DIPLOMARBEIT

Titel

Zur Erlangung des akademischen Grades

Diplom-Ingenieur

Im Rahmen des Masterstudiums

Wirtschaftsmathematik und Statistik

Ausgeführt am

Institut für Stochastik und Wirtschaftsmathematik
Fakultät für Mathematik und Geoinformation
Technische Universität Wien

Unter der Anleitung von

Univ.Prof. Dipl.-Ing. Dr.techn. Peter Filzmoser

Eingereicht von

Alexander Schwaiger, BSc
Matrikelnummer: 11775205

Wien, am 1st May 2023

Alexander Schwaiger (Verfasser) Peter Filzmoser (Betreuer)

Contents

1. Introduction

1.1. Motivation

Multivariate count data is a reoccurring theme in real world applications. While there exist various methods among the classical statistical models to handle such data, there exist less methods to handle it in a time series context. Even more so, when there is an excessive amount of zeros or missing values present. In this thesis we compare various models for such data and compare their predictive power. We test our models on real world data which was kindly provided to us. In the following we will shortly describe the general framework and objective.

A company is operating numerous vending machines with food, ranging from appetizers, main course, snacks and beverages. Each week the vending machines, or in the following also called fridges, are being restocked and the number of items sold in this week is being recorded. In addition, non-sold items are being disposed off which result in monetary losses. The objective is to find a model with a view to predicting the amount they need to order in a bid to minimise the loss.

1.2. Data Description

In this section we describe the structure of our data which is essential in choosing the right model. We have several multivariate time series with integer values, with each series representing a vending machine. The dimensions represent the various categories of the food. Each item is of one of the four main categories 1,2,3,4 and one of the various subcategories. We mainly analyse the time series on the aggregated level of the main categories, however the models can also be applied to the subcategories. In this case we have a model for each main category instead of each vending machine. The values for each category represent the number of items sold. For a fridge f denote this time series with

$$\left\{\boldsymbol{Y}_t : t \in \mathbb{N}, \boldsymbol{Y}_t \in \mathbb{N}_0^K\right\}_f \tag{1.1}$$

where K stands for the number of categories and $\mathbb{N}_0^K := \underbrace{\mathbb{N}_0 \times \mathbb{N}_0}_{K-times}$. The data is measured weekly and hence our points in time are equidistant. A special feature of our data is the amount of 0 and NA values. How they are handled is explained in later sections. Another characteristic of our data is the difference in length for various time series.

While for some time series we have 70+ data points, for others we have less than 10. An example view of our data would be:

Fridge ID	Week Date	Main Category	Sub Category	Sold
111	2021-01-18	1	3	6
111	2021-01-18	1	8	7
111	2021-01-25	2	6	4
222	2022-06-06	3	15	1
222	2022-06-06	4	11	0
222	2022-06-13	1	100061	0
222	2022-06-20	2	6	30
222	2022-06-20	2	10	15

Table 1.1.: Example Data

As mentioned before, we mainly aggregate our data on main category level. This means that we do not differentiate between the subcategories and are only interested in the number of items sold for each main category. Our data in 1.1 would then change to 1.2:

Fridge ID	Week Date	Main Category	Sold
111	2021-01-18	1	13
111	2021-01-25	2	4
222	2022-06-06	3	1
222	2022-06-06	4	0
222	2022-06-13	1	0
222	2022-06-20	2	45

Table 1.2.: Example Data aggregated on Main Category level

1.3. Outlook

The remainder of the thesis is split in the following way. In chapter 2 we describe our methodologies used and the reasoning why we are using them. We provide a short literature review about count data time series in 2.1.1. In these sections we also lay the mathematical groundwork for both of those methods. In section 2.3 we shortly describe other methods considered. In chapter 3 we explain the specification and tuning options for our models and also introduce an error measure to evaluate their performance. We show the results on some exemplary time series and then show the results of each tuning parameter. In the conclusion 4 we summarise our findings and provide a further outlook on the topic.

2. Methodology

2.1. INGARCH

In this section we introduce the INGARCH(p,q) model. First we provide a motivation on why we chose this model and review some other possible models for discrete time series count data. The review is mainly based on [Lib16] and [Hei03]. A more detailed review can be found in [MMM97]. Subsequently we define the INGARCH(p,q) model itself and list some of its properties.

2.1.1. Motivation

This section is based on [Lib16] and [Hei03].
Since our data can be seen as a discrete time series with count data, we want a model which is able to take these properties into account. In addition, autocorrelation and overdispersion are two common features in count data. One common way to deal with count data are Markov chains. The dependent variable can take on all possible values in the so called state space and the probability of changing states is then modelled as a transition probability. A limitation is the fact that these models become cumbersome if the state space gets too big and lose tractability. As an extension to the basic Markov chains models, Hidden Markov chains are proposed by [MMM97]. However, since there is no generally accepted way to determine the order of this model, it can cause problems if the data structure does not provide intuitive ways to do it. Another issue is that the number of parameters which need to be estimated gets big quickly, especially if the order of the model is big.

Other common models for time series data are the ARMA models. There exists a discrete version of them in the form of the Discrete Autoregressive Moving Average (DARMA) models. They can be defined as a mixture of discrete probability distributions and a suitable chosen marginal probability function [BS09]. While there have been various applications, for example in [CDK87], there seem to be difficulties in their estimation [Hei03].

State space models with conjugated priors are proposed by [HF89]. The observations are assumed to be drawn from a Poisson distribution whose mean itself follows a Gamma distribution. The parameters of the Gamma distribution are chosen in such a way that its mean is constant but its variance is increasing. While there are ways proposed by [Zeg88] to handle overdispersion, these models have the weakness of needing further assumptions to handle zeros while also having more complicated model specifications [Hei03].

While there are many more possible models, we decided to focus on the class of Generalised Linear Models (GLM). In the case of discrete time series with count data, the observations are modelled conditionally on the past and follow a discrete distribution. The conditional mean is then connected with a link function to the past observations and conditional means. Furthermore, a covariate vector can be introduced to account for external influence. While being easy to use and estimate they still provide a good amount of flexibility. In addition, a wide array of tools is available for various tests and forecasts. From the class of the GLMs we compare the INGARCH(p,q) and a log-linear model, which will be discussed in section 2.3. We then chose the INGARCH(p,q) model based on its superior performance and stability.

2.1.2. INGARCH Model

Take again our time series $\left\{\boldsymbol{Y}_t : t \in \mathbb{N}, \boldsymbol{Y}_t \in \mathbb{N}_0^K\right\}_f$ for fridge f and denote the univariate time series for category k with $\{Y_{k_t} : t \in \mathbb{N}, Y_{k_t} \in \mathbb{N}_0\}_f$ for $k = 1, \ldots, K$. Denote a r-dimensional time varying covariate vector with $\mathbf{X}_t = (X_{t,1}, \ldots, X_{t,r})^T$. Let the conditional mean be $\lambda_{k_t} = \mathbb{E}[Y_{k_t}|\mathcal{F}_{t-1}]$ where $\mathcal{F}_{t-1}$ is the sigma-field generated by Y_{k_t} and λ_l for $l < t$, $\mathcal{F}_{t-1} = \sigma(Y_{k_1}, \ldots, Y_{k_l}, \lambda_1, \ldots, \lambda_l)$. Therefore, the conditional mean of the time series is dependent on its combined history of the past conditional means and its past values. With this, we can define the integer valued generalized autoregressive conditional heteroskedasticity model of order (p,q) (INGARCH(p,q) model) as,

$$Y_{k_t}|\mathcal{F}_{t-1} \sim P(\lambda_{k_t}); \forall t \in \mathbb{N}, \tag{2.1}$$

$$\mathbb{E}[Y_{k_t}|\mathcal{F}_{t-1}] = \lambda_{k_t} = \beta_0 + \sum_{i=1}^{p} \beta_i Y_{k_{t-i}} + \sum_{j=1}^{q} \alpha_j \lambda_{k_{t-j}} \tag{2.2}$$

where $p, q \in \mathbb{N}$ and $P(\lambda_{k_t})$ is a Poisson distribution with mean λ_{k_t}. The integer p defines the number of past values to regress on, whereas q does the same for the past conditional means. In order to account for external effects as well, we add the covariate vector $\mathbf{X}_t$

$$Y_{k_t}|\mathcal{F}_{t-1} \sim P(\lambda_{k_t}); \forall t \in \mathbb{N}, \tag{2.3}$$

$$\mathbb{E}[Y_{k_t}|\mathcal{F}_{t-1}] = \lambda_{k_t} = \beta_0 + \sum_{i=1}^{p} \beta_i Y_{k_{t-i}} + \sum_{j=1}^{q} \alpha_j \lambda_{k_{t-j}} + \boldsymbol{\eta}^T \mathbf{X}_t \tag{2.4}$$

where $\boldsymbol{\eta}$ is the parameter for the covariates.
The distributional assumptions $Y_{k_t}|\mathcal{F}_{t-1} \sim P(\lambda_{k_t})$ implies

$$p_t(y; \boldsymbol{\theta}) = \mathbb{P}(Y_{k_t} = y|\mathcal{F}_{t-1}) = \frac{\lambda_{k_t}^y \exp(-\lambda_{k_t})}{y!}, \quad y \in \mathbb{N}_0. \tag{2.5}$$

Furthermore it can be shown that conditionally on the past history $\mathcal{F}_{t-1}$ the model is equidispersed, i.e. it holds $\lambda_{k_t} = \mathbb{E}[Y_{k_t}|\mathcal{F}_{t-1}] = \mathbb{V}[Y_{k_t}|\mathcal{F}_{t-1}]$. However, unconditionally the model exhibits overdispersion. In that case it holds $\mathbb{E}[Y_{k_t}] \leq \mathbb{V}[Y_{k_t}]$ [Hei03].

Estimation of the INGARCH Model

We summarise the estimation of the INGARCH(p,q) Model as described in [Lib16].

The parameter space for the INGARCH(p,q) model with external effects 2.3 is given by

$$\Theta = \left\{ \boldsymbol{\theta} \in \mathbb{R}^{p+q+r+1} : \beta_0 > 0, \beta_1, \ldots, \beta_p, \alpha_1, \ldots, \alpha_q, \eta_1, \ldots, \eta_r \geq 0, \sum_{i=1}^{p} \beta_i + \sum_{j=1}^{q} \alpha_j < 1 \right\}.$$

To ensure positivity of the conditional mean λ_{k_t}, the intercept β_0 must be positive while all other parameters must be non negative. The upper bound of the sum ensures that the model has a stationary and ergodic solution with moments of any order [FLO06; FRT09; DFT12]. A quasi maximum likelihood approach is used to estimate the parameters $\boldsymbol{\theta}$. For observations $\mathbf{y} = (y_1, \ldots, y_n)^T$ the conditional quasi log-likelihood function, up to a constant, is given by,

$$\ell(\boldsymbol{\theta}) = \sum_{t=1}^{n} \log p_t(y_t; \boldsymbol{\theta}) = \sum_{t=1}^{n} \left(y_t \log(\lambda_{k_t}(\boldsymbol{\theta})) - \lambda_{k_t}\theta \right). \tag{2.6}$$

where $p_t(y_t; \boldsymbol{\theta})$ is the probability density function defined in 2.5. The conditional mean is seen as a function $\lambda_{k_t} : \Theta \to \mathbb{R}^+$. The conditional score function is given by,

$$S_n(\boldsymbol{\theta}) = \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{t=1}^{n} \left(\frac{y_t}{\lambda_{k_t}(\boldsymbol{\theta})} - 1 \right) \frac{\partial \lambda_{k_t}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{2.7}$$

The vector $\frac{\partial \lambda_{k_t}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ can be computed recursively. The conditional information matrix is given by,

$$\begin{aligned} G_n(\theta) &= \sum_{t=1}^{n} Cov\left(\frac{\partial \ell(\boldsymbol{\theta}; Y_{k_t})}{\partial \boldsymbol{\theta}} \middle| \mathcal{F}_{t-1} \right) \\ &= \sum_{t=1}^{n} \left(\frac{1}{\lambda_{k_t}(\boldsymbol{\theta})} \right) \left(\frac{\partial \lambda_{k_t}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right) \left(\frac{\partial \lambda_{k_t}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right)^T. \end{aligned}$$

Finally, assuming that the quasi maximum likelihood estimator (QMLE) $\hat{\boldsymbol{\theta}}_n$ of $\boldsymbol{\theta}$ exists, it is the solution to

$$\hat{\boldsymbol{\theta}} := \hat{\boldsymbol{\theta}}_n = \arg\max_{\boldsymbol{\theta} \in \Theta}(\ell(\boldsymbol{\theta})). \tag{2.8}$$

2.2. CoDA

2.2.1. Motivation

One way to see our data is as a compositional time series. The exact definition of compositional will follow later but in general compositional data, which is by nature

multivariate, describes relations between the parts instead of absolute values. We transform the data in such a way, that the values of each category can be seen as the relative share of the total amount at the current time. We then predict the relative share of the category for the next point in time. Since we are ultimately interested in the absolute value, we include the total sum of all categories as an additional variable and then use it for calculating the absolute value. This is modelled as the so-called $\mathcal{T}$-Space which will be introduced later. Since VAR models are easy to estimate and interpret and have some beneficial properties with our choice of transformation, we opt to focus on them. One such property is the fact that the VAR model does not depend on the concrete choice of the ilr-transformation [KFH15].

2.2.2. Preliminaries

The basis of this section is given by [KFH15], [Ego+03] and [FH20].

CoDA, which is short for "'Compositional Data Analysis"' works with compositional data. The key to compositional data is the fact that the absolute value of its parts is less important than the relative relation of the parts to each other. To define compositional data, we first need to define the $(D-1)$-dimensional simplex,

$$\mathbb{S}^D := \left\{ (x_1, \ldots, x_D)^T : x_i > 0, i = 1, \ldots, D; \sum_{i=1}^{D} x_i = \kappa \right\}$$

where κ is a positive constant [KFH15]. The choice of κ is not relevant, as the relative information in the compositional parts stays the same. A D-dimensional vector $\mathbf{x} = (x_1, \ldots, x_D)^T$ is said to be compositional if it is part of $\mathbb{S}^D$. Next we can induce a $(D-1)$-dimensional vector space on $\mathbb{S}^D$ by perturbation and power transformation. For compositions $\mathbf{x}, \mathbf{z} \in \mathbb{S}^D$ and $a \in \mathbb{R}$ they are defined respectively as [KFH15]

$$\mathbf{x} \oplus \mathbf{z} := \mathcal{C}(x_1 z_1, x_2 z_2, \ldots, x_D z_D)^T, \quad a \odot \mathbf{x} := \mathcal{C}(x_1^a, x_2^a, \ldots, x_D^a)^T.$$

Here $\mathcal{C}$ is the closure operation that maps each compositional vector from the real value space $\mathbb{R}_+^D$ into its representation in $\mathbb{S}^D$

$$\mathcal{C}(\boldsymbol{x}) := \left(\frac{\kappa x_1}{\sum_{i=1}^{D} x_i}, \ldots, \frac{\kappa x_D}{\sum_{i=1}^{D} x_i} \right)^T.$$

Using $z^{-1} := \mathcal{C}(z_1^{-1}, z_2^{-1}, \ldots, z_D^{-1})$, the inverse perturbation can be defined as

$$\mathbf{x} \ominus \mathbf{z} := \mathbf{x} \oplus \mathbf{z}^{-1}$$

Now we further define an inner product in order to have an inner product space over the simplex $\mathbb{S}^D$. For two compositions $\mathbf{x}, \mathbf{z} \in \mathbb{S}^D$ define the Aitchison inner product as

$$\langle \mathbf{x}, \mathbf{z} \rangle_a := \frac{1}{2D} \sum_{i=1}^{D} \sum_{j=1}^{D} \log(\frac{x_i}{x_j}) \log(\frac{z_i}{z_j}).$$

In addition, a norm and distance measure can be defined

$$\|\mathbf{x}\|_a^2 := \langle \mathbf{x}, \mathbf{x} \rangle_a , \quad d_a(\mathbf{x}, \mathbf{z}) := \|\mathbf{x} \ominus \mathbf{z}\|_a .$$

This induced geometry is called the Aitchison geometry and it allows us to express a composition $\mathbf{x} \in \mathbb{S}^D$ as a perturbation-linear combination of a basis of $\mathbb{S}^D$.

