Directional Change Framework for Target Transformation in Time Series Forecasting

Yu-Kuan, Lin

Supervisor: Dr. Bart de Keijzer

Faculty of Natural, Mathematical & Engineering Sciences

The Department of Informatics - MSc in Computational Finance

Abstract

We propose and evaluate Directional Change Transformation (DCT). DCT is a novel target transformation method for time series forecasting based on the Directional Change Intrinsic Time Framework. DCT has several interesting features. First, it has denoising and smoothing ability while capturing the extreme events in time series. Secondly, it has feature generation ability which can be seen as a characterisation of time series. Thirdly, DCT is parameterised in such a way that it is very flexible and can enable tremendous customisation. Based on our large-scale experiment, we find that applying DCT can help improve model performance in forecasting tasks.

Originality Avowal

Acknowledgements

This postgraduate study has been an extraordinary journey for me. I want to take this opportunity to express my appreciation to the individuals and institutions that have inspired, guided, supported and accompanied me along this way. I first want to thank my dissertation supervisor Dr. Bart de Keijzer. Dr. Keijzer has been very supportive throughout my dissertation work. Moreover, he inspires me that it takes courage to explore the unknown, and we should never forget to respect ourselves for that. Also, I want to thank my external supervisor Dr. Chrystalla Pavlou from TurinTech AI, for her patience and tirelessly support for all my queries. Her guidance paves the way for the completion of this report, and I cannot thank her enough for that.

In addition, I want to thank King's College London for fostering such an excellent, academically inclined environment from which I benefited greatly as a student. I am also immensely grateful to TurinTech AI for allowing me to work with some of the industry's brightest minds to extend the frontier of knowledge. In particular, I want to thank Dr. Lingbo Li and Dr. Buhong Liu for their exceptional coordination and for providing me with cutting-edge technology that makes my implementation of large-scale experiments possible.

I want to thank my cohorts in MSc Computational Finance, King's College London, for always being there for me and bringing out the best in me. Their companionship is a constant reminder to me that we are by no means alone, and we can always accomplish more together. Finally, I want to thank my family. Words cannot express how grateful I am for their unconditional support for me to pursue my dreams. They have shown and taught me the purest form of love I shall forever deem one of the most precious things I possess.

To the part in me that will stay a student forever.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Objectives

The primary objective of this report is to propose and evaluate the Directional Change Transformation (DCT), a novel target transformation method for time series forecasting based on the Directional Change (DC) Intrinsic Time Framework. Target transformation is a preprocessing method often used as an additional layer within time series modelling procedures. Instead of operating on the feature space of a regression problem like typical feature engineering methods, target transformation takes a step further and operates on the target to improve modelling performance. Metaphorically, preprocessing the feature space in modelling is like organising the ingredients to make cooking a dish easier for the chef. On the other hand, operating on the target is like having a better recipe. Due to such nature, target transformation is also referred to as problem statement transformation. DC Intrinsic Time Framework is used to study financial time series, especially high-frequency time series. It provides novel insights utilised by trading strategies and many market analyses. Learning its ability to extract information from time series motivates us to turn it into a target transformation method - we hope it can provide models with a better target statement to learn from in time series forecasting tasks.

1.2 Report Structure

In the next chapter, we give a comprehensive background theory on three topics related to our proposition. First, we address time series forecasting with Machine Learning regression models and many general notions regarding the procedure.

Secondly, building on the introduction, we cover how target transformation methods can be embedded into the general procedure of Machine Learning modelling; we also discuss important considerations for such a implementation. Thirdly, we elaborate on the DC Intrinsic Time Framework in-depth and provide detailed pseudocode for the Directional Change event identification algorithm. In Chapter 3, we survey the important literature regarding target transformation and the DC framework that established the ground for our exploration of bringing the two together. In Chapter 4, we elaborate on the technical details of the proposed DC Transformation and the experiment we used to investigate its characteristics. In Chapter 5, we present the results yielded from our experiment and the analyses about DCT. Finally, a conclusion is provided in Chapter 6. In addition, further information for our experimental settings can be found in Appendix A, and we briefly talk about the programme we create for this report in Appendix B.

Chapter 2

Background

2.1 Univariate Time Series Forecasting

2.1.1 General Notions of Time Series Forecasting

A time series is a sequential collection of random variables indexed by time. If the random variables are of one dimension, then the time series is considered univariate[1]. In this dissertation, we focus on univariate time series forecasting. Let $Y = \{Y_{t_i}\}_{i=1,2,\ldots,n}$ be an univariate time series with $Y_{t_i} \in \mathbb{R},\ \forall i,\ n \in \mathbb{N}$ and $t = \{t_i\}_{i=1,2,\ldots,n}$ being the set of time indices of Y (also referred to as the set of timestamps). Let $\mathcal{Y}_{t_k}$ be the largest information set available about Y at time point $t_k \in t$, e.g., we might have the observations of Y being known ($\{Y_{t_j} = y_{t_j}\}_{j=1,2,\ldots,k} \subset \mathcal{Y}_{t_k}$). Then in the context of modelling, for an unknown (random) target Y_{t_s} with $t_s > t_k,\ t_s \in t$, we can articulate the notion of forecasting as the following:

> Forecasting the value Y_{t_s} at time point t_k is to find a function $f(\mathcal{Y}_{t_k}) = \widehat{Y}_{t_s}$, such that $\widehat{Y}_{t_s}$ is a good estimation of Y_{t_s}.

We can develop all sorts of functions f for such forecasting objectives. Functions devised to serve the objective are referred to as models. In a general sense, the notion of modelling refers to the methodologies of devising a model that serves the objective well. In particular, for any arbitrary pair of t_k and t_s, we want to have a model $f(\mathcal{Y}_{t_k})$ such that it gives us a reliable estimation of Y_{t_s}.

[1]If the random variables are of dimension higher than one, then the time series is considered multivariate. Datasets in such form are also referred to as panel data.

Gap

Notice that a modelling objective is parameterised by the pair of time indices (t_k, t_s). The timestamp t_k directly affects the information set $\mathcal{Y}_{t_k}$ and thus determines the information the model f can utilise. The timestamp t_s, on the other hand, determines how far in the future we are forecasting. If the gap between the two timestamps is big, the model is asked to forecast further into the future. If the gap between the two timestamps is small, then the objective might be considered easier because we are only trying to look a tiny step ahead into the future. In order to better communicate and characterise the forecasting objective, we formalise this gap between the pair of timestamps as the *gap* and denote it as τ:

$$\tau = t_s - t_k$$

With such a notion of the gap, we can then articulate the forecasting objective as τ ahead forecasting. τ takes the format of the timestamps. Depending on the format of the timestamps, the objective can be one-step ahead forecasting or one-year ahead forecasting. We will go into topics concerning timestamps in one of the upcoming paragraphs.

Forecast Horizon

Observe that the target we have in the previous forecasting objective Y_{t_s} is a single value in the future. It is possible to generalise the target and have multiple targets in the future. The number of targets we try to forecast is called the *forecast horizon.* Having a forecast horizon equal to one is to forecast one value into the future, and having a forecast horizon equal to five is to forecast all five values into the future. To put it formally, let $\langle\cdot\rangle$ be a counting operation that counts the number of elements for a finite set, and let a task have a finite collection of unknown values $Y_S = \{Y_{s_1}, Y_{s_2}, \cdots\}_{s_1, s_2, \ldots \in t,\ s_1, s_2, \ldots > t_k}$. Then the forecast horizon of such a task is denoted as

$$h = \langle Y_S \rangle$$

In this report, we forcus on objectives with forecast horizon being one.

Timestamps

In a time series, the time index of a random variable carries information about the time point in which the random variable lives in the time domain. In some sense, the timestamps mark the 'location' of the random variables on a timeline. For example, a monthly revenue time series Y in an arbitrary year can have the set of months in a year as its timestamp set and be denoted as $Y = \{Y_{Jan}, Y_{Feb}, Y_{Mar}, \cdots, Y_{Dec}\}$. The time indices also tell us the time-relevance (geological relationship on a timeline) of the random variables with one another. In fact, the time-relevance of the random variables in a time series plays a crucial role in time series analysis. We often have to perform mathematical operations involving such relationships. An example is our coming up with the gap measurement we addressed in the previous paragraph. Another example is the calculation of the relative growth of the time series. The need for these math operations motivates modellers to devise innovative ways to define the timestamps because we cannot easily perform calculations on notations like *September* or *Friday*. To put it in math terms, what we often do is to have a mapping from physical timestamps to the real number line (or a subset of the real number set, say, the natural number set) and use the target set of this mapping as the timestamp set for math operations. In the next paragraph, we discuss some examples of such mapping.

Take the previous monthly revenue time series as an example; one simplest way is to index the time series chronologically with natural numbers $\{1, 2, 3, \cdots, 12\}$. The new index system allows for mathematical operations on the timestamps, such as addition. The objective of two-month ahead forecasting can be considered as two-step ahead forecasting with $\tau = 2$. Three-month moving average of the time series can now be of a generalised form of a three-step moving average. Another good example is the financial studies of stochastic processes, in which we often use the non-negative real line and adopt an annual scale, i.e., the starting point of the time series is indexed as 0, one month after that is indexed 0.0833, one-year time point is indexed 1.0, and so on. This is particularly useful when we expand the time series studies using Stochastic Differential Equations (SDE). Let the SDE of a stochastic process dY_t be given as

$$\frac{dY_t}{Y_t} = \mu_t dt + \sigma_t dW_t, \quad dW_t \sim N(0, dt).$$

dt in the drift term is now properly defined as a real number on which we can do all sorts of math operations (observe how dW_t is defined as a Brownian Motion that follows a Gaussian distribution with variance dt).

Time Heterogeneity

There are cases where the timestamps of the time series are not identically spaced between consecutive random variables. The previous example of monthly revenue in a year is one of them due to the months having different durations. Time series as such is technically referred to as being *time heterogeneous*[2]. Time heterogeneity can be an interesting source of information carried by the time series but can also be a major issue in time series studies. The following paragraph presents a common problem caused by time heterogeneity.

Calculating measurements related to the unit of time can be a problem with time heterogeneous time series. One common example of such measurement is the return used in finance. Return measures the relative change of the price (or, say, value) over a period of time with respect to its initial level. Several definitions can be drawn to the notion of return, but we will look at the simplest one as they all exhibit the same relationship with time heterogeneity. Let $Y = \{Y_{t_i}\}_{i=1,2,...n},\ n \in \mathbb{N}$ now be a time series of the price movement of an asset over time, and the time index set being $t = \{t_i\}_{i=1,2,...,n}$. The corresponding net return measure is given as

$$R_{t_i} = \frac{Y_{t_i} - Y_{t_{i-1}}}{Y_{t_{i-1}}}, \quad i \in \{2, 3, \cdots, n\}. \tag{2.1}$$

The net return R_{t_i} at time t_i is thus the relative price change of Y_{t_i} over the time period $t_i - t_{i-1}$ with respect to $Y_{t_{i-1}}$. Let $R = \{R_{t_i}\}_{i=2,3,...,n}$ be the time series of one-step net returns derived from Y. If t is equally spaced by, say, a day, the time series Y is time-homogeneous, and the time series R we calculated is a time series of daily net return of Y. Nevertheless, in the case where t is not equally spaced, the time series Y is time heterogeneous, and we no longer have a unified interpretation of the net returns R we calculated from the time series Y. This is an example of how time heterogeneity can complicate time series analyses. Due to the nature of financial markets, time heterogeneity in finance has been studied a lot. Popularising electronic systems in financial activities is also a huge contributing factor to time heterogeneity in finance. See Dacarogna et al. (2001) for more information.

2.1.2 Machine Learning Regression Modelling

This section discusses univariate time series modelling and addresses how we approach the articulated forecasting objective with Machine Learning (ML) regression modelling.

[2]Time heterogeneity is common in financial time series due to the nature of how financial markets work. For example, most markets are open only during working hours on working days. Another example is the high-frequency financial time series (see Dacarogna et al. 2001). *Time homogeneity* is the counterpart of time heterogeneity, specifying time series which have equally spaced timestamps

In addition to addressing the approach, we also provide relevant statistical notions that can be seen as the underlying theoretical foundation of the standard Machine Learning procedure.

Recall the forecasting objective is to come up with a model f capable of generating 'good' estimations of some target Y_{t_s} at time t_k using the provided information $\mathcal{Y}_{t_k}$. We will see in the later paragraphs that this is very similar to the process of solving for a Quasi-Maximum Likelihood Estimation (QMLE) in a statistical sense (White (1982)). In the context of modelling, the procedure is normally to gather the available information and formulate it into an optimisation problem: we define a fitness measure (like the 'L' in QMLE), which should be a function of the estimates $f(\mathcal{Y}_{t_k})$ and the target, and we then optimise the fitness as an objective function with respect to the model f in some algorithmic way (the 'M' in QMLE). The final output of such a procedure will be a model (function f) that optimally serves our objective (the 'E' in QMLE). In the remainder of this section, we discuss the general framework of how the procedure works.

The Sliding Window, Design Matrix and Target

In regression problems, we formulate a modelling environment using accessible information with the creation of the *design matrix* and *target*. This paragraph discusses how information is formulated into a modelling problem by developing the design matrix and target in univariate time series modelling. The formulation of the modelling environment depends heavily on the forecasting objective. Recall that the forecasting objective is to estimate Y_{t_s} at time point t_k, with the gap $\tau = t_s - t_k > 0$ and the model is only able to utilise the accessible information set $\mathcal{Y}_{t_k}$. This dataset is called the *training set*. For our modelling problem, we can only use what is provided in our training set $\mathcal{Y}_{t_k}$. Without loss of generality, for the target Y_{t_s}, we can have $\mathcal{Y}_{t_k} = \mathcal{Y}_{t_s-\tau}$[3]. In the event we have a one dimensional time series, $\mathcal{Y}_{t_s-\tau}$ is simply the collection of realised values of Y until time point $t_s - \tau = t_k$, namely

$$\mathcal{Y}_{t_s-\tau} = \{Y_{t_s-\tau} = y_{t_s-\tau}, Y_{t_{s-1}-\tau} = y_{t_{s-1}-\tau}, Y_{t_{s-2}-\tau} = y_{t_{s-2}-\tau}, \cdots, Y_{t_1} = y_{t_1}\}. \tag{2.2}$$

To make use of this long list of past observations, the idea is to create a sandbox in which we simulate the model making predictions. The environment in which the model makes a prediction is simply a mapping of the information it can use for a

[3]We embed the τ parameter characterising the objective into the information set accessible as $\mathcal{Y}_{t_s-\tau}$.

single target. We do this using the methodology called the *sliding window.* The sliding window is parameterised by a single constant parameter $\lambda,\ \lambda \in \mathbb{N},\ \lambda > 2$ that controls the width of the window. The decision of λ should take into account modelling configurations, including the size of our training set, forecast horizon, gap, and information we want the model to use in a single prediction task. Once λ is decided, we move the window chronologically along $\mathcal{Y}_{t_s-\tau}$. For every step of the window, we create an independent prediction scenario for the model using the values contained within the window - the final value in a single window is the target for horizon one forecasting, and the rest of the values are potentially accessible to the model for its prediction depending on τ. We then have numerous such forecasting instances for the model from which we can foster the whole sandbox. The sandbox consists of the design matrix $\mathbf{X}$ and the target $\mathbf{y}$. Given the time series noted in expression 2.2 and the forecast horizon $h = 1$, $\mathbf{y}$ and $\mathbf{X}$ can be formulated as

$$\mathbf{y} = \begin{bmatrix} y_{t_\lambda} \\ y_{t_{\lambda+1}} \\ \cdot \\ \cdot \\ y_{t_{k-1}} \\ y_{t_k} \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} y_{t_\lambda-\tau} & y_{t_{\lambda-1}-\tau} & \cdots & y_{t_1} \\ y_{t_{\lambda+1}-\tau} & y_{t_\lambda-\tau} & \cdots & y_{t_2} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ y_{t_{k-1}-\tau} & y_{t_{k-2}-\tau} & \cdots & y_{t_{k-\lambda}-\tau} \\ y_{t_k-\tau} & y_{t_{k-1}-\tau} & \cdots & y_{t_{k-\lambda+1}-\tau} \end{bmatrix}.$$

The target $\mathbf{y}$ is a $k-\lambda+1$ by 1 column matrix and the design matrix $\mathbf{X}$ is of $k-\lambda+1$ by λ. In this training environment, the model has $k-\lambda+1$ predictions to make, each being to use a row vector in $\mathbf{X}$, denoted as $\mathbf{X}_i,\ i \in \{1, 2, \cdots, k-\lambda+1\}$ and estimate the corresponding row element in $\mathbf{y}$, denoted as $\mathbf{y}_i$. The idea is to find a model that best maps the rows in $\mathbf{X}$ to elements in $\mathbf{y}$ in general, and we say this is our best model f that serves the objective of a τ ahead forecasting task given the time series we have.

Given a time series being the training set, we denote such formulation of the design matrix and target as formulator $\mathcal{S}(\mathcal{Y}; \tau, h, \lambda)$. Formulator $\mathcal{S}$ is an operator parameterised by objective configurations (gap τ and forecast horizon h) and modelling configuration (width of sliding window λ). Then given the parameters, $\mathcal{S}$ takes the training set and creates the according target $\mathbf{y}$ and design matrix $\mathbf{X}$.

The Number of Lags

One remark regarding the design matrix is the notion of the *number of lags.* The number of lags describes how many latest observations the model is allowed to use for a single forecasting task, i.e., the number of columns in $\mathbf{X}$. Given a forecasting objective parameterised by (t_k, t_s) and modelling configurations, the number of lags equals to $\lambda - h - \langle \{t_i\}_{i=k+1,k+2,\dots,s-1} \rangle$[4]. Its naming originates in autoregressive estimation in time series studies. Such estimation aims at finding the optimal order of the autoregressive feature of a time series. This is equivalent to finding the optimal number of latest observations the model should use for a single prediction. Typically, such autoregressive order is referred to as the *lag*.

Modelling

We start by describing what a model is in more detail. Given the forecasting objective, modelling configuration, available information $\mathcal{Y}$ and the modelling environment formulator $\mathcal{S}$, the forecasting objective is now transformed into coming up with a model f that 'best' maps the rows in $\mathbf{X}$ to the corresponding element in $\mathbf{y}$. This is formulated into an optimisation problem.

Consider f as an arbitrary machine learning regression model with a known structure. Knowing the structure of f implies we know the structure of its parameter set and how f maps $\mathbf{X}_i$ to $\mathbf{y}_i$ for some i. Let θ be the mapping that generates the parameters that go into f. The output of θ is parameterised by its input set; let it be ϕ. Parameter set like ϕ is called the *hyperparameters* - it characterises the parameters of f. The model f in our forecasting objective can thus be noted as

$$f(\theta(\phi); \mathbf{X}) = \widehat{\mathbf{y}} \sim \mathbf{y}.$$

To quantify the performance of such an estimation, the model f comes with a designating loss function $\mathcal{E}$. The loss function takes the estimations generated by f and returns a real number signifying how bad they fit the target. The error generated by the model is negatively correlated to its fitness for the prediction task. We can finally formulate our modelling objective as

$$\arg_{\theta(\phi)} \min \mathcal{E}(\widehat{\mathbf{y}}, \mathbf{y}),\ \widehat{\mathbf{y}} = f(\theta(\phi); \mathbf{X}). \tag{2.3}$$

This process is what we call *model training.* In particular, in ML operations, training the model according to $\mathbf{y}$ and $\mathbf{X}$ is also referred to as *fitting the model.*

To give an example, if f is a simple linear regression model, then the loss (error)

[4]$\langle \{t_i\}_{i=k+1,k+2,\dots,s-1} \rangle$ counts the number of instances in the timestamp set of which their corresponding values are not accounted for in the forecasting task due to the gap configuration.

function $\mathcal{E}$ is normally an L2 norm error function. In addition, we know θ gives a tuple of real numbers (known as weights) which the model uses to generate a linear combination of its inputs, in this case, a row matrix $\mathbf{X}_i$. Specifications of θ are then controlled by its input ϕ, e.g., whether there is an intercept term, specifications of the loss function, or the number of elements of the tuple[5]. Not knowing the exact values of θ (and certainly ϕ as well) describes the state of the model f being untrained. Then the modelling objective is to find ϕ and θ such that the error function $\mathcal{E}(f(\theta(\phi); \mathbf{X}), \mathbf{y})$ is minimised.

The Statistical Resemblance

Analogously, training a machine learning model can be put into statistical terms. In particular, it resembles the Quasi-Maximum Likelihood Estimation (QMLE) process with some minor tweaks. Consider our objective with Y, but with the elements following an arbitrary distribution $\mathcal{D}$ characterised by θ, i.e., $Y_{t_i} \sim \mathcal{D}(\theta),\ \forall\ t_i \in t$[6]. Consider an arbitrary pair of timestamps (t_k, t_s) and let $\mathcal{F}_{Y_{t_s}|\mathcal{Y}_{t_k}}(\cdot; \theta)$ be the conditional joint probability density function (pdf) of the random variable Y_{t_s} given $\mathcal{Y}_{t_k}$ and θ. Then the expression

$$\mathcal{F}_{Y_{t_s}|\mathcal{Y}_{t_k}}(y_{t_s}|\mathcal{Y}_{t_k}; \theta)$$

describes the probability of observing $Y_{t_s} = y_{t_s}$ conditional on the past observations and parameter θ. Let the value of $Y_{t_s} = y_{t_s}$ be given, then the objective of QMLE is to find the θ conditional on observing $\mathcal{Y}_{t_k}$, under which y_{t_s} is most likely to be observed. We can formalise such objectives as

$$\arg_\theta \max \mathcal{F}_{Y_{t_s}|\mathcal{Y}_{t_k}}(\theta; y_{t_s}|\mathcal{Y}_{t_k}).$$

Notice how the variable is now θ while the observations are given. The new objective function to be optimised is called the *likelihood function*, denoted as $\mathcal{L}(\theta)$. $\mathcal{L}(\theta)$ is essentially still a pdf. It returns the probability of observing y_{t_s} conditional on $\mathcal{Y}_{t_k}$ with the parameter θ. Maximising such likelihood function with respect to θ is thus equivalent to finding a θ dictating the generating mechanism of the random variable Y_{t_s} such that it is most likely to be observed as y_{t_s}.

[5]Note that the number of elements in the tuple depends on the number of lags we have in making the design matrix $\mathbf{X}$, i.e., $\lambda \in \phi$.

[6]The 'Quasi-' simply means we do not know whether the distribution $\mathcal{D}$ is Gaussian or not (see White 1982).

Training a machine learning model bears some resemblance to QMLE. Training a model aims to find a mechanism to reproduce the target as the observation. At the same time, QMLE tries to find the set of parameters characterising the distribution of the target variable conditional on its past realisations. Both methodologies tackle the problem with an optimisation framework involving an objective function: QMLE utilises the probability density function (pdf) that comes with a random variable with a known distribution. ML models have their own associated loss functions to evaluate the goodness of the estimation. The outcome of training an ML model is to have a deterministic function that generates the target and serves the purpose of forecasting. On the other hand, QMLE yields a set of parameters dictating the underlying distribution of the target variables, i.e., you end up having a recipe for constructing a probabilistic distribution, not deterministically generating a target value. We hope this section contributes to a better theoretical understanding of ML modelling in terms of statistical analysis.

