NPS-AM-26-271

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Forecasting Intermittent Demand for Aircraft Spare Parts using Machine Learning

December 2025

Capt Allan Goncalves Almeida, Brazillian Air Force

Thesis Advisors: Dr. Geraldo Ferrer, Professor
Dr. Eva Regnier, Professor

Department of Acquisition, Finance and Manpower

Naval Postgraduate School

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

ACQUISITION RESEARCH PROGRAM
DEPARTMENT OF ACQUISITION, FINANCE AND MANPPOWER
NAVAL POSTGRADUATE SCHOOL

The research presented in this report was supported by the Acquisition Research Program of the Department of Acquisition, Finance and Manpower Naval Postgraduate School.

To request defense acquisition research, to become a research sponsor, or to print additional copies of reports, please contact the Acquisition Research Program (ARP) via email, arp@nps.edu or at 831-656-3793.

ACQUISITION RESEARCH PROGRAM
DEPARTMENT OF ACQUISITION, FINANCE AND MANPPOWER
NAVAL POSTGRADUATE SCHOOL

ABSTRACT

Intermittent demand poses a complex challenge for military aviation logistics due to long periods of zero consumption, followed by sudden peaks of demand, which makes traditional forecasting methods unreliable. This thesis develops and evaluates machine-learning approaches for forecasting intermittent aircraft spare parts demand within the Brazilian Air Force (FAB), with the goal of improving prediction accuracy and, consequently, enhancing readiness levels and reducing stockouts.

The study gathers and preprocesses 10 years of historical spare-parts demand data from the T-27 (EMB-312) TUCANO aircraft fleets and applies different forecasting models, including Moving Average, Simple Exponential Smoothing, and Croston's method as traditional time-series baselines, and gradient boosted decision trees (XGBoost), and artificial neural networks as machine learning models. Accuracy is assessed using MASE as the primary metric, complemented by error distributions.

Results demonstrate that machine learning models, particularly XGBoost combined with engineered features, achieve significant gains over classical methods in forecasting accuracy. The findings provide a replicable framework for modernizing FAB's spare parts planning process and highlight opportunities for broader adoption of advanced analytics in defense logistics.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank the person without whom none of this would have been possible: my wife, Desirêe. Thank you for agreeing to embark on this journey with me, moving to a new country, embracing a new culture and language, and still supporting me in every possible way. I love you.

To my family, who always believed in me and, through their confidence, helped me believe that I was capable of achieving this goal.

I am grateful to the Brazilian Air Force for trusting me to represent our country. To PAMA LS, for providing all the data necessary for the development of this thesis. To ILA, for their confidence and partnership. I hope to repay this investment through the knowledge gained during my master's studies, contributing in the years ahead to a modern and exemplary Air Force.

To my advisors, Dr. Geraldo Ferrer and Dr. Eva Regnier, who guided me so patiently throughout my stay in the United States: The lessons I learned from you extended far beyond this thesis; they have shaped me both professionally and personally. You can always count on me, and I hope to meet you again soon in Brazil.

To NPS, for offering exceptional academic instruction and for the constant support extended to me and my family. If we felt welcomed in Monterey and had such a positive experience, it was thanks to the dedication and kindness of everyone here.

Finally, but never least, I thank God for blessing our family throughout every stage of our lives. I give thanks for each person mentioned here and for everything the Lord has provided us.

THIS PAGE INTENTIONALLY LEFT BLANK

NPS-AM-26-271

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Forecasting Intermittent Demand for Aircraft Spare Parts using Machine Learning

December 2025

Capt Allan Goncalves Almeida, Brazillian Air Force

Thesis Advisors: Dr. Geraldo Ferrer, Professor
Dr. Eva Regnier, Professor

Department of Acquisition, Finance and Manpower

Naval Postgraduate School

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

ACQUISITION RESEARCH PROGRAM
DEPARTMENT OF ACQUISITION, FINANCE AND MANPPOWER
NAVAL POSTGRADUATE SCHOOL

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
B. THE BRAZILIAN AIR FORCE CONTEXT 3
C. PROBLEM STATEMENT 4
D. RESEARCH QUESTION AND OBJECTIVES 5
E. BENEFITS AND CONTRIBUTIONS 7
F. ORGANIZATION OF THE STUDY 8

II. LITERATURE REVIEW 11
A. FORECAST 11
1. Qualitative vs. Quantitative 11
2. Time-series 12
3. Intermittent demand 15
4. Univariate vs. Multivariate 17
B. MACHINE LEARNING 18
1. Supervised vs. Unsupervised learning 18
2. Application in forecasting models 19

III. METHODOLOGY 21
A. RESEARCH DESIGN 21
B. DATA DESCRIPTION 21
C. DEMAND CATEGORIZATION 22
D. DATA TRIMMING EARLY OF ZERO-DEMAND MODELS 23
E. FORECASTING MODELS 24
1. Traditional Forecasting Methods 24
2. Machine learning methods 26
3. Comparative considerations 30
F. FEATURE ENGINEERING FOR MACHINE LEARNING MODELS 30
1. Lag features 31
2. Demand activity indicators 32
3. Temporal and calendar features 32
4. Zero sequences 32
5. Encoding 33
G. MODEL TRAINING AND VALIDATION 33
1. Rolling validation 33

2. Data splitting 34
3. Training procedure 34
H. EVALUATION METRICS 36
1. Scale-dependent metrics 37
2. Scale-free error metric 38
3. Evaluation framework 39
I. TOOLS AND IMPLEMENTATION 39
1. Python 39
2. Microsoft Excel 40
3. Computational workflow and reproducibility 40

IV. RESULTS AND ANALYSIS 43
A. OVERVIEW 43
B. STATISTICAL ACCURACY COMPARISON 43
C. PATTERNS IN FORECAST ERRORS 48
1. Moving average 51
2. Simple exponential smoothing 54
3. Croston Syntetos-Boylan Approximation (Croston SBA) 57
4. Artificial Neural Network (NN) 59
5. XGBoost 62
6. Model Comparison 65
D. SUMMARY 68