However, in order to use standard statistical tools, it is desirable to move from this geometry to the Euclidean real space [FH20]. There are various ways to map the data from the simplex $\mathbb{S}^D$ to the real space $\mathbb{R}^D$. A review of the most common transformations is provided in the following section.

2.2.3. Common Transformations

Let $\mathbf{x}, \mathbf{z} \in \mathbb{S}^D$ be D-part compositions.

alr Coordinates

The additive log-ratio (alr) Coordinates are defined as

$$\mathbf{z}^{(k)} = alr_k(\mathbf{x}) := \left(\log\left(\frac{x_1}{x_k}\right), \ldots, \log\left(\frac{x_{k-1}}{x_k}\right), \log\left(\frac{x_{k+1}}{x_k}\right), \ldots, \log\left(\frac{x_D}{x_k}\right) \right).$$

and map the composition $\mathbf{x}$ to the real space $\mathbb{R}^D$. They are mainly mentioned for historic purposes since they are an intuitive way of transformation. However, limitations are posed by their dependence on the choice of the denominator x_k and the fact that they are not orthogonal to each other [FH20].

clr Coefficients

Let $g(\mathbf{x})$ be the geometric mean of $\mathbf{x}$. The centered log-ratio coefficients are then defined as

$$\mathbf{w} = (w_1, \ldots, w_D)^T = clr(\mathbf{x}) := \left(\log\left(\frac{x_1}{g(\mathbf{x})}\right), \ldots, \log\left(\frac{x_D}{g(\mathbf{x})}\right) \right)^T .$$

This transformation maps $\mathbf{x}$ into the hyperplane $V = \left\{ \mathbf{w} \in \mathbb{R}^D : \sum_{i=1}^D w_i = 0 \right\} \subset \mathbb{R}^D$. Hence the transformed data is constrained, which is emphasised by the term 'coefficient' instead of 'coordinates' [FH20]. It can be shown that the clr transformation is an isometry[Ego+03]. Therefore it holds

$$\begin{aligned} \langle \mathbf{x}, \mathbf{z} \rangle_a &= \langle clr(\mathbf{x}), clr(\mathbf{z}) \rangle_a , \\ d(\mathbf{x}, \mathbf{z})_a &= d(clr(\mathbf{x}), clr(\mathbf{z})). \end{aligned}$$

ilr Coordinates

The isometric log-ratio (ilr) are closely related to the clr Coefficients. Assume the inverse clr transformation is isometric. Let $\{v_1, \dots, v_n\}$ be an orthonormal base in in the D-dimensional hyperplane V. Then $\mathbf{e}_i = clr^{-1}(v_i), i = 1, \dots, D-1$ is an orthonormal basis of the simplex $\mathbb{S}$. For $\mathbf{x} \in \mathbb{S}$, the ilr transformation can then be defined as

$$\mathbf{u} = ilr(\mathbf{x}) = \left(\langle \mathbf{x}, \mathbf{e}_1 \rangle_a, \dots, \langle \mathbf{x}, \mathbf{e}_{D-1} \rangle_a\right)^T.$$

In addition to being isometric, the ilr transformation is also isomorph. Let $\mathbf{x}, \mathbf{z}$ be two compositions and $a, b \in \mathbb{R}$. Then,

$$ilr(a \odot \mathbf{x} \oplus b \odot \mathbf{z}) = a \cdot ilr(\mathbf{x}) + b \cdot ilr(\mathbf{z})$$

as well as,

$$\begin{aligned} \langle \mathbf{x}, \mathbf{z} \rangle_a &= \langle ilr(\mathbf{x}), ilr(\mathbf{z}) \rangle_a, \\ d(\mathbf{x}, \mathbf{z})_a &= d(ilr(\mathbf{x}), ilr(\mathbf{z})), \\ \|x\|_a &= \|ilr(x)\| = \|u\|. \end{aligned}$$

From the definition of the ilr coordinates it can be seen that they can be expressed as a linear combination of the basis induced by the clr coefficients as seen above. Let $\mathbf{V}$ be a $D \times (D-1)$ matrix with columns $\mathbf{v}_i = clr(\mathbf{e}_i)$. For a composition $\mathbf{x}$ the vector of ilr coordinates associated with $\mathbf{V}$ is given by,

$$\mathbf{u}_\mathbf{V} = ilr_\mathbf{V}(\mathbf{x}) = \mathbf{V}^T clr(\mathbf{x}) = \mathbf{V}^T \log(\mathbf{x}).$$

The matrix $\mathbf{V}$ is the contrast matrix with the orthonormal basis $(\mathbf{e}_i)_{i=1}^{D-1}$ [Ego+03]. A special choice of orthogonal coordinates leads to the coordinates

$$\begin{aligned} ilr(\mathbf{x}) &= (u_1, \dots, u_{D-1})^T, \\ u_j &= \sqrt{\frac{D-j}{D-j+1}} \log\left(\frac{x_j}{\sqrt[D-j]{\prod_{l=j+1}^{D} x_l}}\right), \quad j = 1, \dots, D-1. \end{aligned}$$

With this choice, the problem of interpretation, which arises from the relative nature of the compositional data and the dimension of the simplex, can be solved. The part x_1 is only contained in z_1 and therefore contains all relative information of x_1 [FH20].

To transform the data back in the simplex, the inverse transformation is given by,

$$x_1 = \exp\left(\sqrt{\frac{D-1}{D}}u_1\right),$$
$$x_i = \exp\left(\sum_{j=1}^{i-1}\frac{1}{\sqrt{(D-j+1)(D-j)}}u_j + \sqrt{\frac{D-i}{D-i+1}}u_i\right),\ \ i = 2,\ldots,D-1,$$
$$x_D = \exp\left(-\sum_{j=1}^{D-1}\frac{1}{\sqrt{(D-j+1)(D-j)}}u_j\right).$$

2.2.4. The VAR Model

Since we have established the basic setting we can now introduce compositional time series (CTS). A CTS $\{\boldsymbol{x}_t : t = 1,\ldots,n\}$ can be defined as a series where $\boldsymbol{x}_t = (x_{1t},\ldots,x_{Dt})^T \in \mathbb{S}^D$. They are thus characterised by their positive components which sum up to a constant κ_t for each point in time $t = 1,\ldots,n$

$$\sum_{i=1}^{D} x_{it} = \kappa_t,\ \ x_i > 0, i = 1,\ldots D\,; t = 1,\ldots,n.$$

Let $\left\{\boldsymbol{Y}_t : t \in \mathbb{N}, \boldsymbol{Y}_t \in \mathbb{N}_0^K\right\}_f$ be our time series for fridge f and assume that $\boldsymbol{Y}_t$ is a D-dimensional compositional vector measured at time $t, t = 1,\ldots,n$. Further, let $\mathbf{u}_t = ilr(\boldsymbol{Y}_t)$ be its ilr transformation determined by the matrix $\mathbf{V}$. Then the VAR model with lag order p is given by [KFH15]

$$\mathbf{u}_t = \mathbf{c_V} + \mathbf{A}_\mathbf{V}^{(1)}\mathbf{u}_{t-1} + \mathbf{A}_\mathbf{V}^{(2)}\mathbf{u}_{t-2} + \ldots + \mathbf{A}_\mathbf{V}^{(p)}\mathbf{u}_{t-p} + \boldsymbol{\epsilon}_t. \tag{2.9}$$

where $\mathbf{c_V} \in \mathbb{R}^{D-1}$ is a real vector, $\mathbf{A}_\mathbf{V}^{(i)} \in \mathbb{R}^{(D-1)\times(D-1)}$ are parameter matrices and $\boldsymbol{\epsilon}_t$ is a white noise process with covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$. The observation $\mathbf{u}_t$ therefore depends on the p past observations $\mathbf{u}_{t-1},\ldots,\mathbf{u}_{t-p}$. It can be shown, that two VAR(p) models resulting from different ilr transformations are compositionally equivalent which means that the same predictions are obtained [KFH15].

Estimation of the VAR Model

Assuming n observations are used for the model, equation 2.9 can be written in matrix form as

$$\mathbf{U} = \mathbf{ZB} + \mathbf{E},$$
$$\mathbf{U} = (\mathbf{u}_1,\ldots,\mathbf{u}_n)^T \in \mathbb{R}^{n\times(D-1)},$$
$$\mathbf{Z} \in \mathbb{R}^{n\times[(D-1)p+1]} \text{ with } \mathbf{Z}_t = \left(1, \mathbf{u}_{t-1}^T,\ldots,\mathbf{u}_{t-p}^T\right)^T,$$
$$\mathbf{B} = \left[\mathbf{c}, \mathbf{A}^{(1)},\ldots,\mathbf{A}^{(p)}\right]^T \in \mathbb{R}^{(D-1)p+1\times(D-1)}.$$

The parameter $\mathbf{B}$ can then be estimated separately for each column of $\mathbf{U}$ by the ordinary least squares (OLS) method. In addition, if there are no restrictions posed on the parameter, the estimator is equal to the generalised least squares (GLS). If the VAR(p) process is normally distributed and the rows of the error matrix $\mathbf{E}$ represent a white noise process, thus $\boldsymbol{E} \sim WN(\Sigma)$ where Σ is the covariance matrix, then the estimator is also equal to the maximum likelihood (ML) estimator. Under these assumptions it can be shown that the OLS estimator is consistent and asymptotic normal [KFH15] [Lüt07].

2.2.5. $\mathcal{T}$-Spaces

Since in our context absolute values should be predicted eventually, we model this information in form of the sum of the original parts. Let $\mathbf{x}$ be a D-dimensional compositional vector and define an extended vector space $\mathcal{T} = \mathbb{R}_+ \times \mathbb{S}^D$. An element of $\mathcal{T}$ now has the form $\tilde{\boldsymbol{x}} = [t(\boldsymbol{x}), \mathcal{C}(\boldsymbol{x})] = [t_x, \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_D]$ where $t(\boldsymbol{x}) = \sum_{i=1}^{D} x_i$. This allows us to model the relative structure of the data as well as the total sum of the compositional parts. Often times the logarithm of the sum is taken. In the subsequent analysis, the compositional part $\boldsymbol{x}$ is then transformed using one of the possible transformations and then the statistical analysis is performed. In our case, back transformation of the results are required as well as scaling the proportional values back to the absolute values to get a final result. [KFH15]

2.3. Other Methods

2.3.1. Naive Random Walk

This method acts as kind of benchmark model and is what is currently used for forecasting. Let $\{Y_{k_t} : t \in \mathbb{N}, Y_{k_t} \in \mathbb{N}_0\}_f$ be again the univariate time series for category k for $k = 1, \ldots, K$ and fridge f. Then

$$\hat{Y}_{k_{t+1}} = Y_{k_t}, \quad \forall t \in \mathbb{N}, k = 1, \ldots, K \tag{2.10}$$

where $\hat{Y}_{k_{t+1}}$ is the predicted value at time t. In other words, the last known value is the predicted value.

2.3.2. Zero-Inflated Models

Since we encounter a large number of zeros we also consider zero-inflated models. The idea of zero-inflated models is to add a degenerated distribution with mass at zero to the probability mass function. This way we can explain the large amount of zero values which would not be expected in a normal Poisson distribution. The zero-inflated INGARCH(p,q) model is equivalent to 2.1 and only the conditional distribution of $Y_{k_t} = y|\mathcal{F}_{t-1}$ changes. Instead of 2.5 it is now

$$\mathbb{P}(Y_{k_t} = y|\mathcal{F}_{t-1}) = \omega\delta_{y,0} + (1-\omega)\frac{\lambda_t^y \exp(-\lambda_t)}{y!}, \quad y \in \mathbb{N}_0. \tag{2.11}$$

where $0 < \omega < 1$ and $\delta_{y,0}$ is the Kronecker delta for which $\delta_{y,0} = 1$ if $y = 0$ and $\delta_{y,0} = 0$ else. This way our zeros can come from two different sources. If $\omega = 0$ then we get the normal INGARCH(p,q) model discussed above. More details about zero-inflated models and especially zero-inflated INGARCH(p,q) model can be found in [Zhu12].

2.3.3. Log-Linear Models

As mentioned in 2.1.1 we also investigate log-linear models. These models are structurally very similar to the normal INGARCH(p,q) model only with a logarithmic link function. It has the form

$$\nu_t = \log(\lambda_t) = \beta_0 + \sum_{i=1}^{p} \beta_i \log(Y_{k_{t-i}} + 1) + \sum_{j=1}^{q} \alpha_j \nu_{t-j}. \tag{2.12}$$

The past values get transformed by $h(x) = \log(x + 1)$ to get them on the same scale as ν_t and avoid zero values in the logarithm [Lib16; FT11]. In comparison to the INGARCH(p,q) model, the log-linear model also supports modeling of negative serial correlation [Lib16]. It is further discussed in [FT11; WMH11; DDM13].

2.3.4. Vector Generalised Additive Models

Because we work with multivariate count data, we also look at vector generalised additive models (VGAMs) which extend generalised additive models (GAMs) to higher dimensions. GAMs allow us to reveal and model non-linear relationship in our data, as opposed to linear models or generalised linear models [YW96]. Let y be a univariate response with a distribution in the exponential family and mean μ. Further take a p-dimensional covariate vector and $\boldsymbol{x} = (x_1, \ldots, x_p)^T$. Then the generalised additive model (GAM) is given by

$$g(\mu) = \nu(\boldsymbol{x}) = \beta_0 + f_1(x_1) + \ldots f_p(x_p) \tag{2.13}$$

with f_j being arbitrary smooth functions [YW96]. To extend this model to the multivariate case, we replace the functions f_j with vector functions. Let $\boldsymbol{f}_k(Y_{k_t}) = (f_{(1)k}(Y_{k_t}), \ldots, f_{(M)k}(Y_{k_t}))^T$ with $M \in \mathbb{N}$ be an arbitrary smooth vector function. Then the vector generalised additive model is given by

$$\mathbb{E}[\boldsymbol{Y}_t] = \boldsymbol{\beta}_0 + \sum_{k=1}^{K} \boldsymbol{f}_k(Y_{k_{t-1}}) \tag{2.14}$$

where $\mathbb{E}[\boldsymbol{Y}_t] = (\mathbb{E}[Y_{1_t}], \ldots, \mathbb{E}[Y_{K_t}])^T$ [YW96]. Further theoretical background about VGAMs are given in [YW96; Yee15; Woo04].

3. Application

3.1. Model Specifications

As our data has a specific structure, some transformations can be made to increase performance and stability. The most prominent characteristic of our data is its amount of 0 or null values. As both CoDA can't handle an excessive amount of 0 values, we have to accommodate for this. The concrete way to do this will be described in the following subsections.

Another varying factor is the history. We define as history the length of the time series for fridge f and denote it with T_f. While at first it may seem obvious to use as much data as possible, it may actually not always result in a better model. Older values may contain outdated information which influences the estimation the of parameters. Therefore we compare the performance of the models with various history lengths.