Model Selection with Validation

In the phase of training a model using $\mathcal{Y}_{t_k}$, notice that we have to first have the hyperparameters ϕ before we actually start searching for optimal θ. Such process of finding ϕ before actually fitting the model with respect to θ is called *model selection* or *hyperparameter tuning*. The way we approach model selection starts by selecting a subset of the training set with respect to the row and calling it the *validation set*[7]. The validation set acts as an unseen dataset we will use to validate the hyperparameter of choice. Let the size of the validation set be $v,\ v \in \mathbb{N},\ v \ll k - \lambda + 1$[8]. We then try (validate) whether a ϕ candidate works fine using the validation set by devising a validation function $\mathcal{V}$. For a given hyperparameter candidate ϕ_0, we know the exact structure of $\theta(\phi_0)$ such that we can run the optimisation regime, i.e., model training using equation 2.3. Then for a single ϕ_0, we utilise the training set $\mathcal{Y}_{t_k}$, decide v and we do the *validation* illustrated as algorithm 1. The result for validating a single hyperparameter ϕ_0 is the validation score V_{ϕ_0} we compute in the final step of the algorithm. Normally, we will come up with a set of k candidate hyperparameters $\phi = \{\phi_i\}_{i=1,2,\ldots,k},\ k \in \mathbb{N}$ and repeat such validation process k times and yield k validation scores $\{V_{\phi_i}\},\ \forall i \in \{1, 2, \cdots, k\}$. We then take the hyperparameter set

[7]The choice of validation set is usually a proportion of the latest instances from the training set, say, the latest ten instances or the latest ten percents of instances of the training set.

[8]The decision of v depends on the size of our training dataset. The bigger v we wish to have, the less data is left for the training, but the more data is used for validation. v is a part of modelling configuration.

Algorithm 1 Model Validation

```
Let a validation measure $\mathcal{V}$ be given.
Let $L$ be an empty list.
Let $s = k - \lambda + 1 - v$ be the first index of the validation set.
for $i \in \{0, 1, 2, \cdots, v-1\}$ do
    $\mathbf{y}_{train}, \mathbf{X}_{train} \leftarrow \mathcal{S}(\mathcal{Y}_{t_{s-1+i}}; \tau, h = 1, \lambda)$
    Do training: $\theta(\phi_0)_0 \leftarrow \arg_{\theta(\phi_0)} \min \mathcal{E}(\widehat{\mathbf{y}}_{train}, \mathbf{y}_{train}),\ \widehat{\mathbf{y}}_{train} = f(\theta(\phi_0); \mathbf{X}_{train})$
    $\mathbf{X}_{val} \leftarrow \{y_{t_{s+i-\tau+j}}\}_{j=0,1,\ldots,\lambda-1}$
    Make prediction: $\widehat{\mathbf{y}}_{val} \leftarrow f(\mathbf{X}_{val}; \theta(\phi_0)_0)$
    $\mathbf{y}_{val} \leftarrow y_{t_{s+i}}$ (validation target)
    Compute validation score for the $\theta(\phi_0)_0$: $\nu \leftarrow \mathcal{V}(\widehat{\mathbf{y}}_{val}, \mathbf{y}_{val})$
    Store the validation score $\nu$ in list $L$
end for
Let $V_{\phi_0} \equiv \frac{1}{v}\sum L$ be the mean of the scores stored in list $L$.
```

with the most preferable fitness score

$$\phi_{best} = \arg_{\phi_i} \max\{V_{\phi_i}\}_{i=1,2,\ldots,k}$$

and say ϕ_{best} is the optimal hyperparameter set to use in our modelling problem. The set of k candidate hyperparameters ϕ is referred to as the *hyperparameter space*. And hyperparameter tuning is the act of searching for an optimal hyperparameter in the hyperparameter space. In practice, there are plenty of frameworks developed for such optimisation, e.g., grid search, random search, gradient-based search, and evolutionary algorithms like genetic programming.

Model Evaluation - Testing

In the event we wish to conclude the model's performance using some measure, this is called *model evaluation* or *testing the model*. Testing is done similarly to the validation we addressed in model selection (notice how we use a validation score to measure the performance of a candidate hyperparameter). In some cases, we could simply use the validation score of the best-performing hyperparameter $V_{\phi_{best}}$ because this score is also a concluding measure of our model performance. However, if we want to actually simulate the situation in which our model is put into production, then we can use a part of the training data as unseen data to the model and perform a single round of validation. We refer to this as model evaluation, and the dataset used as the *test set*. Let κ denote the size of the test dataset just as we have to decide the size of the validation set. Since the validation set and the test set should not overlap and model selection comes before testing, we have to push the validation set and let the latest κ instances in the training set be the test set. Our usage of available information $\mathcal{Y}_{t_k}$ is now turned and renamed into

$$\begin{aligned} \mathcal{Y}_{t_k,train} &= \{y_i\}_{i=1,2,\ldots,k-\kappa-v} \\ \mathcal{Y}_{t_k,val} &= \{y_i\}_{i=k-\kappa-v+1,k-\kappa-v+2,\ldots,k-\kappa} \\ \mathcal{Y}_{t_k,test} &= \{y_i\}_{i=k-\kappa+1,k-\kappa+2,\ldots,k} \end{aligned}$$

To sum up, the *training set* now consists of $k-\kappa-v$ instances, the *validation set* consists of v instances and the *test set* consists of the latest κ instances. The three sets take up all the information we have in our accessible information set $\mathcal{Y}_{t_k}$. Given that we already have the best-performing hyperparameter $V_{\phi_{best}}$, we can simply do a single round of validation described in Algorithm 1 using the test set and come up with a score that concludes the model performance. The score is referred to as the *test score.* And that concludes a typical modelling operation.

Retrain Window

During the single round of model selection and evaluation of a model, the model is to be fitted v times for model selection and κ times for model evaluation. This can be quite computationally expensive when the validation or testing set is large. Therefore, there is the notion of *retrain window.* The retrain window marks a window length in which the model does not get refitted again - it simply uses the new observations for new predictions. Take model selection as an example; if there are a hundred validation points, the model will be fitted a hundred times to come up with a single validation score with the retrain window being zero. Say we implemented a retrain window with the size of ten. Then through the hundred validation points, the model is only refitted every ten points. That leads to approximately ten times less computational cost for model selection. Setting a retrain window is a trade-off between computational efficiency and exploiting the training dataset.

Data Leakage

In this section, we briefly cover the topic of *data leakage* in ML modelling. Recall that our modelling process operates in a hypothetical sandbox environment that simulates instances in which the model tries to fulfil a specific objective. Such a simulated environment should be coherent to the objective and real-world situation - the model should not be trained in an environment where it is allowed to do things that it cannot do in the real world. One of the most common incidents where such a ‘breach’ is accidentally incorporated into the construction is the allowance of the model to access information from beyond the time point where it should be, i.e., the model is allowed to peek into the future when making predictions. We refer to such a problem as *data leakage*. As outlandish as this may sound, it often happens, especially when complicated operations are embedded into the modelling procedure. The complexity of the modelling procedure tends to be positively correlated with the risk of data leakages. Therefore, modellers should be wary of data leakage when designing and building the modelling procedures for data leakage can seriously jeopardise the result.

2.2 Target Transformation in Univariate Time Series Forecasting

Building on the ML regression modelling we established in the previous section, we can now address the topic of target transformation in univariate time series forecasting. For a forecasting task on a time series $Y = \{Y_{t_i}\}_{i=1,2,\ldots}$, we try to devise a model f that deterministically produce predictions. We utilise all information available and formulate a modelling procedure that goes as the following:

1. Find out the available information set $\mathcal{Y}_{t_k}$.
2. Find out the forecasting configuration according to the objective: gap τ. (In this paper, we discuss $h = 1$).
3. Decide the modelling configuration: validation set size v, test set size κ, width of sliding window λ, and the validation function $\mathcal{V}$.
4. Choose a family of the model f.
5. Decide a hyperparameter space ϕ to search.
6. Perform model selection: conduct validation procedure 1 for all hyperparameters and choose the optimal one.
7. Finally, test the model with the optimal hyperparameter ϕ_{best} and we yield a deterministic function $f(\cdot;\theta(\phi_{best})_{best})$ with its test score.

Observe that aside from our appointing the model f and the hyperparameter set ϕ, the model we obtain through the procedure relies heavily on the information contained within the dataset $\mathcal{Y}_{t_k}$ precisely because of our formulation of the modelling procedure. In other words, the utility of our model depends heavily on the information set (dataset). This leads to many studies trying to find more clever ways to process the information before putting it into the optimisation operation to boost model performance. Such an additional layer of operation on datasets in the modelling procedure is called *data preprocessing*. Moreover, if the additional operation is performed only on the design matrix $\mathbf{X}$ such operations are referred to as *feature engineering*[9]. When the operation involves the target $\mathbf{y}$, it is called *target transformation* or *response transformation*[10]. In the rest of the section, we cover the technical notions of target transformation.

2.2.1 Implementation of Target Transformation

Transforming the Target

Let $\mathcal{T} : \mathbb{R}^p \longmapsto \mathbb{R}^q$ be an arbitrary target transformation that maps a p dimensional vector to a q dimensional vector. Normally we have $p >= q,\ p \approx q$[11]. Throughout the modelling procedure, target transformation is to be embedded before and possibly after every model fitting stated in Equation 2.3. Consider an arbitrary model fitting in the modelling procedure to be conducted with the information set $\mathcal{Y}_{t_k} = \{y_{t_i}\}_{i=1,2,...,k}$, we practice

$$\mathcal{Y}'_{t_k} = \mathcal{T}(\mathcal{Y}_{t_k})$$

and use the transformed time series $\mathcal{Y}'_{t_k}$ for the rest of the model fitting operation: we generate the training target $\mathbf{y}'_{train}$, training design matrix $\mathbf{X}'_{train}$ with $\mathcal{Y}'_{t_k}$ through

$$\mathbf{y}'_{train},\ \mathbf{X}'_{train} = \mathcal{S}(\mathcal{Y}'_{t_k}; \tau, h, \lambda)$$

and fit the model

$$\theta(\phi_0)_0 = \arg_{\theta(\phi_0)} \min \mathcal{E}(\widehat{\mathbf{y}}'_{train}, \mathbf{y}'_{train}),\ \widehat{\mathbf{y}}'_{train} = f(\theta(\phi_0); \mathbf{X}'_{train}).$$

[9]In regression operations, the columns in the design matrix are referred to as features or independent variables.

[10]In regression operations, the target, being a column matrix, is also called the response variable or dependent variable

[11]If $\mathcal{T}$ reduces the dimension, adopting such a target transformation reduces the data point we can use during training. For example, a five-step moving average transformation causes the loss of the first 4 data points. Seeing that we normally prefer as many data points as possible, such loss of data points is a cost that might come with the implementation of the technique.

After the optimisation using the transformed time series, we make the input used for generating a validation prediction

$$\mathbf{X}'_{val} = \{y'_{t_{k-i}}\}_{i=0,1,\ldots,\lambda-1}$$

and make a prediction using the trained model

$$\widehat{\mathbf{y}}'_{val} = f(\mathbf{X}'_{val}; \theta(\phi_0)_0).$$

The usual next step is to make the validation target and validation design matrix and compute the fitness score of $\theta(\phi_0)_0$. The fitness score is computed by comparing the prediction generated by f using the transformed input against a validation target[12]. We can have validation target

$$\mathbf{y}_{val} = y_{t_k+\tau}$$

We want to highlight that, with or without target transformation throughout the modelling process, the validation target should always be untransformed because we want to maintain the forecasting objective. Before computing the fitness score, we inevitably have to consider the question of whether f's prediction using transformed input is comparable to an untransformed target. This obviously depends on the nature of the transformation $\mathcal{T}$. If the transformed prediction is uncomparable to the untransformed target, we need an additional operation to reverse the scale of the prediction generated by f back to its comparable untransformed level. Such reversing operation is called the *back transformation* which we cover in the next section.

Back Transformation

The major criterion causing the need of the back transformation is that $\mathcal{T}$ changes the scale of the time series, i.e., $\mathcal{Y}_{t_k}$ and $\mathcal{Y}'_{t_k}$ have different scales such that $\mathbf{y}_{val}$ and $\widehat{\mathbf{y}}'_{val} = f(\mathbf{X}'_{val}; \theta(\phi_0)_0)$ are incomparable. Consider $\mathcal{T}^{(b-)}$ be the corresponding back transformation operator that restores our prediction to its original comparable level. We can do

$$\widehat{\mathbf{y}}^{b-}_{val} = \mathcal{T}^{(b-)}(\widehat{\mathbf{y}}'_{val})$$

and use this back-transformed prediction to compute the fitness score for whatever validation purposes given by

$$\mathcal{V}(\widehat{\mathbf{y}}^{b-}_{val}, \mathbf{y}_{val}).$$

[12]Note that f operates on transformed values because it was trained on transformed values.

And that concludes the implementation of target transformation.

In addition, we want to highlight some technical challenges concerning coming up with such a back transformation. Intuitively, such back transformation should base on the inverse mapping of $\mathcal{T}$. This is indeed sensible but often not sufficient because $\mathcal{Y}'_{t_k}$ might not contain all the information needed to reconstruct the set $\mathcal{Y}_{t_k}$ (recall that $\mathcal{T}$ might reduce the dimension of its input). Fortunately, we are not banned from consulting the information in $\mathcal{Y}_{t_k}$. We can thus formulate $\mathcal{T}^{(b-)}$ with the help of $\mathcal{Y}_{t_k}$. The second problem, which might be considered the major problem, is that $\mathcal{T}$ might not be invertible. In such circumstances, if resorting to numerically estimating the inverse of $\mathcal{T}$ or coming up with a sensible mapping are not feasible, then one might have to consider $\mathcal{T}$ inapplicable as a target transformation technique. The feasibility of back transformation is thus one of the main concerns when choosing the transformation techniques.

2.2.2 An Example - Log-differencing

This section provides a simple demonstration of implementing a target transformation. An usual target transformation used in financial studies is the *log-difference* transformation. It has the effect of fostering stationarity to the original time series. The log-difference operation is a mathematical simulation of the net return we mentioned in Equation 2.1. Let the log-difference operation be denoted as $\mathcal{T}^{(logr)}$ and given by[13]

$$\mathcal{Y}^{(logr)}_{t_k} = \mathcal{T}^{(logr)}(\mathcal{Y}_{t_k}) = \begin{cases} \emptyset & \text{if } i = 1 \\ \log(\frac{y_{t_i}}{y_{t_{i-1}}}) & \text{if } i = 2, 3, \cdots, k \end{cases} \tag{2.4}$$

Observe that for an arbitrary $i \in \{2, 3, \cdots, k\}$

$$\lim_{R_{t_i} \to 0} R_{t_i} - \log(\frac{y_{t_i}}{y_{t_{i-1}}}) = 0.$$

This is why log-differencing is a mathematical simulation of the net return and often is referred to as the *log return* (hence our notation *logr*). We move on to generating the training target and training design matrix through

$$\mathbf{y}^{(logr)}_{train}, \ \mathbf{X}^{(logr)}_{train} = \mathcal{S}(\mathcal{Y}^{(logr)}_{t_k}; \tau, h = 1, \lambda)$$

[13]Notice that the log-difference mapping does lose one degree of dimension due to the first element in its output being undefined. This causes us to have one training instance short compared to not implementing target transformation.

and perform the optimisation

$$\theta(\phi_0)_0 = \arg_{\theta(\phi_0)} \min \mathcal{E}(\widehat{\mathbf{y}}_{train}^{(logr)}, \mathbf{y}_{train}^{(logr)}),\ \widehat{\mathbf{y}}_{train}^{(logr)} = f(\theta(\phi_0); \mathbf{X}_{train}^{(logr)}).$$

We then make the input matrix

$$\mathbf{X}_{val}^{(logr)} = \{y_{t_{k-i}}^{(logr)}\}_{i=0,1,\ldots,\lambda-1}$$

and make a prediction using the trained model

$$\widehat{\mathbf{y}}_{val}^{(logr)} = f(\mathbf{X}_{val}^{(logr)}; \theta(\phi_0)_0).$$

Obtain the validation target that we use for validation

$$\mathbf{y}_{val} = y_{t_k+\tau}.$$

Before we go into computing the validation score, notice that despite the scale $\mathcal{Y}_{t_k}$ originally was, $\mathcal{Y}_{t_k}^{(logr)}$ is now of a scale around 0 for the normalising effect. This is one of the cases where we need a complementing back transformation because our model operates on a $(logr)$ scale, which is not comparable to the untransformed validation target. We have to devise a back transformation to bridge the spread between two incomparable scales. We can have the back transformation be denoted as $\mathcal{T}^{(b-logr)}$ and given by

$$\mathcal{T}^{(b-logr)}(\widehat{y}_{t_{k_i}}^{(logr)}; \mathcal{Y}_{t_{k_i}}) = \begin{cases} y_{t_1} & \text{if } i = 1 \\ e^{\widehat{y}_{t_{k_i}}^{(logr)}} \times y_{t_{k_{i-1}}} & \text{otherwise} \end{cases}. \tag{2.5}$$

Then we back-transform the prediction

$$\widehat{\mathbf{y}}_{val}^{(b-logr)} = \mathcal{T}^{(b-logr)}(\widehat{\mathbf{y}}_{val}^{(logr)}; \mathcal{Y}_{t_k})$$

to use it for fitness computing

$$\mathcal{V}(\widehat{\mathbf{y}}_{val}^{(b-logr)}, \mathbf{y}_{val}).$$

And that concludes the implementation of log-difference target transformation in a single model fitting. Observe that log-differencing is essentially an invertible function. However, the construction of the back transformation in Equation 2.5 involves the use of a subset of the original training set $\mathcal{Y}_{t_{k_i}}$. This is because $\mathcal{T}^{(logr)}$ retains only the relative information among the values of the original time series[14]. Therefore, it is not possible to reconstruct the original scale just by using $\mathcal{Y}_{t_k}^{(logr)}$. We hope this demonstration provides a vivid image of how target transformation is implemented.

2.2.3 Discussions

In this section, we discuss some notions concerning target transformation. Implementing target transformation can further complex the whole modelling procedure. In particular, transformations are often parameterised as they essentially are functions. Finding the optimal parameters for the transformation can make the training process much more expensive. Let ω be the set of parameters characterising a transformation. As ω does not belong to θ that deterministically affects the output of f, ω has to be considered as a part of the hyperparameters ϕ just like modelling configurations such as sliding window size λ. Including ω increases the dimension of ϕ and further enlarges the hyperparameter space to search during model selection. This is a potential cost regarding training efficiency when implementing target transformation.

As we will cover in the later Section 3.1, numerous target transformation methods deal with different types of problems. In common practice, it is not rare that the target transformation $\mathcal{T}$ consists of multiple functions chained together to solve multiple problems induced by the dataset. In these situations, such a chain of operations has to be appropriately organised, for instance, in terms of their order, as their effects might influence one another. Last but not least, as mentioned in Section 2.1.2, increasing the complexity of the modelling process can increase the risk of data leakage, and this certainly applies to implementing target transformation as well. The modeller should be extra careful in the design and implementation in this regard.

2.3 Directional Change Intrinsic Time Framework

In this section, we dive into the technical details of the Directional Change (DC) Intrinsic Time Framework. To elaborate on the naming, 'Directional Change' describes the *Directional Change Events* identified from a given time series by an algorithm and 'Intrinsic Time' is one of the key characteristics of the DC events. We start by addressing such an algorithm in Section 2.3.1, and we discuss the framework and its intrinsic time ideology in Section 2.3.2.

[14]Log-differencing does not keep track of the original level of the values. We can only induce their relative change with respect to others from its output set.

2.3.1 The Algorithm for Directional Change Events

The algorithm in the DC framework operates much like a state machine. Given a univariate time series, it classifies every entry in the time series into *states*. The algorithm's output is thus the same time series but with a state assigned to each entry by the algorithm. The primary goal of such an algorithm is to provide information on directional change events. Moreover, the assignment of states is also the source of many other interesting analyses and implementations carried out under the DC framework. Let an arbitrary time series be $\mathcal{Y}_{t_k} = \{y_{t_i}\}_{i=1,2,\ldots,k}$. Then the algorithm $\mathcal{A}^{(DC)}$ can be noted as

$$\mathcal{Y}_{t_k}^{(DC)} = \mathcal{A}^{(DC)}(\mathcal{Y}_{t_k}; \delta, \mathcal{R}(x;y)) = \{(y_{t_i}, s)\}_{i=1,2,\ldots,k}, \ s \in S$$

where

- $\delta = (\delta_{down}, \delta_{up})$ is a pair of threshold values parameterising the classification
- $\mathcal{R}(x;y)$ is a measure of change of value x with respect to a given reference y[15].
- S is a set of possible states identified by the algorithm.
- An entry $(y_{t_i}, s) \in \mathcal{Y}_{t_k}^{(DC)}$ signifies the value y_{t_i} being classified as state s by $\mathcal{A}^{(DC)}$.

Having consecutive values assigned to the same state means these values form an event with duration. Such events can be *Bullish Trend*, *Bullish Overshoot*, *Bearish Trend*, or *Bearish Overshoot*. By construction of the algorithm, bullish and bearish trends appear alternatively with the gap being filled by an overshoot following the direction of the former trend. An overshoot is essentially the residual of a trend before reaching the start of the next opposite-direction trend. For example:

$$\begin{aligned}&\text{bullish trend} \to \text{bullish overshoot} \to \text{bearish trend} \to \text{bearish overshoot}\\ &\to \text{bullish trend} \to \text{bullish overshoot} \to \text{bearish trend} \cdots .\end{aligned}$$

In order to mark the events, set S consists of seven possible states that could be assigned to the values in $\mathcal{Y}_{t_k}$:

1. Local Extreme: The starting point of a time series is identified as a local extreme. Other than the starting point, the rest of the local extreme points are the start of a trend and also the end of the previous overshoot. One

[15]For instance, the net return is $\mathcal{R}(x;y) = \frac{x-y}{y}$.

thing to highlight is that, except for the starting point, all local extrema are only recognisable once its latter trend is identified. In other words, they are determined in hindsight.

2.,3. Bullish or Bearish Trend: The value is identified as experiencing a bullish or bearish trend.

4.,5. Bullish or Bearish Overshoot: Upon identifying a bullish or bearish trend, the price is at a bullish or bearish overshoot before it reaches a local extreme (or, say, the next trend with the opposite direction).

6.,7. Bullish or Bearish DC Confirmation: This value confirms the occurrence and ends a bullish or bearish trend. Note that the end of a bullish or bearish trend is also the end of a DC event and the start of the next one. This is identified by the value that has risen or decreased more than δ_{up} or δ_{down} compared to the previously spotted local extreme.

Figure 2.1 is a figure we found in Petrov et al. (2018) that illustrates the Directional Change Framework.

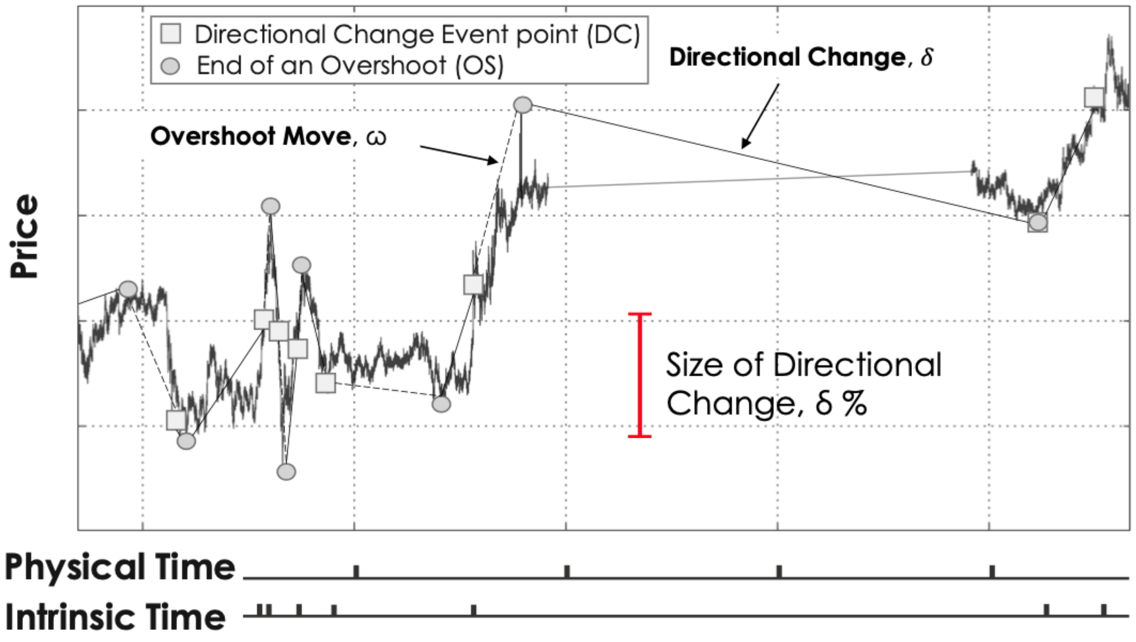

Figure 2.1: Illustration of DC Framework

This figure is a direct reference to the one in Petrov et al. (2018). This is an example of an exchange rate price time series and the directional change events identified using an arbitrary symmetric δ threshold.

