampled.index.astype(str)
42.
43.  #因为美股开盘收盘时间为9：30-16:00 stock marcket open and close time is 9：30-16:00
44.  df_resampled.index = pd.to_datetime(df_resampled.index)
45.  start_time = datetime.strptime('09:30', '%H:%M').time()
46.  end_time = datetime.strptime('16:00', '%H:%M').time()
47.  mask = (df_resampled.index.time >= start_time) & (df_resampled.index.time <= end_time)
48.  df = df_resampled[mask]
49.  df = df.dropna()
50.
51.  df = df.loc['2020-01-01 09:30':]
52.  df
53.
54.  # 每日同一时点加和求均值 Average everyday same time
55.  mean_by_time = df.groupby(df.index.time).mean()
56.  mean_by_time[['Volume']].plot(color='black', figsize=(20, 6), grid=True)
57.  plt.fill_between(mean_by_time.index, mean_by_time['Volume'], color='gray', alpha=0.3)
58.
59.  df_20230803 = df.loc['2023-08-03']
60.  df_20230803['Volume'].plot(color='black', figsize=(20, 6), grid=True)
61.
62.  df.loc[:, 'Weekday'] = df.index.day_name()
63.  #print(df2)
64.  mean_by_weekday = df.groupby('Weekday').mean()
65.  weekday_labels = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
66.  mean_by_weekday.index = weekday_labels
67.  mean_by_weekday[['Volume']].plot(color='black', figsize=(10, 6), grid=True, marker='o')
68.  plt.xlabel('Weekday')
69.  plt.ylabel('Mean Volume')
70.  plt.title('Mean Volume by Weekday')
71.  plt.legend(['Volume'])
72.  plt.show()
73.  ###上图可以看出 符合通常的周一和周五交易比较多的情况 值得一提的是周三交易量也较多 说明周内不同工作日
     存在一定的差异
74.  ###The chart above shows that, in line with the usual pattern of Monday and Friday trading, there was also a lot of volume
     on Wednesday, indicating that there are some differences between the working days of the week
75.
76.  df['Volume'].plot(subplots=True, color='black',figsize=(20, 6), grid=True, )
77.
78.
79.  # 将索引转换为datetime 类型 index to datetime
80.  df.index = pd.to_datetime(df.index)
81.  # 筛选出第一列中包含 0 的行并显示 shows columns with 0
82.  rows_with_zero = df[df['Open'] == 0]
83.  #print(rows_with_zero)
84.  df = df[df['Open'] != 0]
```

```
#print(df)
# 计算每日数据数量 counts numbers of each day's volume
daily_counts = df.index.to_series().dt.date.value_counts()
# 找出非 27 天的日期 find the day special(non27)
non_27_days = daily_counts[daily_counts != 27]
print(non_27_days)

# Assuming non_27_days_index is created correctly and contains datetime-like objects
non_27_days_index = pd.Index(non_27_days.index)
# Convert the DataFrame index to date-only format if it's in datetime format
df_index_as_date = df.index.normalize()
# Filter out the non 27 days
df2 = df[~df_index_as_date.isin(non_27_days_index)]
print(df2)

print("Skewness 偏度: ", df2["Volume"].skew())
print("Kurtosis 峰度: ", df2["Volume"].kurt())
print("Median 中位数: ", df2["Volume"].median())
print("10th percentile: ", df2["Volume"].quantile(0.1))
print("90th percentile: ", df2["Volume"].quantile(0.9))
minmaxrange = df2["Volume"].max() - df2["Volume"].min()
print("范围 minmaxrange: ", minmaxrange)
cv = df2["Volume"].std() / df2["Volume"].mean()
print("Coefficient of variation 变异系数: ", cv)
df2["Volume"].describe()
```

# Appendix A.2 Code of Time Features

```
# 2. Time Features
## 1. Add weekday and indextime LABLE ENCODING
df2.loc[:, 'Weekday'] = df2.index.day_name()
#print(df2)
remaining_dates_count = df2.index.normalize().nunique()
print(remaining_dates_count)

timeseq = []
for d in range(0,918):
    i = 0
    for t in range(0,27):
        i += 1
        timeseq.append(i)
timeseq = pd.DataFrame(timeseq)
df2['index_time'] = timeseq.values
df2

## 2. 循环编码 cyclical encoding with sine/cosine transformation
day_mapping = {
    'Monday': 0,
```

```
    'Tuesday': 1,
    'Wednesday': 2,
    'Thursday': 3,
    'Friday': 4
}
df2['numeric_day'] = df2['Weekday'].map(day_mapping)
df2['sin_weekday'] = np.sin(2 * np.pi * df2['numeric_day'] / 5)
df2['cos_weekday'] = np.cos(2 * np.pi * df2['numeric_day'] / 5)

plt.figure(figsize=(12, 8))
plt.plot(df2.index[:2000], df2['sin_weekday'][:2000], label='Sin of Weekday')
plt.plot(df2.index[:2000], df2['cos_weekday'][:2000], label='Cos of Weekday', linestyle='--')
plt.title('Cyclic Encoding of Weekday over Time (First 2000 Entries)')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(14, 7))
# Plot for cyclic encoding of weekdays
plt.subplot(1, 2, 1)
plt.scatter(df2['sin_weekday'], df2['cos_weekday'])
plt.title(' Cyclic Encoding of Weekdays')
plt.xlabel(' Sin of Weekday')
plt.ylabel(' Cos of Weekday')
plt.axhline(0, color='grey', lw=0.5)
plt.axvline(0, color='grey', lw=0.5)
plt.grid(True)

df2['sin_time'] = np.sin(2 * np.pi * df2['index_time'] / 27.0)
df2['cos_time'] = np.cos(2 * np.pi * df2['index_time'] / 27.0)
df2

plt.figure(figsize=(14, 7))