Closely related to the length of the history, is the shape of the window used. The window determines which values are used to estimate the parameters at each point in time. The shape includes both the initial length of it and the way it handles new values. As the different time series vary in length, we choose the possible window length as a fraction of the time series history. Let w_f be the initial window length. For the way how new values are handled, we focus on two different approaches. The first one uses a fixed window length. This means when a new time point is available, it will be included in the estimation while simultaneously the oldest time point will be removed from the estimation. This has the advantage of only using the most recent and relevant information. The second approach, extends the window at each point in time. When a new value is available, it is included in the estimation of the parameter. With this approach we have more data available at each step and combined with the varying history length we don't have to rely on information that is too old.

3.1.1. CoDA Specifications

As mentioned above the CoDA model must not include any zero values. Since in the CoDA context we see our data as relative data, a value of zero is not defined. Therefore we need to replace them. In order to keep things simple, we consider two options. The first one adds 0.5 to all time series values. The second one only replaces zero values with 0.5.

As already hinted in the description of the methodology we consider the use of $\mathcal{T}$-Spaces. For this, at each time point, we calculate the total amount and include it as an additional variable in the model. In addition we can choose to take the logarithm of the

sum. This means $\boldsymbol{u}_t$ in model 2.9 changes to

$$\boldsymbol{u}_t = [ilr(\boldsymbol{Y}_t), t(\boldsymbol{Y}_t)]$$

with $t(\boldsymbol{Y}_t) = \sum_{k=1}^{K} Y_{k_t}$ or $t(\boldsymbol{Y}_t) = \log\left(\sum_{k=1}^{K} Y_{k_t}\right)$.

Another characteristic of our data are the low values for some categories of it. Even at the aggregated main category level there are instances with low values for some of the categories. This is the case especially for category 3 and 4. As such, we inspect a method which we will call in the following one-vs-all. The principle is the following. A category k is chosen as the pivot category k_{pivot}. For all the chosen time points, at each point, the values of the other categories get summed up

$$Y_{other_t} = \sum_{\substack{k=0 \\ k \neq k_{pivot}}}^{K} Y_{k_t}.$$

Together with the pivot category, the sum of the other categories are then transformed as usual and the VAR model is calculated

$$\boldsymbol{u}_t = ilr([Y_{other_t}, Y_{k_{pivot_t}}]).$$

All categories are chosen as a pivot category at one point and the predicted values of the pivot groups are then used as the final result.

3.1.2. INGARCH Specifications

As an alternative to the Poisson distribution in 2.5, a negative binomial distribution can be used as well. This would change 2.5 to

$$p_t(y; \boldsymbol{\theta}) = \mathbb{P}(Y_{k_t} = y | \mathcal{F}_{t-1}) = \frac{\Gamma(\phi + y)}{\Gamma(y+1)\Gamma(\phi)} \left(\frac{\phi}{\phi + \lambda_t}\right)^{\phi} \left(\frac{\lambda_t}{\phi + \lambda_t}\right)^{y}, \quad y \in \mathbb{N}_0.$$

With the negative Binomial Distribution the conditional variance is larger than the conditional mean $\lambda_t = \mathbb{V}[Y_{k_t} | \mathcal{F}_{t-1}] > \mathbb{E}[Y_{k_t} | \mathcal{F}_{t-1}]$.

As seen in the model 2.3 we can also choose to include external factors or not. However, as our data is of the structure where we don't have information about $\boldsymbol{X}_t$ at time t, we cannot make use of it. The values p and q are also varying parameters which have to be chosen.

3.1.3. Error Measure

In order to compare the results of the methods with each other we will introduce a new error measure. The goal of this measure is to get a performance indicator for each fridge which can be used for comparison and summarisation. Since the scales of the fridges vary, the measure should be scale independent. Because our data contains many zeros,

we cannot use a percentage error measure. In addition we want to penalise big absolute difference between the predicted values and actual values. These requirements leads us to the following measure.

For a fridge f, let $t = 1, \ldots, n$ denote the point in time and $k = 1, \ldots, K$ the category. Then y_{ftk} is the t-th true value of the time series for category k, $\hat{y}_{ftk}$ the predicted value and $y_{naive_{ftk}}$ the naive predicted value . Then we define our measure as

$$E_f = \frac{\sum_{k=1}^{K} \sum_{t=1}^{n} (y_{ftk} - \hat{y}_{ftk})^2}{\sum_{k=1}^{K} \sum_{t=1}^{n} (y_{ftk} - y_{naive_{ftk}})^2}. \tag{3.1}$$

With the use of the squared difference we penalise big deviations from the true value. By taking the naive random walk model as a benchmark, we achieve scale independence and are able to compare the performance of our model over different time series. This error measure is basically the ration of the mean MSEs for the chosen model and the naive random walk model

$$E_f = \frac{\frac{1}{K} \sum_{k=1}^{K} MSE_{fk}}{\frac{1}{K} \sum_{k=1}^{K} MSE_{naive_{fk}}}. \tag{3.2}$$

If the ratio is below 1, the mean of the MSEs of our methods is lower than that of the naive method and vice versa. This provides a performance indicator for our models.

Extension of the Error Measure

The measure in 3.1 can be further extended. For example, by allowing to use a subset of all possible categories instead of all. Let $G_K \subset \{1, \ldots, K\}$ then

$$E_f^{GK} = \frac{\sum_{k \in G_K} \sum_{t=1}^{n} (y_{ftk} - \hat{y}_{ftk})^2}{\sum_{k \in G_K} \sum_{t=1}^{n} (y_{ftk} - y_{naive_{ftk}})^2}. \tag{3.3}$$

This allows us to compare the performance on the subset of categories over various fridges.

Another possible extension is to take the square root

$$\widetilde{E}_f = \frac{\sum_{k=1}^{K} \sqrt{\sum_{t=1}^{n} (y_{ftk} - \hat{y}_{ftk})^2}}{\sum_{k=1}^{K} \sqrt{\sum_{t=1}^{n} (y_{ftk} - y_{naive_{ftk}})^2}}. \tag{3.4}$$

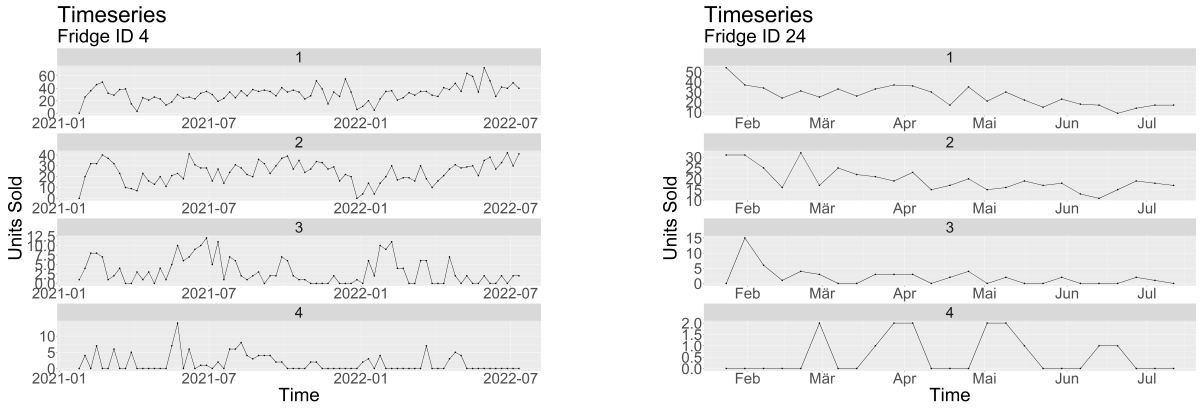
One future extension which can be investigated is the introduction of weights. This could be used for example when the performance of the model in one category should be put more into focus.

3.2. Examples of model application

To improve understanding of our data and the models we show some application of the models on some exemplary fridges. We choose fridges 4 and 24. Hence $f \in \{4, 24\}$.

Furthermore we start with analysing the aggregated 4 main categories which means $K = 4$.

We first begin with plotting the values of time series. The x-axis shows the time and the y-axis the number of units sold. Since we have four main categorie for each fridge, we have four subplots.



(a) Fridge 4 with all four main categories

(b) Fridge 24 with all four main categories

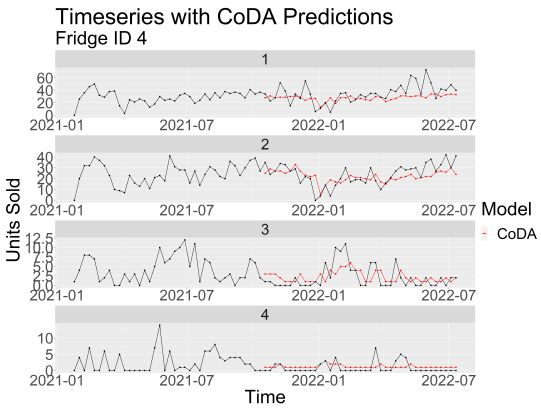
Figure 3.1.: Time series for two fridges

The two plots in 3.1 are good examples of the composition of our data. The scales of the sold units within a fridge vary widely. For example in figure 3.1b the values for category 1 vary from above 50 to as low as 10, while for category 4 we only have values in the range of 0 to 2. In both figures 3.1 for category 4, we can see the excessive amount of zero values in our data which makes the previously mentioned transformations necessary.

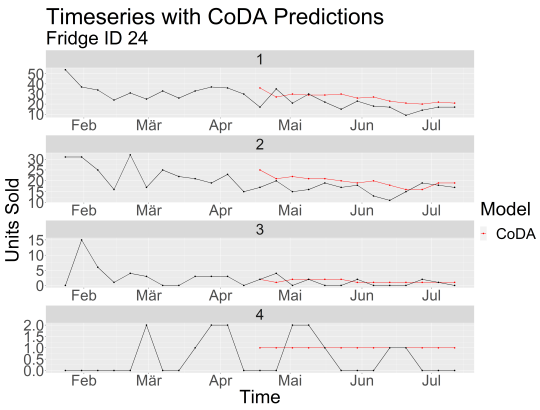
Next in figure 3.2, we add the predictions of the CoDA model. For this model we used the whole history and half of the data for the window length. In addition we extend the window at every time point, add 0.5 to all values and use the one-vs-all method. We can see that this captures the general trend well however, struggles with unexpected high peaks. In addition it is able to handle the difference in scales as seen in 3.2a. Both, categories 1 and 2 with bigger values and categories 3 and 4 with lower values, are in general modelled well. Also in time series with less data available, as in fridge 24 3.2b, the model works well. Especially category 3 with its low values is predicted well.

In figure 3.3 we apply the INGARCH model to the time series. For this, we used the whole history T_f, half of the data for the initial window length $w_f = \frac{T_f}{2}$, extend the window at every time point, add nohting to the zero values and used the poisson distribution. We used no external factors and set $p = 1, q = 1$ in model 2.3. The general trend is again captured well and in the instance of 3.3a it seems to be more reactive to sudden peaks, as often the value predicted after such a peak is heavily influenced by it.

To directly compare both models, we plot the predictions in one figure 3.4. The model specifications are the same as above. We can see that the models produce similar results to each other. In this instances it appears that INGARCH predicts slightly higher values than CoDA.

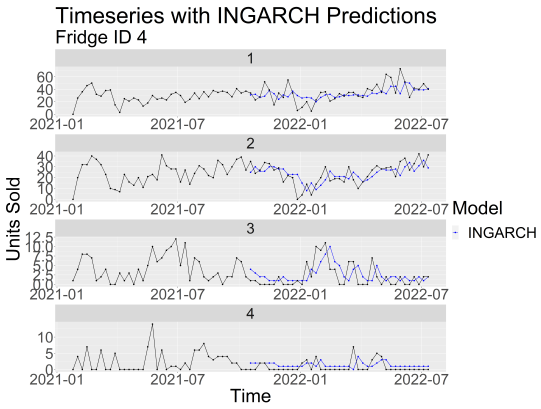


(a) Fridge 4 with the CoDA model

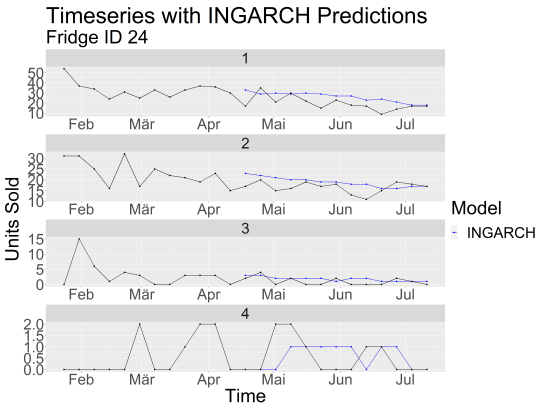


(b) Fridge 24 with the CoDA model

Figure 3.2.: Time series with CoDA model



(a) Fridge 4 with the INGARCH model



(b) Fridge 24 with the INGARCH model

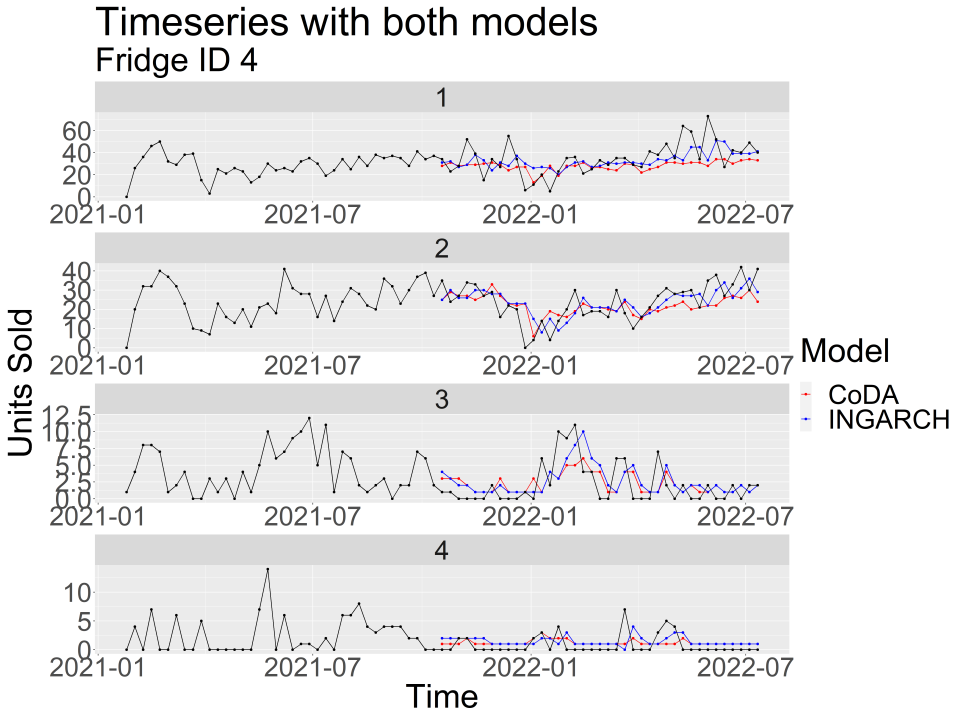
Figure 3.3.: Time series with INGARCH model

In order to get some further insight in the accuracy of our predictions, we added 95 % prediction intervals 3.5. Here we can see some differences between the intervals. While for categories with bigger values the bands are quite similar in width, for categories with lower values, CoDA has much wider bands. This is especially visible in 3.5a for category 3 and 4. However, most data points are covered by both bands.

