Vrije Universiteit Amsterdam

IreckonU

Master Thesis

Automated Detection of Seasonality and Anomalies in IT Hospitality Infrastructure

Author: Evangelos Niklitsiotis (2778262)

1st supervisor:	P.J. de Andrade Serra	(VU)
daily supervisor:	Rik Van Leeuwen	(IreckonU)
2nd reader:	Sandjai Bhulai	(VU)

A thesis submitted in fulfillment of the requirements for the Master of Science degree in Business Analytics at Vrije University Amsterdam

December 1, 2024

Abstract

In today's fast-paced environment of the hospitality sector, reliable IT services are critical to keep customers satisfied and operations running smoothly. In this setting, anomaly detection has received much attention in recent years. In order to prevent software system failures, the key is to integrate tools that control and monitor the IT system's behavior. However, the industry's inherited seasonality and unpredictable demand changes make effective IT system management challenging. This study provides evidence of an advanced automatic detection of seasonal patterns and detection of anomalies in hospitality IT time series data, transitioning away from a reacting approach toward proactively avoiding them. Our framework leverages time series analysis and machine learning methods to identify repeating patterns and detect anomalous events in operational data extracted by advanced hospitality IT systems. The results validated the model's performance on synthetic time series data, proving that it can correctly identify seasonal patterns, and detect potential anomalies. Key findings highlight the need to consider seasonality as a starting point, which enhances anomaly detection recall and overall performance. The findings of this study have important implications for IT management in the hospitality domain, providing a scalable solution to increase service reliability and elevate the client experience. The framework paves the door for more complex automated IT management, allowing hospitality businesses to anticipate and respond to changing business demands with better precision and agility.

Acknowledgement

This report was written as a prerequisite for my Master of Business Analytics degree at Vrije Universiteit Amsterdam, and it ends an unforgettable internship at IreckonU. I am grateful for the assistance and opportunity offered by each institution, which allowed me to perform my research.

I would like to express deep appreciation to my business supervisor, Rik Van Leeuwen, for his invaluable help and industry insights, which have significantly improved my thesis and widened my practical understanding. My deepest gratitude also goes to my university supervisor, P.J. de Andrade Serra, for his knowledge and helpful input, which helped to shape my research. I am also grateful to my colleagues and friends for their technical help and encouragement; our talks and shared views were crucial.

To all who contributed to my academic and personal growth, I am deeply grateful. The project could not have been accomplished without your constant support.

Contents

Acknowledgement **i**

1 Introduction **1**

1.1 Anomalies and Seasonality 2

1.2 Research questions 4

1.3 Structure 6

2 Related Work **7**

3 Data Insights and Visualization **10**

3.1 Data Description 10

3.2 Data Preparation 12

3.2.1 Handling Missing Data 12

3.2.2 Data aggregation 13

3.3 Exploratory Data Analysis 14

3.4 Selected Dataset Insights 17

3.4.1 Descriptive Statistics 17

3.4.2 Distribution Analysis 18

4 Methodology **22**

4.1 Time series Modeling 22

4.2 Synthetic Data Generation 23

4.2.1 Noise Generation 24

4.2.2 Seasonality and Trend Generation 25

4.2.3 Anomalies Generation 26

4.2.4 Time Series Composition 27

4.3 Seasonality Detection 27

4.3.1 ACF: Baseline Seasonality Detection 27

4.3.2 Seasonal Quantile Regression: Advanced method 29
4.3.2.1 Seasonal Quantile Regression Design 29
4.4 Anomaly Detection 32
4.4.1 AD-PCI: Baseline Anomaly detection 33
4.4.2 Neural Prophet: Advanced Anomaly Detection 34
4.4.2.1 Neural Prophet Design 35

5 Experimental Setup **37**
5.1 Evaluation Metrics 37
5.2 Time series splitting 38
5.3 Hyperparameter Tuning 38
5.3.1 Autocorrelation Function (ACF) model 39
5.3.2 Seasonal Quantile Regression (SQR) Model 39
5.3.3 Anomaly Detection Prediction Confidence Interval (AD-PCI) Model 39
5.3.4 Neural Prophet Model 39

6 Results **41**
6.1 Data Similarity Assessment 41
6.1.1 Permutation tests 42
6.1.1.1 Average Length of Time Between Consecutive Anomalies . 42
6.1.1.2 Difference between Non-Anomalous Points 43
6.1.1.3 Approximate Entropy 44
6.2 Seasonality Detection Results 45
6.2.1 Results on Synthetic Data 45
6.2.2 Results on Real Data 48
6.2.3 Cross-validation Setup 49
6.3 Anomaly Detection Results 50
6.3.1 Results on Synthetic Data 50
6.3.2 Results on Real Data 51
6.4 Computational Efficiency Analysis 54

7 Conclusion **55**

8 Limitations and Future Work **57**

References **59**

List of Figures **67**

List of Tables **69**

1

Introduction

There has never been such an information explosion as witnessed in the current digital age. Data generation has increased dramatically as a result of the development of sensors, mobile phones, and various cloud-based systems, including those used in cars and home appliances. Several things we utilize on a daily basis, such as the smartphones we carry around in our pockets, are continuously generating data. However, the true driving factor behind this boom isn't solely how easy it is to acquire these digital assets—rather, it's the increasing importance of data analysis and the automated processes these analytics create. Such technological advancements have driven businesses' reliance on advanced IT systems (1), centering their operations on collecting, processing, and leveraging data to drive decision-making.

The hospitality industry, in particular, has actively embraced Information Technology (IT) systems to streamline operations. This sector heavily relies on technologies such as Property Management Systems (PMS),(2) (3), which manage everything from reservations to customer data and communication between various systems. Guests visit hotels for a variety of reasons; therefore, hotels gather data to help them make data-driven decisions. The growing number of guest data, alongside hotels operating around the clock, hampers decision-making and poses a challenge for hotel chains seeking to improve customer satisfaction while optimizing their processes. Thus, this infrastructure requires continuous monitoring using the generated data, particularly when it comes to time series data. Due to the middleware solution, businesses can monitor and analyze the frequency of API calls to a specific system (i.e., a payment system) over a defined period. The massive volumes of data make it impossible to extract meaningful information without the help of automated solutions and may even result in the risk of hampering monitoring tasks as data processing and analysis take too long.

In this setting, both seasonality and anomaly detection emerge as tools to monitor these systems' behavior and keep track of irregularities over time. Businesses may apply mathematical algorithms to evaluate patterns in data at group and/or individual levels to identify deviations from the expected behavior (4). This process is particularly useful for hotels where system failures, such as check-in and out issues, can negatively affect the experience of visitors. Identifying and resolving issues promptly improves operational efficiency and prevents further decreases in client satisfaction. Effective decision-making and quick resolving issues are essential for building long-lasting connections with visitors.

Undoubtedly, both seasonality and anomaly detection have lately received substantial research attention, yielding insights with a wide range of possible applications. Besides hospitality, seasonality detection is beneficial in a range of areas, including supply chain, as it helps organizations optimize inventory levels and minimize costs (5). Similarly, anomaly detection is beneficial in a variety of scenarios. For example, in Financial Services, it assists in monitoring spending trends for fraud detection (6), in Healthcare, it tracks patient information and alerts professionals to potential health emergencies (7), and in Network Security (8), it detects any suspicious network activity that could lead to a cyber-attack.

1.1 Anomalies and Seasonality

Anomaly detection is not an innovative concept. Its research goes back to the 18th century, when, in 1777, Bernoulli commented on the common method of neglecting outlier data when no prior knowledge was available (9). The important point is to deliver a formal definition of the concept of anomaly. This is essential because different definitions of anomalies imply varying methods to detect them. The most common definition of anomalies is the following:

> *"Anomalies are patterns in data that do not conform to a well-defined notion of normal behavior." – Chandola et al. (10)*

In our context, we can describe the anomaly in time series data as the data point(s) (or observations) at time step(s) that differ significantly from previous time steps. Following that, we classify the types of anomalies related to time series data as follows: `point anomalies, contextual anomalies, and collective anomalies` (11).

A `point anomaly` is a data point or a sequence that significantly deviates from the norm. Such anomalies may appear to be temporal noise and are often caused by sensor errors or abnormal system operations. For detection, operators traditionally set upper

and lower control limits, commonly referred to as UCL and LCL, respectively, based on historical data. Values that exist outside those limits are regarded as point anomalies. A sequence of data points may be classified as anomalous even if individual points are not. For example, a low number of web accesses might be normal during night hours but abnormal during the day. Such anomalies are called `Collective anomalies`. The third type is called `contextual anomalies,` where some points can be normal in a certain context while detected as an anomaly in another context. This study predominantly focuses on point and collective anomalies because they occur frequently in the given dataset and have a direct influence on operational efficiency. Figure 1.1 below illustrates three types of anomalies commonly observed in time series data.

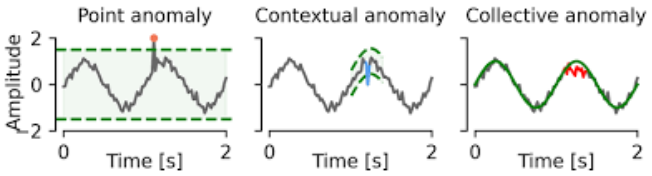

Figure 1.1: Types of Anomalies

Even though numerous businesses are heavily investing in data collection (12) and analysis to identify anomalous patterns in order to serve their customers better, the majority of the data is in the form of streaming time series, which are characterized by seasonality, trend, and noise, making accurate anomaly detection a challenging task.

Seasonality is defined as a periodic fluctuation in data values that occurs at regular and predictable times of the year, known as seasons. Seasonality in the hospitality sector is an important element that occurs through the days, weeks, and months. Poor understanding of seasonal trends would have an impact on hotel operations, particularly if increased demand could not be predicted during peak seasons such as summer. This would result in lacking customer service, lengthy lineups, and poor client satisfaction. Unlike longer-term trends, which can last for a longer length of time, seasonal patterns often recur at short specified intervals, such as days, weeks, months, or quarters. The hospitality industry is an ideal representation of seasonality, involving daily customer check-ins and check-outs. A hotel enables customers to check in and out at particular times during the day. Customers, for example, can check out between 11:00 and 12:00 and check in between 16:00 and 17:00. This is not an arbitrary pattern, but rather a pattern based on the operational hours of each hotel, which determine peak hours. This consistent timing might be considered a seasonal pattern because it repeats regularly and becomes highly predictable.

This restriction provides a significant challenge for hotel chains since changes in client behavior typically result in changes in the data. Anomaly detection evolves into more than just a necessity; it is a critical component of excellence in operations, enhancing client satisfaction. A wide range of strategies for anomaly detection have lately been studied, ranging from classic statistical methods (13), (14), to sophisticated machine learning algorithms (15). However, many people overlook the importance of seasonal trends, which are especially noticeable in the hospitality domain. There is still a gap in effective anomaly detection in the extremely dynamic and seasonal environment of hotel enterprises. Thus, this research gap emphasizes the importance of building a robust anomaly detection system capable of reliably identifying seasonal time series while maintaining sensitivity to real anomalies.

1.2 Research questions

IreckonU's current anomaly detection module uses the `Moving Average` (MA) method as a time series forecasting model. While this model works well and provides accurate forecasts for the time series data at a constant level, it does not account for seasonal patterns. Our primary goal in this study is to identify seasonal patterns in the data and implement the most appropriate forecasting technique, focusing on maximizing the detection of true anomalies and ensuring applicability in the hospitality domain. Therefore, the following is our primary study question:

RQ1: *How does accounting for seasonality improve the recall and performance of identifying anomalies in time series data?*

Three sub-questions arise from this research question, which are formulated as follows:

SQ1: *What are the most effective statistical methods and machine learning algorithms for identifying seasonal patterns and anomalies in time series data?*

SQ2: *How to handle anomaly detection in seasonal time series?*

SQ3: *How can the identified statistical methods or machine learning algorithms for seasonality and anomaly detection be implemented in practice, considering computational efficiency, scalability, and interpretability?*

Developing an effective seasonality and anomaly detection system in today's data-driven world faces many key challenges. As data volumes grow dramatically, scalability presents as one of the main issues, making traditional methods like distance-based approaches

inefficient and slow. Managing these databases is essential, or we run the risk of missing important information because of insufficient computational capacity (5). A second, related difficulty brought about by the acceleration of data generation is to obtain adequately labeled datasets. This emphasizes the importance of techniques that produce satisfactory results even in the absence of labeled data i.e., in an unsupervised manner. Third, systems must be both accurate and fast, demanding computing efficiency to immediately respond to incoming data.

These issues are especially apparent in the hospitality industry's IT infrastructure, which generates massive amounts of guest data, particularly during peak seasons. The vast range of generated data, from customer reservations to transactions, makes it challenging to maintain accurate and up-to-date labeled datasets for model training. Furthermore, the creation of a fast and reliable system is critical since the hotel industry operates around the clock, and the ability to detect and handle anomalies promptly has a direct influence on overall business efficiency. In addition, a white-box approach is crucial since it provides a simple and straightforward framework for non-technical IT domain experts, resulting in quicker acceptance.

In light of the need for an anomaly detection module that accounts for seasonality, this research paper wants to develop a sophisticated method that is characterized by its ability to function effectively within the dynamic context of hospitality technology, addressing the current limitations and challenges while aligning with the following main characteristics:

1. Is unsupervised
2. Accounts for seasonal patterns
3. Scales to process large amounts of data
4. Easy explainable (White box)
5. Incorporate Confidence Prediction intervals (CPI)

This thesis project makes significant contributions by addressing a crucial gap in anomaly detection for seasonal time series data within the hospitality industry. Specifically, it introduces and implements the **SQR-AD** algorithm, a seasonality-aware anomaly detection framework designed to address the challenges of unsupervised time series data. The main contribution of this research is an extension of the published research (16) to include a mechanism for seasonality detection into anomaly detection for IreckonU, setting a new standard for future developments in hospitality IT infrastructure. Through its methodology and empirical validation, this research demonstrates the critical importance of adapting to seasonal patterns in dynamic data environments, making a substantial technical and practical advancement in the field.

1.3 Structure

The structure of this report is as follows. First, an overview of previous research studies is given in Chapter 2. It will include details about anomaly detection methods and how they were used in prior research. Secondly, Chapter 3 offers a detailed overview of the dataset along with pre-processing and key visualizations. Thirdly, Chapter 4 delves into the foundational theory of time series and the algorithms employed, establishing the framework for this research. After that, Chapter 5 discusses the experimental design, including performance metrics and hyperparameter tuning processes. A summary of the findings from the tasks of seasonality, anomaly detection, and data similarity assessment is given in Chapter 6. Finally, Chapter 7 outlines the study's limitations and provides suggestions for future research.

2

Related Work

This section examines current anomaly detection approaches, with an emphasis on time series forecasting techniques and cutting-edge models. It examines the evolution of these methods, their practical uses, and inadequacies, highlighting the need for a new strategy capable of effectively addressing the unique issues related to time series data, particularly in scenarios with seasonal patterns.

A.Evolution of Anomaly Detection Techniques

The field of anomaly detection has advanced significantly from simple statistical approaches to complicated machine learning and deep learning approaches. Several such methods are statistically rooted in the seminal publication of Hawkins' study on outlier detection (17). However, these traditional approaches frequently presume static data distributions, rendering them unsuitable for dealing with the seasonal trends in time series data. The area of anomaly detection is wide and includes a variety of classification frameworks intended to handle its different approaches and scenarios. A commonly accepted categorization, as explained in (18), is based on the level of supervision needed by the method: `supervised`, versus `unsupervised`.

Labeled data is not required for unsupervised learning techniques like clustering algorithms (e.g., K-means, DBSCAN). These methods identify anomalies by identifying data points that are distant from clusters or occur exclusively in sparse areas of the dataset (19). While these strategies may be useful, they neglect seasonal variations in time series data, resulting in incorrect classification. Instead, supervised machine learning approaches use labeled datasets to train models and detect irregularities using classification models such as Support Vector Machines (SVM), decision trees, and artificial neural networks (20).

Such approaches produce higher accuracy results, but they rely largely on the existence of labeled data, which can be costly and time-consuming to obtain.

Recent improvements in transformer-based models have resulted in powerful methods for detecting anomalies in time series data. For example, the Anomaly Transformer (21) employs attention mechanisms to successfully capture both long-term and short-term dependencies in temporal data, resulting in cutting-edge performance across many benchmarks. However, these developments frequently need significant computational resources, which might limit their applicability in situations requiring low latency or operating within resource limits, such as hospitality IT systems.

B.Prediction-Based Techniques

Prediction-based anomaly detection is especially useful for time series data since it involves predicting future values and identifying anomalies based on deviations from these predictions. These approaches are ideal for tracking system performance because they can adapt to shifting patterns. There are two related studies found, both detecting anomalies using time series forecasting.

Shirani et al. (22) used the ARIMA (Autoregressive Integrated Moving Average) model to build a predictive model on web service data and estimate the next time step. A Prediction Confidence Interval of 95% was used to compare the prediction errors, which correspond to the differences between the predicted and actual values. In other words, if the estimated prediction error exceeded the PCI threshold value, that data point was marked as an anomaly. This approach was extremely accurate, with a 97.3% positive rate and a false positive rate of 0.0154. However, ARIMA's power, specifically its inability to manage overlapping seasonal patterns, makes it less useful in highly seasonal sectors such as hospitality.

A similar study by He and Zhao et al. (23) established a Temporal Convolutional Net (TCN)-based model that very successfully represents time series data using convolution neural network architecture. This method outperforms conventional methods in terms of accuracy and efficiency. The TCN model enhanced precision by detecting anomalies using reconstruction errors, attaining scores of 0.946, 0.880, and 0.800 for beta values of 0.1, 0.05, and 0.05, respectively. Regardless of its capabilities, TCN's reliance on precisely calibrated thresholds limits its flexibility in contexts with highly variable seasonal components.