# Plot for cyclic encoding of weekdays
plt.subplot(1, 2, 1)
plt.scatter(df2['sin_time'], df2['cos_time'])
plt.title(' Cyclic Encoding of time')
plt.xlabel(' Sin of Weekday')
plt.ylabel(' Cos of Weekday')
plt.axhline(0, color='grey', lw=0.5)
plt.axvline(0, color='grey', lw=0.5)
plt.grid(True)

plt.figure(figsize=(12, 8))
plt.plot(df2.index[:100], df2['sin_time'][:100], label='Sin of time')
plt.plot(df2.index[:100], df2['cos_time'][:100], label='Cos of time', linestyle='--')
```

```
plt.title('Cyclic Encoding of  Time (First 2000 Entries)')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.tight_layout()
plt.show()

## 3. radial basis functions 径向基函数
df2['135_weekday'] = (np.arange(len(df2)) % 135) + 1 #27*5=135
from sklego.preprocessing import RepeatingBasisFunction

rbf = RepeatingBasisFunction(n_periods=5,
                             column="135_weekday",
                             input_range=(1,140),
                             remainder="drop")
rbf.fit(df2)

X_3 = pd.DataFrame(data=rbf.transform(df2), index=df2.index)
X_3.plot(subplots=True, figsize=(18, 8),
         sharex=True, title="Radial Basis Functions",
         legend=False)
plt.show()
X_3.iloc[:2000].plot(subplots=True, figsize=(18, 8), sharex=True, title="Radial Basis Functions", legend=False)
plt.show()
df3 = pd.concat([df2, X_3], axis=1)
df3

returns = df3['Close'].pct_change()
df4 = df3.drop(columns=['High', 'Low', 'Close','Open','Weekday','135_weekday'])
df4 = pd.concat([df4, returns.rename('Returns')], axis=1)
df4 = df4.fillna(0)
df4

other_columns = [col for col in df4.columns if col != 'Returns']
new_order = [other_columns[0], 'Returns'] + other_columns[1:]
df5= df4[new_order]
rename_dict = {df5.columns[i + 8]: f'ra{i}' for i in range(5)}
df5.rename(columns=rename_dict, inplace=True)
df5

# Save data
dfbasic = df5.drop(columns=['index_time', 'numeric_day', 'sin_weekday','cos_weekday','ra0','ra1','ra2','ra3','ra4','sin_time','cos_time']
)
dfbasic
dfdu = df5.drop(columns=['sin_weekday','cos_weekday','ra0','ra1','ra2','ra3','ra4','sin_time','cos_time'])
dfdu
dfsc = df5.drop(columns=['index_time', 'numeric_day', 'ra0','ra1','ra2','ra3','ra4'])
dfsc
dfra = df5.drop(columns=['index_time', 'numeric_day', 'sin_weekday','cos_weekday','sin_time','cos_time'])
```

```
dfra

dfbasic2020 = dfbasic.loc['2020-01-01 09:30':]
dfdu2020 = dfdu.loc['2020-01-01 09:30':]
dfsc2020 = dfsc.loc['2020-01-01 09:30':]
dfra2020 = dfra.loc['2020-01-01 09:30':]
%store dfbasic2020
%store dfdu2020
%store dfsc2020
%store dfra2020
```

# Appendix A.3 Code of CEEMDAN

```
# 3. CEEMDAN
df6 = df5.drop(columns=['index_time', 'numeric_day', 'sin_weekday','cos_weekday','ra0','ra1','ra2','ra3','ra4','Returns','sin_time','cos_time'])
df6
from PyEMD import CEEMDAN
from PyEMD import Visualisation
import scipy
import statsmodels as sm

# Create a function for CEEMDAN
def apply_ceemdan(signal):
    ceemdan = CEEMDAN()
    imfs_and_residue = ceemdan(signal)
    return imfs_and_residue

# Extract the 'Volume' column as a numpy array
volume_data = df6['Volume'].values

# Apply CEEMDAN to the 'Volume' column
imfs_and_residue = apply_ceemdan(volume_data)

# The last row is the residue
num_imfs = imfs_and_residue.shape[0] - 1  # Subtracting 1 to exclude the residue
residue = imfs_and_residue[-1]

# Add the decomposed IMFs and residue back to the DataFrame
for i in range(num_imfs):
    df6[f"imfs_{i+1}"] = imfs_and_residue[i]

# Add residue as the last 'imf' column
df6['residue'] = residue

# Display the DataFrame with the IMFs and residue
```

```
print(df6)

plt.figure(figsize=(14, 24))
plt.subplot(num_imfs + 2, 1, 1)  # Add 2 for the residue plot
plt.plot(df6.index, df6['Volume'], label='Volume')
plt.title('Original Volume Data')
plt.legend()

for i in range(num_imfs):
    plt.subplot(num_imfs + 2, 1, i + 2)  # Changed to num_imfs + 2
    plt.plot(df6.index, df6[f"imfs_{i+1}"], label=f"IMF {i+1}")
    plt.title(f"IMF {i+1}")
    plt.legend()

plt.subplot(num_imfs + 2, 1, num_imfs + 2)  # The last plot for the residue
plt.plot(df6.index, df6['residue'], label='Residue', color='purple')
plt.title('Residue')
plt.legend()
plt.tight_layout()
plt.show()

dfimfs = df6.copy()
dfimfs

from scipy import stats

# Set P value
p_value_threshold = 0.05
high_freq_imfs = []
low_freq_imfs = []
trend = df6['residue']
found_threshold = False  # mark the IMF first smaller than P

for i in range(1, num_imfs+1):
    imf_col = f'imfs_{i}'
    t_stat, p_value = stats.ttest_1samp(df6[imf_col], 0)

    if p_value < p_value_threshold and not found_threshold:
        found_threshold = True
        low_freq_imfs.append(df6[imf_col])
    elif not found_threshold:
        high_freq_imfs.append(df6[imf_col])
    else:
        low_freq_imfs.append(df6[imf_col])

