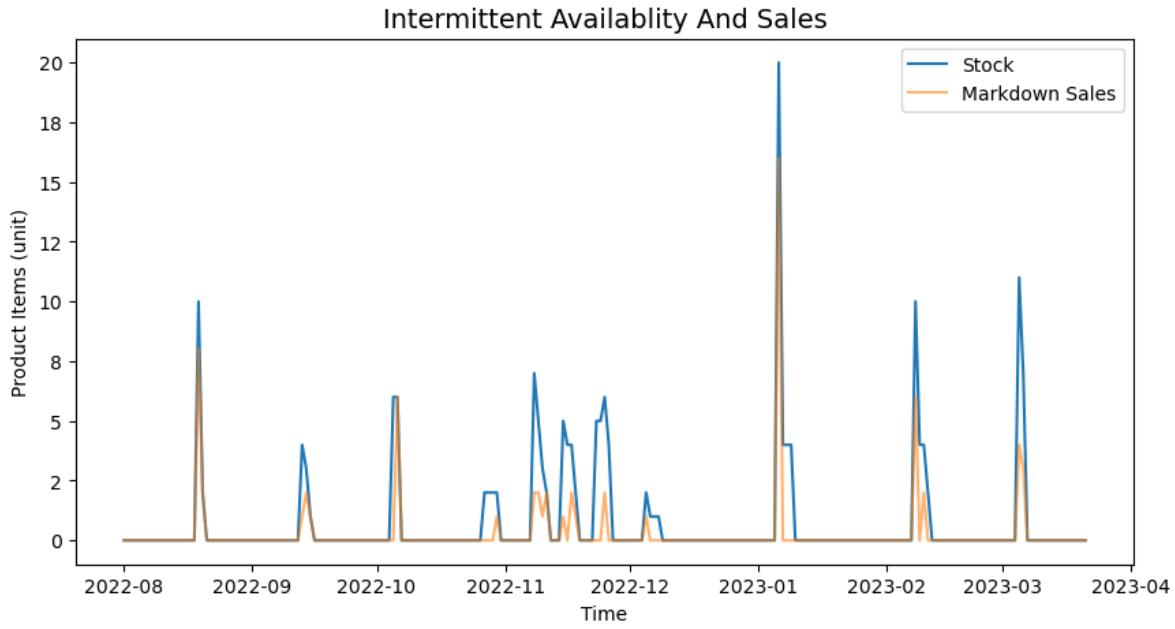


Sparse Time Series Demand Forecasting for Intermittent Availability

A deep learning solution using Temporal Fusion Transformers for marked-down perishable products with a limited shelf life

Master's thesis in Mathematical Sciences

Oscar Helgesson
Norbert Laszlo

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Sparse Time Series Demand Forecasting for Intermittent Availability

A deep learning solution using Temporal Fusion Transformers for marked-down perishable products with a limited shelf life

OSCAR HELGESSON

NORBERT LASZLO

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Sparse Time Series Demand Forecasting for Intermittent Availability
A deep learning solution using Temporal Fusion Transformers for marked-down perishable products with a limited shelf life

Oscar Helgesson
Norbert Laszlo

Supervisor: Emil Carlsson, Data Science och AI
Examiner: Axel Ringh, Applied Mathematics and Statistics

Master's Thesis 2023
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Intermittent availability (stock) and markdown sales over time of a sample product from the dataset.

Typeset in LaTeX
Gothenburg, Sweden 2023

Sparse Time Series Demand Forecasting for Intermittent Availability
A deep learning solution using Temporal Fusion Transformers for marked-down perishable products with a limited shelf life

Oscar Helgesson
Norbert Laszlo
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis addresses the challenge of forecasting sales for individual perishable markdown products using historical sales data and other relevant features in the form of time series. The target time series, i.e., daily sales of marked-down units of a certain product, is intermittent, sparse, and highly irregular, and sales can only occur if the marked-down product is available. To solve the problem, various methods were evaluated, ranging from well-established statistical models to newer deep learning-based models. This thesis proposes an interpretable novel method that improves the Temporal Fusion Transformer model with cluster encodings by applying random convolutional kernel transformations to time series.

The study found that the compared deep learning models outperformed the baseline statistical models, particularly the RNN and Temporal Fusion Transformer. The novel approach of clustering the markdown series based on markdown features showed no significant change in performance regarding day-to-day prediction. However, it did show a significant improvement in multi-horizon aggregated predictions. Moreover, using clustering resulted in decreased time in training the models.

Overall, the results suggest that deep learning models and the Temporal Fusion Transformer with added cluster encodings are promising models for predicting intermittent series with known available inventory. This study has practical implications for retailers and businesses that sell perishable products. Accurately forecasting sales of markdown products can help reduce waste and optimize inventory management, resulting in cost savings and increased profitability.

Keywords: Time Series Forecasting, Intermittent Time Series, Intermittent Forecasting, Machine Learning, Deep Learning, Temporal Fusion Transformer, Time Series Clustering, ROCKET

Acknowledgements

Firstly, we would like to express our sincere gratitude to our academic supervisor Emil Carlsson for his guidance, continuous support, and expert input throughout the project. Secondly, we would also like to thank Axel Ringh, our Chalmers examiner, for his expert input and thoughtful feedback on the thesis, contributing to a better final result.

Lastly, to all those individuals who have directly or indirectly contributed to the realization of this project, we extend our heartfelt appreciation.

Oscar Helgesson and Norbert Laszlo, Gothenburg, June 2023

Contents

List of Figures

List of Tables

1

Introduction

Forecasting future sales and demand is an essential practice for organizations in the business of buying and selling goods. Predicting future sales can help businesses optimize their operations, yield increased revenue, reduce waste, and more. This type of forecasting is most often done by using historical sales data up to some point in time (a time series) to estimate future values.

Methods for forecasting future sales have evolved from using human judgment and experience to later statistical models, and now in today's age, machine learning (ML) based models. The continuous advances in time series forecasting have enabled more challenging and niche instances of forecasting problems to be solved. However, although there are more capable models and tools today, some problems remain challenging. One such problem is instances when the available data over time is sparse, resulting in intermittent time series that are periodically zero.

Intermittent demand is a phenomenon in time series forecasting where the demand is often non-existent and, therefore, null for periods until it spikes as an effect of some event. These periods of no demand in the time series make them non-continuous, thus harder to forecast. Intermittent time series are naturally sparse and often irregular in how they spike, making them even more challenging. The irregularities introduce the additional problem of knowing when there is a demand, not only the magnitude of the demand.

This thesis will explore some of the challenges and potential solutions of forecasting intermittent sparse time series. The methods this thesis evaluates for this area range from well-researched statistical methods to newer deep learning-based models, of which available research in this area is limited. This thesis additionally proposes a novel method for improving a Temporal Fusion Transformer model with cluster encodings by applying random convolutional kernel transformations to time series.

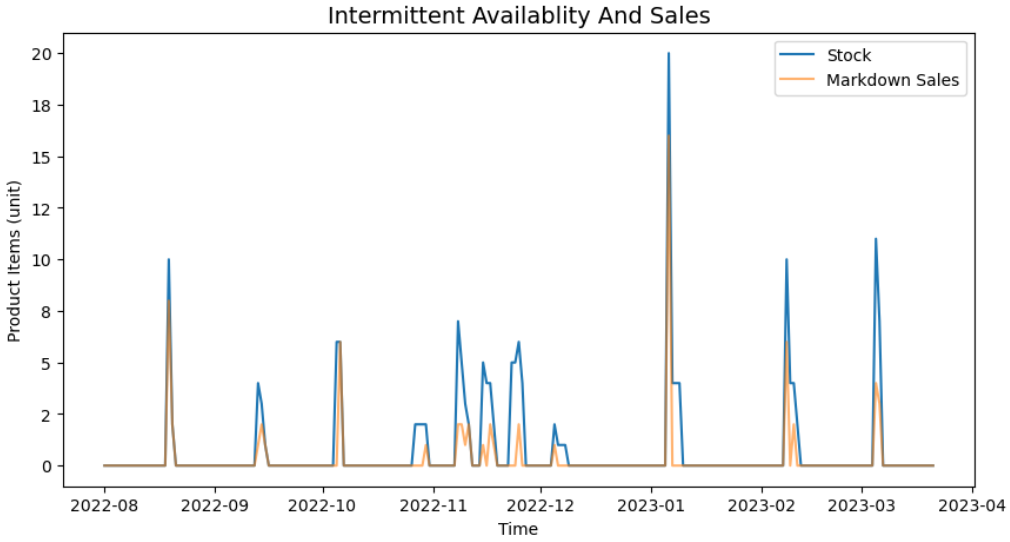


Figure 1.1: Intermittent availability (stock) and markdown sales over time.

1.1 Background

In sales forecasting, intermittent demand for a product to be forecasted can be actualized by having intermittent availability. If there is no availability, no matter how high the demand, no sale can ever be made. Availability is thus a prerequisite for the actualized demand and sales to be non-zero. However, this does not guarantee that any sales will be made as the sales of products may depend on many other variables.

Perishable products are a particularly interesting segment of products from a sales forecasting perspective. Perishable products are products with a finite (limited) shelf-life. This makes forecasting future sales of these items particularly interesting, as their supply will inevitably drop even without demand as they will become waste [1], making accurate prediction more critical. Examples of perishable products could be food such as meat or dairy products. It could also include prescription drugs and airline tickets [2]. In all of these product segments, the end of shelf-life (best-by-date) is known from the start.

Since perishable products have a finite lifespan, their demand naturally drops as they grow closer to their best-by-date. This creates an incentive to discount them to manage their demand and avoid waste. This action will be referred to as performing a markdown, i.e., when a batch of items of a particular perishable product is selected and applied a discount until expiration. This action creates a new availability as a marked-down version of that product. Thus, for every markdown instance, an availability window in time is created for that marked-down product. Repeatedly doing this creates a time series with intermittent demand due to its underlying intermittent availability. Thus, every product target for markdowns will have two time series representing its sales. One time series for the product's ordinary sales with continuous demand, and another for the product's markdown sales with intermittent

demand. Figure 1.1 shows the intermittent availability and sales of a markdown product created by a finite stock that is sparsely positive.

Forecasting the sales of perishable markdown products makes an interesting area of research as it is a special case of intermittent sales forecasting that depends on intermittent availability. This intermittent availability makes many of the more traditional methods used for time series forecasting unfitting for the purpose, as they rely on continuity. Therefore, more complex solutions may be required to solve this type of forecasting.

1.2 Problem Formulation

This thesis aims to forecast the future sales of individual perishable markdown products using their time series. The time series representing the number of sales of a perishable marked-down product will be referred to as the target time series.

The challenges that characterize the target time series are that they are intermittent, sparse, and often highly irregular. Furthermore, sales in the target time series can only occur if there is an availability in time, that is, a positive (non-zero) marked-down inventory of the product that the target time series represents. However, this occurs only when the product has an active markdown. During such an active markdown, we know the number of product items in the marked-down inventory, the discount applied to these product items, and the best-by-date, marking the date which after the items cannot be sold. Formally put, the problem is the following: given an inventory of a known quantity of product items, a discount, and an interval in time, approximate the daily number of future sales of a marked-down product in the given interval. Additionally, aggregating the daily predictions for the markdown interval yields a prediction for how many in a batch of markdown items sell in the given interval.

This thesis aims to find suitable methods to solve the defined problem and empirically evaluate these methods against each other to present what we believe to be the best solution available. Additionally, we present and discuss the strengths and weaknesses of the different solutions in different scenarios.

1.3 Limitations

The scope of this thesis is restricted to exploring viable methods for forecasting intermittent time series in the domain of perishable products. Although some findings may extend beyond the domain of perishable products, the focus will be centered on this domain. To explore which methods might prove viable, this thesis will focus on evaluating one state-of-the-art forecasting method against baseline methods of differing levels of complexity that have historically been proven to work for time series forecasting in the general and intermittent domain. Furthermore, since the length of a future availability window is known when forecasting the markdown sales

of the products, tackling the challenge of predicting availability windows will not be covered, as would be in the general case of intermittent time series forecasting. Instead, the focus lies on evaluating methods for forecasting demand for periods when the availability window is known.

1.4 Thesis Outline

The remainder of this thesis is divided into six chapters, Theory, Data Exploration, Methods, Results, Discussion, and Conclusion.

The Theory chapter introduces the theoretical concepts necessary to comprehend information presented in the subsequent chapters and helps gain a deeper appreciation for the problem this thesis aims to solve. The chapter covers critical concepts related to time series forecasting and intermittent data, statistical and machine learning models used later in the thesis, and the tools used to evaluate the results.

Chapter 3, Data Exploration, presents the dataset and evaluates and presents significant data properties needed for forecasting. Next, Chapter 4, Method, presents the essential techniques and methods used to solve the defined problem partially based on Chapter 3. First, the preprocessing methods and steps applied to the data are described, after which the most relevant details about the models' implementations are presented. Lastly, a thorough description of how the models are evaluated, both individually and relative to each other, is presented.

In Chapter 5, the results of the models' evaluation are presented, both how the models perform individually and compared to each other. Then, the proposed cluster-based encoding results are interpreted, and its effect on the Temporal Fusion Transformer is presented. Lastly, the chapter further reveals details about how the Temporal Fusion Transformer model makes predictions, giving more insight into how the model works.

The Discussion chapter discusses the results of the models' evaluations and how these results can be interpreted. Also, the problem's challenges, which models are more viable for the outlined problem, and why are also discussed. The chapter ends with several further research propositions identified during the thesis and the ethical considerations that have been taken into consideration related to the work of the thesis.

Lastly, in Chapter 7, the key takeaways of the challenges of the problem are presented together with the conclusions that can be drawn from the evaluations and their results.

2

Theory

This chapter contains the theory needed to comprehend the methods used and the results achieved. The chapter begins by introducing important concepts related to time series and demand forecasting. Then, two standard statistical models used for time series forecasting are presented. Next to last, the inner workings of Artificial Neural Networks and Transformer models are described to give an appreciation of how the more complex forecasting models work. Finally, the Temporal Fusion Transformer is presented, the primary model that will be evaluated in this thesis.

2.1 Time-series

Time series is a form of temporal data, representing a collection of values obtained in a sequential manner over time, where the data is often large in size and is updated continuously [3], [4]. Time series are often defined as:

Definition 2.1. *Univariate Time Series*: A *time-series* is an ordered sequence of n real-valued variables

$$\boldsymbol{y} = (y_1, ..., y_n), \qquad y_i \in \mathbb{R}.$$

When a time-series is updated, this then results in a new time-series of size $n + 1$. Figure 2.1 shows an example of a univariate time series. Time series can also be multivariate, where multiple univariate time series (vectors) are stacked together to create a multivariate time series (matrix). In such a case, a m-variable time series would be denoted:

Definition 2.2. *Multivariate Time Series*: A m-dimensional *time-series* is an ordered sequence of n m-dimensional vectors

$$\mathbf{Y} = (\boldsymbol{y_1}, ..., \boldsymbol{y_n}), \qquad \boldsymbol{y_i} \in \mathbb{R}^m.$$

Time series may also contain patterns such as *trend* and *seasonality*. Two key time series concepts that are often included when decomposing a time series into three

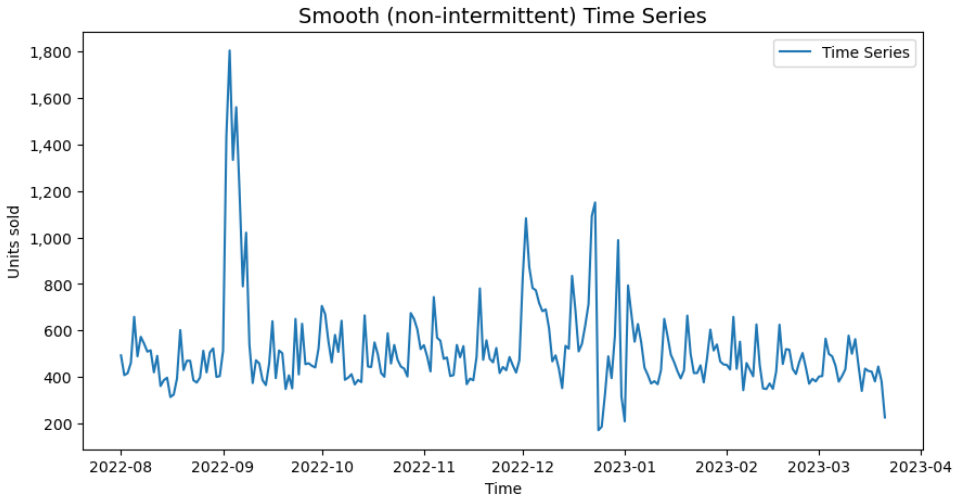


Figure 2.1: An example smooth (non-intermittent) univariate time series.

components are the *trend*, T_t component, the *seasonal* S_t component, as well as the *error* ϵ_t component[5, Section 3.2]. The easiest way of decomposing it is through the *additive* decomposition:

$$y_t = T_t + S_t + \epsilon_t.$$

The simple model simplifies how a time series can be constructed, and often, there are more components to a series. Also, including such components can aid in reducing the error component ϵ_t and better facilitate understanding a series.

2.1.1 Trends & Seasonality

Many business-related and economic time series exhibit what is known as trend or seasonal variations. There are also different types of seasonality caused by factors such as weather, yearly holidays, and even the behavior of the agents in the system [6].

Additionally, time series can also contain trends. The trend can be seen as a long-term increase or decrease in the time series values. These trends, especially in combination with seasonality, can substantially affect various forecasting models and methods. See Figure 2.2 for an illustration of trends and seasonality in a time series. Time series data with trends are non-stationary, a key assumption in accurate forecasting (see Section 2.2.1). Therefore, being able to model seasonality and trend variations in the data plays a significant part in being effective in forecasting in many domains [6].

2.2 Demand Forecasting

Demand forecasting is a pivotal type of predictive analysis used to predict and understand customer demand in supply chain management [7]. Being able to predict the demand allows a large variety of industries to remove bottlenecks and allow for

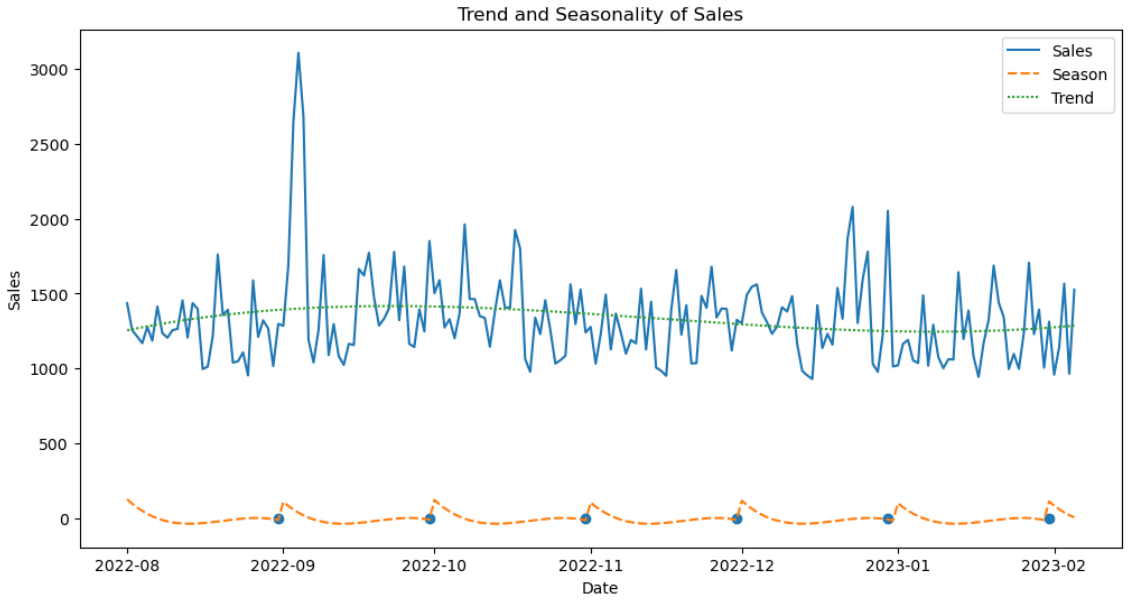


Figure 2.2: A plot depicting an example of a time series from the data set used in this thesis (blue), the corresponding trend series (green), and the seasonality (e.g. here shown on a monthly frequency) where each blue dot represents the salary payday that month.

efficient use of resources and inventory management that also enhances customer experience. The data generated from historical sales often used as a proxy for historical customer demand, can be analyzed and used to attempt to forecast future demand [8], [9]. Historically, forecasting of sales has been conducted on a judgmental basis, but in later times the historical sales data has been used to, in some way, predict it automatically.

Demand forecasting is often done using time-series models, thus viewing the historical data as temporal. One can thus use this fact to predict future demand for multiple time steps into the future (multi-horizon forecasting). However, one can restrict the forecast to only predict the following value for the next time step in the order rather than predicting the values for multiple time steps [9].

Different industries, based on specific requirements, use different techniques to forecast. These techniques were initially limited to statistical models that aimed to estimate the trend and seasonality parameters as described in 2.1 (T_t and S_t). In many types of problems, this is still sufficient. However, more complex machine learning (ML) and deep learning (DL) models have become more prevalent for more complex problems in later years [10].

2.2.1 The Demand Forecasting Problem

The time series demand forecasting problem is to predict future values of a target series $\boldsymbol{y}_i$ for a given time series entity i. The simplest form of the problem is *one-*

step-ahead: $\hat{y}_{i,t+1} = f(y_{i,t-k:t}, \boldsymbol{x}_{i,t-k:t})$, where $\hat{\boldsymbol{y}}_i$ denotes the values being predicted. Here, the value of the series $\boldsymbol{y}_i$ is predicted at time step $t+1$ given the series from k steps back $(t-k)$ up to the immediately preceding time step t. A feature vector $\boldsymbol{x}$ can also be used to help with forecasting the target. Depending on the specific problem at hand, this vector can be temporal, static, or both. $f(\cdot)$ is the prediction function for the model. The goal is to find such a function $f(\cdot)$ such that it minimizes the error in the predictions.

To do so, an important concept is that of *stationary*. A series that upholds stationarity is called a stationary process. Then, it is a process where the statistical properties of the process do not change over time. Therefore, if signs of trends are present in the data, these will have to be removed to achieve a stationary process. Stationary can be defined in different ways [11, Section 6.5]; however, a standard (quite informal) definition is that a time series is stationary if the following conditions are met: Constant mean μ and variance σ^2 for each timestep $t \in T$, and that the autocovariance function between y_{t_1} and y_{t_2} depend only on the lag between t_1 and t_2. To test whether a series is stationary, a standard test is the Augmented Dickey-Fuller Test (ADF-Test), which is a statistical hypothesis test developed by Dickey and Fuller in 1979 [12].