See Algorithm 2 for the detailed pseudocode of $\mathcal{A}^{(DC)}(\mathcal{Y};\delta,\mathcal{R})$.

2.3.2 Discussions

A directional change event is a downturn or upturn event which is a combination of an overshoot and its consecutive trend[16], and the identification of DC events is the primary goal of the $\mathcal{A}^{(DC)}$ algorithm. Here are some remarks concerning the algorithm and its output.

1. Observe that a DC event is considered complete only when the time series has moved passed the threshold set by $\delta = (\delta_{down}, \delta_{up})$ according to measure $\mathcal{R}$. Therefore, higher δ values lead to fewer total DC events identified by the algorithm and vice versa.

2. Due to how $\mathcal{A}^{(DC)}$ is parameterised by δ, the number of DC events is negatively correlated to the volatility of $\mathcal{Y}_{t_k}$.

3. In the event where $\delta_{down} \neq \delta_{up}$, the algorithm is set to have different sensitivity to bullish and bearish trends.

4. $\mathcal{A}^{(DC)}$ is a deterministic mapping if $\mathcal{Y}_{t_k}$, δ and $\mathcal{R}$ are given.

5. As mentioned previously, except for the first one, identifying a local extreme can only be made until its following trend is confirmed. In time series analysis, we say that the process of identifying local extrema uses *look-ahead information.* This means that knowing the past values is not enough; classifying a local extreme does not happen in real-time as the algorithm loops through the values in $\mathcal{Y}_{t_k}$ chronologically.

6. Let $\mathcal{Y}^{(IT)}_{t_k}$ be the subset of $\mathcal{Y}^{(DC)}_{t_k}$ such that

$$\mathcal{Y}^{(IT)}_{t_k} = \{(y_{t_i}, s) | s = \text{DC Confirmation}\}_{i=1,2,\ldots,k}.$$

Then the timestamp set of $\mathcal{Y}^{(IT)}_{t_k}$, which we denote as $t^{(IT)}$, marks the time points where directional change events are confirmed. $t^{(IT)}$ is called the *Directional Change Event-based Intrinsic Time* of $\mathcal{Y}_{t_k}$ based on δ and $\mathcal{R}$.

[16]An overshoot and its consecutive trend have opposite directions.

Algorithm 2 The Algorithm for Directional Change Events

```
Input: a time series $\mathcal{Y}_{t_k} = \{y_{t_i}\}_{i=1,2,...,k}$
Input: a pair of downward and upward thresholds $\delta = (\delta_{down}, \delta_{up})$
Input: a return measure $\mathcal{R}(x; y)$ for a spot value $x$ with respect to a given reference
value $y$
Initialise current mode to *none*
Initialise local extreme $e$ to *none*
**for** $y_{t_i} \in \{y_{t_1}, y_{t_2}, \cdots, y_{t_k}\}$ **do**
    **if** $i$ is 1 **then**
        Update local extreme $e$ to $y_{t_i}$ and move on to the next iteration.
    **end if**
    **if** Current mode is *none* **then**
        $r \leftarrow \mathcal{R}(y_{t_i}; e)$
        **if** $r \geq \delta_{up}$ **then**
            Mark a bullish trend from the last local extreme to current $y_{t_i}$.
            Update current mode to *bullish* and move on to the next iteration.
        **else if** $r < \delta_{down}$ **then**
            Mark a bearish trend from the last local extreme to current $y_{t_i}$.
            Update current mode to *bearish* and move on to the next iteration.
        **end if**
        Mark $y_{t_i}$ to unidentified state.
    **end if**
    **if** Current mode is *bullish* **then**
        **if** $y_{t_i} \geq e$ **then**
            Update the local extreme $e$ to $y_{t_i}$.
        **end if**
        $r \leftarrow \mathcal{R}(y_{t_i}; e)$
        **if** $r \leq \delta_{down}$ **then**
            Update current mode to *bearish*.
            Mark a bearish trend from the local extreme to the current $y_{t_i}$.
            Move on to the next iteration.
        **end if**
        Mark $y_{t_i}$ as going through a bullish overshoot.
    **else if** Current mode is *bearish* **then**
        **if** $y_{t_i} \leq e$ **then**
            Update the local extreme $e$ to $y_{t_i}$.
        **end if**
        $r \leftarrow \mathcal{R}(y_{t_i}; e)$
        **if** $r \geq \delta_{up}$ **then**
            Update current mode to *bullish*.
            Mark a bullish trend from the local extreme to the current $y_{t_i}$.
            Move on to the next iteration.
        **end if**
        Mark $y_{t_i}$ as going through a bearish overshoot.
    **end if**
**end for**
```

Chapter 3

Literature Review

Target transformation techniques have been researched as part of the modelling process over the past sixty years. The same applies to the analyses of the underlying mechanisms in financial dynamics. This paper builds on both research directions and analyses a proposed target transformation technique for time series modelling based on the Directional Change (DC) Intrinsic Time framework. To the best of our knowledge, as this paper was written, we are unaware of any other attempt to bring these two methodologies together. In this chapter, we cover the most relevant works in both realms. We first review some target transformation developments in Section 3.1 and then look at some of the advancements in the Directional Change Intrinsic Time framework in Section 3.2.

3.1 Target Transformation

In most cases, time series analyses yielded from modelling are justified conditionally on the assumptions made by the models about the data. If the assumptions are not met by the original data, applying some transformation (often non-linear) to the data can help generate these conditions. This has led to various transformation techniques for these types of purposes. In this section, we go through some important assumptions made by the models and some of the most popular corresponding transformations found in the literature.

Homoscedasticity condition[1] is one of the most common assumptions for models

[1]The homoscedasticity (homogeneity of variances) condition requires the variances of different subsets of the sample to be the same. In the case of time series modelling, it is equivalent to

involving statistical inference. As a reference to the analysis of variance, M. S. Bartlett (1947) provided one of the earliest summaries of transformations on raw data addressing this. He covered parametric transformations used in stabilising the variance of modelling error, especially for Poisson and Binomial distributed variables where the variance is a known function of the mean. He discussed, both theoretically and empirically, some of the optimal scales and families of transformations to choose from given different prerequisites. His work showed that modelling tasks do benefit from suitable transformations.

Another common assumption made by the models is normality. Many statistical inference methods assume the variable of interest is normally distributed, including t-test, Analysis of Variance (ANOVA) and Linear Regression. Therefore, it is of paramount importance to transform the data in a way that such a condition is met, if possible. Box & Cox (1964) made a major contribution in this regard by proposing the well-known Box-Cox transformation. The Box-Cox transformation includes both power and logarithmic transformations. It aims at achieving normality of the observations and has been popular in developing modelling methodologies ever since (see Atkinson et al. 2021 and Osborne 2010). An example of applying Box-Cox and achieving better model performance was given in C. Bergmeier et al. (2016). Combined with the widespread exponential smoothing (ETS) forecasting method and the bootstrap aggregation (bagging) technique in machine learning, they proposed a bagging exponential smoothing method using STL decomposition and Box-Cox transformation. The proposed method was tested on the M3 competition dataset and achieved better results than all the original M3 participants (see Makridakis et al. 2000 for the M3 competition).

The method published by Box & Cox, however, is only valid for positive real values. Modifications of the Box-Cox transformation have thus been proposed to address this constraint. A significant extension was proposed by Bickel & Doksum (1981). They embedded a sign function to the power transformation such that the transformation function covers the whole real line. Nevertheless, this modification has its shortcomings: it was shown by Yeo & Johnson (2000) that Bickel & Doksum's modified version of the transformation handles skewed distribution poorly. Yeo & Johnson pointed out the reason being the signed power transformation was designed primarily for handling kurtosis, thus losing its edge concerning skewness. Following up, Yeo & Johnson proposed a new version of the power transformation in the same

requiring a constant variance throughout time.

publication (2000). Their transformation is a generalised version of the Box-Cox transformation. It approximates normality while being well-defined on the real line and inducing appropriate symmetricity.

In the previous paragraphs, we have described transformation techniques that can be used to satisfy mathematical and statistical conditions assumed by the models. Meeting these assumptions improves the robustness of the conclusions drawn by the modelling results. On a higher level, one can say that the transformations help the models to get better at 'learning' the problem such that they generate more robust outputs. In the upcoming paragraphs, we take on this perspective of treating the transformation techniques as helpers in terms of the learning process of the models and look at some transformation techniques very different from what was covered previously.

Decomposition methods constitute a significant category of techniques that can be used to transform the target in order to help the models learn from the target variable. These methods decompose the mixture of information contained within the observation into patterns, trends, cycles, or other dynamics that are easier to model, i.e., easier for models to learn from. A good example of such methods includes a large number of studies on Fourier-styled transformations in time series modelling[2] (see Kay & Marple 1981 and Bloomfield 2004). Typical Fourier methodologies build on transforming the observations sampled from the time domain into the frequency domain and decomposing them into more informative signals. Out of the great history and advancements of spectrum analysis, we selectively provide a brief overview of some relevant methodologies in the context of target transformation.

Building on Fourier methods, a novel type of transformation operates in the time-frequency domain, i.e., these methods generate time-frequency representations of the observations. The empirical mode decomposition (EMD) proposed by Huang et al. (1998) is one of such key methodologies. EMD decomposes the original time series into 'intrinsic mode functions' (IMF). The IMFs carry information on the underlying structures contained in the observations and can then be used for modelling tasks. The family of wavelet transform methods constitutes another class of methods that operates in the time-frequency domain (see Daubechies 1992 and Percival & Walden 2000). Shensa et al. 1992 first proposed and provided a framework for the discrete wavelet transform (DWT), which belongs to the wavelet transform family. DWT filters the original time series in several folds and yields a denoised version of the

[2]Such studies are also called spectrum analysis.

observation. The information carried by the transformed time series is then more clear and possibly easier to learn.

Another important family of decomposition methods consists of statistical-related methods. These decomposition methods generally consider the time series as a mixture of three components: seasonal, trend and remainder. They are often used to filter different information contained within the time series (see Wang, Smith, & Hyndman 2006). For a real-world example, monthly unemployment data are usually presented after removing the seasonality. The resulting time series is hence more indicative of the variation of the general economy instead of seasonal disturbance (see Chapter 3.2 in Hyndman & Athanasopoulos 2021). The STL decomposition proposed by Cleveland et al. (1990) has been a robust method for decomposition into seasonal and trend signals. The abbreviation stands for Seasonal and Trend decomposition using Loess. STL considers a time series as a sum (additive) or product (multiplicative) of the seasonal, trend and remainder components. STL is flexible and applicable to many use cases as it can handle any type of seasonality. Its flexibility also resides in its allowance for the user to have control over the time-varying seasonal component and smoothness of the trend cycle.

To provide some methodologies extending on STL decomposition, the X-11 method and the Seasonal Extraction in ARIMA Time Series (SEATS) procedure are time series models that rely heavily on seasonal and trend decompositions. They have had many variants and are favoured by official statistical agencies around the world (see Dagum & Bianconcini 2016)[3]. One of the state-of-the-art variants of this family is the X-13ARIMA-SEATS method produced, distributed and maintained by the US Census Bureau (see US Census Bureau 2012 and Monsell & Blakely 2013). It inherits powerful features from X-11, SEATS, and ARIMA methodologies while specialising in seasonal adjustment in extensive time series modelling. The model is conveniently accessible online[4]. The rationale of using such techniques as target transformation in time series modelling is to provide the models with a more informative, e.g., less noisy, dataset to learn from. We hope this extra procedure helps with the ultimate goal of producing a better trained (learned) model.

[3]X-11 was initially developed by the US Census Bureau, and SEATS was created by the Bank of Spain.

[4]A webpage demonstration of the model is accessible on `http://www.seasonal.website/`; the open-source implementation of the model can be found in the `seasonal` package in R, and a distributed version can be found in the US Census Bureau website `https://www.census.gov/data/software/x13as.X-13ARIMA-SEATS.html`.

3.2 Directional Change Intrinsic Time Framework

The technical core of the Directional Change (DC) intrinsic time framework is quite simple; it is an algorithm that assigns state information to a given time series. By analysing the properties of the resulting time series along with the state information, it has been found that despite the simplicity of this algorithm, it provides powerful perspectives for looking at market dynamics in the time domain. In this section, we first look at critical works that contribute to the advancements of the DC Intrinsic Time framework. Then we cover some applications that further developed the framework's value by trying to harness its potential.

Like the developments of many frameworks, the DC Intrinsic Time framework started out being simple and has developed over time. Guillaume et al. (1997) first published the Directional Change algorithm. The algorithm was presented and used to generate a set of measurements (statistics, variables), from which the authors presented a set of stylised facts found empirically in the spot intra-daily foreign exchange (FX) markets. These stylised facts shed new light on our understanding of market dynamics, especially concerning micro-structure topics, including time-heterogeneity, price formation, market efficiency, liquidity, and both the modelling and the learning process of the market. A little more than a decade later, Tsang formalised the definition of a Directional Change in Tsang (2010), and Glattfelder et al. (2011) discovered a set of twelve scaling laws derived from the output of DC algorithm. The discovery of the laws added a theoretical foundation to the DC algorithm as it was shown that the state information attached to the time series carries not only qualitative information (stylised facts) but also interesting quantitative properties. As the DC algorithm's ability to extract information has been studied, it has given rise to the methodology becoming a framework (see Tsang's introduction of a set of profiles (indicators) derived under the DC framework in Tsang (2015) and (2017)).

A well-known fact about analysing financial time series is that the source of many of the challenges can be traced back to the use of physical time (see Dacarogna et al. (2001)). Aloud et al. (2012) discussed the potential of studying financial time series using the DC framework (referred to as the DC approach in their paper) resides in its underlying 'intrinsic time' paradigm. They pointed out that mapping financial time series from the physical time to event-based intrinsic time is the key to how the approach filters out irrelevant information and disturbance observed in the dataset and generates valuable market insights of our interests. Inspired by the studies of complex systems, Petrov et al. (2018) took a different route of demonstrating this point with the use of agent-based modelling. They created a market with trading agents that operate in event-based intrinsic time and found that the price movements generated under such conditions experience statistical properties we observed in real-world physical time FX markets. Such reproduction of real-world stylised facts is another indication of the intrinsic time mechanism being one of the contributing factors to the market dynamics. Recently, Glattfelder & Golub (2022) derived an analytical relationship between physical and intrinsic time based on the scaling laws. In particular, the expression they derived decomposes the movements of the physical-time time series into volatility and liquidity components expressed in intrinsic time. That allows us to explicitly characterise the dynamics observed in physical time using its intrinsic-time representation.

As DC intrinsic time framework becomes theoretically sound, applications building on the framework have been devised. Golub et al. (2016) introduced the Intrinsic Network - an event-based framework based on directional changes. Combining the Intrinsic Network and information theory, they devised a liquidity measure that was shown to be able to predict market stress in terms of liquidity shocks. In Golub et al. (2018), the liquidity measure was integrated with other implementations derived from the DC framework and an algorithmic trading strategy called The Alpha Engine was introduced. The Alpha Engine has several interesting features. First, the bare-bones version of the model (without tweaking) has been shown to be robust, profitable, and can be implemented in real-time. Second, Alpha Engine provides liquidity in the market, i.e., it opens long positions when other market players intend to short and vice versa. The Alpha Engine thus contributes to the healthiness of the market as a participant. Third, Alpha Engine 'beats' random walk processes - it is shown to be profitable even on price dynamics generated by a random walk. Within the context of volatility and risk management in finance, Petrov et al. (2019a) proposed an instantaneous volatility measure under the DC Intrinsic Time framework. They found seasonality patterns and long memory (large degree of autoregressive feature) of volatility through empirical studies. Their work contributes not only to the development of practical tools but also to the understanding of the underlying stochastic drive of financial dynamics. Two further generalisations of the DC framework have been proposed. First, Petrov et al. (2019b) brought the framework into multidimensional space by extending the analytical expressions yielded from one-dimensional analyses to multidimensional space. Their methodology implies that previous works in one-dimensional space (analytical insights, empirical findings, and all the tools and implementations) can be extended to higher dimensions.

Another generalisation was developed with respect to the types of stochastic processes (Mayerhofer (2019)). These generalisations, as well as the advancements of the DC Intrinsic Time framework discussed previously, are indicative of the framework's promising potential worthy of further exploration.

In this chapter, we reviewed relevant works in two different realms: target transformations being used as an additional layer in modelling and the DC Intrinsic Time framework being a rising methodology. The literature surveyed outlines their history and demonstrates excellent potential in both research directions that justifies our curiosity in bringing them together. In the next chapter, we go into detail about the methodologies of combing the DC framework and target transformation in time series modelling.

Chapter 4

Methodology

This chapter covers the technical details methodologies. We present our proposed target transformation technique Directional Change Transformation (DCT) in Section 4.1. Then in Section 4.2, we address the experiment we design to evaluate the transformation.

4.1 DC Target Transformation

Our proposed target transformation is called *Directional Change (DC) Transformation*. The DC Transformation is a chain of operations to be embedded as the general target transformation procedure introduced in Section 2.2.1. Consider the general notation for a time series $\mathcal{Y}_{t_k} = \{y_{t_i}\}_{i=1,2,\ldots,k}$ and its timestamp set being t; for some mapping $X(\mathcal{Y}_{t_k}) = \mathcal{Y}_{t_k}^{(X)}$, we denote $\mathcal{Y}_{t_k}^{(X)}$'s elements as $y_{t_i}^{(X)}$ and its timestamp set $t^{(X)}$. Recall the log-difference transformation $\mathcal{T}^{(logr)}$ and DC algorithm $\mathcal{A}^{(DC)}(\cdot;\delta,\mathcal{R}(x;y))$ we addressed earlier. Then for an arbitrary model fitting to be conducted with time series $\mathcal{Y}_{t_k}$, DC Transformation goes as the following:

We start by performing the DC classification algorithm for a given δ and $\mathcal{R}$

$$\mathcal{Y}_{t_k}^{(DC)} = \mathcal{A}^{(DC)}(\mathcal{Y}_{t_k};\delta,\mathcal{R}(x;y)).$$

We then collect the values marked as local extrema or confirmation of directional change events. Let $\mathcal{Y}_{t_k}^{(E)}$ be a subset of $\mathcal{Y}_{t_k}^{(DC)}$ such that

$$\mathcal{Y}_{t_k}^{(E)} = \{(y_{t_i}, s) | s = \text{local extreme} \vee \text{DC confirmation}\}_{t_i \in t}.$$

As $\mathcal{Y}_{t_k}^{(E)} \subset \mathcal{Y}_{t_k}^{(DC)}$, we look at the y_{t_i} values in $\mathcal{Y}_{t_k}^{(E)}$ and perform interpolation for $t_i \in t - t^{(E)}$[1] to fill in the gaps between the event points in $\mathcal{Y}_{t_k}^{(E)}$. Note that the states s assigned by $\mathcal{A}^{(DC)}$ are preserved and left unchanged because we are only operating on the values y_{t_i}. Let $\mathcal{I}(\mathcal{Y}_{t_k}^{(E)}, \mathcal{Y}_{t_k}^{(DC)})$ be an interpolation operation such that we have a new set given by

$$\mathcal{Y}_{t_k}^{(EI)} = \mathcal{I}(\mathcal{Y}_{t_k}^{(E)}, \mathcal{Y}_{t_k}^{(DC)}) = \begin{cases} (y_{t_i}, s) & \text{if } t_i \in t - t^{(E)} \\ (y_{t_i}^{(I)}, s) & \text{otherwise} \end{cases}, \forall i \in \{1, 2, \cdots, k\}$$

where $y_{t_i}^{(I)}$ are the interpolated values generated by an interpolation $\mathcal{I}$. We note that after the interpolation, $\langle \mathcal{Y}_{t_k}^{(EI)} \rangle = \langle \mathcal{Y}_{t_k} \rangle$[2].

We visualise $\mathcal{Y}_{t_k}^{(EI)}$ and $\mathcal{Y}_{t_k}$ alongside with each other and with different δ and $\mathcal{I}$ settings to illustrate their effects. Let $\delta = (10\%, -10\%)$, $\mathcal{I}$ be a linear interpolation[3] and $\mathcal{R}$ be simple return as stated in Equation 2.1, then we can have Figure 4.1. The line in blue is $\mathcal{Y}_{t_k}$. The circles and diamonds mark the values in $\mathcal{Y}_{t_k}^{(E)}$. The line in green, along with the circles and diamonds, illustrates $\mathcal{Y}_{t_k}^{(EI)}$.

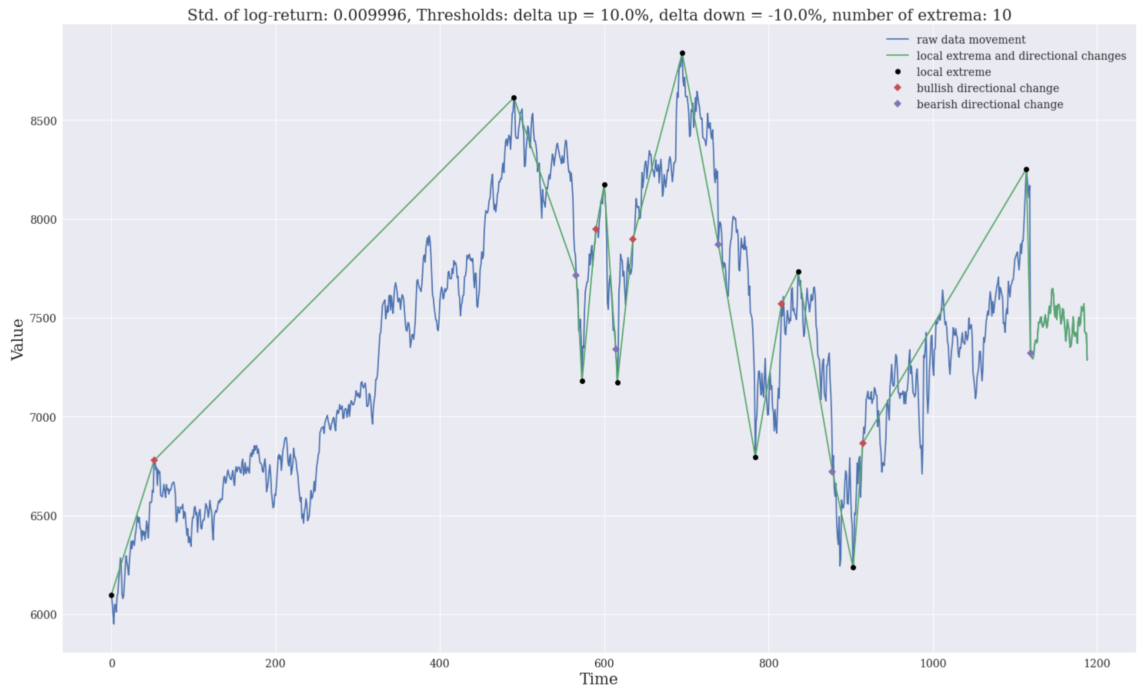

Figure 4.1: DC Transformed and original time series ($\delta = \pm 10\%$, linear)

[1]The set $t - t^{(E)}$ marks all the intervals between local extrema and DC events by timestamps.

[2]Recall that we let $\langle \cdot \rangle$ be a counting operator of the number of elements in a finite set.

[3]If extrapolation is needed in the tail of the series, we simply leave the values unchanged.

To observe the effect of different parameters, Figure 4.2 has a lower δ of $(5\%, -5\%)$. Decreasing the threshold values leads to the DC identification algorithm $\mathcal{A}^{(DC)}$ being more sensitive to events. We thus get a larger $\mathcal{Y}^{(E)}_{t_k}$ and narrower intervals to interpolate.