C.Advanced Deep Learning Techniques

Deep learning techniques can enhance anomaly detection by addressing non-linear interactions and complex data patterns. Munir et al. (24) developed DeepAnT, an unsupervised deep learning method for identifying anomalies in time-series data, which may be used in non-streaming scenarios. DeepAnT's architecture consists of two modules: a time series predictor and an anomaly detector. The time series predictor module ought to predict the next timestamp on the specified horizon (used as a context) by using the deep convolutional neural network (CNN). This predicted value is then passed into an anomaly detection module, which classifies the relevant timestamp as normal or abnormal. DeepAnT is a deep learning model that detects point and contextual anomalies in time series data with periodic and seasonal patterns.

Recent advances in anomaly detection include hybrid models, which integrate neural networks with statistical approaches to improve accuracy and flexibility. In her paper, Farzaneh Khoshnevisa (25) employs the Robust Seasonal Multivariate Generative Adversarial Network (RSM-GAN) to address noise and seasonality challenges associated with high-dimensional data. This model extends GANs with convolutional-LSTM layers and attention mechanisms, capturing temporal and spatial relationships while dealing with data contamination. RSM-GAN leverages adversarial learning to accurately capture temporal and spatial dependencies in the data, while simultaneously training an additional encoder to handle training data contamination. The attention mechanism in the recurrent layers of RSM-GAN enables the model to adjust complex seasonal patterns observed in the data. The comparison with other existing classical and deep-learning AD models shows that this architecture is associated with the lowest false positive rate and improves precision by 30% and 16% in real-world and synthetic data, respectively.

D. Gaps and Opportunities

Despite recent significant advances in anomaly detection, fundamental obstacles remain. Seasonality is frequently overlooked within existing strategies, particularly in unsupervised environments, and computational efficiency is often inadequately addressed. This constraint is especially relevant in businesses like hospitality, where anomalies fluctuate dramatically as time period changes. To overcome these limitations, we want to develop a robust seasonality-aware anomaly detection model in an unsupervised setting, ensuring better flexibility and reliability.

3

Data Insights and Visualization

Having an extensive knowledge of the dataset of interest is the first, and the most crucial step towards developing a model that is meant to act. Every dataset is different, and figuring out which model works best depends a lot on understanding what makes each dataset unique. The following chapter provides a full description of the dataset, including its structure and data preparation, as well as emphasizing significant findings from exploratory analysis. This effort sets the framework for the research's later phases.

3.1 Data Description

This dataset, provided by IreckonU, is a collection of API call logs all derived from a cloud environment. An API call is a request to deliver or receive data from one software program to another software program. In our dataset, there are a total of 175,809 rows and 8 columns representing various actions performed across various systems. The details of each dataset's column are as follows:

1. **Category:** Every entry is classified under the 'Cloud' category, denoting that the interactions are based on cloud technology.

2. **System:** This column identifies the specific system within the IT infrastructure targeted by the API call. The dataset includes seven distinct systems, that could represent property management systems (PMS), customer relationship management (CRM), and kiosk among others.

3. **Action:** Specifies the type of API request or action performed. There are 23 unique actions listed, including 'Check-in', 'Check-out', 'Create Reservation', 'Update Profile', and 'Check Availability', which are vital for regular hospitality operations.

4. **TimeStamp:** Records the date and time when the API call was executed.
5. **Timespan:** Indicates the duration for batch API calls, set at '00:05:00'.
6. **AvgDuration:** Measures the average time taken for the API Calls documented in each entry, expressed in milliseconds.
7. **SuccessCount:** Quantifies the number of successful API calls.
8. **ErrorCount:** Measures the amount of failed API calls, which might provide useful information about potential issues with the back-end system.

The time series data spans from `2023-02-11 17:00:00` until `2023-04-12 20:30:00` with a 5-minute time interval. Three columns are of integer type, reflecting numerical values, the *AvgDuration*, *SuccessCount*, *ErrorCount*. Five columns are object types including *Category*, *System*, *Action*, *TimeStamp*, *Timespan*. The dataset captures interactions among multiple systems and their accompanying actions, including timestamps, average durations, success, and error counts.

For instance, on timestamp `2023-02-11 17:00:00`, three distinct records highlight activities involving two systems : *System1* and *System2* and three different actions: *Action5*, *Action6*, and *Action8*. Specifically, *System1* executed two different actions: *Action5* and *Action6* at the specified timestamp. *Action5* had an average duration of 33 milliseconds, yielding two successful calls with no mistakes. In contrast, *Action6*, also on *System1*, completed more quickly in just 13 milliseconds, but achieved a higher success count of 5, indicating perhaps a more efficient or simpler task, also without errors. On the other hand, *System2* engaged in *Action8* at the same time but took considerably longer, 689 milliseconds, to complete with only 2 successes, suggesting a more complex or resource-intensive task. The illustration 3.1 below shows how actions within the same or distinct systems can vary significantly in duration and efficiency, providing insight into operational dynamics.

System	Action	TimeStamp	Timespan	AvgDuration	SuccessCount	ErrorCount
System1	Action5	2023-02-11T17:00:00	0:05:00	33	2	0
System1	Action6	2023-02-11T17:00:00	0:05:00	13	5	0
System2	Action8	2023-02-11T17:00:00	0:05:00	689	2	0

Figure 3.1: Dataset snapshot

3.2 Data Preparation

A time series analysis requires careful and consistent data preparation. Data integrity and continuity should be our primary goal, ensuring that missing timestamps are detected and proper imputation techniques are provided. Missing values in our dataset correspond to the cases where no actions were performed. In order to do that, we start by determining the earliest timestamp: `February 11, 2023, 17:00:00` and the latest timestamps: `April 12, 2023, 20:30:00` to establish the whole time period that our dataset should span. We utilize the pandas DataFrame library to determine the total 5-minute periods in this timeframe. Since events are captured every 5 minutes, we anticipate 17,323 unique timestamps.

Furthermore, it is critical to establish the number of time series related to each variable. For those mentioned above, we identify every potential pairing of System and Action in our dataset, each representing a distinct time series. There are a set of 23 possible combinations for each of the three numerical variables. Examples of such combinations include *(System1, Action1)*, *(System2, Action8)*, and *(System5, Action17)* among others. Notably, *System3* does not account for any action. Table 8.1 thoroughly analyzes each pair's present and missing timestamps. In this table, the 'MissingTimestamps' column indicates the difference between the expected and present timestamps, highlighting the number of timestamps that were not recorded for each pair.

3.2.1 Handling Missing Data

Missing data must be handled cautiously in time series analysis since it could threaten the dataset's integrity and introduce biases into the models. In this study, when a timestamp is missing, the *AvgDuration* variable is assigned to the NaN value, indicating that no activity was recorded at the specified timestamp. We use the NaN value to guarantee that the absence of data cannot be interpreted as a zero-duration event. Assigning a zero value in place of NaNs may provide the misleading impression that an API request was performed and finished immediately, introducing significant bias into the study. The dataset lacks regular patterns for NaN records, making it difficult to assess their relevance or causes across time.

In contrast, variables such as *SuccessCount* and *ErrorCount* are imputed with zero values, indicating that no events, either success or error, occurred during the missing time period. The *Timespan* column is uniformly set to '00:05:00' for consistency across all timestamps, representing the intended recording interval. This method is executed for

all unique System-Action combinations, resulting in a rebuilt dataset with no gaps in the timestamp records.

3.2.2 Data aggregation

After imputing the missing values of our dataset the pre-processing continues with a data aggregation process. Data aggregation is an important step in time-series studies that aims to simplify data patterns, reveal long-term trends, and reduce the noise created by short-term variations. This study reformatted an initial 5-minute dataset into three broader intervals: `30 minutes`, `1 hour`, and `1 day`. This transformation involved reformatting the *TimeStamp* column to group consecutive 5-minute entries into unified time frames (e.g., 30-min), ensuring a coherent and structured representation of temporal activity. Each record had a System and Action ID (i.e. *System1-Action5*) that was linked to it and remained with it during the merging process. We were able to represent every possible combination of System and Action within each aggregated time period by taking the IDs and using the *TimeStamp* (i.e. *12/04/2023 1:30:00 PM*) column to match them with the data. Consequently, each System-Action pair was treated as a distinct time series, enabling a detailed exploration of trends and patterns across multiple aggregation levels without altering the dataset's foundational structure.

The initial approach was to sum the successful and unsuccessful calls of each timestamp into a single *TotalCalls* variable, which essentially encompasses the overall activity level in a given time period (1 hour, etc.). The two initial variables recording the successful (*SuccessCount*) and failed (*ErrorCount*) API calls, were renamed to *TotalSuccessCount* and *TotalErrorCount* to highlight that these variables are aggregated after processing. These renamed metrics now provide cumulative totals for successful and unsuccessful calls within each granularity level.

The procedure also produces the *WeightedAvgDuration* column, which provides an accurate and relevant picture of the third variable, *AvgDuration*. This is accomplished by a weighted average technique, with weights determined by *TotalCalls* variable—the sum of successful and error counts for each group. Using this method, times of higher activity, which suggest greater volumes of data and more dependability, have a stronger effect on the final value. For each group, the non-missing (i.e., non-NaN) *AvgDuration* values are multiplied by their respective *TotalCalls*, and the resulting products are summed together. This amount is then divided by the total number of calls in that group, assuming that the sum of *TotalCalls* is not zero. When no requests for the API occur during the time

frame specified, the value is set to NaN. This strategy avoids distortions that would arise if all durations were weighted identically, ensuring that the aggregated number accurately reflects both the degree of activity and the relative importance of each time period. Figure 8.7 shows how individual durations are aggregated into a single weighted average for the specified time period. Mathematically, the method is articulated as:

$$\text{WeightedAvgDuration} = \frac{\sum_{i=1}^{n}(\text{AvgDuration}_i \times \text{TotalCalls}_i)}{\sum_{i=1}^{n} \text{TotalCalls}_i} \tag{3.1}$$

where:

- n is the number of time periods or entries considered.
- AvgDuration_i is the average duration for the i-th time period or entry.
- TotalCalls_i is the total number of calls for the i-th time period or entry.

Several important considerations influenced the choice to aggregate the data. The initial 5-minute dataset was extremely detailed, recording even little fluctuations that introduced significant noise into the data. While this granular data may be useful for specific analysis, it tends to add significant noise, as the short-term variations obscure the systemic tendencies to which we are ultimately trying to respond. Moreover, the enormous amount of data generated was computationally expensive and introduced complexity in modeling and analytic processes. By aggregating the data over larger time durations, the noise is reduced, more meaningful trends emerge, and fewer changes within a short term make it easier to control and analyze. Given that most time series of the *TotalErrorCount* recorded zero errors, we focused on *TotalSuccessCount* and *WeightedAvgDuration* at 30-minute, 1-hour, and 1-day levels to identify the best degree of aggregation. This method allowed us to establish a balance between clarity and detail, enhancing our ability to recognize patterns and anomalies.

3.3 Exploratory Data Analysis

In this phase, the exploratory data analysis (or EDA) will highlight the important aspects of our various aggregated datasets. Our main objective is also to select the best dataset for in-depth analysis. This process provides a strong base for our future modeling endeavors. To analyze the impact of different aggregation levels, we visualized the key variables: *TotalSuccessCount* and *WeightedAvgDuration* for *System6-Action22* using line graphs. These visualizations were generated for the initial 5-minute detailed dataset and

each of the three aggregated datasets: `30 minutes`, `1 hour`, and `1 day`. The line graphs for *TotalSuccessCount* (Figure 3.2) and *WeightedAvgDuration* (Figure 3.3) revealed the effect of varying aggregation granularities on the representation of patterns and deviations.

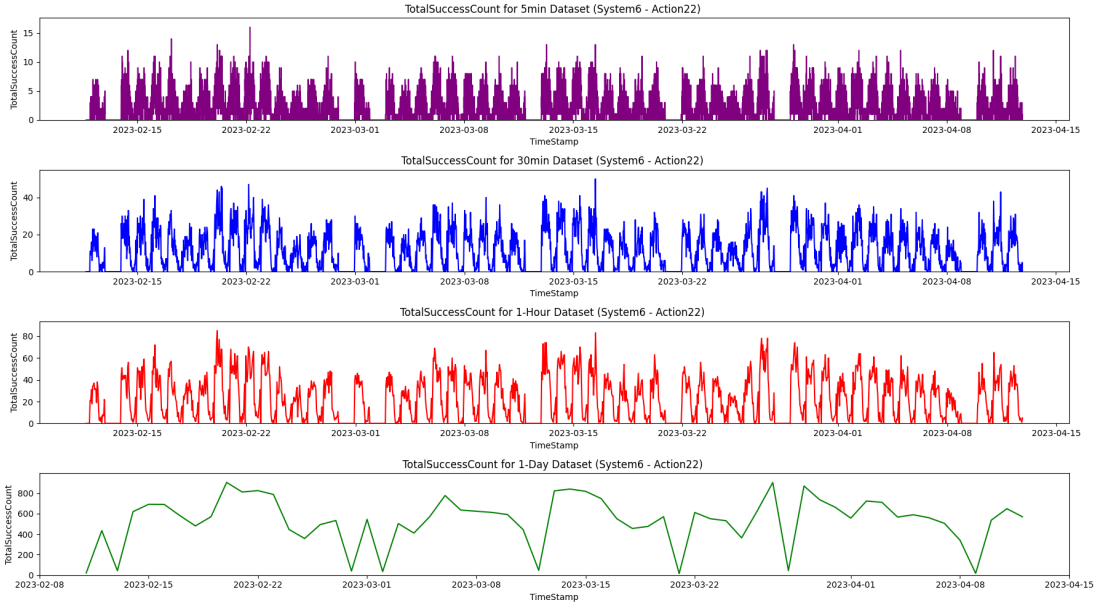

Figure 3.2: Line Graphs of TotalSuccessCount for 5-min, 30min, 1-Hour, and 1-Day Datasets (System6-Action22)

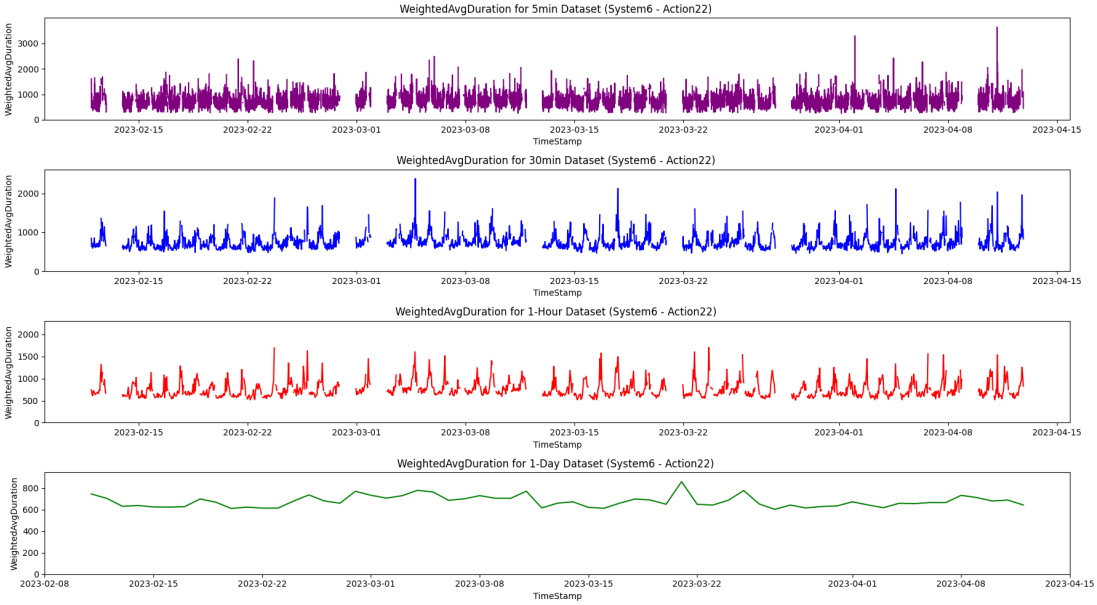

Figure 3.3: Line Graphs of WeightedAvgDuration for 5-min, 30-min, 1-Hour, and 1-Day Datasets (System6-Action22)

The 30-minute and 1-hour datasets demonstrate variability during normal business operations, with comparable behavior across time and both upward and downward trends. While the 30-minute dataset is more detailed, it also contains more noise, making it more difficult to detect important seasonal changes. Having additional data points complicates the analysis, making it difficult to decide which features are worth investigating. On the other side, the 1-day dataset provides a broader view showing major patterns, but it masks finer details and variability.

The boxplots in Figure 3.4 show how *TotalSuccessCount* and *WeightedAvgDuration* are distributed among different systems, using a logarithmic scale used to emphasize variations in each aggregated dataset. The 1-hour dataset serves as a middle ground, striking a balance between the detailed, noisy 30-minute dataset and the too-generic 1-day dataset. It reduces noise and minor fluctuations observed at 30-minute intervals while preserving enough granularity to identify major operational trends and anomalies. In addition, utilizing a 1-hour aggregation reduces sensitivity to short-term deviations that may arise during shorter time periods. That is very useful for focusing on long-term trends rather than reacting to possible misleading short-term fluctuations. A sudden surge in call volume in 30 minutes may not look as substantial as it would in an hour, for example.

