**Faculty of Natural, Mathematical and Engineering Sciences**
Department of Informatics


**Name:**

**Student Number:**

**Degree Programme:** MSc Computational Finance

**Project Title:** Evaluation of Discrete Wavelet Transform as a Target Transformation in Time Series Forecasting

**Supervisor:**

**Word Count:**

Plagiarism Statement

All work submitted as part of the requirements for any examination or assessment must be expressed in your own words and incorporates your own ideas and judgements. Plagiarism is the taking and using of another person's thoughts, words, judgements, ideas, etc., as your own without any indication that they are those of another person.

Plagiarism is a serious examination offence. An allegation of plagiarism can result in action being taken under the *B3 Misconduct Regulations*.

I acknowledge that I have read and understood the above information and that the work I am submitting is my own.

**Signature:** **Date:** August 8, 2022

Department of Informatics
King's College London
WC2R 2LS London
United Kingdom

# Evaluation of Discrete Wavelet Transform as a Target Transformation in Time Series Forecasting


Thesis submitted as part of the requirements for the award of the MSc in Computational Finance.
7CCSMPRJ - MSc Individual Project - 2022

# Abstract

In this dissertation, we investigate the impact of target transformation on time series forecasting. Specifically, we investigate a target transformation method based on the Discrete Wavelet Transform. For this study, we use a large-scale time series dataset from the M3 competition. The experiment starts with implementing DWT to transform a native time series and obtain a transformed time series with additional information. We perform the transformation, and then we train a range of regression and statistical models: Elastic Net, Support Vector Regressor, K-Nearest Neighbour Regressor, Random Forest Regressor, Multi-layer Perceptron, and ETS model. The results indicate that target transformation can help improve forecasting performance under specific conditions. More specifically, we observe that the forecasting accuracy is improved with the use of the transformation method for most models on around 50% of the studied time series, whereas for the K-Nearest Neighbours is improved on around 70% of the time series.

# Contents

# List of Figures



# List of Tables

# Abbreviations

| | |
|---|---|
| $\phi$ | Father wavelet |
| $\psi$ | Mother wavelet |
| $f(t)$ | Original time series |
| CWT | Continuous Wavelet Transform |
| DWT | Discrete Wavelet Transform |
| EN | Elastic Net |
| ETS | Error, Trend, Seasonal |
| KNN | K-Nearest Neighbours |
| MLP | Multi-layer Perceptron |
| RF | Random Forest |
| sMAPE | Symmetric Mean Absolute Percentage Error |
| SVR | Linear Support Vector Regressor |
| WT | Wavelet Transform |

# 1 Introduction

Time series data plays a vital role in our daily life: they are used to record the values of a variable of interest obtained through measurements over time. Time series can be used in data analysis in various sectors, such as mathematical finance, weather forecasting, price forecasting, and traffic prediction. Time series forecasting is currently a significant task that aims to establish a model from historical observations capable of analysing the relationship within the data. In the beginning, people assumed the time series include a stationary property[1], a property that can make the forecasting task easier; however, most real-world time series is non-stationary and with volatile variance through time.

In order to improve the results obtained from forecasting, an increasing number of research[2] have sprung up and proposed several statistical and state-of-the art models as well as several mathematical processes transform the time series so that they become more easy to forecast. The STL[3] is a kind of decomposition of seasonality and trend which can extract the informative time series for forecasting. Additionally, transformation is one available approach to remove noisy information or generate helpful details. A Box-Cox transformation[4] is used to transform a skewed distribution of time series into a Gaussian-like distribution, which can lift the accuracy of particular regression models. The decomposition is another approach to degrade the time series data into a set of decomposed subseries with finer information. Discrete Wavelet Transform (DWT) is a technique from signal processing that allows decomposing the time series into high-frequency and low-frequency components and reveals detailed information from subseries. After decomposition, the DWT has the ability to remove specific noises and perform a smoother transformed time series after reconstruction.

With respect to investigating the effectiveness of DWT transformation, the establishment of the heterogeneous models provides competitive performance on forecasting problems and gives the comparison of transformation for each model. The classic forecasting models are widely considered statistical methods for developing the forecasting model. The ETS model is a well-known forecasting model derived from the exponential smoothing model[5] and has the capabilities of describing

the time series with different statistical components. With the rapid development of machine learning techniques, the relevant regression models, such as K-Nearest Neighbor (KNN) Regressor, Support Vector Regressor (SVR), and tree structure regressor, are widely implemented to perform the forecasting tasks. Subsequently, the researchers start using neural networks (NN); for example, multi-layer perceptron (MLP), which has multiple hidden layers and several hidden neurons, can tackle non-linear data.

## 1.1 Project Objectives

In this paper, the aim is to investigate the forecasting performance of target transformation by comparing how models perform when trained on the untransformed and transformed time series. In order to investigate the transformation, we implement the model selection which allows the models to determine a group of hyperparameters with the best performance. Specifically, we anticipate to investigate whether the transformation is beneficial to the forecasting performance and look into what kind of time series and hyperparameters can achieve an improvement with transformation.

For achieving a comprehensive comparison, we use a large-scale time series dataset from M3 competition which has over 3000 time series data. We apply the transformation on the 1043 selected time series from various domains which allows us to perform a generalised comparison.

## 1.2 Report Structure

In this section, we describe the report structure and give a brief introduction to the following chapters. We mention the background theories of the machine learning models and statistical models in Chapter 2. We give the mathematical representations and explain the relevant terms and parameters of each model which are used for establising a model.

Chapter 3 discusses the literature review regarding a variety of transformation approaches, the application of wavelet transform, and the implementation of model

selection. These provide several concepts and ideas in terms of implementing transformation and developing an experiment.

Later, we are going to explain the methodology of DWT in Section 4 and give a detailed explanation of the DWT method. We describe not only the DWT approaches but also some practical details which are worth of attention.

Chapter 5 contains the design pattern of our experiment, such as the procedure of the experiment and hyperparameters' configurations, and the precise description of the entire experiment.

Chapter 6 gives the results of our experiment and the results of our investigation of whether the DWT transformation helps to improve the time series forecasting problems. We do not only give an overall evaluation but also analyse whether the transformation works under specific dimensions.

# 2 Background Theories

In this chapter, we discuss basic concepts relevant to the dissertation. We introduce important models and their mathematical representations.

## 2.1 Models

In order to establish fusion models constructed by WT and various models such as the statistical forecasting model and regression model. The following paragraph discusses the background knowledge and development of the candidate models, which are Elastic Net (EN), Support Vector Machine (SVM), K-Nearest Neighbours (KNN), Random Forest (RF), Multi-Layer Perceptron (MLP), Exponential Smoothing (ETS) and Naive method. The difference between the regression and statistical models is the mechanism of fitting input data. The regression models accept features and target as training data and estimate the target using the feature variables. With respect to the statistical model, the target is the single input variable used to compute the statistical properties of the data and make predictions.

### 2.1.1 Elastic Net

Elastic net[6] is a regulariser based on linear regression with $L_1$ norm and $L_2$ norm penalty, which is believed to perform well than Ridge and Least Absolute Shrinkage and Selection Operator (LASSO).

The general linear regression model given $n$ variables can be defined as follows,

$$y = w_0 + x_1 w_1 + x_2 w_2 + ... + x_n w_n \tag{2.1}$$

Where $w_n$ is the weight which can make up weight vector $w$ and $w_0$ is typically called intercept or bias.

When confronting large numbers of variables, linear regression is not able to provide precise estimations due to overfitting. Introducing regularisation is an idea to reduce the complexity of the linear regression model, and the Ridge and Lasso

are two linear models constraining by $L_2$ norm and $L_1$ norm, respectively. The advantage of Ridge is stopping the model from underfitting and overfitting by reducing the number of coefficients which gives an effective penalty. However, Ridge is unable to shrink the coefficients to zero and can not be used for sparsification. LASSO is capable of reducing plenty of dimensions and selecting variables by setting the coefficients to zero, which Ridge regression can not do. The disadvantage of LASSO is that LASSO can barely accept one variable from a group of variables with high pairwise correlation and ignores the others. The paper[6] proposes a new regularisation technique called Elastic Net, which combines the benefits of Ridge and LASSO and can shrink the coefficients and variable selection. On the other hand, the incapability to handle the non-linear problem is the weakness of the elastic net. The loss function of the elastic net is defined as follows,

$$Loss = \sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda_1||w||_1 + \lambda_2||w||_2^2 \tag{2.2}$$

In order to control the $L_1$ and $L_2$ penalty separately, two hyperparameters, $\alpha$ and $\rho$ are introduced to represent the sum of $\lambda$ and the proportion of $L_1$ penalty. When $\rho = 0$, the elastic net will become Ridge regression. The elastic net will be simplified into LASSO regression when $\rho = 1$. Thus, the minimiser of Equation (2.2) can be written in:

$$\underset{w}{argmin}(\sum_{i=1}^{n}(y_i - w^T x_i)^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2) \tag{2.3}$$

where $\alpha = \lambda_1 + \lambda_2$ and $\rho = \dfrac{\lambda_1}{\lambda_1 + \lambda_2}$

### 2.1.2 Linear Support Vector Regression

Support Vector Machine (SVM)[7, 8] is a kind of supervised learning which is used to estimate a hyperplane to separate data in the classification problems or fit the data points in the regression problems.

A linear model with the weight vector $w$ and bias $b$ has the form:

$$y = w^T x + b \tag{2.4}$$

The changes in the weight vector will affect the distance between the linear model and the decision boundary. The SVM is expected to find an optimal decision boundary by specifying a subset of training data called support vectors. The solution of hard margin SVM is a simple optimisation problem defined by

$$\max_{w,b} \frac{2}{||w||} \rightarrow \min_{w,b} \frac{1}{2}||w||^2 \tag{2.5}$$

such that $y_i(w^T x_i + b) \geq 1$, $\forall i = 1, ..., n$

In soft margin SVM, a variable slack ($\xi$) is introduced to give some tolerance if the high dimensional hyperplane does not exist. Then, the optimisation problem becomes the following equation:

$$\min_{w,b} (\frac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i) \tag{2.6}$$

subject to $\xi_i \geq 0$, $\forall i = 1, ..., n$, and $y_i(w^T x_i + b) \geq 1 - \xi_i$, $\forall i = 1, ..., n$, where $C$ is the amount of slack that is allowed. The slack controls the trade-off between the training errors minimisation and the margin maximisation.

Regression is a generalisation of classification and we interested in $y \in \mathbb{R}$ instead of $y \in \{-1, 1\}$. According to the $\epsilon$-insensitive loss function[9], $|y - f(x)|_\epsilon$, we expect to minimise

$$\min_{w,b} (\frac{1}{2}||w||^2 + C \sum_{i=1}^{n} |y - f(x)|_\epsilon) \tag{2.7}$$

A soft margin Support Vector Regression (SVR) introduces slack to transform into the constrained optimisation problem, which can be written in

$$\min_{w,b}(\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)) \tag{2.8}$$

subject to $f(x_i) - y_i \leq \epsilon + \xi_i$, $y_i - f(x_i) \leq \epsilon + \xi_i^*$, and $\xi_i, \xi_i^* \geq 0 \ \forall i = 1, ..., n$

Then, we can obtain a regression estimate with Lagrange multipliers and kernel $\mathbb{K}$

$$f(x) = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\mathbb{K}(x_i, x) + b \tag{2.9}$$

From the SVM theory, only support vectors provide non-zero Lagrange multipliers $\alpha_i$, which contributes to the solution. The others training samples with zero Lagrange multipliers will not affect the solution. In addition, the support vectors are typically a small subset of training samples which benefits from avoiding overfitting.

### 2.1.3 K-Nearest Neighbours Regression

K nearest neighbour (KNN) is a distance-based non-parametric algorithm. In KNN regression, the K nearest neighbour of the specific data sample is selected by calculating the Euclidean distance between the data sample and the rest of the training data. Then, the output estimation takes an average of those K nearest neighbours' values. The KNN is beneficial for handling both continuous and discrete data values and generates a smoother outcome with a larger K. However, the larger K neighbours may include irrelevant data samples, and the small K introduces enormous noise.

### 2.1.4 Random Forest Regression

Random forest[10] is an ensemble learning method which integrates numbers of basic decision trees and implements bagging aggregation to provide predictions. In the classification problem, the random forest gives the prediction from the

majority of decision tree predictions. With respect to the regression problem, the random forest performs an average estimation from the prediction of each decision tree.

Bootstrapping randomly chooses data samples from the training set, which has the ability to give randomness due to the heterogeneous form of the decision tree. Otherwise, the decision trees do not differ in the forest without bootstrapping. However, a second variation is implemented in a random forest, the number of features considered when executing a split in individual tree nodes.

### 2.1.5 Multi-layer Perceptron

Perceptron is the simplest form of ANN, contains a single neuron and has the ability to handle linear separable problems. A perceptron calculates inputs with weights and applies an activation function at the end. The perceptron can be written by

$$\hat{y} = g(\sum_{i} x_i w_i + b) \tag{2.10}$$

Where $w$ is the weights, $b$ is the intercept, and $x$ is the input variables. $g$ is the activation function that defines the neuron's output and aims at mimicking a biological neuron.

The multi-layer perceptron[11] (MLP) is a fully connected network and is usually known as an ANN. MLP consists of one or more hidden layers and is able to represent a highly non-linear function. An MLP with a single hidden layer is defined as follows:

$$h_j = g(\sum_{i=1}^{N} x_i w_{i,j} + b_i), \forall j \in \{1, ..., h\} \tag{2.11a}$$

$$\hat{y} = \sum_{v=1}^{H} x_j w_j + b_j \tag{2.11b}$$

Where the Equation (2.11a) indicates the result from the input layer with $N$ nodes to the hidden layer with $H$ hidden nodes, the Equation (2.11b) represents the output estimate from the hidden layer to the output layer without applying the activation function.

The training process performs gradient descent to minimise the loss function to find the global minimum. Then, a classic backpropagation is used to adjust the weights by moving backwards, and the weight adjustment is proportional to the error function's value. The error surface of MLP may consist of many local minima, and the backpropagation method may potentially be stuck in the local minimum. Therefore, momentum is introduced to solve this problem, which can roll over the local minima.

### 2.1.6 ETS

ETS[12] is abbreviated by Error, Trend, Seasonal and is a state space model derived from Simple Exponential Smoothing (SES), which gives statistical forecasting. The SES is able to forecast data without the apparent trend and seasonality and is defined by

$$\hat{y}_{T+1|T} = \sum_{n} \alpha(1-\alpha)^n y_{T-n}, \forall n = 0, 1, 2, \cdots \tag{2.12}$$

Where $0 \leq \alpha \leq 1$ is a smoothing parameter, the value of one-step ahead forecasting is the weighted average of the past observations. A larger $\alpha$ gives the observations closer to the current state, the higher weights and the weights on past observations are exponentially decreasing.

Subsequently, Holt et al.[5], and Winter et al.[13] introduce the trends and seasonality components to the SES, allowing forecasting of the data with trends and seasonality. Therefore, the exponential smoothing statistical model is called the state space model and has the ability to describe the changes in level, trend, and season of unseen data through time.

With respect to distinguishing the additive and multiplicative properties of data,

the exponential smoothing model is labelled as ETS by error, trend, and seasonal. In order to represent different kinds of ETS, a set of notations of error, trend, and seasonal are used for describing the statistical model. The error can define an additive and multiplicative model by $A$ and $M$; the trend with the notation of $None$, $A$, and $A_d$ has the ability to describe that there is no trend or additive trend or damped additive trend, respectively; then, $None$, $A$, $M$ are used to represent seasonal components with no seasonality, additive seasonality, and multiplicative seasonality.

### 2.1.7 Naive Forecaster

The naive method is a special case of SES when $\alpha = 1$ in the Equation (2.12) and the one step ahead forecasting can be written in

$$\hat{y}_{T+1|T} = y_T \tag{2.13}$$

This means we can simply use the last observation as the current forecasting value.

# 3 Literature Survey

In time series forecasting, a stationary time series with stable statistical properties has the potential of better predictability; however, most real-life time series are non-stationary. Therefore, finding proper transformations to transform non-stationary time series into stationary time series has been proved as an effective way to enhance time series forecasting performance.

## 3.1 Review of Various Transformations

Salles et al.[14] review a number of different transformation approaches to investigate their capability of improving the time series forecasting performance after transforming time series. The 18 mentioned transformations can be divided into two main categories: mapping and splitting methods. The mapping methods transform time series into different scales by using mathematical processes, such as logarithmic transformation, differencing, Box-Cox transformation[4], and moving average. Most of the mapping methods are parametric, simple to use and to obtain stationary series, and invertible. The splitting methods are usually non-parametric methods non-parametric data-driven methods aiming to infer the time series characteristics. Decomposition is the majority of the splitting method which is used to split a series into a set of subseries with various scales and/or domains. In order to extract information from time series, decomposition methods create high-frequency, low-frequency, and seasonal components, that may be useful in interpreting the data and making predictions. Nowadays, the wavelet transform (WT) and empirical mode decomposition (EMD) are the most popular decomposition techniques, and an increasing number of work have started uses these transformations on time series.