Another key concept is that of *auto-correlation*. Auto-correlation defines how the temporal data correlates with lagged instances of itself. A lag of k means comparing time step t with instances at k time steps back. The key is that series with specific lags might correlate more strongly than other lags, e.g., a lag of seven might indicate recurring weekly correlation (seasonality). Auto-correlation (r_k) can be defined as

$$r_k = \frac{\sum_{t=k+1}^{N}(\boldsymbol{y}_t - \hat{\boldsymbol{y}})(\boldsymbol{y}_{t-k} - \hat{\boldsymbol{y}})}{\sum_{t=1}^{N}(\boldsymbol{y}_t - \hat{\boldsymbol{y}})^2}.$$

Evaluating the auto-correlation at different lags can highlight the existence of patterns, such as seasonality in the data. Moreover, this definition of auto-correlation requires a stationary process to be properly used. Therefore, if it is found that the data is non-stationary, it will have to be transformed to make it stationary.

Another key aspect in time series forecasting is Granger causality. Granger causality is a statistical method used in time series forecasting to assess whether one time series provides useful information in predicting another [13]. This approach determines whether past values of one time series ($\boldsymbol{x}$) significantly improve the forecast accuracy of another time series ($\boldsymbol{y}$) beyond what can be achieved using its own ($\boldsymbol{y}$) past values. Granger causality is particularly beneficial in modeling complex systems where multiple variables interact over time. The Granger causality measure can therefore help identify relationships between variables, which can help to enhance the accuracy of multi-variable time series forecasting models.

2.2.2 Intermittent Demand Forecasting

Intermittent demand series (also known as *sporadic demand series* and *count series*) is often described as a time series where the inter-demand arrival interval is long, and

the quantity of the demand is low [14]. Comparatively, a more traditional demand time series with short inter-demand arrival intervals and low demand is known as smooth.

Intermittent time series forecasting is considered among the most challenging forecasting problem there is [15], [16]. The intermittent demand time series forecasting presents two problems [17]:

1. When will the next period of demand occur?
2. When there is demand, what will be the demand volume?

Of these two problems, (1) is a non-existent problem in traditional time series forecasting, while (2) is generally harder to solve for intermittent time series. This is partially due to two consequences of the long periods of zero demand. Firstly, due to zero-demand periods, the *active* periods are often separated from previous active periods, making it generally more challenging to learn how these periods affect each other. Secondly, the sporadic nature of the demand arrival times further increases the problem's difficulty compared to if the demand arrived with a fixed interval.

Typical products described via an intermittent time series are spare parts, heavy machinery, and electronics [18]. Being unable to successfully predict such items as a consequence leads to a loss of revenue or increased stock-keeping costs. In the case of marked-down perishable products, being unable to forecast future demand successfully either leads to a loss of revenue or wasted products (as the items cannot be stored forever, unlike spare parts). A typical intermittent time series might look as seen in Figure 2.3.

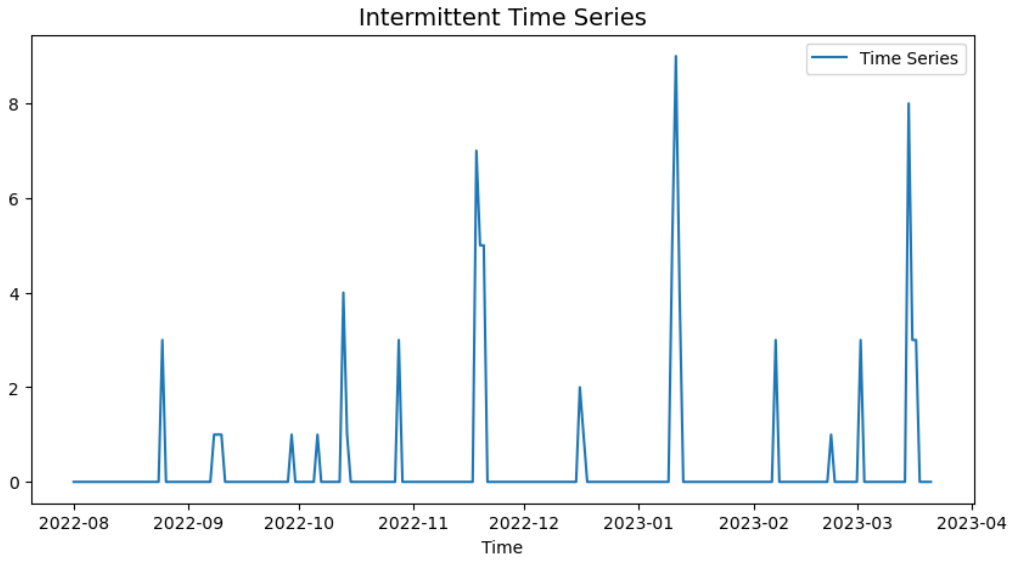


Figure 2.3: Example of an intermittent time series.

2.3 Statistical Models For Time Series Forecasting

In this section, two commonly used statistical models in the field of time series forecasting are presented. The models are Exponential Smoothing and Croston's model, which is an extension of Exponential Smoothing.

2.3.1 Exponential Smoothing

Exponential Smoothing is a *class* of forecasting methods based on using weighted combinations over past observations of a target time series to predict future values [19, Section 1.3]. The original idea was introduced in 1944, but improvements have been developed and published during most of the 20th century and in the early 21st century.

The most simple form of the Exponential Smoothing formula, often shortened *SES* for *Simple Exponential Smoothing*, is given by [20, Section 6.4.3.1]:

$$\begin{aligned} S_0 &= y_0 \\ S_t &= \alpha y_{t-1} + (1-\alpha)S_{t-1}, \quad t > 0 \end{aligned}$$

where $\boldsymbol{S}$ is the *smoothed target series*, $\boldsymbol{y}$ is the target series, and $0 < \alpha < 1$ is known as the *smoothing factor*. This model was developed by Holt to predict future values of time series showing no trend or seasonality. Thus, both *double* and *triple* Exponential Smoothing was also developed, taking trends, as well as trends and seasonality, respectively, into consideration. As allowing for no trends, or seasonality is a limiting factor, triple Exponential Smoothing is used as a baseline in this thesis. There, the forecast is produced via a combination of target, trend, and seasonal smoothing as follows [20, Section 6.4.3.5]:

$$\begin{aligned} S_t &= \alpha\frac{y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} + b_{t-1}) && \text{Target Smoothing} \\ b_t &= \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1} && \text{Trend Smoothing} \\ I_t &= \beta\frac{y_t}{S_t} + (1-\beta)I_{t-L} && \text{Seasonal Smoothing} \\ \hat{y}_{t+m} &= (S_t + mb_t)I_{t-L+m}. && \text{Forecast} \end{aligned}$$

Here,

- $\boldsymbol{y}$ is the observed target series
- $\boldsymbol{S}$ is the smoothed observation
- m is the forecast horizon
- L is the seasonality period
- $\boldsymbol{b}$ is the trend factor
- $\boldsymbol{I}$ is the seasonal index
- $\hat{\boldsymbol{y}}$ is the forecast at m periods ahead
- t is an index denoting a time period
- α, β, and γ are constants that need to be estimated to minimize the MSE.

2.3.2 Croston's Method

Intermittent data have often been argued to differ significantly from generally smooth (normal) time series, and such, specific models have been introduced specifically for the intermittent time series [21]. One such is Croston's original method and later developed variants. Croston's idea was to model the demand for slow-moving items using two separate quantities, one for the demand size and another for the frequency of the demand (also known as the inter-demand interval) [22]. This method often outperforms exponential smoothing models in the case of intermittent demand. Essentially, the original Croston's method was a combination of two exponential smoothing models as follows:

Let us denote a as the estimate and y as the true demand. Then the following is used to estimate the demand size. α is an exponential smoothing parameter:

$$\begin{aligned} &\text{if } y_t > 0, \text{ then } a_{t+1} = \alpha y_t + (1-\alpha)a_t, \\ &\text{if } y_t = 0, \text{ then } a_{t+1} = a_t. \end{aligned}$$

Secondly, the following is used to estimate demand frequency using p for the estimated time between occurrences and q as the time elapsed since the previous demand occurrence:

$$\begin{aligned} &\text{if } y_t > 0, \text{ then } p_{t+1} = \alpha q + (1-\alpha)p_t, \\ &\text{if } y_t = 0, \text{ then } p_{t+1} = p_t. \end{aligned}$$

Using the parameters demand size a and demand frequency p, a forecast can be produced via:

$$\hat{y}_{t+1} = \frac{a_t}{p_t}.$$

Since parameters are only updated when demand is observed, the forecasts will be constant in the inter-arrival period of the demand. As this is a major limitation of the model, improvements have been made. The most notable improvement is the TSB variant that updates the periodicity parameter p even when there is no observed demand [23]. Instead, rather than viewing p as the expected number of periods between demand, TSB views it as the frequency of demand. Moreover, a separate β Exponential Smoothing parameter is introduced for demand frequency. They then change the p parameter update and prediction $\hat{y}_{t+1}$ as:

$$\begin{aligned} &\text{if } y_t > 0, \text{ then } p_{t+1} = 1 \cdot \beta + (1-\beta)p_t = \beta + (1-\beta)p_t, \\ &\text{if } y_t = 0, \text{ then } p_{t+1} = 0 \cdot \beta + (1-\beta)p_t = (1-\beta)p_t, \\ &\hat{y}_{t+1} = p_{t+1}a_{t+1}. \end{aligned}$$

2.4 Artificial Neural Networks

Artificial neural networks (ANNs) are a powerful class of machine learning models used to learn complex nonlinear relationships in large data sets. Their versatility

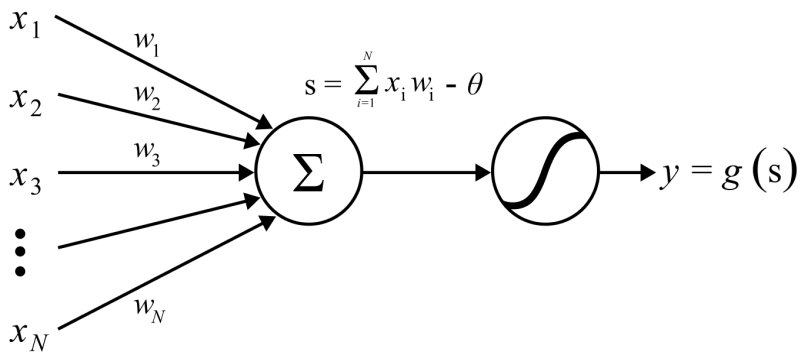


Figure 2.4: The inner working of an artificial neuron visualized with the calculations that map $y = f(\boldsymbol{x}; \boldsymbol{w}, \theta)$.

makes them fitting for many ML purposes, including sales forecasting but at the cost of higher computational complexity and, in most cases, lack of interpretability.

Formally, in supervised learning, neural networks learn through training examples to approximate an assumed true underlying model $y = f(x) + \epsilon$ where ϵ is the noise. In their simplest form, neural networks perform this approximation via the function $\boldsymbol{y} = f(\boldsymbol{x}; \mathbf{W}, \boldsymbol{\Theta})$ between some input vector $\boldsymbol{x}$ and output $\boldsymbol{y}$ that can be both a vector or a scalar value. The variables $\mathbf{W}$ and $\boldsymbol{\theta}$ are the model's inner parameters which the model adjusts during the training process to learn the relationship between the inputs and outputs. Once the model has learned to approximate this relationship, it can make predictions given an input vector $\boldsymbol{x}$. The predicted output is defined as $\hat{\boldsymbol{y}}$ to differentiate it from its actual value $\boldsymbol{y}$.

This section further describes the inner workings of neural networks and incrementally introduces more advanced neural network models that can learn from temporal data used for time series forecasting.

2.4.1 The Neuron

The neuron is the smallest building block in neural networks, and its task is to process a set of inputs to produce an output. In computational sciences, the neuron is defined as a mathematical operation between an input vector $\boldsymbol{x}$, a weight vector $\boldsymbol{w}$, bias (threshold) scalar value θ, and some activation function $g(\cdot)$. The neuron receives N inputs as $\boldsymbol{x} = (x_1, ..., x_N)^{\mathrm{T}}$ and has N weights $\boldsymbol{w} = (w_1, ..., w_N)^{\mathrm{T}}$ corresponding to each respective input x where both $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^N$. See Figure 2.4 for visual representation.

The resulting output of the neuron is calculated by the dot product between the input vector and the weight vectors and then subtracting the bias value, resulting in a scalar value that the activation function transforms into the output. The activation function can be any function that maps a real value to another $g : \mathbb{R} \to \mathbb{R}$ but is often

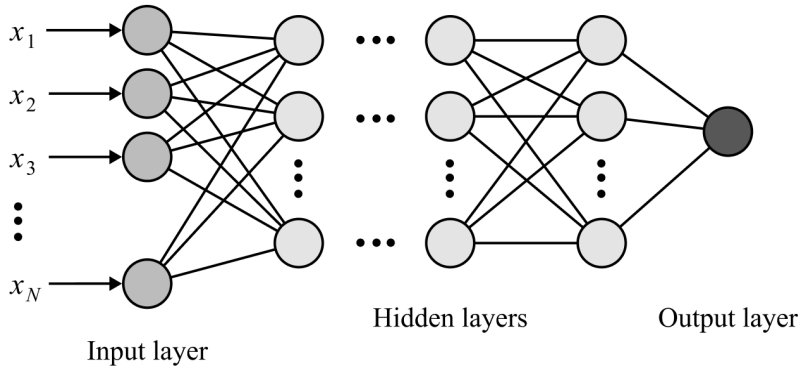


Figure 2.5: Architecture of a deep (multi-layer) neural network.

chosen with care for specific purposes. Mathematically these steps are expressed as:

$$\hat{\boldsymbol{y}} = g\left(\boldsymbol{x} \cdot \boldsymbol{w} - \theta\right) = g\left(\sum_{i=1}^{N} x_i w_i - \theta\right). \tag{2.1}$$

2.4.2 Deep Neural Networks

Deep neural networks (DNNs) are created by using a large number of neurons and stacking them in layers, enabling the network to learn more complex patterns. They work on the same premise as the single neuron, but here the output of each neuron is propagated to multiple neurons in the next layer. Figure 2.5 visualizes the multi-layer architecture of a DNN.

Training a neural network consists of two processes, forward propagation and backward propagation. During this process, the model learns the relationship between the input vectors $\boldsymbol{x}$ and output vectors $\boldsymbol{y}$. The forward propagation process is expressed mathematically as

$$\begin{aligned} \boldsymbol{V}^{(1)} &= g(\mathbf{W}^{(l)}\boldsymbol{x} + \boldsymbol{\theta}^{(1)}) \\ \boldsymbol{V}^{(l)} &= g(\mathbf{W}^{(l)}\boldsymbol{V}^{(l-1)} + \boldsymbol{\theta}^{(l)}), \quad 2 \leq l < L \\ \hat{\boldsymbol{y}} &= g(\mathbf{W}^{(L)}\boldsymbol{V}^{(L-1)} + \boldsymbol{\theta}^{(L)}) \end{aligned} \tag{2.2}$$

where $\boldsymbol{V}^{(l)}$ is the output of the l:th layer, L is the number of layers in the network, and $\mathbf{W}^{(l)}$ is the weight matrix of the l:th layer where $\mathbf{W}^{(l)} \in \mathbb{R}^{||\boldsymbol{V}^{(l)}|| \times ||\boldsymbol{V}^{(l-1)}||}$.

After the forward propagation step, the model outputs a prediction $\hat{\boldsymbol{y}}$, which is used to compute the model's error. This error is computed by the cost (loss) function $H : \mathbb{R}^{n \times n} \to \mathbb{R}^n$, which quantifies the difference between the actual value $\boldsymbol{y}$ and the prediction $\hat{\boldsymbol{y}}$. The most common cost function is the mean squared error, defined as

$$H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2N} \sum_{i=1}^{N} \left(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\right)^2 \tag{2.3}$$

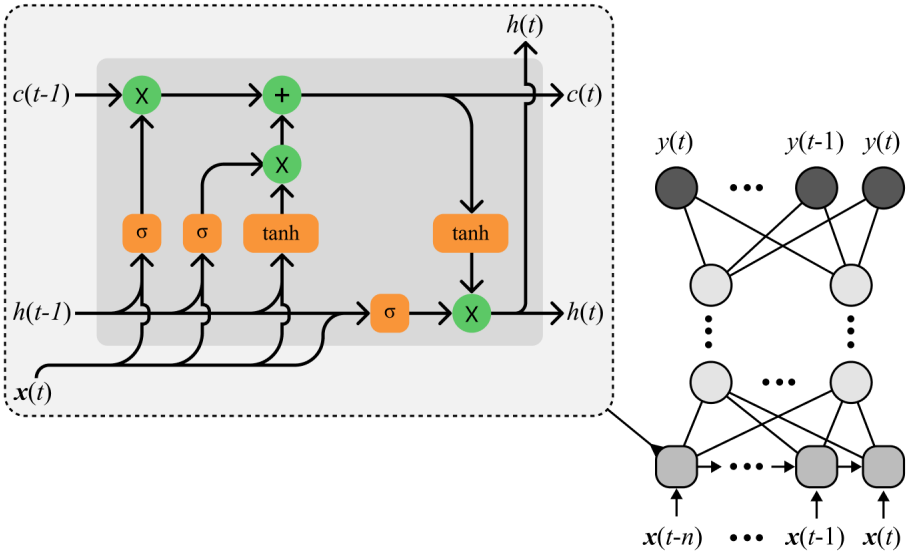


Figure 2.6: Figure visualizes the layout of an RNN together with the internal structure of an LSTM cell where $\boldsymbol{c}$ is the cell's state vector, and $\boldsymbol{h}$ is the hidden state vector, i.e., the unit-to-unit output vector.

where N is the number of training examples. The resulting cost value is used to adjust the weights and biases $(\mathbf{W}^{(l)}, \boldsymbol{\theta}^{(l)})$ of the network to lower the loss of the system. This process is done through stochastic gradient descent, which computes the gradients of the parameters in the model with respect to the cost function. This is done using the chain rule, starting from the output layer and working backward through the network. Once the gradients are computed, the parameters' values are adjusted incrementally so that the predictions get closer to their true values [24, Chapter 6].

2.4.3 Recurrent Neural Networks

For sequential learning (or temporal learning), as in demand forecasting, a specialized family of neural networks called recurrent neural networks (RNNs) is used. The main difference between the RNN-based networks and the previously introduced feed-forward type of ANNs is that RNNs support sequential learning for sequences (e.g., time series) where the values are not independent in time [25]. RNNs architecture enables learning through time by neural units that are coupled sequentially in recurrent layers of a network. These types of interlayer connections intend to mimic memory over multiple time steps. To model such connections, the regular neural units are exchanged with more complex units that can selectively pass information from unit to unit in the recurrent layers [26]. One of these types of units is the long-short-term memory (LSTM) illustrated in Figure 2.6 in the context of an RNN.

The key differentiating factor in input format with RNNs is that they take in a matrix $\mathbf{X} = \left[\boldsymbol{x}(t-n)^{\mathrm{T}}, .., \boldsymbol{x}(t)^{\mathrm{T}}\right]$ of values instead of a vector where t is the time

step index and n is the number of time steps in the input layer. Each vector

$$\boldsymbol{x}(t) = \begin{bmatrix} x(t)_1 \\ \vdots \\ x(t)_N \end{bmatrix}$$

contains, in turn, the input values of each time step as before. In terms of time series, each value $(x(t-n)_i, .., x(t)_i)$ would be a value for in the i:th time step, e.g., the daily sales of a perishable product.

Some of the drawbacks of RNNs are that they are computationally expensive and sensitive to the exploding and vanishing gradient problems for longer sequences [27]. This can, to some degree, be combated with different regularization techniques. However, it remains a challenge, which is why RNNs are viewed as a group of models that are hard to train [26].

2.4.4 Transformer

A further evolution of neural networks for temporal learning is the Transformer architecture, first introduced in 2017 [27]. Since its introduction, it has been adapted to other fields and models, including the Temporal Fusion Transformer, introduced in Section 2.5. The proposed Transformer architecture achieves temporal learning by using a (multi-head) self-attention mechanism instead of sequential information-passing, which RNNs use, addressing the issues mentioned with RNNs.

The self-attention process is performed in multi-head attention layers in the Transformer. These layers can be placed in different stages of the model to capture dependencies between all pairs of inputs, regardless of their position in the sequence. The model's lack of perception of sequence order in the input is solved by first passing the inputs through a *Positional Encoding*[27] component, which encodes the order of the data to maintain knowledge of the original sequence through the attention layers. The attention layer works by for each head h_i in the attention layer first transforming the position-encoded input matrix $\overline{\mathbf{X}}$ into a query and key-value pairs of dimensions d_k and d_v represented by the three matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ respectively. This transformation is achieved through a linear transformation with the help of the different weight matrices learned during training, uniquely defined for each matrix as

$$\mathbf{Q} = \overline{\mathbf{X}}\mathbf{W}^Q \quad \mathbf{K} = \overline{\mathbf{X}}\mathbf{W}^K \quad \mathbf{V} = \overline{\mathbf{X}}\mathbf{W}^V. \tag{2.4}$$

The final weights on the values are obtained by applying the *Scaled Dot-Product Attention* function defined as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\mathbf{V}, \tag{2.5}$$

$$\text{where} \quad \text{softmax}(z_i) = \frac{\exp z_i}{\sum_{j=1}^{K} \exp z_j}.$$

To get the most out of self-attention, the authors of the Transformer paper use multi-head attention. For multi-head attention, the process for a single head is repeated for the desired number of heads with different linear projections. The resulting weight vector of each head is then concatenated, creating a matrix. Then a final dot product operation is performed with the weight matrix $\mathbf{W}^O$ to obtain the correct dimensions for the Transformer. The multi-head attention function is defined as

$$\begin{aligned} \text{head}_i &= \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ \text{MultiHead}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)\mathbf{W}^O \end{aligned} \tag{2.6}$$

where

- h is the number of heads
- d_{model} is the embedding dimensions of the model
- $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $\mathbf{W}_i^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

2.5 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) model is a state-of-the-art ML model for temporal learning introduced in 2021 [28]. The TFT model introduces new improvements to Transformer models to create an interpretable temporal learning model that achieves significant improvements in multi-horizon time series forecasting tasks. TFT also supports learning from inputs of known future covariates and static covariates of time series to use information from more data sources when making predictions.