# Sum high and low
high_freq_series = np.sum(high_freq_imfs, axis=0) if high_freq_imfs else np.zeros(len(df6))
low_freq_series = np.sum(low_freq_imfs, axis=0) if low_freq_imfs else np.zeros(len(df6))

```

```
for i in range(1, num_imfs+1):
    df6.drop(columns=[f'imfs_{i}'], inplace=True)

df6['high_freq_series'] = high_freq_series
df6['low_freq_series'] = low_freq_series
df6['trend'] = trend

plt.figure(figsize=(14, 6))

plt.subplot(3, 1, 1)
plt.plot(df6.index, df6['high_freq_series'], label='High Frequency Series')
plt.title('High Frequency Series')
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(df6.index, df6['low_freq_series'], label='Low Frequency Series')
plt.title('Low Frequency Series')
plt.legend()

plt.subplot(3, 1, 3)
plt.plot(df6.index, df6['trend'], label='Trend')
plt.title('Trend (Residue)')
plt.legend()

plt.tight_layout()
plt.show()

high_freq_imfs #we could know 123 is high 456..14 is low

df6_last_2000 = df6.iloc[-2000:]
plt.figure(figsize=(14, 10))
plt.subplot(3, 1, 1)
plt.plot(df6_last_2000.index, df6_last_2000['high_freq_series'], label='High Frequency Series')
plt.title('High Frequency Series (Last 2000)')
plt.legend()

plt.subplot(3, 1, 2)
plt.plot(df6_last_2000.index, df6_last_2000['low_freq_series'], label='Low Frequency Series')
plt.title('Low Frequency Series (Last 2000)')
plt.legend()

plt.subplot(3, 1, 3)
plt.plot(df6_last_2000.index, df6_last_2000['trend'], label='Trend Series')
plt.title('Trend Series (Last 2000)')
plt.legend()

plt.tight_layout()
plt.show()
```

```
### SAVE DATA
%store dfce
dfce2020 = dfce.loc['2020-01-01 09:30':]
%store dfce2020
dfce2020
dfcehigh=dfce2020.drop(columns=['Volume','low_freq_series','trend'])
dfcehigh
dfcelow=dfce2020.drop(columns=['high_freq_series','trend','Volume'])
dfcelow
dfcetrend=dfce2020.drop(columns=['Volume','high_freq_series','low_freq_series'])
dfcetrend
dfcevolume=dfce2020.drop(columns=['trend','high_freq_series','low_freq_series'])
dfcevolume
%store dfcehigh
%store dfcelow
%store dfcetrend
%store dfcevolume

dfhigh12 = pd.DataFrame()
dfhigh12['dfhigh12']= dfimfs['imfs_1'] + dfimfs['imfs_2']
dfhigh12
dfhigh23 = pd.DataFrame()
dfhigh23['dfhigh23'] = dfimfs['imfs_3'] + dfimfs['imfs_2']
dfhigh23
dfhigh13 = pd.DataFrame()
dfhigh13['dfhigh23'] = dfimfs['imfs_3'] + dfimfs['imfs_1']
dfhigh13
%store dfhigh12
%store dfhigh13
%store dfhigh23
```

# Appendix B Code of LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_variance_score, median_absolute_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from keras.wrappers.scikit_learn import KerasRegressor
```

```
import pickle
from keras.models import load_model

%store -r dfbasic2020
%store -r dfdu2020
%store -r dfsc2020
%store -r dfra2020
%store -r dfce2020
%store -r dfcehigh
%store -r dfcelow
%store -r dfcetrend

import os
def ensure_directory_exists(directory_path):
    if not os.path.exists(directory_path):
        os.makedirs(directory_path)
        print(f"Directory created: {directory_path}")
    else:
        print(f"Directory already exists: {directory_path}")
directory_path = './model_saves'
ensure_directory_exists(directory_path)

def check_write_permission(directory_path):
    if os.access(directory_path, os.W_OK):
        print(f"Write permission is granted for the directory: {directory_path}")
    else:
        print(f"Write permission is not granted for the directory: {directory_path}")
check_write_permission(directory_path)

# MinMaxScaler
def preprocess_data(df):
    scaler = MinMaxScaler(feature_range=(0, 1))
    df.iloc[:, 0:1] = scaler.fit_transform(df.iloc[:, 0:1])  # 只归一化第一列
    return df, scaler

# split_data
def split_data(df):
    train_split = int(len(df) * 0.6)
    validation_split = int(len(df) * 0.8)
    df_for_training = df[:train_split]
    df_for_validation = df[train_split:validation_split]
    df_for_testing = df[validation_split:]
    return df_for_training, df_for_validation, df_for_testing

# create_dataset
def create_dataset(dataset, n_past):
    dataX, dataY = [], []
    for i in range(n_past, len(dataset)):
        dataX.append(dataset.iloc[i - n_past:i, :].values)  # Assume all columns except target are features
```

```
        dataY.append(dataset.iloc[i, 0])  # Assuming target is the first column
    return np.array(dataX), np.array(dataY)

# build  model LSTM
def build_model(input_shape):
    model = Sequential([
        LSTM(50, return_sequences=True, input_shape=input_shape),
        LSTM(50),
        Dropout(0.2),
        Dense(1)
    ])
    model.compile(loss='mae', optimizer=Adam(learning_rate=0.001))
    return model

def keras_model_wrapper(n_past, input_dim):
    def model():
        return build_model((n_past, input_dim))
    return model

# evaluate  model
def evaluate_model(model, X, y, scaler, dataset_type="Test"):
    prediction = model.predict(X)
    prediction = scaler.inverse_transform(prediction.reshape(-1, 1))
    y = scaler.inverse_transform(y.reshape(-1, 1))
    mse = mean_squared_error(y, prediction)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y, prediction)
    r2 = r2_score(y, prediction)
    evs = explained_variance_score(y, prediction)
    medae = median_absolute_error(y, prediction)
    print(f"{dataset_type} Data: R^2 Score: {r2}, MSE: {mse}, RMSE: {rmse}, MAE: {mae}, Explained Variance Score: {evs}, Median Absolute Error: {medae}")

    plt.figure(figsize=(10, 5))
    plt.plot(y, label='Actual', color='blue')
    plt.plot(prediction, label='Predicted', color='red')
    plt.title(f'{dataset_type} Data - Actual vs Predicted')
    plt.xlabel('Sample Index')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

# know process
def process_datasets(datasets):
    for name, df in datasets.items():
        print(f"Processing dataset: {name}")
        main(df, name)