In [14], various methods are used to transform five different time series datasets from various forecasting competitions. The ARMA model is used and it is investigated whether transformations improve the performance of the model. The results indicate that differencing is the best method and performs well on every time series dataset. The splitting decomposition methods, WT and EMD, also provide a competitive improvement. At last, it is concluded that the transfor-

mation plays a significant role in time series forecasting problems and selecting a suitable transformation on specific data is very important.

## 3.2 Applications of WT

Time series forecasting is widely used in several fields, such as renewable energy prediction, electricity price forecasting, traffic volume prediction, and the forecasting of financial products. The following paragraphs review some applications of WT transformation to forecasting tasks that include various statistical and machine learning models.

Qian et al.[15] discuss a number of related works to the wind speed prediction problem, where decomposition-based transformations are used, including the WT and the EMD transformations. In short-term forecasting, this work indicates that using one forecaster with selected features or inputs performs better than developing multiple forecasters for each decomposed series and then reconstructing it back to the original time series. However, it is highlighted that using decomposition-based transformed targets gives better results than using the original time-series to forecast wind speed. In addition, the paper brings up the importance of considering data leakage when implementing the transformation on time series data. The data leaked when doing transformation on "unknown" data samples; thus, it should be avoided and only transform the training data. At the same time, the decomposition level and length of WT are two topics that have not been developed into a recognised standard which is worthy of discussion in future research. Doucoure et al.[16] propose a hybrid model that combines WT with an artificial neural network (ANN) and introduce the Hurst coefficient[17] to analyse the predictability of the decomposed wind speed time series data. The Hurst coefficient is able to distinguish the informative time series from the non-informative ones, and the two lowest Hurst coefficients are used to decide the removal of the noisy decomposed series. However, The results of their experiments show that although the test does not improve the accuracy, it reduces the run time of the experiments. Liu et al.[18] establish a hybrid model constructed by DWT transformation and Support Vector Machine (SVM) and uses a causality test to determine the input variables, such as decomposed time series and lags. Then, a genetic algorithm (GA) is used to esti-

mate the best parameters of the SVM model and eventually prove that the model with DWT transformation is capable of improving the forecasting performance compared with the model without DWT transformation.

Freire et al.[19] investigate the application of DWT and ANN in streamflow prediction problems. The DWT decomposes the series into two parts: the details component (high-frequency), which is considered to be noisy, and the approximations component (low-frequency). The approximation component can be further decomposed, and so on. Their work investigates the performance of various combinations of different levels of approximations as inputs of the ANN model as well as large number of basic wavelets from seven different wavelet families. After experimenting with 54 basic wavelets and 34 combinations of decomposed approximation components, their results indicate that the error measures are significantly improved. The impact of basic wavelets as well as of other fundamental factors , such as decomposition levels and edge effect, on daily and monthly streamflow forecasting tasks is highlighted in [20]. It is noted that the factors can considerably affect the outcome and may provide an improved result or increase errors.

The randomness of financial prices makes it difficult to predict the price directly. A financial forecasting study[21] proposes a stock tendency prediction classifier involving DWT and Extreme Learning Machine (ELM) to investigate how the DWT denoising technique affects the forecasting performance with daily financial data. The paper indicates that short-term stock prices contain a large amount of white noise, significantly affecting the model's performance. Thus, eliminating the noise by a threshold may enhance the forecasting accuracy. Their results show that the hybrid model achieves better results compared to the pure ELM model. A couple of other studies[22, 23] for forecasting daily stock prices experiment with a hybrid model of DWT and ANN. They propose different methods for denoising; the first one removes all of the detail components, whereas the second one keeps the detail component obtained from one level of decomposition. It is believed that the aforementioned component may contain information useful for short-term predictions. Furthermore, two other works[24, 25] based on DWT and ANN introduce Stacked Autoencoders (SAEs) and Stepwise Regression-Correlation Selection (SRCS) approaches based on DWT and ANN hybrid models and show that they are able to

select effective candidate features.

Finally, hybrid models using WT are used for electricity price predictions. Conejo et al.[26] propose a hybrid model that couples WT with ARIMA and investigate whether it can improve the predictions of Spanish electricity prices. It is shown that the model is able to provide better prediction than the naive method and the pure Arima model. Tan et al.[27] establishes a wavelet, GARCH, ARIMA mixture model which GARCH has better ability to capture high volatile of detail decomposed time series and the GARCH-ARIMA mixture model is used to predict the approximation component. The results show that the mixture model has ability to perform better than ARIMA model. Considering the non-linear and non-stationary characteristics of electricity price, Yang et al. and Li et al.[28, 29] introduce a hybrid model combining WT, ARMA, and kernel ELM and the ARMA model and kernel ELM have the ability to forecast the stationary and non-stationary time series, respectively. The kernel ELM achieves a faster learning rate when training the model which may be one of the reasons that the hybrid model reduces more errors than the others hybrid models.

## 3.3 Machine Learning Model Comparison

Ahmed et al.[30] establishes a pipeline for time series regression and forecasting problems in order to compare the performance among several machine learning algorithms with large numbers of time series data. The large scale dataset from M3 competition includes more than one thousand monthly time series data from various sectors which is considered that having ability to evaluate the generalisation of machine learning models. Regarding the data prepossessing, they suggest three kinds of transformations: log transformation, deseasonalisation, and scaling. The transformations are applied to the data in the order mentioned above. After the transformations, the lagged targets, differencing value or moving average are considered as input features of the regression model generated by the transformed time series. In addition, they use horizon equal to one, which means they use lagged targets to predict the next step. With respect to selecting the best model, two types of hyperparameters are considered: the input features, and parameters defining the complexity of the model. For instance, the input features can be

the different numbers of lags, and the number of hidden layers and hidden units are parameters that define the complexity of the multi-layer perceptron or neural network. Different datasets have their own scales which does not allow the comparison of the typical error measures, such as mean square error or mean absolute error. The authors propose three measures which can be used to compare different time series data and different machine learning models. One of the measure is the symmetric mean absolute percentage error (sMAPE)[31] that computes the percentage error which can distinguish the performance among heterogeneous time series. Ranking and fraction-best that are be used to compare the rank of each model and calculate the proportion of time series in M3 that a model has the best performance. The conclusion shows that the multi-layer perceptron works well with lagged targets and moving average as input features and the Gaussian Process is the most robust model.

# 4 Methodology

Target transformation refers to applying transformation techniques to the target data. A variety of methods can be utilised to transform the target data mentioned in the previous Section 3.1. We implement discrete wavelet transform to transform time series data. The following sections introduce the scientific theory of DWT and the usage of transforming time series by decomposition, thresholding, and reconstruction.

Two issues regarding data leakage and inverse transformation should be carefully contemplated. The target transformation can easily cause the problem of leaking information from "unseen" data if not carefully used, and this will be discussed in the following sections. Another notable issue is regarding the order of transformations and inverse transformations. The transformed data is required to execute a back-transform by inverting the order of transformation done in the beginning.

## 4.1 Wavelet Transform

The wavelet theory[32, 33, 34] is a mathematical transformation that indicates an oscillating mother wavelet can represent a signal with limited length. The wavelet transform (WT) is derived from the Fourier transform (FT) and has broadened advantages over FT. FT can transform functions into frequency domain but can not locate the information in different time scales. The WT improves this weakness and allows to perform timescale representation in the time-frequency domain, which can be used for analysing non-stationary time series. The WT have two approaches: continuous wavelet transform(CWT) and discrete wavelet transform(DWT).

The CWT calculates and decomposes a signal $x(t)$ with a mother wavelet $\psi$, which can be translated and scaled on the time-frequency domain. The CWT can be represented by the Equation (4.1) where $a$ and $b$ are two characteristics regarding frequency and time, which can indicate the mother wavelet's dilation and translation. In the CWT, $a$ is positive and $a$ and $b$ are continuous values. However, the demerits of CWT are time-consuming and computational expensive because

of the decomposition on every scale and excessive information generated from the process. With the limited data samples and constrained resources, DWT performs more efficiently than CWT and is still able to provide precise wavelet analysis.

$$CWT(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\psi_{a,b}(\frac{t-b}{a}) \tag{4.1}$$

## 4.2 Discrete Wavelet Transform

The procedure of DWT[35, 36] typically consists of three phases, decomposing a signal into a time-frequency domain, thresholding of decomposed signal, and reconstructing the reduced signal to the original domain. In the following subsections, we are going to give more details for each of the points.

### 4.2.1 Wavelet Family

The basic wavelets are categorised into a variety of families with different kinds of mathematical properties. With different wavelets, the DWT transformation potentially achieves an obvious difference in signal representation and performs different smoothing outcomes with various magnitudes. Lahmiri et al.[23] indicate that several wavelets are widely used for forecasting financial time series. Accordingly, Haar and Daubechies are two wavelets from different wavelet families and are the most popular wavelets for DWT. The Coiflets and Symmlet are other wavelets also commonly applied for DWT. The following Figure 1 gives examples of 7 wavelets: haar, db5, db6, db19, db20, coif4, and coif5 from 3 wavelet families.

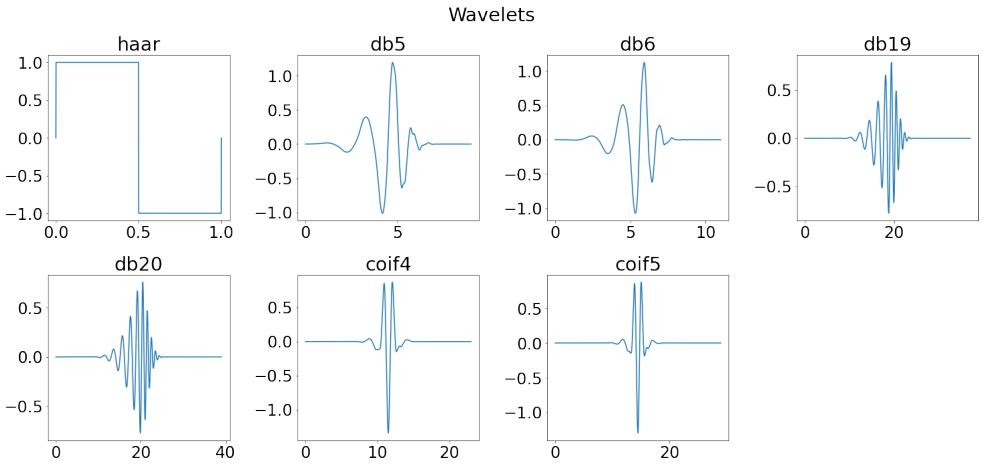


Figure 1: Example of Wavelets

### 4.2.2 Decomposition

The original signal $f(t)$ is decomposed into two components of high and low frequency which can be described by the mother and father wavelet. The mother and father wavelet is defined in Equation (4.3) and (4.2). The high-frequency component is also called detail coefficient $d$, which is written in Equation (4.5) and the low-frequency component $a$ is a smoother raw signal whose terminology is approximation coefficient represented in Equation (4.4).

$$\phi_{j,k}(t) = 2^{-\frac{j}{2}}\phi(2^{-j}t - k) \tag{4.2}$$

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}}\psi(2^{-j}t - k) \tag{4.3}$$

where $j = 1, ..., J$ represents the $j$ level decomposition and the mother and father wavelets satisfy $\int \psi(t)dt = 0$ and $\int \phi(t)dt = 1$.

$$a_{J,k} = \int \phi_{J,k} f(t) dt \tag{4.4}$$

$$d_{j,k} = \int \psi_{j,k} f(t) dt, \text{ where } j = 1, ..., J \tag{4.5}$$

The DWT decomposition convolves the original time series $f(t)$ into approximation coefficient $a$ and detail coefficient $d$ via a low-pass filter and a high-pass filter. The high-pass and low-pass filters only accept high-frequency and low-frequency components. The filter down-sample the time series and generate a new coefficient with half of the time series length in the previous decomposition level. According to the DWT theory, the decomposition process keeps downgrading the approximation coefficient in each decomposition level until the maximum decomposition level[37]. The Equation (4.6) of maximum decomposition level $J$ is defined by the length of time series $N$ and the length of wavelet filter $F$.

$$J = \log_2(\frac{N}{F-1}) \tag{4.6}$$

Consequently, the orthogonal wavelet $f(t)$ is defined by Equation (4.7) and the simple form of $f(t)$ can be written in Equation (4.8).

$$f(t) = \sum_k a_{J,k}\phi_{J,k}(t) + \sum_k d_{J,k}\psi_{J,k}(t) + \sum_k d_{J-1,k}\psi_{J-1,k}(t) + ... + \sum_k d_{1,k}\psi_{1,k}(t) \tag{4.7}$$

$$f(t) = A_J(t) + D_J(t) + D_{J-1}(t) + ... + D_1(t) \tag{4.8}$$

### 4.2.3 Thresholding

Most commonly, the detail coefficients are plenty of redundant information that is harmful to giving effective analysis. The thresholding method proposes a threshold computed by the detail coefficient in the first decomposition level. A universal

threshold[38, 36, 39] $t$ is defined by the following Equation (4.9).

$$\begin{aligned} threshold &= \sigma\sqrt{2\ln N} \\ s &\approx \frac{median(|x_i|)}{0.6745} \end{aligned} \tag{4.9}$$

Where $N$ is the length of data and $\sigma$ is the standard deviation of the noise. However, the $\sigma$ is usually unknown in most of the data, but it can be substituted by the noise estimation[40] $s$, which is calculated from the first decomposed detail coefficient $x_i$.

With the determined threshold, a thresholding method[38] is available for the reduction of detail coefficients. The three continuous thresholding methods are Hard, Soft, and Garrote. The hard thresholding method is defined in Equation (4.10a) and only if the coefficient is greater than the threshold will be kept in the low-frequency components. The soft thresholding in Equation (4.10b) removes small-scale coefficients like hard thresholding but shrinks the coefficients with large size. The Garrote thresholding from Equation (4.10c) is a trade-off between hard and soft thresholding; however, this approach does not shrink the large magnitude coefficients as much as soft thresholding.

$$x_i^* = \begin{cases} 0 & \text{if } |x_i| \leq \text{threshold} \\ x_i & \text{if } |x_i| > \text{threshold} \end{cases} \tag{4.10a}$$

$$x_i^* = \begin{cases} 0 & \text{if } |x_i| \leq \text{threshold} \\ sign(x_i)(|x_i| - t) & \text{if } |x_i| > \text{threshold} \end{cases} \tag{4.10b}$$

$$x_i^* = \begin{cases} 0 & \text{if } |x_i| \leq \text{threshold} \\ x_i - \frac{t^2}{x_i} & \text{if } |x_i| > \text{threshold} \end{cases} \tag{4.10c}$$

### 4.2.4 Reconstruction

The reconstruction of inverse DWT starts from the last decomposition level, and the Equation is mentioned in (4.7). The approximation and detail coefficients of the maximum decomposition level are up-sampled by the low-pass and high-pass filters independently to reconstruct an approximation coefficient into a higher decomposition level. Then, repeat the process for each decomposition level until the native time series are reconstructed. Figure 2 gives an example of applying DWT transformation with a Daubechies wavelet on one of the M3 time series, where the red dotted line represents the transformed time series which shows a smoother plot than the original time series.

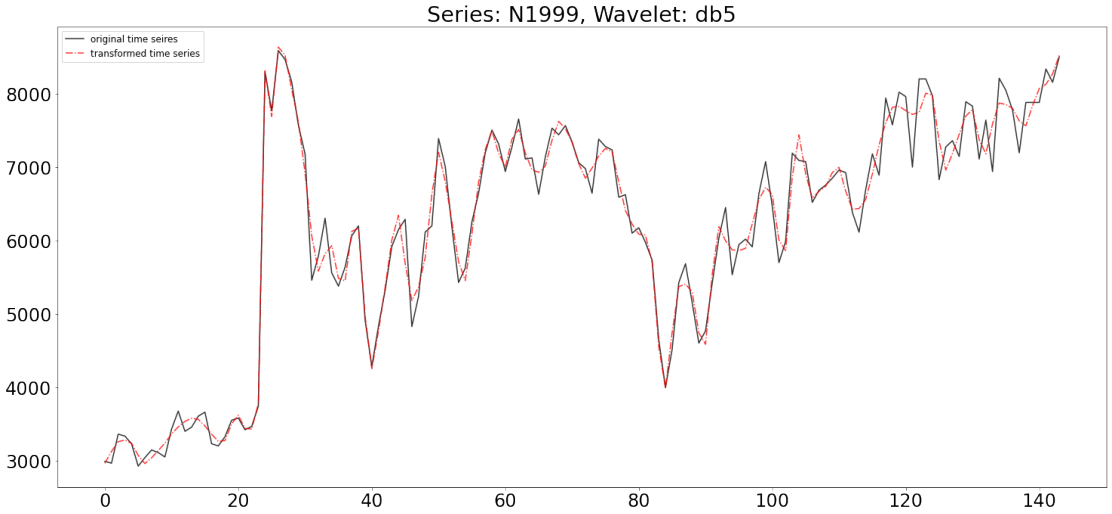


Figure 2: Example of applying DWT with wavelet db5

## 4.3 DWT for Target Transformation

The approach [23] (Figure 3) starts with the same procedure to decompose the time series with a determined wavelet and decomposition level. Initially, we decompose the time series with a selected wavelet and receive an approximation part and detail part. Then, we keep decomposing the approximation part with the same wavelet until the defined decomposition level. The level of decomposition is determined by the wavelet and the length of the original time series. The subseries

$\{D_1, D_2, ...D_n, A_n\}$ are that we obtain from the complete decomposition. However, many times of decomposition generates more subseries and leads to obtaining too much information. Consideration of fixing the maximum decomposition level as a small number may be reasonable. In our experiment, we set the maximum decomposition level equal to 1. Subsequently, we apply a hard thresholding method with a universal threshold for every detail component, which strictly replaces the value less than the threshold by zero, and reconstructs each subseries by the residual data. Then, we acquire a set of subseries with the same length as the original time series and are feasible to make these subseries as the features of the target time series. Finally, the summation of the reconstructed subseries is available because of the additive property of wavelet decomposition.