The TFT model adapts the attention mechanism (described in 2.4.4) from Transformer models and combines it with *Variable Selection Networks*, *Gated Residual Networks*, and distributional quantile outputs making it a more architecturally complex model, (see Figure 2.7). Moreover, thanks to the attention and variable selection mechanisms offering tools for interpretability, it is also more interpretable than even relatively simple RNNs.

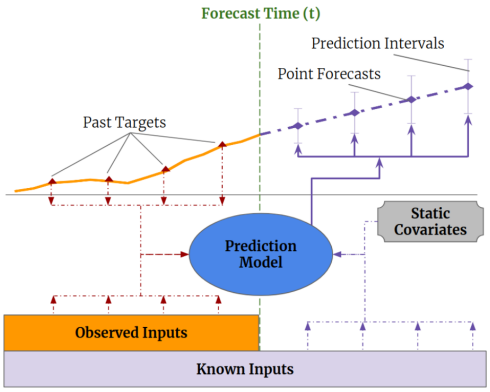


Figure 2.8: Figure visualizes the original (non-modified) representation of the TFT model's input types as presented in the original paper by Lim et al. [28]. (Used under Creative Common license CC BY 4.0 [29].)

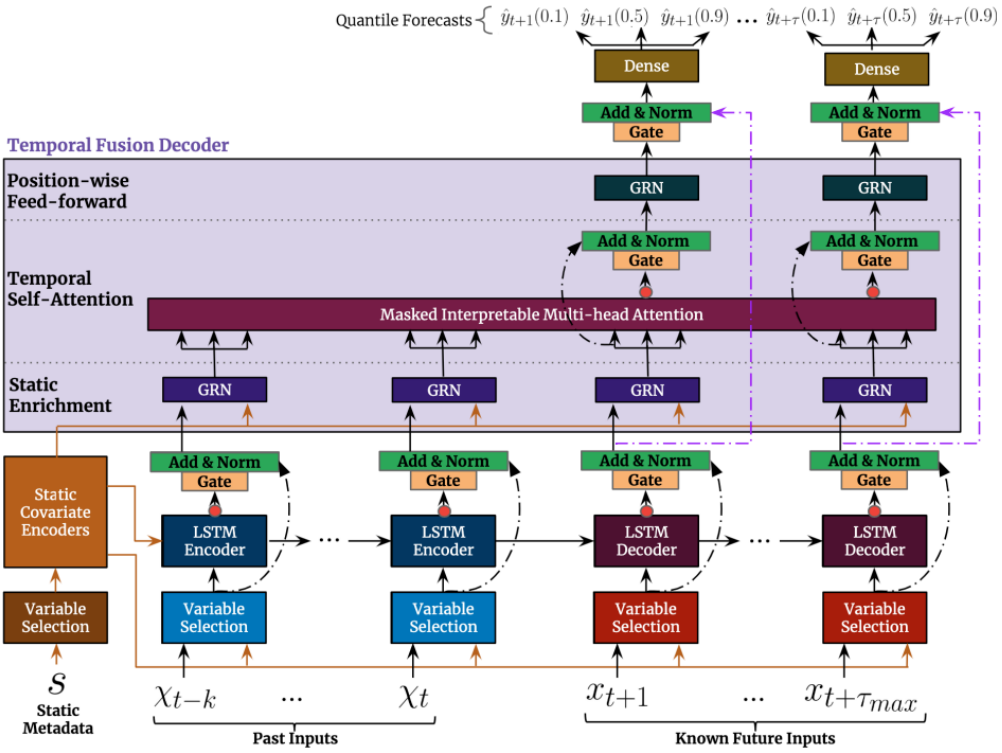


Figure 2.7: Figure visualizes the original (non-modified) architecture of the TFT model as presented in the original paper by Lim et al. [28]. (Used under Creative Common license CC BY 4.0 [29].)

2.5.1 Variable Input Types

TFT supports learning from three data sources: observed inputs, known inputs, and static covariates, as shown in Figure 2.8. Observed inputs cover the class of historical time series observed up to the prediction time, including both the target time series and other exogenous time series acting as covariates. For instance, the target might be the daily sales of a perishable marked-down product, and the covariate time series might be the daily revenue or customers visiting a store. Input time series which stretch beyond the prediction time is treated as known inputs. These could, for example, be time-related features, e.g., days of the week, as it is known that after Tuesday comes Wednesday and so on at prediction time. Lastly, static covariates are time-independent features of time series. For example, suppose that the target time series is the daily sales of a perishable marked-down product; then static covariates of that time series might be the type of product it is or its fixed weight.

2.5.2 Gated Residual Network

The Gated Residual Network (GRN) is a neural network architecture that, through a gating mechanism using *Gated Linear Units* (GLUs), adds more flexibility to the model by applying non-linear processing of inputs only where needed [30]. Applying such a gating mechanism allows the TFT model to selectively update the network's hidden state at each time step [28]. The TFT models implementation differs from

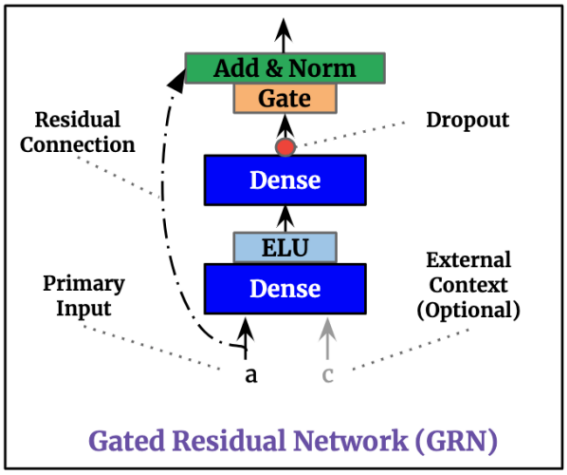


Figure 2.9: Figure visualizes the original (non-modified) representation of the TFT model's Gated Residual Network as presented in the original paper by Lim et al. [28]. (Used under Creative Common license CC BY 4.0 [29].)

the one originally proposed GRN as it takes both a primary input vector $\boldsymbol{a}$ and an optional context vector $\boldsymbol{c}$ which originate from the encoded static covariates (see Figure 2.7).

Formally, the TFT's implementation works as shown in Figure 2.9 where the *Dense* blocks are regular fully connected feed-forward networks, *ELU* is the Exponential Linear Unit activation function, Gate is the GLU, and lastly the *Add & Norm* block is a layer normalization [31] that facilitates the training process of the network. In an oversimplified way, the TFT's GRN works by regulating an internal parameter $k \in [0, 1]$. When $k = 1$, the GRN will act as a linear function suppressing information flow through the Dense blocks only using the residual connection. When $0 < k < 1$, the GRN processes both the linear residual connection and the non-linear processing of inputs through the Dense blocks. Regulating the parameter k enables the GRN to use non-linear processing of the inputs only when needed, otherwise suppressing the complex transformations and defaulting to only use linear transformations.

2.5.3 Variable Selection Network

The purpose of Variable Selection Networks (VSNs) is to assign weights to the input features in proportionality to how important they are in predicting the targets. For the static covariates, this is done collectively, and for the time-dependent features, this is done at every time step t_i in the model. Having a separate VSN at each time step allows the TFT model to assign weights to each feature relative to their sequence in time. The VSNs might assign some features more importance closer to the prediction time while assigning other seasonal features more importance in seasonal intervals. Furthermore, VSNs also passively remove noisy input features that could negatively impact the model due to lacking correlation with the target [28].

2.5.4 Quantile Predictions

TFT uses a quantile loss function and quantile regression in its forecasting, enabling the model to output not only a scalar value prediction but a distribution as the prediction [28]. The model does this by sampling multiple predictions from which the results can be interpreted as a distribution, representing how certain the model is in its prediction. For example, using the 10th and 90th quantiles of the prediction distribution, the result can be interpreted as the prediction is in this interval with a 90% certainty. These quantile outputs of the TFT are generated by linear transformation on the model's outputs with a weight matrix $\mathbf{W}_q$ in the following way:

$$\hat{\boldsymbol{y}}(q, t, \tau) = \mathbf{W}_q \tilde{\psi}(t, \tau) + b_q \tag{2.7}$$

where q is the specified quantile, τ is the future time step to be forecasted, t, is some time step in the model, $\tilde{\psi}(t, \tau)$, is the output of the last component in the model, and $\mathbf{W}_q \in \mathbb{R}$, b_q is the coefficient of quantile q [28].

2.6 Clustering

Clustering is the process of finding groupings, known as *clusters*, in a dataset. This is with the formal goal of maximizing the intercluster variance while minimizing the intracluster variance and, in simpler terms, finding distinct clusters that are homogeneous. When using temporal data as input, features could be extracted from the time series such that typical clustering models can be used on those features.

To achieve this, some measure of similarity (or, inversely, distance) measure is needed. It is often defined as $\mathcal{D}(p, q)$ where $\mathcal{D}$ is a function of two observations p and q in a n-dimensional space that returns the distance between these two observations. A typical similarity measure is the Euclidean distance

$$d_{euclid}(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

and another could also be the Manhattan distance.

The are multiple reasons for using clustering in time series [32]. Firstly, to add an additional features to use during prediction, indicating that certain entities behave similarly. Secondly, to divide the products into separate clusters where each cluster is used as training data to k (where k is the number of clusters) different models that are used to predict the behavior of entities in a specific cluster. Lastly, to counteract sparse data by finding similarly behaved entities in the data and using the assigned clusters *as if* entities in a cluster are the same, providing more data per "entity" but somewhat fewer total entities.

2.6.1 *K*-means Algorithm

The K-means algorithm is a partitioning clustering algorithm with the goal of dividing n data points consisting of m features into K clusters such that the within-cluster

sum of squares (Euclidean distance) is as small as possible [33]. The algorithm is one of the most popular there is in the sphere of cluster analysis.

The algorithm works as follows. Firstly, initialize K centroids; these do not have to be any of the n data points. Then, each of the n samples is compared to the K centroids and assigned to the closest centroid in terms of Euclidean distance. This is known as the *expectation* step. After this step, each centroid is recalculated as the mean of samples assigned to the cluster; this is at times known as the *maximization* step. The assignment and update step is performed repeatedly until the algorithm terminates when none of the K centroids changes in the update.

A key factor of the K-means is that it produces locally optimal solutions rather than globally optimal ones in which the algorithm's speed highly depends on how close the random initialization of centroids is to a local optimum.

2.6.2 Silhouette Score

To evaluate the quality of some clustering, such as which value of K to use in K-means clustering, the Silhouette score s_i is an appropriate measure [34]. The Silhouette score is defined for each observation i and is defined as [34], [35]

$$s_i = \frac{(b_i - a_i)}{\max(a_i, b_i)}$$

where a_i is the mean intra-cluster distance from observation i to each other point in the same cluster, and b_i is the mean nearest-cluster distance for observation i. They are defined as

$$a_i = \frac{1}{n_{c_l} - 1} \sum_{j \in c_l, i \neq j} d(i, j), \qquad \forall i \in c_l$$

$$b_i = \min_{m \neq l} \frac{1}{n_{c_m}} \sum_{j \in c_m} d(i, j), \qquad \forall i \in c_l.$$

The Silhouette score is bounded between -1 and $+1$. A Silhouette score of near $+1$ means that the sample is in the correct cluster, a score of near 0 signifies that it might be a better fit in another cluster, and a near -1 score means that the observation is in a wrong cluster. The individual observations Silhouette scores can then be used in two ways: by calculating the mean Silhouette score over all samples or by using tools to evaluate the score of each entity.

2.7 Random Convolutional Kernel Transformation

Rand**o**m **C**onvolutional **Ke**rnel **T**ransformation (ROCKET) is a method introduced in 2019 [36] for transforming time-dependent data (time series) into time-independent data. ROCKET was developed to make accurate time series classification more

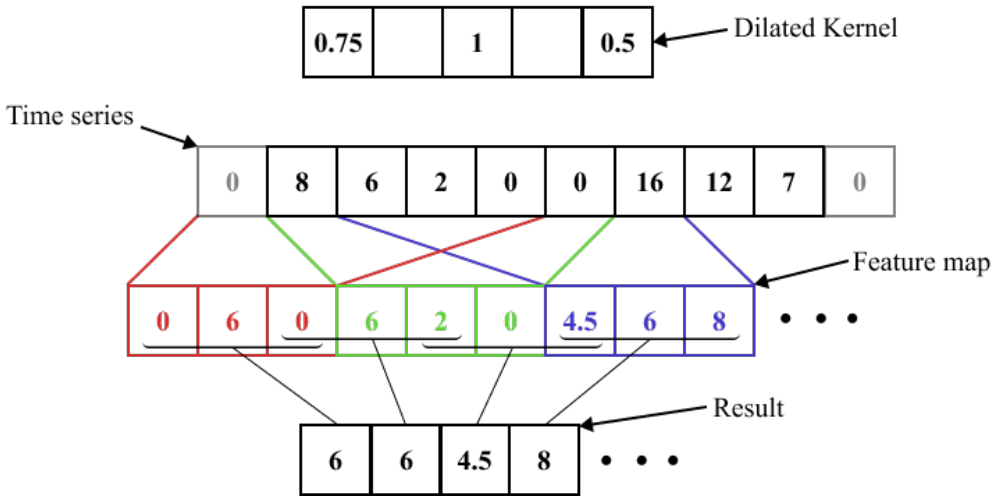


Figure 2.10: Example of the dilated convolutional transformation, with kernel length 3 with weights $[0.75, 1, 0.5]$, stride 1, dilation 1 (skipping every other value), padding 1, max pooling with pooling size 3 and pooling stride 2, (and bias 0). The dilated kernels traverse the time series from left to right with the given stride, resulting in a feature map that is then pooled to produce the final results.

accessible, as alternative methods have high computational complexity making them intractable for larger datasets. The ROCKET paper's authors show that transforming time series datasets with random convolutional kernels and then applying simple linear classifiers to the transformed data can achieve state-of-the-art accuracy in time series classification with a fraction of the computational expense of alternative methods [36]. ROCKET transformations also allow multiple time series to be transformed as one entity, enabling a collection of time series to be considered as one data point after the transformation.

2.7.1 Dilated Convolutional Transformations

Dilated convolutional transformation is a technique used to extract patterns and feature representations from larger data structures such as images or, in the case of ROCKET, time series. Figure 2.10 demonstrates this for time series. The dilated convolutional transformation is performed with the help of a dilated kernel, which is a one-dimensional vector in the case of time series. A regular kernel contains values (weights) between 0 and 1. The transformation uses the dilated kernel (skipping over values) to perform a series of dot-product operations on the time series vector, moving one stride at a time and repeating the process until it reaches the end. This produces a new vector called a feature map. Also, sometimes a padding of zeros is added to the ends of the time series to enable the kernel to cover all values in the time series. Next, a pooling transformation is performed on the feature map. The resulting vector is the result of the dilated convolutional transformation.

ROCKET uses dilated convolutional kernels for its transformations, similar to those used in Convolutional Neural Networks (CNNs) for image classification. How-

ever, instead of learning the kernels, ROCKET uses kernels with randomly sampled lengths, weights, biases, dilations, and padding. This random sampling uses predefined designated distributions for each parameter, considering the input length of the series when doing so. Furthermore, ROCKET differentiates itself in a couple of other ways from the more traditional use of dilated convolutional transformations. Firstly, it uses a vast number of kernels (by default, 10'000) and a large variety of kernels. Secondly, it also uses random values to dilute each kernel, helping it to capture patterns of different frequencies and scales. Thirdly, it applies a bias (random scalar value) at the end of each transformation to differentiate two otherwise identical kernels. Lastly, ROCKET uses a combination of global max pooling and proportion of positive values to transform the feature maps [36].

2.8 Forecasting Evaluation Metrics

Four different evaluation metrics will be used to evaluate the quality of the models. These are *Cumulated Forecast Error* (CFE)[1], *Mean Absolute Deviation* (MAD)[2] and *Percentage Better* (PB), which is essentially only a comparative value to see how much better (compared to some baseline) that some model is than another in some specific metric. These metrics are recommended by intermittent demand forecasting literature[37], [38] and aim to capture different aspects of forecasting intermittent demand. The different metrics are defined as follows (* denotes a baseline score):

$$CFE_t = \sum_{i=1}^{t} (\hat{y}_i - y_i)$$

$$MAD = \frac{\sum_{i=1}^{t} |(y_i - \hat{y}_i)|}{t}$$

$$PBMAD = \frac{|MAD - MAD^*|}{MAD^*}$$

$$PBCFE = \frac{|CFE - CFE^*|}{CFE^*}.$$

An example of how the MAD and CFE metrics function can be exemplified via Table 2.1. The MAD and CFE metrics are calculated as above and would result in the values:

$$MAD_{example} = \frac{(2+1+0+1+1)}{5} = 1$$
$$CFE_{example} = 2 + (-1) + 0 + (-1) + 0 = 0.$$

In words, MAD is, as the name suggests, the mean absolute deviation at each timestep, while the CFE is the sum of all (signed) deviations over all timesteps. The

[1] Also known as the Forecast Bias.
[2] Also known as the Mean Absolute Error, MAE.

Table 2.1: An example time series, where each column represents a time step.

t	1	2	3	4	5
Predicted	2	2	1	0	0
True	0	3	1	1	0
Diff	2	-1	0	-1	0

reason for using CFE is because it allows calculating the cumulative error over a whole period, producing signed errors where positive and negative errors cancel out. This eventually represents the models' bias over the series.

The MAD essentially encodes the same information as that of the *Mean Squared Error* (MSE), and in some ways, also presents somewhat similar information as the CFE. However, there are a few notable differences that justify its inclusion. Firstly, it shows an unsigned, non-squared error which allows for great interpretability of the performance of a model. The metric, like CFE, is not necessarily suitable for evaluating a single model between series. However, when the different series are of equal or similar scale, the MAD metric is also able to perform such comparisons. Like the CFE metric, it is also suitable for comparing models against each other on the same series. Lastly, of the included metrics, the PB (PBMAD and PBCFE) is essentially only an easy-to-digest comparison tool between some models and a baseline model.

The most notable exclusion from the selected list of metrics is the MASE (*Mean Absolute Scaled Error*) metric [38]. The metric is often viewed as one of the most suitable in intermittent time series forecasting. The exclusion is based on two factors. Firstly, as mentioned by the same author that recommended MASE, when the different series are on essentially the same scale, using a *scale-based* metric such as MASE is not necessary to compare a models performance on different series [39]. Secondly, the goal is to evaluate the models' performance on *active periods*, i.e., when there is marked-downed inventory. These periods inherently follow a period of no inventory, and thus, the naive forecasting of MASE used to scale the data provide no additional value compared to the MAD metric.

Other commonly used time series evaluation metrics excluded are the *percentage-based* metrics such as MAPE and sMAPE. This is because these metrics are prone to produce undesirable evaluation scores [38]. This is a consequence of their form, which divides the predicted value with the true value, which is more often than not zero, which is undefined (division by zero). For a similar reason, *relative-based* metrics, such as *Median Relative Absolute Error*, are also unsuitable for the intermittent domain, as errors are often small in intermittent data, leading to division by zero (or close to) and thus undesirably large evaluation scores.

3

Data Exploration

To evaluate and explore models on intermittent data, a real-world data set consisting of ordinary product sales and markdown product sales from two physical stores is used. This chapter introduces the data set used and presents important properties of the data set. These properties are essential in the choice of models and methods used. The findings in this chapter directly correlate with the models' results and limitations.

3.1 Data

This section describes the dataset used in more detail. The features present in the original dataset are presented together with the additional features that were either engineered from the existing features or added from external sources to complement the dataset. Details about the methods that were used to split the data into suitable training, validation, and test sets are also explained in this section. Lastly, methods for analyzing the essential characteristics of the data, or other necessary properties, are presented here.

3.1.1 Dataset

The dataset that will be used to evaluate the models is a real-world dataset consisting of two physical stores, each containing thousands of distinct products. Each product has, in turn, both static and temporal features. The temporal features are available as time series containing values aggregated on a daily frequency and stretches over an 8-month period from the start of August 2022 to mid-March 2023, containing 233 samples each.

Product Features

There are fifteen product features, of which four are static features that do not change over time and eleven temporal features with moving values. The product's static features include the product's price and categorical features as product groups that

Table 3.1: Table presents the most relevant product features with a short description of each feature. Appendix A Table A.1 contains an exhaustive list of product features.

Product features		
Feature name	Feature type	Description
Markdown units sold	Temporal	Daily number of marked down product items sold
Regular units sold	Temporal	Daily number of non-markdown product items sold
Markdown stock	Temporal	Number of product items on markdown if there is an active markdown
Products' department	Static	A distinguishing category of the product
Products' assortment	Static	A distinguishing subcategory of the product

Table 3.2: Table presents a slice of some temporal features' time series during an active markdown period.

Date	Days to expiration	Markdown discount	Markdown inventory	Markdown units sold	Regular units sold	...
⋮						
2022-09-10	0	0.00	0	0	4	...
2022-09-11	2	0.34	12	3	13	...
2022-09-12	1	0.34	9	6	12	...
2022-09-13	0	0.00	0	0	8	...
⋮						

describe what type of product it is. The product's temporal features consist of several sales-related features aggregated daily. These include the number of items sold per day, the number of marked-down items sold per day, the number of items currently on markdown, the number of days to the expiration date of the marked-down items, and more. Table 3.1 contains the most relevant product features with a short description of each feature. In Appendix A Table A.1 the exhaustive list of the product features used is presented. Table 3.2 also shows an example of the temporal features over a few days, illustrating a markdown period between 2022-09-11 to 2022-09-12. Hereafter we will refer to these periods where the *days to expiration* feature is positive in time as active periods in the products markdown data.

Environmental Features

In addition to the products' features, the stores also offer valuable data giving more insight into the environment in which the products are sold. Each store has daily

Table 3.3: Table presents the most relevant environmental features with a short description of each feature. Appendix A Table A.2 contains an exhaustive list of environmental features.

Environmental features		
Feature name	Feature type	Description
Value sold	Temporal	Daily value (in currency) of the sales
Units sold on regular sale	Temporal	Daily number of units sold in the store
Units sold on markdown	Temporal	Daily number of markdown units sold
Units sold on promotion	Temporal	Daily number of promotion units sold

aggregated sale numbers of different types that indicate the general demand over time. We will refer to the features originating from the stores and other features that can be connected to the stores as environmental features since they describe the environment of the variables of interest. In total, there are fifteen such environmental features. Table 3.3 presents a list of the most relevant environmental features with a short description of each feature. Appendix A Table A.2 presents the exhaustive list of the environmental features used.