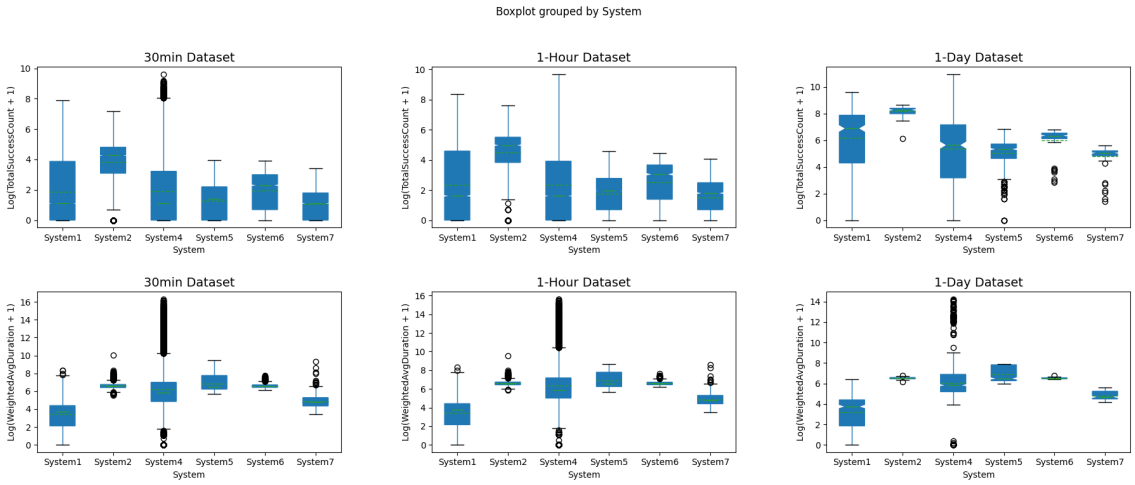

Figure 3.4: Boxplots of TotalSuccessCount and WeightedAvgDuration Across Systems for 30min, 1-Hour, and 1-Day Datasets on log Scale

In a nutshell, selecting the 1-hour dataset for subsequent analysis was a prudent choice. With fewer data points to evaluate, it minimizes excessive noise and computational complexity yet offers sufficient granularity to reveal crucial operational insights and trends.

This approach offers an extensive and insightful investigation, providing a solid basis for understanding and forecasting the behavior of the system under study.

3.4 Selected Dataset Insights

After concluding the dataset preparation and carrying out exploratory data analysis, the focus shifted to our selected 1 hour dataset for further analysis, which consisted of 1,444 timestamps for each pair. This dataset was analyzed using plots and descriptive statistics.

3.4.1 Descriptive Statistics

The dataset reveals significant differences in System-Action efficiency. The pair (or time series) with the greatest overall call volume is *System4 - Action11*, which recorded 2,072,001 calls, highlighting the importance of operations in IT hospitality systems. *System4-Action9*, on the other hand, got the lowest number of API calls (980), indicating that it is rarely utilized. Execution times also exhibited significant differences. *System4-Action10* had the longest average duration of 380,677.54 milliseconds, most likely owing to its participation in complicated operations including sophisticated booking management with significant customer service demand. *System1-Action3* had the shortest average execution time of only 0.35 milliseconds, indicating that it is associated with simpler processes such as regular check-outs.

Regarding the *TotalSuccessCount*, *System4-Action11* had the highest average success rate (1,434.89) and the most successful tasks (16,113). This demonstrates its capacity to handle large task quantities efficiently. In contrast, *System1-Action4* and *System4-Action14* had the lowest success rates, with average success counts of only 0.75, indicating occasional successful task results. Errors in the *TotalErrorCount* were relatively infrequent, with an average error count of zero across most combinations. The maximum known error count for any pairing was 249. However, this level was rarely achieved, indicating the systems' general reliability. The bar chart 3.5 provides a visual representation of the number of successful API Calls across different systems. Each bar corresponds to a system, and the length of the bar indicates the total number of successful outcomes recorded by that system.

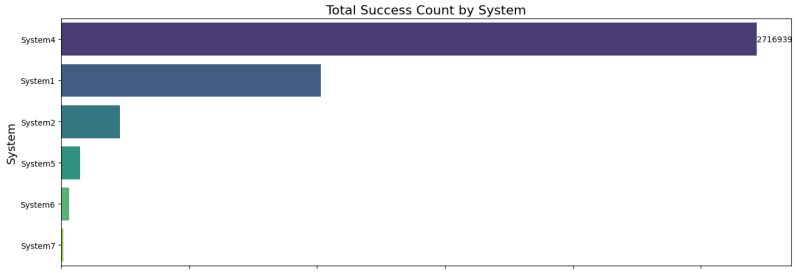

Figure 3.5: Total Success Count box plots by System

3.4.2 Distribution Analysis

Understanding the data distribution of our selected dataset (1 hour) is a critical step toward understanding the behavior of the variables under study and creating an accurate simulation model. Plotting histograms of each variable for all time series as illustrated in Figure 3.6 for the *TotalSuccessCount* variable and in our Appendix (Figure 8.1 and Figure 8.2) for the rest variables reveals some noteworthy patterns. The histograms are predominantly right-skewed, with many low activity counts and few high counts, showing that the majority of combinations do not follow a normal distribution. The distribution varies between System-Action pairings, with some showing large variations in action counts and others being more stable. Furthermore, many combinations have high zero counts, indicating no successful actions, which is frequent in event-driven systems.

In order to ensure the creation of an accurate simulation model, we used the Kolmogorov-Smirnov (KS) test to analyze and match the dataset to an acceptable probability distribution. This non-parametric test compares the sample's empirical distribution to a known probability distribution by determining the largest absolute difference between their cumulative distribution functions (CDFs). A low KS statistic value shows that the selected distribution closely matches the actual distribution of the data, implying that the theoretical distribution fits well and can be used for further research, or simulation.

In our study, we examined different distributions to accurately model our data, including Normal, Log-Normal, and Poisson due to each right-skew characteristic. The Maximum Likelihood Estimation (MLE) was employed to estimate the distribution parameters of the *TotalSuccessCount* variable, which consists of discrete values. Interestingly, as the values increased, its distribution began to resemble a continuous one, aligning with the Central Limit Theorem, which states that the sum of many independent random variables tends to a normal distribution, regardless of the initial distribution. Therefore, we also approximated *TotalSuccessCount* using continuous distributions like the Gaussian and Log-Normal.

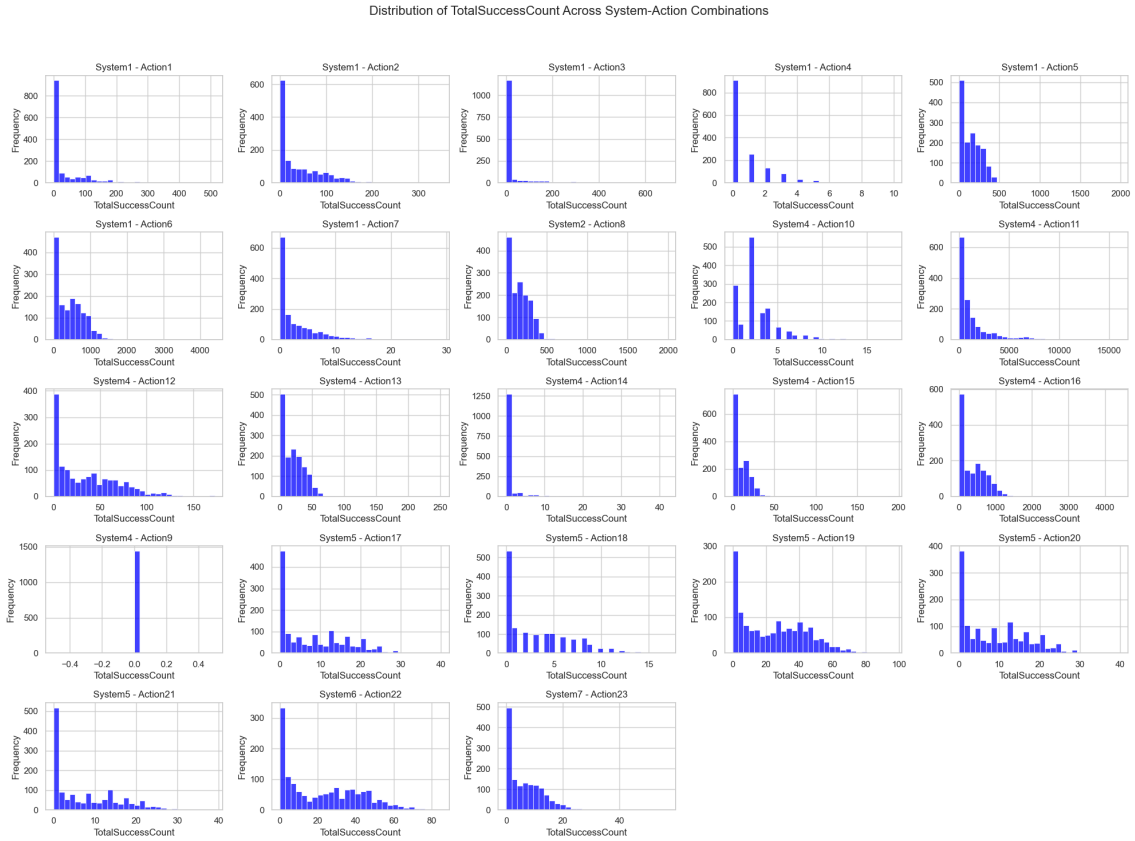

Figure 3.6: Distribution of TotalSuccessCount by System-Action Combination

Given this behavior of the *TotalSuccessCount* variable, we explored both discrete (Poisson) and continuous (Gaussian and Log-Normal) distributions. The Poisson distribution, suitable for count data, captured the right-skewed nature effectively. In contrast, the Gaussian (defined by mean and standard deviation) and Log-Normal (with three parameters: μ, σ, and location) distributions provided flexibility for modeling higher-value trends. These parameters are estimated directly from data using MLE using specialized Python packages (such as `scipy.stats`), which fit the distribution to the data and return estimates for these parameters.

After estimating the parameters for each distribution, the Kolmogorov-Smirnov (KS) test is performed to determine how well the theoretical distributions match the observed data. The p-value indicates the probability that the observed data could have originated from the reference distribution under the null hypothesis. To identify the best-fitting distribution, we focused on the one with the lowest KS value, which indicates the closest match between the actual data and the theoretical model. This best-fit distribution will then be utilized

in our simulation model in Section 4.2 to confirm that the model correctly represents the variables being studied. Here, in Table 3.1, the Kolmogorov-Smirnov (KS) test results are presented for different scenarios, and the best-fitted distributions.

System	Action	Metric	Best Fit Distribution	KS statistic
System1	Action2	WeightedAvgDuration	LogNormal	0.1734
System5	Action18	WeightedAvgDuration	LogNormal	0.1697
System1	Action6	TotalSuccessCount	Gaussian	0.1149
System6	Action22	TotalSuccessCount	Gaussian	0.0944

Table 3.1: Kolmogorov-Smirnov (KS) Statistics for Best-Fitted Distributions

We assessed the suitability of several probability distributions for modeling using two different time series for each of the two variables as representative scenarios. All the fitted distributions for the *TotalSuccessCount* variable in *System1-Action6* and *System6-Action22* can be seen in Figure 3.7. The Gaussian distribution fits best for both time series, effectively modeling both the central peak and the tail behavior. The data was less skewed and more symmetric, making Normal distribution a reasonable approximation.

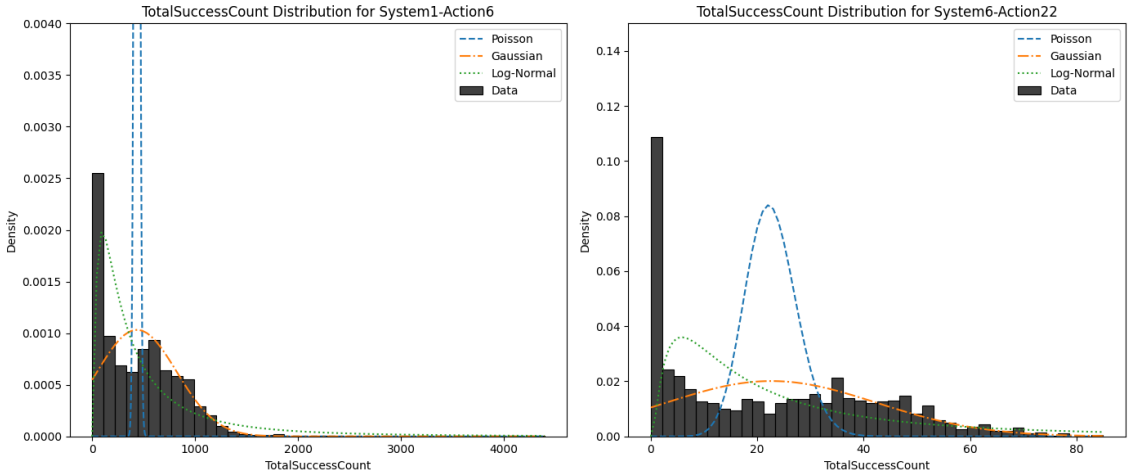

Figure 3.7: Distribution fits for TotalSuccessCount showing the best fit distributions for System1 - Action6 (Gaussian) and System6 - Action22 (Gaussian).

Similarly, for the *WeightedAvgDuration* variable Figure 3.8 showed that the LogNormal distribution provided the best fit for both the time series, the *System1 - Action2* and the *System5 - Action18* as it accurately captured the positive skew and variability observed in the histogram.

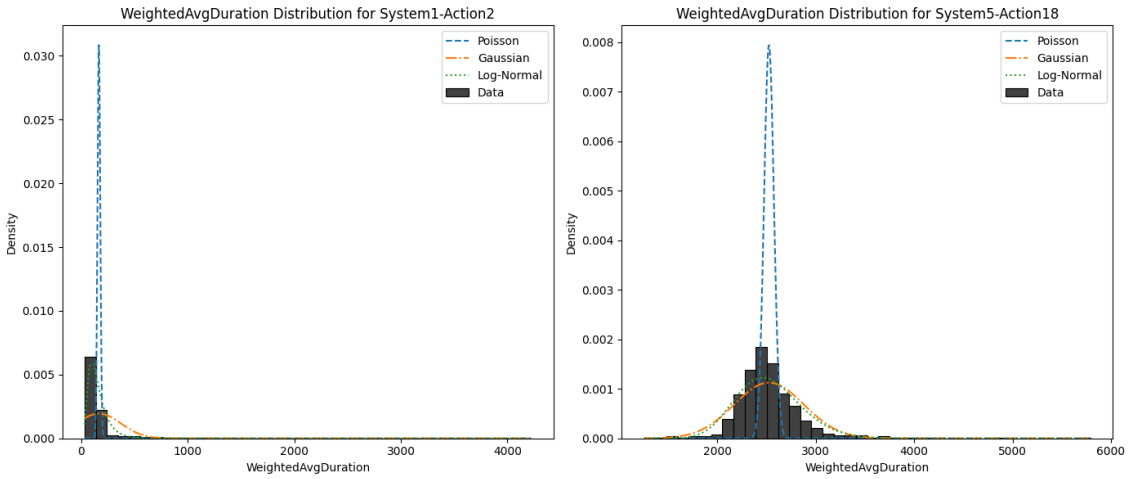

Figure 3.8: Distribution fits for WeightedAvgDuration showing the best fit distributions for System1 - Action2 (LogNormal) and System5 - Action18 (LogNormal).

4

Methodology

This chapter outlines the basic ideas as well as algorithms used in the study including the subsequent time series modeling, simulation model development, followed by seasonality and anomaly detection modeling. The intent is to present a reliable technique for determining whether the time series data can be classified as seasonal or non-seasonal. Following that, a forecasting approach with a built-in confidence prediction interval is utilized to identify anomalies within the identified seasonal time series.

4.1 Time series Modeling

Before proceeding to more complex analytical methods, it is critical to identify and understand the fundamental concepts connected with time series data. Formal definitions define time series data as a sequence of observations $\{X_t\}$ indexed by time t, where $t \in \mathbb{Z}$ (discrete) or $t \in \mathbb{R}$ (continuous). In most cases, however, time series data represents observations recorded at discrete, equally spaced intervals, such as hourly, daily, or yearly observations. The observation X_t may refer to anything from the hourly temperatures of a machine over one hour to the stock values of a company over a day. In contrast to cross-sectional data, which has no recurring components, time series data exhibit a continuous association throughout time. This continuity creates challenges for analysis as time series data cannot be treated as a collection of isolated points; instead, the temporal structure must be incorporated into the analysis.

In practical applications, the data related to time series are observed at a time exhibit three primary characteristics: **Trend**, **Seasonality**, and **Noise**. Trend is defined as an increase or decrease of the values over a reasonably long period while seasonality applies where there are regular or periodic variations of the activity over fixed shorter intervals.

On the other hand, noise bears the random changes in the data that lack any repeating patterns. These components can be modeled separately in an additive format as:

$$X_t = T_t + S_t + R_t \tag{4.1}$$

where T_t represents the trend, S_t the seasonal, and R_t the residual/ noise components.

Time series data can be divided into two types based on the number of variables observed: `Univariate` and `Multivariate`. A univariate time series is a collection of data gathered over time on a certain variable, such as a stock's daily closing price. A multivariate time series, on the other hand, considers more than one variable, such as tracking an economic system across time using indicators such as interest rates, inflation, and unemployment, among others. This paper focuses on univariate time series; therefore, we use the word 'time series' to stand for Univariate time series. The next section will attempt to create multiple time series of such forms.

4.2 Synthetic Data Generation

The primary purpose of this section and consequently of the `TimeSeriesSimulator` class is to create a robust and flexible framework for generating synthetic time series data with real-world characteristics like seasonality, trends, noise patterns, and anomalies. This framework is meant to be a sandbox to execute various data analysis tasks, particularly seasonality and anomaly detection. The TimeSeriesSimulator offers users the ability to provide many configurable parameters that allow substantial control over the attributes that will influence the produced time series. The configurable parameters are as follows:

- **Number of Series (`num_series`):** This parameter specifies how many unique time series will be created. For example, if we were to set this to 100, it would create 100 individual time series.
- **Frequency (`freq`):** Define how often data points are in each time series. The default is set to `1h`, meaning that each data point is one hour apart.
- **Lengths of Series (`lengths`):** This parameter defines the number of data points per time series. For our research, each series has 1,444 data points, which is approximately two months of hourly data, consistent with what we've aggregated in our selected dataset.