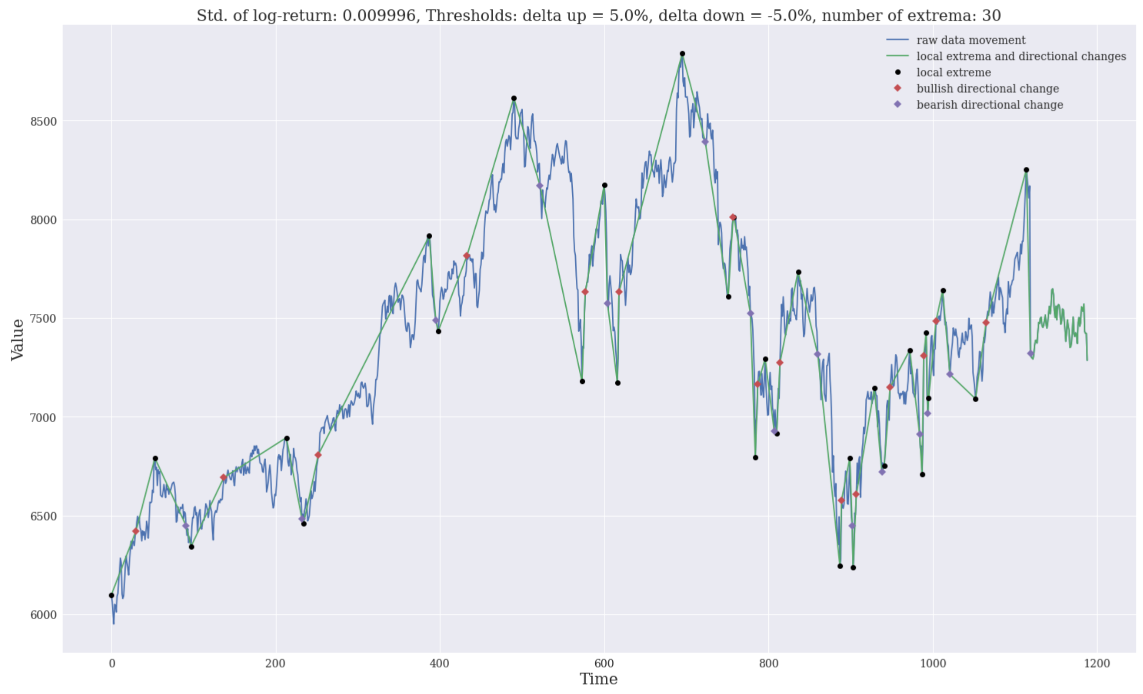

Figure 4.2: DC Transformed and original time series ($\delta = \pm 5\%$, linear)

As to different interpolation methods, with Akima interpolation (Akima (1970)), which is a variant of polynomial interpolation, we get what is shown in Figure 4.3.

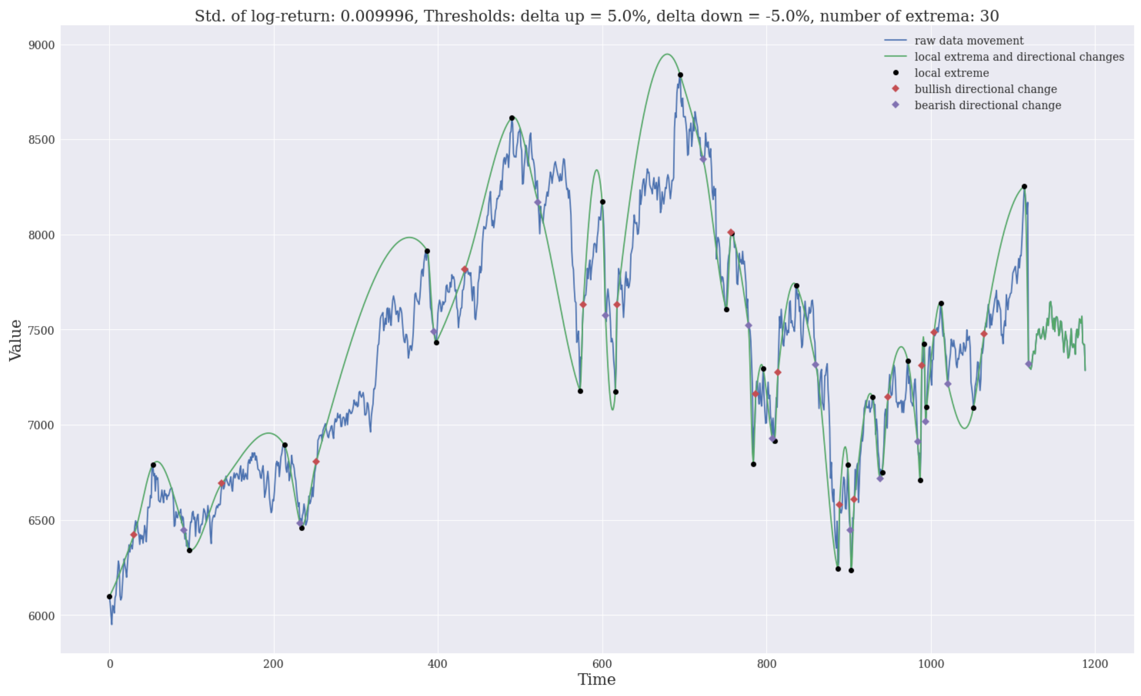

Figure 4.3: DC Transformed and original time series ($\delta = \pm 5\%$, akima)

After having $\mathcal{Y}_{t_k}^{(EI)}$, we then apply the log-difference target transformation, which was introduced in Section 2.2.2. We get:

$$\mathcal{Y}_{t_k}^{(logrEI)} = \mathcal{T}^{(logr)}(\mathcal{Y}_{t_k}^{(EI)}) = \begin{cases} (\emptyset, s) & \text{if } i = 1 \\ (\log(\frac{y_{t_i}^{(EI)}}{y_{t_{i-1}}^{(EI)}}), s) & \text{if } i = 2, 3, \cdots, k \end{cases}.$$

Figure 4.4 demonstrates an example of $\mathcal{Y}_{t_k}^{(logrEI)}$, which is the log-differenced $\mathcal{Y}_{t_k}^{(EI)}$ shown in Figure 4.3.

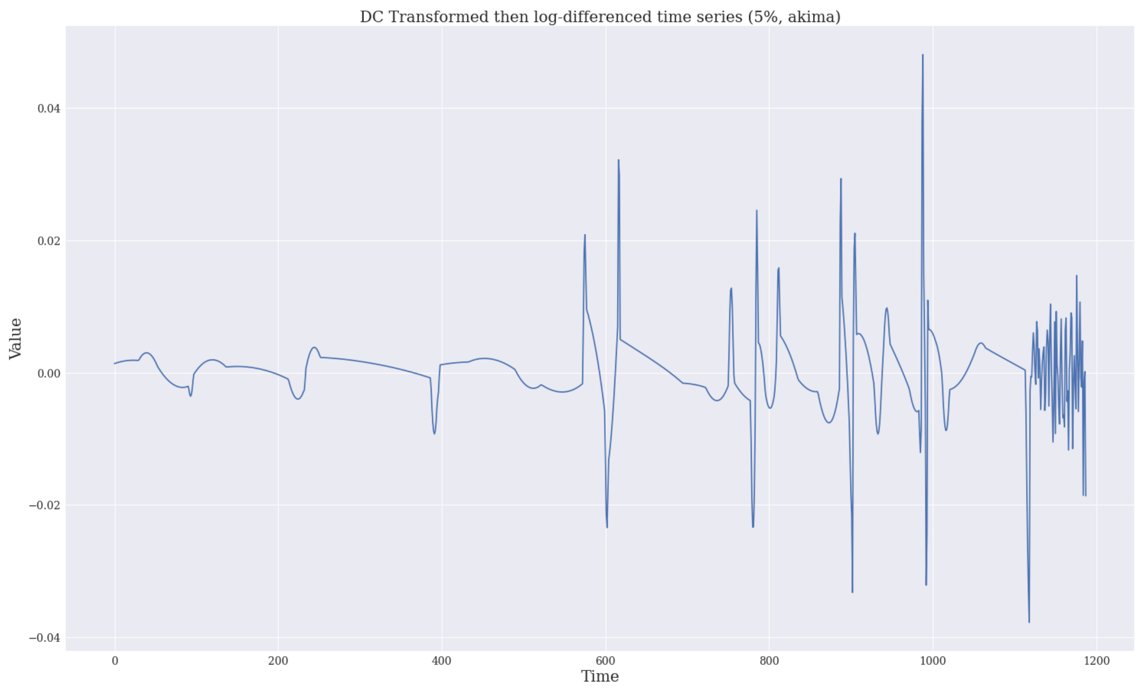

Figure 4.4: DC Transformed then log-differenced time series ($\delta = \pm 5\%$, akima)

We can then generate the training target and design matrix with the $y_{t_i}^{(logrEI)}$ values in $\mathcal{Y}_{t_k}^{(logrEI)}$

$$\mathbf{y}_{train}^{(logrEI)},\ \mathbf{X}_{train}^{(logrEI)} = \mathcal{S}(\mathcal{Y}_{t_k}^{(logrEI)}; \tau, h = 1, \lambda)$$

$$\mathbf{y}_{train}^{(logrEI)} = \begin{bmatrix} y_{t_\lambda}^{(logrEI)} \\ y_{t_{\lambda+1}}^{(logrEI)} \\ . \\ . \\ y_{t_{k-1}}^{(logrEI)} \\ y_{t_k}^{(logrEI)} \end{bmatrix}, \quad \mathbf{X}_{train}^{(logrEI)} = \begin{bmatrix} y_{t_{\lambda-\tau}}^{(logrEI)} & y_{t_{\lambda-1-\tau}}^{(logrEI)} & \cdots & y_{t_1}^{(logrEI)} \\ y_{t_{\lambda+1-\tau}}^{(logrEI)} & y_{t_{\lambda-\tau}}^{(logrEI)} & \cdots & y_{t_2}^{(logrEI)} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ y_{t_{k-1-\tau}}^{(logrEI)} & y_{t_{k-2-\tau}}^{(logrEI)} & \cdots & y_{t_{k-\lambda-\tau}}^{(logrEI)} \\ y_{t_{k-\tau}}^{(logrEI)} & y_{t_{k-1-\tau}}^{(logrEI)} & \cdots & y_{t_{k-\lambda+1-\tau}}^{(logrEI)} \end{bmatrix}.$$

Before fitting the model, we also introduce a feature generation method utilising the states produced by $\mathcal{A}^{(DC)}$. For each instance $y_{t_i}^{(logrEI)}$ in the training target $\mathbf{y}_{train}^{(logrEI)}$, we use the state s of $y_{t_i-\tau}^{(logrEI)}$ from $\mathcal{Y}_{t_k}^{(logrEI)}$. Since s is a categorical variable with seven possible outcomes as listed in Section 2.3.1, it needs some further preprocessing to be suitable for predicting. We thus use *one-hot encoding* for the categorical variable s. For $k - \lambda - 1$ predictions, we get a $k - \lambda - 1$ by seven matrix, where each row is the one-hot encoded vector of the corresponding s. We concatenate the one-hot encode matrix on the right side of $\mathbf{X}_{train}^{(logrEI)}$ and have a new design matrix $\mathbf{X}^{(SlogrEI)}$ with the one-hot encoded states.

We then fit the model with its loss function $\mathcal{E}$ to find the optimal parameter θ for the given hyperparameter ϕ_0

$$\theta(\phi_0)_0 = \arg_{\theta(\phi_0)} \min \mathcal{E}(\widehat{\mathbf{y}}^{(SlogrEI)}_{train}, \mathbf{y}^{(logrEI)}_{train}),\ \widehat{\mathbf{y}}^{(SlogrEI)}_{train} = f(\theta(\phi_0); \mathbf{X}^{(SlogrEI)}_{train}).$$

After having the model, we make the input matrix also with the one-hot encoded state information and denote as[4]

$$\mathbf{X}^{(SlogrEI)}_{val}$$

and make a prediction using the trained model

$$\widehat{\mathbf{y}}^{(SlogrEI)}_{val} = f(\mathbf{X}^{(SlogrEI)}_{val}; \theta(\phi_0)_0).$$

Obtain the validation target that we need for validation

$$\mathbf{y}_{val} = y_{t_k+\tau}.$$

Since the operations related to the DC framework do not contribute to the transformed prediction not being comparable to the original level, we only have to back-transform the change of scale introduced by log-differencing. Let the back transformation of log-differencing be denoted $\mathcal{T}^{(b-logr)}$ as described in Equation 2.4 and we have

$$\widehat{\mathbf{y}}^{(b-SlogrEI)}_{val} = \mathcal{T}^{(b-logr)}(\widehat{\mathbf{y}}^{(SlogrEI)}_{val}; \mathcal{Y}_{t_k}).$$

Finally, we compute the validation score using the back-transformed prediction and the validation target by

$$\mathcal{V}(\widehat{\mathbf{y}}^{(b-SlogrEI)}_{val}, \mathbf{y}_{val}).$$

And that concludes the whole implementation of DC Transformation in a single model fitting. Then, as the stated standard modelling procedure, such a fitting process is repeated in the validation operation to compute the validation score for a single model setting (hyperparameter). We perform validation for all the hyperparameters and choose the one with the best validation score.

Let $\mathcal{T}^{(DC)}$ denotes the DC Transformation. $\mathcal{T}^{(DC)}$ is parameterised by the pair of threshold values θ, the movement measure $\mathcal{R}$ and the interpolation method $\mathcal{I}$. As to back transformation, applying $\mathcal{T}^{(DC)}$ needs the log-differencing back transformation.

[4]$\mathbf{X}^{(SlogrEI)}_{val}$ is of shape 1 by the number of lags plus seven.

In practice, all $\theta, \mathcal{R}$ and $\mathcal{I}$ can be explored during model selection. We can see that there are two components, each having different effects on the output: the DC algorithm $\mathcal{A}^{(DC)}$ captures the extreme events while the interpolation component $\mathcal{I}$ creates a smooth process in between them. The threshold parameter δ controls the sensitivity of $\mathcal{A}^{(DC)}$ to events and thus affecting the number of intervals needed for interpolating and their width.

4.2 Experiment

This section discusses the experiment to investigate whether our proposed DC Transformation can positively affect univariate time series forecasting. The effectiveness of a target transformation technique depends on both the complementing model (the choice of f) and the dataset. To obtain robust and objective results, we experiment with different types of models that can be used for univariate time series forecasting and a large number of time series. In particular, we train five models with and without DC Transformation over many time series and compare model performance.

4.2.1 Dataset

Our experiment's dataset is the one used in the M4 Competition (see Makridakis et al. 2020). It consists of 100,000 univariate time series with different frequencies and domains (see Table 4.1). Although we intend to scale the experiment to have robust results, we cannot experiment on all the time series given our limited time. Therefore, we picked a subset of time series to experiment with. We intend to experiment on higher frequency and finance/economic related time series. We tested on a total of 458 time series from the M4 dataset: 66 of them are weekly financial time series, 252 of them are daily time series from three different domains (Microeconomics, Macroeconomics, and Finance), and 140 of them are hourly time series from unknown domains (see Table 4.2)[5]. The average length of weekly time series is 1260.7, that of the daily ones is 1179.99, and that of the hourly ones is 1008.

[5]The time series in M4 dataset are indexed from 1 to 100000. We choose by their length: the daily and hourly time series chosen are the first in their domain with a length between 1000 and 1500; and the weekly financial time series chosen are the first to have a length between 600 and 1500.

	Micro	Industry	Macro	Finance	Demographic	Other	Total
Yearly	6538	3716	3903	6519	1088	1236	23000
Quarterly	6020	4637	5315	5305	1858	865	24000
Monthly	10975	10017	10016	10987	5728	277	48000
Weekly	112	6	41	164	24	12	359
Daily	1476	422	127	1559	10	633	4227
Hourly	0	0	0	0	0	414	414
Total	25121	18798	19402	24534	8708	3437	100000

Table 4.1: Number of series in the M4 dataset per frequency and domain
This table is a direct reference to the one in Makridakis et al. (2020).

	Micro	Macro	Finance	Other	Total
Weekly	-	-	66	-	66
Daily	98	54	100	-	252
Hourly	-	-	-	140	140
Total	98	60	166	140	458

Table 4.2: number of series used per frequency and domain

4.2.2 DC Transformation Settings

Recall that DC Transformation is characterised by some parameters. For the movement measure $\mathcal{R}(x;y)$, we used the net return given by

$$\mathcal{R}(x;y) = \frac{x-y}{y}.$$

As to the interpolation method $\mathcal{I}(S_1, S_2)$ for two sets $S_1 \subset S_2$, we used a simple linear interpolation - the values in S_1 are connected with a straight line such that $\langle S_1 \rangle = \langle S_2 \rangle$. If extrapolation is needed, we simply keep the values unchanged. We did try other interpolation methods during exploratory trials. We will mention this in Chapter 5.

Regarding the most important parameter $\delta = (\delta_{down}, \delta_{up})$, we include this parameter into the hyperparameter set ϕ that has to be tuned. Since δ is correlated to the diffusion of the time series if considered as a stochastic process (see Appendix A in Petrov et al. (2018)), the hyperparameter space is decided according to our simple estimation of the diffusion of the training time series if considered as a stochastic process. Let $\mathcal{Y}_{t_k}$ be an arbitrary training set, $\mathcal{T}^{(logr)}$ be the log-differencing operator described as Equation 2.4 in Section 2.2.2 and Var$(\cdot)$ be the variance for a one-dimensional series excluding the empty values if there is any, then our estimation of the diffusion σ is given by

$$\begin{aligned}\mathcal{Y}_{t_k}^{(logr)} &= \mathcal{T}^{(logr)}(\mathcal{Y}_{t_k}) \\ \sigma &= \sqrt{\mathrm{Var}(\mathcal{Y}_{t_k}^{(logr)})}.\end{aligned}$$

The hyperparameter space of δ that we denoted as $H(\delta)$ is then based on the direct proportion of σ and is given by[6]

$$\begin{gathered}A = \{0.01\sigma, 0.21\sigma, 0.41\sigma, 0.61\sigma, 0.81\sigma, 1.01\sigma, 1.21\sigma, 1.41\sigma, 1.61\sigma, 1.81\sigma, 2.01\sigma\} \\ H(\delta) = A \times A.\end{gathered}$$

Observe that $H(\delta)$ is a set with $11 \times 11 = 121$ two-dimensional tuples. The first element in a tuple is the number for δ_{down} and the second for δ_{up}. This is a grid search approach.

4.2.3 Models

This section covers the models we used and the hyperparameters we tuned in our experiment, for which we adopt an easy grid search approach. The actual hyperparameter space we searched per model is provided in Appendix A. For the discussion in the rest of this section, consider an arbitrary training set $\mathcal{Y}_{t_k} = \{y_{t_i}\}_{i=1,2,\ldots,k}$ and its corresponding $\mathbf{y}$ and $\mathbf{X}$ derived from $\mathcal{S}$ be target and design matrix with n rows (instances), ϕ as the hyperparameter set, and θ the parameter set for the model.

Elastic Net Regression Model

We use Elastic Net (EN) as our representative for linear regression methodologies. EN can be written as

$$f^{(EN)}(\mathbf{X}; \theta = (\mathbf{w}, \beta)) = \mathbf{X}\mathbf{w} + \beta = \widehat{\mathbf{y}}.$$

The structure and determination of θ is controlled by hyperparameters $\phi = (l, \alpha, \rho)$ and the optimisation

$$\theta_{best} = \arg_\theta \min \frac{1}{2n} L_2(\mathbf{X}\mathbf{w} + \beta - \mathbf{y}) + \alpha\rho L_1(\mathbf{w}) + \frac{\alpha(1-\rho)}{2} L_2(\mathbf{w})$$

[6]The operator '$\times$' between two finite sets is a cartesian product that gives all possible combinations of the elements in the two finite sets. For example, $\{a, b\} \times \{1, 2\} = \{(a, 1), (a, 2), (b, 1), (b, 2)\}$.

where

- n is the number of training instances, i.e., the length of $\mathbf{y}$,
- l is the number of lags. As mentioned in Section 2.1.2, the bigger l is, the more past information the model is allowed to use for a single prediction. l also affects the length of the weight vector $\mathbf{w}$.
- $\alpha > 0, \ \alpha \in \mathbb{R}$ is the regularisation constant determining the scale of the penalty.
- L_1 and L_2 are L1 and L2 norm respectively.
- $\rho \in [0, 1]$ is the weight assigned to L_1 penalty.

Observe the objective function in the optimisation operation is the loss function $\mathcal{E}$ associated with EN.

Multilayer Perceptron Regressor

To have a non-linear regression model, we chose the Multilayer Perceptron (MLP) regressor. MLP is controlled by $\phi = (l, strucs, maxiter)$ where

- l is the number of lags,
- *strucs* is a tuple. Element number i in *stucs* determines the number of hidden neurons in hidden layer i. For example, $strucs = (5, 10)$ means the MLP has two hidden layers, with the first having five hidden neurons and the second having ten.
- *maxiter* is the maximum iterations the solver is allowed to go through.

Every neuron in the MLP model is a linear combination of an input vector. $neuron : \mathbb{R}^m \longmapsto \mathbb{R}$. Depending on the dimension of the input, i.e., m, a neuron is parameterised by a m dimension weight vector. These neurons form a neural network, which is a non-linear function of the input vector. The weights are trained using a variant of stochastic gradient descent solver (see Kingma et al. 2014).

Linear Support Vector Regressor

Linear Support Vector Regressor (LSVR) is a member of the *Kernel Methods* that uses a linear kernel. We chose LSVR for its faster implementation than typical

SVR with other kernels. LSVR is parameterised by $\theta = (\mathbf{w}, \mathbf{b})$ much like a linear regression model

$$f^{(LSVR)}(\mathbf{X}; \theta = (\mathbf{w}, \mathbf{b})) = \mathbf{X}\mathbf{w} + \mathbf{b} = \widehat{\mathbf{y}}.$$

where θ is decided by solving the optimisation problem

$$\theta_{best} = \arg_{\theta=(\mathbf{w},\mathbf{b})} \min(\frac{1}{2}L_2(\mathbf{w}) + C\sum_{i=1}^{n} \max(0, |\mathbf{y}_i - (\mathbf{w}^T I(\mathbf{X}_i) + \mathbf{b})| - \epsilon))$$

where

- L_2 is the L2 norm.
- I is the identity function.

The hyperparameter set characterising the decision of θ we tune is $\phi = (l, tol, C, \epsilon)$ where

- l is the number of lags,
- tol is the tolerance for the stopping condition during the,
- C is the regularisation constant similar to α for EN. However, by the implementation, C is inversely proportional to the strength of the penalty.
- ϵ characterises the *Vapnik's ϵ-insensitive loss function* that makes a 'tube' with radius ϵ around the target $\mathbf{y}$. The decision of this value should depend on the scale of $\mathbf{y}$. We have it defaulted to 0 as suggested by scikit learn.

Random Forest Regressor

Our representative for tree-based models is Random Forest Regressor (RF) because it trains faster than other tree-based methods, such as Gradient Boosting Machine or Light Gradient Boosting Machine. RF is an ensemble method that applies *Bootstrap Aggregation (Bagging)* method on both the training instances and the features. In other words, the trees in the ensemble are trained on different subsets of the training set and feature space. Bagging on features is applied randomly (the subsets of feature space are sampled randomly). Such a method is called the *random subspace method.* The hyperparameter set we tuned is $\phi = (l, minsamplesplit)$ where

- l is the number of lags.

- *minsamplesplit* sets the minimum number of samples required to split an existing node. The value is positively correlated to the strength of pruning as a regularisation measure.

The parameter θ is a combination of decision criteria of the decision trees in the ensemble.