V. CONCLUSION 71
A. SUMMARY OF FINDINGS 71
B. THEORETICAL CONTRIBUTIONS 71
C. OPERATIONAL IMPLICATIONS 72
1. Impact on Stockouts and Readiness 72
2. Impact on Inventory Costs 72
D. MANAGERIAL AND POLICY IMPLICATIONS 72
1. Forecast Integration 72
2. Decision Support for Procurement 73
3. Scalable Frameworks for Other Fleets 73
E. LIMITATIONS AND FUTURE RESEARCH 73
F. FINAL REMARKS 74

APPENDIX A. FILLING MONTH ROWS CODE 75

APPENDIX B. FILTERING INTERMITTENT PATTERN CODE 79

APPENDIX C. MOVING AVERAGE CODE 85

APPENDIX D. SIMPLE EXPONENTIAL SMOOTHING CODE 87

APPENDIX E. CROSTON SBA CODE 89

APPENDIX F. NEURAL NETWORK CODE..... 91

APPENDIX G. XGBOOST CODE 95

APPENDIX H. ERROR MEASURES (MAE, RMSE AND MASE) CODE..... 105

LIST OF REFERENCES..... 109

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Intermittent vs. Regular pattern 8

Figure 2. Horizontal Pattern. Adapted from Makridakis et al. (1983). 13

Figure 3. Seasonal Pattern. Adapted from Makridakis et al. (1983). 14

Figure 4. Cyclical Pattern. Adapted from Makridakis et al. (1983). 14

Figure 5. Trend Pattern. Adapted from Makridakis et al. (1983). 15

Figure 6. Intermittent demand pattern. 16

Figure 7. Decision tree example 27

Figure 8. XGBoost Overview 28

Figure 9. Neural Network diagram 30

Figure 10. Rolling validation diagram 33

Figure 11. Distribution of the number of months per part number (PN) in which at least one forecasting model produced a valid forecast. 44

Figure 12. PN A – Rare extreme Spike 48

Figure 13. PN B – Intermittent bursts 49

Figure 14. PN C – lightly intermittent demand 49

Figure 15. PN D – High-variance series 50

Figure 16. PN E – Declining demand 50

Figure 17. Moving average forecasting for demand with rare extreme spikes 51

Figure 18. Moving average forecasting for demand with intermittent burst pattern 52

Figure 19. Moving average forecasting for demand with lightly intermittent pattern 52

Figure 20. Moving average forecasting for demand with high variance 53

Figure 21. Moving average forecasting for declining demand 53

Figure 22. Simple exponential smoothing forecasting for demand with rare extreme spikes 54
Figure 23. Simple exponential smoothing forecasting for demand with intermittent burst pattern 55
Figure 24. Simple exponential smoothing forecasting for demand with lightly intermittent pattern 55
Figure 25. Simple exponential smoothing forecasting for demand with high variance 56
Figure 26. Simple exponential smoothing forecasting for demand with declining demand 56
Figure 27. Croston SBA forecasting for demand with rare extreme spikes 57
Figure 28. Croston SBA forecasting for demand with intermittent bursts pattern 58
Figure 29. Croston SBA forecasting for demand with lightly intermittent pattern 58
Figure 30. Croston SBA forecasting for demand with high variance 59
Figure 31. Croston SBA forecasting for declining demand 59
Figure 32. NN forecasting for demand with rare extreme spikes 60
Figure 33. NN forecasting for demand with intermittent bursts pattern 60
Figure 34. NN forecasting for demand with lightly intermittent pattern 61
Figure 35. NN forecasting for demand with high variance 61
Figure 36. NN forecasting for declining demand 62
Figure 37. XGBoost forecasting for demand with rare extreme spikes 63
Figure 38. XGBoost forecasting for demand with intermittent bursts pattern 63
Figure 39. XGBoost forecasting for demand with lightly intermittent pattern 64
Figure 40. XGBoost forecasting for demand with high variance 64
Figure 41. XGBoost forecasting for declining demand 65
Figure 42. Models forecasting comparison for demand with rare extreme spikes 66
Figure 43. Models forecasting comparison for intermittent bursts pattern 67

Figure 44. Models forecasting comparison for demand with lightly intermittent pattern 67

Figure 45. Models forecasting comparison for demand with high variance 68

Figure 46. Models forecasting comparison for declining demand 68

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Model Performance Comparison (2,428 PNs)..... 45

Table 2. Individual performance comparison on individual PNs 46

Table 3. Individual performance comparison by demand volume..... 46

Table 4. MASE dispersion across all PNs by model 47

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AFA	Academia da Força Aérea / Brazilian Air Force Academy
AI	Artificial Intelligence
CV^2	Squared Coefficient of Variation
DoD	Department of Defense
ERP	Enterprise Resource Planning
FAB	Força Aérea Brasileira / Brazilian Air Force
GAO	Government Accountability Office
MA	Moving Average
MAE	Mean Absolute Error
MASE	Mean Absolute Scaled Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
NN	Neural Network
PN	Part Number
RMSE	Root Mean Squared Error
SBA	Syntetos–Boylan Approximation
SES	Simple Exponential Smoothing
SILOMS	Sistema Integrado de Logística de Material e Serviços (Materiel and Service Logistics System)
SMA	Simple Moving Average
T-27	Embraer EMB-312 Tucano aircraft model
XGBoost	Extreme Gradient Boosting

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis evaluates forecasting models for aircraft spare parts demand within the Brazilian Air Force (FAB), focusing on the T-27 Tucano fleet as a representative platform. Accurate demand forecasting is essential for maintaining aircraft readiness, preventing stockouts, and reducing the dependence on emergency procurement processes. However, spare-parts demand for military aircraft typically exhibits intermittent patterns, i.e., long sequences of zero demand interrupted by sudden demand, which makes forecasting challenging and reduces the accuracy of traditional statistical models.