Furthermore, there are a variety of other approaches we anticipate giving experiments. One of the approach[19] proposes a method to reconstruct every approximation component of different decomposition levels as inputs of a neural network. Instead of giving an entire combination, create many combinations from a subset of approximation components and make the models to decide the one with the best performance. The other approach is a basic and widespread method which obtains a group of subseries and distributes them into independent models. Those models can be identical or hybrid structures, such as a statistical ARIMA model and artificial neural network. Then, we combine every prediction as the final prediction. However, this approach is computationally expensive and time-consuming due to the multiple time series when training the models separately.

## 4.4 Data Leakage

Data leakage[41] in machine learning is not widely concerned in recent years, but the importance can not be ignored, especially in time series datasets, graphs and sound problems. Data leakage creates an overly optimistic model with unbelievable, extraordinary results leading to the incapability of future forecasting and production. In our experiment, data leakage can potentially happen when preparing the training data; there are scaling, transformation and feature generation.

The data leakage of scaling happens when we scale the entire dataset instead of

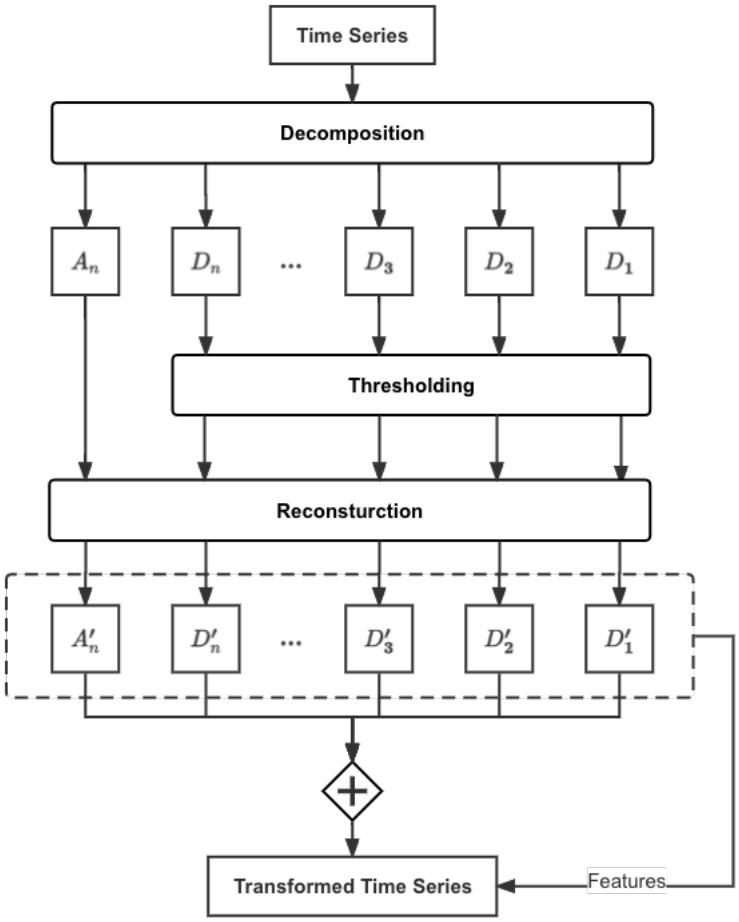


Figure 3: Approach of DWT decomposition

the training set because the presence of extreme values in the testing set will affect the scaling of the training data. Thus, we should establish the scalar from training data and apply it for testing data to avoid data leakage. When it comes to transformation, data leakage exists because of the mathematical process of DWT decomposition. The current value of the transformed time series is determined by its past and future observation. The presence of future observations changes the past observations, and the implicit information we should not receive is leaked to the models. The earlier trial contains data leakage, and the models are able to forecast perfectly with barely one future observation. The last example of data leakage is regarding feature generation. The datasets from the external resource

sometimes lack detailed descriptions, making it difficult for the data practitioners and researchers to examine whether the data contain leakage. The time series data from M-competition are pure time series without any feature, and we only focus on the features we generate. First, we examine whether the transformed time series and decomposed subseries contain data leakage. Then, checking the features generated from the transformed time series and decomposed subseries to secure the substitution of features are entirely correct.

To conclude, the data leakage gives a highly positive result without people's concern and leads to generating an unusable model. In order to avoid data leakage within our experiment, we carefully check every part when experimenting, such as transformation and model validation. Confirming a faith experiment without data leakage allows us to reach the truth of the target transformation.

# 5 Design and Implementation

As mentioned earlier, the goal of the proposed experiment is to investigate whether applying target transformation on time series data can improve the accuracy achieved in forecasting tasks. In this chapter, we give the details of the design and implementation of the experiment.

## 5.1 Overview

We design and implement our experiment in the following steps: transformation, hyperparameter tuning, and model validation. The process starts by transforming the target time series and fitting the model with the training set and evaluating the validation set to select the best hyperparameter combination. When the best hyperparameters are selected, we are able to refit the model with a complete training set and analyse the forecasting results with the testing set. The pseudocode of the procedure is given in Algorithm 1.

**Algorithm 1** Target Transformation

$L, W, M \leftarrow lags, wavelets, model\ parameters$
$n \leftarrow length\ of\ validation\ set$
$TS \leftarrow time\ series$
$DWT \leftarrow DWT\ transformation$
$F \leftarrow Forecasting\ model$
$E \leftarrow Error\ measure$
**for** $c \in Combination(L, W, M)$ **do**
  **for** $i \in \{0, 1, 2, \cdots, n-1\}$ **do**
    $y_{train}, X_{train}, y_{val}, X_{val} \leftarrow DWT(TS_i)$
    $F_{c,trained} \leftarrow F_c(y_{train}, X_{train})$
    $Fitness_i \leftarrow E(y_{val}, F_{c,trained}(X_{val}))$
  **end for**
  $Fitness(F_c) = Average(Fitness_I)$, where $I = \{0, 1, 2, \cdots, n-1\}$
**end for**
$F_{best}$ is determined by $maximum(Fitness(F_C))$,
where $C = Combination(L, W, M)$

## 5.2 Dataset

Time series data plays a vital role in our daily life, recording the changes when time moves and the observations are ordered. We use the time series data from M3 competition[31] as benchmark data. The M3 dataset contains 3003 time series with a variety of frequencies, such as yearly, quarterly, and monthly. The International Journal of Forecasting holds the M-competition, and the data sources are from various domains, such as micro, macro, finance, industry, demographic, and others. In the experiment, we use the monthly time series with a data length greater than 100; the data length ranges between 104 and 144. Table 1 shows how the time series used in our experiment, i.e., the monthly time series with a length greater than 100, are divided into different categories.

| | **Micro** | **Macro** | **Finance** | **Industry** | **Demographic** | **Other** | **Total** |
|---|---|---|---|---|---|---|---|
| **Season** | 178 | 238 | 100 | 252 | 67 | 1 | 836 |
| **No Season** | 19 | 62 | 21 | 81 | 23 | 1 | 207 |
| **Stationary** | 8 | 3 | 0 | 38 | 4 | 0 | 53 |
| **Non Stationary** | 189 | 297 | 121 | 295 | 86 | 2 | 990 |
| **Total** | 197 | 300 | 121 | 333 | 90 | 2 | 1043 |

Table 1: Analysis of M3 monthly data

In addition, we perform two kinds of hypothesis tests to determine whether the time series is stationary and/or contains seasonal components. The seasonality test[42] is provided by the M-competition, which is used to test the seasonality in time series. Regarding stationarity, Augmented Dickey Fuller(ADF)[43] test and Kwiatkowski-Phillips-Schmidt-Shin(KPSS)[44] test whether the time series is stationary. ADF test is unable to reject the null hypothesis when the time series is non-stationary. KPSS test is opposite to the ADF test and rejects the null hypothesis when the time series is non-stationary. It is reasonable to apply both statistical tests to confirm the stationary condition; otherwise, the time series can only have a weak stationary condition when one of the tests performs stationary, and the other gives the opposite outcome. If the ADF and KPSS both give a stationary result, the time series can be concluded as stationary. If both stationarity tests conclude a non-stationary time series, the time series is non-stationary. Interestingly, then the time series is trend stationary when both ADF and KPSS

are not capable of rejecting the hypothesis and conclude that the time series is non-stationary and stationary, respectively. On the contrary, when both ADF and KPSS tests reject the hypothesis, the time series is a differencing stationarity.

## 5.3 Data Splitting

When it comes to model training, we have to split data into training, validating, and testing sets. First, we keep the last 20 time series data points as the testing and training sets take the rest of the data points. Subsequently, we split the training data and remain the last 20 data points for model validation. For instance, a time series with a length of 126 will be divided into 86, 20, and 20, which are training, validating, and testing set, respectively. The training and validating parts are used to discover the best-performed model. Then, combining training and validating parts as an original training set and the model performance is evaluated by these larger training and testing sets.

## 5.4 Hyperparameter Determination

The hyperparameters provide a search space for the experiment where we explore and select the parameters that work better for each model given the data. In our experiment, the hyperparameters can be separated into two parts and represent two kinds of parameters. The first group of hyperparameters is the number of lagged targets and the type of the basic wavelet of DWT. The other group of hyperparameters are model-specific parameters and are used to control the model complexity.

### 5.4.1 Hyperparameter of Wavelet Types and Lags

When wavelet decomposition is used, several parameters can be used to control the decomposition and may lead to different decompositions; for instance, the choices of basic wavelet, different thresholds, heterogeneous thresholding methods, and decomposition levels. In our experiment, we decide to tune several basic wavelets from certain wavelet families: Haar, Daubechies(db), and Coiflets(coif). The different forms of these basic wavelets are capable of performing different

representations of transformed time series and providing different information from decomposed components.

As described above, in our experiment, we use time series from the M3 competition that is monthly data. That is, the interval between two consecutive points is one month, and a group of twelve adjacent data samples represents an entire year. Therefore, we suggest lagged target is equal to 1 or 12 to investigate whether the past one month or the whole year has the ability to provide further helpful information. The following Matrix (5.1) gives an example of generated features with lag 12, where $x_i$ represents the lagged value at time $i$ and $y_i$ indicates the target value at time $i$.

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_{12} & y_{13} \\ x_2 & x_3 & \cdots & x_{13} & y_{14} \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n-12} & x_{n-11} & \cdots & x_{n-1} & y_n \end{pmatrix} \tag{5.1}$$

### 5.4.2 Hyperparameter of Model

The models we have established in the experiment include machine learning regression models, a statistical model, and the naive method. There are elastic net, linear support vector regression, k-nearest neighbours regression, random forest regression, multi-layer perceptron, ETS, and naive forecaster. Different models have background theories and require specific parameters to develop the model. For some models that require many parameters to build a trained model, we fix specific parameters which are not hugely significant to reduce the run time but select the tunable parameters that can vastly affect the training process and generated results. On the other hand, some of the models bring their regularisation parameters capable of decreasing the complexity of the model. These parameters change the models crucially and should be considered the priority.

The following bullet points describe and determine the model's parameters setting. The usage of models are from two *Python* packages *scikit learn*[45] and *sktime*[46].

The non-mentioned parameters are set as defaults from the Python packages.

- Elastic net is derived from the linear model and benefits from the Ridge and Lasso with the advantages of regularisation, $L_1$ norm and $L_2$ norm. From the Equation (2.3), the $\alpha$ and $\rho$ are two parameters that can determine the regularisation of the elastic net. The $\alpha$ decides the strength of the penalty, and the $\rho$ provides the ratio of $L_1$ norm regularisation to $L_2$ norm regularisation. We set $\alpha = [0.01, 0.1, 1, 10]$ and $\rho = [0.1, 0.3, 0.5, 0.7, 0.9]$ in our experiment. Furthermore, we let the model randomly update the coefficients in each iteration with a higher optimisation tolerance of 0.001, which can decrease the model's run time.

- Linear SVR is a support vector machine in regression using the linear kernel. The key parameters from Equation (2.7) are $\epsilon$ in the $\epsilon$-insensitive loss function[9] and the inverse of regularisation term $C$. The parameter $\epsilon$ is the radius from the model we want to fit the data, and $C$ is used to penalise the model for deciding the margin. In our experiment, we have the $\epsilon$ from 0 to 0.8 with step 0.4 and the $C = [1, 5, 10, 20]$.

- KNN regression is a generalisation of the KNN classifier. The prediction of KNN regression is estimated by interpolating the targets with the nearest neighbours. Therefore, the number of neighbours $N$ is the tunable parameter in our experiment, which is considered as [2, 4, 8, 12, 16, 20]. In addition, the model is allowed to decide on an algorithm to compute the nearest neighbours[1].

- Random forest regression is a generalised method compared to the random forest classifier. Probst et al.[47] investigate the parameter importance of RF, which can significantly influence the model performance and run time. Concerning RF regression, the number of features is considered the most important parameter that has the ability to improve the model performance. Moreover, enough decision trees can give a more accurate prediction, and the larger node size can exponentially reduce the run time. From our experiment

[1]The algorithm of searching the nearest neighbours are BallTree, KDTree and brute-force search.

regarding the regression problem, the tunable parameters are the number of trees and the number of candidate features, which are [50, 100] and [0.3, 1.0, sqrt], respectively. The fixed parameters are the node size equal to 5, which is believed to have the ability to accelerate the process and not bootstrap because of the nature of the time series.

- Multi-layer perceptron is stacked by several neural layers with a determined number of neurons. The minimum number of layers of MLP is three, which are the input, hidden, and output layers. The complexity of MLP can be easily increased by adding additional hidden layers and hidden neurons. We propose different combination of the hidden layers and neurons which are [(8, 4, 2, ), (4, 2, ), (8, 4, ), (4, ), (8, ), ]. The length of each tuple in the list represents the number of layers, and the value inside the tuple indicates the number of neurons. The activation function applied to the hidden layer is another important tunable parameter: logistic, relu, tanh, and identity function. In terms of the fixed parameters, we do not allow the model to shuffle the data and give a larger number of iterations due to the characteristics of time series and avoidance of not converging.

- ETS is an exponential smoothing state space model[12] with various representations, and the respective parameters are Error, Trend, and Seasonal. In a ETS model, the tunable parameters and their variables are as follows: Error = [additive, multiplicative], Trend = [none, additive, damped additive], and Seasonal = [none, additive, multiplicative]. We eventually obtain a tuning strategy from these tunable parameters; however, the variable of multiplicative trend usually performs worse and can be considered removed from the searching space.

- Naive forecaster is a naive method to make the last observation as the future prediction. Therefore, we make a strategy for a Naive forecaster with a fixed parameter [last], and the model will always follow this naive method.

## 5.5 Data Preprocessing

Data preprocessing is generally a time-consuming but essential procedure. A better-preprocessed data has the ability to provide informative features for model training and may lead to better predictions. The non-stationary time series may include linear or exponential trend, seasonality, and oscillating pattern. In order to obtain a stationary time series, we introduce the following methods to eliminate trivial information.

- Detrending
- Deseasonalisation
- Denoising

### 5.5.1 Detrending and Deseasonalisation

We are fitting a least squares polynomial with degree 1 and removing the trend by returned polynomial coefficients for the detrending. In terms of removing the seasonality, we use a seasonality test to find the seasonal pattern in the time series. The seasonality test is based on calculating the 12-month lag with 90% confidence level hypothesis[4] to test whether the seasonality exists in time series. If the seasonality exists, we compute the average seasonality of each month and deduct them from the time series by their index.