Exponential Smoothing Forecasting Model

Exponential Smoothing (ETS) is not a regression model, as in, it does not estimate its parameters with our target $\mathbf{y}$ and design matrix $\mathbf{X}$ environment. ETS trains directly on the univariate time series $\mathcal{Y}_{t_k} = \{y_{t_i}\}_{i=1,2,...,k}$. ETS has many variants. The one we used here is based on the implementation of R Version of ETS (see Hyndman et al. 2008b and Hyndman et al. 2008a). This version of ETS implicitly tunes the hyperparameter $\phi = (trend, seasonal, damped)$ where

- $seasonal \in \{\text{additive}, \text{multiplicative}\}$ is a binary variable setting whether the seasonal component in the model is additive or multiplicative. An additive seasonal component fits better when there is not much seasonal variation, while a multiplicative seasonal component is made for the cases where seasonal variation is proportional to the level of the time series.

- $trend \in \{\text{additive}, \text{multiplicative}\}$ is a binary variable setting whether the trend component in the model is additive or multiplicative.

- $damped \in \{\text{True}, \text{False}\}$ is a binary variable setting whether the damped trend methods is applied. The damped trend method dampens the trend component to a constant into the future to prevent the model from over-forecasting for future values.

ETS essentially uses a weighted average of the past observations as its prediction. In the sense that a weighted average is a form of linear combination, ETS is much like EN according to how we train it. θ is a vector of weights assigned to the past observations ETS uses for its prediction.

4.2.4 Agents

For each model we mentioned in the previous section, we train the model with and without applying DC Transformation on the set of time series we chose from the M4 dataset. As a result, we end up having ten agents

$$\{\text{EN}, \text{MLP}, \text{LSVR}, \text{RF}, \text{ETS}\} \times \{\text{with DC Transform}, \text{without DC Transform}\}$$

each being trained on many time series.

4.2.5 Performance Metrics

We consult the experiment described in Ahmed et al. (2010) for our performance metrics. Our main performance measure is the Symmetric Mean Absolute Percentage Error (SMAPE)[7]. This is used as our $\mathcal{V}$ measure denoted in the modelling procedure. For a single prediction $\widehat{\mathbf{y}}_i \in \widehat{\mathbf{y}}$ and the target $\mathbf{y}_i \in \mathbf{y}$, the Symmetric Absolute Percentage Error (SAPE) is given by

$$SAPE(\widehat{\mathbf{y}}_i, \mathbf{y}_i) = \frac{2|\widehat{\mathbf{y}}_i - \mathbf{y}_i|}{|\widehat{\mathbf{y}}_i| + |\mathbf{y}_i|}$$

where the operator $|\cdot|$ is the norm of a vector (can be L1, L2 or others). However, seeing that our forecasting horizon is fixed to one for all forecasting tasks, we have $\widehat{\mathbf{y}}_i, \mathbf{y}_i \in \mathbb{R}$. So we can see $|\cdot|$ as the absolute operator of a real number. Then for a set of m validation or testing points, SMAPE is given by

$$SMAPE(\widehat{\mathbf{y}}, \mathbf{y}) = \frac{1}{m} \sum_{i=1}^{m} \frac{2|\widehat{\mathbf{y}}_i - \mathbf{y}_i|}{|\widehat{\mathbf{y}}_i| + |\mathbf{y}_i|}$$

Observe that SMAPE is scale-free because it is a relative error measure. We can combine the SMAPE yielded from different time series with different scales. This is one of the main reasons we chose SMAPE as our primary error measure.

We also calculate a ranking measure as a summary statistics. Let $Q = 10$ be the total number of agents. For a time series, we have Q agents trained and their corresponding SMAPE scores. We rank these Q agents based on their SMAPE - the agent with the lowest SMAPE is ranked number one, the largest the last. For P time series that all Q agents have been trained on, an agent has P ranks. Let $R_{q,p}$ denote the rank of model q on time series p. We have a distribution of ranks for an agent q and can calculate the mean given by

$$R_q = \frac{1}{P} \sum_{p=1}^{P} R_{q,p}.$$

[7]This is also the main error measure for M3 Competition (see Makridak et al. 2000)

Moreover, treating R_q as a random variable, we can come up with a confidence interval for the ranking of an agent q. Let $c_{\alpha,Q}$ be the upper α percentile of the range for Q independent standard Gaussian variables. Then the α percent confidence interval of the ranking for agent q is given by

$$R_q \pm 0.5 c_{\alpha,Q} \sqrt{\frac{Q(Q+1)}{12P}}.$$

See Koning et al. (2005) for further details of such statistics.

Another summary performance measure we calculated is called the 'fraction-best' measure (see Section Five in Ahmed et al. (2010)). The fraction-best is also a rank-based measure. It measures how often an agent ranks top-performing out of all P time series tested. That is, if agent q is ranked number one for n times out of P time series, the fraction-best for agent q is given by

$$FB_q = \frac{n}{P}.$$

There could be occasions where an agent is conditionally performant, i.e., an agent is particularly good only under limited circumstances. In such cases, the agent can have a mediocre average SMAPE score but an impressive fraction-best score. Calculating the fraction-best score helps identify such agents if they exist.

4.2.6 Modelling Configurations

In this section, we present some other configurations for our experiment. All agents use log-differencing as their basic target transformation method during the model fitting. Log-differencing helps make the time series stationary. The validation size v and test size κ are fixed to ten percent of the length of the training set size for all time series. We treat the time series as time-homogeneous, i.e., the values in a time series are indexed by the nature number set sequentially. The forecast horizon is fixed to one, and the gap is fixed to one step for all forecasting objectives. As to the retrain window, we have a retrain window of ten, i.e., the model updates its parameters every ten steps. We thought about the drawback in model training of introducing a retrain window. However, since the goal of our experiment is to compare how models perform with and without the DC Transformation, the retrain window should not affect our conclusions: all agents compete under the same conditions.

Chapter 5

Results and Evaluation

In this chapter, we present our experimental results and their corresponding evaluation. Our experiment was built to evaluate whether DC Transformation helps improve model performance in univariate time series forecasting. Our experiment covers weekly, daily, and hourly frequencies and the results are investigated separately based on the performance metrics we addressed in Section 4.2.5. For a model 'M', agents using model 'M' without DC Transformation are named 'M_raw' and agents using model 'M' with DC Transformation are named 'M_tran'.

5.1 Analysing Weekly Time Series

We start by looking at the weekly financial time series and the summary statistics of the agents. Figure 5.1 shows that the agents have about the same level of overall performance across the time series tested. Agents with and without DC Transformation do not generate different ranges of errors in general.

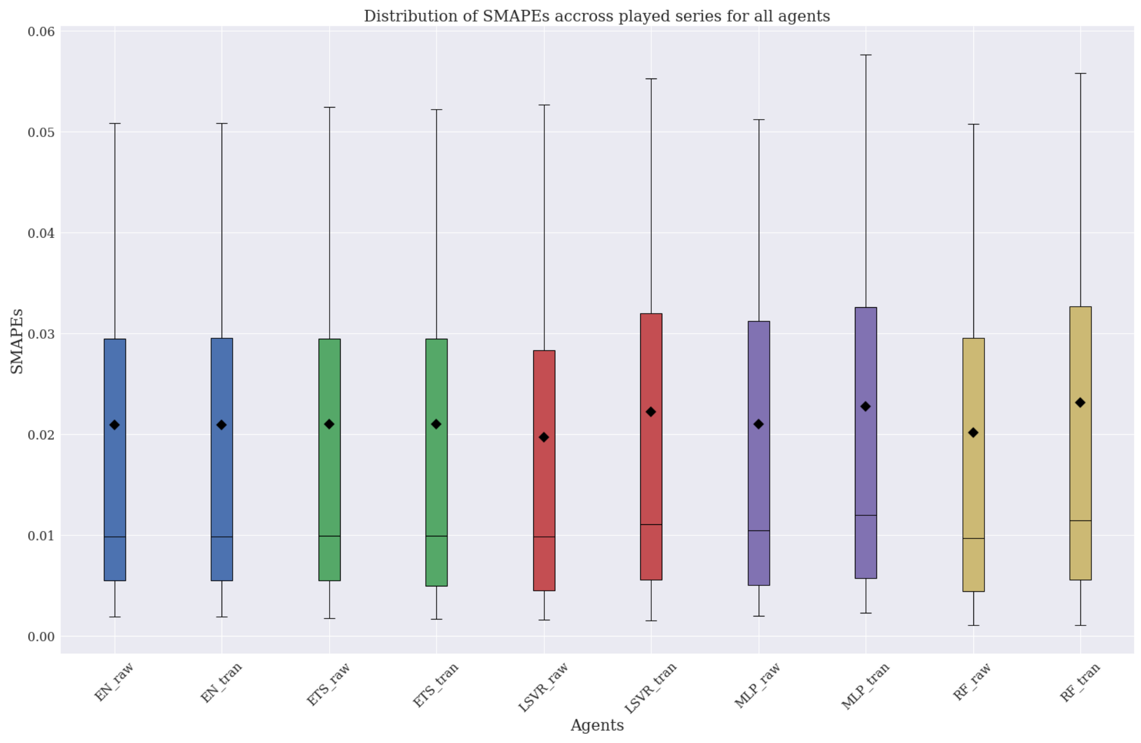

Figure 5.1: Distribution of SMAPE across 66 weekly time series per agent
The dataset used for this graph consists of 66 weekly financial time series. The boxplots are distributions of SMAPEs (described in Equation 4.2.5) across 66 time series per agent. The diamonds are the mean of the corresponding distribution. Agents with the same model are coloured the same.

Figure 5.2 shows each agent's 90% rank interval as presented in Equation 4.2.5. Such statistics has the nature of distinguishing the agents even more because the ranks do not overlap regardless of how close the agents' actual errors are. We can see that linear models like Elastic Net and Exponential Smoothing are on the same level of performance with and without DCT compared to the other models.

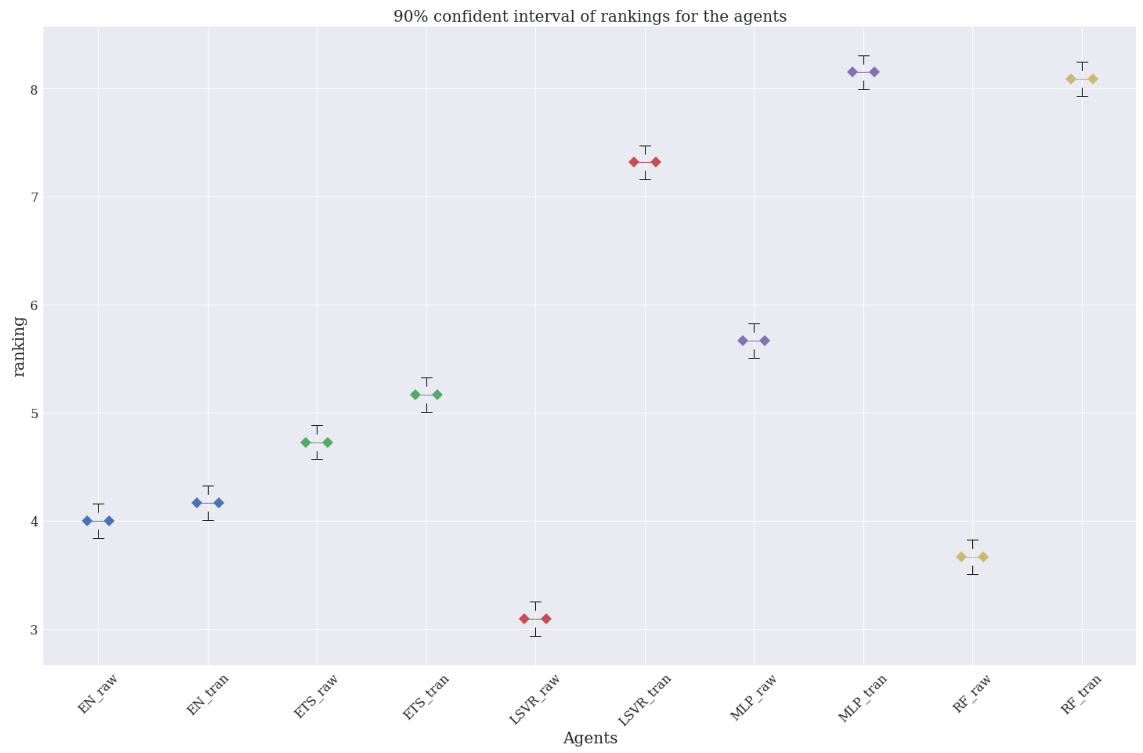

Figure 5.2: 90% rank interval across 66 weekly time series per agent
The dataset used for this graph consists of 66 weekly financial time series. The intervals are rank intervals under 90% confidence level (Equation 4.2.5) over 66 time series per agent. Agents with the same model are coloured the same.

Table 5.1 contains the summary statistics across all agents. The fraction-best score confirms what has been shown in Figure 5.2 that LSVR_raw and RF_raw are the most performant agents, and EN and ETS agents generate similar errors with and without DCT.

	Avg. SMAPE	Std. SMAPE	Avg. rank	Std. rank	90% rank interval	frac best
EN_raw	0.02091	0.024271	4.0	1.954	(3.753, 4.247)	0.121
EN_tran	0.020935	0.024349	4.167	1.919	(3.92, 4.414)	0.106
ETS_raw	0.020979	0.024556	4.727	2.042	(4.48, 4.974)	0.061
ETS_tran	0.021029	0.02474	5.167	2.086	(4.92, 5.414)	0.03
LSVR_raw	0.019724	0.022382	3.091	2.227	(2.844, 3.338)	0.348
LSVR_tran	0.022205	0.024924	7.318	2.14	(7.071, 7.565)	0.015
MLP_raw	0.021029	0.023338	5.667	2.749	(5.42, 5.914)	0.03
MLP_tran	0.022734	0.024506	8.152	2.251	(7.905, 8.399)	0.0
RF_raw	0.020129	0.022872	3.667	2.809	(3.42, 3.914)	0.348
RF_tran	0.02314	0.026118	8.091	2.598	(7.844, 8.338)	0.045

Table 5.1: Summary statistics of all agents on weekly financial time series
The dataset consists of 66 time series. The average length of the time series is 1260.70. The first and second columns are the mean and standard deviation of the SMAPEs for each agent. The third and fourth columns are the mean and standard deviation of the ranks (Equation 4.2.5) for each agent. The fifth column has the 90% rank intervals. The final column has the fraction best (Equation 4.2.5).

As part of the modelling procedure, target transformation techniques are policies the modeller might consider applying depending on the situation, e.g., the dataset, modelling objective, model constraints and assumptions. It would be too ambitious to harbour the intention of seeing a target transformation technique improve performance for all combinations of model and dataset. The overall statistics we showed is a sanity check about whether using DCT risks worsening model performance in general, and the statistics showed it does not. We then examine how often DCT works in favour of model performance and when it happens. For a model M and 66 time series, we can compute the 66 SMAPE reductions caused by DCT. Let $SMAPE_{\text{M\_raw},i}$ and $SMAPE_{\text{M\_tran},i}$ be the SMAPE for an agent using model M without DCT on time series number $i \in \{1, 2, \cdots, 66\}$ and that with DCT on the same time series i. Then the SMAPE reduction caused by DCT for model M on time series i is given by

$$\frac{SMAPE_{\text{M\_raw},i} - SMAPE_{\text{M\_tran},i}}{SMAPE_{\text{M\_raw},i}} \times 100\%.$$

The first and second columns in Table 5.2 contain the mean and standard deviation of SMAPE reductions over 66 time series per model. Note that positive reduction means DCT works in the model's favour while negative reduction means it is better not to apply DCT. The third column in Table 5.2 shows the proportion of SMAPE reduction being positive over 66 time series. We refer to this as the positive SMAPE reduction ratio. This number signifies how often should DCT be chosen in our dataset. And the fourth and fifth columns contain the mean and standard deviation of the positive SMAPE reductions per model. These show the effectiveness in reducing SMAPE when DCT is chosen. We can see that although EN and ETS have a higher positive SMAPE reduction ratio (19.7% for EN and 24.24% for ETS), their reduction is not significant. And this is not surprising from what we saw in the previous summary statistics that EN and ETS do not really get affected by DCT in this dataset. On the other hand, LSVR, MLP and RF have about a ten percent positive SMAPE reduction ratio, while the average reduction in these cases is all over ten percent (11.03% for LSVR, 10.03% for MLP and 16.64% for RF). This is to say that there is about a ten percent chance that a modeller should consider using DCT in this dataset because it reduces SMAPE by at least ten percent on average (see Figure 5.3 for the distribution of positive SMAPE reductions per model).

	Avg. % reduction	Std. % reduction	+ reduction ratio	Avg. + % reduction	Std. + % reduction
EN	-0.04	0.18	19.70	0.05	0.06
ETS	0.64	8.32	24.24	4.67	16.09
LSVR	-13.43	16.83	10.61	11.03	20.43
MLP	-12.28	19.25	12.12	10.03	7.67
RF	-16.08	18.54	9.09	16.64	25.41

Table 5.2: Statistics of SMAPE reduction (%) using DCT (weekly finance)
This table shows the statistics of SMAPE reduction (in percentage) using DC Transformation. The dataset used consists of 66 weekly financial time series. The first two columns are the mean and standard deviation of SMAPE reduction for all 66 time series. The third column is the ratio of SMAPE reduction being positive out of 66 time series. The fourth and fifth columns are the mean and standard deviation of positive SMAPE reductions.

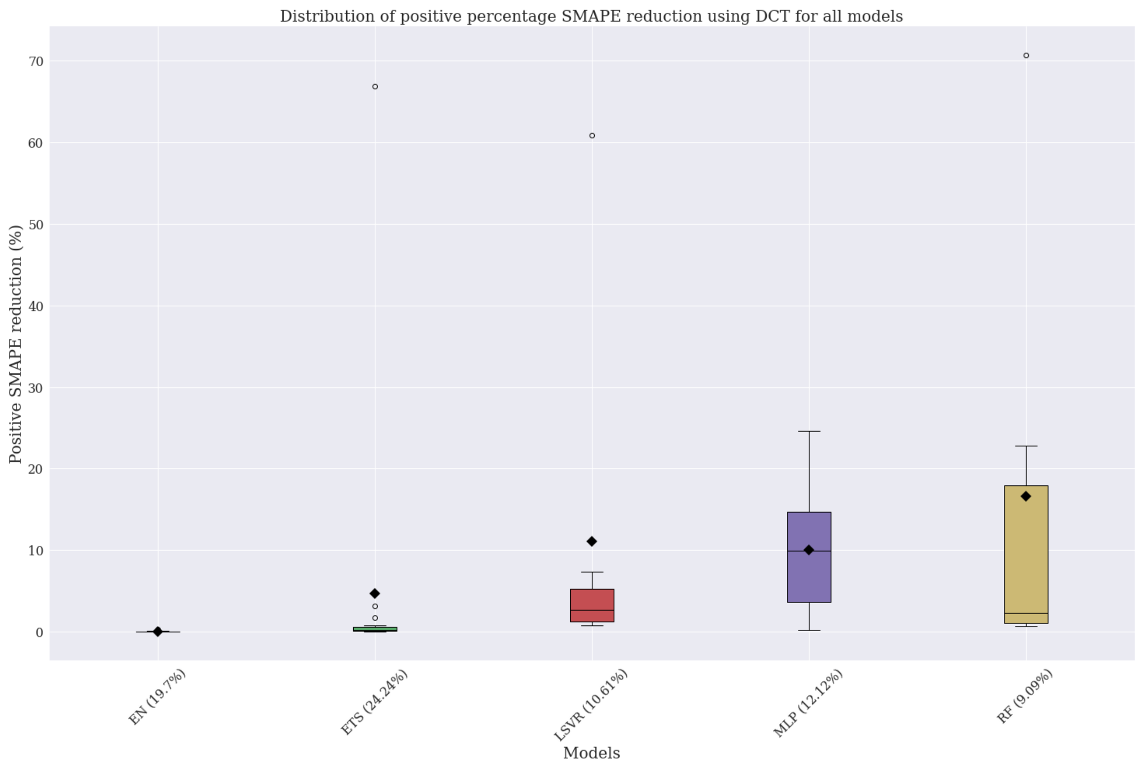

Figure 5.3: Distribution of positive SMAPE reductions (%) using DCT
The dataset used for this graph consists of 66 weekly time series in the financial domain. The boxplots are the distributions of positive SMAPE reductions (in percentage) using DC Transformation for each model. The diamonds are the mean of the corresponding distribution. The number next to the x-labels is the ratio of DCT giving positive SMAPE reductions out of 66 time series tested.

To investigate how exactly DCT help reduces SMAPE, we look at some time series in which we have high SMAPE reduction. Figure 5.4 is time series number 153 from the M4 dataset. It is a weekly financial time series. Figure 5.5 is MLP's prediction on the test set of time series number 153. The line in blue is the original data. The line in green marks the predictions of MLP without DCT. The line in red marks the predictions of MLP with DCT. On this test, MLP with DCT generates 24.6554% less error than MLP without DCT. We can see that the red line follows the blue line better while the green line floats above it. It seems DCT helps MLP to be more responsive to the downward trend.

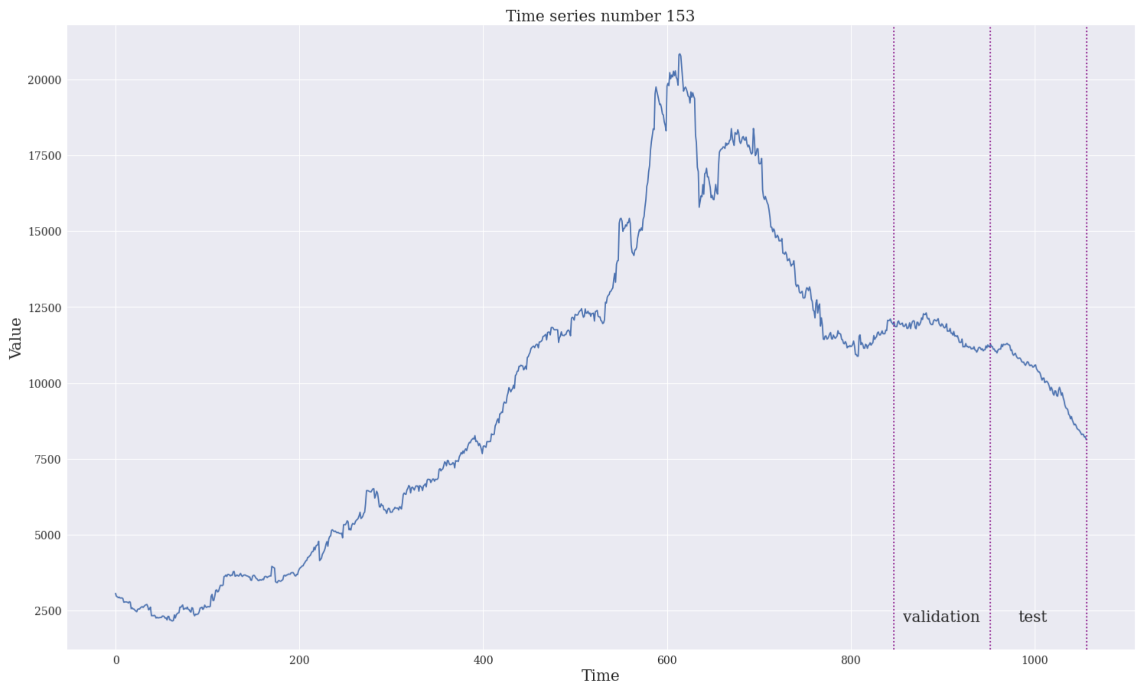

Figure 5.4: Time series No. 153 from M4 dataset

Time series No. 153 from the M4 dataset is a weekly financial time series. We mark the segments used for validation (model selection) and testing.