The study analyzes a dataset of ten years of monthly demand for 2,428 part numbers (PNs), representing over 250,000 observations. The following forecasting models were implemented: Moving Average with a 24-month window, Simple Exponential Smoothing (SES), Croston's method with Syntetos-Boylan approximation (Croston-SBA) for intermittent series, a neural network, and an XGBoost gradient-boosted model. All models were compared using error measures such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and especially Mean Absolute Scaled Error (MASE), the primary accuracy metric due to its suitability for intermittent data.

Across the entire dataset, XGBoost demonstrated the strongest overall performance, achieving the lowest median MASE. This indicates not only higher accuracy but also more consistent results across parts with different demand levels. Among statistical methods, SES performed the best but remained close to the naïve benchmark. The 24-month moving average, currently used by FAB, generated stable yet slow-reacting forecasts, while Croston-SBA surprisingly struggled with long zero-demand sequences. The neural network underperformed relative to the other methods due to limited data per series and the difficulty of learning patterns from sparse demand.

Representative PNs were selected to illustrate common demand behaviors, such as rare spikes, intermittent bursts, lightly intermittent demand, high variance demand, and declining demand patterns, and the way each model forecasted each of the common behaviors was analyzed. Classical methods cannot anticipate spikes because they rely only

on historical data for a single part, causing delayed and smoothed responses. In contrast, XGBoost occasionally predicts spikes in advance because it learns nonlinear relationships across PNs, allowing it to capture structural patterns in the dataset. This ability to incorporate information about other PNs explains why XGBoost is the only method capable of anticipating future bursts.

The managerial implications for FAB are significant, given that XGBoost can be integrated into existing logistics systems as a monthly forecasting module and that improved forecast accuracy supports more efficient procurement planning and reduces the risks associated with stockouts, which can result in grounded aircraft and lower readiness levels. Moreover, the modeling framework applied is flexible and can also be adapted to platforms with more regular demand patterns.

Overall, the study shows that using machine learning in forecasting, especially XGBoost, offers FAB a practical and scalable way to improve inventory planning. XGBoost delivers meaningful forecasting gains in an environment characterized by irregular, intermittent demand, enabling stronger decision-making across supply and maintenance planning.

I. INTRODUCTION

A. BACKGROUND

Since the beginning of human civilization, logistics has proven to be one of the decisive factors for competitive advantage over adversaries. From Alexander the Great to Napoleon to Operation Desert Storm in the Gulf War, impeccable logistics fundamentals were decisive to favorable outcomes (Van Creveld, 1977). Logisticians have focused on accurately predicting future demand as one of the most important weapons for victory.

As time passed, scholars, mathematicians, statisticians, entrepreneurs, businessmen, and others understood that this capacity is not restricted to warfare but to all operations (Van Creveld, 1977). This resulted in extensive research around the topic and the creation of new forecasting methodologies. Quantitative methods started to stand out over qualitative methods due to their lack of bias and reliance on real data. Univariate forecasting, which relies on historical data of a single variable, evolved from analyzing only the past demand to considering the trend and seasonality of the demand pattern. After that, the creation of multivariate forecasting came to fulfill the needs of more robust and accurate forecasting, now considering multiple variables and their interdependency (Makridrakis et al., 1983).

Military organizations have historically adapted commercial forecasting models to suit their operational needs. However, the characteristics of defense logistics are highly irregular demand, expensive parts, and extensive lead times, meaning that methods optimized for usual markets often fail to achieve good results in the military environment.

Therefore, the industry started using different techniques, such as moving averages, exponential smoothing, regression, and others, to solve problems, resulting in improved forecast accuracy. However, even though there has been great development throughout the years, the significant number of different markets with different needs and behaviors prevents the creation of a specific forecast technique for each specific situation.

While commercial sectors benefited from advances in forecasting techniques, military logistics continued to stand out as a more complex domain, with high uncertainty, gigantic consequences, one where accurate forecasting plays an essential role.

Intermittent demand is defined as a time series where demand spikes occur, followed by periods with zero demand. Usually, these spikes happen with high variability and unpredictable patterns. This pattern is particularly common in aviation maintenance, where failures are often event-driven or condition-based. Components such as brake pads, hydraulic actuators, and avionic modules, whose failure is related to usage instead of time-based maintenance, commonly present intermittent behavior. The irregularity violates key assumptions in classical univariate and multivariate quantitative forecasting methods, making accurate prediction and inventory optimization extremely difficult.

One of the most critical challenges in military logistics is accurately predicting the demand for spare parts. This capability is fundamental because the ultimate objective is to ensure that each component is available when needed, guaranteeing mission readiness and operational capability while still keeping costs under control. The operational impact of poor forecasting has been documented in real conflict scenarios. During the early stages of Operation Iraqi Freedom, for example, U.S. Army units faced lower levels of operational readiness due to unavailable parts, revealing a disconnection between forecasting models and real-time logistical needs (Government Accountability Office [GAO], 2003).

Similarly, forecasting failures affect air forces, and they are significantly more costly than ground-based fleets. Unlike them, aircraft systems have high interdependency and require precise part availability to ensure safety and performance (Blanchard & Fabrycky, 2023). Even a small missing component can ground an aircraft, delaying missions or training and impacting overall operational readiness. Therefore, accurate demand forecasting is not just a logistical function but a strategic enabler.

Researchers have proposed dedicated models for intermittent demand, such as Croston's method (Croston, 1972), which addresses this pattern by splitting the forecast into two parts: the average demand size when it occurs and the average time interval between events occurrences. While it performs better than simple averages in several

contexts, it is limited by its inability to incorporate external factors or the relationship between similar parts.