### 5.5.2 Transformation and Denoising

Most commonly, many irregular patterns and fluctuations within the time series data may seriously influence the forecasting performance. As we know, WT is famous for its capability of denoising. In our experiment, we perform seven wavelet: *haar*, *db5*, *db6*, *db19*, *db20*, *coif4*, *coif5* from three most popular wavelet families which can be seen in Figure 1. The experimental models have the ability to select the wavelets to achieve finer performance and implement forecasting with lower errors. The DWT is able to decompose original time series into high-frequency and low-frequency components with heterogeneous wavelets in the time-frequency domain. The thresholding method proposes the substitution with a universal thresh-

old, allowing us to adjust the magnitude of the noise we expect to eliminate. Then, the decomposed subseries can be reconstructed into the original domain without trivial noise. Moreover, the DWT is not only a method to eliminate noise but also a decomposition to provide informative features. We are able to extract the information from the time series by decomposition and implement the denoising technique at the same time.

## 5.6 Horizon

The horizon of the time series forecasting must be declared clearly to determine the forecasting problems. In order to investigate the forecasting problems in different topics and domains, the number of future observations we expect to forecast at once is significant. Accordingly, the length of time into the future, namely *horizon*. Heterogeneous forecasting problems require a different configuration of the horizon. For example, regarding daily data, the traffic volumes prediction for the next ten days indicates the horizon is equal to 10, and the next day's stock price prediction is one-step ahead forecasting with horizon 1. In our experiment, we assume horizon as 1 all the time in order to implement the one-step ahead forecasting for every time series from various domains.

## 5.7 Feature Generation

The heterogeneous mechanism of regression models and statistical models accepts different formats of input variables. However, regression models do not take into account the order of the samples. To achieve that, we can create new features that will include the lagged values of the target. As statistical models do take the order of samples into account, there is no need to give the lagged values as extra features. Therefore, we train the statistical models for one input feature, the training part of the one dimensional time series without extra generated features. On the other hand, the regression models require additional features as input variables used to estimate the value of the target.

In feature generation, it is mainly associated with the autocorrelation of lags and the decomposition of DWT. From the transformation approaches mentioned in

Section 4.3, we propose two DWT approaches and generate two kinds of features for regression models, which are lagged target and decomposed time series components. The determination of lag's hyperparameter indicates the number of lagged targets used as the feature. We select the $N$ previous observations from current state at time $t$ to create a set of lagged values $\{v_{t-N}, ..., v_{t-1}\}$ and expand $N$ columns of feature. In our implementation with given lag $\in \{1, 12\}$, we provide regression model choices to choose the informative lag, which can give beneficial improvements. The second transformation approach concatenates further information of decomposed high-frequency and low-frequency components into input variables and expects to achieve an improved outcome. The down-sampled technique of high-pass and low-pass filters leads to the inequivalent length between decomposed components and the original time series. It requires the reconstruction of each subseries before integrating it into the features matrix. Eventually, the expansion of the decomposed components depends on the maximum decomposition level $J$, and the increase of the input variable is $J + 1$.

To conclude, the input variable of statistical models is original time series without any feature. The input regression features without the transformation and with the transformation include $N$ and $N + J + 1$ features, respectively.

## 5.8 Model Selection

With the prepared time series datasets, we establish the forecasting models and implement hyperparameter tuning for exploring a set of optimal parameters from the searching space. This allows us to evaluate the fitness of the trained model with determined hyperparameters. The procedure of model selection can be split into two components: model retraining and model validation.

The design pattern of the experiment includes retraining training data with new observations and calculating a fitness score. Subsequently, repeating the same process with different combinations of hyperparameters and obtaining a set of scores about the models' fitness. The best-performed parameters are used to set up the models for evaluation by testing data.

### 5.8.1 Expanding Window

When validating the model with the training and validation set, we implement an approach, namely *expanding window*. The idea is that expand the training data with the validation/testing data after each validation/evaluation. Then, the model is fitted with the expanded training data and validated by the next validation sample. This aims to simulate real-world forecasting by keeping updating the new observation received. For instance, in an one step ahead forecasting, we start to fit the model with training set $T = \{t_0, ..., t_9\}$ and validation set $v_0$. Afterwards, the training set becomes $T = \{t_0, ..., t_{10}\}$ where $t_{10}$ is the validation data $v_0$ from previous iteration and validation set becomes $v_1$. Finally, repeat the procedure until finishing every validation data.

### 5.8.2 Validation

When performing the model selection with a combination of hyperparameters, we apply the expanding window to simulate real-world conditions and obtain a score of fitness from each iteration. It should be noted that the inverse transformation is considered before calculating a score. In our experiment, the implementation of detrending and deseasonalisation should be inverted before evaluation; however, the DWT transformation does not need to be inverted as it does not change the scale of the target.

Hence, we get a list of validation scores when the expanding window dilates, and these scores are evaluated by each expanded training set and validation set. We have the average validation score for each combination of hyperparameters, and we select the ones with the best score. After finishing hyperparameter tuning through model selection, the optimal hyperparameters are used to build the best model and to get the best score for the testing set.

## 5.9 Error and Performance Measure

The M3 dataset contains thousands of time series from heterogeneous domains with different scales, making the time series not comparable. Therefore, the most commonly used error measures, such as root mean square error and mean abso-

lute error, are not considered. Although these error measures are not feasible for comparing different time series, they still have the ability to analyse the model performance and the improvement regarding applying transformation in a particular time series.

The major error measure implemented in the experiment is the symmetric mean absolute percentage error (sMAPE) defined in Equation (5.2). The sMAPE is based on the percentage error and has the ability to compare different datasets on various scales. The contribution of sMAPE gives an insight into the comparison of the time series with and without transformation. With the sMAPE, we can investigate the overall improvement of applying transformation on a variety of time series.

$$sMAPE = \frac{2}{N}\sum_{n=1}^{N}\frac{|y_n - \hat{y_n}|}{|y_n| + |\hat{y_n}|} \tag{5.2}$$

Where $y_n$ is actual value and $\hat{y_n}$ is forecasting value.

With respect to the performance measure, we introduce the fraction-best [30] measure, which is used for counting the proportion of a model's capability of ranking at first place in each time series forecasting. The fraction-best can give the comparison between untransformed and transformed time series and allows us to investigate whether a particular model prefers untransformed or transformed datasets.

# 6 Results and Evaluation

In this chapter, we are going to present the results of our experiment in several directions: improvement between untransformed and transformed time series and wavelet selection. In order to discuss the performance of our transformation, we analyse the comprehensive comparison between untransformed and transformed time series. Then, conducting the analysis regarding whether the transformation can improve specific domains. In the end, we are talking about wavelet selection and investigating the preference of wavelets from untransformed time series to transformed time series. With respect to the preprocessing methods, detrend and deseasonalisation, we analyse the results within and without preprocessing techniques, respectively.

## 6.1 Analysis with Original Time Series

To begin with, the original time series is directly used for forecasting without preprocessing technique, and we call it the ORIGIN method in the following paragraphs. We calculate the sMAPE of the test set for every model and obtain 1043 of sMAPE from the M-competition time series data. Then, computing the average and the standard deviation of these sMAPE for each model will help us investigate the amount of error and the robustness of the models. We compare the improvement of the mean sMAPE and standard deviation of each model between the untransformed and transformed time series, and the comparison is shown in Table 2. The terminology of the *TRANS* and the *UNTRANS* represent whether we apply DWT transformation on the time series or not.

| | | EN | SVR | KNN | RF | MLP | ETS |
|---|---|---|---|---|---|---|---|
| **Mean** | **UNTRANS** | 8.49 | 11.31 | 9.68 | 9.03 | 15.41 | 8.92 |
| | **TRANS** | 9.13 | 11.35 | 9.42 | 9.00 | 16.61 | 8.90 |
| | **Improvement** | -7.59 | -0.39 | **2.63** | **0.30** | -7.78 | **0.27** |
| **Standard Deviation** | **UNTRANS** | 10.77 | 14.98 | 10.84 | 10.51 | 17.52 | 11.82 |
| | **TRANS** | 11.46 | 14.56 | 10.60 | 10.19 | 18.79 | 12.01 |
| | **Improvement** | -6.37 | **2.81** | **2.17** | **3.01** | -7.28 | -1.58 |

Table 2: Mean sMAPE for all models with the transformed and untransformed method

From the table, we can discover that the implementation of transformation is improved for the KNN, RF, and ETS models with 2.63%, 0.3% and 0.27%, respectively. We also calculate the standard deviation of sMAPE of each model described in Table 2 which can be an indicator of the robustness of each model. The evidence shows that the transformations reduce the standard deviation of sMAPE at 2.81%, 2.17%, and 3.01% within the SVR, KNN, and RF models.

### 6.1.1 Seasonal Categorisation

Afterwards, we consider discussing the improvements of transformation in different dimensions. The seasonality and stationary test are separately performed to categorise the M3 datasets into different conditions, which have the ability to give various insights into the transformation. The following Table 3 exhibits the improved average of sMAPE and discusses the improvement in different conditions. The *Season* and *No Season* represent whether the time series consists of seasonal components and these two terms have the entire 1043 time series. Similarly, the *Stationary* and *Non Stationary* contain the mean sMAPE of 1043 datasets and have the ability to distinguish the improvement of averaged sMAPE in the stationary and non-stationary time series categories. The table shows that the transformation works for most models when the time series do not consist of seasonal components, and the SVR obtains the most considerable improvement at 5.83% for the non-seasonal time series. Interestingly, the KNN model provides improved

performance on all kinds of time series, and the ETS and RF are able to perform improvements on most kinds of time series. Concerning the stationary time series, the ETS model achieves a 7.05% improvement and outperforms the other models significantly.

| | **EN** | **SVR** | **KNN** | **RF** | **MLP** | **ETS** |
|---|---|---|---|---|---|---|
| **Season** | -8.5579 | -2.1482 | **2.7241** | -0.6069 | -7.2300 | **0.0711** |
| **No Season** | -3.9287 | **5.8391** | **2.3366** | **3.4552** | -10.0513 | **1.0919** |
| **Stationary** | -5.3845 | -8.0534 | **0.6670** | **1.6360** | -14.2798 | **7.0502** |
| **Non Stationary** | -7.8019 | **0.3611** | **2.8048** | **0.1774** | -7.2633 | -0.4381 |

Table 3: Improvement of sMAPE for all models in different conditions

Figure 4 depicts a stationary time series with an example of the ETS model's predictions of untransformed and transformed time series represented by blue and red dotted lines, respectively. The prediction of transformed time series can provide smoother results and better fit with the actual results.

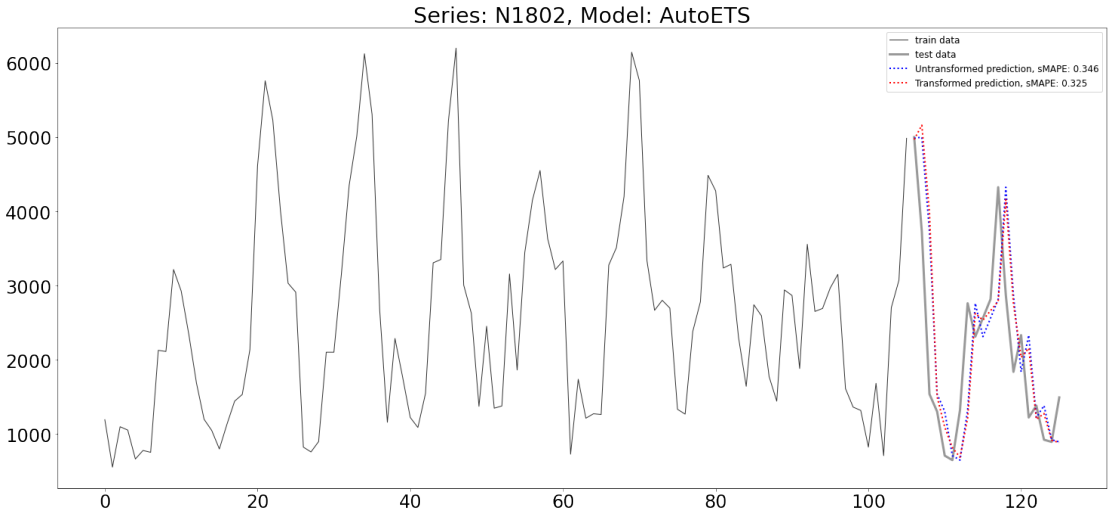


Figure 4: The ETS's predictions in a stationary time series

### 6.1.2 Domain Categorisation

On the other hand, we classify the time series by the categorical domains: demographic, finance, industry, macro, micro, and other. We expect to explore the enhancement of DWT, which has the capability to reduce the errors of averaged sMAPE and achieve an improvement. With the ORIGIN method, the transformation is improved in the micro domain, and most models can improve generated error. Those models are the SVR, KNN, RF, and MLP with an improved mean sMAPE at 5.0148%, 3.4613%, 1.1753%, and 6.4109%. In terms of the model's performance, the KNN and RF models are able to generate better forecasting with the reduced average of sMAPE when implementing DWT transformation. Remarkably, the performance of the SVR in the demographic domain gives an 8.1435% improvement in the entire table, and one specific demographic time series predicted by the SVR is plotted in Figure 5. The figure indicates that the prediction of untransformed time series mimics a naive forecaster and is outperformed by the transformed time series.

| | **EN** | **SVR** | **KNN** | **RF** | **MLP** | **ETS** |
|---|---|---|---|---|---|---|
| **DEMOGRAPHIC** | -1.4840 | **8.1435** | -0.4904 | **5.7318** | -44.4621 | -16.0158 |
| **FINANCE** | -6.5673 | -4.2267 | -2.3066 | -7.1702 | -22.6042 | **0.4006** |
| **INDUSTRY** | -8.2811 | -5.5576 | **3.4880** | **0.6851** | -14.6832 | **3.5600** |
| **MACRO** | -12.2030 | -3.7644 | **2.9426** | **1.1979** | -6.5518 | -3.1676 |
| **MICRO** | -6.1751 | **5.0148** | **3.4613** | **1.1753** | **6.4109** | -0.6667 |
| **OTHER** | -16.5339 | -7.0225 | **1.2397** | -0.9597 | -2.2556 | -20.3219 |

Table 4: Improvement of sMAPE for all models in different domains

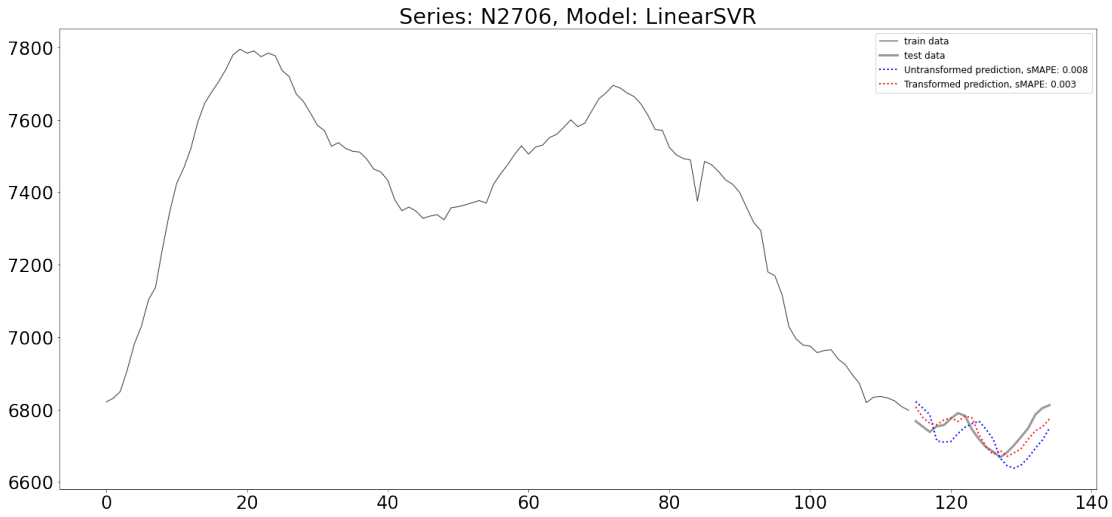


Figure 5: The SVR's predictions in the demographic domain

### 6.1.3 Wavelets Selection

Table 5 reveals the numbers of different wavelets performed in the improved time series and discusses the total amount of every model. In each time series, the presence of 6 models requires 6 wavelets to implement the DWT transformation, and we calculate the types and the times of wavelets from the improved models. Therefore, the rows represent the summations of wavelets from improved models, and the columns indicate the total improved transformation for every model. With respect to the wavelet selection, the models have heterogeneous preferences to transform time series by different wavelets. The EN is prone to use *db5* and obtains the improvements 108 times; the SVR performs better and has improved with the Coiflet family with a total of 186 times. The KNN, RF, and ETS models are capable of enhancing the forecasting results by the *Haar* wavelet. Eventually, the MLP is discovered with no apparent preferences and performs the improved transformation with various wavelets on an average basis.

| | EN | SVR | KNN | RF | MLP | ETS | Total |
|---|---|---|---|---|---|---|---|
| **coif4** | 50 | 97 | 68 | 44 | 65 | 57 | 381 |
| **coif5** | 30 | 89 | 81 | 91 | 66 | 81 | 438 |
| **db19** | 81 | 81 | 92 | 86 | 84 | 90 | 514 |
| **db20** | 61 | 51 | 73 | 77 | 66 | 100 | 428 |
| **db5** | 108 | 84 | 121 | 97 | 82 | 93 | 585 |
| **db6** | 74 | 84 | 104 | 84 | 62 | 60 | 468 |
| **haar** | 87 | 69 | 199 | 140 | 62 | 133 | 690 |
| **Total** | 491 | 555 | 738 | 619 | 487 | 614 | 3504 |

Table 5: Numbers of wavelet with improved models

### 6.1.4 Fraction-Best

Finally, we compute the fraction-best of every model to investigate whether the transformation has the ability to give the models improved performance. Table 6 shows the ratio of best-performed forecasting between untransformed and transformed time series, and the *Mean* column represents the 55.9923% of the time the models with transformed time series can perform better than the untransformed time series. Surprisingly, the KNN has a 70.7574% probability that the model improves with DWT transformation. The SVR, RF, and ETS can improve the performance over half the time with 53.2119%, 59.348%, and 58.8686%, respectively.

| | EN | SVR | KNN | RF | MLP | ETS | Mean |
|---|---|---|---|---|---|---|---|
| **UNTRANS** | 52.9243 | 46.7881 | 29.2426 | 40.652 | 53.3078 | 41.1314 | 44.0077 |
| **TRANS** | 47.0757 | 53.2119 | 70.7574 | 59.348 | 46.6922 | 58.8686 | 55.9923 |

Table 6: Fraction-best for each model

## 6.2 Analysis with Preprocessed Time Series

Compared to the ORIGIN method, we evaluate the experiment by preprocessing method, namely *PREPROCESSED* method. If needed, the detrend and deseasonalisation techniques are applied on both untransformed and transformed time series.