- **Noise Type (`noise_type`):** The type of noise that will be added to each series is indicated by this option. All series will be subject to a particular noise type such as `gaussian`, `poisson`,`log_normal`, which are analyzed in subsection 3.4.1.
- **Seasonality Periods (`seasonal_periods`):** Each time series is assigned a set of seasonal periods that determine the recurring patterns in the data. The model uses three main periods: daily (24 hours), weekly (168 hours), and monthly (720 hours).
- **Trend Type (`trend_types`):** This simulation model adds a trend in each series, which may be either linear or non-linear, to show the underlying behavior over time.
- **Trend Slopes (`trend_slopes`):** Each series is given a trend slope selected by the user (i.e., 0.001). These slopes control the rate and direction of the series trend.
- **Anomaly Probabilities (`anomaly_probs`):** Each series is assigned a probability of anomalies happening at a given data point. This parameter enables a controlled but varied level of anomaly injection across distinct series, reflecting true deviations.
- **Application of Zero Values (`zero_indices`):** In order to accurately fit the observed pattern, we added zero values to the time series. A parameter that accounts for the percentage of zeros in each time series was used to accomplish this. This metric seeks to characterize the right skew of our dataset, which consists mainly of zero values.
- **Seasonality (`include_seasonality`):** A boolean flag is added to each series to indicate whether the seasonal components are present or not. A random number between 0 and 1 is generated, and if it exceeds a specified probability (e.g., 25%), the flag is set to True, introducing seasonality into the series; otherwise, it is set to False. This method enables the simulator to generate a wide range of time series, some having periodicity and others not.

These configurable parameters are highly adjustable, allowing IreckonU domain experts to customize them to meet specific needs and demands, particularly when analyzing extended time series data.

4.2.1 Noise Generation

Noise in synthetic time series resembles the random fluctuations seen in real-world data. The `generate_noise()` method of the time series simulator generates a noise component

per each series based on the type of noise specified. For series with Gaussian noise, the simulator creates data points using the normal distribution. This approach employs user-defined mean and variance values derived from the data given by the company, thereby ensuring that the simulated noise closely resembles real-world characteristics. This technique, with its symmetrical distribution around the mean, guarantees that the noise will be very similar to Gaussian noise. When utilizing Poisson noise, the simulator uses the series' specified mean as the lambda parameter to generate values with a Poisson distribution. While the Poisson distribution is suitable for modeling count-based data, we seldom employed it in our study due to its limited effectiveness in best-fitting distribution, as seen in the results of subsection 3.4.1.

Finally, the simulator computes the distribution parameters, mu, and sigma, for series with log-normal characteristics using the mean and variance specified by the user. Because of its inherent skewness, the log-normal distribution—which produces the noise values—is well-suited for simulating phenomena with positively skewed data, such as financial returns or data processing systems. This type of noise makes the generated time series more realistic, and more aligned with what we found when examining the 1-hour dataset which led to discovering more signs of log-normal behavior in the series.

4.2.2 Seasonality and Trend Generation

A realistic simulation of time series data must take into consideration fundamental factors such as seasonality and trends, which represent repeating patterns and directional movements across time. The `add_seasonality()` method adds seasonal components, which are modeled using sine and cosine functions to capture data in a cyclical form. This seasonal component is applied to the time series if the seasonality period is greater than zero. The seasonal component is irrelevant in the absence of a positive period because there is no repetition to the model. The simulator can produce complex seasonal patterns by combining sine waves of different frequencies and amplitudes. For example, this may represent a scenario in which the data exhibits a strong daily pattern impacted by a weekly pattern, as well as other modest oscillates across longer time periods.

This simulation model adds three forms of seasonal cycles: daily, weekly, and monthly, each with a distinct amplitude. These time intervals were selected to correspond with the expected seasonal fluctuations in the hotel business. The daily cycle reflects intraday demand fluctuations, which are critical for daily operational choices like staff and hotel room availability. Demand, for example, usually peaks and drops during the day based on

check-in and check-out times. The weekly pattern can account for differences in demand for weekdays and weekends, which is important for the hotel business. While business travelers might swoop into the property during the week, leisure activity tends to deliver more muscle to weekends in terms of room demand. The monthly cycle does allow you to identify trends over a longer time frame such as more travel during vacation months when business conferences or tourism events are assembled.

The method also uses the `add_trend()` method to add linear or non-linear trends to the series. Trend can be upward, downward, or flat, controlled by `trend_slopes` parameter, depending on the model used. A linear trend is formed by multiplying the slope by the time index. This results in a straightforward upward or downward trend in the data over time, depending on whether the slope is positive or negative. Non-linear trends, on the other hand, might exhibit more complicated patterns, such as a combination of polynomial growth and oscillatory activity. In this model, the non-linear trend creates a trend where the value increases (or decreases) following a power law and adds a quadratic component to the trend, which introduces further curvature.

4.2.3 Anomalies Generation

Anomalies are important to our simulation model because they act as deviations from the normal data and help us to evaluate the accuracy of our anomaly detection algorithms. The `inject_anomalies()` method is designed to generate anomalies, depending on a specified anomaly probability (e.g., 10% for each series). These anomalies result from the injection of large, random deviations into typical values. Their amplitude and frequency might vary, posing a variety of obstacles to anomaly detection systems. For example, a time series can exhibit anomalies seldom, with only a 2% chance of an anomaly occurring at any given time. These anomalies may be in plain sight, large, and perhaps in the range of 100-200 units in terms of variation from the typical value. On the other hand, time series might have outliers more often—10% of data points—but these are relatively small anomalies, resulting in deviations of only 10–20 units. These subtle, repeated deviations can go unnoticed since they are blended in with the surrounding noise. The diversity of the scale and the existence of outliers give credibility to the simulation model as a representation of the real-life scenario in which anomalies in actual data differ in both magnitude and impact.

4.2.4 Time Series Composition

The final synthetic time series is generated by combining each of the constituent components—noise, seasonality, trends, and anomalies—into a cohesive unit. The `generate_data()` method creates the final synthetic time series, Y_t, by combining several components at each time point t. The mathematical formulation of Y_t is given by:

$$Y_t = B_t + T_t + S_t + A_t \tag{4.2}$$

where:

- B_t represents the base noise, which provides random fluctuations in the data, mimicking natural variability.
- T_t denotes the trend component, capturing any long-term increase, decrease or stay flat over the long term in the series.
- S_t is the seasonal component, corresponding to regular predictable patterns occurring at fixed positive periods.
- A_t stands for anomalies, which are deviations from the expected behavior.

Furthermore, the model provides a non-negativity constraint, which ensures that all data points remain above zero and that the values are integer type. This functionality is critical for some forms of time series data, such as sales figures and API call data, in which negative and decimal values are neither practical nor possible.

4.3 Seasonality Detection

This section attempts to identify seasonality in time series data and then classify them as seasonal or non-seasonal. The process involves testing on both synthetic and the company provided time series data using a baseline model against a more advanced approach.

4.3.1 ACF: Baseline Seasonality Detection

The next part provides a baseline model for identifying seasonality using the `Autocorrelation Function (ACF)` method. This baseline approach leads one to probe whether a time series maintains seasonal or non-seasonal characteristics which can be reflected in the autocorrelation patterns at various time lags. This method is a commonly used statistical tool, which measures the correlation of a time series at different time periods (lags). In

the context of time series analysis, these lags relate to prior time points or values in the sequence, allowing researchers to investigate the link between past and present data points. That makes it particularly effective at identifying repeating patterns in time series data, which yields considerable insights into periodicity.

The model computes autocorrelation values (ACF) for the input time series across various lags, which serve as the model's hyperparameter. These values indicate how closely current observations are linked to past observations. Strong autocorrelation at certain periods of time (lags) indicates a potential cyclic behavior. Given a time series with x_t data points, the autocorrelation at lag k is computed as follows:

$$\text{ACF}(k) = \frac{\sum_{t=1}^{T-k}(x_t - \mu)(x_{t+k} - \mu)}{\sum_{t=1}^{T}(x_t - \mu)^2} \tag{4.3}$$

Where:

- x_t represents the value at time t,
- μ is the mean of the time series,
- T is the length of the time series.

However, not all ACF values show repeating patterns; some may be due to random noise or minor oscillations with little to no meaning. Therefore, the classification is guided by a threshold scheme to filter out noisy or inaccurate ACF values, paying attention to events that attain an agreed-upon threshold of significance. Specifically, if any ACF value (except zero lag) exceeds a predetermined threshold, the time series is classified as **seasonal (1)**. In contrast, if all ACF values remain below the threshold, the series is characterized as **non-seasonal (0)**, which means that they do not have a significant repeating structure. The classification rule is expressed mathematically as:

$$y = \begin{cases} 1 & \text{if } \max(\text{ACF}(1), \ldots, \text{ACF}(k)) > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

Where k is the number of lags and the threshold is the predefined cutoff for the ACF values.

This binary classification approach identifies time series with and without recurring patterns. However, as a baseline model, it may fail to handle complicated scenarios like time series with several seasonal cycles or outliers.

4.3.2 Seasonal Quantile Regression: Advanced method

Quantile regression, first introduced by Roger Koenker in his landmark work 'Regression Quantiles', published in 1978 (26), was a key improvement in regression analysis. Traditional regression methods, particularly ordinary least squares (OLS), estimate the conditional mean of the response variable. However, data frequently violate assumptions of error normality, homoskedasticity (constant error variance across observations), and the presence of extreme values (outliers). Due to these limitations of mean-based approaches, the demand for a more robust alternative resulted in the invention of quantile regression. Quantile regression is especially well adapted for our study owing to its innate robustness concerning outliers. It generates accurate predictions even when there are outliers and deviations in the dataset like in our scenario, by minimizing asymmetrically weighted absolute residuals.

Quantile regression has already been applied to investigate how predictors interact across quantiles in the distribution of a response variable, revealing important information regarding data variability. However, since this work targets seasonality detection, we extend the approach by altering it for seasonality detection. In order to achieve this, we suggest using a seasonal quantile regression model designed particularly for detecting seasonal patterns in data. The model assumes a linear connection between the predictors and the response variable, which is appropriate for our exploratory objectives.

4.3.2.1 Seasonal Quantile Regression Design

The following part describes the Seasonal Quantile Regression model's design and implementation, with a focus on how it is employed to classify seasonal and non-seasonal time series data. The model will be applied to both the data generated by `TimeSeriesSimulator` in Section 4.2 and our selected dataset. Each series is analyzed to extract elements that reflect the inherent cycles seen in the hotel business. Thus, several time-related features are extracted:

- **Hour:** This feature tracks the hour of the day (00 to 23). Certain IT systems see spikes in API calls at particular times of the day. For example, check-ins are typically done during afternoon hours (14:00-16:00), while check-outs are typical in the morning (10:00-12:00).

- **Days:** This feature records the day of the month (1-31), helping to identify specific day-based patterns (also known as daily seasonality). For instance, hotels may experience an increase in reservations or cancellations on the last day of the month.
- **Weeks:** This function determines the number of weeks in a given year, allowing patterns to be followed in the weekly calendar. There can be significant spikes during high-activity periods, like holidays or vacation weeks such as Christmas. Certain weeks exhibit recurring patterns, indicating seasonality.
- **Months:** This feature represents each month of the year (1-12) and helps to capture broad seasonal patterns. Different months might have their peaks and lows of the season. Strong seasonality on a monthly time frame is indicated by significant trends like higher activity levels in December or during summer.
- **Day Of Week:** This element consists of the day of the week (0 is Monday, 6 is Sunday). It is beneficial in identifying weekly patterns and variations in customer behavior. Ongoing trends, such as increased weekend traffic, indicate seasonality.

The quantile regression model is employed for each time series to identify the relationship between the time-based features and the target variable, i.e. *TotalSuccessCount*. The model is fitted for three quantiles: 0.05 (lower quantile), 0.50 (median), and 0.95 (upper quantile), which can be considered hyperparameters of our model. This non-parametric approach allows for the analysis of various conditional quantiles of the response variable as a function of the engineered time features. The regression equation is:

$$Q_\tau(\mathrm{Y} \mid X) = \beta_0 + \sum_{i=1}^{4} \beta_i \cdot \mathrm{Time\_Feature}_i \tag{4.5}$$

where:

- Q_τ represents the τ-th quantile of the response variable Y, which can represent the number of successful or failed API calls.
- $\beta_0(\tau), \beta_1(\tau), \ldots, \beta_5(\tau)$ are the regression coefficients of predictors for the intercept and the predictors at quantile τ.
- X represents the vector of predictors (Hour, Day, Week, Month, DayOfWeek).

After fitting the seasonal quantile regression model, we extract the coefficients (β values), focusing on the median quantile (0.50). These coefficients measure the magnitude and direction of each feature's association with the response variable while keeping all other variables constant. As a result, each quantile produces a unique set of coefficients that describe how the predictors interact with various sections of the response distribution. For example, a positive coefficient for 'Hour' in the middle quantile indicates that later hours of the day are associated with more successful API calls, indicating evening check-in activity.

This is accomplished through the standard errors, which measure the variability and uncertainty of the coefficients. This leads to much more precision, which translates into more confidence about the correctness of your estimations. The t-statistic, calculated as a ratio of the coefficient to its standard error, is then used to derive p-values in order to gauge statistical significance. These p-values assist us in identifying the temporal characteristics that have a significant impact on the variable that we are studying. The hypotheses for detecting seasonality based on temporal predictors are as follows:

- $\mathbf{H_0}$: The temporal predictors have no significant effect on the response variable, implying no seasonality in the time series. Formally:

$$H_0 : \beta_1 = \beta_2 = \beta_3 = \beta_4 = \beta_5 = 0$$

- $\mathbf{H_1}$: At least one temporal predictor (β_i) has a significant effect on the response variable, supporting the presence of seasonality. Formally:

$$H_1 : \beta_i \neq 0 \quad \text{for at least one } i \in \{1, 2, 3, 4, 5\}$$

For instance, a small p-value (e.g., < 0.05) for 'Days' at the median quantile implies daily seasonality making the time series classified as **Seasonal**. On the other hand, non-significant p-values indicate that the temporal feature has no meaningful impact and thus it is labeled as **Non-seasonal**. Therefore, a time series is seasonal if one or more of the time features (i.e., hour) where consistently affect the response variable, showing that regular variations over time could occur. The seasonality classification model is then enhanced using further refinements that would improve model performance and accuracy.

1. **Variance Threshold:** A variance threshold was introduced to reduce the influence of high variance time series, which makes seasonal trends difficult to spot and classify. With this criterion, we can now label time series as seasonal or non-seasonal and also flag those that show significant variability for further investigation.

2. **Data Sparsity Rule:** In the case of sparse time series, a rule was developed. If the percentage of non-zero data points is less than 5% in the given time series, then it is classified as non-seasonal since random spikes might mask underlying patterns and provide a sense of seasonality.

The following flowchart in **Figure 4.1** illustrates the step-by-step process for classifying N time series based on seasonality.

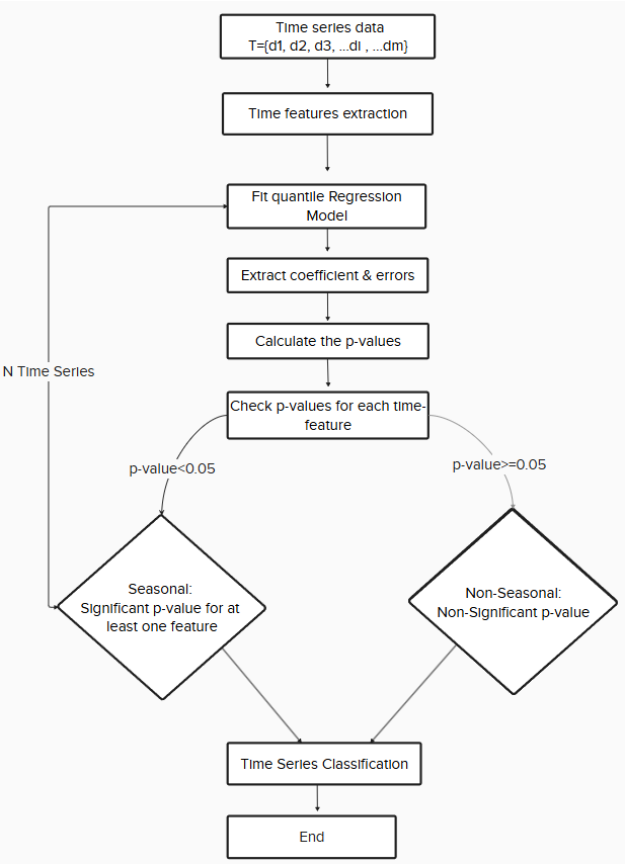

Figure 4.1: Flowchart of the Time Series Classification Process

4.4 Anomaly Detection

In this section, the anomaly detection module attempts to discover anomalies in the identified seasonal time series provided in Section 4.3. It involves analyzing both synthetic and company provided time series data using an anomaly baseline approach against a more advanced algorithm prediction-driven approach.

4.4.1 AD-PCI: Baseline Anomaly detection

In this part, the baseline anomaly detection technique comprises of a basic forecasting approach and constructed Predictive Confidence Interval (PCI). This method predicts the expected values of the data point by taking into account a weighting k past and k future data points around a certain time step. A non-linear weighting structure is used to enhance nearby time points while gradually decreasing the significance of farther away data points. The basic principle is determining a weighted average of neighboring values, which is then used to establish the upper and lower confidence intervals. These boundaries assist in assessing whether current observation deviates significantly from predicted patterns.