In recent years, with the growing availability of computational power and data, it has become possible to use models that perform advanced mathematical calculations and that are relatively easy to implement. Machine learning techniques have become more available every day and offer an opportunity to revisit and potentially enhance some forecast accuracy. However, their utilization and effectiveness in the context of military intermittent demand remain underexplored.

B. THE BRAZILIAN AIR FORCE CONTEXT

At the beginning of the 20th Century, the world was fascinated by the advances in the aviation area, and following this trend, the Air Force Ministry was created on January 20th, 1941 (Brazilian Air Force, 2025). Approximately 4 years later, acknowledging the essential role of logistics for efficiency and operationality, on August 23rd, 1945, the Air Force Quartermaster Service was created as the logistics core of the Brazilian Air Force (FAB) (Brazilian Air Force, 2024).

During the following years, the Brazilian government invested in the aviation industry growth with the creation of the Air Force Institute of Technology (ITA) in 1950, with the focus of training aeronautical engineers, which resulted in the development of the Bandeirante aircraft and its first flight in 1968, followed by the creation of the Brazilian Aeronautical Company (Embraer) in 1969.

In the 1980s, Embraer created the EMB-312 Tucano as a training turboprop, offering comparable performance at a lower cost than contemporary jet models. (Embraer, 2025). Those characteristics helped the Brazilian Air Force to select the aircraft in 1983, naming it the T-27 Tucano, and using it for training the pilot cadets in the Brazilian Air Force Academy (AFA) (Brazilian Air Force, 2023). Alongside the operation of this aircraft, FAB understood that it was important to rely on a logistics system that could control inventory, spare parts management, transportation, and procurement, resulting in the adoption of a few logistics information systems, with SILOMS being the one used today.

The Materiel and Service Logistics System (SILOMS) is the Enterprise Resource Planning (ERP) software used for all logistics operations in FAB. An ERP is a software solution that focuses on integrating the complete range of a business's processes and functions to provide a comprehensive view of the organization from a single information and IT architecture (Klaus et al., 2000).

The SILOMS creation was an essential step for FAB, adding the capability of aggregating all inventory, spare part management, manufacturing, transportation, and procurement in the same place, providing the proper visualization of all data and permitting data-driven decision making. However, some branches inside the system were not correctly updated to match the fluid behavior of logistics through time, resulting in a disconnection with modern methods used in research and in the industry.

For example, it was possible to notice that FAB was facing intermittent demand behavior with its aircraft spare parts demand. An important aircraft, like the T-27 Tucano, used exclusively for the training of new military pilots, has thousands of spare parts needed for its maintenance, and many of them have intermittent demand. However, FAB relied on simple forecasting techniques like the Simple Moving Average to foresee future demand values. While this technique is easy to implement, it struggles with the complexity of the intermittent demand, resulting in frequent shortages, procurement inefficiency, and lower levels of mission readiness.

C. PROBLEM STATEMENT

An accurate forecast is a determinant factor for the performance of an organization. As recognized by Boylan and Syntentos (2021), the application of inaccurate forecast methods leads to either excess or shortages in stock, depending on the direction of the forecast error. Excess inventory results in high holding costs and potential obsolescence, while shortages compromise service levels and mission readiness, particularly critical in defense operations where logistical failures may delay operations or compromise safety.

In military logistics, this challenge becomes more complex due to the behavior of demand for certain items, especially aircraft spare parts. These items frequently exhibit intermittent demand, characterized by irregular timing and variable quantities of requisition, often with many zero-demand periods interspersed with sporadic spikes. Croston (1972) first introduced a method to deal with such demand types, recognizing that traditional time series techniques were not designed to handle the discontinuity and sparsity present in this context.

Despite decades of research and several enhancements to Croston's method, such as the Syntetos-Boylan Approximation (SBA) (Xu et al., 2012), forecasting of intermittent demand remains a complex challenge. As Hyndman and Athanasopoulos (2018) note, the assumptions underlying most statistical forecasting models are not met when applied to zero-inflated, non-seasonal data with irregular intervals between demand occurrences.

In recent years, machine learning (ML) techniques have shown promise in modeling complex, nonlinear relationships in forecasting problems. Methods such as gradient boosting (e.g., XGBoost) and neural networks (e.g., Multi-Layer Perceptron) have demonstrated strong performance in retail and industrial contexts, especially where traditional time series models fall short (Salinas et al., 2020). However, their application to military logistics, and specifically to intermittent demand forecasting in air forces, remains limited in the academic literature.

Thus, a clear research gap exists at the intersection of modern machine learning approaches and the practical forecasting challenges faced by military organizations. The lack of empirical studies validating ML models against intermittent demand in operational defense environments limits both theoretical advancement and practical innovation. This thesis addresses that gap by evaluating whether machine learning methods can improve forecast accuracy for critical aircraft spare parts with intermittent demand in the Brazilian Air Force, compared to traditional statistical methods

D. RESEARCH QUESTION AND OBJECTIVES

Forecasting accuracy is a key determinant of logistical efficiency, especially in military operations where equipment availability and mission success are often constrained

by resource planning. With the increasing availability of computational power and large datasets, more advanced techniques such as machine learning have emerged as viable alternatives to traditional forecasting approaches. Boylan et al. (2021) highlight that computationally intensive models, including neural networks, can offer promising results in the context of intermittent demand, given their ability to adapt to complex, data-driven patterns.

Despite growing interest in applying artificial intelligence and machine learning to logistics, most academic studies focus on commercial applications, such as retail inventory, e-commerce, and manufacturing. Most models are tested on regular or seasonal demand patterns, not on intermittent, zero-inflated time series, which leaves a critical research gap. Far fewer studies explore the unique constraints of defense systems.

This thesis is driven by the central question of whether machine learning models can improve the forecasting accuracy of aircraft spare parts with intermittent demand in the Brazilian Air Force. The study aims to determine which machine learning techniques are most appropriate for this type of demand and to compare their forecasting performance with that of traditional statistical models such as moving averages and exponential smoothing. The study also examines the practical limitations and data requirements for applying machine learning in a defense logistics context, including model interpretability and operational feasibility.