To start with, we calculate the mean sMAPE from 1043 untransformed and transformed time series for each model with the PREPROCESSED method and generate each improvement of the mean sMAPE. With the statistical investigation from Table 7, it is obvious that most of the models: EN, SVR, KNN, and RF, have reduced the sMAPE by DWT transformation with improvements of 0.47%, 4.39%, 1.16%, 2.09%, and 2.13%, respectively.

| | | **EN** | **SVR** | **KNN** | **RF** | **MLP** | **ETS** |
|---|---|---|---|---|---|---|---|
| **Mean** | **UNTRANS** | 16.77 | 19.55 | 17.15 | 17.17 | 18.30 | 16.16 |
| | **TRANS** | 16.69 | 18.69 | 16.95 | 16.81 | 17.91 | 16.25 |
| | **Improvement** | **0.47** | **4.39** | **1.16** | **2.09** | **2.13** | -0.55 |
| **Standard Deviation** | **UNTRANS** | 27.76 | 29.92 | 27.31 | 27.63 | 28.16 | 27.40 |
| | **TRANS** | 27.68 | 28.83 | 27.29 | 27.18 | 27.94 | 27.36 |
| | **Improvement** | **0.29** | **3.62** | **0.04** | **1.61** | **0.77** | **0.12** |

Table 7: Mean sMAPE for all models with the tranformed and untransformed method

Table 7 exhibits the standard deviation of sMAPE for both untransformed and transformed experiments with each model. Although some models only provide slight improvement, such as the KNN and the ETS, with improvements of 0.04% and 0.12%. The DWT transformation is capable of giving a comprehensive improvement of sMAPE's standard deviation for every model and provides the largest improved standard deviation for the SVR at 3.62%.

### 6.2.1 Seasonal Categorisation

| | EN | SVR | KNN | RF | MLP | ETS |
|---|---|---|---|---|---|---|
| **Season** | **1.0660** | **4.1738** | **1.0861** | **2.0438** | **2.2609** | -0.1920 |
| **No Season** | -4.8519 | **6.1254** | **2.0326** | **2.7974** | **1.2195** | -3.2804 |
| **Stationary** | -1.4539 | **0.2138** | **1.3350** | **1.3099** | **0.0483** | -1.6917 |
| **Non Stationary** | **0.5609** | **4.7243** | **1.1919** | **2.1939** | **2.2748** | -0.4450 |

Table 8: Improvement of sMAPE for all models in different conditions

Subsequently, we discuss the improvement of the average of sMAPE in different conditions with the PREPROCESSED method, whether the time series contains seasonal components and is stationary. Regarding the seasonality, the mean sMAPE of seasonal time series is improved in all regression models except for the ETS. Even though the time series without seasonal components are able to provide enhanced performance in most models, the SVR is highlighted because of the vast reduction of transformation's error at 4.1738% and 6.1254%. An example of the SVR's prediction for the time series without a seasonal component is depicted in Figure 6. The untransformed prediction is volatile and can not give feasible forecasting; the transformation provides the SVR with a robust model that can estimate helpful results compared to the actual values.

With respect to comparing the stationary and non-stationary time series, the transformation of the non-stationary time series is able to achieve comprehensive improvement and performs better than the transformation of the stationary time series. The PREPROCESSED method also makes the SVR, KNN, RF, and MLP improve on all kinds of conditions; especially, the SVR and MLP provide improvements of non-stationary time series multiple times as much as those of stationary time series. We can discover that the DWT transformation has the ability to give models better improvements on non-stationary time series by the PREPROCESSED method.

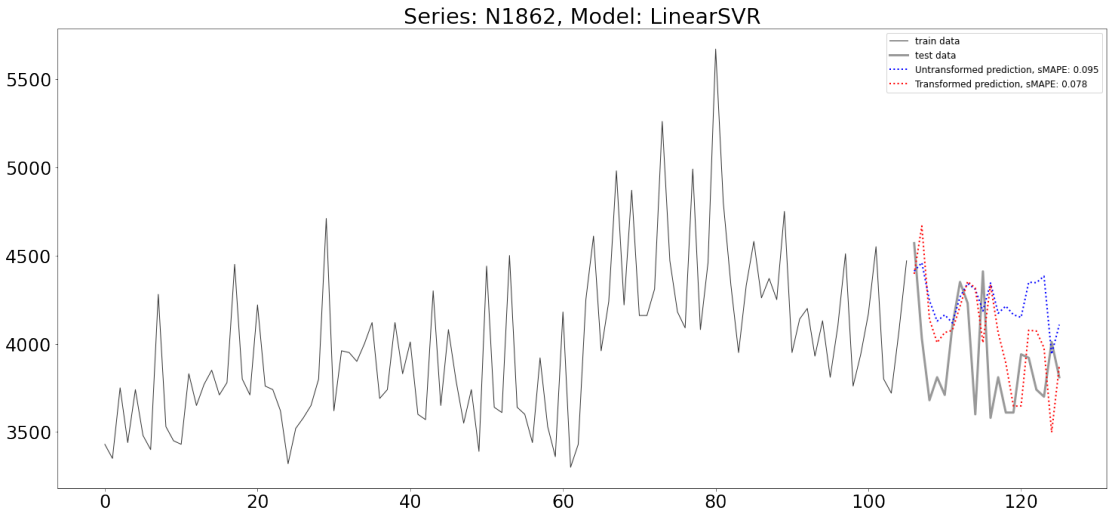


Figure 6: The SVR's predictions of a non-seasonal time series

### 6.2.2 Domain Categorisation

| | EN | SVR | KNN | RF | MLP | ETS |
|---|---|---|---|---|---|---|
| **DEMOGRAPHIC** | -1.1919 | **1.2652** | -0.6287 | **1.3241** | **4.2874** | -1.2224 |
| **FINANCE** | -2.2652 | **1.6919** | -0.7374 | -0.8278 | **0.0258** | -0.4172 |
| **INDUSTRY** | -1.7878 | **1.0970** | **1.5907** | **1.8091** | -1.7508 | -1.7371 |
| **MACRO** | -2.7215 | **0.7880** | **2.3958** | -0.2604 | -1.9672 | -0.8240 |
| **MICRO** | **6.0323** | **11.1661** | **1.1533** | **5.0931** | **9.4775** | **1.4215** |
| **OTHER** | -13.1761 | -19.3789 | -1.4726 | -3.9846 | -7.1329 | -6.2927 |

Table 9: Improvement of sMAPE for all models in different domains

In terms of distinguishing time series by the categorical domains, we are going to investigate which kinds of time series is benefited from transformation with the PREPROCESSED method. The Table 9 shows the massive improvements on every model in micro domain at 6.0323%, 11.1661%, 1.1533%, 5.0931%, 9.4775%, and 1.4215%, respectively. In order to dive into the improvements of the SVR and MLP in the micro domain, the Figure 7 illustrates the box plots to describe the

robustness of the model between untransformed and transformed time series. It can be observed that the SVR and MLP perform the lower mean sMAPE, and each of them obtains a more robust model.

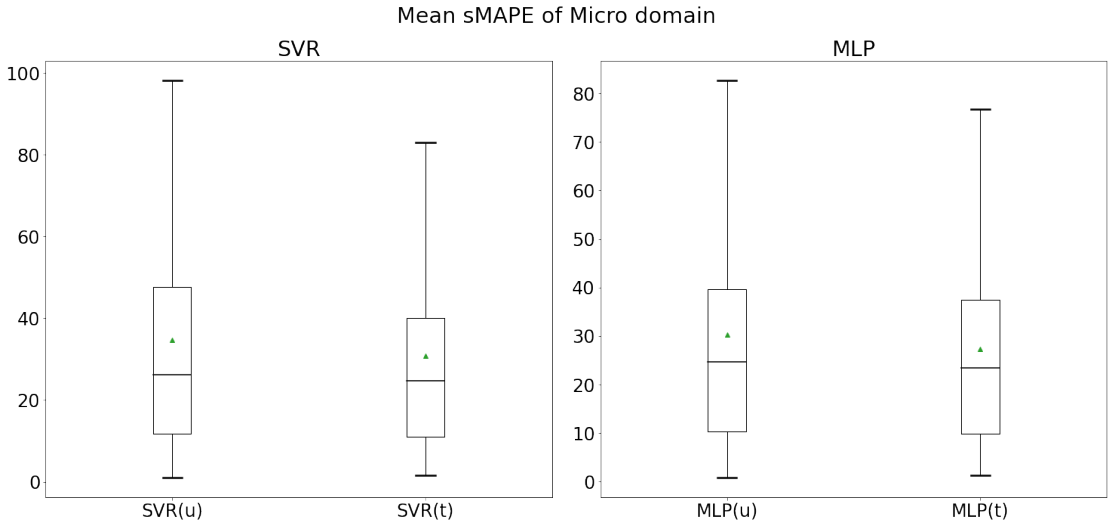


Figure 7: Mean sMAPE of Micro domain

On the other hand, the SVR has the capability to achieve improvements on most of the domains and the MLP performs effective enhancements in the demographic and micro domains with 4.2874% and 9.4775%. In addition, the "other" domain consists of barely two samples, which should be excluded from the analysis due to the lack of statistical evidence. The following Figure 8 exhibits the comparison of the MLP's predictions between the implementation of transformation and without transformation. It reveals that the model trained by the untransformed time series can not capture the peaks of the actual time series; however, the transformed time series gives the MLP capability of estimating a proper prediction much more similar to the original time series.

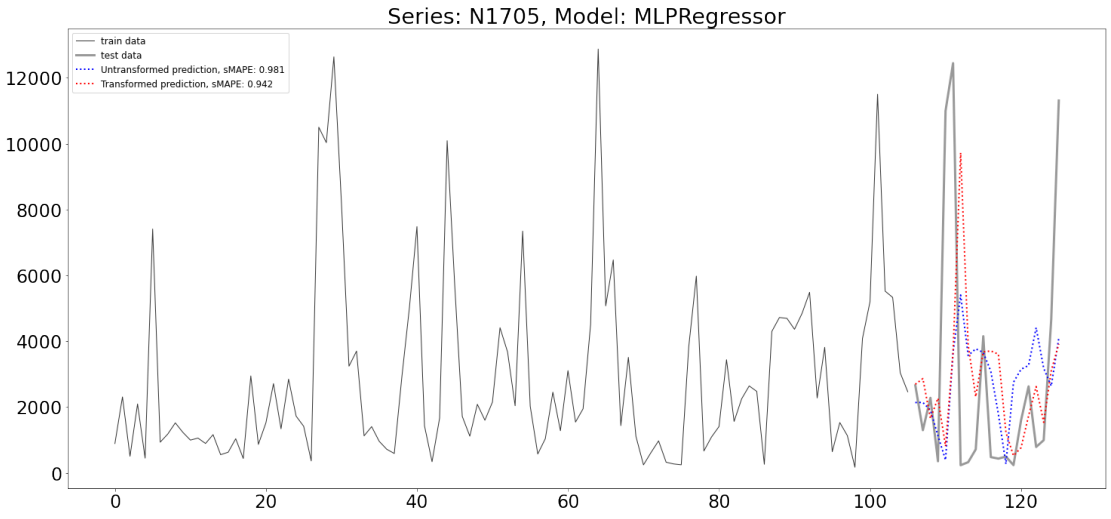


Figure 8: The MLP's predictions in the micro domain

### 6.2.3 Wavelets Selection

In this paragraph, we will investigate the improvements regarding the models' preferences using the PREPROCESSED method in Table 10. In order to discuss the helpful wavelets of each model, we calculate the cumulative quantities of each wavelet of every improved model. We can observe that the haar wavelet is the most considered wavelet within the improved models and the coif4 and coif5 are the least wavelets, with 402 and 397 times used for the enhanced models. The EN and SVR are discovered to have more opportunities to improve the forecasting by transformation with Daubechies wavelets; the KNN and RF prefer haar with 209 and 143 times than the other kinds of wavelets. The MLP and ETS are not obvious for a specific wavelet and achieve the improvements with each wavelet family on average.

| | EN | SVR | KNN | RF | MLP | ETS | Total |
|---|---|---|---|---|---|---|---|
| **coif4** | 50 | 69 | 56 | 77 | 97 | 53 | 402 |
| **coif5** | 46 | 76 | 62 | 68 | 73 | 72 | 397 |
| **db19** | 124 | 102 | 116 | 92 | 86 | 98 | 618 |
| **db20** | 64 | 77 | 88 | 82 | 73 | 84 | 468 |
| **db5** | 127 | 81 | 109 | 91 | 65 | 66 | 539 |
| **db6** | 67 | 91 | 77 | 75 | 86 | 43 | 439 |
| **haar** | 104 | 69 | 209 | 143 | 88 | 104 | 717 |
| **Total** | 582 | 565 | 717 | 628 | 568 | 520 | 3580 |

Table 10: Numbers of wavelet with improved model

### 6.2.4 Fraction-Best

At last, we compare the frequency that the model with untransformed and transformed time series is capable of outperforming each other and illustrate the proportion of improvement for each model in Table 11. With the transformation, the models fitted by the transformed time series have the ability to beat the model without transformation and obtain improvements in the 57.2067% of the time. The PREPROCESSED method is helpful for most of the models to generate improved performance by applying DWT transformation. Even the ETS with the least Fraction-best at 49.8562% selects the model with transformation around half the time.

| | EN | SVR | KNN | RF | MLP | ETS | Mean |
|---|---|---|---|---|---|---|---|
| **UNTRANS** | 44.1994 | 45.8293 | 31.256 | 39.7891 | 45.5417 | 50.1438 | 42.7932 |
| **TRANS** | 55.8006 | 54.1707 | 68.744 | 60.2109 | 54.4583 | 49.8562 | 57.2067 |

Table 11: Fraction-best for each model

## 6.3 Discussion

To conclude, we propose the evaluation with the ORIGIN and PREPROCESSED methods to investigate the capability of DWT transformation. Within the ORIGIN method, there is no deterministic evidence showing that the transformation always delivers improvement. However, we can observe that the KNN and RF have the overall improvement for mean sMAPE and the standard deviation of sMAPE and have the ability to achieve an improvement in most of the conditions and domains. The SVR and MLP perform well in specific conditions and domains by using transformation. The performance measure, fraction-best, gives that the KNN performs better with the transformation 70% of the time, which is inspiring, and the mean fraction-best indicates that the transformation is useful 55% of the time.

With respect to the PREPROCESSED method, the DWT transformation gives a comprehensive improvement of the standard deviation of sMAPE and has the ability to improve the mean sMAPE for most models. Surprisingly, most models receive improved performance by transformation within different conditions except for the ETS. On the other hand, the transformation helps every model improve in the micro domain and is beneficial for the SVR to achieve improvement in most domains, excluding the "other" domain. The fraction-best shows that over half of the time, the transformation boosts the performance of the model with transformed time series and the KNN again achieves an improvement by the transformation in almost 70% of the time.

Eventually, the investigation of the wavelet selection indicates that there is not a dominant wavelet that has the ability to provide the best performance for target transformation in time series forecasting and each model has its preference of wavelet when applying DWT transformation. Therefore, the opportunity for wavelet selection is necessary for a model to perform a better transformation.