For each time step t in the time series, the approach computes a weighted average of both previous and future values. This weighted average represents the expected behavior for time step t, which is compared to the actual observation to detect anomalies. The formula for computing X_t is given by:

$$X_t = \frac{\sum_{j=1}^{k} w_{t-j} X_{t-j} + \sum_{j=1}^{k} w_{t+j} X_{t+j}}{\sum_{j=1}^{k} w_{t-j} + \sum_{j=1}^{k} w_{t+j}} \tag{4.6}$$

Where:

- X_{t-j} and X_{t+j} are the past and future values around time t,
- w_{t-j} and w_{t+j} are the weights associated with each past and future value, inversely proportional to the distance from the current point t.
- The weights are defined as: $w_j = \frac{1}{j+1}$

This approach prioritizes the nearest points while decreasing the weight of the points further away. The method is guided by two important parameters: *alpha* (a), which limits the allowed width of the Prediction Confidence Interval (PCI), and k, which specifies the window length, or the number of previously observed data points used in the assessment. The forecasting procedure would need to be modified to allow the model to run both online and offline, as it currently only operates in offline mode.

After we have calculated all the X_t values, the next step is to check if the actual value at time t falls within the expected range using a Prediction Confidence Interval (PCI) around X_t. These intervals are based on the standard deviation s of the available data points within the window around t and use the t-distribution to account for the variability of small samples, in particular when working with smaller window sizes. The formula used for the confidence interval is:

$$PCI = X_t \pm t_{\alpha,2k-1} \cdot s \cdot \sqrt{1 + \frac{1}{2k}} \tag{4.7}$$

Where:

- $t_{\alpha,2k-1}$ is the critical value from the t-distribution for a confidence level α, with degrees of freedom $2k - 1$,
- s is the standard deviation of the values in the window around t,
- k is the window size, and the term $\sqrt{1 + \frac{1}{2k}}$ adjusts the width of the confidence interval based on the window size.

Once the confidence interval is calculated, the observed value at time t, denoted as y_t, is compared against the confidence bounds. The anomaly detection rule is as follows:

$$\text{Flag as Anomaly} = \begin{cases} 1 & \text{if } y_t \notin [\text{Lower Bound}, \text{Upper Bound}] \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

In this framework, we label a data point y_t as an anomaly if falls outside of the specified range (i.e., the PCI). Conversely, if y_t lies within the bounds, it is considered a normal point, meaning it conforms to the expected behavior based on the model's prediction. While this approach can be effective for series with clear and consistent seasonal patterns, it lacks the ability to adjust dynamically to changing distributions or trends over time, which may result in false positives or negatives in highly non-stationary data.

4.4.2 Neural Prophet: Advanced Anomaly Detection

Neural Prophet, developed by Triebe et al. (2021), is an explainable forecasting framework that boosts scalability and interpretability for time series analysis (27). Neural Prophet, which builds on Facebook's Prophet model (28), uses neural networks to more precisely capture complicated non-linear trends and seasonal patterns. This makes it particularly useful for detecting anomalies because it not only produces forecasts but also offers adjustable prediction intervals. Neural Prophet preserves Prophet's decomposition technique, which divides time series data into trend, seasonality, and residuals, but it now integrates neural network components to reveal more complex relationships within the data.

4.4.2.1 Neural Prophet Design

In order to harness the power of the algorithm, we implement a data preprocessing and model training strategy, splitting each time series into training, calibration, and test sets to enhance generalization and limit the risk of overfitting. Neural Prophet can identify long-term trends, seasonality, and potential anomalies through training from historical data. In addition to single-point forecasts, Neural Prophet can generate prediction intervals using the quantile regression approach, which identifies intervals where future values are expected to fall. Instead of developing a single forecast, Neural Prophet employs a quantile regression-based loss function to predict several quantiles of the target variable, resulting in a probabilistic picture of future values. For example, using upper and lower quantiles q_l and q_u (e.g., the 10th and 90th percentiles), we obtain a prediction interval:

$$\text{Interval}(t) = [\hat{y}_{q_l}(t), \hat{y}_{q_u}(t)] \tag{4.9}$$

which typically captures the middle 80% of the distribution.

While prediction intervals can also give information regarding coverage, which is useful for determining the uncertainty about future data points, intervals may not behave as anticipated based on data characteristics or under certain conditions. This is where conformal prediction comes into play, mitigating this challenge by calibrating these ranges to better match the intended confidence level. Conformal prediction is a statistical approach that analyzes data's residuals on a calibration dataset to ensure that prediction intervals fall where they ought to, allowing for more flexible interval modifications. For a target confidence level of $1 - \alpha$, conformal prediction guarantees that:

$$P(y(t) \in \text{Interval}(t)) \geq 1 - \alpha \tag{4.10}$$

Our approach uses a simple yet effective form of conformal prediction, known as **naïve conformal prediction** (29). When compared to advanced methods such as Conformalized Quantile Regression (CQR), this methodology requires no further training, making it computationally efficient and well-suited for massive, diverse datasets like the one we have. Specifically, using the quantile regression technique to produce prediction intervals, we extend the intervals by adding or subtracting the greatest absolute residual $r_{\max}$ from the calibration set:

$$\text{Adjusted Interval}(t) = [\hat{y}_{q_1}(t) - r_{\max}, \hat{y}_{q_2}(t) + r_{\max}] \tag{4.11}$$

These adjustments maintain the intervals balanced—neither too cautious nor too narrow—improving accuracy and reliability without requiring too much processing work.

The next phase is to establish a rule for identifying potential anomalies. This is accomplished by comparing actual observations from the test set to the predicted confidence intervals. If an observation goes outside of these limits—either surpassing the upper limit or dropping below the lower limit—it is classified as an anomaly, indicating an unexpected deviation from the model's predictions.

Given the specific nature of our dataset (e.g., API Call data), all predicted values must stay non-negative. Negative predictions are inconsistent and may diminish the efficiency of anomaly detection. To overcome this, we employ a post-processing phase that clips any negative predictions to zero, placing a lower constraint on the forecasts. Figure 4.2 shows the Workflow of the Neural Prophet anomaly detection model.

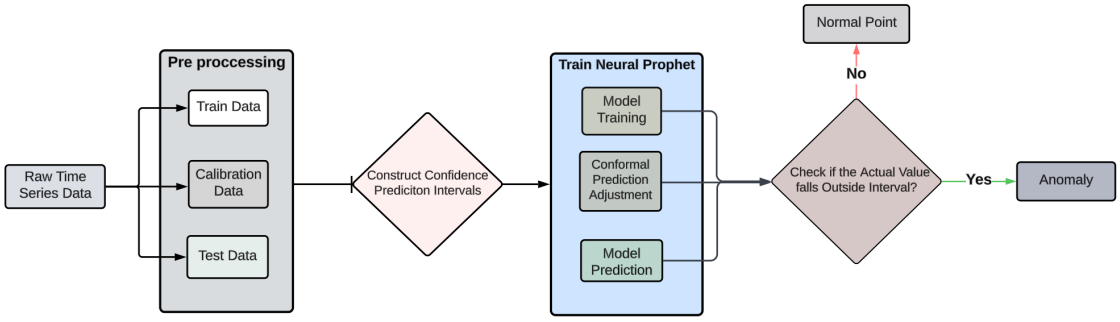

Figure 4.2: Workflow of the Neural Prophet model

5

Experimental Setup

5.1 Evaluation Metrics

To evaluate how effective the model is in identifying seasonality and anomalies, we utilize a number of key metrics, each of which provides a distinct view of its effectiveness and reliability. The following metrics were used in this research:

1. **Recall**: Recall evaluates the model's ability to identify all actual anomalies.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.1}$$

2. **Precision**: Precision measures the proportion of true anomalies among all detected anomalies.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

3. F_β**-score**: The F_β score is calculated as:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \tag{5.3}$$

   In this case, $\beta = 2$, which places 2 times more emphasis on recall.

4. **Confusion Matrix**: The confusion matrix provides a summary of the model's performance by displaying true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

(5.4)

5. **ROC Curve & AUC (Area Under the Curve)**: The Receiver Operating Characteristic(ROC) curve is a graphical representation of the trade-off between the True Positive Rate (Recall) and the False Positive Rate (FPR) across different threshold values. The AUC summarizes the performance of the model: a model with an AUC close to 1 is considered excellent, while an AUC of 0.5 suggests random guessing.

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN} \tag{5.5}$$

5.2 Time series splitting

In time series modeling, it is important to effectively split the dataset for developing a solid framework, which captures historical patterns and makes precise forecasts. In order to accomplish this, the data has been divided chronologically into three parts: training, calibration, and testing sets.

- **Training Set (70%)**: This set consists of the first 70% data and serves as the basis for predicting accuracy. It enables the model to identify the underlying patterns and potential anomalies in the data.
- **Calibration Set (10%)**: A small number between the training and testing sets. In conformal prediction, this set aids in calibrating a prediction interval that closely reflects the distribution of errors in the forecast.
- **Testing Set (20%)**: This segment contains the most current data and remains unseen throughout the training process. It is primarily utilized to assess the model's performance.

5.3 Hyperparameter Tuning

Hyperparameter tuning is the process of tweaking hyperparameters that control the model's learning process. This process seeks a balance between overfitting and underfitting in order to achieve optimal performance. Conventional approaches, such as grid search and random search, are inefficient, especially when applied to high-dimensional search space. To accomplish this, a more balanced approach is the `Bayesian optimization`. This method is built on the Hyperopt python package, which refines hyperparameters by utilizing previous knowledge of the search space and continually narrowing it down.

Internally, Hyperopt uses Tree-structured Parzen Estimators (TPE), a Bayesian technique that classifies assessments as successes or failures, directing the search to regions of the parameter space with a high probability of producing positive outcomes. TPE is especially useful when optimizing for complex search spaces like deep learning or time series forecasting, where extensive parameter research can be time-consuming. Throughout this process, each model's particular hyperparameters were adjusted to improve performance. The important hyperparameters that have been modified for each model are summarized here:

5.3.1 Autocorrelation Function (ACF) model

- **Number of Lags**: The number of lags determines how many time periods are used to calculate ACF values. Selecting the appropriate range of lags is critical for detecting seasonality, which might occur at various frequencies.
- **ACF Threshold**: This number distinguishes the seasonal and non-seasonal time series. This enables for the reliable detection of significant autocorrelations while avoiding allowing noise to result in false correlations.

5.3.2 Seasonal Quantile Regression (SQR) Model

- **Quantiles**: These are the predicted quantiles, which allow the model to incorporate variability in the data at various distribution levels (low, median, and high).

5.3.3 Anomaly Detection Prediction Confidence Interval (AD-PCI) Model

- **Alpha (confidence level)**: The threshold for sensitivity in anomaly detection to be used to control the likelihood of an observation being classified as anomalous.
- **k (data points to include)**: The number of previous and future points to be used to forecast the actual values.
- **beta**: Weight for recall versus precision in the F-beta score.

5.3.4 Neural Prophet Model

- **n_lags**: The number of previous time steps used to predict the current value, capturing seasonality across past data points.

- **trend_reg**: Determines how strong the trend is, and hence how much we want to regularize it to avoid overfitting.
- **seasonality_reg**: Determines how much regularization to apply to the seasonal component to ensure a smooth seasonal pattern.
- **learning_rate**: Determines the speed at which the model updates its weights during training.
- **changepoints_range**: Defines the portion of the data used to detect trend changepoints.
- **fourier_order**: Number of Fourier terms used to capture seasonality.
- **epochs**: The number of complete passes through the dataset during training.
- **batch_size**: Number of samples per training batch, affecting the speed and accuracy of model training.
- **quantiles_list_index**: Refers to the specific quantiles being used for probabilistic predictions.
- **beta**: Weight for recall versus precision in the F-beta score.

Each model's hyperparameters were tweaked to achieve the optimal balance between accuracy and recall. Since the costs of FP (false positive) and FN (false negative) are unknown, it is difficult to select the appropriate β value in this study. However, higher recall has been favored in the domain of hospitality IT systems, where reliability and visitor experiences are crucial. False negatives (missing a potential issue) lead to unaddressed failures or service disruptions, more damaging than the occasional false positives (warning falsely to a potential issue). Consequently, hyperparameter tuning prioritizes recall while maintaining a sufficient level of precision by maximizing the F_β-score with $\beta > 1$. This also ensures critical events don't go undetected, reducing the chances of anything negatively impacting system reliability.

Given the wide range of time series examined, each with its own set of ideal hyperparameters, reporting all individual results would be excessive and confuse the study. For reproducibility, histograms displaying the distributions of selected hyperparameter values across different methods used in this research can be found in Figure 8.4, Figure 8.5, and in Figure 8.6 of our Appendix.

6

Results

This section provides an extensive review of the results, starting with an assessment of the alignment of the data that the company provided and the data that we simulated, followed by seasonality and anomaly detection outcomes. The model's performance is assessed using performance metrics, confusion matrices, and ROC curves.

6.1 Data Similarity Assessment

In statistical analysis, it is critical to ensure that the simulated datasets accurately reflect the behavior of the actual data. Plotting the dataset's graphs is a straightforward way to compare the similarities between them. By visualizing the data for both series, one may quickly see patterns, trends, and potential anomalies, allowing us to determine similarities and dissimilarities between the datasets.

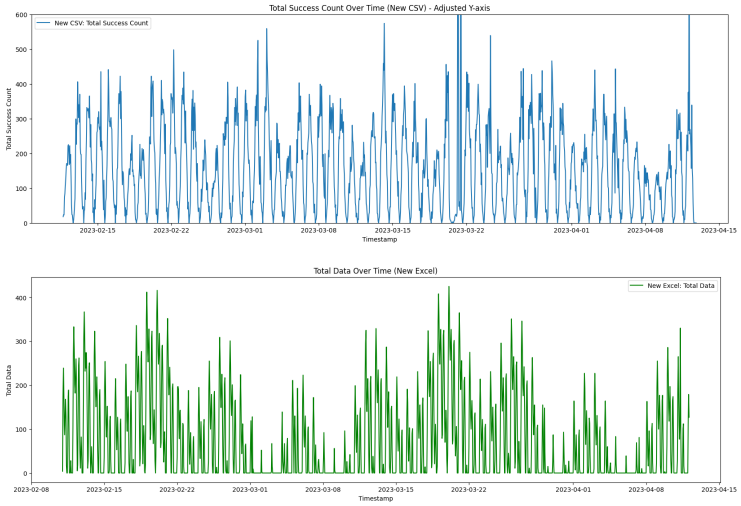

Figure 6.1: Visual comparison of time series data from simulated and real dataset

Both graphs in Figure 6.1 show similar patterns, with consistent peaks and declines that indicate typical weekly periodicity. This shows that the simulated data closely resembles the behavior of the real data, including essential characteristics like periodicity and amplitude. The visual comparison is beneficial, but quantitative procedures such as permutation tests are required to properly analyze similarity, especially for datasets with contain anomalies, time-varying patterns, or non-standard distributions.

6.1.1 Permutation tests

Permutation tests are a non-parametric approach to compare datasets that makes no assumptions about the distribution of the datasets. Before computing the target measure, the approach performs a series of random permutations of the data's label to generate a null distribution that serves as a baseline for comparison. Such a process is highly efficient in identifying correlations in features between simulated and real data sets that go beyond the usual metrics of mean and variance.

In our research, permutation tests will be used to evaluate important properties of our datasets. Comparisons of these will help us to determine how well the simulated data replicates IreckonU's dataset and confirm the suggested methods' accuracy. Specifically, permutation tests will help us to detect subtle differences or similarities between our datasets, which might not be captured by conventional statistical metrics. To facilitate this study, we formulate the following hypotheses:

- $\mathbf{H_0}$: There is no significant difference between the simulated time series and the dataset sourced directly from the company.
- $\mathbf{H_1}$: There is a significant difference between the simulated time series and the dataset sourced directly from the company.

In our scenario, we have a set of 23 simulated time series (**Group A**) that mimic the *TotalSuccessCount* variable and a set of 23 time series derived from each System-Action combination of the *TotalSuccessCount* (**Group B**). The objective is to determine whether a significant difference exists between these two groups, examining different features.

6.1.1.1 Average Length of Time Between Consecutive Anomalies

The first feature examined was the average time between two consecutive anomalies. In this test, anomalies were defined as data points that deviated from the mean by more than two standard deviations. By setting this threshold, the function identifies anomalies outside

of this 95% range, capturing values that are particularly far away from the mean—the extreme 5% of data points. This setting strikes an appropriate balance between sensitivity and false positives. A lower threshold would classify too many data points as unusual, including typical oscillations around the mean while a higher threshold would miss some significant outliers. When all of the peaks had been identified, we estimated the time gaps between the peaks in both the real and simulated groups, yielding two types of average intervals. The observed statistic in this model indicates the difference in average interval length between two anomalies for the simulated and real-time series groups. This difference indicates if anomalies in the simulated data occur more or less frequently than in the real data. By concentrating on the average interval length between two consecutive anomalies, we can create an easily interpretable metric that reflects an important feature of time series behavior: how frequently two consecutive anomalies occur. This feature matters since anomalies are produced randomly rather than at specified time intervals, making the test applicable.

Under the null hypothesis that there was no significant difference between the two datasets, interval lengths were merged before running the permutation test. It shuffled and then split the intervals into two groups, repeating the process 10,000 times. To get the p-value, the observed statistic was compared to the distribution of permuted differences. Mathematically, the p-value can be calculated using the following formula:

$$p\text{-value} = \frac{\text{Number of permuted test statistics} \geq \text{Actual test statistic}}{\text{Total number of permutations}} \tag{6.1}$$

A low p-value (typically less than 0.05) indicates that the observed difference is unlikely to have happened by randomness, implying a significant difference in interval lengths between the simulated and real data groups. In this test, the observed difference was around 0.64, with a high p-value of 0.4187, showing no significant difference in interval lengths between the simulated and actual datasets. A difference of 0.64 indicates that, on average, the simulated group has less than one unit difference from the real group.

6.1.1.2 Difference between Non-Anomalous Points

To determine the differences between the two sets of data points, we looked at the number of non-anomalous points within two standard deviations of the mean. A custom function has been produced for each batch of time series data to determine the mean, standard deviation, and number of non-anomalous points. We repeated the test 10,000 times to compare the difference in non-anomalous numbers between the two data groups.