```

```python
# predict and save results
def predict_and_save_results(model, X, y, scaler, dataset_name):
    try:
        predictions = model.predict(X)
        predictions_inverse = scaler.inverse_transform(predictions.reshape(-1, 1))
        y_inverse = scaler.inverse_transform(y.reshape(-1, 1))

        prediction_file = f'predictions_{dataset_name}.pkl'
        true_file = f'true_values_{dataset_name}.pkl'

        with open(prediction_file, 'wb') as f:
            pickle.dump(predictions_inverse, f)
        with open(true_file, 'wb') as f:
            pickle.dump(y_inverse, f)

        if os.path.exists(prediction_file) and os.path.exists(true_file):
            print(f"Predictions and actual values successfully saved for {dataset_name}.")
        else:
            print(f"Error: File not saved for {dataset_name}.")

    except Exception as e:
        print(f"Failed to save predictions for {dataset_name}: {str(e)}")

# main
def main(df, dataset_name):
    start_time = time.time()

    df, scaler = preprocess_data(df)

    df_for_training, df_for_validation, df_for_testing = split_data(df)

    # Creat train/val/test
    n_past = 56
    trainX, trainY = create_dataset(df_for_training, n_past)
    valX, valY = create_dataset(df_for_validation, n_past)
    testX, testY = create_dataset(df_for_testing, n_past)

    # Creat model
    model = KerasRegressor(build_fn=keras_model_wrapper(n_past, df_for_training.shape[1]), verbose=1)
    parameters = {'batch_size': [16, 32], 'epochs': [50, 100]}

    # grid_search
    grid_search = GridSearchCV(estimator=model, param_grid=parameters, cv=2)
    grid_search_result = grid_search.fit(trainX, trainY, validation_data=(valX, valY))

    # best_model & evaluate_model
    best_model = grid_search_result.best_estimator_.model
    evaluate_model(best_model, trainX, trainY, scaler, f"Train {dataset_name}")
    evaluate_model(best_model, valX, valY, scaler, f"Validation {dataset_name}")
```

```
    evaluate_model(best_model, testX, testY, scaler, f"Test {dataset_name}")

    # predict and save results
    predict_and_save_results(best_model, testX, testY, scaler, dataset_name)

    # save best model
    best_model.save(f'best_model_{dataset_name}.h5')

    # save grid search info
    grid_search_info = {
        'best_params': grid_search_result.best_params_,
        'best_score': grid_search_result.best_score_
    }
    with open(f'grid_search_info_{dataset_name}.pkl', 'wb') as f:
        pickle.dump(grid_search_info, f)
    with open(f'scaler_{dataset_name}.pkl', 'wb') as f:
        pickle.dump(scaler, f)

    end_time = time.time()
    print(f"Total time taken for {dataset_name}: {end_time - start_time:.2f} seconds")

if __name__ == "__main__":
    datasets = {
        'dfbasic2020': dfbasic2020,
        'dfdu2020': dfdu2020,
        'dfsc2020': dfsc2020,
        'dfra2020': dfra2020,
        'dfcehigh': dfcehigh,
        'dfcelow': dfcelow,
        'dfcetrend': dfcetrend,
        'dfhigh12': dfhigh12,
        'dfhigh23': dfhigh23,
        'dfhigh13': dfhigh13,
    }
    process_datasets(datasets)

import pickle

def load_and_plot_comparisons(dataset_names, true_dataset_name):
    plt.figure(figsize=(20, 8))
    palette = plt.get_cmap('Accent')

    with open(f'true_values_{true_dataset_name}.pkl', 'rb') as f:
        true_values = pickle.load(f)
    plt.plot(true_values, label=f'Actual', color=palette(1), linestyle='-', markersize=4)

    name_mapping = {
        'dfsc2020': 'Cyclical',
        'dfra2020': 'RBF',
```

```
        'dfbasic2020': 'Basic',
        'dfdu2020': 'Label'
    }

    for idx, dataset_name in enumerate(dataset_names):
        with open(f'predictions_{dataset_name}.pkl', 'rb') as f:
            predictions = pickle.load(f)
        if dataset_name == 'dfbasic2020':
            plt.plot(predictions, label=f'Predicted {name_mapping[dataset_name]}', color=palette((idx + 2) % 8), linestyle='-', marker='x', markersize=2)
        else:
            plt.plot(predictions, label=f'Predicted {name_mapping[dataset_name]}', color=palette((idx + 2) % 8), linestyle='-')

    plt.title('Comparison of Actual and Predicted Values for ' + true_dataset_name)
    plt.xlabel('Sample Index')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

dataset_names = ['dfsc2020', 'dfra2020', 'dfbasic2020', 'dfdu2020']
true_dataset_name = 'dfbasic2020'
load_and_plot_comparisons(dataset_names, true_dataset_name)

def load_predictions(file_name):
    with open(file_name, 'rb') as f:
        return pickle.load(f)

def evaluate_predictions(true_values, combined_predictions):
    mse = mean_squared_error(true_values, combined_predictions)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(true_values, combined_predictions)
    r2 = r2_score(true_values, combined_predictions)
    evs = explained_variance_score(true_values, combined_predictions)
    medae = median_absolute_error(true_values, combined_predictions)
    print(f"Data Evaluation - R^2 Score: {r2}, MSE: {mse}, RMSE: {rmse}, MAE: {mae}, Explained Variance Score: {evs}, Median Absolute Error: {medae}")

    plt.figure(figsize=(10, 5))
    plt.plot(true_values, label='Actual', color='blue')
    plt.plot(combined_predictions, label='Predicted', color='red')
    plt.title('Comparison of Actual Values and Predicted Values')
    plt.xlabel('Sample Index')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

predictions_high = load_predictions('predictions_dfcehigh.pkl')
predictions_high12 = load_predictions('predictions_dfhigh12.pkl')
predictions_high23 = load_predictions('predictions_dfhigh23.pkl')
```

```
predictions_high13 = load_predictions('predictions_dfhigh13.pkl')
predictions_low = load_predictions('predictions_dfcelow.pkl')
predictions_trend = load_predictions('predictions_dfcetrend.pkl')

combined_predictions123 = predictions_high + predictions_low + predictions_trend
combined_predictions12 = predictions_high12 + predictions_low + predictions_trend
combined_predictions23 = predictions_high23 + predictions_low + predictions_trend
combined_predictions13 = predictions_high13 + predictions_low + predictions_trend

with open('combined_predictions123.pkl', 'wb') as f:
    pickle.dump(combined_predictions123, f)
with open('combined_predictions12.pkl', 'wb') as f:
    pickle.dump(combined_predictions12, f)
with open('combined_predictions23.pkl', 'wb') as f:
    pickle.dump(combined_predictions23, f)
with open('combined_predictions13.pkl', 'wb') as f:
    pickle.dump(combined_predictions13, f)

true_values = load_predictions('true_values_dfbasic2020.pkl')

evaluate_predictions(true_values, combined_predictions123)
evaluate_predictions(true_values, combined_predictions12)
evaluate_predictions(true_values, combined_predictions23)
evaluate_predictions(true_values, combined_predictions13)

def load_and_plot_comparisons(dataset_names, true_dataset_name):
    plt.figure(figsize=(20, 8))

    colors = {
        'predictions_dfbasic2020': (0.929, 0.408, 0.145),  # Orange
        'combined_predictions12': (0.718, 0.216, 0.502)   # Purple
    }

    with open(f'true_values_{true_dataset_name}.pkl', 'rb') as f:
        true_values = pickle.load(f)
    plt.plot(true_values, label='Actual', color='grey', linestyle='-', marker='x', markersize=0)  # Black for actual values

    name_mapping = {
        'predictions_dfbasic2020': 'Basic',
        'combined_predictions12': 'CEEMDAN'
    }

    for dataset_name in dataset_names:
        with open(f'{dataset_name}.pkl', 'rb') as f:
            predictions = pickle.load(f)
        plt.plot(predictions, label=f'Predicted {name_mapping.get(dataset_name, "Unknown")}', color=colors.get(dataset_nam
e, 'gray'), linestyle='-', marker='.', markersize=2)

    plt.title(f'Comparison of Actual and Predicted Values for CEEMDAN and Basic Model')
```

```
    plt.xlabel('Sample Index')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

dataset_names = ['predictions_dfbasic2020', 'combined_predictions12']
true_dataset_name = 'dfbasic2020'
load_and_plot_comparisons(dataset_names, true_dataset_name)