The main goal of this study is to assess and compare traditional and machine learning forecasting methods using historical demand data from the Brazilian Air Force. The research aims to determine whether machine learning can achieve significant improvements in forecast accuracy for spare parts with intermittent demand by applying these techniques in a real military scenario. In doing so, it also aims to assess the potential applicability of these models to support more effective decision-making in military logistics environments.

Given that machine learning models, such as decision trees and neural networks, can identify nonlinear relationships between inputs and outcomes, and some models like XGBoost can learn patterns across multiple related series, their application to intermittent

demand spare part forecasting is appropriate. However, these methods require careful data preparation, tuning, and validation, especially when dealing with zero-heavy sequences.

Therefore, this thesis investigates whether machine learning methods can outperform traditional statistical approaches in forecasting the intermittent demand of aircraft spare parts, using the Brazilian Air Force as a case study. It presents a detailed review of the forecasting literature, describes the methodology adopted for model selection and evaluation, and analyzes the comparative results of machine learning and statistical models using real demand data from the Brazilian Air Force. Finally, the thesis concludes with a discussion on practical implications and recommendations for enhancing inventory management through improved forecasting.

E. BENEFITS AND CONTRIBUTIONS

This study contributes both theoretically and practically to the fields of forecasting and defense logistics. From a theoretical standpoint, it addresses a well-documented but underexplored forecasting challenge: intermittent demand. Figure 1 illustrates how complex an intermittent demand pattern can be compared to a regular demand pattern. While previous research has proposed statistical techniques such as Croston's method and its variants, the application of modern machine learning models to this specific type of demand remains limited, especially in military contexts. By applying these models to zero-inflated demand data from the Brazilian Air Force, the study expands the academic understanding of how machine learning can be used to improve forecast accuracy for complex, irregular time series.

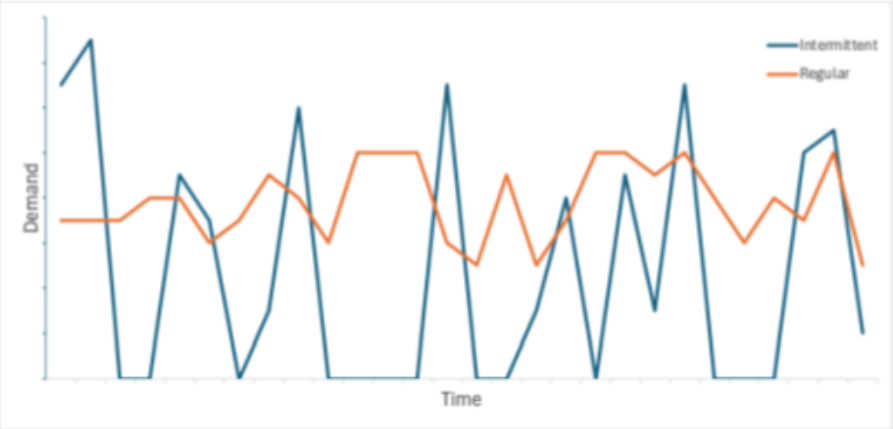


Figure 1. Intermittent vs. Regular pattern

Therefore, this thesis evaluates forecasting models that can be integrated into existing logistics systems, such as SILOMS, as improved forecasting accuracy has the potential to directly enhance operational readiness by reducing part shortages and excessive inventory levels. Furthermore, more accurate forecasts can lead to better procurement planning, cost savings, and reduced logistical uncertainty. Finally, the findings of this study may also be relevant to other defense organizations facing similar challenges with intermittent demand.

F. ORGANIZATION OF THE STUDY

This thesis is organized into five chapters. Chapter II presents the literature review, covering the theoretical foundations of forecasting methods, intermittent demand behavior, and recent developments in machine learning models.

Chapter III presents the research methodology used in the study, describing the dataset, the modeling designs, the criteria for model evaluation, and the rationale for comparing machine learning and traditional forecasting techniques.

Chapter IV presents the results of the forecasting models, including a comparative analysis of their performance using real data from the Brazilian Air Force. The findings are analyzed considering the research objectives, and their practical implications are discussed.

Finally, Chapter V offers concluding remarks, including a summary of key findings, recommendations for implementation, and suggestions for future research. This

chapter also examines the broader implications of the study for defense logistics planning and forecasting strategies.

THIS PAGE INTENTIONALLY LEFT BLANK

II. LITERATURE REVIEW

A. FORECAST

1. Qualitative vs. Quantitative

Demand forecasting sets the entire supply chain in motion (Boylan & Syntetos, 2021) and it can be classified in two categories: qualitative and quantitative.

Qualitative forecasting relies on expert judgment, intuition, and subjective assessments. It must be used when there is no demand history, or the available data is not relevant for the forecast. There are cases in which this qualitative forecasting is the only possible option, such as when historical data is lacking, for example when launching a new product, or operating in new and unique environmental conditions, as during military operations in a novel environment (Hyndman & Athanasopoulos, 2018).

In contrast, quantitative methods can be divided into two groups: time-series and causal. Time-series methods rely entirely on the historical behavior of a variable and are appropriate only when past patterns are expected to continue into the future. Causal methods, on the other hand, relate the variable of interest to one or more explanatory factors and, therefore, do not depend on the assumption that historical patterns will repeat (Hyndman & Athanasopoulos, 2018). Because they are data-driven, quantitative methods are generally more consistent and objective in generating forecasts using only judgement, though they may fail to capture external factors or sudden shifts not reflected in the historical record (Hyndman & Athanasopoulos, 2018).