Time-related and Other Exogenous Features

Time-related features are highly beneficial for time series forecasting as they can help forecasting models identify correlations between fluctuations in demand and time. For example, how the demand is affected by days of the week, holidays, pay days, etc. Incorporating time-related features can help the models find patterns relating to seasonalities and trends, improving the accuracy of the models.

The time-related time series added to the data set are year, month, day, and day of the week. Furthermore, features such as pay-day and holidays are adjusted to the local practices where the products are sold. How these features and all other features are encoded and transformed to work with the models is described in Section 4.1.

3.1.2 Target and Covariates

In this thesis, we differentiate between four data types: target, past covariates, future covariates, and static covariates. All but the static covariates are data in the form of time series. Figure 3.1 visualizes how the different time series data types play into the forecasting of the target.

The target feature in this thesis is the *markdown units sold*, which is the time series of the daily number of units sold on markdown for a particular product. The forecasting will therefore yield the predicted continuation of the target series.

The past and future covariates data types are covariate time series used as en-

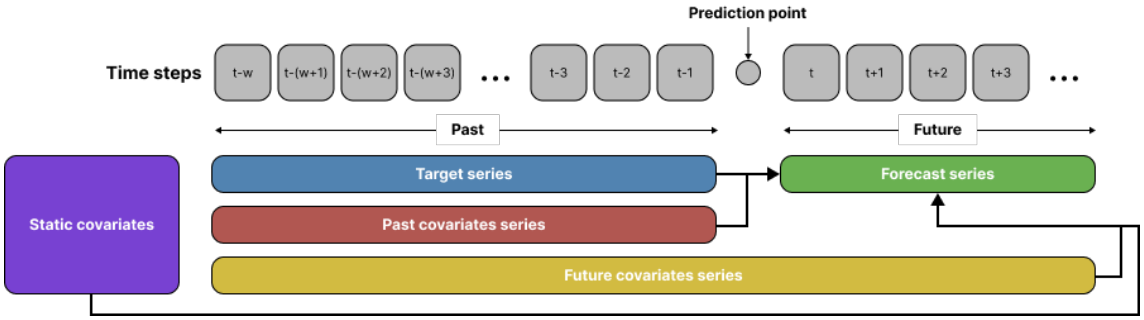


Figure 3.1: Figure shows how different types of time series are used for forecasting relative to the prediction point in time.

riching information for forecasting. Temporal covariate information with either a positive or negative correlation with the target series can contribute further information to the forecasting, improving the accuracy of predictions. Past covariate time series are time series with information that can be obtained up to the time of the prediction only and no further. Using information from these past covariate time series beyond the prediction time would mean that the forecasting would use privileged information about the future. Time series containing historical sales data are examples of past covariate time series, as those are only available up to the prediction time.

On the other hand, future covariate series are time series whose future values are known and can be obtained at prediction time. Thus, future covariate time series are deterministic. For example, information such as the day of the week, month, etc., for future time steps can all be obtained at prediction time.

3.1.3 Data Selection

To evaluate the models, we use a subset of products in the dataset containing the top 200 products with the most sales on markdown. This results in the dataset containing 127 products from store 1 and 73 products from store 2. We plot three distributions to gain insight into this selected dataset's properties. Figure 3.2 shows the average number of daily units marked down (during active periods). Here it can be seen that in most instances, the number of daily markdown sales for products is limited to only a couple. However, some outliers create a high variance, potentially making these hard to predict accurately. Figure 3.3 shows the distribution of sell-through rates for products, i.e., the proportion of marked-down units that sell, ranging from 0 to 1 with a mean of 0.36. Lastly, Figure 3.4 shows the proportion of time in the dataset's interval that a product is on markdown, ranging from 0.4% to 39% with a mean of 17%. This tells us that some products have much more historical markdown periods to learn from than others, potentially making those with fewer harder to predict. In summary, the dataset is relatively noisy, reflecting that many confounding variables affect the sale of markdown items.

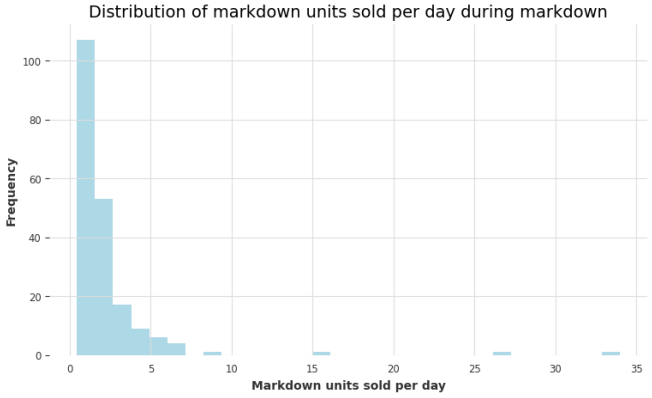


Figure 3.2: Histogram representing the distribution of markdown units sold daily during markdown periods. The x-axis represents the number of markdown units sold per day.

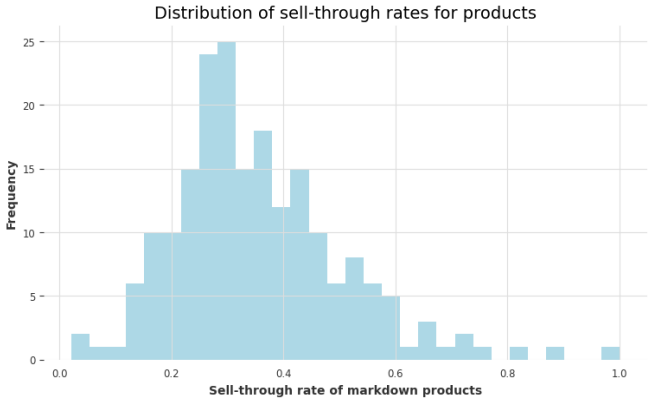


Figure 3.3: Histogram representing the distribution of products' sell-through rates. The x-axis represents the sell-through rate of markdown products, i.e., the proportion of the markdown products that are sold and not wasted.

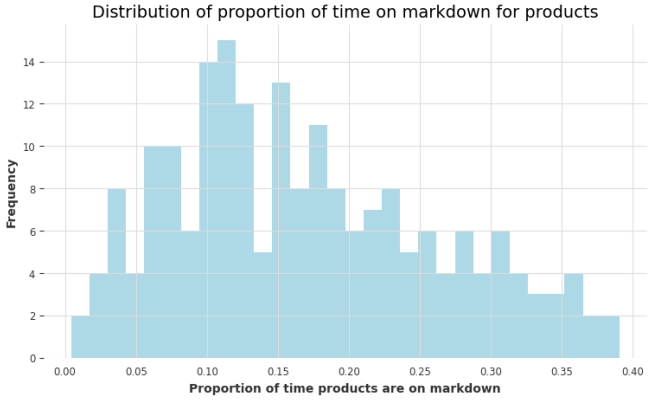


Figure 3.4: Histogram representing the distribution of the proportion of time different products are on markdown. The x-axis represents the proportion of time that products are on markdown in the data set (0-1).

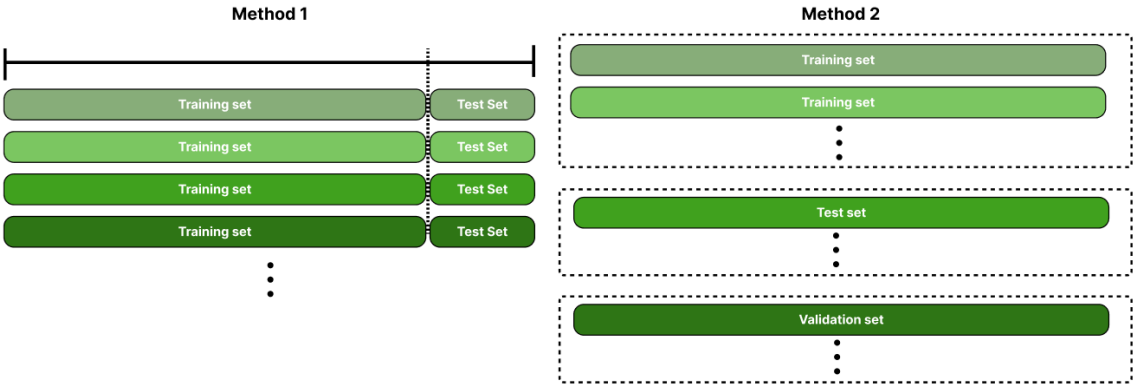


Figure 3.5: The two data splitting methods used to train and evaluate the models.

3.1.4 Data Splitting

We split the data using two methods. In the first method, we split each product's time series by a chosen timestamp so that each time series is split into two, creating two sets of time series, see the left side of Figure 3.5. The first set of time series is reserved for training, and the second set of time series is used to evaluate the models. The timestamp to split the time series by is chosen so that the training time series contains the first 70% of the time steps in the time series, and the test time series contains the remaining 30%, which is the tail of the time series. We use this method to enable evaluation between all models, as the models work slightly differently. For the ML models which use validation data during training, the test set is also used as validation data due to the limiting length of the time series.

In the second method for splitting the data, we split the dataset by products, reserving 60% of the data for training and 20% each for validation and testing. See the right side of Figure 3.5. Since the Exponential Smoothing and Croston models cannot be trained and used to perform predictions on completely unseen time series, this second data splitting method is only used for the ML models RNN and Temporal Fusion Transformer. The purpose of splitting the data by products instead of a timestamp is to evaluate how the models perform on unseen data that was not present during training.

3.2 Data Exploration

We analyze the data from different aspects to verify that time series forecasting is possible and advantageous for the identified problem in the chosen domain. We analyze if the time series in the data set exhibit auto-correlation and stationarity and if there are any seasonalities in the data. Verifying these qualities of the data set is the first step to showing that using time series forecasting is feasible for the problem and helps understand the statistical properties of the data.

3.2.1 Stationarity

Proving stationarity is essential in enabling models, especially statistical ones such as Exponential Smoothing and Croston's method, to function to their maximum potential. Often, intermittent data is stationary. However, some studies show that even intermittent data may contain patterns such as trends, resulting in a non-stationary series[38].

Similarly to the above, the top 200 marked-down products' target time series are evaluated individually to understand whether the individual series are already stationary or need to be processed accordingly. This is done via the Augmented Dickey-Fuller Test (ADF-Test) as described in Section 2.2. The specific tests used come from the `statsmodels` library [40]. When testing for stationarity, a confidence of 95% is used to reject the ADF-Test null hypothesis.

Table 3.4: Number of the products' markdown sales time series that are stationary.

Stationary	Non-stationary
193	7

The results of the ADF-Test indicate that most marked-down product series are stationary, allowing for more simple models to function to their maximal potential.

3.2.2 Auto-correlation

The idea behind using temporal data for forecasting the number of sales during an active markdown period is that lagged versions (history) of the target feature may aid in forecasting future values of that same series. The metric used to measure whether a series does correlate with a lagged version of itself is the auto-correlation (as described in Section 2.2.1). If there is no auto-correlation at any lag, then there is no information in the previous values of the target series that allows for better predicting of the future values.

To evaluate whether there is auto-correlation, the top 200 marked-down products' target time series are evaluated individually to determine the suitability of using time series analysis and forecasting instead of other methods. Table 3.5 shows the count of individual observations exhibiting auto-correlation at some lag up to 21 periods (3 weeks) with a 95% confidence.

Table 3.5: Number of the products' markdown sales time series that exhibits auto-correlation.

Autocorrelated	Non-autocorrelated
173	27

As can be seen, 86.5% of the top marked-downed product shows auto-correlation, indicating that using time series analysis is a viable and reasonable assumption even though the series is of intermittent demand and not smooth.

In Figures 3.6 and 3.7, two example products' auto-correlation values (calculated as defined in Section 2.2.1) are shown for each lag up to 21 where the largest auto-correlation value is highlighted. The two series indicate the two most common ways auto-correlation is realized in the data. Either, as seen in Figure 3.6 where a previous couple of values are a good indicator of current values, or, as seen in Figure 3.7, where that values farther back are better indicators of current sales (likely suggesting some sort of seasonality). In Figures 3.6 and 3.7, the blue area represents the autocorrelation for each time step where there is no statistically significant autocorrelation. Thus, spikes outside the blue area indicate statistically significant autocorrelation.

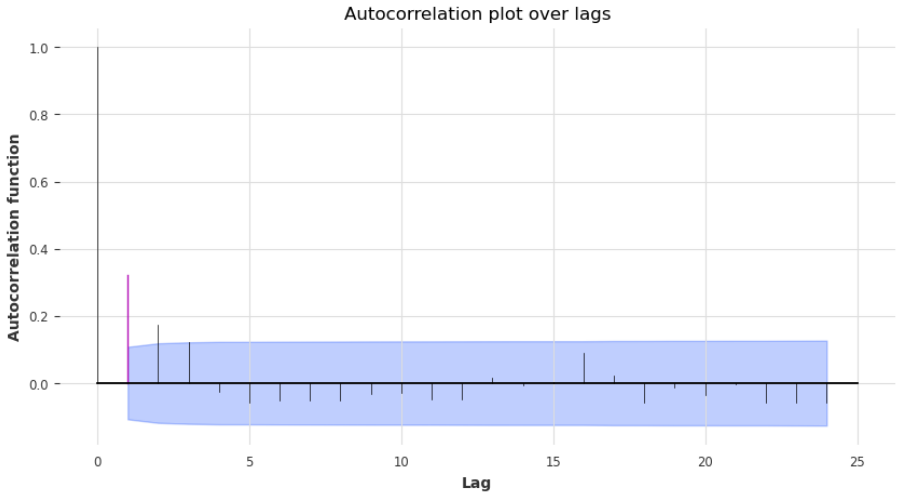


Figure 3.6: Auto-correlation plot of a product's markdown sales with high auto-correlation in the early lag values.

3.2.3 Trends & Seasonality

In evaluating whether the data contains trends and seasonality, the belief going in is that no trend will exist in the vast majority of the data. That belief also holds up, as none of the series observe any statistically significant trend. This is also very common for intermittent data. Additionally, it is vital to explore the data for any seasonalities. Table 3.6 shows the count of the top 200 series that exhibit and do not exhibit statistically significant seasonality (95% confidence).

Table 3.6: Number of the products' markdown sales time series that exhibits seasonalities.

Seasonal	Non-seasonal
188	12

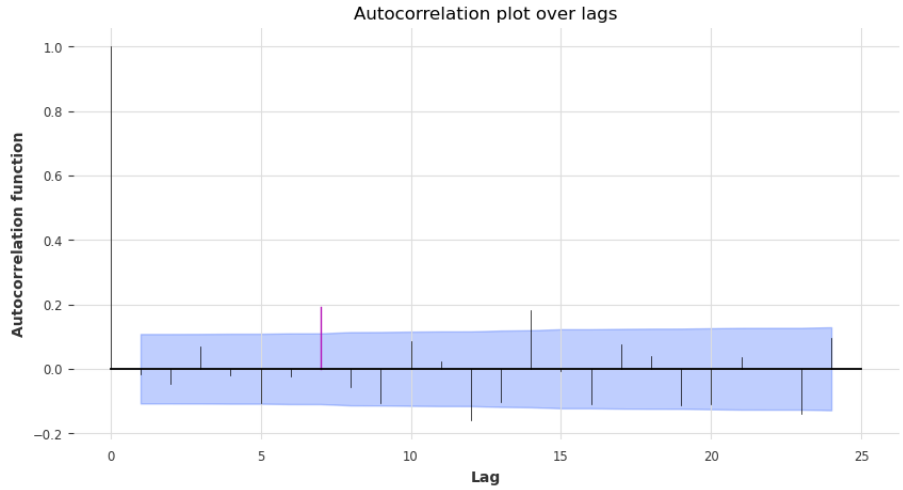


Figure 3.7: Auto-correlation plot of a product's markdown sales with higher auto-correlations in later lag values.

Figure 3.8 presents the seasonality for the top 200 marked-down products. It seems like the mode seasonality lag is 4, with most values being around 2 until 7 in the seasonality period.

Therefore, the critical question is whether this seasonality appears because items are repeatedly marked down on a schedule observed in the seasonality or whether the seasonality simply depends on the length of the markdown periods. In Figure 3.9, the count of markdown period lengths observed in the top 200 products is plotted. As can be seen, this does not look especially similar to the seasonality periods, and thus, the assumption is made that there is some sort of seasonality in the data that is worth capturing. It is, therefore, essential to evaluate models that might either statistically model seasonality as part of their design or use a model complex enough to capture the seasonality indirectly. Moreover, to allow the selected models to perform their best, selecting a look-back window at training and prediction of at least 21 would be optimal.

3.2.4 Feature Correlations

To evaluate what other features might be suitable, in a temporal fashion, used as input data, the Granger causality test is used. Granger causality is similar to the auto-correlation tests, which evaluate how series with different lags correlate. The difference with auto-correlation tests is that they evaluate one series against a lagged version of another series rather than against a lagged version of itself. Granger causality, therefore, indicates if one series contains useful information when predicting another series.

Comparing the target series and markdown sales with other available series, such as inventory level (stock), total sales, and discount, is done using the top 200 marked-

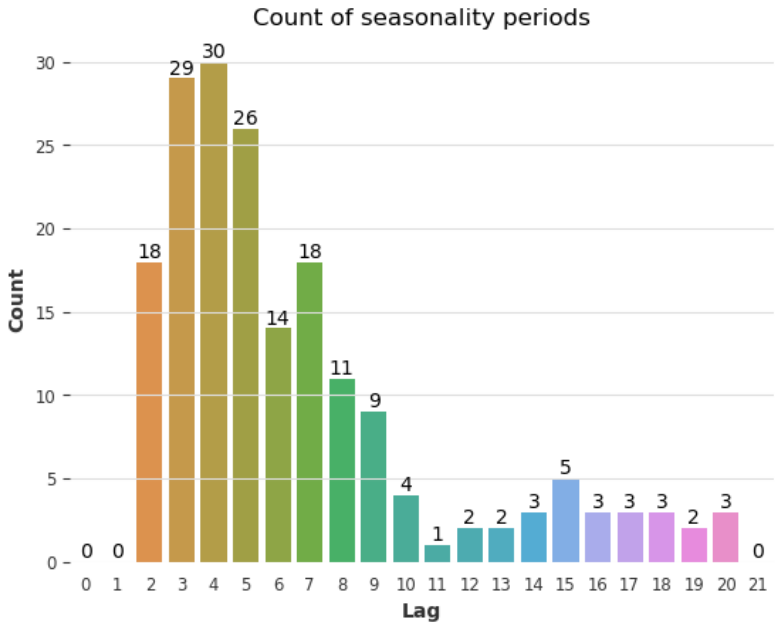


Figure 3.8: A bar graph representing the count of each observed seasonality lag. I.e., presenting how many products exhibit (cyclic) seasonal behavior after a period of a specific number of days.

down products. Each product series is then individually evaluated against each other available series for that product and store. The existence of Granger Causality of each cause (feature) series against the target series is evaluated at a 95% confidence level, and the result is presented in Table 3.7. The data is evaluated at a maximum lag of 21.

The results in Table 3.7 highlight that no feature explains every time series. However, the combination of these features does provide an explanation for each of the top 200 marked-down products. That is, there is no product that does not have a single feature with a statistically significant Granger causality. The exploration of feature correlation indicates using the features in Table 3.7 as a base for using as input when creating a model.

When evaluating the four features where the highest number of series where Granger causality exists, that is **Stock**, **Expired Stock Discount**, and **Is Working Day** a key observation is made. That is, in the vast majority of cases, the Granger causality is present at multiple lags, which further indicates the potential value of using time series analysis for the problem. Looking extra at the **Is Working Day** feature, it suggests that knowing whether it is a weekend is a highly useful feature for a particular group of products.

A key observation is that looking at the top 200 marked-down products, all products display Granger causality with at least one (or more) other features. It is often observed that the same combinations of features have Granger causality with a

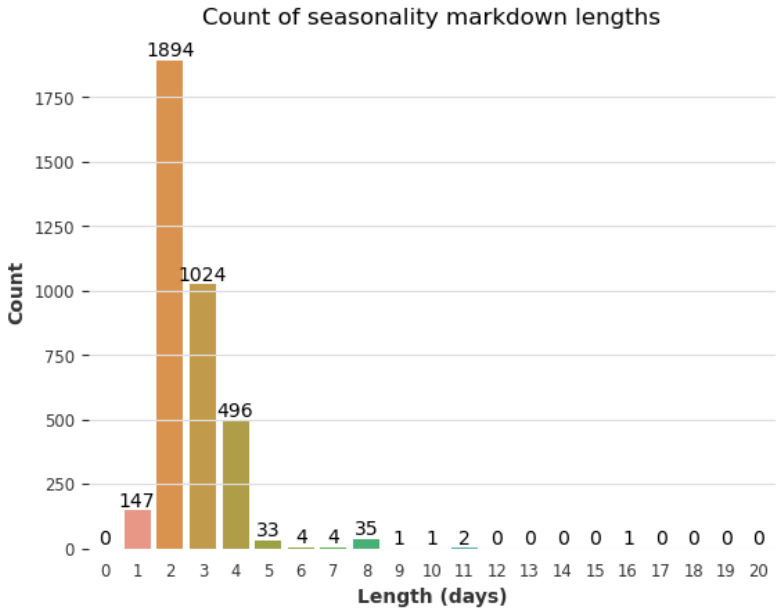


Figure 3.9: The count of observed markdown period lengths, i.e., how long a markdown is active.

group (not only single) products. Therefore, a model that can capture some static (non-temporal) descriptions of the data might be able to better predict future values. Furthermore, a model that can use these other features in a temporal format is also of interest as it might use both the auto-correlation from the target and the other features with Granger causality to better predict future values.

Given this information, we believe a model like the Temporal Fusion Transformer (see Section 2.5) is interesting. Firstly, it allows for multiple input time series other than the target, allowing usage of those features at prediction time. Secondly, it can divide these features into *past* and *future* inputs, allowing information that is known at the beginning of the day (such as stock) to be used. Thirdly, it allows for static inputs, such as descriptions of the products like assortments and even clustering, to further aid the model in understanding which features are more telling for certain products.

Table 3.7: Table shows how many products by each feature time series correlate with the target time series.