```

# Appendix C An approximate depiction of the input data

| Datetime | Volume | Returns |
|---|---|---|
| 2020-01-02 09:30:00 | 3156623.0 | 0.000000 |
| 2020-01-02 09:45:00 | 1074391.0 | 0.000215 |
| 2020-01-02 10:00:00 | 985951.0 | -0.000463 |
| 2020-01-02 10:15:00 | 1100095.0 | -0.002495 |
| 2020-01-02 10:30:00 | 643005.0 | 0.000108 |
| ... | ... | ... |
| 2023-08-31 15:00:00 | 658377.0 | 0.000755 |
| 2023-08-31 15:15:00 | 595590.0 | 0.000368 |
| 2023-08-31 15:30:00 | 635705.0 | 0.000236 |
| 2023-08-31 15:45:00 | 3695453.0 | -0.000770 |
| 2023-08-31 16:00:00 | 481470.0 | -0.001391 |

24786 rows × 2 columns

Figure C.1 Basic LSTM model input data

| Datetime | Volume | Returns | index_time | numeric_day |
|---|---|---|---|---|
| 2020-01-02 09:30:00 | 3156623.0 | 0.000000 | 1 | 3 |
| 2020-01-02 09:45:00 | 1074391.0 | 0.000215 | 2 | 3 |
| 2020-01-02 10:00:00 | 985951.0 | -0.000463 | 3 | 3 |
| 2020-01-02 10:15:00 | 1100095.0 | -0.002495 | 4 | 3 |
| 2020-01-02 10:30:00 | 643005.0 | 0.000108 | 5 | 3 |
| ... | ... | ... | ... | ... |
| 2023-08-31 15:00:00 | 658377.0 | 0.000755 | 23 | 3 |
| 2023-08-31 15:15:00 | 595590.0 | 0.000368 | 24 | 3 |
| 2023-08-31 15:30:00 | 635705.0 | 0.000236 | 25 | 3 |
| 2023-08-31 15:45:00 | 3695453.0 | -0.000770 | 26 | 3 |
| 2023-08-31 16:00:00 | 481470.0 | -0.001391 | 27 | 3 |