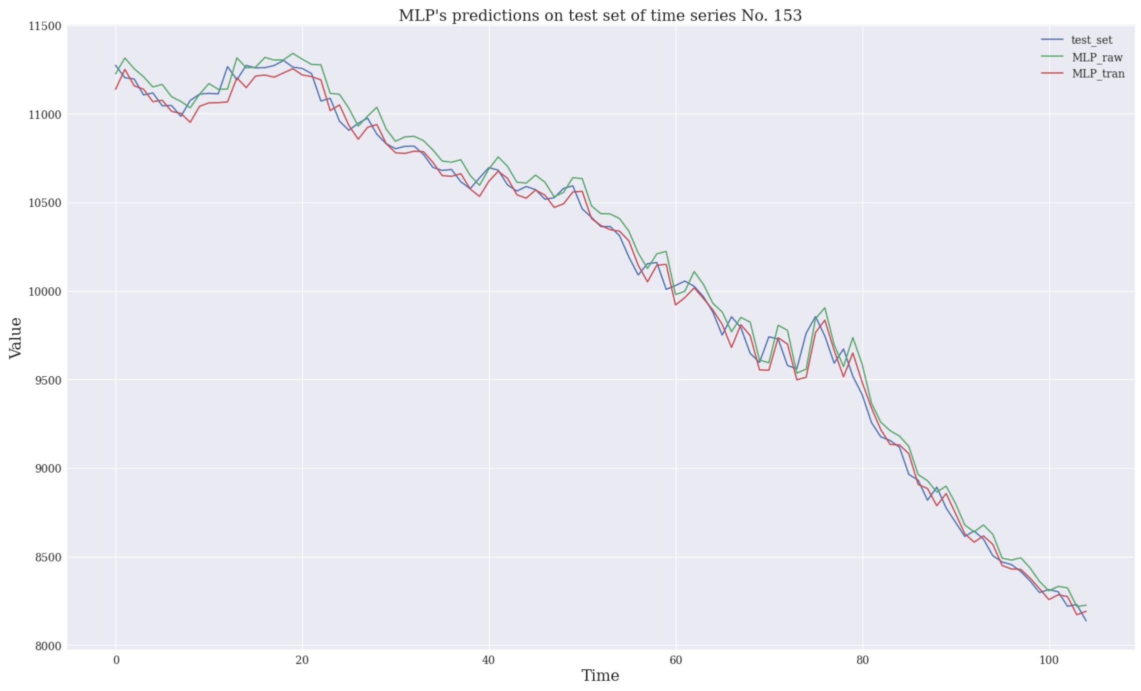

Figure 5.5: MLP on time series No. 153 from M4 dataset

This is MLP's prediction on the test set of time series No. 153 from the M4 dataset. DCT helps reduce 24.6554% of SMAPE in this test set.

Figure 5.6 is time series number 210, and Figure 5.7 is RF's precision on the test set. RF with DCT generates 70.7207% less SMAPE compared to RF_raw in this test set. Although it corrects itself immediately, we can see that the green line tends to deviate away from the blue line occasionally, even though it should not. We suppose DCT emphasises the current trend and stops RF from being impacted by the turbulence observed in the training set.

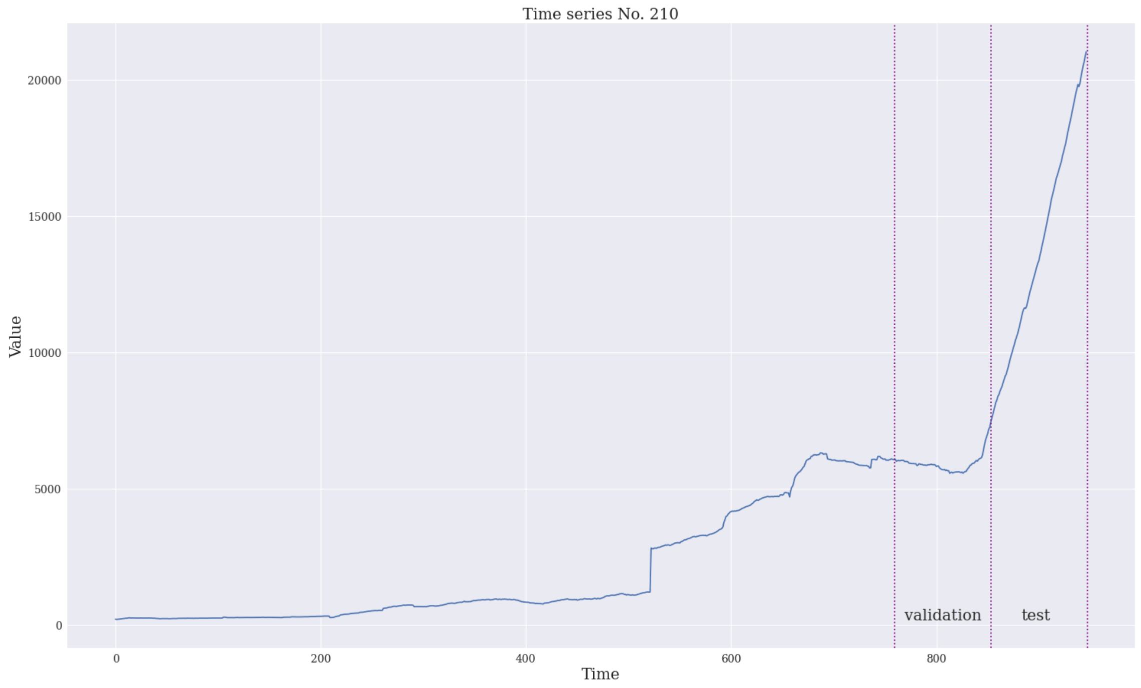

Figure 5.6: Time series No. 210 from M4 dataset

Time series No. 210 from the M4 dataset is a weekly financial time series. We mark the segments used for validation (model selection) and testing.

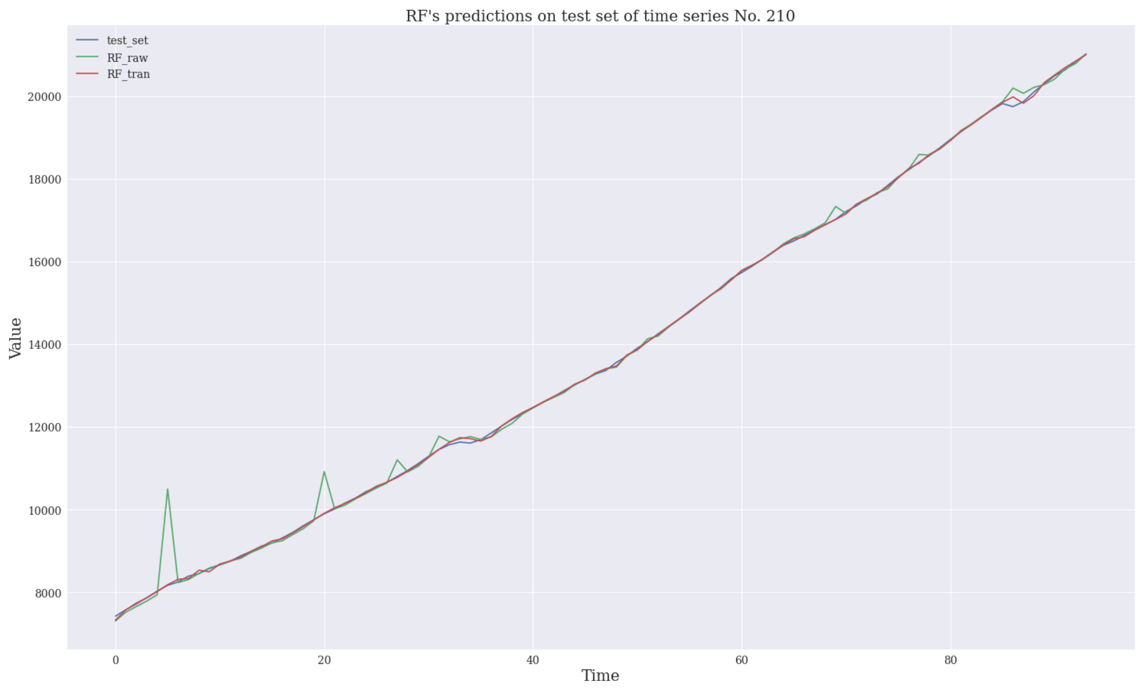

Figure 5.7: RF on time series No. 210 from M4 dataset
This is RF's prediction on the test set of time series No. 210 from the M4 dataset. The line in blue is the original data. The line in green marks the predictions of RF without DCT. The line in red marks the predictions of RF with DCT. On this test, RF with DCT generates 70.7207% less error than RF without DCT.

5.2 Analysing Daily Time Series

As to the daily results in Microeconomics, Macroeconomics and Finance, Figure 5.8 shows the distributions of SMAPEs for all 252 daily time series tested per agent. Figure 5.9 illustrates the agents' respective 90% confidence interval ranking. And Table 5.3 has all the summary statistics. Similar to what has been observed for weekly time series, applying DCT does not impact the overall performance for all models, and linear models continue to be top performant compared to others. Given the nature that the DC framework is sensitive to the frequency of the time series, we estimate that moving from weekly to daily frequency is not significant enough for the algorithm to generate a different impact.

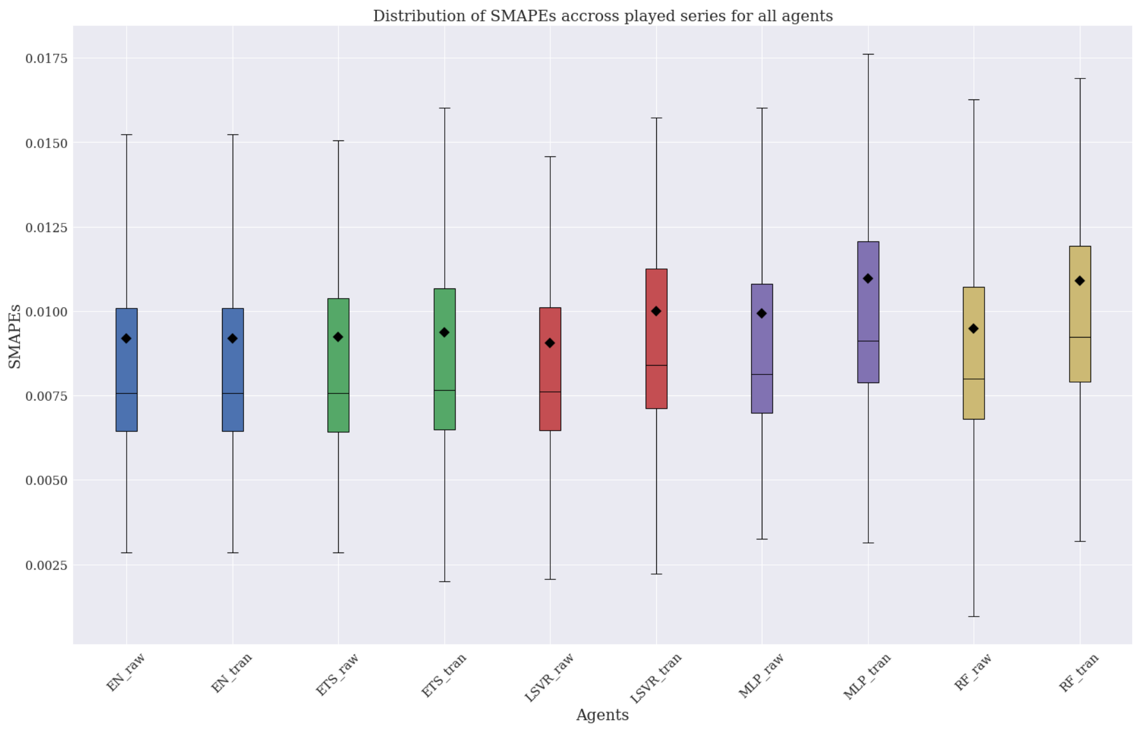

Figure 5.8: Distribution of SMAPE across 252 daily time series per agent

The dataset used for this graph consists of 252 daily time series in micro, macro, and financial domains. The boxplots are distributions of SMAPEs (described in Equation 4.2.5) across all 252 time series per agent. The diamonds are the mean of the corresponding distribution. Agents with the same model are coloured the same.

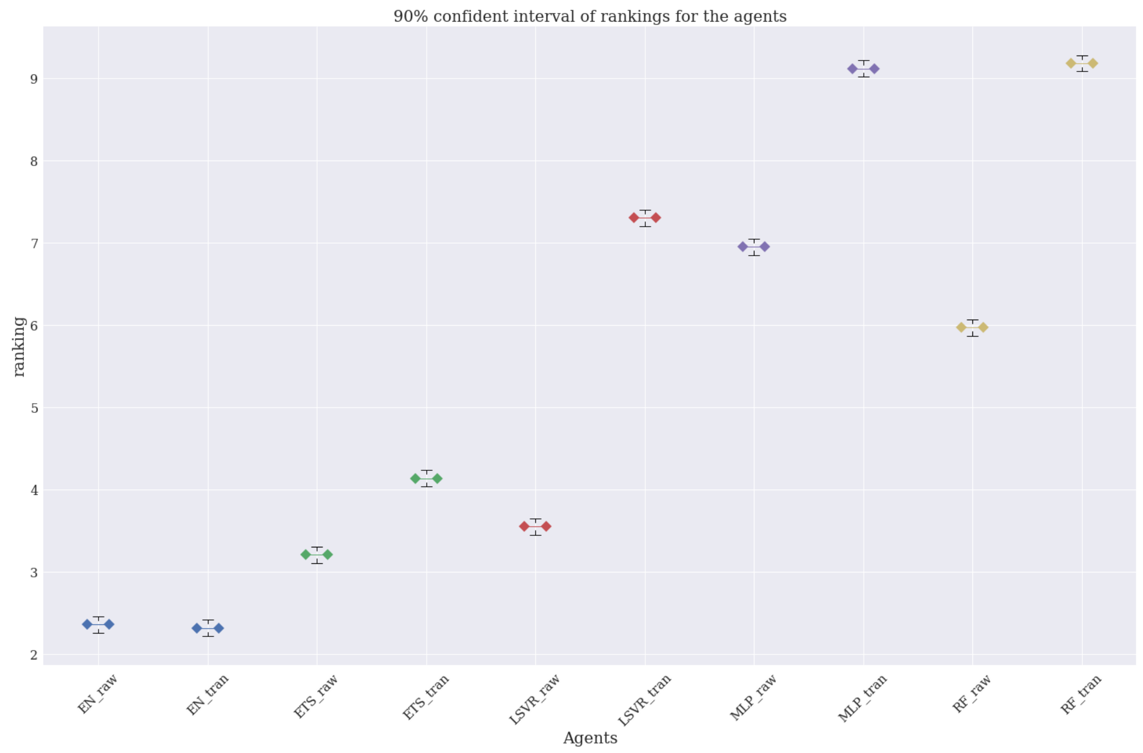

Figure 5.9: 90% rank interval across 252 daily time series per agent
The dataset used for this graph consists of 252 daily time series. The intervals are rank intervals under 90% confidence level (Equation 4.2.5) over 252 time series per agent. Agents with the same model are coloured the same.

	Avg. SMAPE	Std. SMAPE	Avg. rank	Std. rank	90% rank interval	frac best
EN_raw	0.009179	0.00977	2.357	1.285	(2.258, 2.456)	0.266
EN_tran	0.009186	0.009906	2.317	1.267	(2.218, 2.416)	0.298
ETS_raw	0.009236	0.010134	3.206	1.57	(3.107, 3.305)	0.175
ETS_tran	0.009377	0.010207	4.135	1.941	(4.036, 4.234)	0.123
LSVR_raw	0.009059	0.007686	3.548	1.924	(3.449, 3.647)	0.298
LSVR_tran	0.009997	0.0107	7.302	1.326	(7.203, 7.401)	0.004
MLP_raw	0.009934	0.008717	6.948	1.802	(6.849, 7.047)	0.024
MLP_tran	0.01097	0.010123	9.115	1.109	(9.016, 9.214)	0.0
RF_raw	0.009471	0.008224	5.968	1.898	(5.869, 6.067)	0.075
RF_tran	0.010889	0.010496	9.179	1.163	(9.08, 9.278)	0.0

Table 5.3: Summary statistics of all agents on daily time series
The dataset consists of 252 time series including micro, macro and financial domain. The average length of the time series is 1179.99. The first and second columns are the mean and standard deviation of the SMAPEs for each agent. The third and fourth columns are the mean and standard deviation of the ranks (Equation 4.2.5) for each agent. The fifth column has the 90% rank intervals. The final column has the fraction best (Equation 4.2.5).

Moving on to SMAPE reduction caused by DCT per time series, Table 5.4 has the statistics for each model. We can see the continuation of linear models having a high positive reduction ratio in column three but insignificant reduction in column four. On the other hand, MLP with DCT generates less error 11.51% of the time. Moreover, the mean of these positive reductions is 11.25%. This indicates that non-linear models can better process the information provided by DC Transformation on this test. Figure 5.10 has the distributions of positive SMAPE reductions per model.

	Avg. % reduction	Std. % reduction	+ reduction ratio	Avg. + % reduction	Std. + % reduction
EN	0.02	0.24	23.41	0.12	0.44
ETS	-1.13	3.87	30.95	0.66	3.37
LSVR	-9.00	5.70	3.17	1.86	1.12
MLP	-12.04	17.01	11.51	11.25	10.02
RF	-15.09	12.60	3.57	9.61	7.89

Table 5.4: Statistics of SMAPE reduction (%) using DCT (daily cross-domain)
This table shows the statistics of SMAPE reduction (in percentage) using DC Transformation. The dataset comprises 252 daily time series across finance, micro, and macro domains. The first two columns are the mean and standard deviation of SMAPE reduction for all 252 time series. The third column is the ratio of SMAPE reduction being positive out of 252 time series. The fourth and fifth columns are the mean and standard deviation of positive SMAPE reductions.

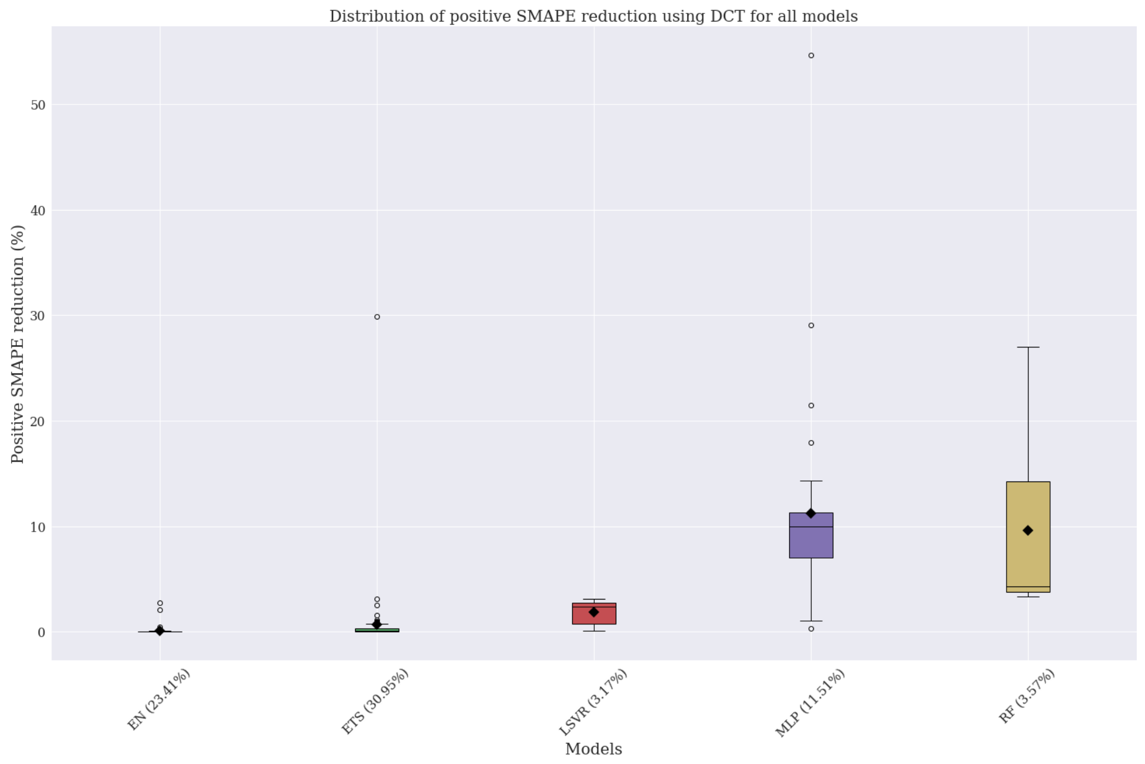

Figure 5.10: Distribution of positive SMAPE reductions (%) using DCT
The dataset used for this graph consists of 252 daily time series in micro, macro and financial domain. The boxplots are the distributions of positive SMAPE reductions (in percentage) using DC Transformation for each model. The diamonds are the mean of the corresponding distribution. The number next to the x-labels is the ratio of DCT giving positive SMAPE reductions out of 252 time series tested.

To look into how the models are doing, Figure 5.11 is a daily macro time series numbered 35 in the M4 dataset and Figure 5.12 has the predictions generated by ETS in the test set. SMAPE reduction caused by DCT in this figure is 29.8947%. We can observe similar characteristics from what we saw in the weekly time series that agents with DCT can react faster and more accurate to a clear trend given the existence of drastic shocks in previous observations.

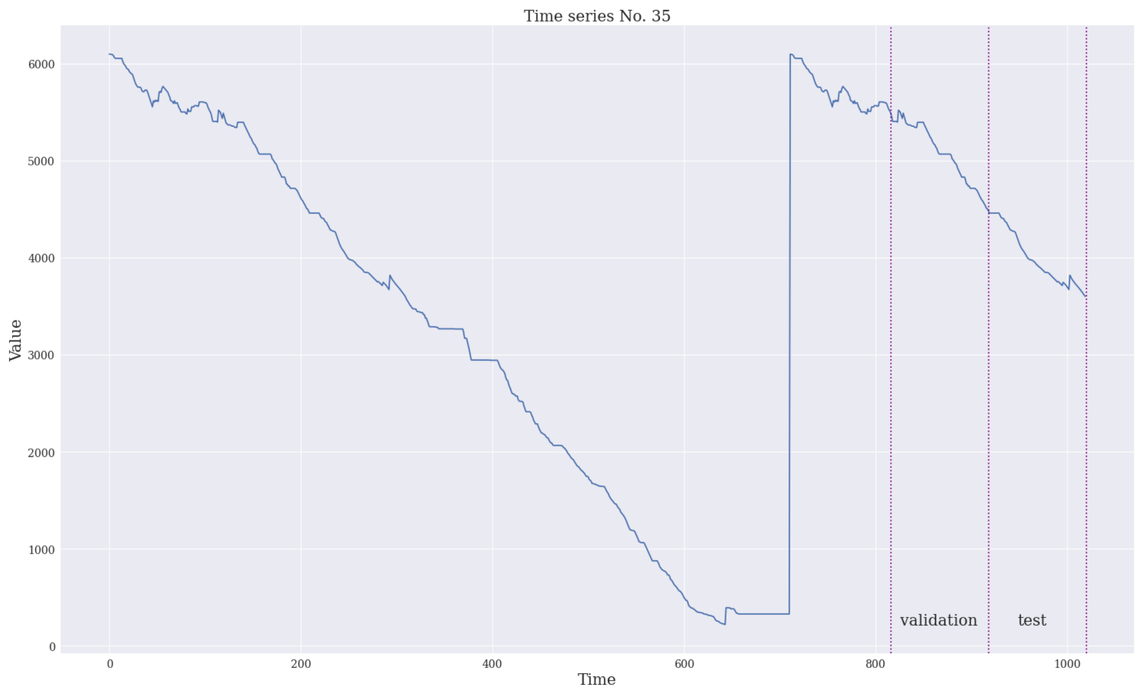

Figure 5.11: Time series No. 35 from M4 dataset

Time series No. 35 from the M4 dataset is a daily macro time series. We mark the segments used for validation (model selection) and testing.