The observed statistic, which represents the actual difference in the average non-anomalous point count, was compared against the distribution resulting from these permutations. The observed value of -1.043 indicates that the real data (Group B) on average had slightly more non-anomalous points than the simulated data (Group A). However, the difference was not statistically significant (p-value= 0.578), suggesting that the variance is due to randomness rather than a true difference between the two datasets.

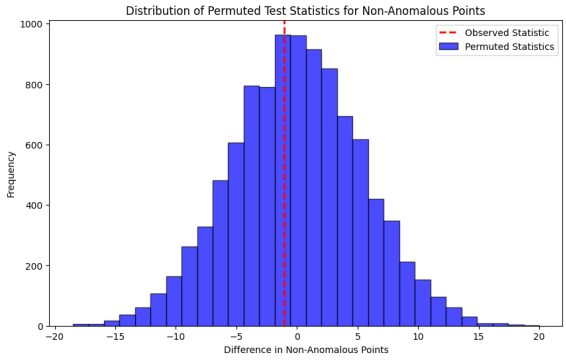

Figure 6.2: Distribution of Permuted Test Statistics for Non-Anomalous Points.

The histogram 6.2 shows the distribution of differences in non-anomalous points between simulated and real datasets, generated by random permutations. The blue bars show the frequency of these differences under the null hypothesis, which assumes there is no significant distinction between the groups. The red dashed line represents the observed difference, which is close to zero. The observed difference falls well within the range of permuted differences, implying that it is most likely due to random chance, suggesting no meaningful difference across the two groups.

6.1.1.3 Approximate Entropy

We also conducted a test to assess the similarity between these 2 groups of time series data using Approximate Entropy (ApEn), a metric that quantifies regularity and predictability in time series. Testing Approximate Entropy matters as it captures the underlying dynamic behavior of the datasets. ApEn works by selecting subsequences of the time series—here, we used subsequences of 72 data points. This subsequence length (3 days) ensures that there are enough data points to capture the patterns and any inherent periodicity. ApEn calculates the likelihood that patterns of closely related observations will remain consistent as the series progresses, for each time series in the actual and simulated data groups. Low ApEn values show regularity and predictability, whereas bigger

ApEn values indicate greater complexity and less predictability. For each time series in both groups, we computed the ApEn metric and then generated two sets of ApEn values. We utilized the absolute difference between the average Approximate Entropy values of real and simulated groups as the permutation test statistic.

For the permutation test, we utilized the absolute difference between the average Approximate Entropy values of real and simulated groups as the permutation test statistic. To carry out the test, we added the ApEn values from both groups and randomized the labels 10,000 times. During each repetition, 23 entropy values were randomly allocated to either the real or simulated groups, with the test statistic recalculated each time. The p-value of 0.4901 suggests that there is no statistically significant difference in the Approximate Entropy (ApEn) of real and simulated time series. This minor discrepancy is most likely due to random change, implying that the groups have comparable dynamic qualities of regularity and complexity.

6.2 Seasonality Detection Results

In the next part, we evaluate the effectiveness of the ACF model against a more sophisticated approach: SQR for identifying and classifying seasonal time series data. We may acquire important insights into how the advanced method improves seasonality detection and classification over the baseline model. The study was conducted using both synthetic time series data and data provided by the company.

6.2.1 Results on Synthetic Data

We start the assessment in the control setting environment, where the outcomes of our methods can be validated. To conduct the test a set of 23 different simulated time series were generated to evaluate the model's ability to detect seasonality. These time series have been generated explicitly to replicate the behavior of the *TotalSuccessCount* variable at different periods and include both seasonal and non-seasonal series. Initially, we assessed the baseline seasonality detection model, which uses the Autocorrelation Function (ACF) approach. This simplified model performed rather well, accurately classifying around 63% of the non-seasonal time series (5 out of 8) and 46.6% of the seasonal time series (7 of 15). As seen in the confusion matrix (Figure 6.3), the ACF model had a greater number of false negatives, frequently misclassifying seasonal series as non-seasonal.

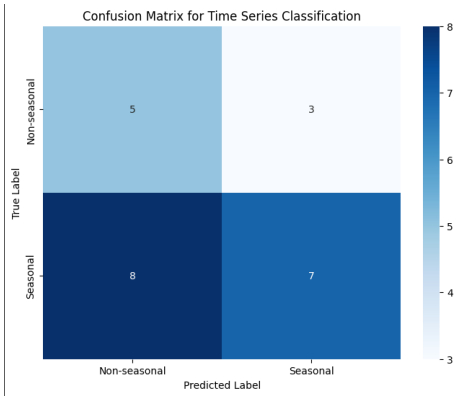

Figure 6.3: Confusion Matrix for ACF method

Given the modest performance of the baseline model, which failed to accurately classify seasonal series, the advanced model performed significantly better in recognizing seasonality. The confusion matrices in Figure 6.4 show that the model correctly identified most of the non-seasonal time series. The model is extremely accurate in a range of scenarios and time periods. For example, in the very first confusion matrix, the model correctly classified 13 out of 14 non-seasonal series. However, there was one case where the model identified a seasonal time series while it had non-seasonal characteristics. This single misclassification demonstrates that, while the model may occasionally identify non-seasonal data as seasonal, such mistakes are rare, happening in less than 10% of all instances. The remaining matrices demonstrate comparable performance. In a balanced situation with an equal number of seasonal and non-seasonal data, the model correctly identified 9 of 11 non-seasonal series. In an unbalanced environment with more seasonal data than non-seasonal ones, the algorithm successfully recognized 5 of 7 non-seasonal series. This consistent performance across situations indicates the model's capacity to recognize stable, non-recurring patterns typical of non-seasonal series, even when datasets have varying class distributions.

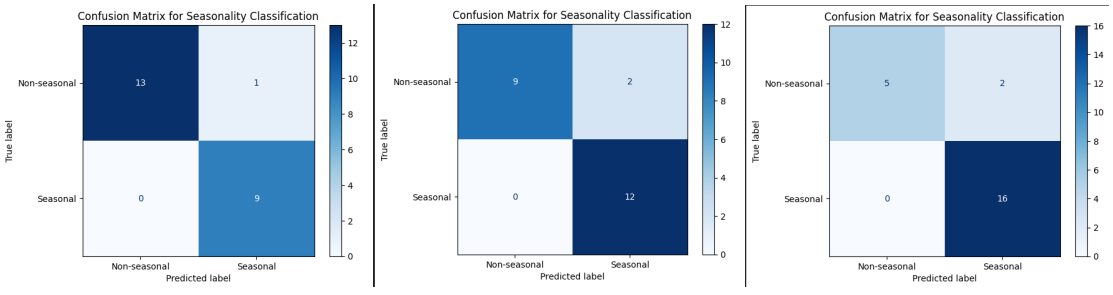

Figure 6.4: Confusion Matrices for Seasonal Quantile Regression at different time periods

The algorithm performs equally well when classifying seasonal series. For the first matrix, all nine seasonal series are accurately identified. This great performance is also constant across the other matrices since each seasonal series is correctly detected and classified. The model's consistent detection of seasonal series demonstrates its ability to capture repeating patterns and cyclical behaviors independent of time period or class distribution.

In addition, the Receiver Operating Characteristic (ROC) curve illustrates the model's ability to distinguish between seasonal and non-seasonal time series by plotting the True Positive Rate (TPR) (also known as recall) and the False Positive Rate (FPR) at various threshold settings. In Figure 6.5 we observe a sharp rise toward the top left corner which indicates a high true positive rate, and the Area Under the Curve (AUC) of 0.94 reflects the model's strong classification performance.

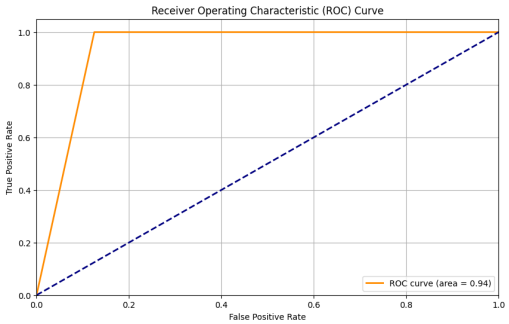

Figure 6.5: ROC Curve for Balanced time series distribution

Part of the analysis focuses on scenarios when the model classifies a time series as seasonal despite the fact that it is not seasonal in nature. Looking more closely at the time series in Figure 6.6, there are irregular spikes, with some of the points rising to significantly higher levels than the base. These peaks occur in an unpredictable manner, as opposed to seasonal data, which has predictable recurring patterns. If the model classified this series as seasonal, it might indicate that the model is susceptible to these random spikes, mistaking them for recurrent seasonal signals.

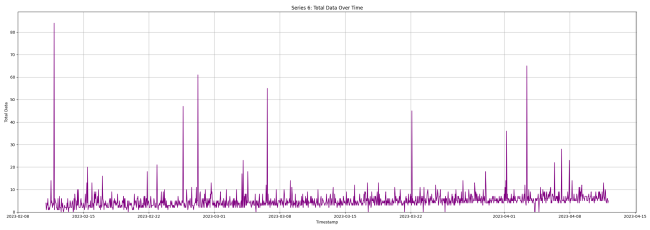

Figure 6.6: Misclassified Series

6.2.2 Results on Real Data

When it comes to our dataset provided by the company the focus of the seasonality detection was placed on variables like *TotalSuccessCount* and *TotalErrorCount*. The decision not to apply the seasonality detection module to the *WeightedAvgDuration* stemmed from the high frequency of non-recorded data points across many time series as discussed in Section 3.1. Since our model required a continuous dataset to properly identify recurrent patterns, the data's large gaps made accurate detection impossible. The seasonal quantile regression model was employed for analyzing these variables, however, due to the lack of pre-labeled data in an unsupervised situation, the model depended mainly on high-quality findings from synthetic data.

As seen in Table 6.1 for *TotalSuccessCount*, most time series are classified as seasonal, with only three identified as non-seasonal. Notably, the pair of *System4-Action11 and 16* as well as *System1-Action6* are classified as seasonal however, it is debatable whether these series are actually seasonal or if the noticeable fluctuation is just the result of random noise. Although the model classifies them as seasonal, additional research is required to understand if these variations are truly caused by periodicity or simply mimic seasonal patterns.

Category	**System-Action Combinations**
Seasonal	System1: Action1,Action2,Action3,Action6 Action4, Action5, Action7 System2: Action8 System4: Action11,Action12,Action13, Action15, Action16 System5: Action17,Action18,Action19, Action20,Action21 System6: Action22 System7: Action23
Non-Seasonal	System4: Action9,Action10,Action14

Table 6.1: Classification Results for Seasonal & Non-Seasonal Time Series for TotalSuccessCount variable.

Conversely, most of the time series of the *TotalErrorCount* variable (as shown in Figure 8.3) were classified as non-seasonal. This classification mainly results from the classical representation of the time series in which zeros are prevalent in most of them. As we see in Figure 8.2 when we look at the distribution of errors (and non-errors), we find limited variability in our dataset since over 95% of the data points are error-free in most of these series. As a result, the model does not detect significant seasonal patterns in the available

time series data. Ultimately, only *System4-Action9* was declared seasonal. The *TotalErrorCount* for that time series had more variance and a lower percentage of zero values compared to other combinations. The relative increase of non-zero data points meant that the model was able to detect relevant patterns associated with time features.

6.2.3 Cross-validation Setup

Model validation is a key critical stage in model development that involves testing the model with new and seen data to ensure that it responds properly. To test the robustness and stability of our model, we used a time series cross-validation approach using a custom quantile loss function (`mean_pinball_loss`) as evaluation metrics. This loss function illustrates the asymmetric nature of quantile predictions by penalizing under- and over-prediction differently. The quantile loss function is defined as follows:

$$L(q, y_{\text{true}}, y_{\text{pred}}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} q \cdot (y_{\text{true},i} - y_{\text{pred},i}), & \text{if } y_{\text{true},i} \geq y_{\text{pred},i} \\ (q-1) \cdot (y_{\text{true},i} - y_{\text{pred},i}), & \text{if } y_{\text{true},i} < y_{\text{pred},i} \end{cases} \tag{6.2}$$

where q is the quantile of interest, y_{true} represents the true values, y_{pred} denotes the predicted values.

Given the sequential structure of time series data, we utilized `TimeSeriesSplit` to preserve it in an appropriate sequence and avoid data leaking (30). This method separates the data into 10 successive folds, guaranteeing that the training data always precedes the validation data. For each fold, quantile loss is calculated for both the training and validation sets. Finally, the training and validation losses are averaged across all folds to provide a summary of the model's performance for each time series. This averaging process helps smooth out the variability across different time splits, offering a more reliable assessment of the model's ability to generalize.

To assess overfitting, we compute and compare the average training and validation quantile losses per time series. The results can be seen in Table 8.2. Overfitting may occur if the validation loss is significantly greater than the training loss. The large gap in time series: *System1-Actions 5 and 6* and *System4-Actions 11 and 16* suggests that the model might fit the training data too well and struggle to generalize to unseen data.

Our approach is to apply the Lasso regularization (L1 penalty) to address this issue, on these specific time series. Our objective was to determine if the classification results, especially the ability to discriminate between seasonal and non-seasonal series, would stay

constant when regularization was implemented. Lasso tends to shrink some model coefficients to zero, which might have an impact on feature significance and the classification process. We tested several levels of regularization (0.01, 0.5, and 1.0), but the classification findings remained consistent, with the selected time series still being recognized as seasonal. This consistency supports the accuracy of our original classification. Even after regularization, the key features in each series remained to highlight crucial seasonal trends, showing the robustness and reliability of our original classification results.

6.3 Anomaly Detection Results

In this result section, we examined two methods for detecting anomalies in seasonal time series data: a baseline model against a more advanced model. The baseline model, which used the typical anomaly detection method, does not account for seasonal variations. In contrast, the Neural Prophet method involves components that model the recurrent patterns, which enhances the ability to detect anomalies.

6.3.1 Results on Synthetic Data

The models were first tested on the identified seasonal synthetic time series, detected using the method presented in 4.3.1. Similar to seasonality detection these synthetic data were specifically designed to test the models' ability to detect anomalies in the presence of seasonal variations. However, once the models are optimized, the results are summarized using the F_β^*-score rather than the F_β-score. This distinction is crucial because F_β^* represents the maximum F_β value achieved during the tuning process, identifying the best configuration of hyperparameters for the Neural Prophet model. Using F_β^* ensures that only the highest performance under optimal conditions is reported, avoiding any ambiguity that could arise from averaging F_β values across less optimal configurations.

The anomaly detection baseline model, which lacks sophisticated mechanisms to address seasonality, achieved a 68% precision. This suggests that it properly recognized a significant percentage of real anomalies among those discovered. However, its recall was significantly lower at 37%, indicating that it missed a large proportion of actual anomalies. The model's low F_β^*-score of 43% reflects its tendency to disregard anomalies associated with large seasonal patterns, as seen by the gap between precision and recall. While the model's conservative approach substantially reduces false positives, it also leaves many critical anomalies undetected.

Model	Precision	Recall	F_β^*-Score
AD-PCI Model	**68%**	37%	43%
Neural Prophet Model	24%	**63%**	**55%**

Table 6.2: Comparison of Model Performance on Seasonal Synthetic Data

On the other hand, the Neural Prophet Model displayed a greater ability to control seasonality and detect an increased number of anomalies. It achieved a 63% recall rate, correctly detecting a substantially higher number of actual anomalies. However, its precision was lower (24%), indicating a greater number of false positives. Despite the trade-off, the model's F_β^*-score of 55% indicates a more balanced overall performance compared to the baseline, suggesting that the Neural Prophet Model may be better suited for cases where maximum recall is critical. This approach detects anomalies more thoroughly while also flagging more occurrences.

Missing anomalies in the hotel business, especially when tracking API call data, might result in significant operational issues. For example, if an anomaly goes unnoticed during peak check-in periods, it could be an obstacle to vital guest services such as room availability, resulting in delays and frustrated guests. In such a high-stakes situation, the maximum recall must be prioritized. Failure to identify an anomaly can have significant consequences for service reliability and customer satisfaction.

The Neural Prophet Model, with a recall rate of 63%, comes out as a more reliable option for our purpose. By collecting a broader variety of real anomalies this approach detects possible issues early on, allowing the team to fix them before they become severe service interruptions. Although the model may generate more false positives, this trade-off is desirable in a situation where missing a true anomaly might have serious effects, making the occasional extra research a minor price to pay.

6.3.2 Results on Real Data

Next, we analyze the results of how the model performs in detecting anomalies within the identified seasonal series in Table 6.1. Similar to seasonality detection, the major challenge is the absence of labeled anomalies, as it prevents direct measurement of model performance metrics like precision and recall. Nevertheless, the main objective remained to find potential anomalies in seasonal time series data, as these irregularities could signal system disruptions or performance degradation.

The time series plots in Figure 6.7 visually display the total number of successful API calls, along with model forecasted values, confidence prediction intervals, and identified anomalies across different System-Action combinations. These graphs demonstrate the model's ability to capture both normal seasonal patterns and deviations that might indicate unexpected activity. The shaded area surrounding the prediction line reveals the model's prediction confidence intervals, which span the 15th to 85th percentiles. This range is meant to represent typical data fluctuations, with data points outside of it marked as anomalies, highlighted in red dots. These highlighted points reveal deviations from normal seasonal patterns, which might indicate anomalous system activity.