24786 rows × 4 columns

Figure C.2 Lable Encoding LSTM model input data

| Datetime | Volume | Returns | sin_weekday | cos_weekday | sin_time | cos_time |
|---|---|---|---|---|---|---|
| 2020-01-02 09:30:00 | 3156623.0 | 0.000000 | -0.587785 | -0.809017 | 2.306159e-01 | 0.973045 |
| 2020-01-02 09:45:00 | 1074391.0 | 0.000215 | -0.587785 | -0.809017 | 4.487992e-01 | 0.893633 |
| 2020-01-02 10:00:00 | 985951.0 | -0.000463 | -0.587785 | -0.809017 | 6.427876e-01 | 0.766044 |
| 2020-01-02 10:15:00 | 1100095.0 | -0.002495 | -0.587785 | -0.809017 | 8.021232e-01 | 0.597159 |
| 2020-01-02 10:30:00 | 643005.0 | 0.000108 | -0.587785 | -0.809017 | 9.182161e-01 | 0.396080 |
| ... | ... | ... | ... | ... | ... | ... |
| 2023-08-31 15:00:00 | 658377.0 | 0.000755 | -0.587785 | -0.809017 | -8.021232e-01 | 0.597159 |
| 2023-08-31 15:15:00 | 595590.0 | 0.000368 | -0.587785 | -0.809017 | -6.427876e-01 | 0.766044 |
| 2023-08-31 15:30:00 | 635705.0 | 0.000236 | -0.587785 | -0.809017 | -4.487992e-01 | 0.893633 |
| 2023-08-31 15:45:00 | 3695453.0 | -0.000770 | -0.587785 | -0.809017 | -2.306159e-01 | 0.973045 |
| 2023-08-31 16:00:00 | 481470.0 | -0.001391 | -0.587785 | -0.809017 | -2.449294e-16 | 1.000000 |

24786 rows × 6 columns

Figure C.3 Cyclical Encoding LSTM model input data

| Datetime | Volume | Returns | ra0 | ra1 | ra2 | ra3 | ra4 |
|---|---|---|---|---|---|---|---|
| 2020-01-02 09:30:00 | 3156623.0 | 0.000000 | 1.000000 | 0.367879 | 0.018316 | 0.018316 | 0.367879 |
| 2020-01-02 09:45:00 | 1074391.0 | 0.000215 | 0.998707 | 0.394810 | 0.021123 | 0.015841 | 0.341900 |
| 2020-01-02 10:00:00 | 985951.0 | -0.000463 | 0.994838 | 0.422616 | 0.024297 | 0.013665 | 0.316934 |
| 2020-01-02 10:15:00 | 1100095.0 | -0.002495 | 0.988422 | 0.451212 | 0.027876 | 0.011757 | 0.293032 |
| 2020-01-02 10:30:00 | 643005.0 | 0.000108 | 0.979510 | 0.480498 | 0.031900 | 0.010090 | 0.270232 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 2023-08-31 15:00:00 | 658377.0 | 0.000755 | 0.005883 | 0.049484 | 0.583634 | 0.931596 | 0.201246 |
| 2023-08-31 15:15:00 | 595590.0 | 0.000368 | 0.006916 | 0.043625 | 0.552906 | 0.948381 | 0.220153 |
| 2023-08-31 15:30:00 | 635705.0 | 0.000236 | 0.008110 | 0.038360 | 0.522442 | 0.962972 | 0.240215 |
| 2023-08-31 15:45:00 | 3695453.0 | -0.000770 | 0.009484 | 0.033643 | 0.492380 | 0.975261 | 0.261428 |
| 2023-08-31 16:00:00 | 481470.0 | -0.001391 | 0.011063 | 0.029430 | 0.462849 | 0.985153 | 0.283778 |

24786 rows × 7 columns

Figure C.4 RBF Encoding LSTM model input data

| Datetime | high_freq_series |
|---|---|
| 2020-01-02 09:30:00 | 9.060558e+05 |
| 2020-01-02 09:45:00 | -7.321459e+05 |
| 2020-01-02 10:00:00 | -3.691569e+05 |
| 2020-01-02 10:15:00 | 1.467421e+05 |
| 2020-01-02 10:30:00 | -4.724748e+03 |
| ... | ... |
| 2023-08-31 15:00:00 | 5.924827e+03 |
| 2023-08-31 15:15:00 | -4.904735e+05 |
| 2023-08-31 15:30:00 | -8.364992e+05 |
| 2023-08-31 15:45:00 | 2.019055e+06 |
| 2023-08-31 16:00:00 | -1.118342e+06 |

24786 rows × 1 columns

IMFs 123

| Datetime | dfhigh12 |
|---|---|
| 2020-01-02 09:30:00 | 6.350186e+05 |
| 2020-01-02 09:45:00 | -6.293364e+05 |
| 2020-01-02 10:00:00 | -6.332796e+04 |
| 2020-01-02 10:15:00 | 4.217965e+05 |
| 2020-01-02 10:30:00 | 1.062559e+05 |
| ... | ... |
| 2023-08-31 15:00:00 | 3.580235e+05 |
| 2023-08-31 15:15:00 | -4.211163e+05 |
| 2023-08-31 15:30:00 | -1.104075e+06 |
| 2023-08-31 15:45:00 | 1.596496e+06 |
| 2023-08-31 16:00:00 | -1.339530e+06 |

24786 rows × 1 columns

IMFs 12

| Datetime | dfhigh23 |
|---|---|
| 2020-01-02 09:30:00 | 563467.967936 |
| 2020-01-02 09:45:00 | -392902.402061 |
| 2020-01-02 10:00:00 | -542457.918750 |
| 2020-01-02 10:15:00 | -176177.311793 |
| 2020-01-02 10:30:00 | 159917.195619 |
| ... | ... |
| 2023-08-31 15:00:00 | -707900.619432 |
| 2023-08-31 15:15:00 | -313054.864837 |
| 2023-08-31 15:30:00 | 347482.842867 |
| 2023-08-31 15:45:00 | 796271.134928 |
| 2023-08-31 16:00:00 | 117587.266031 |