# 7 Conclusions

This dissertation aims to investigate forecasting improvement by using target transformation for time series. The main procedure of the experiment is established by the DWT transformation and model selection. We explain the process of implementing DWT with a wavelet and the setting of hyperparameters which consists of wavelets, lags, and models' parameters. Concerning model selection, we introduce validation with the expanding window method to explore the best combination of hyperparameters for each candidate model. Subsequently, the trained models are developed to generate forecasting results and evaluate the predictions on testing data. The experiment consists of two sub-experiments which capture the cases where some preprocessing is (not) applied before the transformation, respectively. We then implement model training and analysing the improvement of target transformation.

The evaluation is developed to analyse the improvement of the DWT in time series forecasting. The analysis can be divided into three components: overall evaluation of the statistical properties of the forecasting results, analysis of the individual characteristic of time series, and the usage of wavelets for improved models. Generally speaking, the DWT transformation can not provide an improvement on all time series and may lead to opposite outcomes within different conditions. For instance, the EN is improved, but the ETS becomes worse when implementing the PREPROCESSED method. For the aforementioned special case, the transformation for most models performs a more competitive performance than forecasting without transformation. In addition, the DWT transformation is observed to have the ability to generate improvement for most of the models within specific domains such as micro. On the other hand, the transformation is potentially prone to specific models and cooperates well to improve forecasting.

In the last paragraph, we are going to point out several constraints of the experiment and the potential future improvements. To perform a complete experiment, we have to expand each hyperparameter's search space and consider various time series datasets with various characteristics. In our experiment, the limited time and computational resources force us to shrink the search space of wavelets, lags,

and models' parameters. Thus, the larger search space may provide an optimal transformation and better results for forecasting problems. In order to establish an effective investigation and suitable applications, we purely consider the primary machine learning models and the well-developed statistical model. The hybrid and more complicated models can be considered for tackling the complex time series forecasting problems. In addition, an increasing number of the variants of WT are implemented and have the ability to achieve the enhancement and robustness of transformation and so, a future direction would be to investigate more variants.

# References

[1] G. P. Nason, "Stationary and non-stationary time series," *Statistics in volcanology*, vol. 60, 2006.

[2] J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International journal of forecasting*, vol. 22, no. 3, pp. 443–473, 2006.

[3] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "Stl: A seasonal-trend decomposition," *J. Off. Stat*, vol. 6, no. 1, pp. 3–73, 1990.

[4] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[5] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International journal of forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[6] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[7] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[9] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 1999.

[10] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[11] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[12] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[13] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[14] R. Salles, K. Belloze, F. Porto, P. H. Gonzalez, and E. Ogasawara, "Non-stationary time series transformation methods: An experimental review," *Knowledge-Based Systems*, vol. 164, pp. 274–291, 2019.

[15] Z. Qian, Y. Pei, H. Zareipour, and N. Chen, "A review and discussion of decomposition-based hybrid models for wind energy forecasting applications," *Applied energy*, vol. 235, pp. 939–953, 2019.

[16] B. Doucoure, K. Agbossou, and A. Cardenas, "Time series prediction using artificial wavelet neural network and multi-resolution analysis: Application to wind speed data," *Renewable Energy*, vol. 92, pp. 202–211, 2016.

[17] H. E. Hurst, "Long-term storage capacity of reservoirs," *Transactions of the American society of civil engineers*, vol. 116, no. 1, pp. 770–799, 1951.

[18] D. Liu, D. Niu, H. Wang, and L. Fan, "Short-term wind speed forecasting using wavelet transform and support vector machines optimized by genetic algorithm," *Renewable energy*, vol. 62, pp. 592–597, 2014.

[19] P. K. d. M. M. Freire, C. A. G. Santos, and G. B. L. da Silva, "Analysis of the use of discrete wavelet transforms coupled with ann for short-term streamflow forecasting," *Applied Soft Computing*, vol. 80, pp. 494–505, 2019.

[20] Z. Liu, P. Zhou, G. Chen, and L. Guo, "Evaluating a coupled discrete wavelet transform and support vector regression for daily and monthly streamflow forecasting," *Journal of hydrology*, vol. 519, pp. 2822–2831, 2014.

[21] D. Wu, X. Wang, and S. Wu, "A hybrid method based on extreme learning machine and wavelet transform denoising for stock prediction," *Entropy*, vol. 23, no. 4, p. 440, 2021.

[22] H. Yan and H. Ouyang, "Financial time series prediction based on deep learning," *Wireless Personal Communications*, vol. 102, no. 2, pp. 683–700, 2018.

[23] S. Lahmiri, "Wavelet low-and high-frequency components as features for predicting stock prices with backpropagation neural networks," *Journal of King Saud University-Computer and Information Sciences*, vol. 26, no. 2, pp. 218–227, 2014.

[24] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PloS one*, vol. 12, no. 7, p. e0180944, 2017.

[25] T.-J. Hsieh, H.-F. Hsiao, and W.-C. Yeh, "Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm," *Applied soft computing*, vol. 11, no. 2, pp. 2510–2525, 2011.

[26] A. J. Conejo, M. A. Plazas, R. Espinola, and A. B. Molina, "Day-ahead electricity price forecasting using the wavelet transform and arima models," *IEEE transactions on power systems*, vol. 20, no. 2, pp. 1035–1042, 2005.

[27] Z. Tan, J. Zhang, J. Wang, and J. Xu, "Day-ahead electricity price forecasting using wavelet transform combined with arima and garch models," *Applied energy*, vol. 87, no. 11, pp. 3606–3610, 2010.

[28] Z. Yang, L. Ce, and L. Lian, "Electricity price forecasting by a hybrid model, combining wavelet transform, arma and kernel-based extreme learning machine methods," *Applied Energy*, vol. 190, pp. 291–305, 2017.

[29] W. Li, D. Kong, and J. Wu, "A novel hybrid model based on extreme learning machine, k-nearest neighbor regression and wavelet denoising applied to short-term electric load forecasting," *Energies*, vol. 10, no. 5, p. 694, 2017.

[30] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny, "An empirical comparison of machine learning models for time series forecasting," *Econometric reviews*, vol. 29, no. 5-6, pp. 594–621, 2010.

[31] S. Makridakis and M. Hibon, "The m3-competition: results, conclusions and implications," *International journal of forecasting*, vol. 16, no. 4, pp. 451–476, 2000.

[32] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[33] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE transactions on information theory*, vol. 36, no. 5, pp. 961–1005, 1990.

[34] I. Daubechies, *Ten lectures on wavelets*. SIAM, 1992.

[35] O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE signal processing magazine*, vol. 8, no. 4, pp. 14–38, 1991.

[36] C. Cai and P. d. B. Harrington, "Different discrete wavelet transforms applied to denoising analytical data," *Journal of chemical information and computer sciences*, vol. 38, no. 6, pp. 1161–1170, 1998.

[37] G. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O'Leary, "Pywavelets: A python package for wavelet analysis," *Journal of Open Source Software*, vol. 4, no. 36, p. 1237, 2019.

[38] D. L. Donoho, "De-noising by soft-thresholding," *IEEE transactions on information theory*, vol. 41, no. 3, pp. 613–627, 1995.

[39] R. Aggarwal, J. K. Singh, V. K. Gupta, S. Rathore, M. Tiwari, and A. Khare, "Noise reduction of speech signal using wavelet transform with modified universal threshold," *International Journal of Computer Applications*, vol. 20, no. 5, pp. 14–19, 2011.

[40] I. Daubechies, *Different perspectives on wavelets*, vol. 47. American Mathematical Soc., 2016.

[41] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 4, pp. 1–21, 2012.

[42] Mcompetitions, "Mcompetitions/m4-methods: Data, benchmarks, and methods submitted to the m4 forecasting competition."

[43] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American statistical association*, vol. 74, no. 366a, pp. 427–431, 1979.

[44] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?," *Journal of econometrics*, vol. 54, no. 1-3, pp. 159–178, 1992.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[46] M. Löning, A. Bagnall, S. Ganesh, V. Kazakov, J. Lines, and F. J. Király, "sktime: A unified interface for machine learning with time series," *arXiv preprint arXiv:1909.07872*, 2019.

[47] P. Probst, M. N. Wright, and A.-L. Boulesteix, "Hyperparameters and tuning strategies for random forest," *Wiley Interdisciplinary Reviews: data mining and knowledge discovery*, vol. 9, no. 3, p. e1301, 2019.

# Declaration

I declare that this thesis is the solely effort of the author. I did not use any other sources and references than the listed ones. I have marked all contained direct or indirect statements from other sources as such.

Neither this work nor significant parts of it were part of another review process. I did not publish this work partially or completely yet. The electronic copy is consistent with all submitted copies.

Signature and date:

# A Appendix A

## A.1 Source Code

### A.1.1 model_selection.py

```
import argparse
import multiprocessing
import os
import sys
import time
import warnings

import numpy as np
import pandas as pd
from sklearn.model_selection import ParameterGrid,
    train_test_split
from statsmodels.tsa.stattools import adfuller, kpss
from tabulate import tabulate

import settings
from transformation import Transformation
from utils import *


class BestModelSearch:

    def __init__(self, dataset_name=None, dataset=None, test_size
    =0.2, gap=0, log_return=False, detrend=False, worker_id=1):

        self.dataset_name = dataset_name
        self.dataset = dataset
        self.test_size = test_size
        self.gap = gap
        self.log_return = log_return
        self.detrend = detrend
        self.worker_id = worker_id

        self.transform = Transformation(dataset)
        self.de_ts = DeTrendSeason()
```

```
33
34          self.train_X = None
35          self.test_X = None
36          self.train_y = None
37          self.test_y = None
38
39          # models and hyperparameters
40          self.regression_models = settings.regression_models
41          self.forecasting_models = settings.forecasting_models
42          self.tuning_models = list(settings.regression_models.keys
    ()) + list(settings.forecasting_models.keys())
43          self.params = settings.params
44
45          # performance scores of each best model
46          self.metric_info = pd.DataFrame()
47
48          # save information to JSON
49          self.record = Records()
50
51      def save_scores_info(self, scores, model_name):
52          this = pd.Series(scores, index=scores.keys(), name=
    model_name)
53          self.metric_info = pd.concat([self.metric_info, this],
    axis=1)
54
55      #
    ------------------------------------------------------------------
56      # generate transformed Xt and yt from traget y
57      # regression_data: the regression model and the forecasting
    model have different input length of time series, especially
    lag is greater than 0
58      # isnaive: we dont do naive forecaster on log return but
    predict price directly
59      def gen_feature_data(self, series, lags=2, horizon=1,
    transform_threshold=0.005, regression_data=False, isnaive=False
    ):
60          data = []
61          data_transformed = []
62          feature = np.array([])
```

```
63
64          try:
65              # detrend and deseasonal
66              if self.detrend:
67                  raw_series = np.copy(series[:-1])
68                  raw_series = self.de_ts.remove_seanson(raw_series,
     freq=12)
69                  self.raw_series = self.de_ts.detrend(raw_series)
70
71              if transform_threshold:
72                  # if threshold > 0, do transformation
73
74                  if self.detrend:
75                      input_y = self.raw_series
76                  else:
77                      input_y = series[:-1]
78
79                  if isinstance(transform_threshold, str):
80                      feature, series_transformed = self.transform.
    dwt_feature(input_y, wavelet=transform_threshold)
81                  else:
82                      series_transformed = self.transform.dwt(
    input_y, threshold=transform_threshold)
83
84                  # concat test_y
85                  series_transformed = np.concatenate((
    series_transformed, series[-1:, ]))
86              else:
87                  if self.detrend:
88                      series_transformed = np.concatenate((self.
    raw_series, series[-1:, ]))
89                  else:
90                      series_transformed = series
91
92              # transform price into log return excluding naive
    forecaster
93              if self.log_return and not isnaive:
94                  series_transformed = np.diff(np.log(
    series_transformed.ravel()))
```

```
95
96              if regression_data:
97                  # data for regression model is shorter because of
    the lag
98                  for i in range(len(series_transformed) - lags -
    horizon + 1):
99                      end_idx = i + lags + horizon
100                     if feature.any():
101                         data.append(np.concatenate((feature[
    end_idx-2, :].ravel(), series[i: end_idx].ravel())))
102                         data_transformed.append(np.concatenate((
    feature[end_idx-2, :].ravel(), series_transformed[i: end_idx].
    ravel())))
103                     else:
104                         data.append(series[i: end_idx])
105                         data_transformed.append(series_transformed
    [i: end_idx])
106
107                 X = np.array(data)[:, :-1 * horizon]
108                 y = np.array(data)[:, -1 * horizon:]
109
110                 Xt = np.array(data_transformed)[:, :-1 * horizon]
111                 yt = np.array(data_transformed)[:, -1 * horizon:]
112             else:
113                 X, Xt = [], []
114                 y = series.reshape(-1, 1)
115                 yt = series_transformed.reshape(-1, 1)
116
117         except Exception as e:
118             exc_type, exc_obj, exc_tb = sys.exc_info()
119             logger.error(f'(line:{exc_tb.tb_lineno}) {e}')
120
121         return X, y, Xt, yt
122
123     # ------------------------------------------------------------
124     # use target y to generate training part and test part
125     # regression_data: the regression model and the forecasting
    model have different input length of time series, especially
    lag is greater than 0
```

```
126     # isnaive: check if doing naive forecaster
127     def gen_train_test(self, series, lag, threshold, test_len,
    regression_data=False, isnaive=False):
128         # generate expanding window validation set
129         step_size = 1
130 
131         for i in range(test_len):
132             end_idx = -1 * test_len + (i + 1) * step_size - self.
    gap
133             if end_idx == 0:
134                 y = series[:]
135             else:
136                 y = series[:end_idx]
137 
138             # generate (un)transformed features X1, X2, ... by
    target y
139             X, y, Xt, yt = self.gen_feature_data(y, lags=lag,
    transform_threshold=threshold, regression_data=regression_data,
     isnaive=isnaive)
140 
141             train_Xt = Xt[:-1 * step_size]
142             train_yt = yt[:-1 * step_size]
143 
144             test_Xt = Xt[-1 * step_size:]
145             test_y = series[end_idx + self.gap - 1]
146 
147             yield train_Xt, train_yt.ravel(), test_Xt, test_y.
    ravel(), X, y, i
148 
149     # ----------------------------------------
150     # fit model with best parameters and implement retrain window
151     # item: train, test data and index of test data
152     def fit_predict(self, model_name, model, param, item, test_len
    , iteration=7, horizon=1, retrain_window=10):
153         train_Xt, train_yt, test_Xt, test_y, X, y, idx = item
154 
155         if test_len <= 10:
156             retrain_window = 1
157         elif test_len <= 50:
```

```
158             retrain_window = int(test_len / iteration) + 1
159         else:
160             retrain_window = 10
161 
162         fit_model = idx % retrain_window == 0
163 
164         try:
165             # initial model in first row of test data
166             if fit_model and idx == 0:
167                 # fitting model
168                 if model_name in settings.regression_models:
169                     if model_name in ['GRU', 'LSTM']:
170                         param.update({'feature_num': train_Xt.
    shape[1], 'batch_size': int(train_Xt.shape[0]/100)+1})
171                     model = self.regression_models[model_name]()
172                     model.set_params(**param)
173                     model.fit(train_Xt, train_yt)
174                 elif model_name in settings.forecasting_models:
175                     model = self.forecasting_models[model_name](**
    param).fit(train_yt)
176             # fitting and last time fitting
177             elif fit_model:
178                 if model_name in settings.regression_models:
179                     if model_name in ['GRU', 'LSTM']:
180                         model.fit(train_Xt, train_yt)
181                     else:
182                         model.fit(train_Xt, train_yt)
183                 elif model_name in settings.forecasting_models:
184                     model.fit(train_yt)
185 
186             else:
187                 # In the iteration without fitting, we still have
    to update the forecasting model's observation
188                 if model_name in ['GRU', 'LSTM']:
189                     model.fit(train_Xt[-1:], train_yt[-1:])
190                 if model_name in settings.forecasting_models:
191                     model.update(pd.DataFrame(train_yt[-1].reshape
    (-1, 1), index=[len(train_yt) - 1]), update_params=True)
192 
```

```
193             # predicting
194             if model_name in settings.regression_models:
195                 prediction = model.predict(test_Xt)
196             elif model_name in settings.forecasting_models:
197                 prediction = model.predict(fh=[horizon])
198 
199         except Exception as e:
200             exc_type, exc_obj, exc_tb = sys.exc_info()
201             logger.error(f'({model_name}:{exc_tb.tb_lineno}) {e}')
202 
203         if isinstance(prediction, (pd.Series, pd.DataFrame)):
204             prediction = prediction.values.ravel()
205         if isinstance(prediction, (np.floating, float)):
206             prediction = [prediction].ravel()
207 
208         log_pred = prediction
209 
210         if self.log_return and model_name != 'NaiveForecaster':
211         # transform log return back to actual value
212             if model_name in ['AutoETS']:
213                 prediction = np.exp(prediction) * y[-2: -1]
214             else:
215                 prediction = np.exp(prediction) * y[-1:]
216 
217         if self.detrend:
218             prediction = self.de_ts.add_trend(self.raw_series,
    prediction.ravel())
219             prediction = self.de_ts.add_season(self.raw_series,
    prediction, freq=12)
220 
221         return model, prediction, log_pred
222 
223     # -------------------------------------------------
224     # series: entire time series
225     # best_data: the parameters of best_validated model
226     def retrain(self, series, best_data):
227 
228         for model_name, model_value in best_data.items():
229 
```

```
230             model_name, model, param, lag, threshold, horizon =
    list(model_value.values())
231
232             predictions, trues, log_predictions = [], [], []
233             start_time = time.time()
234             regression_data = model_name in self.regression_models
235             for item in self.gen_train_test(series, lag, threshold
    , self.test_size, regression_data=regression_data, isnaive=
    model_name=='NaiveForecaster'):
236
237                 try:
238                     model, prediction, log_pred = self.fit_predict
    (model_name, model, param, item, self.test_size)
239                     predictions.append(prediction.ravel()[0])
240                     log_predictions.append(log_pred)
241                     trues.append(item[-4])
242                 except Exception as e:
243                     exc_type, exc_obj, exc_tb = sys.exc_info()
244                     logger.error(f'({model_name}[{threshold}]:{
    exc_tb.tb_lineno}) {e}')
245
246             predictions = np.array(predictions, dtype=float)
247             log_predictions = np.array(log_predictions, dtype=
    float)
248             trues = np.array(trues, dtype=float)
249
250             p_metric = Performance_metrics(true_y=trues, predict_y
    =predictions)
251             p_metric.measuring(model_name=model_name, dataset_name
    =self.dataset_name)
252
253             # additional information
254             end_time = time.time()
255             elasped_time = end_time - start_time
256
257             additional_info = {}
258             additional_info['retrain_time'] = round(elasped_time,
    4)
259             additional_info['dwt_level'] = self.transform.level
```

```
260             if model_name in ['RandomForestRegressor']:
261                 additional_info['feature_importance'] = model.
    feature_importances_.tolist()
262             if model_name in ['GRU', 'LSTM']:
263                 additional_info['fitted_params'] = dict()
264             else:
265                 additional_info['fitted_params'] = model.
    get_params()
266 
267             # save best model testing result
268             transformed = 'transformed' if isinstance(threshold,
    str) else 'untransformed'
269             self.record.insert_model_info(transformed, model_name,
     **p_metric.scores, prediction=predictions, log_prediction=
    log_predictions, lags=lag, horizon=horizon, threshold=threshold
    , best_params=param, additional_info=additional_info)
270             self.save_scores_info(scores=p_metric.scores,
    model_name=model_name)
271 
272     # ----------------------------------------------------
273     # model_name: current model's name
274     # series: train_y
275     # threshold_lag: combination of thresholds and lags
276     # scoring: error measure
277     def tuning(self, model_name, series, threshold_lag, scoring='
    symmetric_mean_absolute_percentage_error'):
278         p_metric = Performance_metrics()
279 
280         # best info of the model
281         best_data = {
282             'best_model_name': model_name,
283             'best_model': None,
284             'best_param': None,
285             'best_lag': None,
286             'best_threshold': None,
287             'best_horizon': None,
288         }
289 
290         best_score = np.inf
```

```
291
292         # combinations of model's parameters
293         params = list(ParameterGrid(self.params[model_name]))
294         regression_data = model_name in self.regression_models
295         if not regression_data:
296             threshold_lag = list(set([(1, 1, tl[-1])for tl in
    threshold_lag]))
297
298         start = time.time()
299         for lag, horizon, threshold in threshold_lag:
300             for param in params:
301                 try:
302                     predictions, trues = [], []
303                     model = self.regression_models[model_name] if
    regression_data else self.forecasting_models[model_name]
304
305                     # generate validation data
306                     for item in self.gen_train_test(series, lag,
    threshold, self.test_size, regression_data=regression_data,
    isnaive=model_name=='NaiveForecaster'):
307
308                         model, pred, log_pred = self.fit_predict(
    model_name, model, param, item, self.test_size)
309
310                         predictions.append(pred.ravel()[0])
311                         trues.append(item[-4])
312
313                     scores = p_metric.one_measure(scoring=scoring,
     true_y=trues, pred_y=predictions)
314
315                 except Exception as e:
316                     exc_type, exc_obj, exc_tb = sys.exc_info()
317                     logger.error(f'({model_name}[{threshold}]:{
    exc_tb.tb_lineno}) {e}')
318                     scores = np.inf
319                     break
320
321                 # update best model's information
322                 if scores < best_score:
```

```
323                     best_score = scores
324                     best_data['best_model'] = model
325                     best_data['best_param'] = param
326                     best_data['best_lag'] = lag
327                     best_data['best_threshold'] = threshold
328                     best_data['best_horizon'] = horizon
329 
330         logger.debug(f'Worker {self.worker_id}| takes {time.time() - start:.2f}s on tuning ({self.dataset_name}:{model_name})')
331 
332         return best_data
333 
334     # ------------------------------------------------
335     # series: entire time series
336     # threshold_lag: combination of thresholds and lags
337     def hyperparameter_tuning(self, series, threshold_lag):
338 
339         # save training set and testing set information
340         self.train_y, self.test_y = train_test_split(series, test_size=self.test_size, shuffle=False)
341         logr_train_y, logr_test_y = train_test_split(np.diff(np.log(series.ravel())), test_size=self.test_size, shuffle=False)
342         self.record.insert(name=self.dataset_name, series=self.dataset, test_size=self.test_size, gap=self.gap)
343         self.record.insert(is_log_return=self.log_return, is_de_trend_season=self.detrend, threshold_lag=threshold_lag)
344         self.record.insert(train_y=self.train_y, test_y=self.test_y, logr_train_y=logr_train_y, logr_test_y=logr_test_y)
345 
346         best_data = dict()
347         for model_name in self.tuning_models:
348             # use training set to validate model
349             best_data[model_name] = self.tuning(model_name=model_name, series=self.train_y, threshold_lag=threshold_lag)
350         return best_data
351 
352 
353 class MultiWork:
354 
```

```
355     def __init__(self, dataset, lags=range(1, 4), thresholds=np.
    arange(0.04, 0.16, 0.04), gap=22, log_return=False, detrend=
    False, worker_num=1, warning_suppressing=False):
356         self.dataset = list(dataset.items())
357         self.worker_num = worker_num
358         self.lags = lags
359         self.thresholds = thresholds
360         self.gap = gap
361         self.log_return = log_return
362         self.detrend = detrend
363 
364         self.warning_suppressing(ignore=warning_suppressing)
365 
366     def warning_suppressing(self, ignore=True):
367         if not sys.warnoptions and ignore:
368             warnings.simplefilter("ignore")
369             os.environ["PYTHONWARNINGS"] = "ignore"
370             os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
371 
372     def gen_hyperparams(self, lags, horizons, thresholds):
373         hyper_threshold_lag = []
374         for l in lags:
375             for h in horizons:
376                 for t in thresholds:
377                     hyper_threshold_lag.append((l, h, t))
378         return hyper_threshold_lag
379 
380     def work(self, dataset_item):
381         start = time.time()
382 
383         # time series name and value
384         name, dataset = dataset_item
385 
386         # determine the size of test set
387         if len(dataset) > 200:
388             test_size = int(len(dataset) / 5)
389         else:
390             test_size = 20
391 
```

```
392         # check if using multiprocessing
393         try:
394             worker_id = multiprocessing.current_process().
    _identity[0]
395         except:
396             worker_id = 1
397         logger.info(f'Worker {worker_id}| is processing {name}(len
    : {len(dataset)}, test size: {test_size})')
398 
399         try:
400             adf = adfuller(dataset)
401             kpsst = kpss(dataset)
402             logger.info(f'Worker {worker_id}| {name}, ADF: {adf[0]
     < adf[4]["5%"]}, KPSS: {kpsst[0] < kpsst[3]["5%"]}')
403         except Exception as e:
404             print(e)
405 
406         # initial model selection class
407         bms = BestModelSearch(dataset_name=name, dataset=dataset,
    test_size=test_size, gap=self.gap, log_return=self.log_return,
    detrend=self.detrend, worker_id=worker_id)
408 
409         # hyperparameter tuning with/without transformation
410         for label, thresholds in zip(['Untransformed', '
    Transformed'], [[0.], self.thresholds]):
411             try:
412                 # generate combinations of lags and thresholds
413                 threshold_lag = self.gen_hyperparams(lags=self.
    lags, horizons=[1], thresholds=thresholds)
414 
415                 logger.info(f'Worker {worker_id}| is tuning {name}
     ({label})')
416                 # model validation and get best params of each
    model
417                 best_data = bms.hyperparameter_tuning(dataset,
    threshold_lag)
418                 logger.debug(f"Worker {worker_id}| Best model of {
    name} after tuning ({label}):\n{tabulate(pd.DataFrame(best_data
    ), headers='keys', tablefmt='psql')}")
```

```
419
420                 # refit and retrain model
421                 logger.info(f'Worker {worker_id}| is retraining {
    name} ({label})')
422                 bms.retrain(series=dataset, best_data=best_data)
423             except Exception as e:
424                 exc_type, exc_obj, exc_tb = sys.exc_info()
425                 logger.error(f'(line:{exc_tb.tb_lineno}) {e}')
426
427         end = time.time()
428         logger.info(f'Worker {worker_id}| is saving {name}\'s
    results, takes {(end-start):.2f} seconds.')
429         bms.record.save_json(name)
430
431     def set_workers(self, leave_one=True):
432         cpus = multiprocessing.cpu_count() - 1 * leave_one
433         self.worker_num = min(cpus, self.worker_num, len(self.
    dataset))
434         if self.worker_num == 0:
435             self.worker_num = 1
436         logger.info(f'CPUs count: {cpus+1*leave_one} ==> workers:
    {self.worker_num}')
437
438     @save_json
439     def run(self):
440         self.set_workers(leave_one=False)
441         if self.worker_num == 1:
442             for data in self.dataset:
443                 self.work(data)
444         else:
445             pool = multiprocessing.Pool(self.worker_num)
446             pool.imap_unordered(self.work, self.dataset)
447             pool.close()
448             pool.join()
449
450
451 if __name__ == '__main__':
452     parser = argparse.ArgumentParser()
453     parser.add_argument("--thresholds", help="thresholds excludes
```

```
    zero", nargs="*", type=float, default=[0.05])
454  parser.add_argument("--lags", help="lags", nargs="*", type=int
    , default=list(range(1, 6)))
455  parser.add_argument("--gap", help="number of gap", type=int,
    default=0)
456  parser.add_argument("--worker", help="number of worker", type=
    int, default=30)
457  parser.add_argument("--data_num", help="number of data", type=
    int, default=150)
458  parser.add_argument("--data_length", help="minimum length of
    data", type=int, default=100)
459  parser.add_argument("--test", help="test setting", action="
    store_true")
460  args = parser.parse_args()
461
462
463  if args.test:
464      args.lags = [1, 12]
465      args.thresholds = ['haar', 'db5', 'db6', 'db19', 'db20', '
    coif4', 'coif5']
466      args.worker = 8
467      ignore_warn = True
468      logger.info('[TEST MODE]')
469  else:
470      ignore_warn = True
471
472  log_return = False
473  detrend = False
474
475  logger.info(f'Models: {list(settings.regression_models.keys())
    +list(settings.forecasting_models.keys())}')
476  logger.info(f'Lags: {args.lags}, Thresholds: {args.thresholds
    }, Gap: {args.gap}, Log Return: {log_return}, Detrend &
    Seasonal: {detrend}')
477
478  # load time series
479  datasets = load_m3_data(min_length=args.data_length, n_set=
    args.data_num)
480
```

```
481     mw = MultiWork(dataset=datasets, lags=args.lags, thresholds=
    args.thresholds, gap=args.gap, log_return=log_return, detrend=
    detrend, worker_num=args.worker, warning_suppressing=
    ignore_warn)
482     mw.run()
```

Listing 1: model_selection.py

### A.1.2 transformation.py

```
1  import numpy as np
2  import pandas as pd
3  import pywt
4
5
6  class Transformation:
7
8      def __init__(self, series=[]):
9          self.series = series
10         self.series_len = len(series)
11         self.level = None
12
13     def dwt(self, series, threshold=0.5, mode='smooth', wavelet='
    sym8', inverse=False, test=False):
14         series = np.array(series).ravel()
15         self.level = pywt.dwt_max_level(self.series_len, wavelet)
16
17         coeffs = pywt.wavedec(data=series, wavelet=wavelet, mode=
    mode, level=self.level)
18         threshold = threshold * (np.median(np.abs(coeffs[-1]))
    /0.6745) * np.sqrt(2 * np.log(len(coeffs[-1])))
19         coeffs[1:] = [pywt.threshold(i, value=threshold, mode='
    hard') for i in coeffs[1:]]
20         self.series = pywt.waverec(coeffs=coeffs, wavelet=wavelet,
     mode=mode)
21
22         if len(series) % 2 != 0:
23             return self.series[:-1]
24         return self.series
25
```

```
26     def dwt_feature(self, series, threshold=1, mode='smooth',
       wavelet='sym8'):
27         new_coeffs = []
28         series = np.array(series).ravel()
29         self.level = pywt.dwt_max_level(self.series_len, wavelet)
30         self.level = 1
31 
32         # decompose time series
33         coeffs = pywt.wavedec(data=series, wavelet=wavelet, mode=
       mode, level=self.level)
34 
35         threshold = threshold * (np.median(np.abs(coeffs[-1]))
       /0.6745) * np.sqrt(2 * np.log(len(coeffs[-1])))
36 
37         # reconstruct cA
38         i = pywt.upcoef('a', coeffs[0], wavelet, level=len(coeffs)
       -1, take=len(series))
39         new_coeffs.append(i.tolist())
40 
41         # reconstruct cD
42         for id, i in enumerate(coeffs[1:]):
43             i = pywt.threshold(i, value=threshold, mode="hard")
44             i = pywt.upcoef('d', i, wavelet, level=len(coeffs[1:])
       -id, take=len(series))
45 
46             new_coeffs.append(i.tolist())
47 
48         return np.array(new_coeffs).T, np.sum(new_coeffs, axis=0)
```

Listing 2: transformation.py

### A.1.3 utils.py

From this Python script, the class *DeTrendSeason* is a benchmark method provided by the M-competition[2]. The method allows us to preprocess the data by detrending and deseasinalisation.

```
1 import json
```