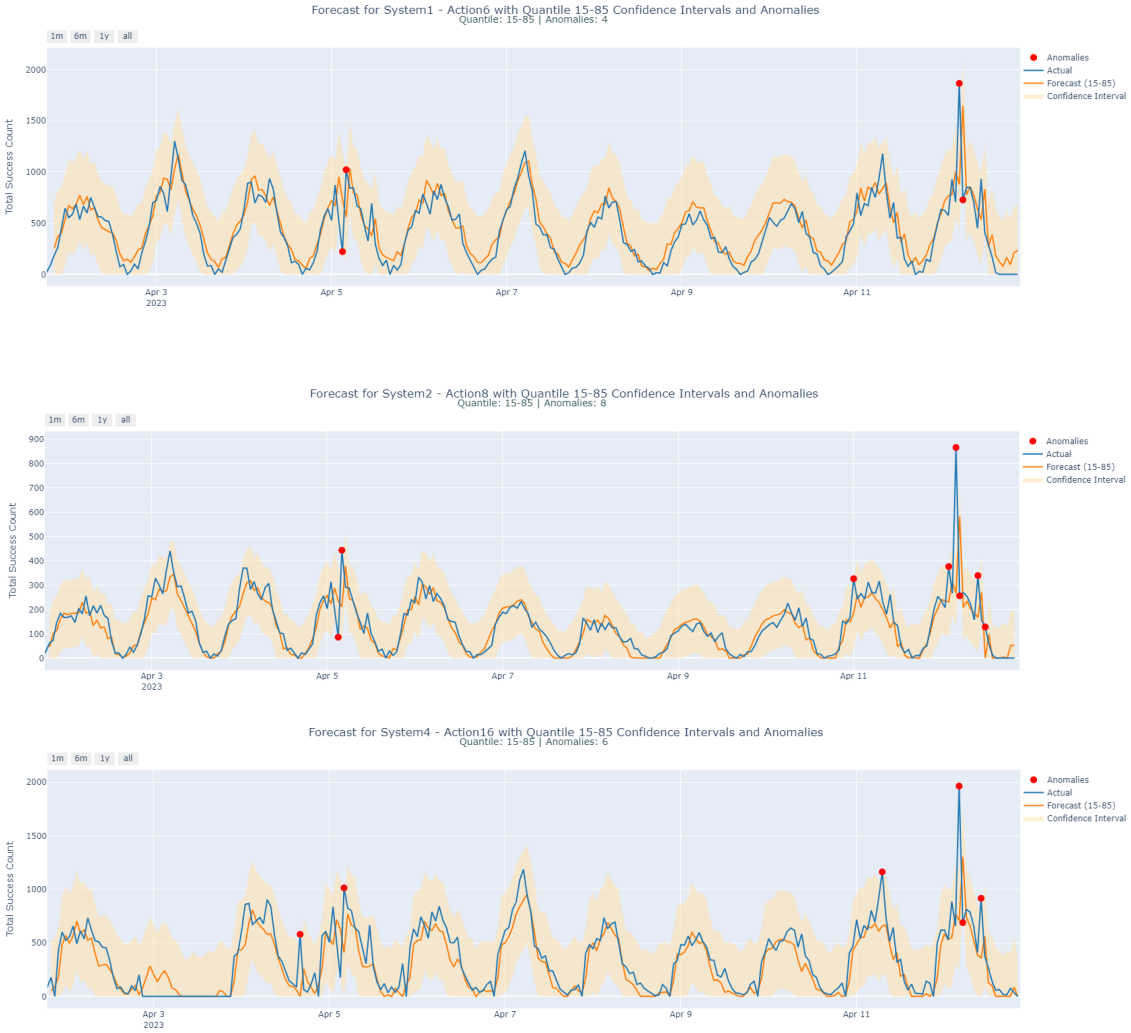

Figure 6.7: Forecasts with Quantile 15-85 Confidence Intervals and Anomalies for different time series

We conducted a thorough evaluation of the model's sensitivity in detecting anomalies by experimenting with various quantile combinations. Each quantile pair establishes the bounds for prediction confidence intervals around the expected value, which determine the threshold for detecting deviations as anomalous. For example, narrow intervals such as $[0.30, 0.70]$ form a tighter border around the forecast, classifying even little deviations as anomalies. Although this increases the model's sensitivity, it also raises the possibility of false positives, since regular oscillations are mistakenly marked as anomalies. In contrast, utilizing broader intervals, such as $[0.05, 0.95]$, gives a larger buffer to accept normal changes, which helps minimize the number of identified anomalies but may overlook tiny, early indicators of system anomalies. It is important to mention that the parameter for the conformal prediction level (α) was consistently set to 90%, ensuring a stable threshold across different interval adjustments.

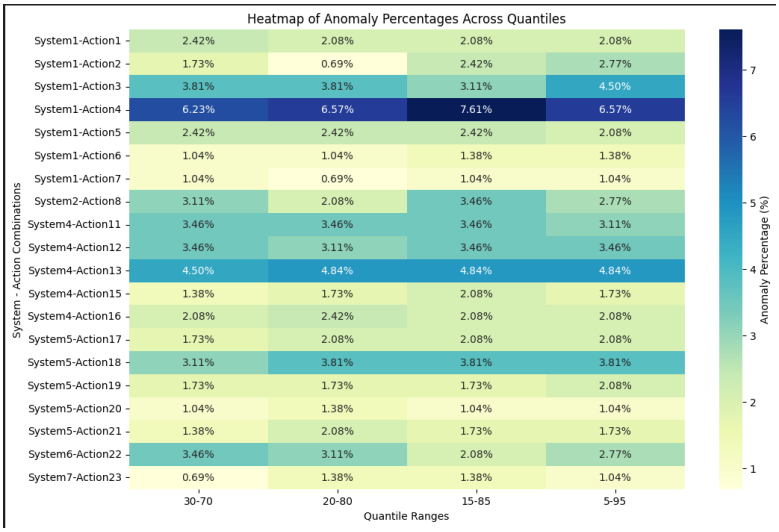

Figure 6.8: Heatmap of Anomaly Percentages Across Quantiles for the seasonal time series.

The heatmap in Figure 6.8 illustrates the percentage of data points marked as anomalies across various quantile ranges for each seasonal time series. Generally, it shows that most combinations have relatively low anomaly percentages, with values frequently between 2-5%. For example, *System1-Action4* regularly exhibits increased anomaly percentages across all quantile ranges, implying that this combination may encounter more anomalies or inherent unpredictability that frequently prompts anomaly detection. On the other side, combinations such as *System7-Action23* have lower anomaly percentages, indicating more consistent behavior with fewer deviations from typical trends.

6.4 Computational Efficiency Analysis

An investigation of the computational efficiency of our suggested framework was used to determine its practicality. Our focus is on two critical components of seasonality and anomaly detection for hospitality IT systems: scalability and computational efficiency. A set of 23 time series were generated and three different datasets with different amounts of data points were used in this investigation. These datasets included 1,444 data points from hourly intervals, 1,920 data points from 45-minute intervals, and 2,888 data points from 30-minute intervals. These datasets were chosen to reflect various temporal resolutions that are frequently encountered in practical applications.

The preprocessing and hyperparameter of all the datasets were examined to ensure optimal performance. The runtimes reported include the total amount of time needed to process to produce the outputs, including both seasonality and anomaly detection.

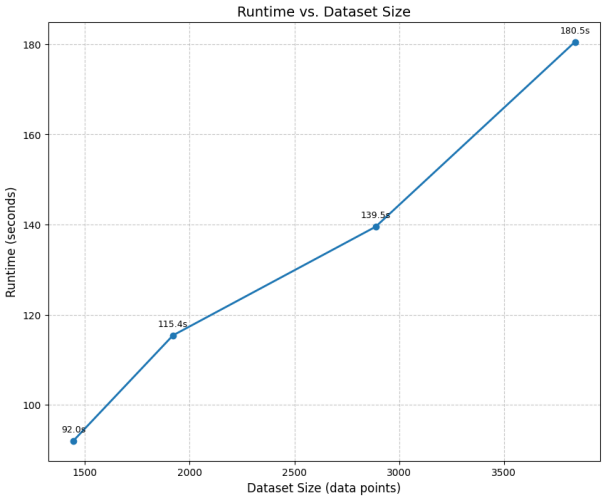

Figure 6.9: Runtime vs. Dataset Size

The framework's scalability was highlighted by the conducting computational research. This process showed a clear correlation between runtime and dataset size. Even for datasets with more data points, the framework consistently produces results in an acceptable amount of time, as seen in Figure 6.9. More specifically, the amount of time taken to perform the seasonality and anomaly detection increases in datasets with more data points, suggesting a constant computing demand as additional data points are added. Because of its effectiveness and scalability, the suggested framework is positioned as a solid and dependable proactive IT management solution for the hotel industry that can meet our operational needs.

7

Conclusion

The essential challenge of identifying anomalies and seasonality in the IT infrastructure for the hospitality industry has been addressed in this research project. The shift from a reactive approach to a proactive, using an automated detection system is an important advancement in an industry where efficiency affects both the customer experience and business success. In this research, the main question lies in how accounting for seasonality in anomaly detection improves both recall and overall performance, especially in time series data. This question was addressed in both theoretical and empirical depth.

Our proposed solution is a two-step approach that first identified and classified seasonal time series data through the Seasonal Quantile Regression (SQR) method, followed by the Neural Prophet method to uncover anomalies. Non-seasonal time series, on the other hand, is still managed using IreckonU's existing moving-average forecasting technique. Such an approach effectively optimizes computational requirements while maintaining accuracy and represents a scalable approach that can efficiently manage high-dimensional datasets.

Key results demonstrate the critical role that seasonality plays in identifying anomalies within time series data, with a significant recall improvement of 64% when seasonality is taken into consideration. Through computational research, we were also able to validate that this framework may be expanded to handle an increased number of data points while maintaining processing efficiency. The framework's goal of explainability further supports its practicality. The system uses a white box model for anomaly detection with a Prediction Confidence Interval (PCI) to generate clear, interpretable outputs that non-technical IT hospitality experts can understand and act upon. The system is not only technically sound but also accessible to a wider audience thanks to the capacity to make adoption easier by explaining model behavior and outputs. The ability to explain model behavior and outputs

fosters trust and facilitates quicker adoption, making the system not just technically robust but also accessible to a broader range of users.

Neural Prophet exhibited a satisfactory recall performance and an average precision-recall trade-off. Its susceptibility to fluctuations, which was exacerbated by the complexity of dynamic high-frequency data, also frequently misclassified regular changes as anomalies. These findings offer some interesting prospects for future studies, particularly related to enhancing algorithms aimed at achieving greater precision.

In the long run, this research will not only improve the technical part of IT system monitoring but also lay the foundation for scalable solutions tailored to the rapidly changing requirements of the hospitality industry. The proposed approach in this work paves the way for increased reliability in operation and customer satisfaction by moving from reactive diagnosis to pro-active avoiding. Scalability, computational efficiency, and explainability are all combined to guarantee that the solutions created are not only unique but also useful, and prepared to handle the industry's next challenges.

8

Limitations and Future Work

Although this study showed encouraging results, there are limitations, which identify areas for improvement in future research. A potential limitation is that seasonality detection is based on a single quantile level. Although it is effective, this technique could miss information, especially if the time series is skewed or there are outlier time series. Future research should explore broader quantile ranges, such as the 15th-85th or 25th-75th percentiles, to gain a better understanding of seasonal trends. In addition, investigating a multi-quantile regression model or treating quantiles individually, may enhance model robustness, which is critical for high-variance datasets.

Yet another obstacle arises in time series with significant variance, where noise may hide seasonality. As a result, accuracy may be decreased in cases when random changes approximate seasonality. Building on possible future work that might use advanced decomposition methods like Seasonal-Trend decomposition and Loess (STL) method to handle seasonal decomposition. Such methods could help to better distinguish noise from actual seasonal components, thereby improving the model's ability to identify periodicity while reducing errors attributed to variance-driven noise.

While our validation of the simulated data involves permutation tests, the model's generalizability is limited due to the inherent limitations of the synthetic data. Synthetic time series data is useful for testing in a familiar context, but it can never truly replicate the complexity, and the unpredictability of datasets we see in reality. Future research is needed to validate this model on real-world datasets from the IT hospitality domain, annotated by domain experts.

The model's ability to detect anomalies exhibits limitations, as seen by its high false positive rate. This emphasizes the necessity of mechanisms that refine precision. Future investigations might utilize a dynamic thresholding approach, in which prediction intervals

are dynamically changed based on the underlying data distribution. Such flexibility would lead to reduced false positives, improved accuracy, and increase the model's resilience across diverse datasets.

This research provides a solid basis, but addressing these drawbacks has profound implications. Future research may result in enhanced, adaptable, and generalizable models, leading to additional improvements in both seasonality and anomaly detection. These initiatives will not only enhance theoretical knowledge but also ensure practical applicability across diverse and real-world datasets.

References

[1] Peter C. Verhoef, Thijs Broekhuizen, Yakov Bart, Abhi Bhattacharya, John Qi Dong, Nicolai Fabian, and Michael Haenlein. **Digital transformation: A multidisciplinary reflection and research agenda**. *Journal of Business Research*, **122**:889–901, 2021. 1

[2] H.M. Moyeenudin, Javed Shaik, Anandan R, and Kumar Narayanan. **Data management with PMS in hotel industry**. *International Journal of Engineering Technology*, **7**:327, 04 2018. 1

[3] We Are Planet. **Property Management System**, Access year. Accessed: insert access date. 1

[4] Arnaldo Sgueglia, Andrea Di Sorbo, Corrado Aaron Visaggio, and Gerardo Canfora. **A systematic literature review of IoT time series anomaly detection solutions**. *Future Generation Computer Systems*, **134**:170–186, 2022. 2

[5] Anna Borucka. **Seasonal Methods of Demand Forecasting in the Supply Chain as Support for the Company's Sustainable Growth**. *Sustainability*, **15**:7399, 04 2023. 2, 5

[6] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. **A survey of anomaly detection techniques in financial domain**. *Future Generation Computer Systems*, **55**:278–288, 2016. 2

[7] Jerome P. Lynch, Charles R. Farrar, and Jennifer E. Michaels. **Structural health monitoring: technological advances to practical implementations [scanning the issue]**. *Proceedings of the IEEE*, **104**(8):1508–1512, 2016. 2

[8] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. **A survey of network anomaly detection techniques**. *Journal of Network and Computer Applications*, **60**:19–31, 2016. 2

[9] Richard Beckman and R. Cook. **Outlier detection**. *Technometrics*, **25**:119–149, 03 2012. 2

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. **Anomaly Detection: A Survey**. *ACM Comput. Surv.*, **41**, 07 2009. 2

[11] Mohammad Braei and Sebastian Wagner. **Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art**. *'Braei'*, 04 2020. 2

[12] Louis Columbus. **53% Of Companies Are Adopting Big Data Analytics**. *Forbes*, 2017. 3

[13] Qin Yu, Lyu Jibin, and Lirui Jiang. **An Improved ARIMA-Based Traffic Anomaly Detection Algorithm for Wireless Sensor Networks**. *International Journal of Distributed Sensor Networks*, **2016**:1–9, 01 2016. 4

[14] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. **LOF: identifying density-based local outliers**. In *ACM SIGMOD Conference*, 2000. 4

[15] Vítor Cerqueira, Luís Torgo, and Carlos Soares. **Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters**. *ArXiv*, **abs/1909.13316**, 2019. 4

[16] Rik van Leeuwen and Ger Koole. **Anomaly detection in univariate time series incorporating active learning**. *Journal of Computational Mathematics and Data Science*, **6**:100072, 2023. 5

[17] G. Enderlein. **Hawkins, D. M.: Identification of Outliers. Chapman and Hall, London – New York 1980, 188 S., £ 14, 50**. *Biometrical Journal*, **29**:198–198, 1987. 7

[18] Mohammad Braei and Sebastian Wagner. **Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art**. *ArXiv*, **abs/2004.00433**, 2020. 7

[19] Iwan Syarif, Adam Prugel-Bennett, and Gary Wills. **Unsupervised clustering approach for network anomaly detection**. In *Networked Digital Technologies: 4th International Conference, NDT 2012, Dubai, UAE, April 24-26, 2012. Proceedings, Part I 4*, pages 135–145. Springer, 2012. 7

[20] Efstathios Kirkos, Charalambos Spathis, and Yannis Manolopoulos. **Support vector machines, Decision Trees and Neural Networks for auditor selection**. *Journal of Computational Methods in Sciences and Engineering*, **8**:213–224, 08 2008. 7

[21] Jiehui Xu. **Anomaly transformer: Time series anomaly detection with association discrepancy**. *arXiv preprint arXiv:2110.02642*, 2021. 8

[22] Paria Shirani, Mohammad Abdollahi Azgomi, and Saed Alrabaee. **A method for intrusion detection in web services based on time series**. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 836–841, 2015. 8

[23] Yangdong He and Jiabao Zhao. **Temporal Convolutional Networks for Anomaly Detection in Time Series**. *Journal of Physics: Conference Series*, **1213**, 2019. 8

[24] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. **DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series**. *IEEE Access*, **7**:1991–2005, 2019. 9

[25] Farzaneh Khoshnevisan, Zhewen Fan, and Vitor R Carvalho. **Improving robustness on seasonality-heavy multivariate time series anomaly detection**. *arXiv preprint arXiv:2007.14254*, 2020. 9

[26] Roger Koenker and Gilbert Bassett Jr. **Regression Quantiles**. *Econometrica: Journal of the Econometric Society*, **46**(1):33–50, 1978. 29

[27] Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Nikolay Laptev, Christoph Bergmeir, and Ram Rajagopal. **NeuralProphet: Explainable Forecasting at Scale**. *CoRR*, **abs/2111.15397**, 2021. 34

[28] Sean J. Taylor and Benjamin Letham. **Prophet: Forecasting at Scale**. `https://facebook.github.io/prophet/`, 2017. Accessed: 2024-11-25. 34

[29] NeuralProphet Team. **Uncertainty Quantification in NeuralProphet**, 2024. Accessed: 2024-11-27. 35

[30] **Time Series Splitting**. 2023. 49

Appendix

System	Action	ActualCount	MissingTimestamps
System1	Action1	7368	9955
System1	Action2	10159	7164
System1	Action3	871	16452
System1	Action4	999	16324
System1	Action5	14589	2734
System1	Action6	14478	2845
System1	Action7	2664	14659
System2	Action8	15375	1948
System4	Action9	456	16867
System4	Action10	2147	15176
System4	Action11	15036	2287
System4	Action12	9343	7980
System4	Action13	9819	7504
System4	Action14	401	16922
System4	Action15	6608	10715
System4	Action16	13652	3671
System5	Action17	6845	10478
System5	Action18	3567	13756
System5	Action19	11331	5992
System5	Action20	7414	9909
System5	Action21	6406	10917
System6	Action22	10516	6807
System7	Action23	5765	11558

Table 8.1: Missing Timestamps for each pair of System-Action. The expected count for all actions is 17,323.