Given that the Brazilian Air Force maintains substantial historical demand data for aircraft spare parts, and that forecasting in this context involves managing thousands of inventory items, the focus of this study is on quantitative forecasting using time-series methods. Nevertheless, understanding the conceptual differences and complementary roles of qualitative approaches provides a broader perspective on forecasting strategy in defense logistics.

2. Time-series

As Boylan and Syntetos (2021, p. 1) describe, "demand will typically be accumulated in some predefined 'time buckets' (periods), such as a day, a week, or a month." Therefore, the length of the time periods is aggregated to form a time bucket is a very important decision. This gathering, observed sequentially over time, is referred to as a time series.

A time series is a sequence of observations collected at consistent and evenly spaced time intervals (Boylan & Syntetos, 2021). This structure enables the visualization of past behavior, which is essential for identifying recurring patterns and fluctuations, and therefore, predicting future behavior based on that.

Time series can present different behaviors depending on the disposition of the demand during the period. According to Makridakis et al. (1983), identifying data patterns is essential to choosing the most appropriate forecasting method, and these patterns can be classified into four different types illustrated in Figures 2, 3, 4 and 5:

a. Horizontal pattern

The demand oscillates around a constant mean.

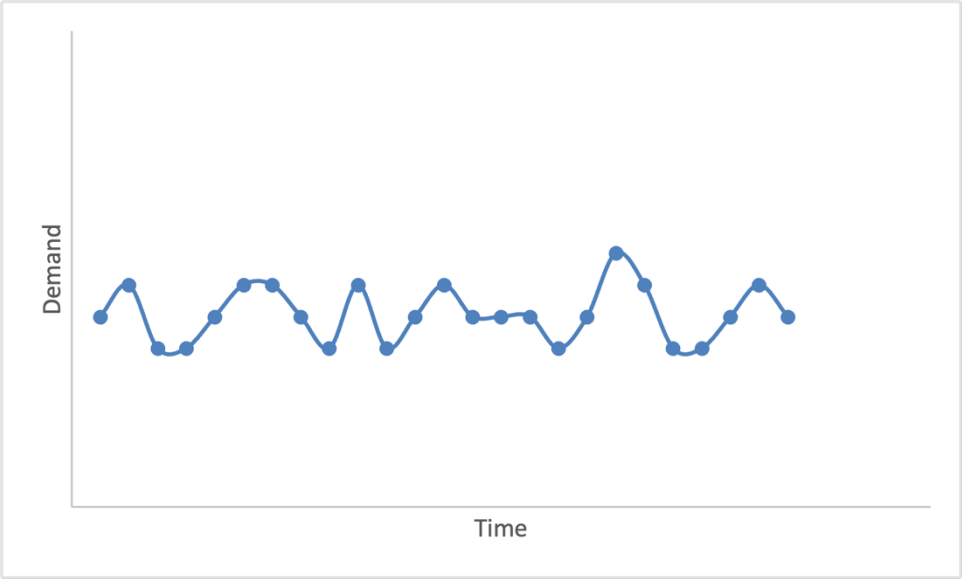


Figure 2. Horizontal Pattern. Adapted from Makridakis et al. (1983).

b. Seasonal pattern

The demand is influenced by seasonal factors such as the day of the week, the month, or the quarter of the year, resulting in regular and constant interval between peaks and between troughs.

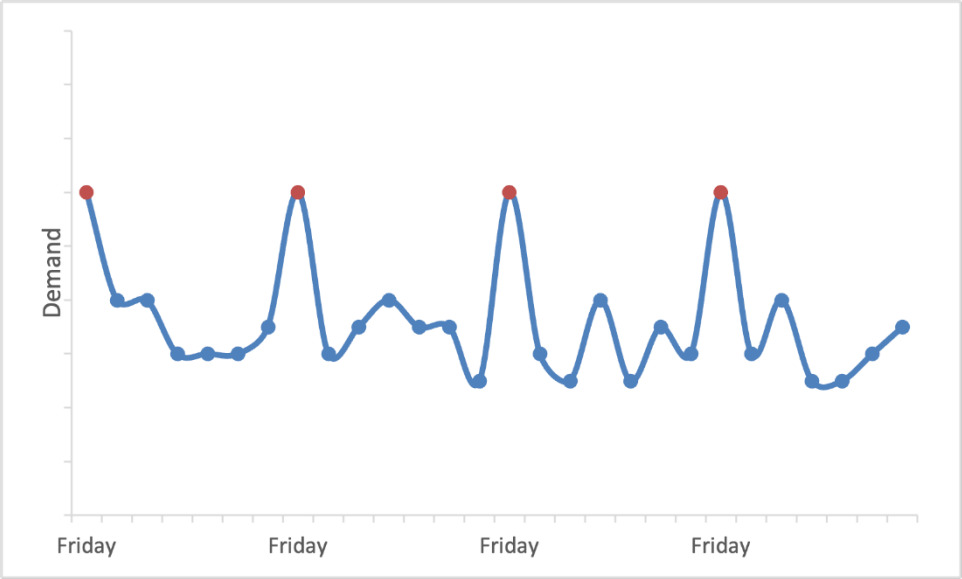


Figure 3. Seasonal Pattern. Adapted from Makridakis et al. (1983).

c. Cyclical pattern

Similarly to the seasonal pattern, the demand is influenced by seasonal factors; however, the variation in demand is related to long-term fluctuations, resulting in a larger length and magnitude.