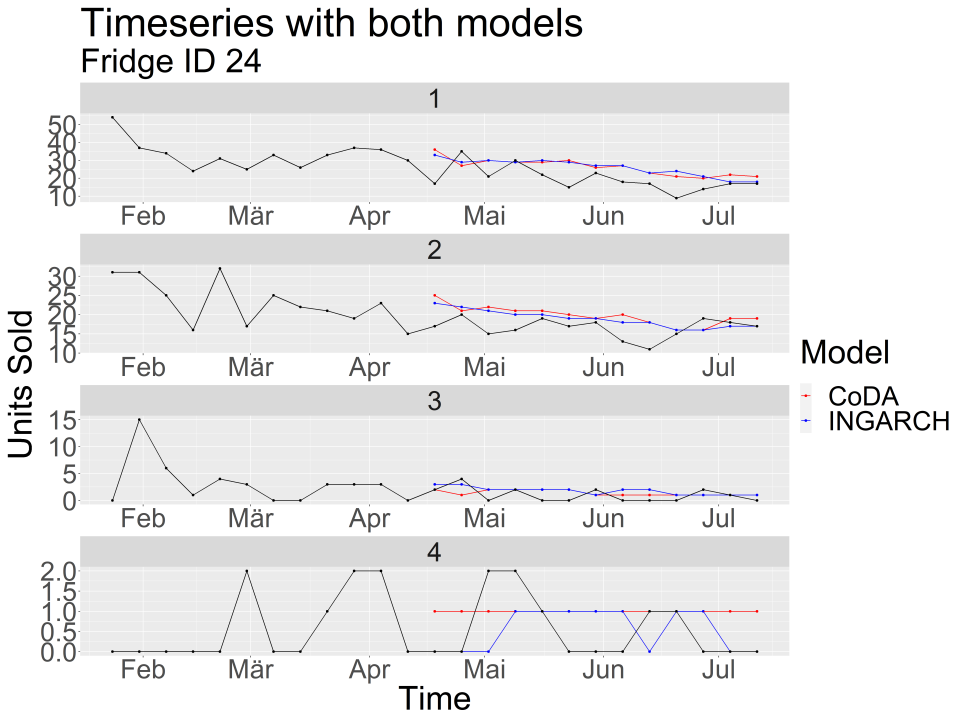
3.3. R-Code

We conducted our analysis in the statistical software R [R C22]. For our data cleansing, data handling and plotting we use the *tidyverse* package [Wic+19]. Further we use the packages *here* [Mül20], *miceadds* [RG23] and *parallel*, which is part of core R, to facilitate our analysis.

For building our CoDA model we use the packages *vars* [Pfa08b; Pfa08a] and *robCompositions* [THF11; FHT18]. Especially the functions `pivotCoord`, which performs

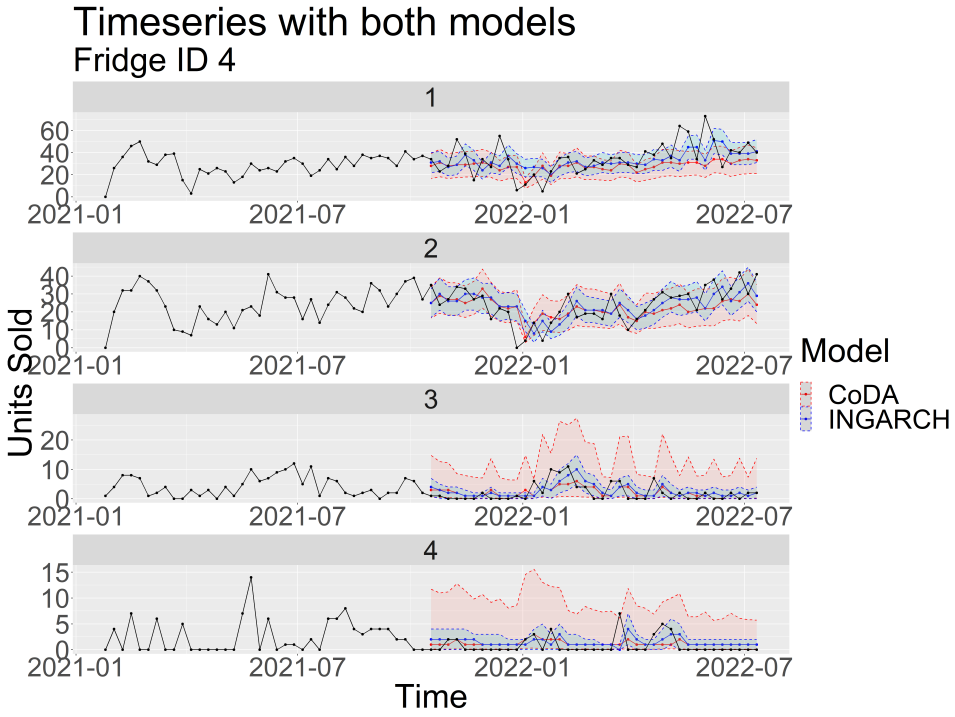


(a) Fridge 4 with the both models

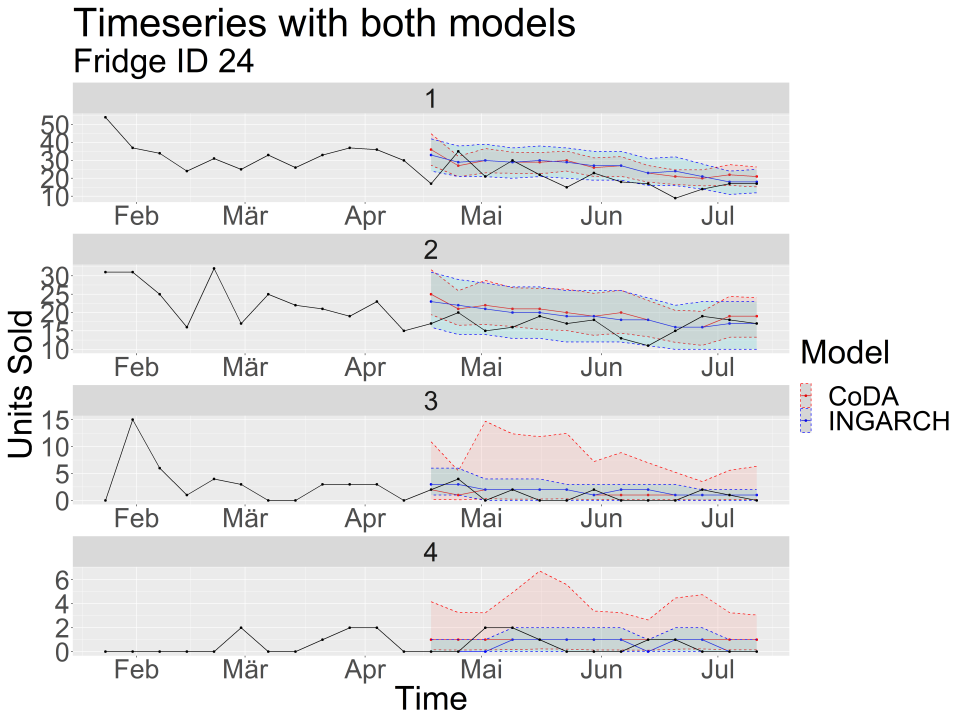


(b) Fridge 24 with the both models

Figure 3.4.: Time series with both models



(a) Fridge 4 with the both models and their prediction intervals



(b) Fridge 24 with the both models and their prediction intervals

Figure 3.5.: Time series with both models and their prediction intervals

the ilr transformation described in 2.2.2, *VAR*, which builds the VAR model described in 2.2.4, and `D2invPC` which performs the necessary back transformation to get predictions in the desired space. The INGARCH(p,q) analysis is mainly done with the package *tscount* [LFF17; Lib+20]. The core function used is `tsglm` which we use to fit the INGARCH(p,q) model as well as the log-linear model. The zero-inflated model were fitted using the function `zeroinfl` from the package *pscl* [ZKJ08]. For the VGAM we used the package *VGAM* [Yee10].

Since we focus our efforts on the INGARCH(p,q) and CoDA model, we will only describe the functions used by them. In general, all functions can be grouped into three categories: general, INGARCH specific and CoDA specific. General functions are used for both, the INGARCH(p,q) and the CoDA model. INGARCH and CoDA specific functions are only used for their respective methods. The code for them can be found in ??.

Notable general functions are `Data.Window` and `Data.Preparation`. The former function splits the time series in the specified windows and the value to be predicted. The models are then fitted on these windows and the prediction result is compared with the actual value. The latter one brings the data in the right format, replaces missing values with 0 and accounts for the length of the history chosen. In addition, for CoDA it also transform the data into the right format needed for the one-vs-all method. Other important functions are `Model.Error` and `Model.ErrorOverall` which implement the error measure introduced in 3.1.3 and summarises it.

There are three INGARCH specific functions. The first function is `Ingarch.DataPreparation` which transforms the data into the right format needed to fit the INGARCH(p,q) model. At its core it uses the `Data.Preparation` function but adds the additional option to replace zero values with 1. The second function is `Ingarch.Prediction` which is the function where the model is fit and the predicted value is calculated. It used the `tsglm` function to fit the model for each window and predicts the next value. The third function is `Ingarch.Analysis` which acts as a wrapper function to streamline and facilitate the analysis. The previously mentioned model specifications can be chosen here as well as various other options.

The CoDA specific functions have the same structure as the INGARCH ones. Again there are `Coda.DataPreparation`,`Coda.Prediction` and `Coda.Analysis` and they act like their respective INGARCH counterparts. `Coda.DataPreparation` transforms the data into the correct format, `Coda.Prediction` fits the model, predicts the future value and compares it with the true value and `Coda.Analysis` is again the wrapper function where the mentioned specifications can be chosen.

3.4. Results

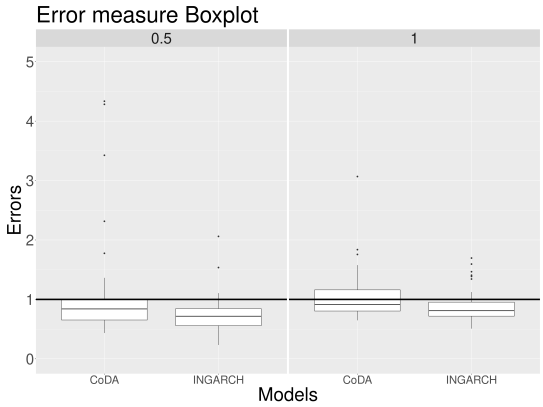
In this section we present and describe the results for our methods with their variations. For this we use the previously introduced error measure, calculate it for all available fridges and summarise the results. We show the results as graphics for easier interpretation. The tables with the exact values are shown in the appendix.

3.4.1. General Specifications

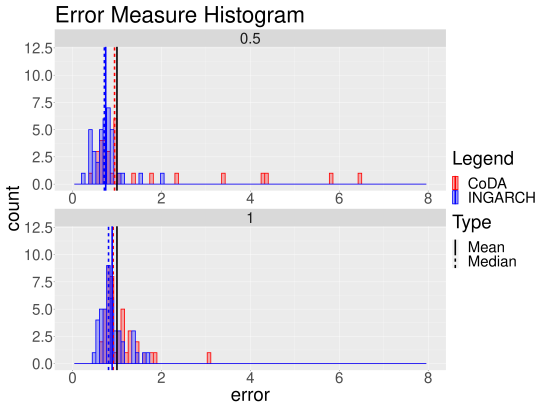
First, we start with specifications which can be chosen for both CoDA and INGARCH.

History

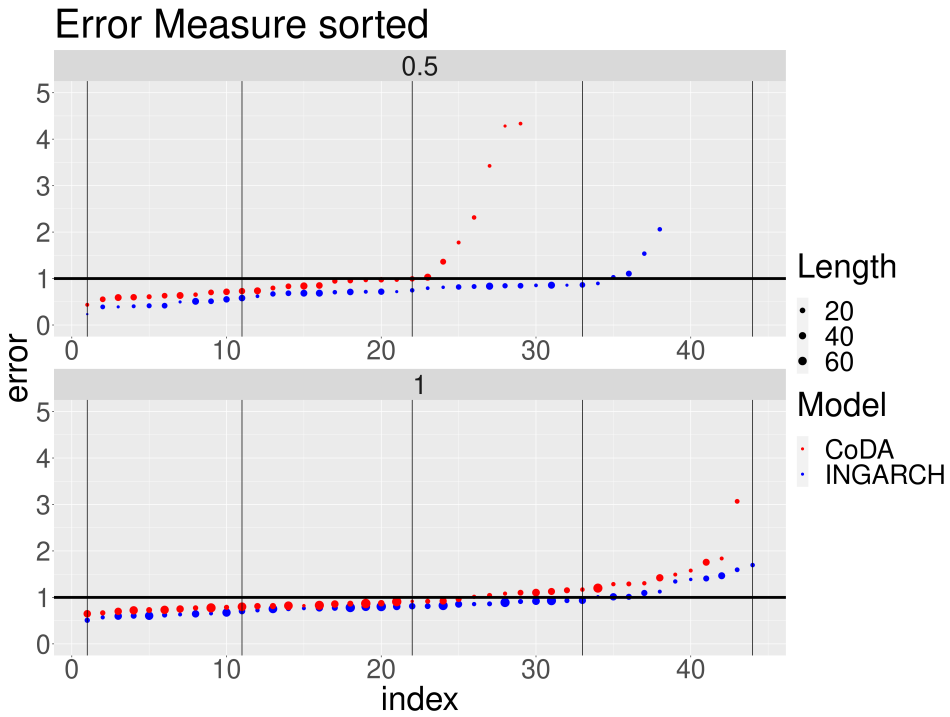
As mentioned various times throughout this thesis, the length of the history is one of the parameters which can be adjusted. Since we deal with time series with different lengths, we take the history as a fraction of the total length. In figure 3.6 we visualise the results as a boxplot, a quantile plot and a histogram.



(a) Boxplot for different history lengths



(b) Histogram for different history lengths



(c) Quantiles Plot for different history lengths

Figure 3.6.: Comparison of different history lengths

In figure 3.6 we can see that the results for CoDA vary for the different history lengths. While for a history of half of the length of the original time series, on around 75% of the fridges the error measure is smaller than 1, this number drops to 50% if we use the whole history. However, one can see in the quantile plot 3.6c that we have 8 less values for the factor 0.5 than for 1. This means that either we have larger values than the limits of the y-axis or that the method was unable to compute any result at all.

For INGARCH, the results are very similar. For a factor of 1 we get slightly higher values for the error measure as seen in 3.6a. But again in 3.6c we see that we have less values for the shorter history. So again they are either too large to be shown, or there do not exist any values at all.

Frame

Next, we vary the initial frame length w_f. We choose to extend the frame with each new data point. In addition, we take the frame as a fraction of the history used. For example, the value $w_f = 0.3$ means that 30% of the data points are chosen for the first estimation. The results are portrayed in 3.7. In general, there is not much difference between the different frames. INGARCH seems to perform better for all three values. In figure 3.7c we can see that for CoDA some time series yielded very high errors or couldn't calculate at all.

Window Shape

We also vary the shape of the window. As explained in 3.1 we either use a fixed amount of points and add and remove points as time goes on, or we continuously add points to the window. The results are in figures 3.8. We can see that there are no big differences between the methods. For CoDA it seems that the fixed methods has some struggles for certain fridges 3.8b. For INGARCH there is no notable difference.

3.4.2. INGARCH Specifications

Next we will investigate the INGARCH specific options.