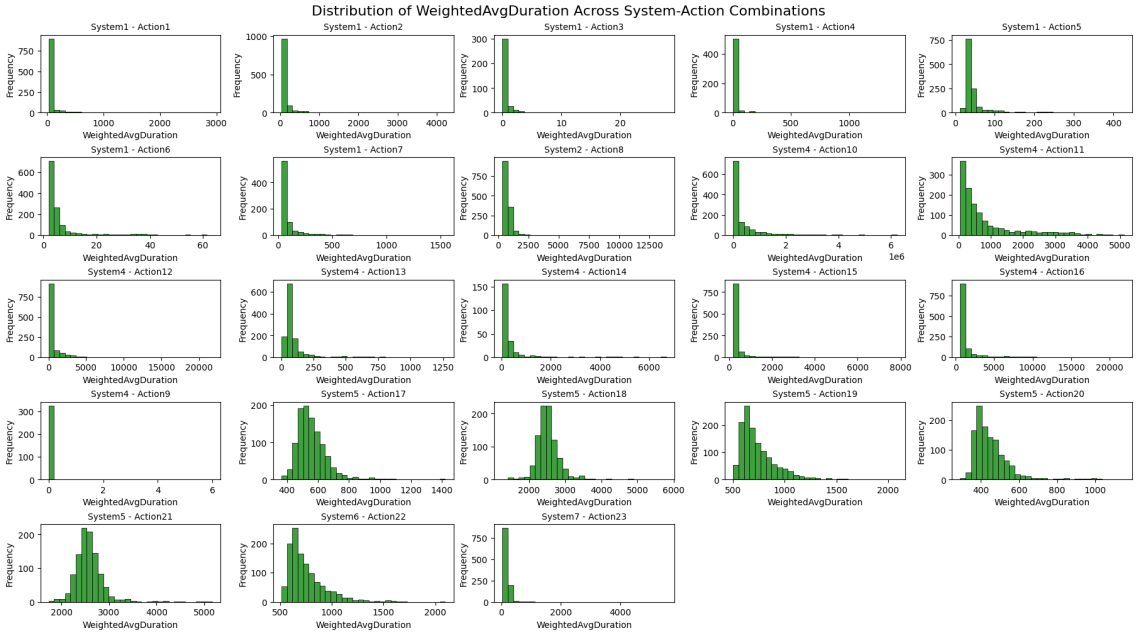

Figure 8.1: Distribution of WeightedAvgDuration across various System-Action combinations

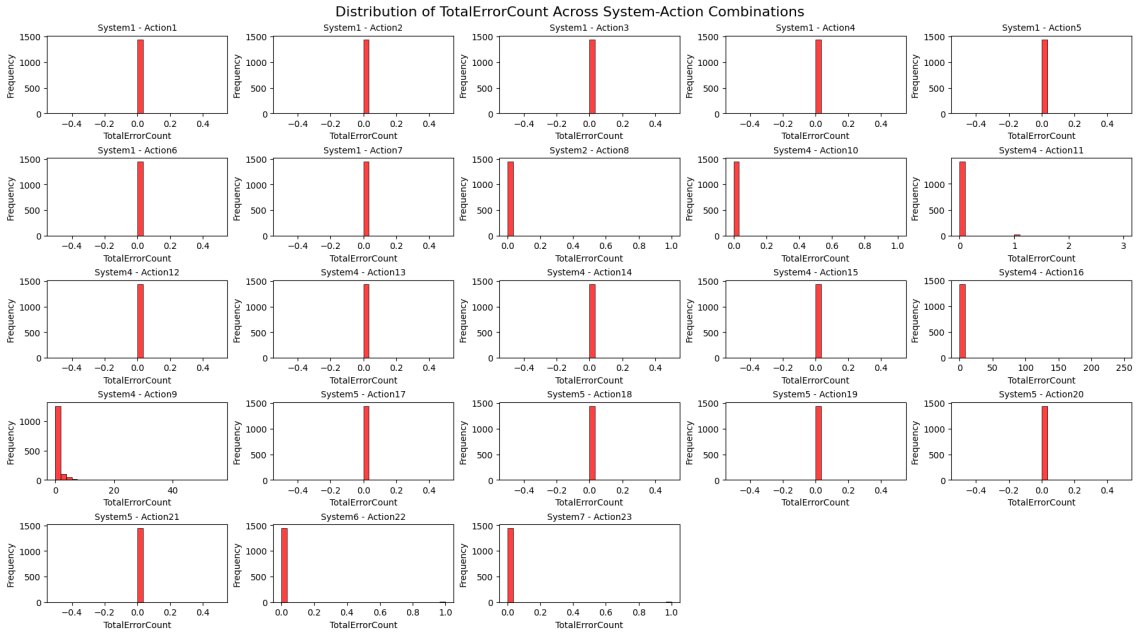

Figure 8.2: Distribution of TotalErrorCount across various System-Action combinations.

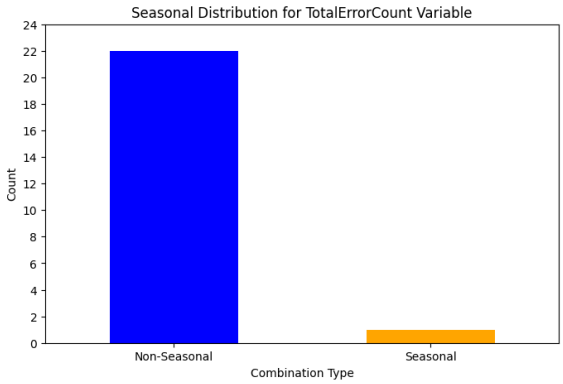

Figure 8.3: Seasonal Distribution for the TotalErrorCount Variable

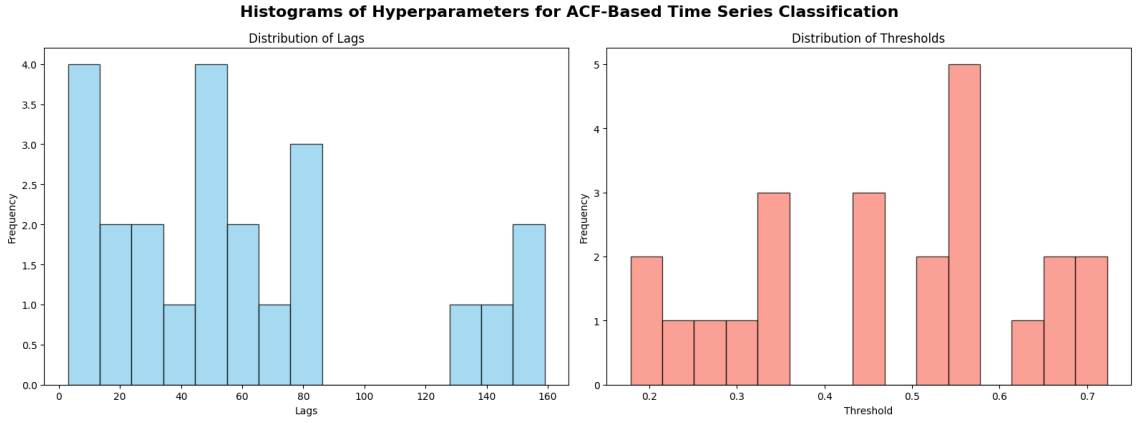

Figure 8.4: Histograms of Hyperparameters: Lags (left) and Threshold (right)

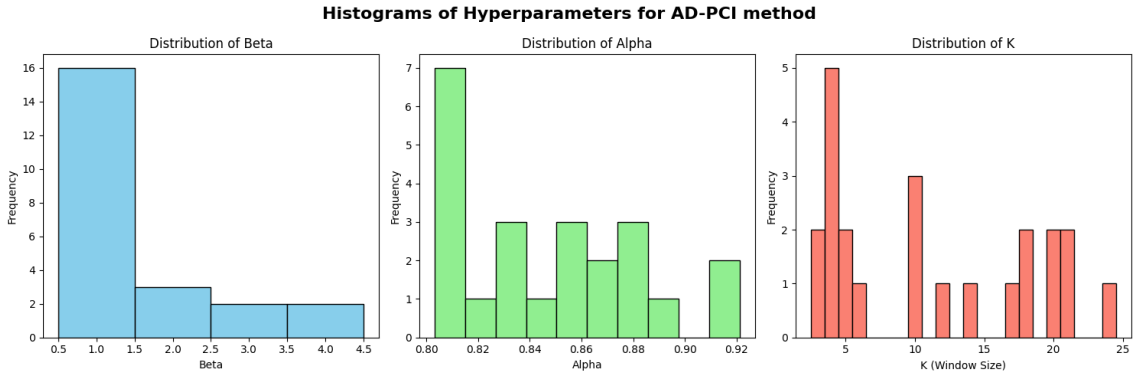

Figure 8.5: Histograms of Hyperparameters: Beta (left), Alpha (middle), and K (right) for AD-PCI Method.

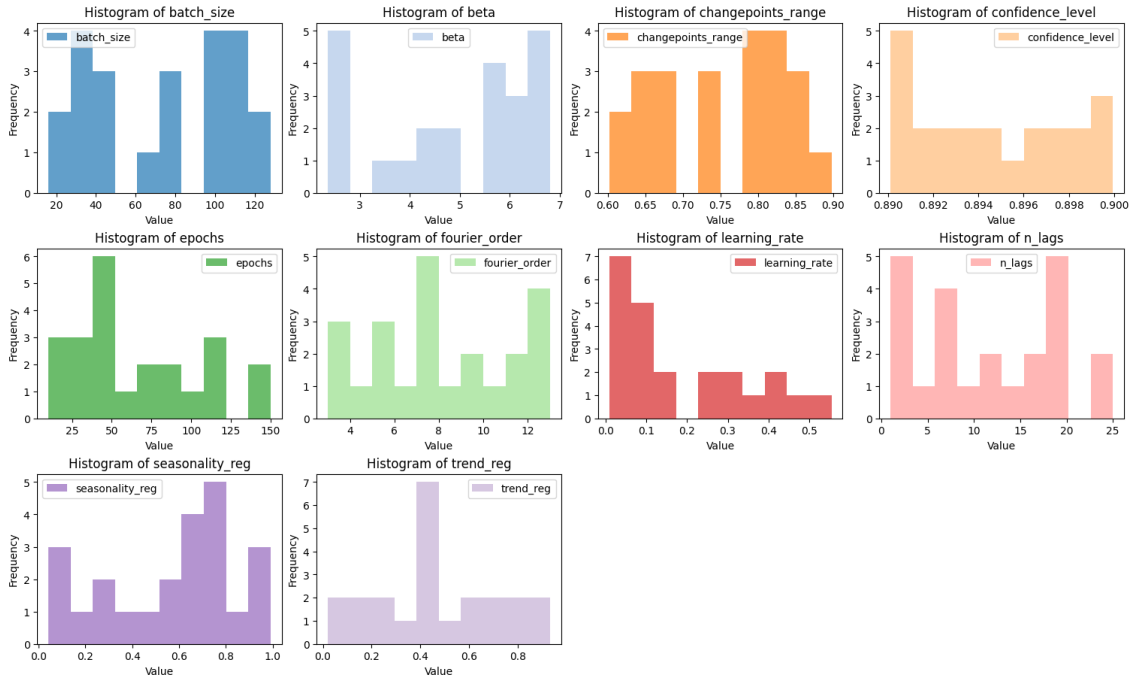

Figure 8.6: Histograms of Hyperparameters for the Neural Prophet model

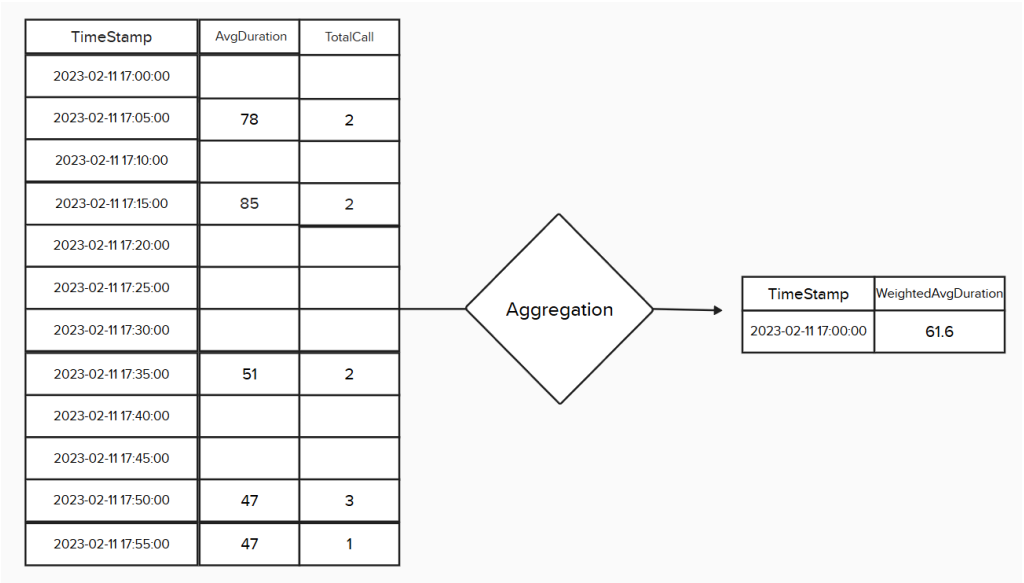

Figure 8.7: Aggregation process for calculating Weighted Average Duration.

System-Action	Avg_Train_Loss	Avg_Validation_Loss	Perceptual Difference (%)
System1-Action1	19.67	20.00	1.66
System1-Action2	13.44	15.63	16.29
System1-Action3	18.40	16.67	-9.41
System1-Action4	0.35	0.38	7.40
System1-Action5	40.56	52.37	29.11
System1-Action6	127.67	150.34	17.75
System1-Action7	1.46	1.35	-7.51
System2-Action8	38.67	48.84	26.31
System4-Action10	0.74	0.93	24.26
System4-Action11	619.34	622.30	0.48
System4-Action12	10.88	14.74	35.44
System4-Action13	5.20	6.46	24.09
System4-Action14	0.42	0.365	-13.16
System4-Action15	2.93	3.59	22.41
System4-Action16	110.16	134.63	22.22
System4-Action9	0.00	0.00	0.00
System5-Action17	2.43	3.42	40.40
System5-Action18	1.17	1.37	17.16
System5-Action19	5.64	7.19	27.51
System5-Action20	2.27	2.99	31.35
System5-Action21	2.50	2.84	13.62
System6-Action22	6.03	7.89	30.90
System7-Action23	1.95	2.20	12.82

Table 8.2: Seasonal Quantile Regression Results for Each Time Series with Perceptual Difference between Training and Validation Loss

List of Figures

1.1 Types of Anomalies 3

3.1 Dataset snapshot 11
3.2 Line Graphs of TotalSuccessCount for 5-min, 30min, 1-Hour, and 1-Day Datasets (System6-Action22) 15
3.3 Line Graphs of WeightedAvgDuration for 5-min, 30-min, 1-Hour, and 1-Day Datasets (System6-Action22) 15
3.4 Boxplots of TotalSuccessCount and WeightedAvgDuration Across Systems for 30min, 1-Hour, and 1-Day Datasets on log Scale 16
3.5 Total Success Count box plots by System 18
3.6 Distribution of TotalSuccessCount by System-Action Combination 19
3.7 Distribution fits for TotalSuccessCount showing the best fit distributions for System1 - Action6 (Gaussian) and System6 - Action22 (Gaussian). 20
3.8 Distribution fits for WeightedAvgDuration showing the best fit distributions for System1 - Action2 (LogNormal) and System5 - Action18 (LogNormal). . 21

4.1 Flowchart of the Time Series Classification Process 32
4.2 Workflow of the Neural Prophet model 36

6.1 Visual comparison of time series data from simulated and real dataset . . . 41
6.2 Distribution of Permuted Test Statistics for Non-Anomalous Points. 44
6.3 Confusion Matrix for ACF method 46
6.4 Confusion Matrices for Seasonal Quantile Regression at different time periods 46
6.5 ROC Curve for Balanced time series distribution 47
6.6 Misclassified Series 47
6.7 Forecasts with Quantile 15-85 Confidence Intervals and Anomalies for different time series 52

6.8 Heatmap of Anomaly Percentages Across Quantiles for the seasonal time series. 53
6.9 Runtime vs. Dataset Size 54

8.1 Distribution of WeightedAvgDuration across various System-Action combinations 63
8.2 Distribution of TotalErrorCount across various System-Action combinations. 63
8.3 Seasonal Distribution for the TotalErrorCount Variable 64
8.4 Histograms of Hyperparameters: Lags (left) and Threshold (right) 64
8.5 Histograms of Hyperparameters: Beta (left), Alpha (middle), and K (right) for AD-PCI Method. 64
8.6 Histograms of Hyperparameters for the Neural Prophet model 65
8.7 Aggregation process for calculating Weighted Average Duration. 65

List of Tables

3.1 Kolmogorov-Smirnov (KS) Statistics for Best-Fitted Distributions 20

6.1 Classification Results for Seasonal & Non-Seasonal Time Series for Total-SuccessCount variable. 48
6.2 Comparison of Model Performance on Seasonal Synthetic Data 51

8.1 Missing Timestamps for each pair of System-Action. The expected count for all actions is 17,323. 62
8.2 Seasonal Quantile Regression Results for Each Time Series with Perceptual Difference between Training and Validation Loss 66