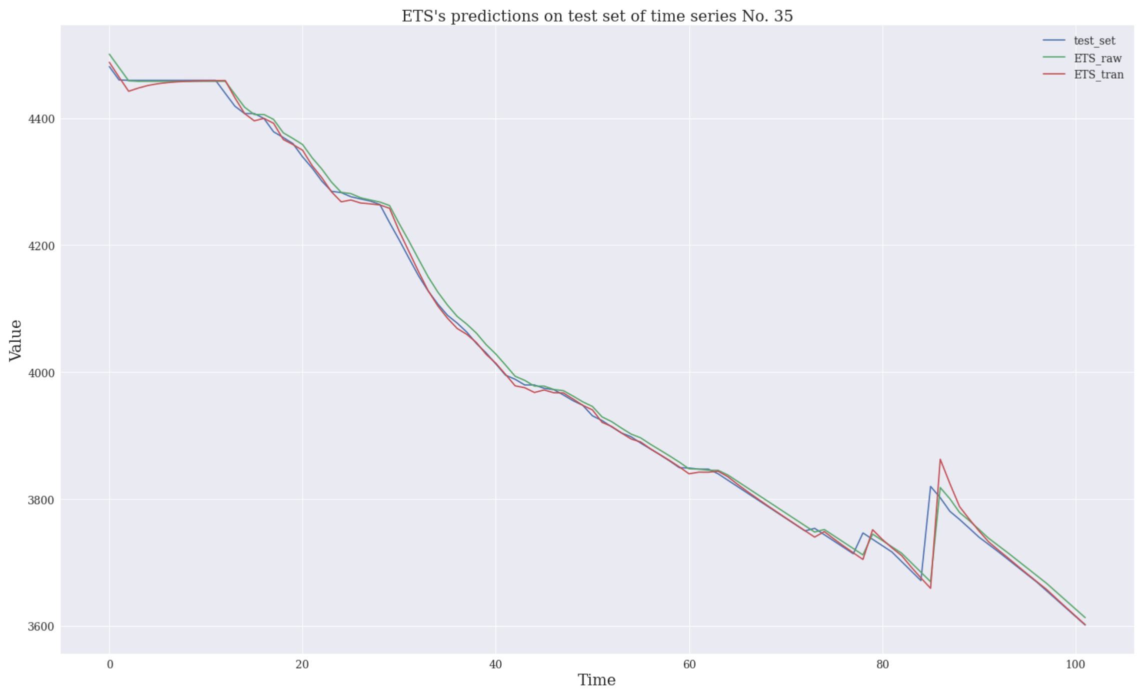

Figure 5.12: ETS on time series No. 35 from M4 dataset

This is ETS's prediction on the test set of time series No. 35 from the M4 dataset.

Figure 5.13 is another daily macro time series in the M4 dataset, and Figure 5.14 has the predictions generated by MLP in the test set. DCT reduces 54.6662% of SMAPE in this test. It is clear that MLP_raw is heavily affected by the drastic rise in the test set and takes a while to accustom to that. On the other hand, MLP_tran is immune to the shock.

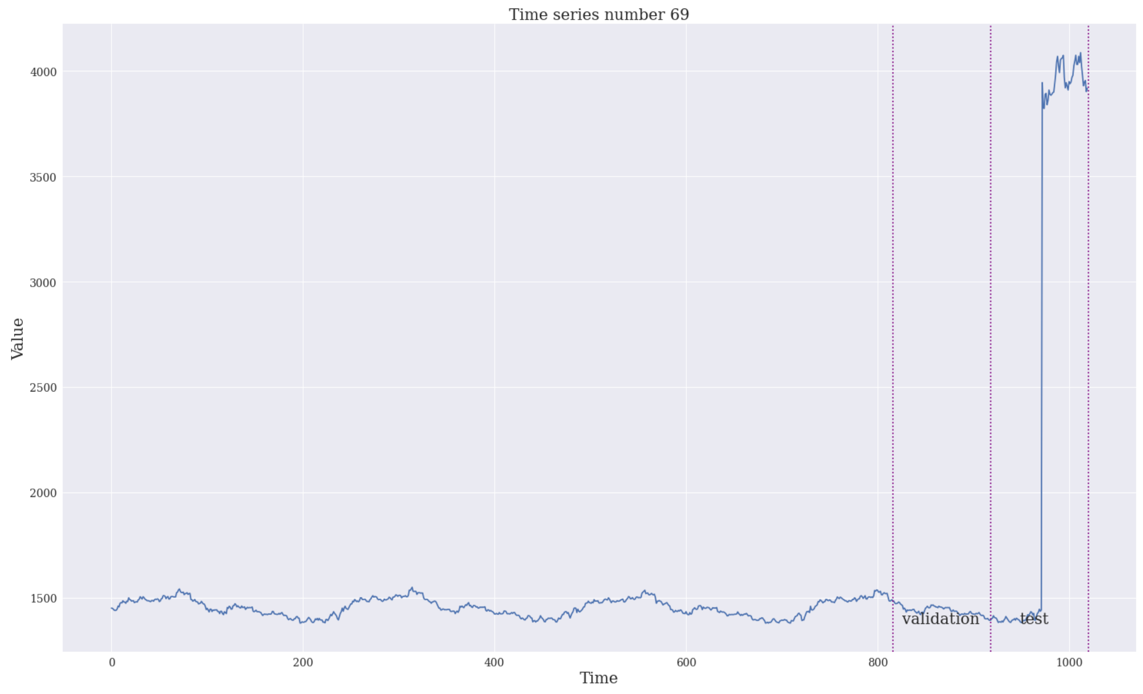

Figure 5.13: Time series No. 69 from M4 dataset

Time series No. 69 from the M4 dataset is a daily macro time series. We mark the segments used for validation (model selection) and testing.

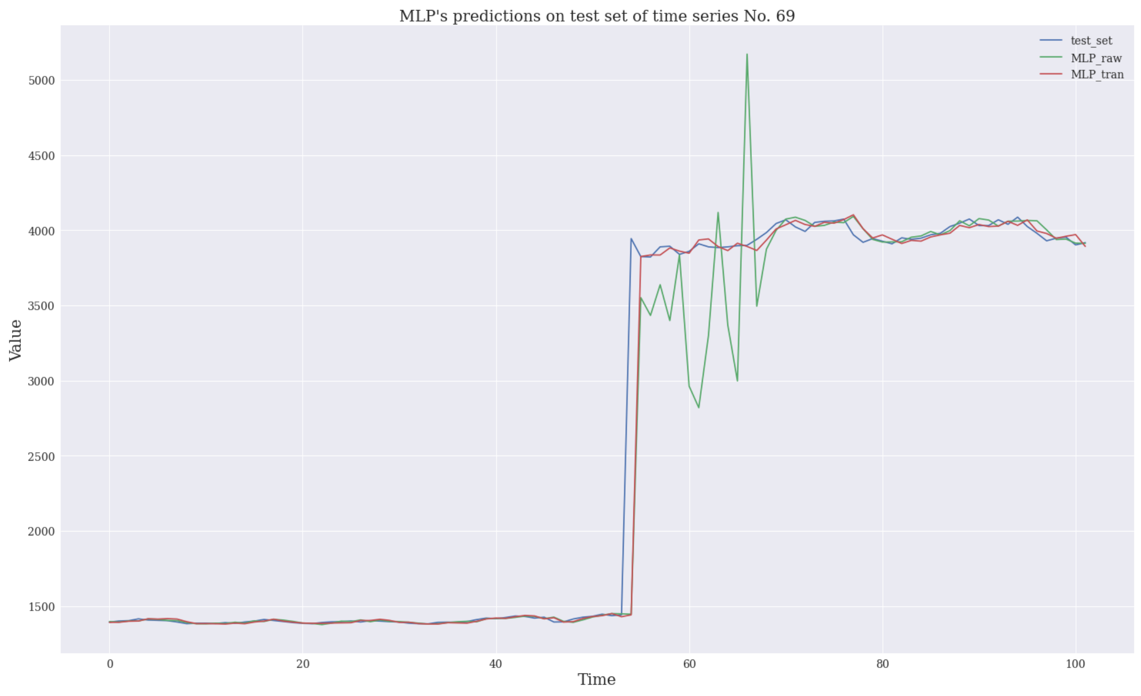

Figure 5.14: MLP on time series No. 69 from M4 dataset
This is MLP's prediction on the test set of time series No. 69 from the M4 dataset.

5.3 Analysing Hourly Time Series

We then analyse experimental results concerning 140 hourly time series in which things start looking differently. Figure 5.15 contains the distributions of SMAPE per agent, Figure 5.16 has the rank intervals, and Table 5.5 holds all the summary statistics. Notice that EN with DCT has a significantly smaller error in general.

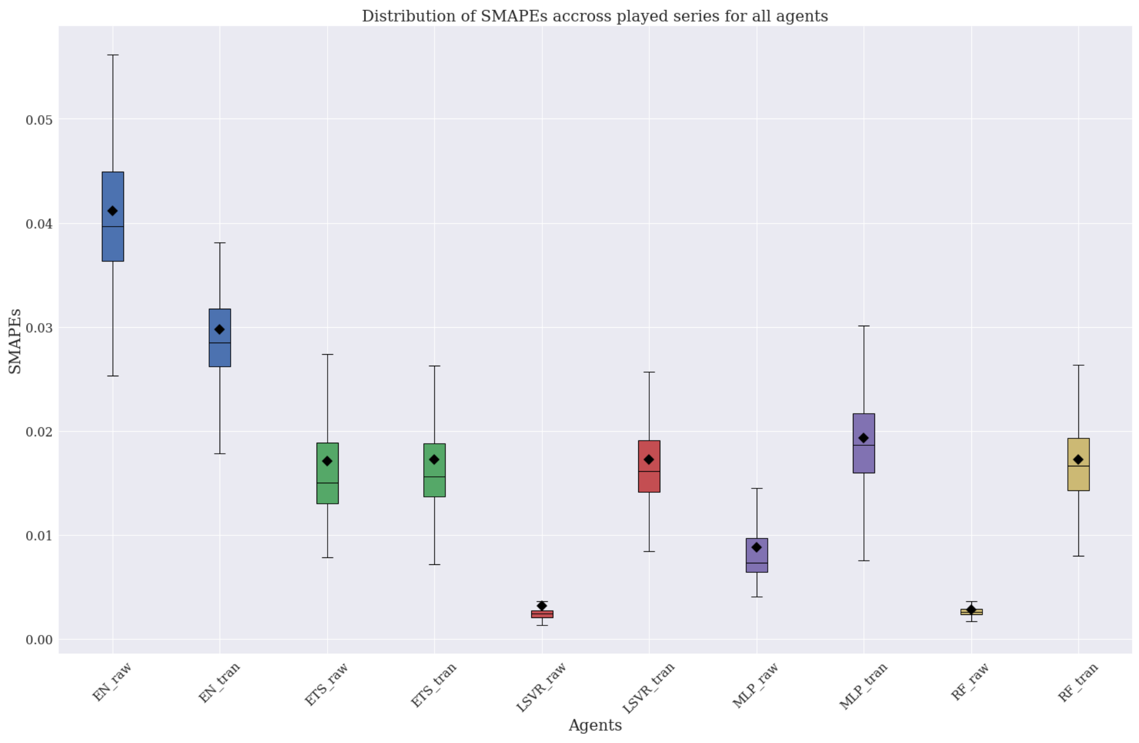

Figure 5.15: Distribution of SMAPE across 140 hourly time series per agent
The dataset used for this graph consists of 140 hourly time series in unknown domains. The boxplots are distributions of SMAPEs (described in Equation 4.2.5) across 140 time series per agent. The diamonds are the mean of the corresponding distribution. Agents with the same model are coloured the same.

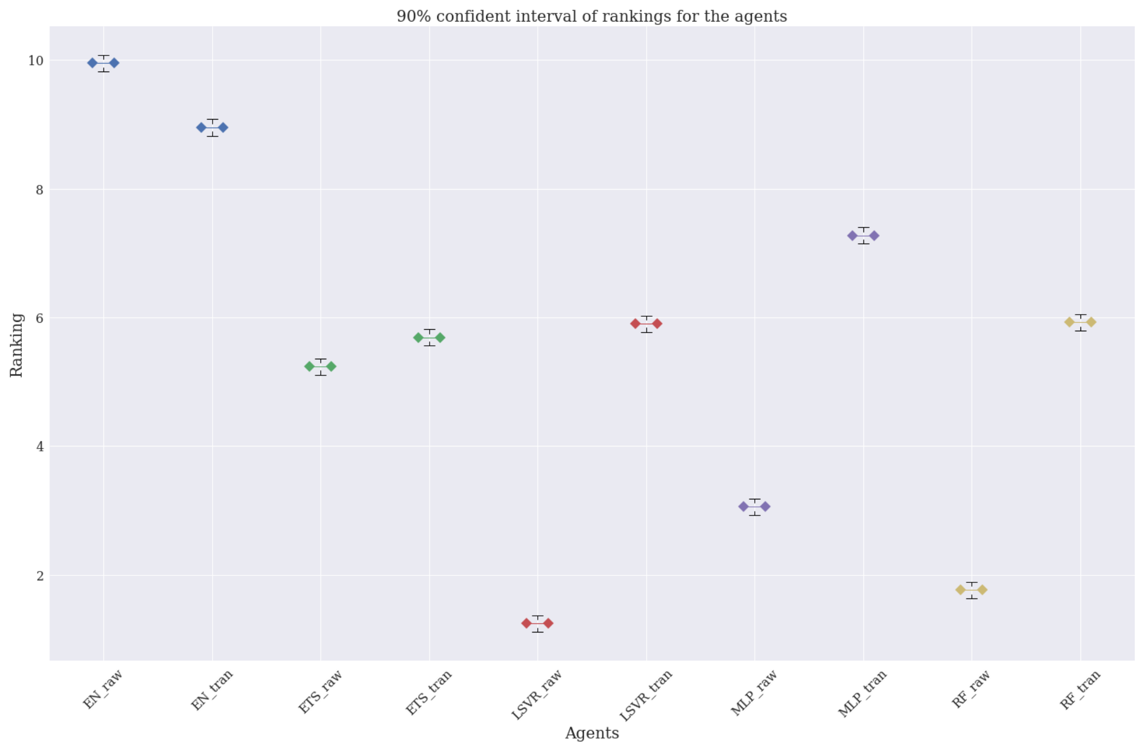

Figure 5.16: 90% rank interval across 140 hourly time series per agent

The dataset used for this graph consists of 140 hourly time series. The intervals are rank intervals under 90% confidence level (Equation 4.2.5) over 140 time series per agent. Agents with the same model are coloured the same.

	Avg. SMAPE	Std. SMAPE	Avg. rank	Std. rank	90% rank interval	frac best
EN_raw	0.041144	0.010202	9.95	0.218	(9.822, 10.078)	0.0
EN_tran	0.029771	0.006564	8.95	0.525	(8.822, 9.078)	0.0
ETS_raw	0.017082	0.007781	5.236	1.477	(5.108, 5.364)	0.0
ETS_tran	0.01727	0.006678	5.686	1.063	(5.558, 5.814)	0.0
LSVR_raw	0.003148	0.004825	1.243	0.445	(1.115, 1.371)	0.764
LSVR_tran	0.017249	0.005926	5.9	1.289	(5.772, 6.028)	0.0
MLP_raw	0.008819	0.006091	3.057	0.41	(2.929, 3.185)	0.0
MLP_tran	0.019317	0.006434	7.271	1.212	(7.143, 7.399)	0.0
RF_raw	0.00284	0.002112	1.764	0.424	(1.636, 1.892)	0.236
RF_tran	0.017274	0.005028	5.921	1.358	(5.793, 6.049)	0.0

Table 5.5: Summary statistics of all agents on hourly unknown domain time series

The dataset consists of 140 time series. The average length of the time series is 1008. The first and second columns are the mean and standard deviation of the SMAPEs for each agent. The third and fourth columns are the mean and standard deviation of the ranks (Equation 4.2.5) for each agent. The fifth column has the 90% rank intervals. The final column has the fraction best (Equation 4.2.5).

Table 5.6 contains the statistics about SMAPE reductions per time series for the models, and Figure 5.17 illustrates the distributions of the positive SMAPE reductions. We can see that EN has a 97.14% positive reduction ratio, with the average positive reduction being 27.33%. Also, LSVR and RF have a 0% positive reduction ratio. Judging from previous summary statistics, we can see that LSVR and RF both do so well without DCT and thus making them hard to outperform (recall that SMAPE reduction is a relative measure).

	Avg. % reduction	Std. % reduction	+ reduction ratio	Avg. + % reduction	Std. + % reduction
EN	26.55	7.72	97.14	27.33	6.33
ETS	-3.09	9.02	27.86	5.75	10.30
LSVR	-598.10	211.77	0.00	NaN	NaN
MLP	-135.48	46.83	0.71	9.02	0.00
RF	-548.46	161.49	0.00	NaN	NaN

Table 5.6: Statistics of SMAPE reduction (%) using DCT (hourly unknown)
This table shows the statistics of SMAPE reduction (in percentage) using DC Transformation. The dataset consists of 140 hourly time series in an unknown domain. The first two columns are the mean and standard deviation of SMAPE reduction for all 140 time series. The third column is the ratio of SMAPE reduction being positive out of 140 time series. The fourth and fifth columns are the mean and standard deviation of positive SMAPE reductions. Due to not having any positive reduction, LSVR and RF have NaN values in the fourth and fifth columns.

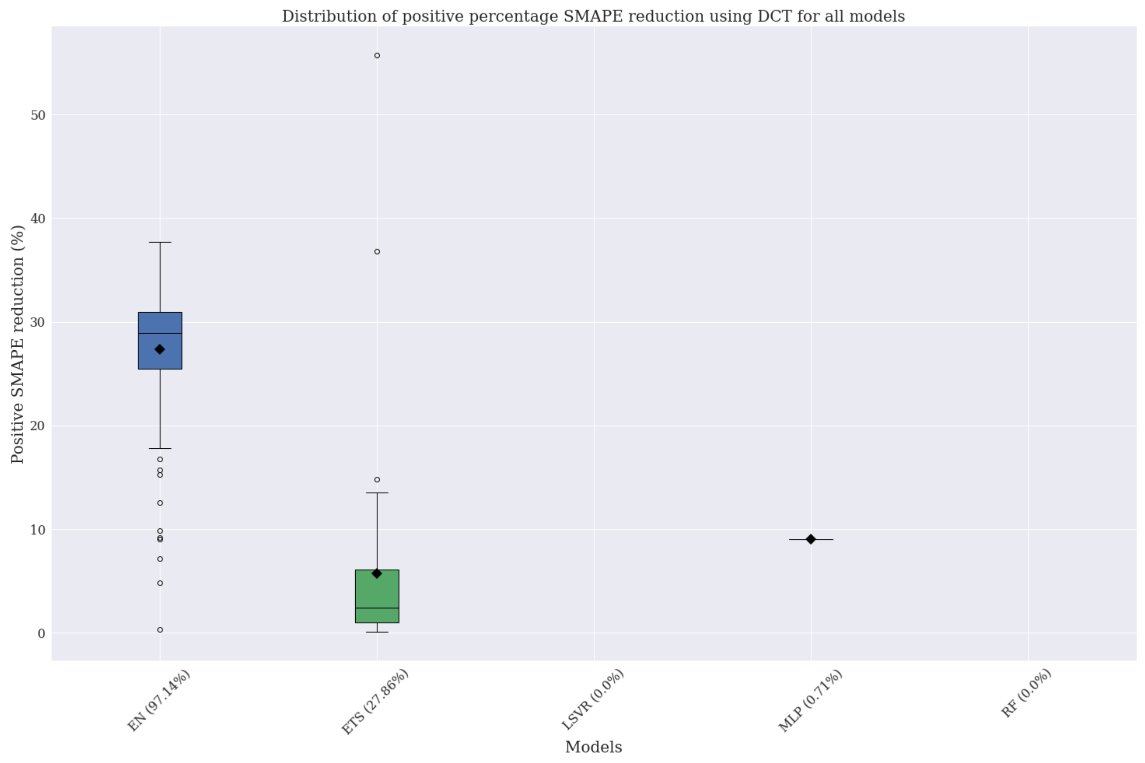

Figure 5.17: Distribution of positive SMAPE reductions (%) using DCT
The dataset used for this graph consists of 140 hourly time series in an unknown domain. The boxplots are the distributions of positive SMAPE reductions (in percentage) using DC Transformation for each model. The diamonds are the mean of the corresponding distribution. The number next to the x-labels is the ratio of DCT giving positive SMAPE reductions out of 140 time series tested.

To further investigate model behaviours, we look at time series number 271 (see Figure 5.18), which is of hourly frequency but an unknown domain. It is clear that such time series behaves very differently from what we have previously for the micro, macro, and financial domains. It turns out most time series of hourly frequency we tested are like this.

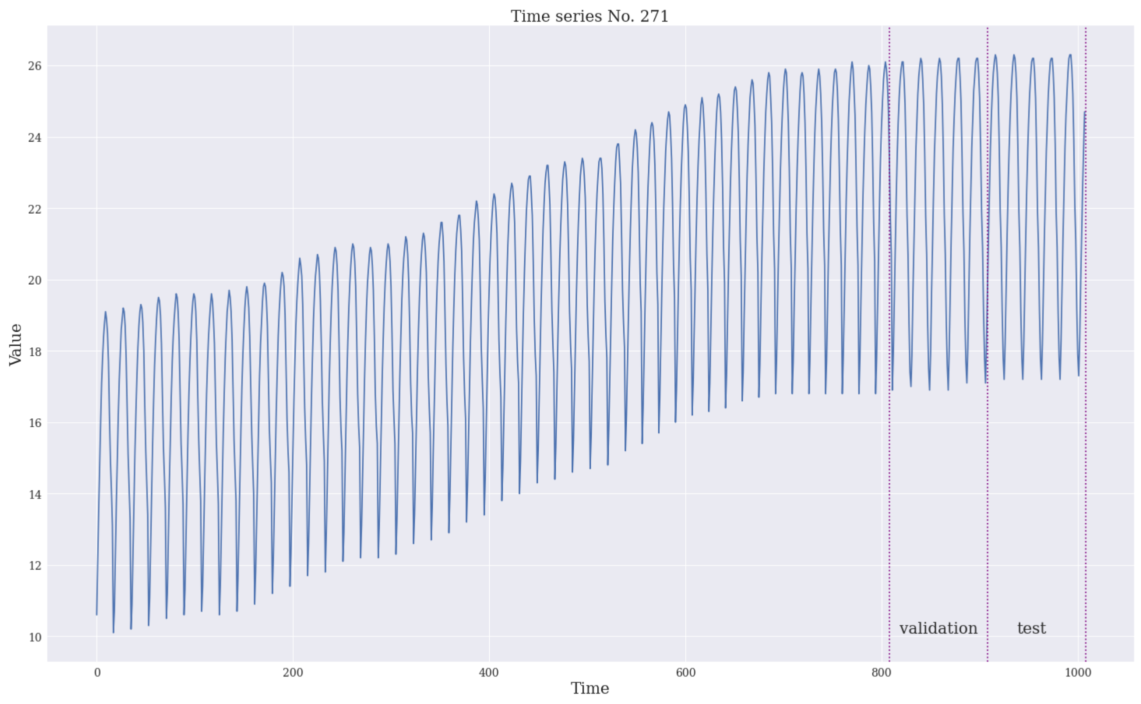

Figure 5.18: Time series No. 271 from M4 dataset
Time series No. 271 from the M4 dataset is an hourly time series from the unknown domain. We mark the segments used for validation (model selection) and testing.

Figure 5.19 marks the predictions of EN on the test set of time series 271 in which DCT reduced 31.1082% of SMAPE. We can see that although EN_tran overreacts around the peaks, the reason EN_tran is doing so well is that such cyclical dynamics spend long period of time experiencing linear trends between one peak and another. And we have already learned from the previous results that DCT is good at following trends. As to other models, Figure 5.20 and 5.21 show that LSVR and MLP without DCT are following the blue line very well such that they are hard to outperform. Moreover, LSVR_raw can even forecast the upcoming turn when there is a peak ahead.