24786 rows × 1 columns

IMFs 23

| Datetime | |
|---|---|
| 2020-01-02 09:30:00 | 6.136250e+05 |
| 2020-01-02 09:45:00 | -4.420530e+05 |
| 2020-01-02 10:00:00 | -1.325278e+05 |
| 2020-01-02 10:15:00 | 4.786503e+04 |
| 2020-01-02 10:30:00 | -2.756225e+05 |
| ... | ... |
| 2023-08-31 15:00:00 | 3.617268e+05 |
| 2023-08-31 15:15:00 | -2.467758e+05 |
| 2023-08-31 15:30:00 | -9.164063e+05 |
| 2023-08-31 15:45:00 | 1.645344e+06 |
| 2023-08-31 16:00:00 | -1.014741e+06 |

24786 rows × 1 columns

IMFs 13

Figure C.5 CEEMDAN High-frequency with different combinations

| Datetime | low_freq_series |
|---|---|
| 2020-01-02 09:30:00 | 8.407721e+05 |
| 2020-01-02 09:45:00 | 3.967354e+05 |
| 2020-01-02 10:00:00 | -5.470000e+04 |
| 2020-01-02 10:15:00 | -4.564613e+05 |
| 2020-01-02 10:30:00 | -7.620908e+05 |
| ... | ... |
| 2023-08-31 15:00:00 | -1.017317e+06 |
| 2023-08-31 15:15:00 | -5.837123e+05 |
| 2023-08-31 15:30:00 | -1.975781e+05 |
| 2023-08-31 15:45:00 | 6.608945e+03 |
| 2023-08-31 16:00:00 | -6.998321e+04 |

24786 rows × 1 columns

Low-frequency

| Datetime | trend |
|---|---|
| 2020-01-02 09:30:00 | 1.409795e+06 |
| 2020-01-02 09:45:00 | 1.409802e+06 |
| 2020-01-02 10:00:00 | 1.409808e+06 |
| 2020-01-02 10:15:00 | 1.409814e+06 |
| 2020-01-02 10:30:00 | 1.409821e+06 |
| ... | ... |
| 2023-08-31 15:00:00 | 1.669769e+06 |
| 2023-08-31 15:15:00 | 1.669776e+06 |
| 2023-08-31 15:30:00 | 1.669782e+06 |
| 2023-08-31 15:45:00 | 1.669789e+06 |
| 2023-08-31 16:00:00 | 1.669795e+06 |

24786 rows × 1 columns

Trend

Figure C.6 CEEMDAN Low-frequency and Trend

# Appendix D Figure of LSTM prediction

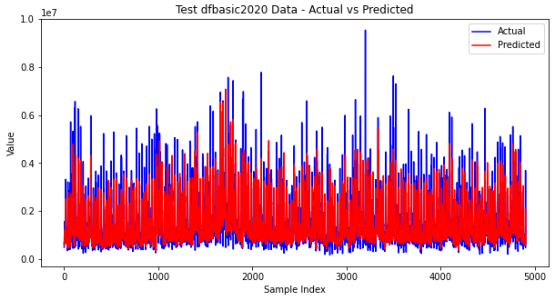



Figure D.1 Comparison of Basic Model Test Sets

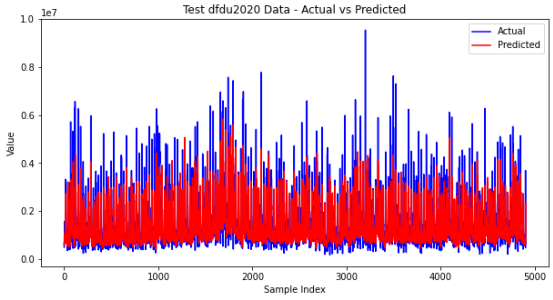



Figure D.2 Comparison of Label Encoding Model Test Sets

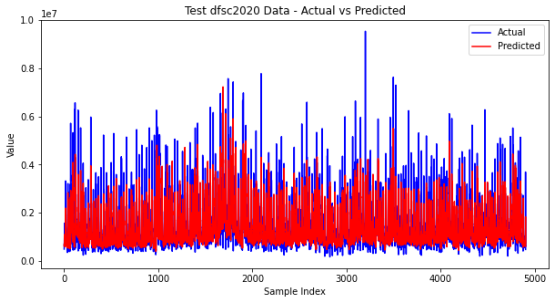



Figure D.3 Comparison of Cyclical Encoding Model Test Sets

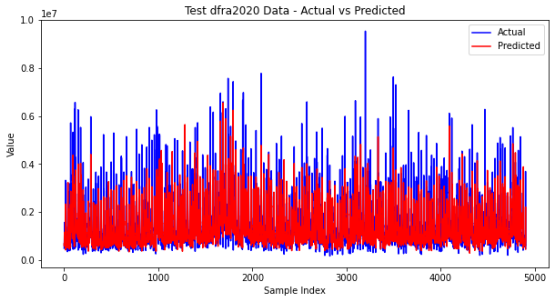



Figure D.4 Comparison of RBF Encoding Model Test Sets

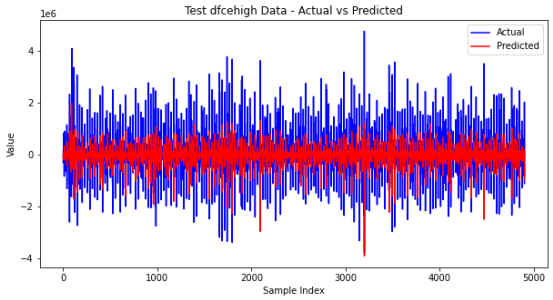



Figure D.5 Comparison of CEEMDAN High Frequency ( IMFs123 )