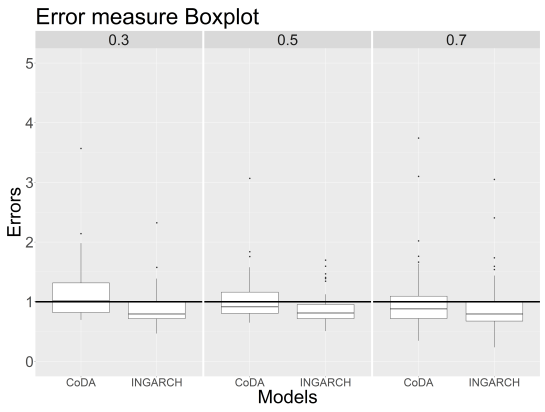
Distribution

As mentioned in 3.4.2 we can replace the Poisson distribution with a Negative Binomial Distribution in 2.5. The results are shown in 3.9.

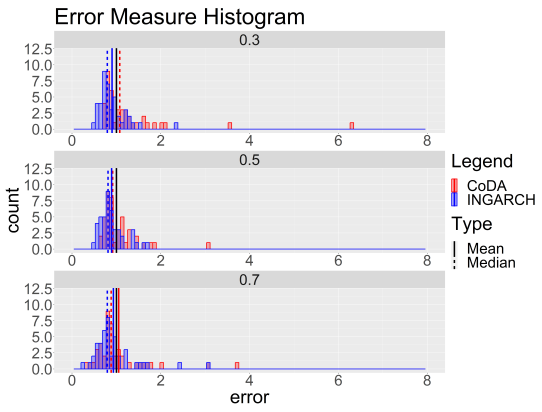
As we can see, we get the exactly the same results for both distributions. Since the Poisson Distribution is a limiting case of the Negative Binomial Distribution when $\phi \longrightarrow \infty$ in 3.1.2, [Lib16].

Number of Past Means and Observations

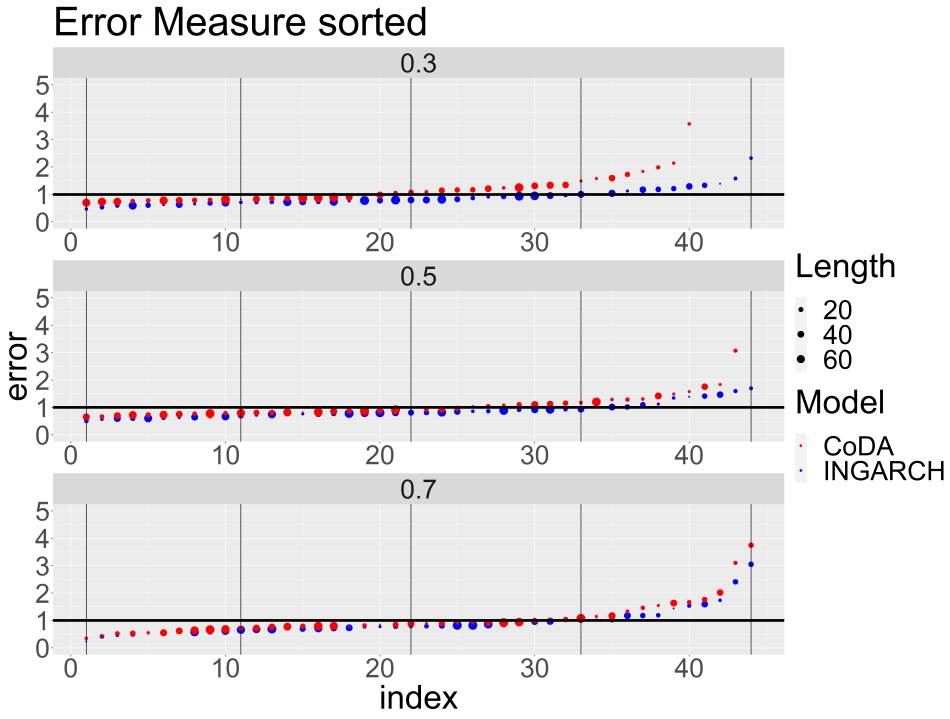
The order in the INGARCH(p,q) model is another parameter which can be chosen. For simplicities sake we only compare our INGARCH(1,1) with an INGARCH(1,2) and



(a) Boxplot for different frames



(b) Histogram for different frames



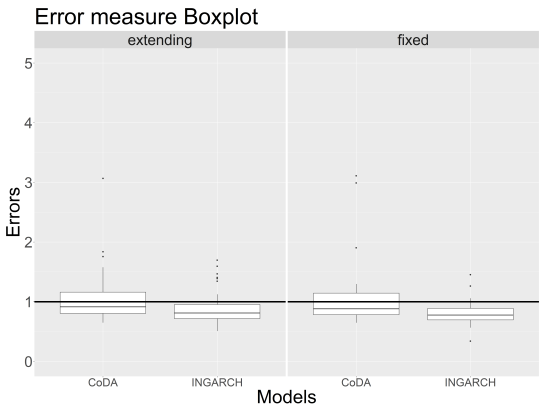
(c) Quantiles for different frames

Figure 3.7.: Comparison of different frames

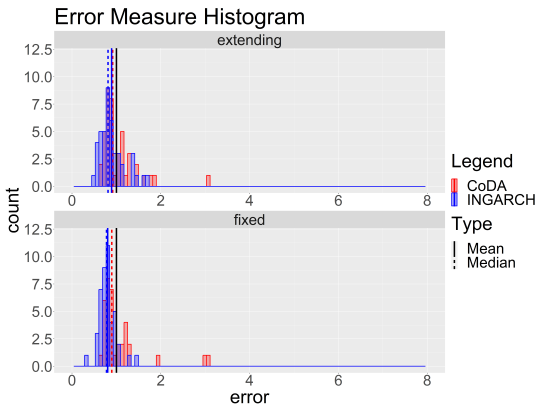
INGARCH(2,1) model. However, further models could be tried out and compared. One could even optimise over the optimal order.

In figure **??** we compare the INGARCH(1,1) model (red) with the INGARCH(1,2) model (blue). We can see that the performance is very similar . Hence we prefer the smaller model.

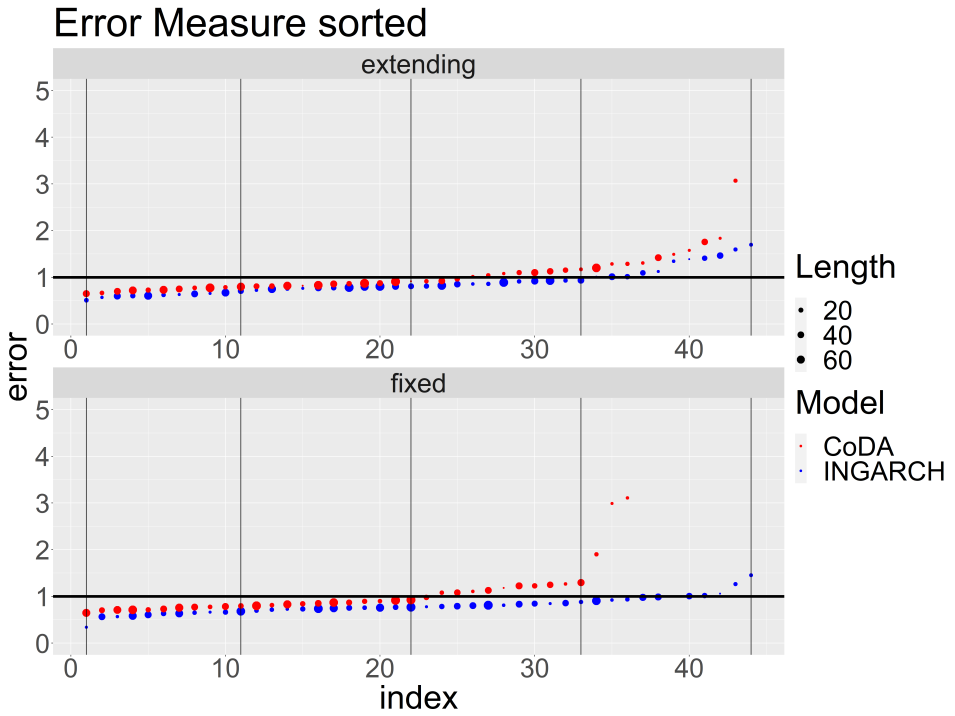
In figure **??** we compare the INGARCH(1,1) (red) model with the INGARCH(2,1) (blue) model. Again the performance is very similar in general.



(a) Boxplot for different window shapes

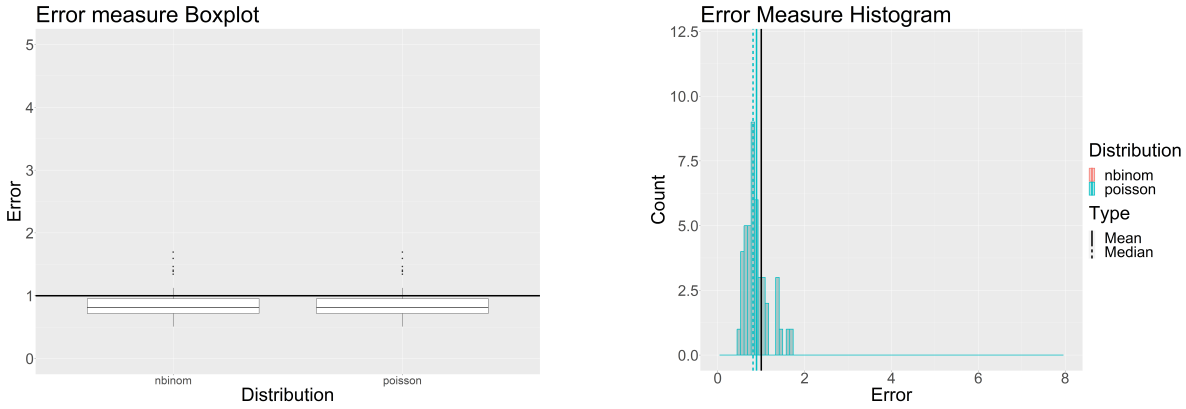


(b) Histogram for different window shapes



(c) Quantiles for different window shapes

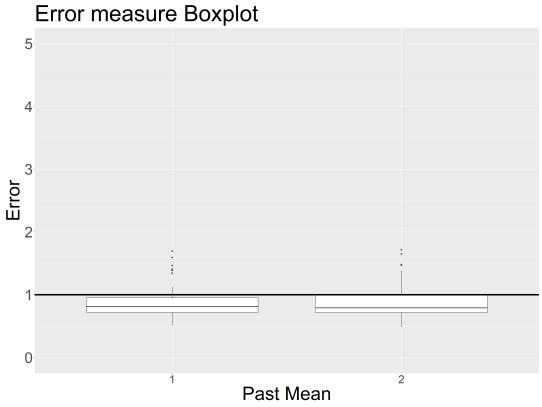
Figure 3.8.: Comparison of different window shapes



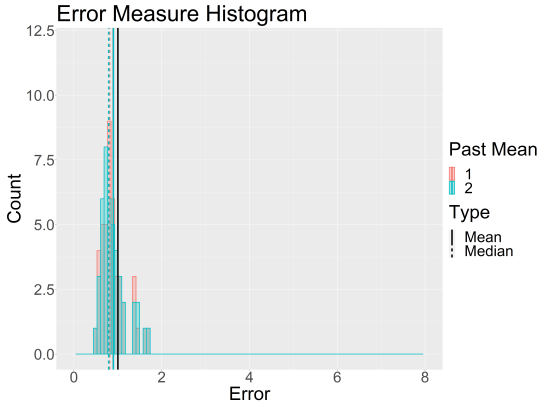
(a) Boxplot for different distributions

(b) Histogram for different distributions

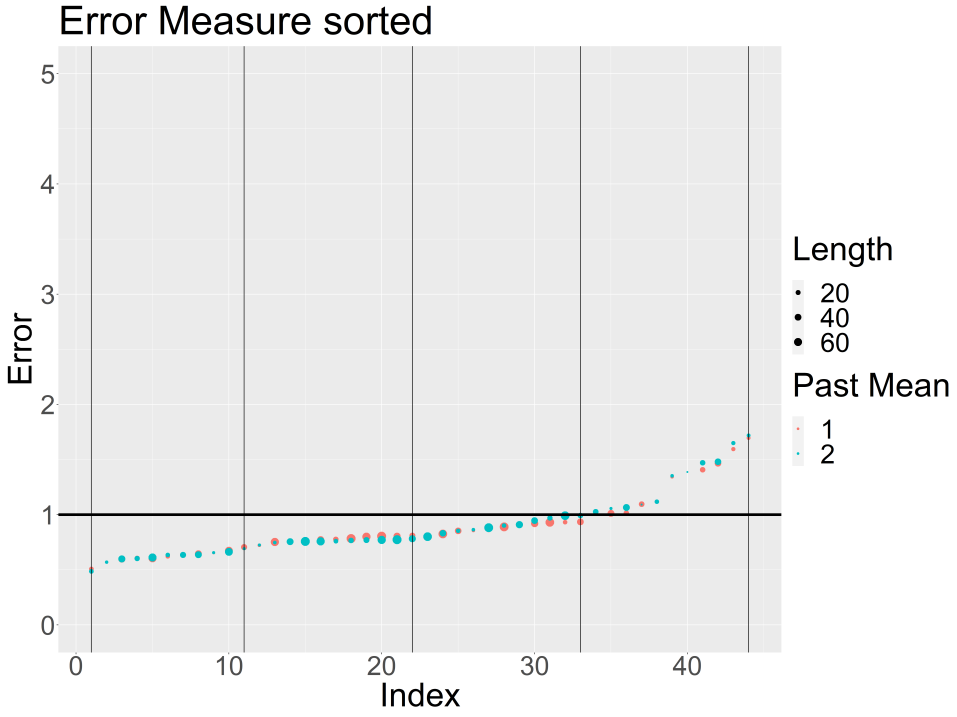
Figure 3.9.: Comparison of different distributions



(a) Boxplot for a different number of past means

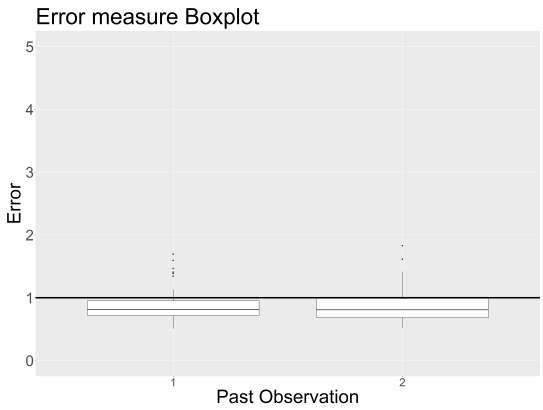


(b) Histogram for a different number of past means



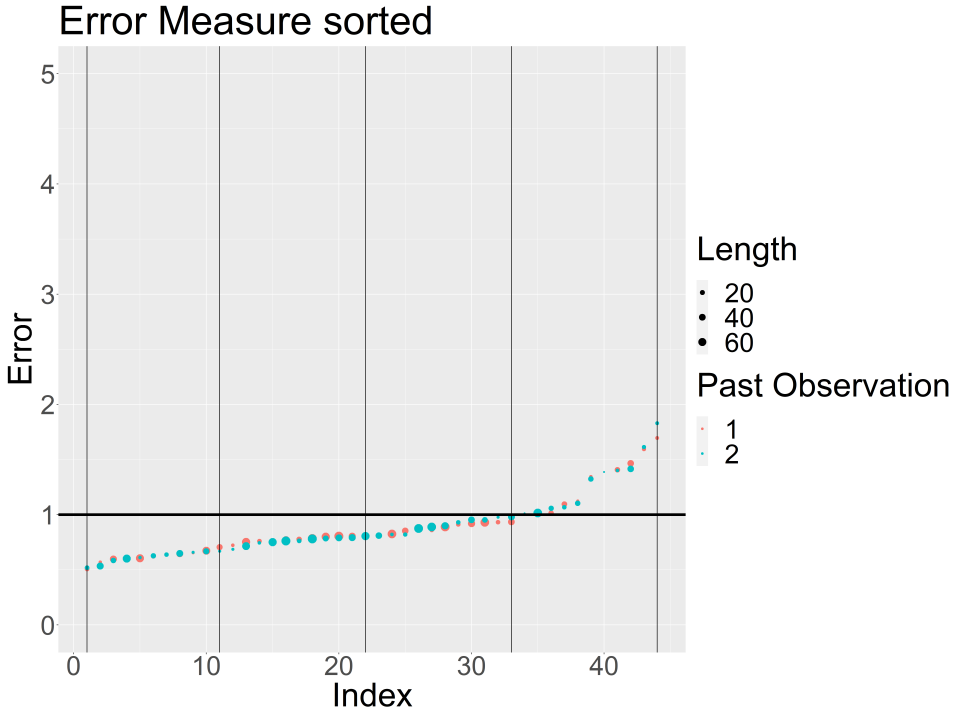
(c) Quantiles for a different number of past means

Figure 3.10.: Comparison of a different number of past means



(a) Boxplot for a different number of past observations

(b) Histogram for a different number of past observations



(c) Quantiles for a different number of past observations

Figure 3.11.: Comparison of a different number of past observations

4. Conclusion

Bibliography

Biswas, Atanu and Peter X.-K. Song. 'Discrete-valued ARMA processes'. In: *Statistics & Probability Letters* 79.17 (2009), pp. 1884–1889. ISSN: 0167-7152. DOI: https://doi.org/10.1016/j.spl.2009.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0167715209001977.