Correlation with target		
Feature	Exist	Non-existing
Stock	176	24
Expired stock	147	53
Discount	110	90
Is Working Day	104	96
Units sold product	83	117
New stock	81	119
Total units sold	78	122
Store value sold	72	128
Sun hours	72	128
Units on promotion	71	129
Payday	69	131
Total ordinary value	67	133
Total ordinary sales	67	133
Total promotion sales	63	137
Total promotion value	63	137
Total markdown sales	59	141
Total markdown value	59	141
Wasted units	48	152
Holiday	47	15
Wasted value	45	155
Days Until Expiration	42	158

4

Methods

This chapter presents the vital steps and components of the methods behind the achieved results. First, the preprocessing and the feature engineering steps applied to the data to fit the models subject to evaluation are described. Next, the chosen models are presented with details about their implementation and training procedures. Lastly, the different evaluation techniques used and how they differ for the models are presented.

4.1 Preprocessing

This section describes the techniques used to preprocess the data and the motivation behind the chosen methods for different types of features. The preprocessing step performs two types of transformations to the dataset. In the case of numerical features, the values are scaled to facilitate the ML models' training, improve their performance, and make them more robust. Scaling the data helps the ML models to converge faster, thus speeding up the training process. Furthermore, scaling the data ensures that the features are on the same scale, thus preventing the models from favoring features based on their scale and making the models less sensitive to outliers in the data. The other type of transformation during preprocessing is to transform static categorical features (which often are in the form of strings) to represent the categories as numerical features.

4.1.1 Numerical Features

Most of the used features are numerical and are represented as time series. However, each feature has a unique range and distribution of values. In some cases transforming the features' values to be as close to a normal (Gaussian) distribution as possible can benefit the model. In our case, transforming features that do not require maintaining proportionality increased the performance of the models. Various scaling methods were experimented with to observe how they transformed the original distribution of each feature. Figure 4.1 depicts how these different methods transformed the total

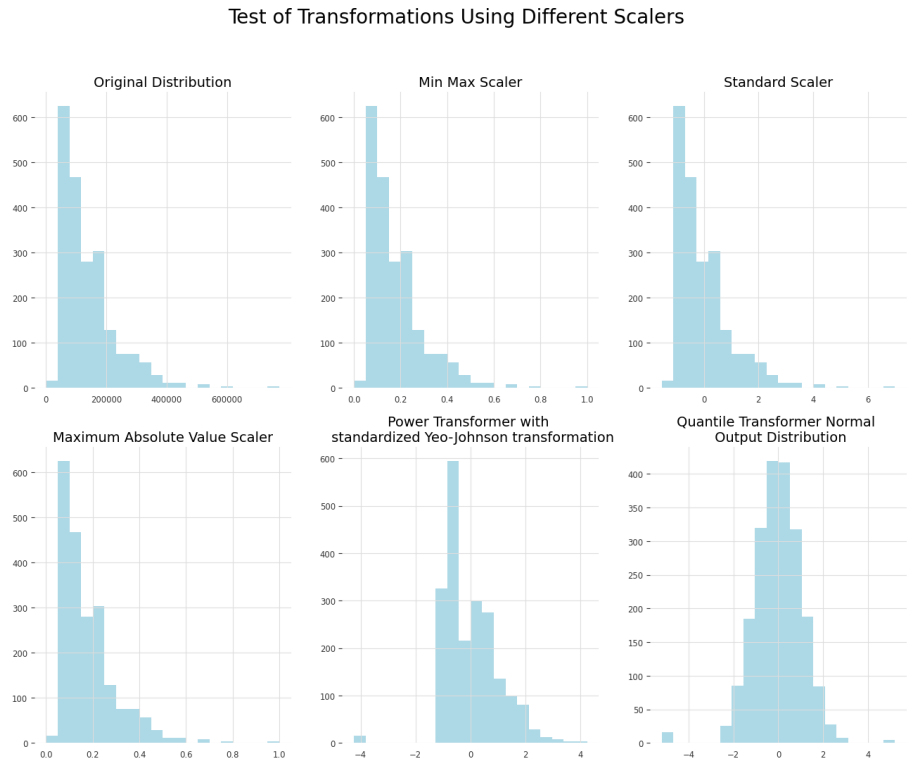


Figure 4.1: Figure visualizes how different scalers transform the values sold per day in stores feature.

values sold per day in stores. All the transformations performed using scalers can be found in the preprocessing package of the `scikit-learn` library [41].

Some features require a scaling method that scales the data proportionally. Furthermore, certain sets of features need to be collectively scaled using the same scalar transformation. One of those feature sets is the features that measure units in the same range, e.g., the *markdown stock*, *markdown units sold*, *markdown units expired*, and *added markdown units*. It is crucial to scale these values using a common scaler so that, for example, a scaled *markdown stock* unit is equivalent to a scaled *markdown units sold* unit. Scaling the features in a feature set independently would break the relationship that we want the models to learn.

4.1.2 Categorical Features

Two sets of features are chosen to be interpreted as categorical. The first type is the set of static covariate features each target time series has. These include the products' characterizing features, such as the department they belong to and their cluster label. The other set of categorical features is those that are time-dependent but which we choose to encode as categorical. These include features such as month and day of the week. The reason for encoding these as categorical is to make it easier for the model to find seasonal correlations between certain months and days of the week with the demand.

The only model that supports static covariate features is the Temporal Fusion Transformer, which expects the categorical static covariate features to be integer encoded. This is because the TFT uses an encoding layer for the static features. When integer-encoding a feature variable, each distinct value found in the variables (training set) is assigned an integer. Furthermore, encoded values not seen in the training set are assigned an integer representing the "unknown" category. Therefore, the range in which each feature's integer encoding is used is $[0, num\_distinct(feature)]$ where $num\_distinct(feature)$ is the number of distinct values of a particular feature and the last integer in the interval represents the "unknown" category.

A technique called *One Hot Encoding* (OHE)[42] is used to transform categorical data for encoding time-dependent features. In OHE, each value of a feature is transformed into a binary vector representation of that value. For example, "Monday" may be transformed into the vector $[1, 0, 0, 0, 0, 0, 0]$ where this vector uniquely identifies Mondays in the data. Thus, one hot encoding the weekdays would create seven new time series containing binary values replacing the previous feature for all weekdays.

4.1.3 Cyclic Features

In some cases, something other than integer encoding or OHE may be more suitable. One such case is working with features like the day of the month, where OHE would create too many new features for such a superficial feature. Another issue with integer encoding and OHE in such a scenario is that the first and last days of the month would be encoded as being far away even though they are not. Instead, a cyclic encoding can be used that is able to express proximity in cycles, thus encoding the last and first values in numerical ranges to be close to each other. This cyclic encoding is performed by transforming the values of a feature so that the feature's unique values are evenly spaced around the unit circle [43]. Therefore, the cyclical encoding of, e.g., days of a month is expressed as a combination of two features as

$$\text{sin encoding}(x) = \sin\left(2\pi\frac{x}{x_{max}}\right), \quad \text{and} \quad \text{cos encoding}(x) = \cos\left(2\pi\frac{x}{x_{max}}\right) \quad (4.1)$$

where x is the unique value, e.g., the numerical value of the day in the month, and x_{max} is the highest value x can take on. In the case of days of the month, $x_{max} = 31$. This cyclic encoding is applied for the days of the month, months of the year features, and also to the days of the week features in addition to its OHE to express this feature in two ways.

4.2 Clustering

To enrich the TFT model with more static information about each product, we propose a clustering-based encoding of the products' markdown-related features. The idea is based on the following logic. By clustering time series, similar time series can be found, and as such, their forecasting should be similar, and the TFT should

be able to use this information to its advantage. To perform such clustering, we use ROCKET to transform time series into time-independent data and apply the K-means clustering algorithm. Through the K-means, the cluster label each product belongs to is extracted in addition to each product's distance to each cluster centroid. The cluster labels are used as static categorical covariates, and the distances to the cluster centers as static numerical covariates in the TFT.

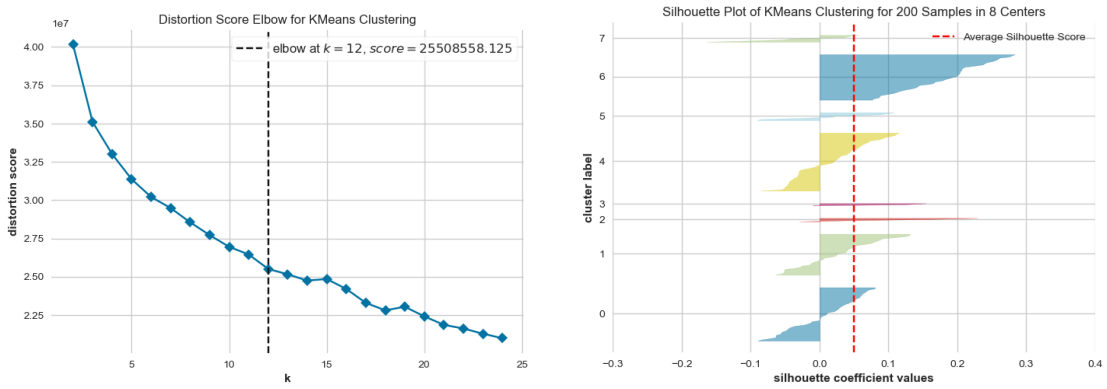
For clustering the products, six markdown-related features are used: *markdown units sold*, *stock markdown*, *discount*, *days to expiration*, *new units marked down*, and *units expired*. These features were chosen because we want the clusters to reflect patterns in how the products' markdown sales behave during markdowns. The dimension of the features used for clustering is (*number of products*, *number of features*, *number of samples*), where the number of samples is the number of days in the products' time series.

4.2.1 Clustering Algorithm

Firstly, the data is normalized (as the authors of the ROCKET paper [36] recommend to do) with a standard scaler transforming the data to have zero mean and unit standard deviation. Then, the ROCKET transformer with the default of 10'000 kernels is fitted with the training set, transforming it from the dimensions (*number of products*, *number of features*, *number of samples*) to (*number of products*, 20'000). The second dimension of size 20'000 comes from the fact that the `scikit-learn` library's [41] implementation of ROCKET sets the transformation output to be two times the number of chosen kernels. The two-dimensional transformed data is then used to fit a simple K-means algorithm with k_{kmeans} clusters to learn the clusters in the dataset. The fitted K-means algorithm is then used to obtain the labels and distances to the cluster centroids for the products in the training, validation, and test sets.

4.2.2 Choosing Optimal Number of Clusters

The optimal number of clusters k^*_{kmeans} is chosen by fitting the K-means algorithm for a range of different k_{kmeans} and analyzing which k_{kmeans} results in the most optimal clustering. Performing a simple elbow test [44] on the plot of distortion scores (sum of squared distances between each item and its cluster centroid) fails in this case as the plot is missing a clear inflection point (see Figure 4.2 (a)). Instead, we analyze the *Silhouette score* and the number of products assigned to each cluster for all k_{kmeans}s tested. The optimal k^*_{kmeans} is defined as the k_{kmeans} that achieves the highest Silhouette score while not having a significantly unbalanced number of products in each cluster [45]. This is done by visualizing the individual Silhouette score residuals for a k_{kmeans} as in Figure 4.2 (b) where the dashed red line is the mean Silhouette score and the width of the clusters represents how many products that are assigned to them. When selecting the optimal k_{kmeans}, we look for the k_{kmeans} where all individual clusters' Silhouette score surpasses the mean Silhouette score and then choose the instance where the clusters are as equal as possible. After assessing the



(a) Distortion scores for fitted K-means models with $k_{kmeans} = \{2, ..., 25\}$.

(b) Silhouette plot for the chosen K-means with $k^*_{kmeans} = 8$.

Figure 4.2: Supporting plots visualizing how the optimal number of clusters for the K-means algorithm was chosen.

Silhouette plots for several k_{kmeans} values, $k^*_{kmeans} = 8$ is selected. Other candidate Silhouette plots are attached in Appendix B Figure B.1 and B.2.

4.3 Baseline models

This section describes the baseline model's parameter settings, feature types used, and any changes in the models that differ from their traditional implementation. All models, including the Temporal Fusion Transformer, were implemented with the `Darts` [46] library for time series forecasting.

The models presented in this section were chosen as the baselines against the more complex TFT model. The reason behind using different models as a baseline is to evaluate the necessity and potential gains in the performance of a more complex model such as the TFT. These baseline models were chosen as they are increasingly more complex. From the most straightforward model for demand forecasting, the Exponential Smoothing model, to the slightly more complex Croston model historically proven to work better for intermittent data. Lastly, the RNN model, which is a far more complex model compared to the other two baseline models, is a standard ML model for predicting sequential data. The RNN is, therefore, on the ladder of complexity just below the TFT model. RNN models are, more or less, the most complex family of models for demand forecasting that can be used before looking at alternative transformer-based models.

4.3.1 Exponential Smoothing

Exponential smoothing is the simplest model of the chosen baselines and performs its predictions only on the univariate time series. As there are no trends, the only optional parameter that can be adjusted is a *seasonal periods* parameter that can be used to set the expected lengths of the seasonal periods in the data. As seen in

Section 3.2.3, there is seasonality in most of the data that need to be accounted for. Currently, the seasonality is automatically found by the `Darts` library by evaluating mode lag spikes in the autocorrelation function and set to 4 (the same as the mode seasonality lag in Section 3.2.3).

The predictions for Exponential Smoothing are performed by fitting the model with the univariate target time series up to a time step t and then predicting n time steps into the future. Since the Exponential Smoothing model is not "trainable" in the same sense as ML models, the fitting is required for every prediction we make. However, due to the model's simplicity, it is fast, so fitting the model for every prediction is not a limiting factor.

4.3.2 Croston's Method

For comparison, the TSB variant of Croston's method is used as it is the most suited version for intermittent time series that also display signs of seasonality. As described in 2.3.2, Croston's TSB variant takes two parameters, α_p (β) and α_d (α), which adjust the periodicity and demand volume smoothing, respectively. The optimal values found for these are $\alpha_p = 0.05$ and $\alpha_d = 0.2$. These are found by performing a grid search through α_p and α_d values in the range $\alpha_p = (0.05, 0.1, 0.15, 0.20, 0.25, 0.30)$ and equally so for α_d. Like the Exponential Smoothing model, the Croston model only uses the univariate target time series as the basis for its prediction.

4.3.3 Recurrent Neural Network

The RNN is the most capable model of the baselines as it, in addition to the target series, also uses future covariates as an instrument when predicting. RNNs do not usually support future covariates as inputs. However, Dart's implementation does this by trading the ability to use past covariates to use future covariates. To enable RNNs to use future covariates, the time series containing future information is shifted some number of steps back. For example, shifting a time series 1 step back would make the future information, which previously was at time step $t + 1$, move to time step t. The input to the RNN is thus a multivariate time series containing the target and the shifted future covariates.

The implemented RNN uses two recurrent layers containing LSTM units. We use an input length (lookback window) of 21 and perform stepwise predictions, always predicting one time step into the future. The complete configuration of the RNN used is summarized in Table C.1 in Appendix C.

When training the RNN model, an early stopping criterion is used to avoid overfitting, interrupting the training when the model no longer improves on the validation set. This is done because the statistical baseline models cannot be overfitted, and thus, we want to ensure that they are compared against ML models that are similarly not overfitted. The early stopping criterion monitors the validation loss of each epoch and stops the training when the validation loss has not changed by more than 0.025

for the past ten epochs, thus showing signs that the model has stabilized.

4.4 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) model is the most complex of the models evaluated in this thesis. Like the other models, we implemented its base model through the `Darts` library [46]. We further extended this base implementation with cluster encoding and added additional functionalities for interpretability purposes due to the lack of support in the `Darts` library.

The TFT uses the same data types as the RNN (target and future covariates) and, in addition, also past and static covariates. However, since TFT natively supports future covariates, there is no need to shift the future covariate time series as with the RNN since TFT uses an encoder-decoder architecture, as shown in Figure 2.7.

The chosen parameter values for the TFT are similar to those used for the RNN to make it a fair base of comparison. The complete configuration of the TFT is attached in Appendix C Table C.2. Furthermore, the same early stopping criterion is used for the TFT as for the RNN, that is, less than 0.025 deviation in validation loss for past 10 epochs.

4.4.1 Static Covariates

The static covariates used for the TFT are divided into two types, categorical and numeric, and come from the products' data and the cluster encodings. The static covariate features from the products' data are department, group, assortment, and price. All of these, except for the price, are integer-encoded categorical features, as described in the preprocessing step. For the cluster encodings, we use the predicted labels and the distance to each cluster centroid for each product. When using 8 clusters, this results in one additional integer encoded categorical feature from the labels and 8 additional numerical features for the distance to each cluster centroid.

4.4.2 Interpretability

One of the TFT model's primary benefits is its support for interpretability. This interpretability is a consequence of the variable selection networks (VSNs) and multi-head attention mechanism, which allows for extracting and interpreting the per-time-step feature weights and the per-time-step attention weights.

Variable Importance

The variable selection weights can be extracted individually from three VSN sources, static covariates, encoder, and decoder weights. The encoder VSN contains both the past and future covariates, while the decoder contains only the future covariates as indicated by Figure 2.7. To interpret the TFT model, we use the same method as the original paper's authors on the TFT [28]. This entails analyzing how the

weights are distributed within each VSN source to see how much the TFT relies on the various features. Furthermore, for the encoder VSN, we also extract how the weights are distributed temporally relative to the prediction point to analyze their importance over time.

The weights from the three VSN sources have the following dimensions:

- Encoder: $(n_{samples}, l_{lookback\_window}, n_{past\_features})$
- Decoder: $(n_{samples}, l_{forecast\_horizon}, n_{future\_features})$
- Static covariates: $(n_{samples}, n_{static\_features})$

where $n_{samples}$ are the number of samples that are sampled from the VSN, $l_{lookback\_window}$ is the number of historical time steps used for predictions, $l_{forecast\_horizon}$ is the number of days we perform predictions for at a time (1), and $n_{past\_features}, n_{future\_features}$, and $n_{static\_features}$ is the number of features used in a particular VSN source.

To perform the analysis of the encoder VSN's weights, we forward propagate all active days' time series through the TFT and obtain matrices of dimensions defined as before where $n_{active\_days} = n_{samples}$, is the number of active days in the data set. For each of the $n_{features}$ features and $l_{lookback\_window}$ time steps, values (from the first dimension) are sampled to obtain the three quantiles 0.1, 0.5, and 0.9 of the weight distribution for each feature time step pair, yielding results in the format visualized in Table 4.1. To obtain the weights for the static and future covariate features, the sampling is only performed for every feature.

Table 4.1: Example of the format for the extracted temporal VSN weights.

Temporal VSN weights example										
Time steps	$t - l_{lookback\_window}$			...	$t-2$			$t-1$		
Quantiles	0.1	0.5	0.9	...	0.1	0.5	0.9	0.1	0.5	0.9
Feature X's weights	...	...	...	...	...	...	...	...	...	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Furthermore, to obtain the past covariate features' general importance, unconcerned of their temporal importance relative to the prediction point, we average the matrix over the second $l_{lookback\_window}$ dimension. Thus instead, obtaining a matrix of size $(n_{active\_days}, n_{features})$ where we again sample the quantiles 0.1, 0.5, and 0.9 from the first dimension yielding each feature's general importance in the network.

Attention

To obtain the attention weights of the TFT model, we forward propagate all active days the same way as we for the variable importance weights. The obtained attention weights have the dimensions $(n_{active\_days},\ l_{lookback\_window},\ 4,\ l_{lookback\_window} + l_{forecast\_horizon})$. The second dimension is the query positions of the attention network, and to obtain the weights for the prediction time t, we only consider the very

last row in this dimension. The third dimension represents the number of attention heads (4), which we average. The fourth and last dimension correlates to the number of time steps in the TFT, 21 past values, and 1 future value making up 22 time steps in total in the attention network. To obtain the TFT attention over the 22 time steps, we sample the three quantiles 0.1, 0.5, and 0.9 from the first dimension for each of the 22 time steps in the fourth dimension. The resulting weight samples are similar to those for a single feature in Table 4.1 but with additional samples for time step t.

4.5 Evaluation

The evaluation strategies of the models are centered around the two prominent use cases we have identified for our problem definition. The first use case is to predict the number of markdown sales for the next day during an active markdown period (using a forecast horizon of one). In this case, all the information up until the prediction date is available, including past and future covariates. As mentioned previously all future covariates used are covariates that are known in advance.

The second use case is to forecast at the start of a markdown period how many products will sell during the whole period. For this case, the forecast horizon is set to the number of active markdown days. The predictions are still performed one step at a time but are summed up over the markdown period to obtain the total sales over the whole period. This case is generally more complex as only the past covariates up to the time of prediction are known. This is not an issue for the models that do not use the past covariates – but for the TFT model that does use and need past covariates up to the day before the day of prediction, the past covariates need to be approximated or assumed. The details regarding this issue are presented later.

Furthermore, a similar issue arises for the models that use future covariates (RNN and TFT) for the features only known one day ahead, such as the stock. These must be adjusted as they depend on the previous day's predictions. For example, the stock has to be adjusted so that the stock for the next day equals the previous day's stock minus the number of predicted sales for the previous day. Because of this, we implemented a custom recursive forecasting function for the models using future covariates (RNN and TFT) that predicts one step at a time and adjusts the future covariates considering the previous days' predictions.

4.5.1 Evaluation Metrics

In evaluating the models, the four metrics presented in Section 2.8 are used. The MAD metric is utilized to assess day-by-day predictions, while for aggregated markdown period predictions, the CFE metric is applied. The reason for using these different metrics is that these two scenarios have different objectives. In the day-by-day prediction scenario, the primary objective is to measure how well the models fit the actual series, with the goal of being as close as possible to the actual values. Therefore, the MAD metric is used to assess the accuracy of each day's prediction.

However, in the aggregated markdown period prediction scenario, the total predicted quantity over the entire period is the focus, regardless of whether the models predict sales on a specific day. Therefore, the CFE metric evaluates the cumulative forecast error over the entire markdown period. It is important to note that all metrics are presented as the mean over the period length. Thus, the day-by-day MAE is the mean absolute error per day, while the markdown period CFE is the cumulative forecast error per markdown period. Lastly, the PBMAD and PBCFE metrics are compared to the baseline of exponential smoothing, allowing for easier relative comparisons.

4.5.2 Evaluation Constraints

To evaluate the models independently and against each other, a number of ground rules were established to make the cross-model evaluations fair and reflect the problem's true nature.