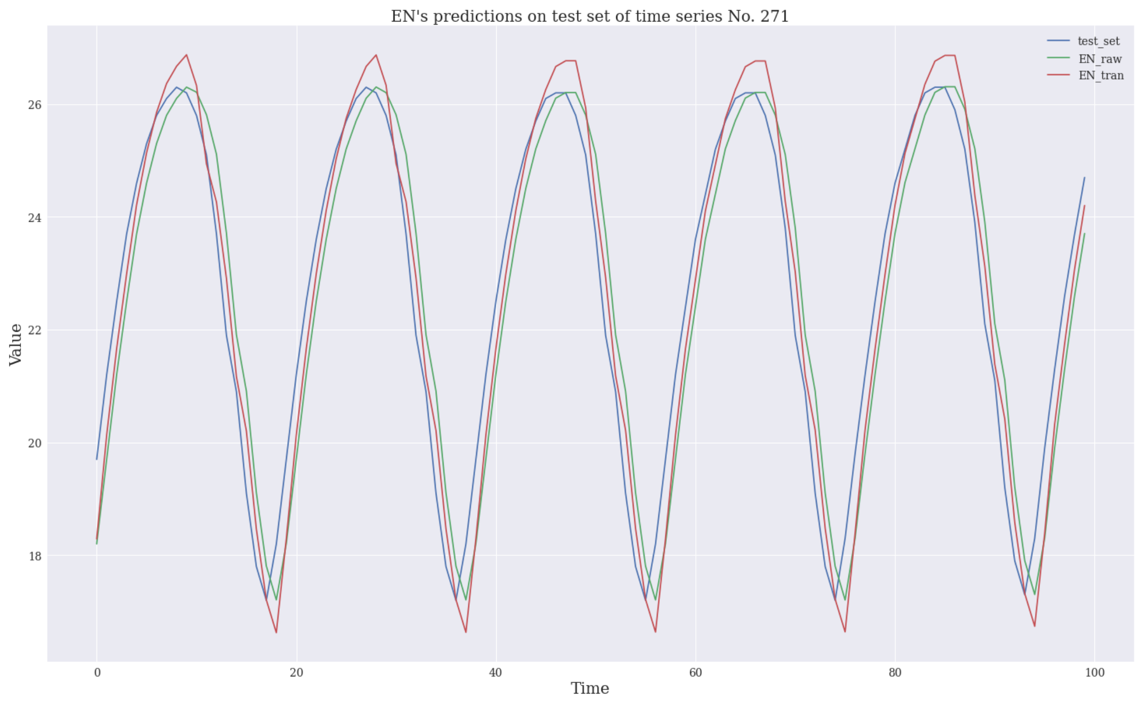

Figure 5.19: EN on time series No. 271 from M4 dataset
This is EN's prediction on the test set of time series No. 271 from the M4 dataset.

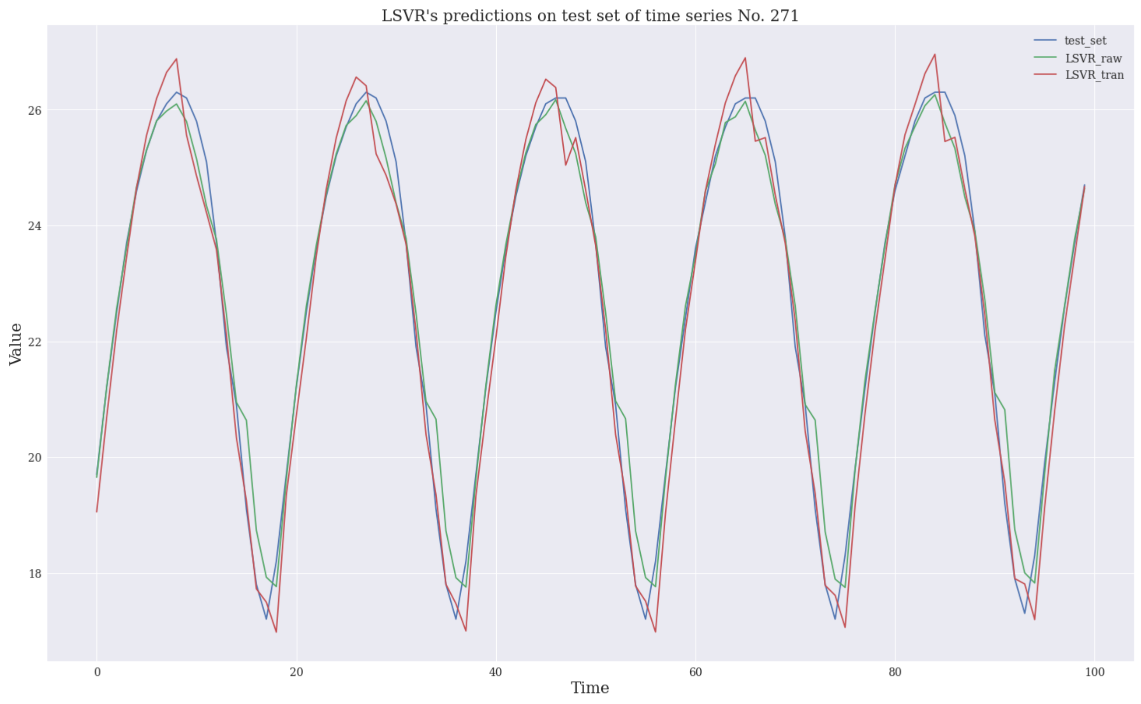

Figure 5.20: LSVR on time series No. 271 from M4 dataset
This is LSVR's prediction on the test set of time series No. 271 from the M4 dataset.

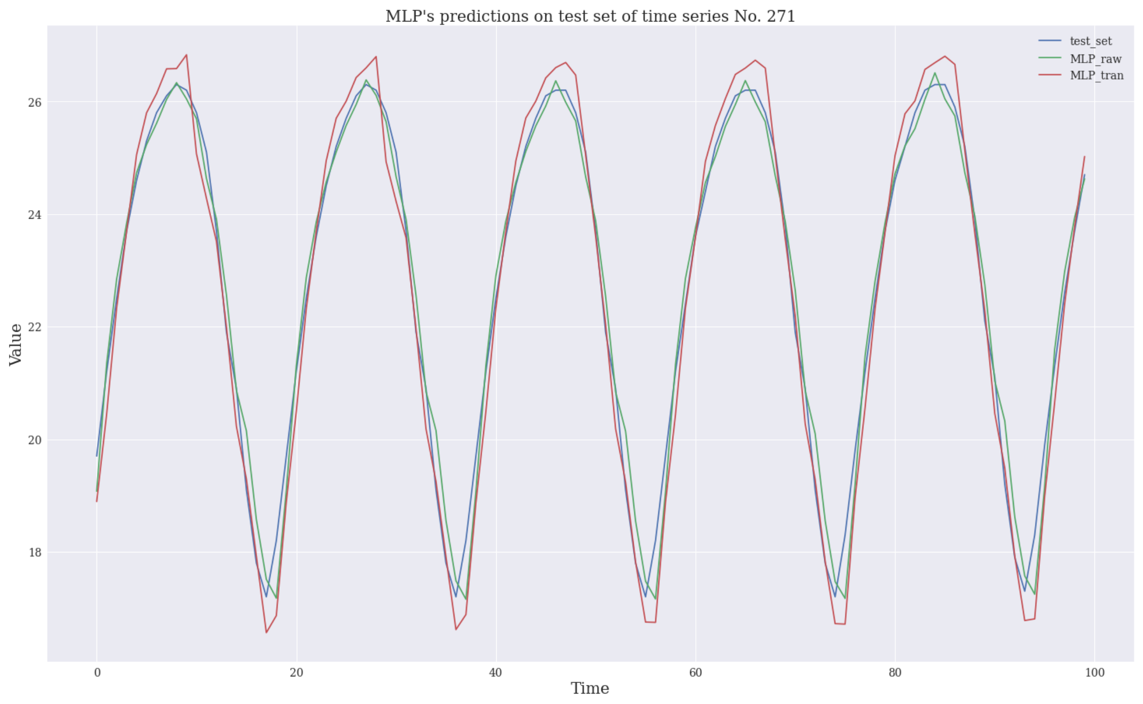

Figure 5.21: MLP on time series No. 271 from M4 dataset
This is MLP's prediction on the test set of time series No. 271 from the M4 dataset.

5.4 Discussions

In our experiment, we trained models with and without DCT in all cases to investigate the features of DCT. However, in practice, applying DCT should be considered an optional policy. One should set a hyperparameter that allows the model to decide not to use DCT if it deems inappropriate during model selection. And our analyses on positive SMAPE reductions show there can be good incentives for the model to choose to apply DCT.

In general, we estimate that it can be worth it to try using DC Transformation in modelling tasks. DCT reduces noise where the interpolation has been performed. We can see that DC Transformation has two components, each having different effects on the output: the DC algorithm captures the extreme events while the interpolation component creates a smooth process between them. Moreover, DCT preserves the information carried by the peaks without toning them down like typical smoothing methods do. This gives DCT the ability to reduce noise while also capturing extreme events. Additionally, due to the interpolation being linear, DCT emphasises the trend in the time series that can lead to better model performance.

Seeing that interpolation does play an essential role in DCT, we tried other interpolation methods during exploratory trials. We tried quadratic and cubic interpolations that fit degree-two or degree-three polynomials in the intervals. We find both interpolations inadequate because they tend to create peaks in the intervals by their polynomial nature, as shown in Figure 4.3. The intervals are supposed to be smooth because it is the DC algorithm's job to capture the extrema. Next, we tried Akima interpolation, which is a middle point between linear and quadratic interpolation (Akima (1970)). We find it does perform better than quadratic interpolation, but we conclude that linear interpolation is still a better choice for our experiments.

Chapter 6

Conclusion

This chapter concludes our report with two paragraphs: we first outline the contributions we have made in this report and then discuss possible extensions to our work.

On the higher level, this is an exploratory report - we present an original idea of bringing the Directional Change Framework and target transformation techniques together for univariate time series forecasting. The report also covers a robust experiment and critical analyses of the proposed methodology. In particular, there are several highlights to this report. First, we comprehensively discuss how target transformation techniques work in univariate time series forecasting. We cover both the intuitions and technical details on the level that can be generalised to most Machine Learning univariate time series regression modelling cases. Secondly, in addition to addressing the DC framework, we present the pseudocode Algorithm 2 for the DC event identification algorithm. Although there are other versions of pseudocode for such an algorithm in other publications, they normally are not for general purposes, i.e., their versions are for specific tasks. For example, the one in Glattfelder et al. (2011) is for counting the number of directional change events and the one in Golub et al. (2016) is for measuring the overshoots with a δ threshold. On the other hand, Algorithm 2 is for general purposes - it marks the states of the input time series and can then be further extended for any other usages. Thirdly, we devise a novel target transformation method based on the DC framework and interpolation methods. Our innovation demonstrates a comprehensive understanding of both methodologies. Finally, we present interesting insights on DC Transformation according to our large-scale experiment, critical analyses, and robust scientific practice.

For prospective future studies on DC Transformation, we reckon several directions are worth further investigation. First, better estimates of the threshold value δ can be devised. Additionally, more complex methods associated with such parameters can be explored. For example, instead of deciding on a δ to use, Alpha Engine utilises four different sets of threshold values at the same time to gain more insights into the intrinsic events (see Golub et al. (2018)). Secondly, as we tune the δ value as a hyperparameter, information on why the model chose a δ value for each time series can be interesting. This can be a potential source of information to further characterise a time series. Thirdly, more sophisticated methods for reconstructing the time series can be devised. DCT simply interpolates the values extracted by the DC algorithm, but perhaps more intelligent mechanisms can better take advantage of the information extracted by the DC algorithm. Fourth, in light of the DC framework research mostly focusing on high-frequency data, datasets with higher frequencies should be tested, e.g., ten-minute or tick-by-tick datasets. We think it is very likely that high-frequency data is where DCT can fully live up to its potential. Finally, other combinations of forecasting objectives and models should be experimented to expand the horizon of our understanding of DC Transformation.

Bibliography

A. Adegboye and M. Kampouridis. Machine learning classification and regression models for predicting directional changes trend reversal in fx markets. *Expert Systems with Applications*, 173:114645, 2021.

S. Aghabozorgi, A. Seyed Shirkhorshidi, and T. Ying Wah. Time-series clustering - a decade review. *Information Systems*, 53:16–38, 2015. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2015.04.007. URL `https://www.sciencedirect.com/science/article/pii/S0306437915000733`.

N. Ahmed, A. Atiya, N. Gayar, and H. El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29:594–621, 08 2010. URL `https://www.tandfonline.com/doi/abs/10.1080/07474938.2010.481556`.

H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *Journal of the ACM (JACM)*, 17(4):589–602, 1970.

M. Aloud, E. Tsang, R. Olsen, and A. Dupuis. A directional-change event approach for studying financial time series. *Economics*, 6(1), 2012.

A. C. Atkinson, M. Riani, and A. Corbellini. The box–cox transformation: Review and extensions. *Statistical Science*, 36(2):239–255, 2021.

M. S. Bartlett. The use of transformations. *Biometrics*, 3(1):39–52, 1947. ISSN 0006341X, 15410420. URL `http://www.jstor.org/stable/3001536`.

C. Bergmeir, R. J. Hyndman, and J. M. Benítez. Bagging exponential smoothing methods using stl decomposition and box–cox transformation. *International journal of forecasting*, 32(2):303–312, 2016.

P. J. Bickel and K. A. Doksum. An analysis of transformations revisited. *Journal of the american statistical association*, 76(374):296–311, 1981.

P. Bloomfield. *Fourier analysis of time series: an introduction.* John Wiley & Sons, 2004.

G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

U. C. Bureau. *X-13ARIMA-SEATS Reference Manual, Version 1.0.* U.S. Census Bureau, U.S. Department of Commerce, 2012.

R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition. *J. Off. Stat*, 6(1):3–73, 1990.

E. B. Dagum and S. Bianconcini. *Seasonal adjustment methods and real time trend-cycle estimation.* Springer, 2016.

I. Daubechies. *Ten lectures on wavelets.* SIAM, 1992.

J. G. De Gooijer and R. J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.01.001. URL `https://www.sciencedirect.com/science/article/pii/S0169207006000021`. Twenty five years of forecasting.

J. D. Farmer and D. Foley. The economy needs agent-based modelling. *Nature*, 460 (7256):685–686, 2009.

W. A. Fuller. *Introduction to statistical time series.* John Wiley & Sons, 2009.

R. Gençay, M. Dacorogna, U. A. Muller, O. Pictet, and R. Olsen. *An introduction to high-frequency finance.* Elsevier, 2001.

J. B. Glattfelder and A. Golub. Bridging the gap: Decoding the intrinsic nature of time in market data. *arXiv preprint arXiv:2204.02682*, 2022.

J. B. Glattfelder, A. Dupuis, and R. B. Olsen. Patterns in high-frequency fx data: discovery of 12 empirical scaling laws. *Quantitative Finance*, 11(4):599–614, 2011.

A. Golub, G. Chliamovitch, A. Dupuis, and B. Chopard. Multi-scale representation of high frequency market liquidity. *Algorithmic Finance*, 5(1-2):3–19, 2016.

A. Golub, J. B. Glattfelder, and R. B. Olsen. The alpha engine: Designing an automated trading algorithm. In *High-Performance Computing in Finance*, pages 49–76. Chapman and Hall/CRC, 2018.

P. Goodwin et al. The holt-winters approach to exponential smoothing: 50 years old and going strong. *Foresight*, 19(19):30–33, 2010.

V. M. Guerrero. Time-series analysis supported by power transformations. *Journal of forecasting*, 12(1):37–48, 1993.

D. M. Guillaume, M. M. Dacorogna, R. R. Davé, U. A. Müller, R. B. Olsen, and O. V. Pictet. From the bird's eye to the microscope: A survey of new stylized facts of the intra-daily foreign exchange markets. *Finance and stochastics*, 1(2): 95–129, 1997.

N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N.-C. Yen, C. C. Tung, and H. H. Liu. The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London. Series A: mathematical, physical and engineering sciences*, 454 (1971):903–995, 1998.

J. C. Hull. *Options futures and other derivatives.* Pearson Education India, 2003.

R. Hyndman and G. Athanasopoulos. *Forecasting: Principle and Practice, 3rd edition.* OTexts: Melbourne, Australia, 2021. URL `OTexts.com/fpp3`. Accessed in June 2022.

R. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder. *Forecasting with exponential smoothing: the state space approach.* Springer Science & Business Media, 2008a.

R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

R. J. Hyndman, M. Akram, and B. C. Archibald. The admissible parameter space for exponential smoothing models. *Annals of the Institute of Statistical Mathematics*, 60(2):407–426, 2008b.

J. Hämäläinen and T. Kärkkäinen. Problem transformation methods with distance-based learning for multi-target regression. 10 2020.

S. M. Kay and S. L. Marple. Spectrum analysis—a modern perspective. *Proceedings* *of the IEEE*, 69(11):1380–1419, 1981.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. J. Koning, P. H. Franses, M. Hibon, and H. Stekler. The m3 competition: Statistical tests of the results. *International Journal of Forecasting*, 21(3): 397–409, 2005. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2004.10.003. URL `https://www.sciencedirect.com/science/article/pii/S0169207004000810`.

S. Makridakis and M. Hibon. The m3-competition: results, conclusions and implications. *International journal of forecasting*, 16(4):451–476, 2000.

S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2019.04.014. URL `https://www.sciencedirect.com/science/article/pii/S0169207019301128`. M4 Competition.

E. Mayerhofer. Three essays on stopping. *Risks*, 7(4):105, 2019.

A.-H. Mihov, N. Firoozye, and P. Treleaven. Towards augmented financial intelligence. *Available at SSRN*, 2022.

B. Monsell and C. Blakely. X-13arima-seats and imetrica. *US Census Bureau, Washington, DC*, 2013.

J. Osborne. Improving your data transformations: Applying the box-cox transformation. *Practical Assessment, Research, and Evaluation*, 15(1):12, 2010.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

D. B. Percival and A. T. Walden. *Wavelet methods for time series analysis*, volume 4. Cambridge university press, 2000.

V. Petrov, A. Golub, and R. B. Olsen. Agent-based model in directional-change intrinsic time. *Available at SSRN 3240456*, 2018.

V. Petrov, A. Golub, and R. Olsen. Instantaneous volatility seasonality of high-frequency markets in directional-change intrinsic time. *Journal of Risk and Financial Management*, 12(2):54, 2019a.

V. Petrov, A. Golub, and R. B. Olsen. Intrinsic time directional-change methodology in higher dimensions. *Available at SSRN 3440628*, 2019b.

M. J. Shensa et al. The discrete wavelet transform: wedding the a trous and mallat algorithms. *IEEE Transactions on signal processing*, 40(10):2464–2482, 1992.

E. Tsang. Directional changes, definitions. *Working Paper WP050-10 Centre for Computational Finance and Economic Agents (CCFEA), University of Essex Revised 1, Tech. Rep.*, 2010.

E. P. Tsang, R. Tao, and S. Ma. Profiling financial market dynamics under directional changes. *Quantitative finance, http://www. tandfonline. com/doi/abs/10.1080/14697688.2016*, 1164887, 2015.

E. P. Tsang, R. Tao, A. Serguieva, and S. Ma. Profiling high-frequency equity price movements in directional changes. *Quantitative finance*, 17(2):217–225, 2017.

X. Wang, K. Smith, and R. Hyndman. Characteristic-based clustering for time series data. *Data mining and knowledge Discovery*, 13(3):335–364, 2006.

H. White. Maximum likelihood estimation of misspecified models. *Econometrica: Journal of the econometric society*, pages 1–25, 1982.

I.-K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 12 2000. ISSN 0006-3444. doi: 10.1093/biomet/87.4.954. URL `https://doi.org/10.1093/biomet/87.4.954`.

Appendix A

Hyperparameter Space

In this chapter, we present the hyperparameter space we search for the models we trained in our experiment in Table A.1 and A.2. Note that for tasks in different frequencies, the hyperparameter space we search are different. The main difference is in the number of lags l. We have such arrangements because time series with different frequencies essentially operates in different space. The forecasting tasks are thus differentiated by the frequencies as well. We do not have a large hyperparameter space for the models due to our limited amount of time for the experiment. It indeed is possible that the hyperparameter space is not large or fine enough such that the models can find their optimal policies. However, our experiment is about distinguishing whether DC Transformation works. So as long as the comparison is fair, our analyses should be robust.

	Hyperparameter space	Total number of policies
EN	l: $\{6, 12, 24\}$ α: $\{0.1, 1, 10, 100\}$ ρ: $\{0.1, 0.5, 0.9\}$	$3 \times 4 \times 3 = 36$
MLP	l: $\{6, 12, 24\}$ $strucs$: $\{(12), (24), (12, 12), (24, 12)\}$ $maxiter$: $\{500\}$	$3 \times 4 \times 1 = 12$
LSVR	l: $\{6, 12, 24\}$ tol: $\{0.001, 0.01\}$ C: $\{0.1, 1\}$	$3 \times 2 \times 2 = 12$
RF	l: $\{6, 12, 24\}$ $minsamplesplit$: $\{0.005, 0.01\}$	$3 \times 2 = 6$
ETS	$seasonal$: $\{\text{additive}, \text{multiplicative}\}$ $trend$: $\{\text{additive}, \text{multiplicative}\}$ $damped$: $\{\text{True}, \text{False}\}$	$2 \times 2 \times 2 = 8$

Table A.1: Hyperparameter space per model for hourly frequency

	Hyperparameter space	Total number of policies
EN	l: $\{3, 7, 14\}$ α: $\{0.1, 1, 10, 100\}$ ρ: $\{0.1, 0.5, 0.9\}$	$3 \times 4 \times 3 = 36$
MLP	l: $\{3, 7, 14\}$ $strucs$: $\{(6,), (12,), (6, 6,), (12, 6,)\}$ $maxiter$: $\{500\}$	$3 \times 4 \times 1 = 12$
LSVR	l: $\{3, 7, 14\}$ tol: $\{0.001, 0.01\}$ C: $\{0.1, 1\}$	$3 \times 2 \times 2 = 12$
RF	l: $\{3, 7, 14\}$ $minsamplesplit$: $\{0.005, 0.01\}$	$3 \times 2 = 6$
ETS	$seasonal$: $\{\text{additive}, \text{multiplicative}\}$ $trend$: $\{\text{additive}, \text{multiplicative}\}$ $damped$: $\{\text{True}, \text{False}\}$	$2 \times 2 \times 2 = 8$

Table A.2: Hyperparameter space per model for daily and weekly frequencies

Appendix B

Programme

In this chapter, we briefly talk about the programme we create for this report. This chapter acts as a high level `README` file for our programme. The whole project can be found using the link to our GitHub repository `https://github.com/LinusLin6769/DCLaboratory01`.

B.1 How the programme works

The usage of the programme goes as the following:

1. Set how the experiment is intended to be done using the `config.json` file.
2. Run the experiment, which will generate experimental data.
3. Investigate the generated data.

B.2 Experiment configurations

The things that are controlled by the `config.json` file include the following:

1. Whether this run is for real, or it is just a test of something during development.
2. Which dataset to use and which time series to run on.
3. Transformation configuration: how the transformation will be performed and its hyperparameter space.

4. Modelling configuration: how the modelling process is done, e.g., validation and test size, retrain window size.

5. What are the models to be included in the experiment.

6. How many CPU cores to use for the implemented multi-processing.

B.3 Running the experiment

The experiment runs by calling

```
python3 main.py
```

from the command-line tool. `main.py` calls the `config.json` file, where the user controls how the experiment should go at the beginning, and the output is a directory created under the `experiment_info/` or `test_experiment_info/` directory depending on whether this is a real run or a test during development. The created directory is named after the starting time of the experiment. For every model trained, all the series are used. A `.JSON` file containing the experiment information of this model is created and placed in the experiment's directory before the programme moves on to another model until all models are trained.

B.4 Analysing the results

To analyse the experiment, one can use a Jupyter Notebook or any other tools to read the data generated by the experiment.

B.5 DC Transformation programme

As the core of this experiment, the programme of Directional Change Transformation is implemented as an independent class called `DCTransformer` in `dc_transformation.py` file. It is easy to use for those interested, and demonstrations can be found at the bottom of `dc_transformation.py`.