Chang, Tiao J., J.W. Delleur and M.L. Kavvas. 'Application of Discrete Autoregressive Moving Average models for estimation of daily runoff'. In: *Journal of Hydrology* 91.1 (1987), pp. 119–135. ISSN: 0022-1694. DOI: https://doi.org/10.1016/0022-1694(87)90132-6. URL: https://www.sciencedirect.com/science/article/pii/0022169487901326.

Douc, R., P. Doukhan and E. Moulines. 'Ergodicity of observation-driven time series models and consistency of the maximum likelihood estimator'. In: *Stochastic Processes and their Applications* 123.7 (2013). A Special Issue on the Occasion of the 2013 International Year of Statistics, pp. 2620–2647. ISSN: 0304-4149. DOI: https://doi.org/10.1016/j.spa.2013.04.010. URL: https://www.sciencedirect.com/science/article/pii/S0304414913001051.

Doukhan, Paul, Konstantinos Fokianos and Dag Tjøstheim. 'On weak dependence conditions for Poisson autoregressions'. In: *Statistics & Probability Letters* 82.5 (2012), pp. 942–948. ISSN: 0167-7152. DOI: https://doi.org/10.1016/j.spl.2012.01.015. URL: https://www.sciencedirect.com/science/article/pii/S0167715212000259.

Egozcue, Juan Jose et al. 'Isometric Logratio Transformations for Compositional Data Analysis'. In: *Mathematical Geology* 35 (Apr. 2003), pp. 279–300. DOI: 10.1023/A:1023818214614.

Ferland, René, Alain Latour and Driss Oraichi. 'Integer-Valued GARCH Process'. In: *Journal of Time Series Analysis* 27.6 (2006), pp. 923–942. DOI: https://doi.org/10.1111/j.1467-9892.2006.00496.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9892.2006.00496.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9892.2006.00496.x.

Filzmoser, Peter and Karel Hron. '2.30 - Compositional Data Analysis in Chemometrics'. In: *Comprehensive Chemometrics (Second Edition)*. Ed. by Steven Brown, Romà Tauler and Beata Walczak. Second Edition. Oxford: Elsevier, 2020, pp. 641–662. ISBN: 978-0-444-64166-3. DOI: https://doi.org/10.1016/B978-0-12-409547-2.14591-3. URL: https://www.sciencedirect.com/science/article/pii/B9780124095472145913.

Filzmoser, Peter, Karel Hron and Matthias Templ. *Applied Compositional Data Analysis. With Worked Examples in R*. Springer Series in Statistics. Springer International Publishing. Springer Nature Switzerland AG, Cham, Switzerland, 2018. ISBN: 978-3-319-96420-1.

Fokianos, Konstantinos, Anders Rahbek and Dag Tjøstheim. 'Poisson Autoregression'. In: *Journal of the American Statistical Association* 104.488 (2009), pp. 1430–1439. DOI: 10.1198/jasa.2009.tm08270. eprint: https://doi.org/10.1198/jasa.2009.tm08270. URL: https://doi.org/10.1198/jasa.2009.tm08270.

Fokianos, Konstantinos and Dag Tjøstheim. 'Log-linear Poisson autoregression'. In: *Journal of Multivariate Analysis* 102.3 (2011), pp. 563–578. ISSN: 0047-259X. DOI: https://doi.org/10.1016/j.jmva.2010.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0047259X10002320.

Harvey, A. C. and C. Fernandes. 'Time Series Models for Count or Qualitative Observations'. In: *Journal of Business & Economic Statistics* 7.4 (1989), pp. 407–417. ISSN: 07350015. URL: http://www.jstor.org/stable/1391639 (visited on 04/04/2023).

Heinen, Andreas. *Modelling Time Series Count Data: An Autoregressive Conditional Poisson Model*. MPRA Paper 8113. University Library of Munich, Germany, July 2003. URL: https://ideas.repec.org/p/pra/mprapa/8113.html.

Kynčlová, Petra, Peter Filzmoser and Karel Hron. 'Modeling Compositional Time Series with Vector Autoregressive Models'. In: *Journal of Forecasting* 34.4 (2015), pp. 303–314. DOI: https://doi.org/10.1002/for.2336. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/for.2336. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.2336.

Liboschik, Tobias. 'Modeling count time series following generalized linear models'. In: 2016.

Liboschik, Tobias, Konstantinos Fokianos and Roland Fried. 'tscount: An R Package for Analysis of Count Time Series Following Generalized Linear Models'. In: *Journal of Statistical Software* 82.5 (2017), pp. 1–51. DOI: 10.18637/jss.v082.i05.

Liboschik, Tobias et al. *tscount: Analysis of Count Time Series*. R package version 1.4.3. 2020. URL: https://CRAN.R-project.org/package=tscount.

Lütkepohl, H. *New Introduction to Multiple Time Series Analysis*. Springer Berlin Heidelberg, 2007. ISBN: 9783540262398. URL: https://books.google.at/books?id=muorJ6FHIiEC.

Macdonald, Lain, Iain L. MacDonald and Iain L. MacDonald. 'Hidden Markov and Other Models for Discrete- valued Time Series'. In: 1997.

Müller, Kirill. *here: A Simpler Way to Find Your Files*. R package version 1.0.1. 2020. URL: https://CRAN.R-project.org/package=here.

Pfaff, B. *Analysis of Integrated and Cointegrated Time Series with R*. Second. ISBN 0-387-27960-1. New York: Springer, 2008. URL: https://www.pfaffikus.de.

Pfaff, Bernhard. 'VAR, SVAR and SVEC Models: Implementation Within R Package vars'. In: *Journal of Statistical Software* 27.4 (2008). URL: https://www.jstatsoft.org/v27/i04/.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2022. URL: https://www.R-project.org/.

Robitzsch, Alexander and Simon Grund. *miceadds: Some Additional Multiple Imputation Functions, Especially for 'mice'*. R package version 3.16-18. 2023. URL: https://CRAN.R-project.org/package=miceadds.

Templ, Matthias, Karel Hron and Peter Filzmoser. *robCompositions: an R-package for robust statistical analysis of compositional data*. John Wiley and Sons, 2011, pp. 341–355. ISBN: 978-0-470-71135-4.

Wickham, Hadley et al. 'Welcome to the tidyverse'. In: *Journal of Open Source Software* 4.43 (2019), p. 1686. DOI: 10.21105/joss.01686.

Wood, Simon N. 'Stable and Efficient Multiple Smoothing Parameter Estimation for Generalized Additive Models'. In: *Journal of the American Statistical Association* 99.467 (2004), pp. 673–686. ISSN: 01621459. URL: http://www.jstor.org/stable/27590439 (visited on 30/04/2023).

Woodard, Dawn B., David S. Matteson and Shane G. Henderson. 'Stationarity of generalized autoregressive moving average models'. In: *Electronic Journal of Statistics* 5.none (2011), pp. 800–828. DOI: 10.1214/11-EJS627. URL: https://doi.org/10.1214/11-EJS627.

Yee, T. W. and C. J. Wild. 'Vector Generalized Additive Models'. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.3 (1996), pp. 481–493. ISSN: 00359246. URL: http://www.jstor.org/stable/2345888 (visited on 30/04/2023).

Yee, Thomas. *Vector Generalized Linear and Additive Models: With an Implementation in R*. Jan. 2015, pp. 1–589. ISBN: 978-1-4939-2817-0. DOI: 10.1007/978-1-4939-2818-7.

Yee, Thomas W. 'The VGAM Package for Categorical Data Analysis'. In: *Journal of Statistical Software* 32.10 (2010), pp. 1–34. DOI: 10.18637/jss.v032.i10.

Zeger, Scott L. 'A Regression Model for Time Series of Counts'. In: *Biometrika* 75.4 (1988), pp. 621–629. ISSN: 00063444. URL: http://www.jstor.org/stable/2336303 (visited on 04/04/2023).

Zeileis, Achim, Christian Kleiber and Simon Jackman. 'Regression Models for Count Data in R'. In: *Journal of Statistical Software* 27.8 (2008). URL: http://www.jstatsoft.org/v27/i08/.

Zhu, Fukang. 'Zero-inflated Poisson and negative binomial integer-valued GARCH models'. In: *Journal of Statistical Planning and Inference* 142.4 (2012), pp. 826–839. ISSN: 0378-3758. DOI: https://doi.org/10.1016/j.jspi.2011.10.002. URL: https://www.sciencedirect.com/science/article/pii/S037837581100379X.

List of Figures

List of Tables

Appendix A.

R-Functions

A.1. General Functions

```
Data.Window <- function(Timeseries,Frame,
Method = c("non-overlapping", "fixed", "extending"),
PredictionStep = 1) {

  Method <- match.arg(Method)
  Timeseries_Length <- dim(Timeseries)[1]

  if (is.null(Timeseries_Length)) {
    Timeseries <- as.matrix(Timeseries, ncol = 1)
    Timeseries_Length <- dim(Timeseries)[1]
  }
  if (is.null(Timeseries_Length))
    stop("Enter valid timeseries")

  stopifnot(Timeseries_Length >= Frame)

  if (Method == "non-overlapping") {
    Window_Number <-
      floor(Timeseries_Length / Frame) -
ifelse(Timeseries_Length %% Frame < PredictionStep, 1, 0) -
      ifelse(Frame < PredictionStep, 1, 0) #div and prediction step

    StartIndex <-
      max(1,Timeseries_Length - Window_Number * Frame - PredictionStep)

    Timeseries <- Timeseries[StartIndex:Timeseries_Length, ]

    Result <- lapply(c(0:(Window_Number - 1)), function(i) {
      return(list(timeSeriesValue_window =
Timeseries[(i * Frame + 1):((i + 1) * Frame), ],
                  timeSeriesValue_future =
```

```
Timeseries[((i + 1) * Frame + PredictionStep), ]))
    })

    names(Result) <- c(1:Window_Number)
  }

  else if (Method == "fixed") {
    #Length of the series - length of the window + prediction step equals the
#Number of windows
    Window_Number <- Timeseries_Length - Frame - PredictionStep + 1

    Result <- lapply(c(1:Window_Number), function(i) {
      return(list(timeSeriesValue_window =
Timeseries[i:(Frame + i - 1), ],
                  timeSeriesValue_future =
Timeseries[(Frame + i - 1 + PredictionStep), ]))
    })
    names(Result) <- c(1:Window_Number)
  }

  else if (Method == "extending") {
    #Length of the series - length of the first window + prediction step
#equals the number of windows
    Window_Number <- Timeseries_Length - Frame - PredictionStep + 1

    Result <- lapply(c(1:Window_Number), function(i) {
      return(list(timeSeriesValue_window =
Timeseries[1:(Frame + i - 1), ],
                  timeSeriesValue_future =
Timeseries[(Frame + i - 1 + PredictionStep), ]))
    })
    names(Result) <- c(1:Window_Number)
  }

  else {
    stop("Enter valid Method")
  }

  return(Result)
}

Data.Preparation <- function(Data_Raw,
                             OneVsAll = F,
                             PivotGroup = "1",
```

```
                                Category = c(1, 2, 3, 4),
                                NA_to = 0,
                                HistoryLength = 1,
                                TakeSubCategory = FALSE){

  #Sub or Main Category
  if(TakeSubCategory){
    Category_Var <- "sub_category_id"
    Data_Raw <- subset(Data_Raw,select = -main_category_id)
  }else{
    Category_Var <- "main_category_id"
#    Data_Raw <- subset(Data_Raw,select = -sub_category_id)
  }


  columns <- c("week_date", as.character(Category))

    Data_Raw <- Data_Raw %>%
    dplyr::filter(get(Category_Var) %in% Category) %>%
    group_by(across(all_of(Category_Var)), week_date) %>%
    dplyr::mutate(sold = sum(sold)) %>%
    dplyr::select(all_of(c("week_date",Category_Var,"sold"))) %>%
    dplyr::distinct() %>%
    pivot_wider(names_from
                = Category_Var, values_from = sold) %>%
    ungroup() %>%
    setnafill(type = "const", fill = NA_to) %>%
    dplyr::select(any_of(columns)) %>%
    dplyr::mutate(across(.cols = as.character(Category),
  .fns = as.double))

  #Determining the length of the Timeseries
  DataRaw_Length <- dim(Data_Raw)[1]

  if(between(HistoryLength,0,1)){
    DataRaw_Start <- round(DataRaw_Length * (1-HistoryLength) + 1)
  }else {
    DataRaw_Start <- DataRaw_Length - HistoryLength
  }

  Data_Raw <- Data_Raw[DataRaw_Start:DataRaw_Length,]

  if (OneVsAll) {
    Data_Raw <- Data_Raw %>%
```

```
      dplyr::mutate(other = dplyr::select(.,-all_of(PivotGroup),
-"week_date") %>%
rowSums(na.rm =F)) %>%
      dplyr::select("week_date", all_of(PivotGroup), "other")

  }

  return(Data_Raw)

}
```