Firstly, the models are only evaluated on active periods, i.e., where *days to expiration* feature is positive. We choose to evaluate the models on only these periods as this would be the only use case of the models in our domain since the time of markdown periods and their durations are known, making the problem of predicting sales for inactive periods trivial. Predicting the sales for inactive periods is trivial as the sales of markdown products will always be zero when there is no availability. Furthermore, including inactive periods in the predictions would contribute to errors in the overall evaluation of the model, making the prediction errors less interpretable due to the intermittent data.

Secondly, no privileged information is used when training or evaluating the models. Privileged information would entail letting the models know about information in the future that would not be available at the prediction time. For example, the available stock during training on the first day of an active markdown period is known (non-privileged information). However, the model cannot know it past that day as that would contribute to the model training on information not known at prediction time. Thus for the ML models that use future covariates, we only forecast one time step at a time. Another example of where avoiding privileged information becomes challenging is when using past covariates for multi-horizon forecasting. Privileged information is not an issue during training, with the forecast horizon set to one. However, as previously mentioned, when predicting the sales for multiple days into the future, past covariates must be handled cautiously to prevent privileged information from being used.

4.5.3 Recursive Forecasting for Multi Horizon Predictions

The custom recursive forecasting function implemented for the TFT and RNN models allows us to make forecasts further into the future than just one day at a time. The function identifies the markdown period's length and performs the first prediction for time step t using the known past covariates up to time $t-1$ and known future covariates up to time t. This first step yields a prediction for the number of markdown

sales for the current time step t. The predicted number of sales is subtracted from the available stock at time step t to update the stock for prediction at time step $t+1$. Moreover, the past covariates need to be adjusted for predictions at time step $t+1$ and onwards since these values are unknown at time t (avoiding using privileged information). We solve this by setting the missing values for the past covariates at time steps later than $t-1$ to their last known value, i.e., the value at $t-1$. These updates and step-vise predictions are then repeated recursively until the end of the markdown period.

4.5.4 Cross-Model Evaluation

The first goal of the evaluation is to cross-evaluate the models to compare their performance. For the cross-model evaluation, the first data-splitting method (left side of Figure 3.5) is used to enable evaluation between all models and evaluate them on both next-day predictions and summed-up whole markdown period predictions. The results give insight into the performance of each model on products on which they have been trained but on periods not seen during training.

4.5.5 Evaluation of Machine Learning Models

The ML models, RNN and TFT, are also evaluated on how they perform on entirely unseen products by using data splitting method 2 (right side of Figure 3.5). This is done to investigate how well the models can generalize from the training set to the test set. Evaluating how well the models can generalize is interesting for two reasons. Firstly, in some cases, it may be unfeasible to train the model on all products; therefore, only a selection of the products may be used for training. Secondly, the product assortment may be rapidly changing due to new products entering the assortment, making retraining the model to account for the new products unfeasible. Therefore, how well the models generalize to new products is an essential aspect of evaluating the models.

4.5.6 Evaluation of Temporal Fusion Transformer

Due to its complexity and the possibility of using features unavailable to the other evaluated models, the TFT model is chosen as a subject for further evaluation. Since it offers tools for interpretability through the VSNs, investigating which input features it puts more importance on can offer more insight into the problem and enables us to draw conclusions on which features can be omitted without sacrificing performance.

Furthermore, to evaluate whether the proposed cluster encodings have had a positive effect on the performance of the TFT, we will compare TFT models trained using cluster encoding against a TFT that does not. The research question is whether there is a statistically significant improvement in using cluster encodings.

5

Results

This chapter presents the results of how the models perform using the different evaluation methods in the two identified use cases. Firstly, the results of the cross-model evaluation showing how the models stack up against each other are presented. Next, further evaluation of the TFT model is presented, beginning with the result of the proposed cluster encodings, followed by an interpretation of what the TFT has learned from the data.

5.1 Cross-Model Performance

This section presents the results of the cross-model evaluation showing how the models perform compared to each other. The models were evaluated by the two previously described use cases, only predicting one day into the future and aggregated predictions over whole markdown periods.

Table 5.1 shows the results of the two evaluations. The MAD score is for the next-day prediction, representing the average daily prediction error. The CFE score is used for the aggregated predictions over whole markdown periods, representing the average error in the aggregated predictions for markdown periods. Both scores are marked with μ in the table, representing the mean scores. The table also includes the standard deviation σ of the scores, the mean scores' 95% confidence values using the standard deviation σ, and the percentage difference between the models using Exponential Smoothing as the baseline for each score and evaluation case.

All models in the cross-model evaluation were tuned to the best of their abilities and trained on all the available covariates data each model supports. The Exponential Smoothing and Croston models only use the target time series since they neither support future, past nor static covariates. The RNN was additionally trained on all the future covariates up to and including the prediction time step. Lastly, the TFT was trained on all past, future, and static covariates, including the cluster encodings. All the models were trained with the parameter setups presented

in Section 4.3 and 4.4 and evaluated on the top 200 products in the data set using data splitting method 1.

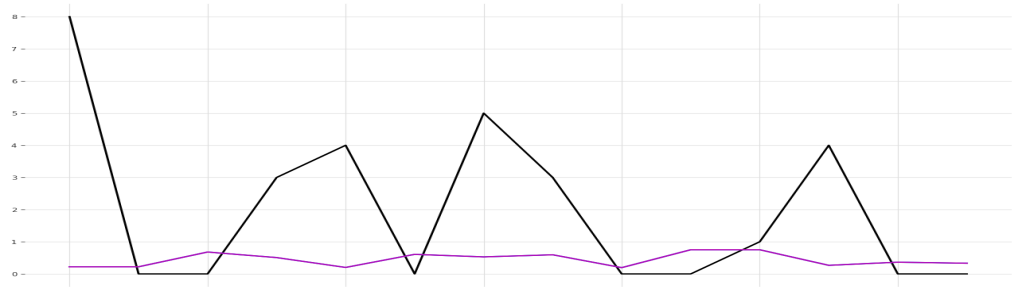
Furthermore, the RNN and TFT models use the recursive forecasting function for predictions over whole markdown periods. This is necessary to avoid using privileged information when predicting multiple time steps ahead. The Exponential Smoothing and Croston methods do not need additional handling since they only use the target series up to the prediction point to forecast future sales.

Table 5.1: Table containing the results of the models evaluated on next-day predictions and aggregated predictions over whole markdown periods. The best result for each metric and each prediction type is marked in **bold**. The PBMAD and PBCFE (in parenthesis) are compared to the Exponential Smoothing (Exp. S) model. Each model's results contain the mean score μ, standard deviation σ, and the 95% confidence interval (CI).

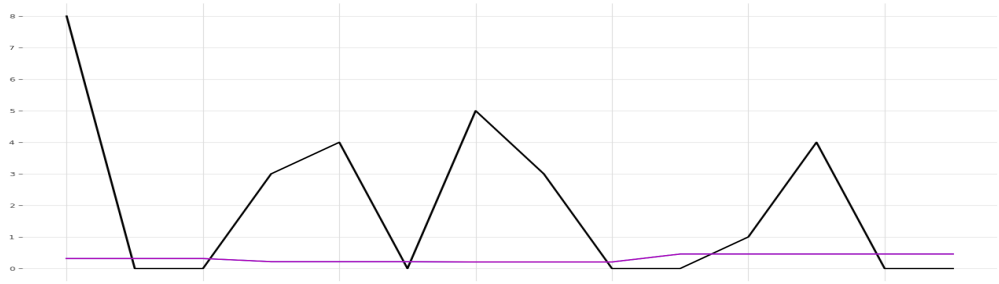
Models' Performance			
		Next-Day Predictions MAD	Markdown Period Predictions CFE
Exp. S	μ	1.04 (0%)	-2.14 (0%)
	σ	2.11	3.89
	CI	$\mu \pm 0.08$	$\mu \pm 0.26$
Croston	μ	1.03 (+0.1%)	-2.34 (-0.9%)
	σ	2.12	3.83
	CI	$\mu \pm 0.08$	$\mu \pm 0.25$
RNN	μ	0.86 (+17.3%)	**-0.97 (+54.7%)**
	σ	1.51	3.99
	CI	$\mu \pm 0.06$	$\mu \pm 0.26$
TFT	μ	**0.81 (+22.1%)**	-1.02 (+52.3%)
	σ	1.46	3.88
	CI	$\mu \pm 0.06$	$\mu \pm 0.26$

The results shown in Table 5.1 show that the TFT model outperforms the other methods for next-day prediction, and RNN takes the edge for aggregated markdown periods. Additionally, all models have negative CFE scores, indicating that the models generally have a negative bias, i.e., the models underestimate the number of sales.

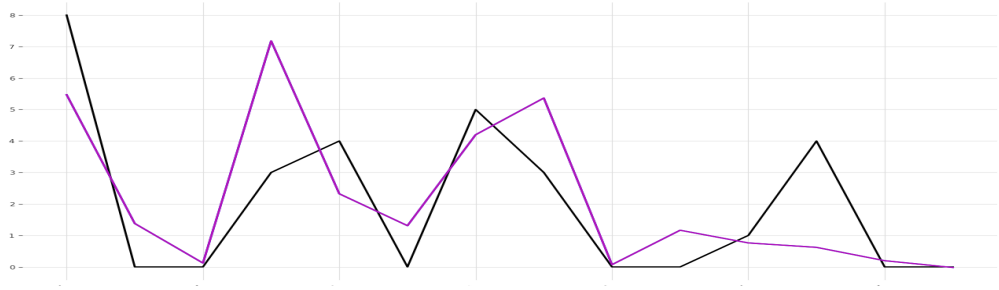
Figure 5.1 shows an example of how each model forecast. The plots for each model in the figure are predictions for the same product and only show the active periods concatenated, omitting the inactive periods. The TFT and RNN model



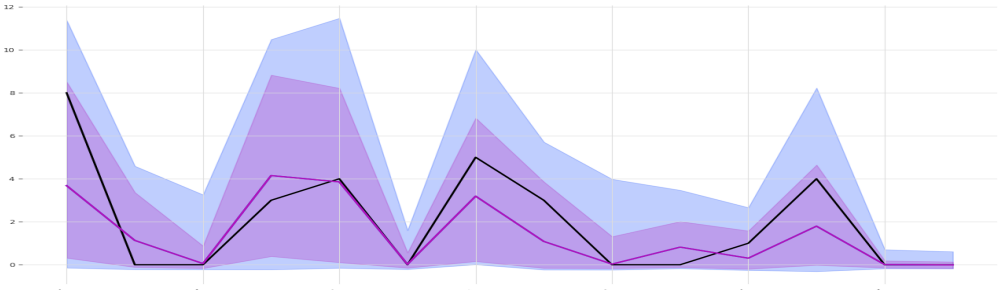
(a) Exponential Smoothing (MAD=2.00, CFE=-1.56)



(b) Croston's model (MAD=2.02, CFE=-1.67)



(c) RNN (MAD=1.39, CFE=0.16)



(d) TFT (MAD=1.02, CFE=-0.51).

Figure 5.1: A figure showing a sample true time series (black) with active periods and the prediction (purple) by each of the models mentioned in Table 5.1. Observe how (a) and (b) continuously predict close to zero, while (c) is prone to overestimation and (d) to slight underestimation. (d) also shows 99% (blue) and 95% (purple) confidence as shaded areas.

.

seems to fit the actual time series better than the Exponential Smoothing and Croston model when there are actual sales. Furthermore, Figure 5.1 also shows that the RNN is more inconsistent in its day-to-day predictions as it sometimes overestimates the sales and other times underestimate the sales, while the TFT more consistently slightly underestimates the sales. This is a contributing factor to the fact that the RNN outperforms the TFT in aggregated predictions as the under and over-estimation equals out in a value that is closer to the true sales over multiple days.

5.2 Performance on Unseen Products

The ML models (RNN and TFT) can be trained and evaluated on different sets of products, which the statistical models (Exponential Smoothing and Croston) inherently cannot. Therefore, the ML-based models are also evaluated on how they perform on unseen products using data splitting method 2. This entails training and evaluating the ML models using separate sets of products. Except for the change in training, validation, and test sets, the ML models for this evaluation use the same setup as before and perform their predictions in the same way. The models were trained on 120 products and evaluated on 40. The remaining 40 products were used as validation data during the training for the early stopping criterion to monitor for overfitting.

The results of how the RNN and TFT perform on unseen products are presented in Table 5.2, showing the results of how the models perform for next-day predictions and whole markdown periods. The table shows the mean results (μ), standard deviation (σ), and the scores' 95% confidence values for the two prediction cases. Additionally, using the RNN as a baseline, the percentage difference for the TFT model is shown to the right of each score. The ML model's evaluation table shows similar results to the cross-model evaluation. The TFT is slightly better at next-day predictions, while the RNN is better at predicting the aggregated sales over whole markdown periods. This is likely as before because the RNN is more inconsistent in its predictions, as the larger standard deviation indicates. This, as mentioned earlier, averages out in an aggregated prediction that is closer to the true sales.

5.3 Temporal Fusion Transformer

The TFT model is subject to further evaluation to evaluate how the proposed cluster encodings affect the TFT's performance—also evaluating what the TFT has learned from the data by analyzing the weights from the VSNs and attention network.

5.3.1 Cluster Encodings' Influence

Three TFT models were trained with differing cluster and static covariates setups to empirically evaluate the effect of the cluster encodings and other static covariates. The results of these three are shown in Table 5.3. The first TFT model in the table

Table 5.2: The results of the models evaluated on next-day predictions and aggregated predictions over whole markdown periods for unseen products. The best result for each metric in each prediction type is marked in **bold**. The PBMAD and PBCFE (in parenthesis) are compared to the RNN model. Each model's results contain the mean score μ, standard deviation σ, and the 95% confidence interval (CI).

Models' Performance on unseen products			
		Next-Day Predictions MAD	Markdown Period Predictions CFE
RNN	μ	1.02 (0%)	**-1.26 (0%)**
	σ	2.00	4.08
	CI	$\mu \pm 0.09$	$\mu \pm 0.33$
TFT	μ	**0.93 (+8.8%)**	-1.73 (-37.3%)
	σ	1.54	3.53
	CI	$\mu \pm 0.07$	$\mu \pm 0.29$

was trained only using the static covariates from the data set, omitting the cluster encodings. The Second model was trained using the opposite setup, only using the cluster encodings and no other static covariate from the data set. Lastly, the last TFT model in the table is the same model used in the ML model evaluation of unseen products using both static covariates and cluster encodings. The three models' loss during their training is shown in Figure 5.2. Table 5.3 shows mixed

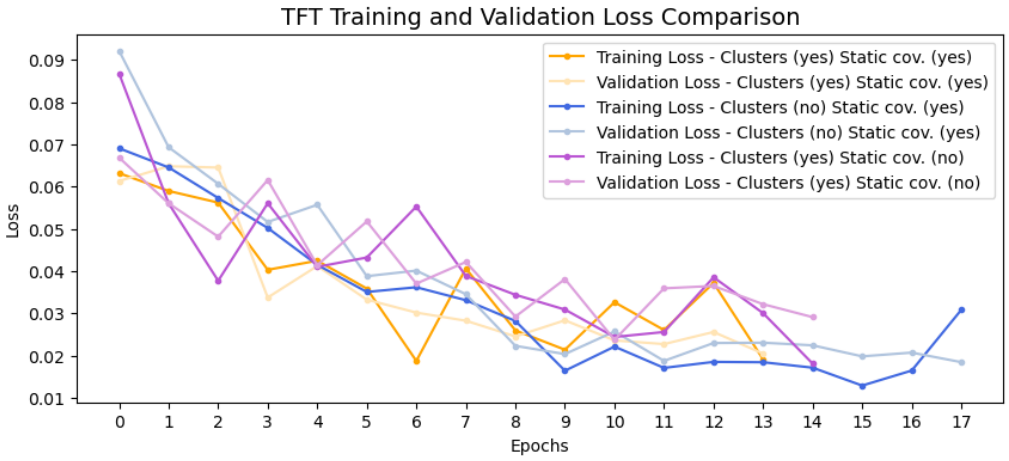


Figure 5.2: Training and validation loss over epochs for the TFT model with and without cluster encodings.

results of the TFT models' comparison. Although the scores for next-day predictions are close between the models, the best-performing model is the TFT with only static covariates and no clusters. For the aggregated predictions over whole markdown periods, the TFT model with only clusters and no other static covariates outperforms the other two models by quite a substantial margin that is statistically significant.

Table 5.3: Table containing the results of three TFT models with differing setups evaluated on next-day predictions and aggregated predictions over whole markdown periods, μ showing the mean scores, σ their standard deviation, and CI the 95% confidence interval of the mean μ. The best result for each metric in each prediction type is marked in **bold**.

Performance of TFT Without Cluster VS. With Clusters			
		Next-Day Predictions MAD	Aggregated Markdown Periods CFE
TFT	μ	**0.91**	-1.80
clusters (no)	σ	1.68	3.83
static cov. (yes)	CI	$\mu \pm 0.08$	$\mu \pm 0.31$
TFT	μ	1.00	**-0.90**
clusters (yes)	σ	1.49	3.91
static cov. (no)	CI	$\mu \pm 0.07$	$\mu \pm 0.32$
TFT	μ	0.93	-1.73
clusters (yes)	σ	1.54	3.53
static cov. (yes)	CI	$\mu \pm 0.07$	$\mu \pm 0.29$

5.3.2 Interpretability

The TFT model was evaluated from an interpretability perspective using its variable selection networks and attention network. Since the primary interest lies in how the TFT makes its prediction when there is availability, the analysis is restricted to the weights of the networks during predictions of active periods only. The evaluation was done by forward propagating active periods in the data set through the TFT, thus creating weights to sample from that only regard active periods. The TFT model used for interpretability results here is the model that was trained on both static covariates and cluster encodings.

Variable Importance

How the TFT model makes its prediction can be interpreted from its variable selection networks' weights, giving insight into how the model ranks the features in terms of importance for the predictions. All weights are obtained by forward propagation of all active periods in the whole dataset of 200 products and sampled by the quantiles 0.1, 0.5, and 0.9 to approximate their distribution. Comparing the upper and lower quantiles to the mean can give an idea about the certainty that a particular feature is important. The features' time-independent importance and the temporal importance relative to the prediction point are presented in this section.

The features' time-independent importance are presented in Tables 5.4, 5.5, and 5.6, sorted from most to least important. Appendix D contains the exhaustive lists of all features' importance. Table 5.4 shows the TFT encoder's VSN weights which

Table 5.4: Top 5 most important features and bottom 3 least important features in the encoder VSN. Values are extracted weights from the VSN.

Encoder Weights of VSN			
Past & Future Cov.	Quantiles		
	0.1	0.5	0.9
Units Expired	0.1038	0.1061	0.1079
Stock Markdown	0.0998	0.1013	0.1038
Value Sold Store	0.0965	0.0993	0.1022
Days To Expiration	0.0402	0.0404	0.0405
Value Sold	0.0340	0.0347	0.0353
⋮	⋮	⋮	⋮
Month 5	0.0017	0.0017	0.0017
Units Sold Promotion	0.0015	0.0015	0.0015
Day	0.0010	0.0011	0.0011

Table 5.5: Top 5 most important features and bottom 3 least important features in the decoder VSN. Values are extracted weights from the VSN.

Decoder Weights of VSN			
Future Covariates	Quantiles		
	0.1	0.5	0.9
Units Sold Markdown Data	0.8439	0.8766	0.8784
Day of week 5	0.0127	0.0144	0.0376
Stock Markdown	0.0073	0.0075	0.0076
Day of week 2	0.0042	0.0043	0.0045
Month 3	0.0042	0.0043	0.0044
⋮	⋮	⋮	⋮
Month 6	0.0013	0.0014	0.0024
Day Sin	0.0009	0.0011	0.0012
Month 12	0.0004	0.0004	0.0010

contain the past and future covariates' importance up to the prediction point. Table 5.5 shows the TFT decoder's VSN weights containing only the future covariates at prediction time. Lastly, Table 5.6 shows the TFT's static covariate VSN's weights.

The top 5 most important features' temporal weight values are visualized in Figure 5.3 showing the encoder's VSN weights up to the prediction point. This graph shows how each feature's importance is weighted over time for the predictions, where $t-1$ is the day before the prediction point and $t-21$ is the day 21 days before the prediction point. Looking at the graph, no significant change can be seen in the variables' importance over time. The features' values are close to equally important for the predictions at all time steps relative to the prediction time.

Attention

The attention network weights for each time step are shown in Figure 5.4, presenting the TFT's attention over time. From plot (a) in the figure, it can be seen that the attention at prediction time t is overwhelmingly higher than for the past time steps. Due to this large difference in weight values, a log-scaled version of the attention weights is also plotted, as shown in plot (b) of Figure 5.4. The original values of the

Table 5.6: Static covariates feature importance based on sampled weights from the static covariates VSN. Values are extracted weights from the VSN.