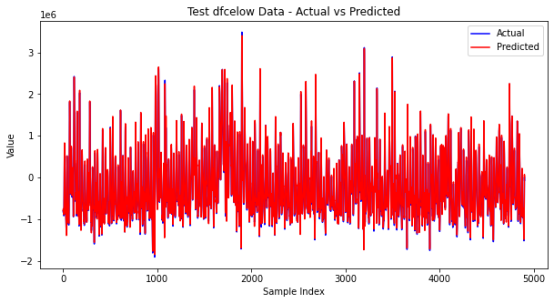



Figure D.6 Comparison of CEEMDAN Low Frequency

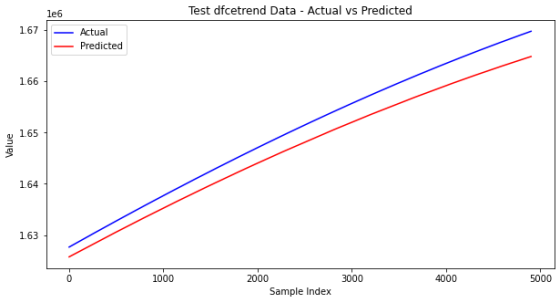



Figure D.7 Comparison of CEEMDAN Trend

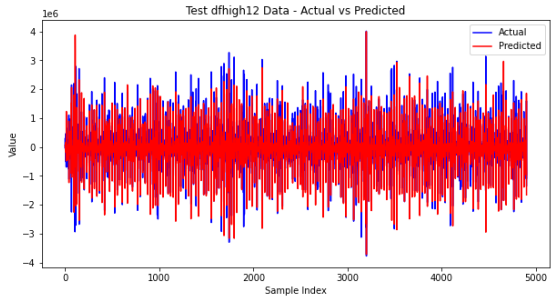



Figure D.8 Comparison of CEEMDAN High Frequency ( IMFs12 )

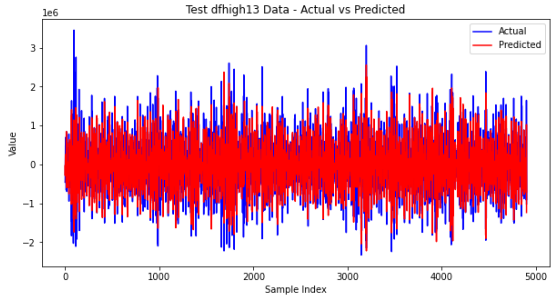



Figure D.9 Comparison of CEEMDAN High Frequency ( IMFs13 )

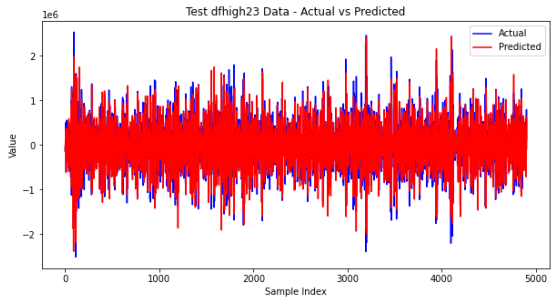



Figure D.10 Comparison of CEEMDAN High Frequency ( IMFs23 )

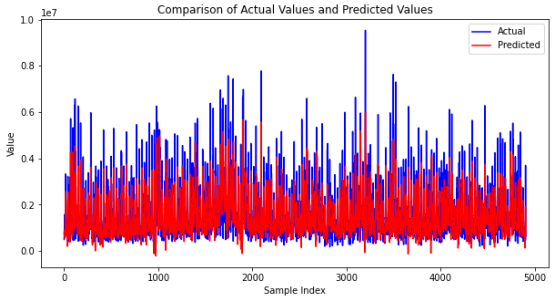



Figure D.11 Comparison of  Refactoring CEEMDAN ( IMFs123 )

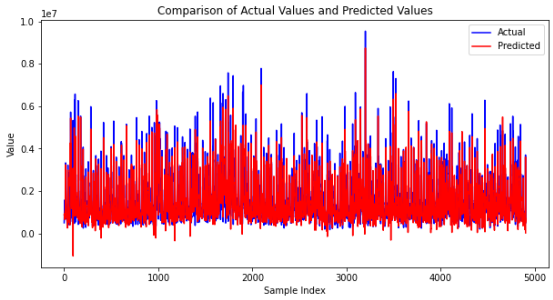



Figure D.12 Comparison of  Refactoring CEEMDAN ( IMFs12 )

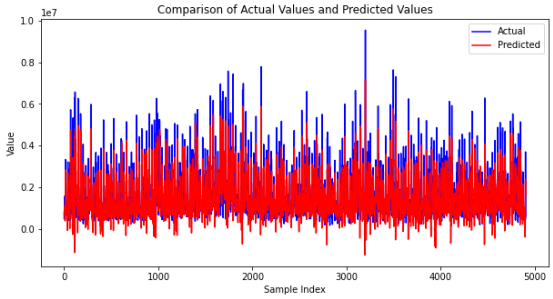



Figure D.13 Comparison of  Refactoring CEEMDAN ( IMFs23 )

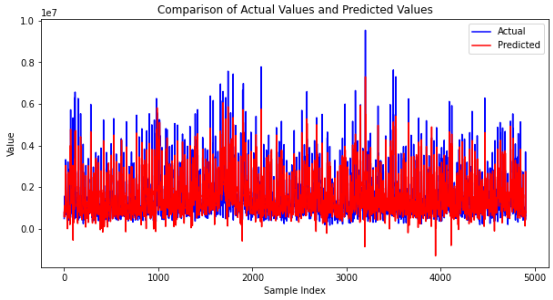



Figure D.14 Comparison of  Refactoring CEEMDAN ( IMFs13 )