A.2. INGARCH Functions

```
Ingarch.DataPreparation <- function(Data_Raw,
                                     ZeroHandling = c("none","zero_to_one"),
                                    HistoryLength = 1,
                                    TakeSubCategory = F){

  ZeroHandling <- match.arg(ZeroHandling)

  if(TakeSubCategory){
    Category <- sort(unique(Data_Raw$sub_category_id))
  }
  else{
    Category <- sort(unique(Data_Raw$main_category_id))
  }

  Data_Prepared <- Data.Preparation(Data_Raw = Data_Raw,
                                    OneVsAll = F,
                                    Category = Category,
                                    NA_to = 0,
                                    HistoryLength = HistoryLength,
                                    TakeSubCategory = TakeSubCategory)

  if (ZeroHandling == "none") {
    return(Data_Prepared)
  }
  else if (ZeroHandling == "zero_to_one") {
    Data_Prepared[Data_Prepared == 0] <- 1
    return(Data_Prepared)
  }
  else{
    stop("Enter valid zero handling method")
```

```
    }
}

Ingarch.Prediction <- function(Data_Window,
                               Data_WindowNoTransform,
                               Category,
                               PredictionStep = 1,
                               Frame = 10,
                               Distribution = "poisson",
                               Plot = F,
                               WindowMethod = "extending",
                               External = FALSE,
                               PastOb = 1,
                               PastMean = 1){

  NumberOfWindows <- length(Data_Window)


  #Calculating the prediction for each window
  Result <- lapply(c(1:NumberOfWindows),function(WindowIndex){


    #Extracting fitting values, true value and last known value
    TimeSeriesValue_Window <-
Data_Window[[WindowIndex]]$timeSeriesValue_window[c("week_date",Category)]
    TimeseriesValue_Future <-
Data_WindowNoTransform[[WindowIndex]]$timeSeriesValue_future[[Category]]
    TimeSeriesValue_LastKnown <- tail(TimeSeriesValue_Window[[Category]],n=1)

    Xreg <-NULL
    Ext <- NULL
    XregFuture <- NULL

    if(External){
      Xreg <- Data_Window[[WindowIndex]]$timeSeriesValue_window %>%
              dplyr::select(-c("week_date", all_of(Category))) %>%
              as.matrix()
      Ext <- rep(TRUE,ncol(Xreg))
      XregFuture <- matrix(round(apply(tail(Xreg,n=5),2,mean)),nrow=1)
      names(XregFuture) <- colnames(Xreg)
    }

    #Fitting the model
```

```
    PastOb_Used <- c(1:PastOb)
    if (PastMean == 0) {
      PastMean_Used <- NULL
    } else{
      PastMean_Used <- c(1:PastMean)
    }

    #Determining init.method. If none works, we skip this fridge
    SkipWindow <- FALSE

    for (method in c("marginal", "iid", "firstobs", "zero")) {

      Model <- tryCatch(
        expr = {tsglm(TimeSeriesValue_Window[[Category]],
                      model = list("past_obs" = PastOb_Used,
                                   "past_mean" = PastMean_Used,
                                   external = Ext),
                      xreg = Xreg,
                      distr = Distribution,
                      link = "identity",
                      init.method = method)},
        error = function (e) e
      )

    if(inherits(Model,"error")){
      if(method == "zero") {
        SkipWindow <- TRUE
        print("No init.method works. Skipping Window.")
        break
      }
      next
    }else{
      initMethod <- method
      break
    }
    }

    if(SkipWindow)return(list(prediction=NA,model=NA))

    #Predicting the future value depending on PredictionStep
    PredictionResult <- predict(Model,n.ahead = PredictionStep,
type = "shortest",
                                level = 0.90,newxreg = XregFuture)
```

```
  #Rounding it since we only have integers
  ValuePredict <- round(PredictionResult$pred)


  #Extracting the lower and upper prediction interval
  PredictionInterval_Lower <- PredictionResult$interval[1,"lower"]
  PredictionInterval_Upper <- PredictionResult$interval[1,"upper"]


  #Calculating the prediction error
  PredictionError <- as.numeric(ValuePredict - TimeseriesValue_Future)


  #Calculating the normed prediction error
  PredictionError_Normed <- PredictionError

  return(list(
    prediction = data.frame(
      predictionError = PredictionError,
      valuePredict = ValuePredict,
      predictionError_normed = PredictionError_Normed,
      lowerBound = PredictionInterval_Lower,
      upperBound = PredictionInterval_Upper,
      valueTrue = TimeseriesValue_Future,
      valueLastKnown = TimeSeriesValue_LastKnown,
      category = Category,
      date = Data_Window[[WindowIndex]]$timeSeriesValue_future[[1]],
      distribution = Distribution,
      window = WindowIndex,
      window_length = dim(TimeSeriesValue_Window)[1],
      window_baseLength = Frame,
      pastOb = PastOb,
      pastMean = PastMean,
      external = External,
      initMethod = initMethod
    ),
    model = Model
  ))

})

#Transforming result in a nicer format
Result <- discard(Result, ~all(is.na(.x)))
```

```
  Result_Prediction <- bind_rows(UnlistListElement(Result, "prediction"))
  Result_Model <- UnlistListElement(Result, "model")
  names(Result_Model) <- sapply(c(1:NumberOfWindows),
function(i){paste("window",i,sep = "")})


  #Calculation the normed prediction error
  div <- sapply(c(1:dim(Result_Prediction)[1]),function(i){

    return(Normation(x = Result_Prediction$valueTrue[1:i],
                     y = Result_Prediction$valueLastKnown[1:i]))}
    )
  if(0 %in% div ) div[div == 0] <- 0.5
  Result_Prediction$predictionError_normed <-
Result_Prediction$predictionError_normed/div


  #Plotting diagnostic plots or not
  #if (Plot) {
  #  plot(model, ask = F)
  #}

  return(list(result = Result_Prediction,
              model = Result_Model))

}

Ingarch.Analysis <- function(Data_Raw,
                             Id,
                             PredictionStep = 1,
                             Distribution = "poisson",
                             ModelType = "ingarch",
                             Plot = F,
                             Category_Main = c("1", "2", "3", "4"),
                             TakeSubCategory = F,
                             Category_Sub = NULL,
                             Frame = 10,
                             WindowMethod = "extending",
                             ZeroHandling = "none",
                             PastOb = 1,
                             PastMean = 1,
                             External = FALSE,
                             HistoryLength = 1,
                             Multicore = TRUE,
```

```
                              NCores = 2
                              ) {

  stopifnot(ModelType %in% c("ingarch","ingarch_OneVsAll"))

  #Only return IDs with results
  Id_Result <- Id

  #Operating on Sub category Level
  SubCategory_Column  <- NULL
  if(TakeSubCategory){
    SubCategory_Column <- "sub_category_id"
    stopifnot(length(Category_Main)==1)
  }
  if(is.null(Category_Sub)){
    Category_Sub <- unique(Data_Raw$sub_category_id)
  }

  #Calculating Prediction results for all ids and each category
  PredictionResult_AllIDAllCategory <- lapply(Id,function(Id_RunVariable){
    print(paste("Calculating for ID:",Id_RunVariable))
    #Preparing data
    Data_Processed <- Data_Raw %>%
      filter(fridge_id == Id_RunVariable &
               main_category_id %in% as.integer(Category_Main) &
               sub_category_id %in% as.integer(Category_Sub)) %>%
      dplyr::select(week_date, main_category_id, any_of(SubCategory_Column) ,sold) %>%
      arrange(week_date)

    Data_Prepared <-
Ingarch.DataPreparation(Data_Raw = Data_Processed,
                        ZeroHandling = ZeroHandling,
HistoryLength = HistoryLength,
TakeSubCategory = TakeSubCategory)

    Data_PreparedNoTransform <-
Ingarch.DataPreparation(Data_Raw = Data_Processed,
ZeroHandling = "none",
HistoryLength = HistoryLength,
TakeSubCategory = TakeSubCategory)
    Category <- names(Data_Prepared)[-1]

    #Calculating the length of the timeseries
    TimeSeries_Length <- length(unique(Data_Prepared$week_date))
```

```
    #If the Frame is given as a fraction, calculate the absolute length.
#We set 5 as the minimum length needed.
    Frame_Help <- "fixed"
    if(dim(Data_Prepared)[1]<5){
      print(paste("Insufficient data. Skipping ID: ",Id_RunVariable))
      Id_Result <<- Id_Result[Id_RunVariable!=Id_Result]
      return(NA)
    }
    if(Frame < 1){
      Frame_Help <- as.character(Frame)
      Frame = round(Frame*dim(Data_Prepared)[1])
      if(Frame < 5){
        Frame = 5
      }
    }

    #Creating fitting and prediction windows
    Data_Window <- Data.Window(Data_Prepared,Frame = Frame,
 Method=WindowMethod,
   PredictionStep = PredictionStep)
    Data_WindowNoTransform <- Data.Window(Data_PreparedNoTransform,
Frame = Frame,
Method=WindowMethod,
PredictionStep = PredictionStep)

    #Calculating Prediction results for each category
    if(Multicore == TRUE){

      Cluster1 <- makeCluster(NCores)
      print("Initiating Cluster")

      invisible(clusterCall(Cluster1, function() {
        source("General_Dependency.R")
        source("General_Function.R")
      }))

      invisible(clusterExport(Cluster1,list("Data.Window",
"Ingarch.DataPreparation",
"Ingarch.Prediction",
"Data_Window",
"External","PastOb",
"PastMean","Distribution",
"Plot"),
```

```
                              envir = environment()))
      print("Starting Calculations")
    PredictionResult_AllCategory <-
parLapply(Cluster1, Category,function(Category_RunVariable){

      PredictionResult <- tryCatch(
        expr = { Ingarch.Prediction(Data_Window = Data_Window,
                                    Data_WindowNoTransform = Data_WindowNoTransform,
                                    Category = Category_RunVariable,
                                    PredictionStep = PredictionStep,
                                    Frame = Frame,
                                    Plot = F,
                                    Distribution = Distribution,
                                    WindowMethod = WindowMethod,
                                    External = External,
                                    PastOb = PastOb,
                                    PastMean = PastMean)},
        error = function (e) e
      )
      if(inherits(PredictionResult,"error")){
        print(paste("Error occured in prediction: ID",Id_RunVariable,",
              Category",Category_RunVariable,PredictionResult))
        return(NA)
      }

      return(list(result=bind_rows(PredictionResult$result),
                  model=PredictionResult$model))

    })
    print("Stopping Calculations")
    print("Stopping Cluster")
    stopCluster(Cluster1)
    }
    else {
      PredictionResult_AllCategory <- lapply(Category,function(Category_RunVariable){

        PredictionResult <- tryCatch(
          expr = { Ingarch.Prediction(Data_Window = Data_Window,
                                      Data_WindowNoTransform = Data_WindowNoTransform,
                                      Category = Category_RunVariable,
                                      PredictionStep = PredictionStep,
                                      Frame = Frame,
                                      Plot = F,
                                      Distribution = Distribution,
```

```
                                        WindowMethod = WindowMethod,
                                        External = External,
                                        PastOb = PastOb,
                                        PastMean = PastMean)},
          error = function (e) e
        )
        if(inherits(PredictionResult,"error")){
          print(paste("Error occured in prediction: ID",Id_RunVariable,",
Category",Category_RunVariable,PredictionResult))
          return(NA)
        }
        return(list(result = bind_rows(PredictionResult$result),
                    model = PredictionResult$model))
      })
    }

    #Transforming data in nicer format and removing NA values
    NA_Index <- which(is.na(PredictionResult_AllCategory)==TRUE)
if(is_empty(NA_Index)){
      ModelNames <- Category
    }else{
      ModelNames <- Category[-NA_Index]
    }
    PredictionResult_AllCategory <-
PredictionResult_AllCategory[!is.na(PredictionResult_AllCategory)]
    Result_Prediction <-
bind_rows(UnlistListElement(PredictionResult_AllCategory,"result"))
    Result_Model <- UnlistListElement(PredictionResult_AllCategory,"model")
    names(Result_Model) <- ModelNames
    Result_Prediction$id <- Id_RunVariable
    Result_Prediction$windowMethod <- WindowMethod
    Result_Prediction$zeroHandling <- ZeroHandling
    Result_Prediction$frame <- Frame_Help
    Result_Prediction$history <- as.character(HistoryLength)
    Result_Prediction$timeseriesLength <- as.character(TimeSeries_Length)

    if(TakeSubCategory){
      Result_Prediction$main_category <- Category_Main
    }

    return(list(result = Result_Prediction,
                model = Result_Model))
  })
```

```
  #Removing NA (aka Timeseries which are too short)
  PredictionResult_AllIDAllCategory <-
PredictionResult_AllIDAllCategory[!is.na(PredictionResult_AllIDAllCategory)]

  #Transforming data in nicer format
  Result_Prediction <-
bind_rows(UnlistListElement(PredictionResult_AllIDAllCategory,"result"))
  Result_Prediction$id <- as.factor(Result_Prediction$id)
  Result_Model <- UnlistListElement(PredictionResult_AllIDAllCategory,"model")
  names(Result_Model) <- Id_Result
  Result_Prediction$model <- ModelType

  return(list(result = Result_Prediction,
              model = Result_Model))
}
```