[2]https://github.com/Mcompetitions/M4-methods/blob/master/ML_benchmarks.py

```
import logging
import os
import time

import numpy as np
import pandas as pd
import tensorflow as tf
from orbit.utils.dataset import load_m3monthly
from sklearn import metrics
from tqdm import tqdm

logger = logging.getLogger()
formatter = logging.Formatter('[%(asctime)s %(levelname)-7s] %(message)s', datefmt='%Y%m%d %H:%M:%S')
logger.setLevel(logging.DEBUG)

stream_handler = logging.StreamHandler()
stream_handler.setLevel(logging.INFO)
stream_handler.setFormatter(formatter)

file_handler = logging.FileHandler(filename='./records.log', mode='w')
file_handler.setFormatter(formatter)
file_handler.setLevel(logging.DEBUG)

if logger.hasHandlers():
    logger.handlers.clear()
logger.addHandler(file_handler)
logger.addHandler(stream_handler)
logging.getLogger('matplotlib.font_manager').disabled = True

# ------------------------------------------------
# min_length: the minimum length of time series
# n_set: the number of different time series
def load_m3_data(min_length=100, n_set=5):
    datasets = dict()
    data = load_m3monthly()
    unique_keys = data['key'].unique().tolist()
    n_set = np.minimum(len(unique_keys)-1, n_set)
```

```
39      unique_keys = np.random.choice(unique_keys, size=n_set,
    replace=False)
40
41      logger.info('Loading M3 dataset')
42      for key in tqdm(unique_keys):
43          this = data[data['key'] == key]
44          if this.shape[0] > min_length:
45              datasets[key] = this['value'].to_numpy()
46              if len(datasets.keys()) == n_set:
47                  break
48      logger.info(f'Get {len(datasets)} datasets with length > {
    min_length}')
49      return datasets
50
51  def load_m4_data(min_length=300, max_length=10000, n_set=5, freq='
    Hourly', name=[]):
52      logger.info('Loading M4 dataset')
53      datasets = dict()
54      selected_datasets = dict()
55
56      for train_test in ['Train', 'Test']:
57          file_name = f'./Dataset/{train_test}/{freq.lower().
    capitalize()}-{train_test.lower()}.csv'
58          with open(file_name, 'r') as file:
59              for line in file.readlines()[1:]:
60                  line = line.strip().replace('"', '').split(',')
61                  dataset_name = line.pop(0)
62                  line = [float(v) for v in line if v]
63
64                  if name and dataset_name not in name:
65                      continue
66
67                  if train_test == 'Train':
68                      datasets[dataset_name] = np.array(line)
69                  else:
70                      entire_series = np.concatenate((datasets[
    dataset_name], np.array(line)), axis=None)
71                      datasets[dataset_name] = entire_series
72
```

```
73                     if len(selected_datasets) < n_set:
74                         if entire_series.size > min_length and
    entire_series.size < max_length:
75                             selected_datasets[dataset_name] =
    entire_series
76                     else:
77                         logger.info(f'Get {len(selected_datasets)}
     datasets ({min_length} < length < {max_length})')
78                         return selected_datasets
79
80     logger.info(f'Get {len(selected_datasets)} datasets with
    length > {min_length}')
81     return selected_datasets
82
83 def load_btc_pkl(freq='d'):
84     logger.info(f'Loading BTC dataset, frequency: {freq}')
85     data = np.load(f'./Dataset/btc_1{freq}.pkl', allow_pickle=True
    )
86     logger.debug(f'Columns: {list(data.columns)}')
87     return data
88
89 def save_json(method):
90     def main_func(*args, **kw):
91         open('./results.json.tmp', 'w').close()
92
93         result = method(*args, **kw)
94
95         if not os.path.isdir('./json_data'):
96             os.makedirs('./json_data')
97
98         target_path = f'./json_data/results_{int(time.time())}.
    json'
99
100         if os.path.exists(target_path):
101             os.remove(target_path)
102         os.rename('./results.json.tmp', target_path)
103         logger.info(f'Done! {target_path} is saved')
104
105         return result
```

```
106     return main_func
107 
108 class Performance_metrics:
109 
110     def __init__(self, true_y=None, predict_y=None):
111         self.true_y = np.array(true_y).ravel()
112         self.predict_y = np.array(predict_y).ravel()
113 
114         self.estimators = {
115             'explained_variance_score': metrics.
    explained_variance_score,
116             'max_error': metrics.max_error,
117             'mean_absolute_error': metrics.mean_absolute_error,
118             'mean_squared_error': metrics.mean_squared_error,
119             'root_mean_squared_error': self.root_mean_square_error
    ,
120             'r2': metrics.r2_score,
121             'mean_absolute_percentage_error': metrics.
    mean_absolute_percentage_error,
122             'symmetric_mean_absolute_percentage_error': self.
    symmetric_mean_absolute_percentage_error,
123             'relative_absolute_error': self.
    relative_absolute_error,
124         }
125         self.scores = dict.fromkeys(self.estimators.keys(), 0)
126 
127     def one_measure(self, scoring, true_y, pred_y):
128         return self.estimators[scoring](np.array(true_y), np.array
    (pred_y))
129 
130     def measuring(self, model_name, dataset_name):
131         for estimator in self.estimators.keys():
132             try:
133                 score = self.estimators[estimator](self.true_y,
    self.predict_y)
134                 self.scores[estimator] = round(score, 5)
135             except Exception as e:
136                 logger.warn(f'({model_name}:{estimator}): {e}')
137                 self.scores[estimator] = '-'
```

```
138
139         self.print_score(model_name, dataset_name)
140
141     def print_score(self, model_name, dataset_name, estimator=None
    ):
142         logger.debug('--'*12+model_name+'('+dataset_name+')'+'--'
    *12)
143         if estimator:
144             logger.debug(f'{estimator}: {self.scores[estimator]}')
145         else:
146             for k, v in self.scores.items():
147                 try:
148                     logger.debug(f'{k}: {v:.5f}')
149                 except:
150                     logger.debug(f'{k}: {v}')
151
152     # customised scoring
153     def root_mean_square_error(self, true_y, predict_y):
154         return np.sqrt(metrics.mean_squared_error(true_y,
    predict_y))
155
156     def symmetric_mean_absolute_percentage_error(self, true_y,
    predict_y):
157         if tf.is_tensor(true_y):
158             return tf.reduce_mean(tf.abs(predict_y - true_y) / ((
    tf.abs(predict_y) + tf.abs(true_y)) / 2))
159         return np.mean(np.abs(predict_y - true_y) / ((np.abs(
    predict_y) + np.abs(true_y)) / 2))
160
161     def relative_absolute_error(self, true_y, predict_y):
162         return np.sum(np.abs(true_y - predict_y)) / np.sum(np.abs(
    true_y - np.mean(true_y)))
163
164
165 class Records:
166
167     def __init__(self):
168         self.record = dict()
169
```

```
170     def insert(self, **kwargs):
171         for k, v in kwargs.items():
172             self.record[k] = self.check_type(v)
173 
174     def insert_model_info(self, transformed, model_name, **kwargs)
    :
175         if transformed not in self.record:
176             self.record[transformed] = dict()
177 
178         info = {
179             model_name: dict()
180         }
181         for k, v in kwargs.items():
182             info[model_name].update({k: self.check_type(v)})
183 
184         self.record[transformed].update(info)
185 
186     def check_type(self, data):
187         if type(data).__module__ == 'numpy':
188             data = data.tolist()
189         elif isinstance(data, (pd.DataFrame, pd.Series)):
190             data = data.values.tolist()
191         return data
192 
193     def save_json(self, name):
194         with open('./results.json.tmp', 'a') as file:
195             try:
196                 file.write(json.dumps(self.record))
197             except Exception as e:
198                 logger.warn(f'Cannot save ndarray into JSON, error
    :{e}, {self.record}')
199             file.write('\n')
200 
201 
202 class DeTrendSeason:
203 
204     def __init__(self):
205         self.a = None
206         self.b = None
```

```
207
208     def detrend(self, insample_data):
209         """
210         Calculates a & b parameters of LRL
211         :param insample_data:
212         :return:
213         """
214         x = np.arange(len(insample_data))
215         self.a, self.b = np.polyfit(x, insample_data, 1)
216         return [insample_data[i] - ((self.a*i) + self.b) for i in
    range(len(insample_data))]
217
218     def add_trend(self, ts, forecast, fh=1):
219         for i in range(0, fh):
220             forecast[i] = forecast[i] + ((self.a * (len(ts) + i +
    1)) + self.b)
221         return forecast
222
223     def deseasonalize(self, original_ts, ppy):
224         original_ts = pd.Series(original_ts)
225         """
226         Calculates and returns seasonal indices
227         :param original_ts: original data
228         :param ppy: periods per year
229         :return:
230         """
231         if self.seasonality_test(original_ts, ppy):
232             # print("seasonal")
233             # ==== get moving averages
234             ma_ts = self.moving_averages(original_ts, ppy)
235
236             # ==== get seasonality indices
237             le_ts = original_ts * 100 / ma_ts
238             le_ts = np.hstack((le_ts, np.full((ppy - (len(le_ts) %
    ppy)), np.nan)))
239             le_ts = np.reshape(le_ts, (-1, ppy))
240             si = np.nanmean(le_ts, 0)
241             norm = np.sum(si) / (ppy * 100)
242             si = si / norm
```

```
243         else:
244             si = np.full(ppy, 100)
245 
246         return si
247 
248     def moving_averages(self, ts_init, window):
249         """
250         Calculates the moving averages for a given TS
251         :param ts_init: the original time series
252         :param window: window length
253         :return: moving averages ts
254         """
255         if len(ts_init) % 2 == 0:
256             # ts_ma = pd.rolling_mean(ts_init, window, center=True
    )
257             ts_ma = ts_init.rolling(window, center=True).mean()
258             ts_ma = ts_ma.rolling(2, center=True).mean()
259             ts_ma = np.roll(ts_ma, -1)
260         else:
261             # ts_ma = pd.rolling_mean(ts_init, window, center=True
    )
262             ts_ma = ts_init.rolling(window, center=True).mean()
263 
264         return ts_ma
265 
266     def seasonality_test(self, original_ts, ppy):
267         """
268         Seasonality test
269         :param original_ts: time series
270         :param ppy: periods per year
271         :return: boolean value: whether the TS is seasonal
272         """
273         s = self.acf(original_ts, 1) ** 2
274         for i in range(2, ppy):
275             s = s + (self.acf(original_ts, i) ** 2)
276 
277         limit = 1.645 * (np.sqrt((1 + 2 * s) / len(original_ts)))
278 
279         return (np.abs(self.acf(original_ts, ppy))) > limit
```

```
280
281     def acf(self, data, k):
282         """
283         Autocorrelation function
284         :param data: time series
285         :param k: lag
286         :return:
287         """
288         m = np.mean(data)
289         s1 = 0
290         for i in range(k, len(data)):
291             s1 = s1 + ((data[i] - m) * (data[i - k] - m))
292 
293         s2 = 0
294         for i in range(0, len(data)):
295             s2 = s2 + ((data[i] - m) ** 2)
296 
297         return float(s1 / s2)
298 
299     # remove seasonality
300     def remove_seanson(self, ts, freq):
301         seasonality_in = self.deseasonalize(ts, freq)
302 
303         for i in range(0, len(ts)):
304             ts[i] = ts[i] * 100 / seasonality_in[i % freq]
305         return ts
306 
307     def add_season(self, ts, forecast, freq, fh=1):
308         seasonality_in = self.deseasonalize(ts, freq)
309 
310         for i in range(len(ts), len(ts) + fh):
311             forecast[i - len(ts)] = forecast[i - len(ts)] *
    seasonality_in[i % freq] / 100
312         return forecast
```

Listing 3: utils.py

### A.1.4 settings.py

```
1 import numpy as np
```

```
import tensorflow

try:
    from tensorflow.keras import Input
    from tensorflow.keras.callbacks import EarlyStopping
    from tensorflow.keras.layers import GRU, LSTM, Bidirectional, Dense, Dropout
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.optimizers import Adam
except:
    from keras import Input
    from keras.callbacks import EarlyStopping
    from keras.layers import GRU, LSTM, Bidirectional, Dense, Dropout
    from keras.models import Sequential
    from keras.optimizers import Adam

from sklearn import ensemble, linear_model, neighbors, neural_network, svm
from sktime.forecasting.arima import AutoARIMA
from sktime.forecasting.ets import AutoETS
from sktime.forecasting.naive import NaiveForecaster

from utils import Performance_metrics


class GRU_model:

    def __init__(self, layers=(10,), epoch=2000, batch_size=1, earlystop=True, bidirectional=False, dropout=False, lr=0.1, loss='mse'):
        self.dim = (None, 1)
        self.layers = layers
        self.epoch = epoch
        self.batch_size = batch_size
        self.earlystop = earlystop
        self.bidirectional = bidirectional
        self.dropout = dropout
        self.lr = lr
```

```
36
37          self.p = Performance_metrics()
38          if loss == 'symmetric_mean_absolute_percentage_error':
39              self.loss = self.p.estimators[loss]
40          else:
41              self.loss = loss
42
43          self.init_model = None
44          self.model = None
45
46      def set_params(self, **kwargs):
47          for k, v in kwargs.items():
48              if k == 'layers':
49                  self.layers = v
50              if k == 'feature_num':
51                  self.dim = (1, v)
52              if k == 'lr':
53                  self.lr = v
54              if k == 'dropout':
55                  self.dropout = v
56              if k == 'batch_size':
57                  self.batch_size = v
58
59          self.build()
60
61      def build(self):
62          self.init_model = Sequential()
63          self.init_model.add(Input(self.dim))
64          for i, neuron in enumerate(self.layers):
65              name = f'GRU-{i+1}'
66              if self.bidirectional:
67                  self.init_model.add(Bidirectional(GRU(units=neuron
    , return_sequences=True), name=name))
68              else:
69                  self.init_model.add(GRU(units=neuron,
    return_sequences=True, activation='relu', kernel_initializer='
    he_uniform', name=name))
70          if self.dropout:
71              self.init_model.add(Dropout(0.2))
```

```
72          self.init_model.add(Dense(int(self.layers[-1]/2),
    activation='relu', kernel_initializer='he_uniform', name='Dense
    '))
73          if self.dropout:
74              self.init_model.add(Dropout(0.2))
75          self.init_model.add(Dense(1, activation='linear',
    kernel_initializer='he_uniform', name='Output'))
76
77          self.init_model.compile(optimizer=Adam(learning_rate=self.
    lr), loss=self.loss)
78
79          self.model = self.init_model
80
81
82      def fit(self, train_x, train_y, verbose=0):
83          # (n_datapoint, row_each_time, n_feature)
84          train_x = train_x.reshape(train_x.shape[0], 1, train_x.
    shape[1])
85          if self.earlystop:
86              self.model.fit(train_x, train_y, epochs=self.epoch,
    batch_size=self.batch_size, shuffle=False, verbose=verbose,
    callbacks=[EarlyStopping(monitor='loss', patience=3)])
87          else:
88              self.model.fit(train_x, train_y, epochs=self.epoch,
    batch_size=self.batch_size, shuffle=False, verbose=verbose)
89
90      def predict(self, test_x):
91          test_x = np.array(test_x).reshape(test_x.shape[0], 1,
    test_x.shape[1])
92          result = self.model.predict(test_x, verbose=0, batch_size=
    self.batch_size).ravel()
93          return result
94
95      def reset_model(self):
96          self.model = self.init_model
97
98  class LSTM_model(GRU_model):
99
100     def __init__(self, layers=(10, ), epoch=2000, batch_size=1,
```

```
    earlystop=True, bidirectional=False, dropout=False, lr=0.1,
    loss='mse'):
101         super().__init__(layers, epoch, batch_size, earlystop,
    bidirectional, dropout, lr, loss)
102 
103     def build(self):
104         self.init_model = Sequential()
105         self.init_model.add(Input(self.dim))
106         for i, neuron in enumerate(self.layers):
107             name = f'LSTM-{i+1}'
108             if self.bidirectional:
109                 self.init_model.add(Bidirectional(GRU(units=neuron
    , return_sequences=True), name=name))
110             else:
111                 self.init_model.add(LSTM(units=neuron,
    return_sequences=True, activation='relu', kernel_initializer='
    he_uniform', name=name))
112         if self.dropout:
113             self.init_model.add(Dropout(0.2))
114         self.init_model.add(Dense(int(self.layers[-1]/2),
    activation='relu', kernel_initializer='he_uniform', name='Dense
    '))
115         if self.dropout:
116             self.init_model.add(Dropout(0.2))
117         self.init_model.add(Dense(1, activation='linear',
    kernel_initializer='he_uniform', name='Output'))
118 
119         self.init_model.compile(optimizer=Adam(learning_rate=self.
    lr), loss=self.loss)
120 
121         self.model = self.init_model
122 
123 
124 regression_models = {
125     'ElasticNet': linear_model.ElasticNet,
126 
127     'LinearSVR': svm.LinearSVR,
128     'KNeighborsRegressor': neighbors.KNeighborsRegressor,
129 
```

```
130     'RandomForestRegressor': ensemble.RandomForestRegressor,
131 
132     'MLPRegressor': neural_network.MLPRegressor,
133     # 'GRU': GRU_model,
134     # 'LSTM': LSTM_model,
135 }
136 
137 forecasting_models = {
138     # 'AutoARIMA': AutoARIMA,
139     'AutoETS': AutoETS,
140     'NaiveForecaster': NaiveForecaster,
141 }
142 
143 # hyperparameters for tuning
144 params = {
145     'ElasticNet': {
146         'alpha': [0.01, 0.1, 1, 10],
147         'l1_ratio': np.arange(0.1, 1., 0.2),
148         'tol': [0.001],
149         'selection': ['random'],
150     },
151 
152     'LinearSVR': {
153         'epsilon': np.arange(0., 1., 0.4),
154         'C': [1, 5, 10, 20],
155     },
156     'KNeighborsRegressor': {
157         'n_neighbors': [2, 4, 8, 12, 16, 20],
158         'algorithm': ['auto'],
159     },
160 
161     'RandomForestRegressor': {
162         'n_estimators': [50, 100],
163         'max_features': [0.3, 1.0, 'sqrt'],
164         'bootstrap': [False],
165         'min_samples_leaf': [5],
166     },
167 
168     'MLPRegressor': {
```

```
169         'hidden_layer_sizes': [(8, 4, 2, ), (4, 2, ), (8, 4, ),
    (4, ), (8, ), ],
170         'activation': ['logistic', 'relu', 'tanh', 'identity'],
171         'shuffle': [False],
172         'max_iter': [500],
173         'learning_rate_init': [0.1],
174         'learning_rate': ['adaptive'],
175     },
176     'GRU': {
177         'layers': [(12, ), ],
178         'lr': [0.1],
179         'dropout': [True],
180     },
181     'LSTM': {
182         'layers': [(12, ), ],
183         'lr': [0.1],
184         'dropout': [True],
185     },
186 
187     'AutoARIMA': {
188         'sp': [7], # daily
189     },
190     'AutoETS': {
191         'auto': [True],
192     },
193     'NaiveForecaster': {
194         'strategy': ['last'],
195     },
196 }
```

Listing 4: settings.py