Static Covariates Feature Importance			
Static Covariates	Quantiles		
	0.1	0.5	0.9
Department	0.0142	0.1823	0.4870
Cluster Distance (3)	0.0718	0.1068	0.1809
Assortment	0.0176	0.0769	0.3298
Cluster Distance (2)	0.0423	0.0741	0.1602
Cluster Distance (4)	0.0439	0.0607	0.1076
Cluster Distance (1)	0.0333	0.0587	0.0918
Cluster Distance (0)	0.0398	0.0563	0.0744
Cluster Label	0.0345	0.0507	0.0841
Cluster Distance (7)	0.0273	0.0455	0.0809
Suggested Price	0.0215	0.0386	0.0680
Store	0.0050	0.0224	0.0942
Group	0.0060	0.0189	0.0910
Cluster Distance (5)	0.0078	0.0142	0.0253
Cluster Distance (6)	0.0048	0.0085	0.0144

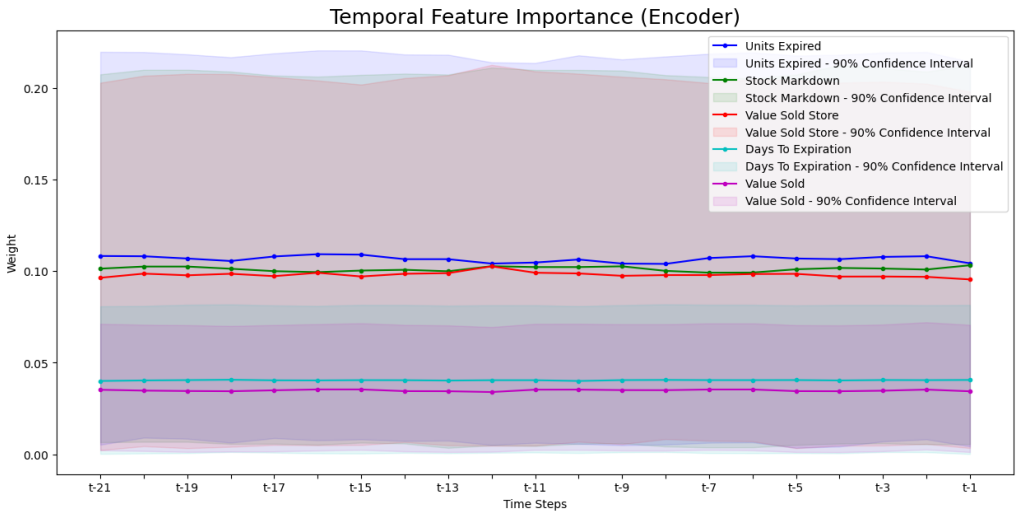
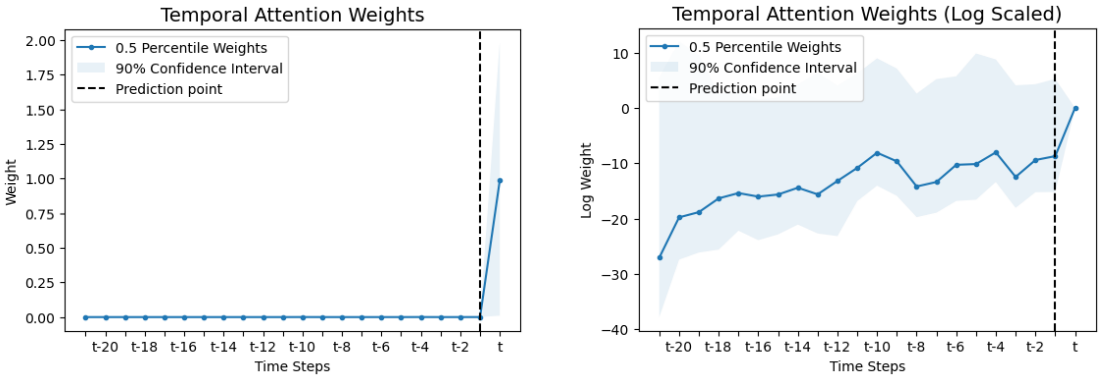


Figure 5.3: Temporal feature importance relative to prediction point, encoding weights from the VSN.

temporal attention weights are attached in Appendix D Table D.3.



(a) Temporal attention weights of TFT.

(b) Log-scaled temporal attention weights of TFT.

Figure 5.4: Temporal attention weights of TFT showing both the ordinary and log-scaled values.

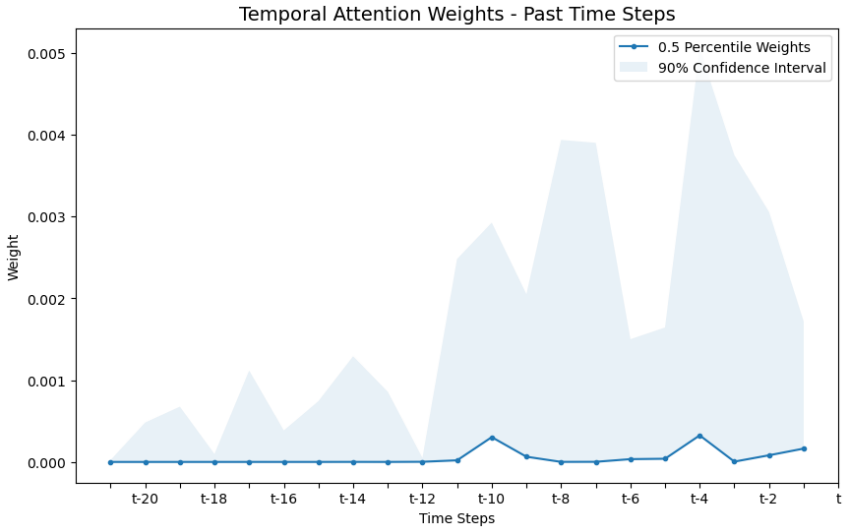


Figure 5.5: Temporal attention weights of TFT showing only weight values for past time steps.

Looking at the attention weights in Figure 5.5 restricted to only showing past time steps, the attention at time steps $t-1, t-4$, and $t-10$ slightly stand out from the rest. Although the weights of these could be seen as negligible compared to the weights at time step t, it is an interesting effect of the attention mechanism. Furthermore, it is worth noting how the confidence interval of the values decays moving away from the prediction time, showing signs that for at least some products, the past values might play into the predictions and that this importance decays moving further back in time.

5.4 Clustering

Figure 5.6 shows the resulting clusters of the K-means clustering using 8 clusters and transformed with t-SNE (t-distributed Stochastic Neighbor Embedding)[47] to enable visualization in two dimensions.

Figure 5.6: Clusters of the 200 products used transformed into two dimensions with t-SNE transformation. The axes represent the cluster encodings with their dimensions reduced (transformed) from 9 to 2 dimensions for illustration purposes.

5.4.1 Correctness of Clustering

Evaluating clustering with no labels, as in our case, is difficult. Except for measuring the compactness of the clusters using a metric such as the Silhouette score, there is no intuitive way of measuring the correctness of product cluster assignments without any actual (true) labels.

To understand what the created clusters might represent, we perform an alternative evaluation using the categorical product features: department, group, and assortment. As these features contain categories with which the products are labeled, we use them as true labels for each cluster. To decide which department, group, and assortment label each created cluster should represent a majority count of each cluster's assigned categorical product label is performed. For example, if **Department A** is the most frequently assigned label for a particular cluster, that cluster's actual department label is set to **Department A**. This is done for all clusters for each feature: department, group, and assortment categories. A regular accuracy score is then used to evaluate these results, comparing the assigned label against the majority count.

The alternative evaluation method implemented to understand the cluster representations is based on the hypothesis that products that are similar to each other

also have similar sales behavior, i.e., should approximately belong to the same clusters. We are essentially measuring the agreeableness of the assigned products in terms of their department, group, and assortment within each cluster.

Table 5.7: Accuracy of clusters using assumed cluster labels from each cluster's majority count of assigned department, group, and assortment labels.

Accuracy of Clusters Using Assumed Labels		
	Accuracy	# Unique Labels
Department	74.5%	3
Group	51.5%	12
Assortment	16.0%	64

Table 5.7 shows the accuracy scores for the clusters using the assumed labels gained from performing the majority count of each cluster. The results indicate that the clusters found correlate with the product categories in the dataset. This is an exciting result as the clustering only uses the time series *markdown units sold*, *stock markdown*, *discount*, *days to expiration*, *new units marked down*, and *units expired*, having no direct information link to the product categories – but still manages to find similar groups as present in the dataset. Worth noting is that the number of clusters constrains the maximum achievable accuracy. Since the clustering is performed with 8 clusters, the department is the only category that could theoretically reach 100% accuracy. We expect that if the model used an increased number of clusters the accuracy would increase as a consequence. However, that would hurt the efficiency of the clusters in general as the optimal number of clusters found in the previous analysis in Section 4.2 was 8.

6

Discussion

This chapter covers the findings and insights gained from the previous results chapter. The results of the forecasting and what they signify are discussed, and how forecasting works in the absence of past data is explained. The impact of cluster encoding on the TFT model's outcomes is delved into, followed by the conclusions that can be drawn from interpreting the TFT model. Potential future research and ethical considerations related to the thesis findings are also addressed.

6.1 Forecasting

The deep learning methods outperform the statistical measures on next-day and markdown period predictions when evaluating the cross-model evaluation as seen in Table 5.1. However, when evaluating the next-day predictions, there seems to be only a small difference in their performance, but, only evaluating the models by the metric scores is insufficient. Manually evaluating the prediction output, as seen in Figure 5.1, highlights substantial differences in their performance. Here, the critical issue with the statistical models becomes apparent. The statistical models continuously predict low sales, improving their MAD score on periods with no or close to no sales.

In contrast, the predictions of the RNN and TFT models are much better aligned with the actual values. Another aspect of interest is that while the CFE scores for all models are negative, indicating a negative bias and by period underestimation of sales, Figure 5.1 highlights how the RNN models predict slightly differently from the other models. The RNN model, more frequently than the other models, overestimates the number of sales by a substantial amount, which positively affects the CFE score over more extended periods.

Given the statistical models' definitions and the high amount of observed days of no sales (zeros) in the dataset, it is no surprise that the exponential smoothing and Croston's model continuously predict low values for future sales. Additionally,

compared to the TFT and RNN models, Croston's model and exponential smoothing have no access to other covariates and their weights, as seen in Table 5.4 and 5.5, highlighting the importance of the covariate series when predicting future values.

As a consequence of this, the statistical models essentially only provide satisfactory forecasts in the case of low-quantity sales situations, making the model quite underwhelming as simply predicting no sales at all times would yield relatively similar results. Thus, given the complex nature of the problem as well as the values of covariate series as seen in Table 5.4 and 5.5, the DL models that have access to these features seem better suited to the specific problem addressed in this thesis.

Evaluating the RNN against the TFT model, they both seem similar in their performance, with the next-day predictions favoring the TFT model and the markdown period predictions favoring the RNN. Therefore, in Section 5.2, the models are further evaluated using a series of products not part of the training set. In this scenario, the TFT model slightly outperforms the RNN regarding next-day predictions, while the RNN performs better at markdown period predictions. However, given the confidence interval regarding the true mean, it is clear that neither result is statistically significant. It is interesting to analyze the models' performance based on their standard deviation σ. It reinforces what is also seen in the cross-model evaluation plots in Figure 5.1. The RNN often outperforms the TFT in terms of the CFE scores, but that is itself a consequence of it having learned the behavior of the true series slightly worse (see MAD scores) than the TFT. Having a large variance of continuously under and overestimating the prediction sales, in the end, yields a better CFE score than the TFT.

Another key aspect is that underestimating sales is in itself a more desirable trait in this domain than overestimating. Overestimating the sales leads to increased waste for a system like this, as the belief is that more units will sell than actually the case. On the contrary, being more prone to underestimating decreases the likelihood of waste.

The reason why the TFT model is slightly better than the RNN in terms of MAD score could depend on a number of things. However, looking at Table 5.6 and Figure 5.4 might lend a plausible explanation. Firstly, the static covariates available to the TFT model but unavailable to the RNN likely aid in predicting the sales of unseen products, as in the training set, there exists products with similar static covariates. This theory gets further support in Table 5.3, as the TFT becomes even better at next-day predictions without clusters. Secondly, in the encoder VSN weights (Table 5.4), the number one weighted feature is "Units Expired", a past covariate not available to the RNN. Thus, this indicates that the existence of static and past covariates plays a role in the TFT outperforming the RNN in terms of fitting the actual series.

6.2 Cluster Encodings

The resulting clusters, as an effect of using cluster encodings, showed interesting traits with the data set used in this thesis. Without any knowledge of existing product groupings in the stores, the ROCKET and K-means combination used for clustering showed strong indications of building clusters around the existing groupings in the data set (see Figure 5.6 and Table 5.7).

The results of the cluster encodings show that the initial hypothesis that products that are alike each other, also have similar sales patterns in the markdowns. Exactly which attributes of the time series used for the clustering that produces this result is hard to identify. However, this indicates that the proposed idea of using ROCKET for clustering can contribute useful information to ML tasks working with temporal data. This is especially true in cases where static categorical covariate time series are unavailable.

The results of comparing TFT models with different setups (Table 5.3) show that the TFT model that only uses clustering and no other static covariates achieved the best scores in terms of the aggregated markdown periods evaluation. This model's improvement in score is statistically significant to the other two models that used static covariates and cluster encodings, as well as only static covariates. A hypothesis for why the model with cluster encodings and static covariates performs worse than the TFT with only clusters could be due to what the static and clustered variables encode. While categorical static covariates have little to do with temporal aspects, the clustering is performed fully using temporal data about the markdown series. This could, as a consequence, lead to the cluster distances and labels being better at encoding how the series behaves over time than the static covariates. If this is the case, that would likely lead to a better markdown period prediction performance. However, this hypothesis cannot be confirmed or rejected with the currently performed experiments but would need further investigation.

Furthermore, a side effect of using clusters in the TFT is that the TFT model trained with cluster encodings consistently needed fewer epochs – and thus less time to reach the same loss during training. This is presented in Figure 5.2, showing the loss progress during training for all TFT variants. In other words, besides contributing to the performance of the TFT, using cluster encodings also shows signs of facilitating the training of the TFT.

6.3 Temporal Fusion Transformer Interpretability

This section will discuss the interpretability results of the TFT model. The importance weights of the features obtained from the VSNs will be examined first, followed by a discussion on how the attention of the TFT is distributed.

Feature Importance

In Section 5.3.2 of the results of the feature importance, we can see that the TFT strongly favors some features while close to ignoring others. Looking at the weights of the encoder and decoder VSNs (Table 5.4 and 5.5), most of the important features are the product-related future covariates and the target feature. This is expected as these are also the features that indicate when there is availability in general and, therefore, also good predictors of demand. Furthermore, from the encoder VSN's weights, features like the *value sold* and *units sold ordinary* also obtain relatively high weights. This indicates that past covariates that indicate the traffic in the stores and general demand do, to some degree, have a predictive influence on the target.

In general, because a feature in our presented instance was assigned a low weight does not necessarily mean it is not helpful. The VSNs can ignore a feature if another one already explains its effect. Table D.2 shows an example of this where the features *value sold* and *units sold* are significantly different in their assigned weights, even though they inherently describe the same thing but with different metrics. Therefore it is not always sufficient to look at the assigned weights to prune the system of features. However, the VSNs are not the entirety of the system; thus, they can not be the sole base for pruning features.

Attention

Analyzing how the TFT model distributes its attention over input time steps shows that the TFT puts most of its attention on the prediction time step t (see Figure 5.4 (a)). This could be explained by the fact that it is the only time step containing future covariate features. Additionally, from the results of the data exploration chapter, we also know that the future covariate features are the best predictors for the target series (see Table 3.7). Combining these two contributing factors could explain the model's attention bias towards the prediction time step t. However, the model's overwhelming attention bias toward step t questions the approach of analyzing this problem as a time series forecasting issue. On the other hand, the data exploration chapter results indicate clear benefits of analyzing the presented problem temporally and that there are useful attributes in the past values that can help forecast future sales.

6.4 Future Work

As the results seem to indicate that transformer-based models seem appropriate for the problem addressed in this thesis, a potential avenue for future research is to evaluate the TFT model against other transformer-based models such as Temporal Convolutional Networks (TCN)[48] and N-BEATS [49]. The convolutional aspect of the TCN could suit itself nicely for the long periods of inactivity in intermittent series by up/down-sampling the data for better finding short and long-range dependencies. Similarly, the N-BEATS model might also suit the problem nicely as it, similarly to the TFT model, allows for good interpretability and multi-time series support.

Additionally, the model is faster to train and has shown good performance in time series competitions.

This thesis showed that using ROCKET and K-means clustering to encode time series information for temporal models has potential. Although using ROCKET for clustering is not a far-fetched idea as it has previously been used for time series classification tasks – to the best of our knowledge, ROCKET has yet to be explored for clustering. Additionally, as far as we know, applying cluster encodings on time series using random convolutional kernel transformations to enrich a model with more static information is a novel idea. Therefore, we propose to further explore this idea by applying the proposed method for cluster encodings on other well-explored open-source data sets to evaluate the effects of this method more in-depth.

Another avenue for future research could be to evaluate the breadth of available features that also help explain the behavior of the intermittent series. As future and past covariates can provide value, there are seemingly endless possibilities for evaluating potential features. Some of these include but are not limited to financial features such as inflation, weather forecasts (temperature, sunny, etc.), and store-specific information such as the number of customers per day.

A further identified aspect of potential research is improving the recursive forecasting algorithm for multi-horizon predictions. Currently, the values from the last day previous to forecasting are used; however, this leads to the past covariates staying static and providing much less potential value to the prediction of the target series. Given more time and resources, one could evaluate the possibility of forecasting the future values of the target series and the past covariates. How this would be done more precisely to not be too time-inefficient and complex remains to be figured out.

The last piece of identified future research concerns learning using privileged information (LUPI)[50]. This paradigm could use the future covariates, available when training but unavailable when predicting, to facilitate learning. This could be an exciting avenue of research. However, evaluating the feasibility of this idea was not possible, given the scope and time frame of this thesis.

6.5 Ethical Considerations

Evaluating the potential use cases for the contents of this thesis from an ethical perspective highlights a few aspects. Firstly, accurately predicting future sales of intermittent data can have a positive effect, especially in domains where waste is an issue. Reducing waste would have a positive effect in terms of environmental and economic sustainable development, as reducing unnecessary waste also means less money being wasted on, in the end, unused products. A system like this could potentially also help businesses mitigate financial losses associated with excess inventory, spoilage, or obsolescence. However, it is also essential to highlight the potential misuse of a system such as the one presented in this thesis to instead maximize profit of the intermittent sales at the expense of increasing waste. A system like the one

presented in this thesis could be used to justify increasing prices compared to not using it, likely having a negative impact, increasing waste, and negatively affecting those with a lower socioeconomic status more dependent on discounted prices.

Secondly, using a black-box AI system is often viewed negatively by some, as it is hard to interpret how such a system reaches a decision, raising accountability questions. While the TFT model used in this thesis still has this issue to some degree, it also lends itself to more interpretability than other AI systems. This, in turn, allows for more accountability and discussion about whether or not a system like this is suitable for certain domains or not, presenting a more sustainable solution.

In summary, while there are potential ethical concerns associated with the implementation of this system, we believe the benefits of waste reduction and the potential for interpretability outweigh the drawbacks. The possible ability to more accurately predict future sales of intermittent perishable markdown products has the potential to address pressing environmental and economic issues, such as minimizing waste and optimizing resource utilization.

7

Conclusion

In conclusion, this thesis aimed to address the challenges of forecasting the sales of individual perishable markdown products using their time series. The target time series is characterized by being intermittent, sparse, and highly irregular, and sales can only occur if the marked-down product is available. Two separate problems were defined to predict the sales on a day-to-day basis, known as next-day predictions and predicting over a full markdown period, aggregating the total sales. To solve this problem, various methods were evaluated, ranging from well-established statistical models to newer deep learning-based models. The proposed solution was a novel method that improves the Temporal Fusion Transformer model with cluster encodings by applying random convolutional kernel transformations to time series.

Evaluating the performance, the deep learning models, the RNN, and Temporal Fusion Transformer outperform the compared statistical models. Especially, the TFT was better at next-day predictions, while the RNN was slightly better at markdown period predictions, albeit none of the results were statistically significant. In addition, the novel approach of clustering the markdown series based on markdown features showed no change in performance in terms of next-day prediction but did show a significant improvement in terms of the markdown period predictions. Moreover, using cluster encodings decreased the training time of the Temporal Fusion Transformer instances where it was included.

Thus, in conclusion, deep learning models and the Temporal Fusion Transformer with added cluster encodings, in particular, seem like a promising model for predicting intermittent series with known active periods. Future research should be conducted into alternative transformer-based time series models and comparing alternative clustering encoding methods and clustering algorithms on a variety of publicly available datasets.

Bibliography

[1] F. Raafat, "Survey of literature on continuously deteriorating inventory models," *Journal of the Operational Research Society*, vol. 42, no. 1, pp. 27–37, 1991. DOI: 10.1057/jors.1991.4.

[2] C. Bayliss, C. S. Currie, J. A. Bennell, and A. Martinez-Sykora, "Dynamic pricing for vehicle ferries: Using packing and simulation to optimize revenues," *European Journal of Operational Research*, vol. 273, no. 1, pp. 288–304, 2019, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2018.08.004.

[3] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–34, 2012.

[4] T.-C. Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

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

[6] G. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *European journal of operational research*, vol. 160, no. 2, pp. 501–514, 2005.

[7] C. Ingle, D. Bakliwal, J. Jain, P. Singh, P. Kale, and V. Chhajed, "Demand forecasting : Literature review on various methodologies," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 2021, pp. 1–7. DOI: 10.1109/ICCCNT51525.2021.9580139.

[8] A. Lasek, N. Cercone, and J. Saunders, "Restaurant sales and customer demand forecasting: Literature survey and categorization of methods," *Smart City 360°: First EAI International Summit, Smart City 360°, Bratislava, Slovakia and Toronto, Canada, October 13-16, 2015. Revised Selected Papers 1*, pp. 479–491, 2016.

[9] Y. Zhang, H. Zhu, Y. Wang, and T. Li, "Demand forecasting: From machine learning to ensemble learning," in *2022 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*, 2022, pp. 461–466. DOI: 10.1109/TOCS56154.2022.10015992.

[10] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and machine learning forecasting methods: Concerns and ways forward," *PLOS ONE*, vol. 13, no. 3, pp. 1–26, Mar. 2018. DOI: 10.1371/journal.pone.0194889.

[11] K. I. Park and M. Park, *Fundamentals of probability and stochastic processes with applications to communications*. Springer, 2018.

[12] Y.-W. Cheung and K. S. Lai, "Lag order and critical values of the augmented dickey–fuller test," *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 277–280, 1995.

[13] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969, ISSN: 00129682, 14680262. [Online]. Available: http://www.jstor.org/stable/1912791.

[14] K. Nemati-Amirkolaii, A. Baboli, M. Shahzad, and R. Tonadre, "Demand forecasting for irregular demands in business aircraft spare parts supply chains by using artificial intelligence (ai)," *IFAC-PapersOnLine*, vol. 50, pp. 15 221–15 226, Jul. 2017. DOI: 10.1016/j.ifacol.2017.08.2371.

[15] M. R. Amin-Naseri and B. R. Tabar, "Neural network approach to lumpy demand forecasting for spare parts in process industries," in *2008 International Conference on Computer and Communication Engineering*, IEEE, 2008, pp. 1378–1382.

[16] S. Mukhopadhyay, A. O. Solis, and R. S. Gutierrez, "The accuracy of non-traditional versus traditional methods of forecasting lumpy demand," *Journal of Forecasting*, vol. 31, no. 8, pp. 721–735, 2012.

[17] Q. Xu, N. Wang, and H. Shi, "Review of croston's method for intermittent demand forecasting," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE, 2012, pp. 1456–1460.

[18] N. Kourentzes, "Intermittent demand forecasts with neural networks," *International Journal of Production Economics*, vol. 143, no. 1, pp. 198–206, 2013.

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

[20] N. A. Heckert *et al.*, "Handbook 151: Nist/sematech e-handbook of statistical methods," 2002.