A.3. CoDA Functions

```
Coda.DataPreparation <- function(Data_Raw,
           ZeroHandling = c("all", "zeros_only", "none"),
           TSpace = FALSE,
           Transform = TRUE,
           Log = FALSE,
           OneVsAll = FALSE,
           PivotGroup = "1",
           HistoryLength = 1) {

    Plus <- function(x,Value=0.5){
         x[is.na(x)] <- 0
      x <- x + Value
      return(x)
    }
    ZeroHandling <- match.arg(ZeroHandling)

    #If we compare one category to all the others we use all categories.
#Otherwise we only use the first 2
    if(OneVsAll) {
      Category <- c(1,2,3,4)
    }
    else {
      Category <- c(1,2)
    }
    Data_Prepared <- Data.Preparation(Data_Raw = Data_Raw,
```

```
                                OneVsAll = OneVsAll,
                                PivotGroup = PivotGroup,
                                NA_to=0,
                                Category = Category,
                                HistoryLength = HistoryLength)


    if (ZeroHandling == "none") {
      Data_Prepared$tsum <- rowSums(Data_Prepared[,-1])
      return(Data_Prepared)
    }

    if (ZeroHandling == "all") {
      Data_Prepared <- Data_Prepared %>%
 mutate(across(where(is.numeric),.fns = Plus))
    }
    else if (ZeroHandling == "zeros_only") {
      Data_Prepared[Data_Prepared == 0] <- 0.5
    }
    else {
      stop("Enter valid zero handling option")
    }

    Data_Ilr <-
cbind(Data_Prepared$week_date, pivotCoord(as.data.frame(Data_Prepared[,-1])))

    if (TSpace) {
      if (Log) {
        Data_Ilr$tsum <- log(rowSums(Data_Prepared[,-1]))
      }
      else {
        Data_Ilr$tsum <- rowSums(Data_Prepared[,-1])
      }
    }
    names(Data_Ilr)[1] <- "week_date"

    return(Data_Ilr)
  }

Coda.Prediction <- function(Data_TransformWindow, Data_NoTransformWindow,
Data_NoTransform, PredictionStep,
                            OneVsAll,TSpace, Log, PivotGroup, Frame = 10) {

  PredictionResult <- lapply(c(1:length(Data_TransformWindow)),
```

```
 function(WindowIndex) {


    # Selecting the fitting data and the data which should be predicted
    TimeSeriesValue_Window <-
Data_TransformWindow[[WindowIndex]]$timeSeriesValue_window[,-1]
    Date <- Data_TransformWindow[[WindowIndex]]$timeSeriesValue_future[1, 1]

    #Depending on whether we have TSpace or not we fit a VAR model or an AR model
    ####TSPACE
    if (TSpace) {
      Window_Length <- dim(TimeSeriesValue_Window)[1]

      Model <- VAR(TimeSeriesValue_Window, p=1 ,lag.max = NULL, ic= "AIC")
      ValuePredict <-  predict(Model, TimeSeriesValue_Window, n.ahead = PredictionStep)
      Size = 2

      #Initialising result vectors
      ValuePredict_Vector <-matrix(data = NA,nrow = 1,ncol = Size)
      LowerBound_Vector<- UpperBound_Vector <- vector(mode="numeric",length=Size)

      #Filling up result vectors
      for (i in 1:Size) {
        ValuePredict_Vector[1, i] <-  ValuePredict[[1]][[i]][PredictionStep]
        LowerBound_Vector[i] <-  ValuePredict[[1]][[i]][PredictionStep+1]
        UpperBound_Vector[i] <-  ValuePredict[[1]][[i]][PredictionStep+2]
      }


      TSum <-
as.numeric(tail(Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window$tsum, 1))

      #Back transformations
      ValuePredict <- ValuePredict_Vector[-Size] %>%
        matrix(nrow = 1) %>%
        D2invPC()

      LowerBound <- LowerBound_Vector[-Size] %>%

        matrix(nrow = 1) %>%
        D2invPC()

      UpperBound <- UpperBound_Vector[-Size] %>%
        matrix(nrow = 1) %>%
```

```
        D2invPC()


      # Back transformations when we use the log of the total sum
      if (Log) {
        ValuePredict <- ValuePredict * exp(ValuePredict_Vector[Size])
        ValuePredict <-
          (append(ValuePredict, exp(ValuePredict_Vector[Size])))

        LowerBound <- LowerBound * exp(ValuePredict_Vector[Size])
        LowerBound <-
          (append(LowerBound, exp(ValuePredict_Vector[Size])))

        UpperBound <- UpperBound * exp(ValuePredict_Vector[Size])
        UpperBound <-
          (append(UpperBound, exp(ValuePredict_Vector[Size])))

      }
      # Normal back transformations
      else{
        ValuePredict <- ValuePredict * ValuePredict_Vector[Size]
        ValuePredict <-
          (append(ValuePredict, ValuePredict_Vector[Size]))

        LowerBound <- LowerBound * ValuePredict_Vector[Size]
        LowerBound <-
          (append(LowerBound, ValuePredict_Vector[Size]))

        UpperBound <- UpperBound * ValuePredict_Vector[Size]
        UpperBound <-
          (append(UpperBound, ValuePredict_Vector[Size]))
      }

      ValuePredict_Naive <-
        as.numeric(
 tail(
Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window[,-1],1))

      #Getting true value and last known values
      Window_Length <-
        dim(Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window)[1]
      ValueTrue <-
        as.numeric(
   Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_future[, -1])
```

```
    ValueLastKnown <-
      as.numeric(
 tail(
Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window[, -1], n =
                         1))


    if (OneVsAll) {
      Category <- factor(c(PivotGroup, "other", "tsum"))
    }
    else{
      Category <- factor(c("1", "2", "tsum"))
    }

  }
  ###No TSPACE
  else{
    Window_Length <- length(TimeSeriesValue_Window)

    Model <- ar(TimeSeriesValue_Window, aic = F, order.max = 1)
    ValuePredict <-
  predict(Model, TimeSeriesValue_Window, n.ahead = PredictionStep)
    Size = 1

    #Initialising result vectors
    ValuePredict_Vector <-matrix(data = NA,nrow = 1,ncol = Size)
    LowerBound_Vector<- UpperBound_Vector <- vector(mode="numeric",length=Size)

    #Filling up result vectors
    for (i in 1:Size) {
      ValuePredict_Vector[1, i] <-  ValuePredict[[1]][[i]][PredictionStep]
      LowerBound_Vector[i] <-  ValuePredict[[1]][[i]][PredictionStep+1]
      UpperBound_Vector[i] <-  ValuePredict[[1]][[i]][PredictionStep+2]
    }


    TSum <-
 as.numeric(
 tail(
Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window$tsum, 1))

    #Back transformations

    ValuePredict <- ValuePredict_Vector %>%
```

```
      matrix(nrow = 1) %>%
      D2invPC()
    ValuePredict <- ValuePredict * TSum


    LowerBound <- LowerBound_Vector %>%
      matrix(nrow = 1) %>%
      D2invPC()
    LowerBound <- as.vector(LowerBound * TSum)


    UpperBound <- UpperBound_Vector %>%
      matrix(nrow = 1) %>%
      D2invPC()
    UpperBound <- as.vector(UpperBound * TSum)


    ValueTrue <-
      as.numeric(
 Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_future[, -c(1, 4)])
    ValueLastKnown <-
      as.numeric(
 tail(
Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window[, -c(1, 4)], n =
                         1))

    if (OneVsAll) {
      Category <- factor(c(PivotGroup, "other"))
    }
    else{
      Category <- factor(c("1", "2"))
    }

    ValuePredict_Naive <-
      as.numeric(
 tail(
Data_NoTransformWindow[[WindowIndex]]$timeSeriesValue_window[, -c(1, 4)], 1))
   }

   ValuePredict <- round(as.numeric(ValuePredict))
   PredictionError <- as.numeric(ValuePredict - ValueTrue)
   PredictionError_Naive <- as.numeric(ValuePredict_Naive - ValueTrue)

   #Calculating the normed prediction error
```

```
PredictionError_Normed <- PredictionError


if(OneVsAll){

  return(
    list(
      prediction = data.frame(
        predictionError = PredictionError,
        predictionError_naive = PredictionError_Naive,
        predictionError_normed = PredictionError_Normed,
        valuePredict = ValuePredict,
        lowerBound = LowerBound,
        upperBound = UpperBound,
        valueTrue = ValueTrue,
        valueLastKnown = ValueLastKnown,
        valuePredict_naive = ValuePredict_Naive,
        category = Category,
        date = Date,
        pivotGroup = PivotGroup,
        window = WindowIndex,
        window_length = Window_Length,
        window_baseLength = Frame
      ),
      model = Model
    )
  )
}
else {
  return(
    list(
      prediction = data.frame(
        predictionError = PredictionError,
        predictionError_naive = PredictionError_Naive,
        predictionError_normed = PredictionError_Normed,
        valuePredict = ValuePredict,
        lowerBound = LowerBound,
        upperBound = UpperBound,
        valueTrue = ValueTrue,
        valueLastKnown = ValueLastKnown,
        valuePredict_naive = ValuePredict_Naive,
        category = Category,
        date = Date,
        window = WindowIndex,
```

```
            window_length = Window_Length,
            window_baseLength = Frame
          ),
          model = Model
          )
        )
      }
      })


  Result_Prediction <- bind_rows(UnlistListElement(PredictionResult,
 "prediction"))
  Result_Model <- UnlistListElement(PredictionResult,"model")
  names(Result_Model) <-
   sapply(c(1:length(Result_Model)),function(i){paste("window",i,sep="")})

  #Calculation the normed prediction error
  for (catg in unique(Result_Prediction$category)) {
    x <- Result_Prediction %>% filter(category==catg)

    div <- sapply(c(1:dim(x)[1]), function(i) {
      return(Normation(
        x = x$valueTrue[1:i],
        y = x$valueLastKnown[1:i]
      ))
    })
    if (0 %in% div)
      div[div == 0] <- 0.5
    Result_Prediction[Result_Prediction$category == catg,
"predictionError_normed"] <-
      Result_Prediction[Result_Prediction$category == catg,
"predictionError_normed"] / div


  }


  return(list(result = Result_Prediction,
              model= Result_Model))
}

Coda.Analysis<-function(Data_Raw, Id,
Frame=10,
ZeroHandling = "zeros_only",
```

```
PredictionStep = 1,
Log = T,
                        TSpace = T, OneVsAll = F ,
PivotGroup = c("1"),
HistoryLength = 1,
                        ModelType = "coda",
WindowMethod ="extending") {

  stopifnot(ModelType %in% c("coda","coda_OneVsAll"))

  #one vs all for all pivot groups

  #Only return IDs with results
  Id_Result <- Id

  if(OneVsAll) {

  PredictionResult_AllIDAllPivotGroup <-
  lapply(Id,function(Id_RunVariable){

    print(paste("Calculating for ID:",Id_RunVariable))
    PredictionResult_AllPivotGroup <-
  lapply(PivotGroup,function(PivotGroup_RunVariable){

        #Preparing raw data
        Data_Prepared <- Data_Raw %>%
          filter(fridge_id == Id_RunVariable &
                   main_category_id %in% c(1, 2, 3, 4)) %>%
          dplyr::select(week_date, main_category_id, sold) %>%
          arrange(week_date)

        #Calculating the length of the timeseries
        TimeSeries_Length <-
  length(unique(Data_Prepared$week_date))


        #Preparing transformed data
        Data_Transform <-
  Coda.DataPreparation(Data_Prepared,
ZeroHandling = ZeroHandling,
TSpace = TSpace,
Log = Log,
OneVsAll = T,
PivotGroup = PivotGroup_RunVariable,
```

```
HistoryLength = HistoryLength) %>%
          arrange(week_date)



        #If the Frame is given as a fraction, calculate the absolute length.
#We set 5 as the minimum length needed.
        Frame_Help <- "fixed"
        if(dim(Data_Transform)[1]<5){
          return(NA)
          }
        if(Frame < 1){
          Frame_Help <- as.character(Frame)
          Frame = round(Frame*dim(Data_Transform)[1])
          if(Frame < 4){
            Frame = 4
          }
        }


        #Splitting transformed data into windows
        Data_TransformWindow <- Data.Window(Data_Transform,
                                            Frame=Frame,
                                            Method = WindowMethod,
                                            PredictionStep = PredictionStep)



        #Preparing non transformed data
        Data_NoTransform <- Coda.DataPreparation(Data_Prepared,
ZeroHandling="none",
TSpace=TSpace,
Log=F,
OneVsAll = T,
PivotGroup = PivotGroup_RunVariable,
HistoryLength = HistoryLength) %>%
          arrange(week_date)

        #Splitting non transformed data into windows
        Data_NoTransformWindow <- Data.Window(Data_NoTransform,
                                              Frame=Frame,
                                              Method = WindowMethod,
                                              PredictionStep = PredictionStep)
```

```
        #Carrying out model fitting and prediction
        PredictionResult <-
  Coda.Prediction(Data_TransformWindow = Data_TransformWindow,
Data_NoTransformWindow = Data_NoTransformWindow,
Data_NoTransform = Data_NoTransform,
PredictionStep = PredictionStep,
OneVsAll = T,
TSpace = TSpace,
Log =  Log,
PivotGroup = PivotGroup_RunVariable,
Frame = Frame)

        PredictionResult$result$id <- Id_RunVariable
        PredictionResult$result$windowMethod <- WindowMethod
        PredictionResult$result$model <- ModelType
        PredictionResult$result$zeroHandling <- ZeroHandling
        PredictionResult$result$frame <- Frame_Help
        PredictionResult$result$history <- as.character(HistoryLength)
        PredictionResult$result$timeseriesLength <- as.character(TimeSeries_Length)


        #Tidying up data
        Result_Prediction <- PredictionResult$result
        Result_Model <- PredictionResult$model

        return(list(result = Result_Prediction,
                    model = Result_Model))

    })

     #Removing NA (aka Timeseries which are too short)
     PredictionResult_AllPivotGroup <-
  PredictionResult_AllPivotGroup[!is.na(PredictionResult_AllPivotGroup)]

     if(length(PredictionResult_AllPivotGroup)==0){
       print(paste("Insufficient data. Skipping ID: ",Id_RunVariable))
       Id_Result <<- Id_Result[Id_RunVariable!=Id_Result]
       return(NA)
     }

     #Tidying up data
     Result_Prediction <-
  bind_rows(UnlistListElement(PredictionResult_AllPivotGroup,"result"))
```

```
      Result_Prediction$id <- as.factor(Result_Prediction$id)
      Result_Model <- UnlistListElement(PredictionResult_AllPivotGroup,"model")
      names(Result_Model) <- PivotGroup

      return(list(result = Result_Prediction,
                  model = Result_Model))


    })

    #Removing NA (aka Timeseries which are too short)
  PredictionResult_AllIDAllPivotGroup <-
  PredictionResult_AllIDAllPivotGroup[!is.na(
PredictionResult_AllIDAllPivotGroup)]

    if(length(PredictionResult_AllIDAllPivotGroup)==0)return(NA)


    #Tidying up data
    Result_Prediction <-
  bind_rows(UnlistListElement(PredictionResult_AllIDAllPivotGroup,
  "result"))
    Result_Model <-
  UnlistListElement(PredictionResult_AllIDAllPivotGroup,"model")
    names(Result_Model) <- Id_Result

    return(list(result = Result_Prediction,
                model = Result_Model))

  }
  #Not one vs all
  else {

    PredictionResult_AllID <- lapply(Id,function(Id_RunVariable){
      print(paste("Calculating for ID:",Id_RunVariable))
        #Preparing raw data
        Data_Prepared <- Data_Raw %>%
          filter(fridge_id == Id_RunVariable &
                   main_category_id %in% c(1, 2, 3, 4)) %>%
          dplyr::select(week_date, main_category_id, sold) %>%
          arrange(week_date)

        #Calculating the length of the timeseries
        TimeSeries_Length <- length(unique(Data_Prepared$week_date))
```

```
        #Preparing transformed data
        Data_Transform <-
  Coda.DataPreparation(Data_Prepared,
ZeroHandling = ZeroHandling,
TSpace = TSpace,
Log = Log,
OneVsAll = F,
HistoryLength = HistoryLength) %>% arrange(week_date)


        #If the Frame is given as a fraction, calculate the absolute length.
#We set 5 as the minimum length needed.
        Frame_Help <- "fixed"
        if(dim(Data_Transform)[1]<5){
          return(NA)
        }
        if(Frame < 1){
          Frame_Help <- as.character(Frame)
          Frame = round(Frame*dim(Data_Transform)[1])
          if(Frame < 5){
            Frame = 5
          }
        }


        #Splitting transformed data into windows
        Data_TransformWindow <- Data.Window(Data_Transform,
                                            Frame= Frame ,
                                            Method = WindowMethod,
                                            PredictionStep = PredictionStep)



        #Preparing non transformed data
        Data_NoTransform <-
  Coda.DataPreparation(Data_Prepared,
ZeroHandling="none",
TSpace=TSpace,
Log=F,
OneVsAll = F,
HistoryLength = HistoryLength) %>% arrange(week_date)

        #Splitting non transformed data into windows
```

```
        Data_NoTransformWindow <- Data.Window(Data_NoTransform,
                                              Frame = Frame,
                                              Method = WindowMethod,
                                              PredictionStep = PredictionStep)


        PredictionResult <-
  Coda.Prediction(Data_TransformWindow = Data_TransformWindow,
  Data_NoTransformWindow = Data_NoTransformWindow,
Data_NoTransform = Data_NoTransform,
PredictionStep = PredictionStep,
OneVsAll = F,
TSpace = TSpace,
Log =  Log,
Frame = Frame)

        PredictionResult$result$id <- Id_RunVariable
        PredictionResult$result$WindowMethod <- WindowMethod
        PredictionResult$result$model <- ModelType
        PredictionResult$result$ZeroHandling <- ZeroHandling
        PredictionResult$result$frame <- Frame_Help
        PredictionResult$result$history <- as.character(HistoryLength)
        PredictionResult$result$timeseriesLength <-
  as.character(TimeSeries_Length)

        #Tidying up data
        Result_Prediction <- PredictionResult$result
        Result_Model <- PredictionResult$model

        return(list(result = Result_Prediction,
                    model = Result_Model))

    })

    #Removing NA (aka Timeseries which are too short)
    PredictionResult_AllID <-
  PredictionResult_AllID[!is.na(PredictionResult_AllID)]

    if(length(PredictionResult_AllID)==0){
      print(paste("Insufficient data. Skipping ID: ",Id_RunVariable))
      Id_Result <<- Id_Result[Id_RunVariable!=Id_Result]
      return(NA)
    }
```

```
    #Tidying up data
    Result_Prediction <-
  bind_rows(UnlistListElement(PredictionResult_AllID,"result"))
    Result_Model <- UnlistListElement(PredictionResult_AllID,"model")
    names(Result_Model) <- Id_Result

    return(list(result = Result_Prediction,
                model = Result_Model))

  }

}
```