[21] R. H. Teunter and L. Duncan, "Forecasting intermittent demand: A comparative study," *Journal of the Operational Research Society*, vol. 60, no. 3, pp. 321–329, 2009. DOI: 10.1057/palgrave.jors.2602569.

[22] J. D. Croston, "Forecasting and stock control for intermittent demands," *Journal of the Operational Research Society*, vol. 23, no. 3, pp. 289–303, 1972. DOI: 10.1057/jors.1972.50.

[23] R. H. Teunter, A. A. Syntetos, and M. Zied Babai, "Intermittent demand: Linking forecasting to inventory obsolescence," *European Journal of Operational Research*, vol. 214, no. 3, pp. 606–615, 2011, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2011.05.018.

[24] B. Mehlig, *Machine Learning with Neural Networks*. Cambridge University Press, Oct. 2021. DOI: 10.1017/9781108860604.

[25] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning*, 2015. arXiv: 1506.00019 [cs.LG].

[26] S. Bai, J. Z. Kolter, and V. Koltun, *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling*, 2018. arXiv: 1803.01271 [cs.LG].

[27] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[28] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.

[29] *Creative commons attribution 4.0 international (cc by 4.0)*, version 4.0, Creative Commons, Apr. 22, 2023. [Online]. Available: https://creativecommons.org/licenses/by/4.0/.

[30] P. Savarese and D. Figueiredo, "Residual gates: A simple mechanism for improved network optimization," in *Proc. Int. Conf. Learn. Representations*, 2017.

[31] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[32] T. Warren Liao, "Clustering of time series data—a survey," *Pattern Recognition*, vol. 38, no. 11, pp. 1857–1874, 2005, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2005.01.025.

[33] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979, ISSN: 00359254, 14679876. [Online]. Available: http://www.jstor.org/stable/2346830.

[34] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, 2020, pp. 747–748. DOI: 10.1109/DSAA49011.2020.00096.

[35] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[36] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, Jul. 2020. DOI: 10.1007/s10618-020-00701-z.

[37] P. Wallström, "Evaluation of forecasting techniques and forecast errors: With focus on intermittent demand," Ph.D. dissertation, Luleå tekniska universitet, 2009.

[38] R. J. Hyndman, "Another look at forecast-accuracy metrics for intermittent demand," *Foresight: The International Journal of Applied Forecasting*, vol. 4, no. 4, pp. 43–46, 2006.

[39] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006, ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2006.03.001.

[40] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, vol. 57, 2010, pp. 10–25 080.

[41] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[42] J. Brownlee, *Why one-hot encode data in machine learning?* Jul. 28, 2017. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.

[43] P.-L. Bescond, *Cyclical features encoding, it's about time!* 2020. [Online]. Available: https://towardsdatascience.com/cyclical-features-encoding-its-about-time-ce23581845ca.

[44] B. Saji, *Elbow method for finding the optimal number of clusters in k-means*, Apr. 25, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/.

[45] A. Tomar, *Stop using elbow method in k-means clustering, instead, use this!* Nov. 17, 2022. [Online]. Available: https://towardsdatascience.com/elbow-method-is-not-sufficient-to-find-best-k-in-k-means-clustering-fc820da0631d.

[46] J. Herzen *et al.*, "Darts: User-friendly modern machine learning for time series," *Journal of Machine Learning Research*, vol. 23, no. 124, pp. 1–6, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1177.html.

[47] K. Erdem, *T-sne clearly explained*, Apr. 13, 2020. [Online]. Available: https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a.

[48] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1003–1012. DOI: 10.1109/CVPR.2017.113.

[49] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-beats: Neural basis expansion analysis for interpretable time series forecasting," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1ecqn4YwB.

[50] V. Vapnik and A. Vashist, "A new learning paradigm: Learning using privileged information," *Neural Networks*, vol. 22, no. 5, pp. 544–557, 2009, Advances in Neural Networks Research: IJCNN2009, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2009.06.042.

A

Product and environmental features

Table A.1: The exhaustive list of product features used.

Product features		
Feature name	Feature type	Description
Markdown units sold	Temporal	Daily number of marked down product items sold
Regular units sold	Temporal	Daily number of non-markdown product items sold
Markdown stock	Temporal	Number of product items on markdown if there is an active markdown
Days to expiration	Temporal	Number of days left of a markdown if there is an active markdown on the product
New units marked down	Temporal	New units added to a markdown. Always positive at the beginning of a markdown period; however, new units can also be added during a markdown
Units expired	Temporal	Units wasted at the end of a markdown due to that they did not sell
Value sold	Temporal	Daily value sold (in currency) of product
Active promotion	Temporal	Binary flag indicating if there is an active promotion on the product
Promotion quantity	Temporal	Number of products to buy for an active promotion to be valid
Promotion discount	Temporal	Discount for an active promotion
Price	Static	Regular price of product
Products' department	Static	A distinguishing category of the product
Products' group	Static	A distinguishing subcategory of the product
Products' assortment	Static	A distinguishing sub-subcategory of the product

Table A.2: The exhaustive list of environmental features.

Environmental features		
Feature name	Feature type	Description
Value sold in total	Temporal	Daily value (in currency) of the sales
Value sold of markdowns	Temporal	Daily value (in currency) of the markdown sales
Value sold of promotions	Temporal	Daily value (in currency) of the promotion sales
Value sold of regular sales	Temporal	Daily value (in currency) of the regular sales
Units sold on regular sale	Temporal	Daily number of units sold in the store
Units sold on markdown	Temporal	Daily number of markdown units sold
Units sold on promotion	Temporal	Daily number of promotion units sold
Units sold in total	Temporal	Daily number of units sold including promotion, markdown, and regular sales
Is holiday	Temporal	Binary flag for if a day is a holiday
Is working day	Temporal	Binary flag for if a day is a working day
Is pay-day	Temporal	Binary flag for if a day is a pay-day
Sun hours	Temporal	Time in hours between sunrise and sunset
Units wasted	Temporal	Daily number of units wasted in store
Value wasted	Temporal	Daily value (in currency) wasted in store
Store identifier	Static	Which store the environmental features regard

B

Clustering

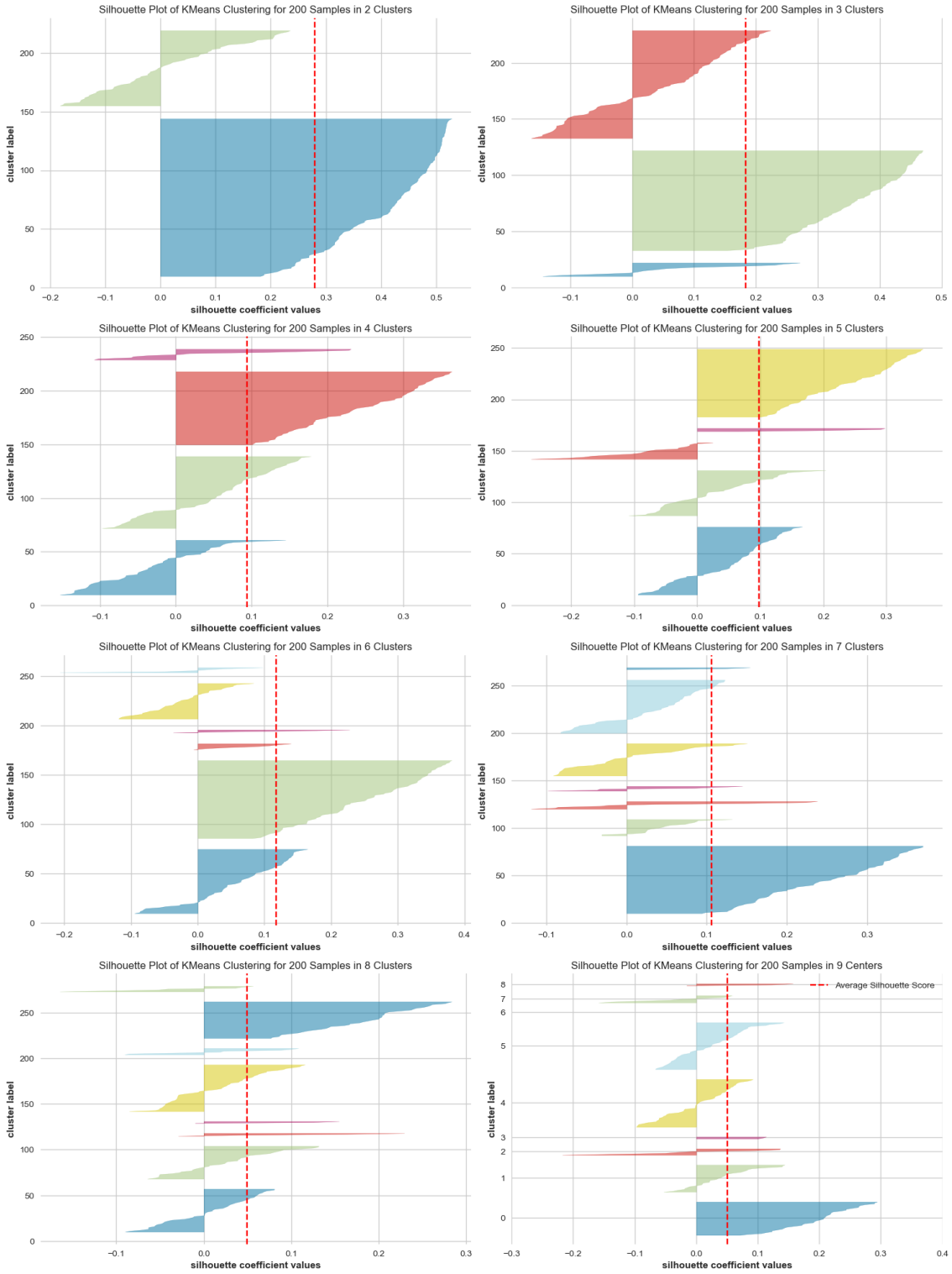


Figure B.1: Silhouette plots for the K-means algorithm for 2-9 clusters. Used for deciding on the optimal number of clusters to use. The ideal is when all clusters are as equal in size as possible and have as high a mean score.

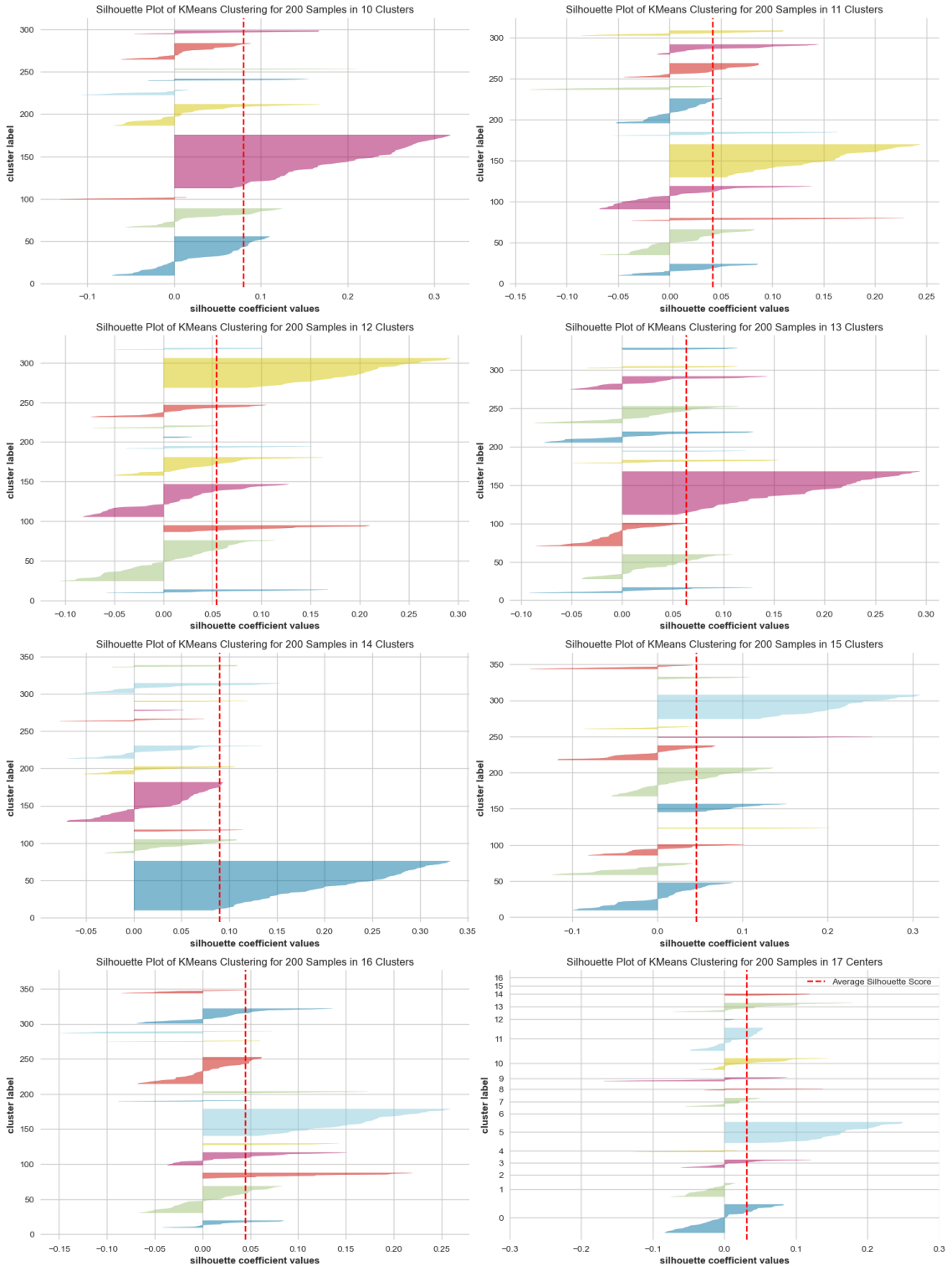


Figure B.2: Silhouette plots for the K-means algorithm for 10-17 clusters. Used for deciding on the optimal number of clusters to use. The ideal is when all clusters are as equal in size as possible and have as high a mean score.

C

Model Configurations

Table C.1: Configuration of the baseline RNN model.

RNN Configuration	
Parameter	Value
Input length	21
Recurrent layers	2
Feed-forward layers	1
Number of units (feed-forward layers)	128
Batch size	32
Dropout rate (feed-forward layers)	0.2
Gated recurrent unit type	LSTM
Learning rate	0.001

Table C.2: Configuration of the TFT model.

TFT Configuration	
Parameter	Value
Input length	21
Recurrent layers	2
Feed-forward layers	1
Number of units (feed-forward layers)	128
Number of attention heads/time step	4
Batch size	32
Dropout rate (feed-forward layers)	0.2
Gated recurrent unit type	GRU (Gated Residual Unit)
Learning rate	0.001

Table D.1: Exhaustive list of the TFT decoders's VSNs' weights.

Decoder Weights of VSN			
Future Cov.	Quantiles		
	0.1	0.5	0.9
Units Sold Markdown Data	0.8439	0.8766	0.8784
Day of week 5	0.0127	0.0144	0.0376
Stock Markdown	0.0073	0.0075	0.0076
Day of week 2	0.0042	0.0043	0.0045
Month 3	0.0042	0.0043	0.0044
Discount Promotion Data	0.0042	0.0043	0.0045
Sun Hours	0.0042	0.0042	0.0044
Month 8	0.0040	0.0040	0.0043
Month 5	0.0039	0.0040	0.0043
Month 11	0.0039	0.0040	0.0041
Month Cos	0.0039	0.0039	0.0041
Discount	0.0039	0.0039	0.0041
Day of week 1	0.0038	0.0039	0.0041
Month 4	0.0029	0.0039	0.0040
Day Cos	0.0038	0.0039	0.0041
Day of week 3	0.0038	0.0038	0.0041
⋮	⋮	⋮	⋮
Day of week 4	0.0037	0.0038	0.0040
Month 10	0.0037	0.0037	0.0040
Is Holiday	0.0036	0.0037	0.0039
Days To Expiration	0.0036	0.0037	0.0039
Is Payday	0.0034	0.0036	0.0037
Month 2	0.0035	0.0036	0.0038
Active Promotion	0.0035	0.0036	0.0038
Month 7	0.0034	0.0034	0.0037
Day	0.0032	0.0033	0.0035
Day of week 6	0.0026	0.0028	0.0048
Promotion Qty	0.0021	0.0025	0.0037
Month Sin	0.0020	0.0021	0.0028
Is Working Day	0.0018	0.0019	0.0023
Year	0.0016	0.0019	0.0023
Month 9	0.0017	0.0018	0.0020
Month 1	0.0014	0.0015	0.0017
Month 6	0.0013	0.0014	0.0024
Day Sin	0.0009	0.0011	0.0012
Month 12	0.0004	0.0004	0.0010

Table D.2: Exhaustive list of the TFT encoder's VSNs' weights.

Encoder Weights of VSN			
Past & Future Cov.	Quantiles		
	0.1	0.5	0.9
Units Expired	0.1038	0.1061	0.1079
Stock Markdown	0.0998	0.1013	0.1038
Value Sold Store	0.0965	0.0993	0.1022
Days To Expiration	0.0402	0.0404	0.0405
Value Sold	0.0340	0.0347	0.0353
Discount	0.0285	0.0288	0.0291
Units Sold Ordinary	0.0279	0.0281	0.0283
Value Sales Ordinary	0.0273	0.0275	0.0277
Value Sales Promotion	0.0265	0.0267	0.0269
Units Wasted	0.0236	0.0238	0.0239
Units Sold Markdown	0.0236	0.0237	0.0239
Sun Hours	0.0234	0.0235	0.0237
Month 1	0.0226	0.0228	0.0229
Month Cos	0.0224	0.0226	0.0227
Discount Promotion Data	0.0210	0.0212	0.0213
Day of week 1	0.0210	0.0211	0.0212
Is Holiday	0.0209	0.0210	0.0211
Month Sin	0.0209	0.0210	0.0211
New Units Marked Down	0.0208	0.0210	0.0211
Month 4	0.0203	0.0204	0.0205
Month 11	0.0197	0.0200	0.0202
Month 3	0.0198	0.0199	0.0200
Month 9	0.0197	0.0199	0.0200
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
Is Payday	0.0194	0.0195	0.0196
Day of week 4	0.0179	0.0180	0.0181
Day of week 5	0.0179	0.0180	0.0181
Day Sin	0.0177	0.0178	0.0179
Month 2	0.0163	0.0164	0.0164
Units Sold Sales Data	0.0142	0.0150	0.0163
Units Sold	0.0114	0.0115	0.0116
Active Promotion	0.0102	0.0102	0.0102
Month 6	0.0098	0.0099	0.0100
Day of week 2	0.0083	0.0084	0.0084
Month 10	0.0076	0.0078	0.0079
Day of week 3	0.0061	0.0061	0.0061
Units Sold Markdown Data	0.0058	0.0058	0.0059
Value Sales Markdown	0.0056	0.0057	0.0058
Value Wasted	0.0045	0.0046	0.0046
Month 8	0.0038	0.0038	0.0038
Year	0.0037	0.0037	0.0037
Is Working Day	0.0034	0.0035	0.0035
Sold Markdown Data	0.0031	0.0032	0.0032
Month 7	0.0031	0.0032	0.0032
Day Cos	0.0027	0.0027	0.0028
Month 12	0.0022	0.0022	0.0022
Promotion Qty	0.0021	0.0022	0.0022
Day of week 6	0.0021	0.0021	0.0021
Month 5	0.0017	0.0017	0.0017
Units Sold Promotion	0.0015	0.0015	0.0015
Day	0.0010	0.0011	0.0011

D

Temporal Fusion Transformer Interpretability

Table D.3: Attention Weights for Each Time step in the TFT.

Temporal Attention Weights			
Time Steps	Quantiles		
	0.1	0.5	0.9
t-21	0.00	0.00	$2.20 \cdot 10^{-5}$
t-20	0.00	0.00	$4.81 \cdot 10^{-4}$
t-19	0.00	$1.00 \cdot 10^{-8}$	$6.76 \cdot 10^{-4}$
t-18	$1.00 \cdot 10^{-8}$	$8.00 \cdot 10^{-8}$	$9.60 \cdot 10^{-5}$
t-17	$1.00 \cdot 10^{-8}$	$2.10 \cdot 10^{-7}$	$1.117 \cdot 10^{-3}$
t-16	$2.00 \cdot 10^{-8}$	$1.10 \cdot 10^{-7}$	$3.84 \cdot 10^{-4}$
t-15	$3.00 \cdot 10^{-8}$	$1.60 \cdot 10^{-7}$	$7.43 \cdot 10^{-4}$
t-14	$1.00 \cdot 10^{-8}$	$5.40 \cdot 10^{-7}$	$1.291 \cdot 10^{-3}$
t-13	0.00	$1.60 \cdot 10^{-7}$	$8.58 \cdot 10^{-4}$
t-12	$4.00 \cdot 10^{-8}$	$1.85 \cdot 10^{-6}$	$4.70 \cdot 10^{-5}$
t-11	$4.00 \cdot 10^{-8}$	$1.97 \cdot 10^{-5}$	$2.46 \cdot 10^{-3}$
t-10	$4.00 \cdot 10^{-8}$	$3.03 \cdot 10^{-4}$	$2.62 \cdot 10^{-3}$
t-9	$5.00 \cdot 10^{-8}$	$6.38 \cdot 10^{-5}$	$1.99 \cdot 10^{-3}$
t-8	$5.00 \cdot 10^{-8}$	$6.80 \cdot 10^{-7}$	$3.94 \cdot 10^{-3}$
t-7	$1.00 \cdot 10^{-8}$	$1.53 \cdot 10^{-6}$	$3.90 \cdot 10^{-3}$
t-6	$1.10 \cdot 10^{-7}$	$3.42 \cdot 10^{-5}$	$1.47 \cdot 10^{-3}$
t-5	0.00	$3.91 \cdot 10^{-5}$	$1.61 \cdot 10^{-3}$
t-4	$5.00 \cdot 10^{-8}$	$3.25 \cdot 10^{-4}$	$4.73 \cdot 10^{-3}$
t-3	$6.00 \cdot 10^{-8}$	$3.83 \cdot 10^{-6}$	$3.75 \cdot 10^{-3}$
t-2	$1.13 \cdot 10^{-6}$	$8.15 \cdot 10^{-5}$	$2.98 \cdot 10^{-3}$
t-1	$8.70 \cdot 10^{-7}$	$1.63 \cdot 10^{-4}$	$1.55 \cdot 10^{-3}$
t	$9.75 \cdot 10^{-1}$	$9.86 \cdot 10^{-1}$	$9.94 \cdot 10^{-1}$