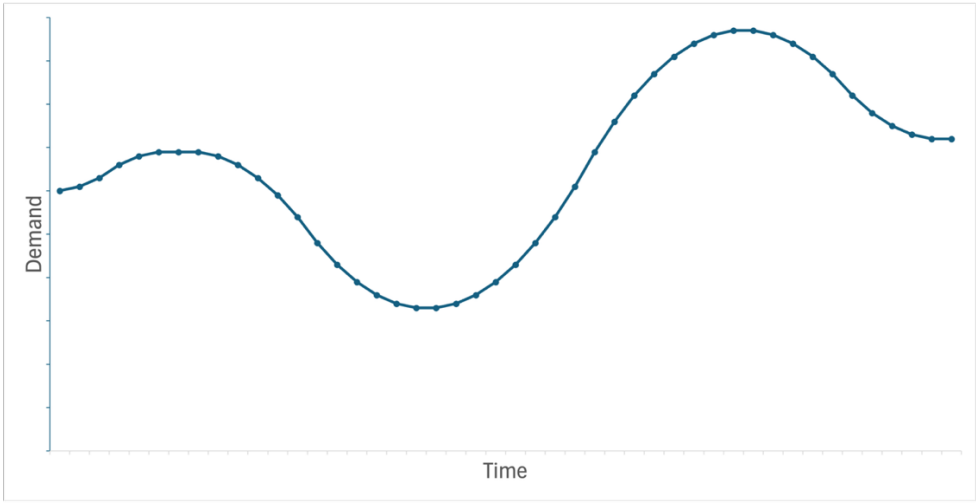


Figure 4. Cyclical Pattern. Adapted from Makridakis et al. (1983).

d. Trend pattern

The demand presents a long-term increase or decreases.

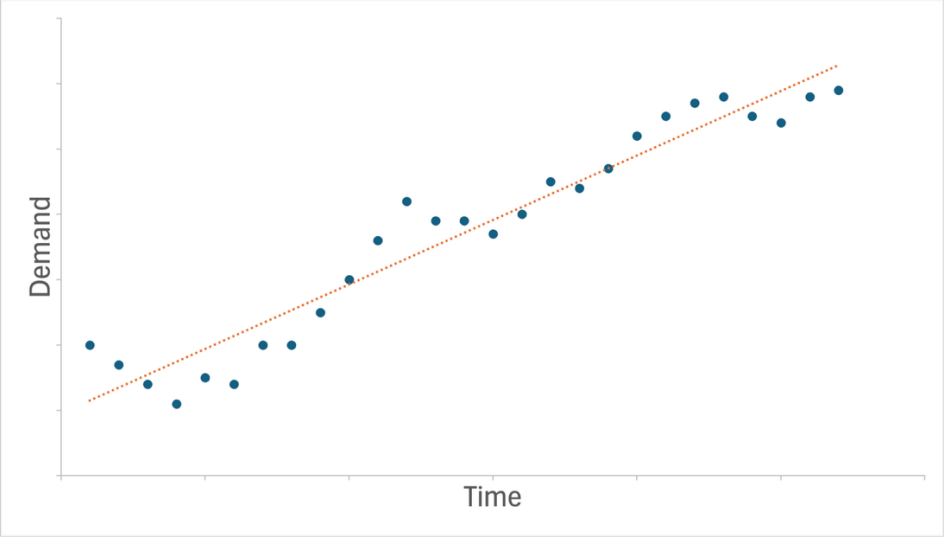


Figure 5. Trend Pattern. Adapted from Makridakis et al. (1983).

Moreover, there are behaviors resulting from changes in the fundamental demand process, such as: the impulse pattern, where demand temporarily increases and then returns to its original level; the step pattern, where demand permanently shifts from one period to the next; and the ramp pattern, where demand that usually remains constant suddenly develops a trend (Montgomery & Johnson, 1976). Although Makridakis et al. (1983) and Montgomery & Johnson (1976) established and defined various pattern types that help evaluate and identify the best mathematical models for forecasting, one pattern remains undefined and unsolved, one that is common in the industry but very complex to model: the intermittent pattern.

3. Intermittent demand

Despite time series forecasting methods being able to incorporate periods of zero demand, they are not designed for situations with long periods of zero in the data. When this occurs, models that rely on trends or seasonality often struggle to recognize patterns,

resulting in unstable forecasts. According to Hyndman & Athanasopoulos (2018) and Boylan & Syntetos (2021), classical smoothing methods typically underperform under these conditions because the assumptions of regular demand intervals are violated. These limitations motivate the use of specialized techniques for intermittent demand forecasting.

"An intermittent pattern occurs when demand appears sporadically, with no demand at all in some periods" (Boylan & Syntetos, 2021, p. 1). Moreover, "when demand occurs, the demand quantity may be constant or variable, perhaps highly so, leading to what is often termed 'lumpy demand'" (Boylan & Syntetos, 2021, p. 3). Figure 6 exemplifies a time series that presents intermittent behavior:

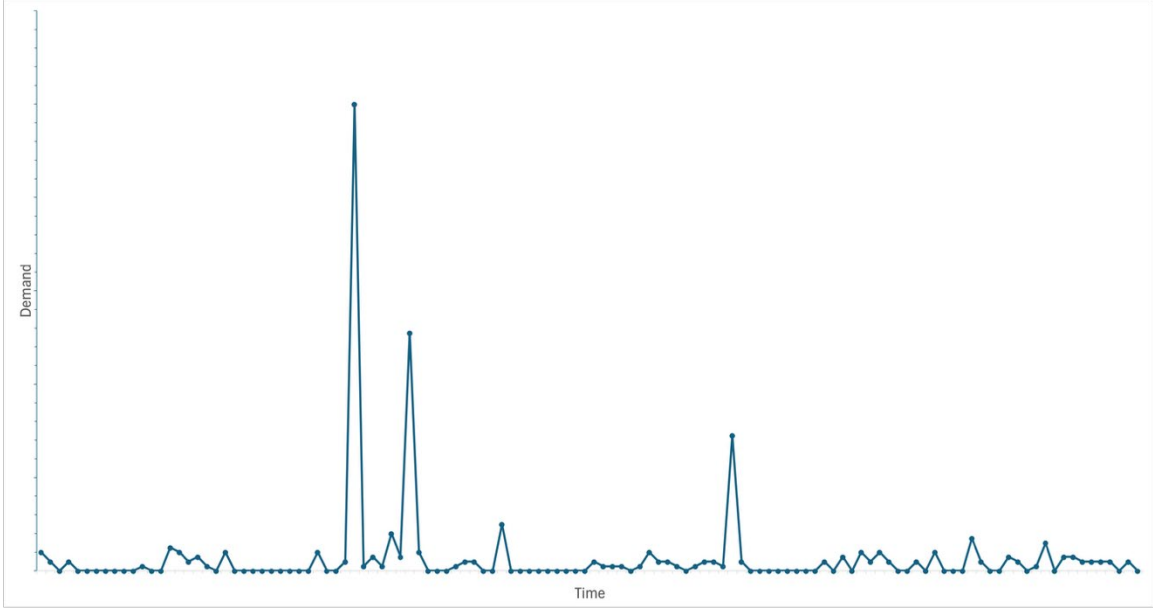


Figure 6. Intermittent demand pattern.

Components or sub-assemblies used in constructing a final product are essentially what service parts, also known as spare parts, are. (Boylan & Syntetos, 2021).A company's inventory can have a majority of components that experience intermittent demand. "In some industries, this proportion may reach 60% or 70%" (Boylan & Syntetos, 2021, p. 1). "Defense inventories have been repeatedly identified as a high-risk area with a direct

impact on a nation's economy" (U.S. Government Accountability Office [GAO], 2015, p. 2).

In the United States, for example, the Department of Defense (DoD) "manages around five million secondary items. These include repairable components, subsystems, assemblies, consumable repair parts, and bulk items (U.S. Government Accountability Office [GAO], 2015, p. 1). Similarly, the situation for the Brazilian Air Force is not different.

In spite of its importance and complexity, the area of intermittent demand forecasting was not explored until 1972, when John Croston made a major contribution by developing a method specifically applied to this type of demand pattern. The technique created could answer the two main questions around the intermittent behavior: 1) When the next demand will occur? and 2) Once demand is realized, what will be the respective size? (Xu et al., 2012) In recent years, after a period of little activity, the area has been revisited, exploring new multivariate methods or univariate enhancements for Croston's technique.

4. Univariate vs. Multivariate

Univariate forecasting utilizes only historical values of the variable being predicted. Most familiar time series forecasting methods are univariate. This type of forecasting is usually simpler to implement, requires less data and are often used in organizations due to its ease of implementation and interpretation (Preez & Witt, 2003). However, its predictions are not very effective in situations where the historical data is not the only factor that influences demand behavior (Preez & Witt, 2003).

On the other hand, multivariate forecasting includes additional explanatory variables believed to relate to the target variable, such as calendar indicators (e.g., month or quarter), part characteristics, inventory levels, or operational activity levels (Preez & Witt, 2003). By analyzing the impact of each variable on another, multivariate models can identify more complex relationships and improve forecast accuracy (Preez & Witt, 2003).

Choosing between univariate and multivariate forecasting depends on data availability and quality, as well as demand behavior. Due to the high variability and intermittency of military aircraft's spare parts demand, univariate models tend to be often used, due to their simplicity and low data requirements. However, the potential of multivariate models to capture non-obvious relationships represents a promising alternative, especially those built using machine learning, which can capture complex, non-obvious relationships that traditional statistical models may overlook.

B. MACHINE LEARNING

In recent years, machine learning and artificial intelligence have become more capable and available, due to the fast development of large language models and their democratization among all users. Machine learning refers to a collection of computational methods that improve their performance by learning patterns from data. In practice, a model is exposed to examples (data) and uses statistical or algorithmic techniques to discover relationships that allow it to make predictions or decisions on new, unseen data (Zhou, 2020).

A central distinction within machine learning is between supervised and unsupervised learning, which will be discussed in the next section.

1. Supervised vs. Unsupervised learning

Supervised learning uses training data with both inputs (also called features, covariates, or independent variables) and labeled outcomes (also called targets, dependent variables, responses). The algorithm learns from the training data to identify labeled outcomes, and the resulting model or program can be used to predict the outcome in new test data, never seen before. Common supervised methods include generalized linear models, decision tree algorithms (such as XGBoost), and artificial neural networks (sometimes called deep learning) (Xu et al., 2020).

Unsupervised learning models are built using only input features, without outcome labels and are often used for finding groups within data (clustering). Typical techniques include k-means clustering, hierarchical clustering, principal components analysis, and t-

SNE. These methods can serve as intermediate steps in creating covariates or outcomes for supervised learning (Xu et al., 2020).

This thesis focuses exclusively on supervised learning algorithms, with the objective of predicting future demand quantities from labeled historical data.

2. Application in forecasting models

Machine learning is increasingly applied to forecasting problems across industries such as retail, e-commerce, manufacturing, energy, and finance (Douaioui et al., 2024). Algorithms such as gradient boosting, neural networks, and other supervised methods have demonstrated their ability to capture complex relationships and improve predictive accuracy compared to traditional models, especially when dealing with large datasets with many features and a certain level of complexity.

Machine learning has also been applied to inventory management to detect nonlinear interactions using explanatory variables such as past demand, product characteristics, pricing, promotions, weather data, and other external factors (Amosu et al., 2024). This approach enables learning across multiple items at once, which is especially useful in large-scale inventory systems (Salinas et al., 2020).

Despite these advancements, the use of machine learning in defense logistics, particularly for forecasting intermittent demand for aircraft spare parts, remains limited in both practice and the academic literature due to its inherent characteristics of low volume and high variability, making forecasting more challenging.

This thesis explores that opportunity by applying machine learning to the forecasting of demand for aircraft spare parts in the Brazilian Air Force, comparing its performance against traditional statistical approaches under a real-world scenario.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY

A. RESEARCH DESIGN

This study's main objective is to determine if machine learning techniques can improve the accuracy of forecasts of intermittent demand for aircraft spare parts, compared to traditional time-series models used currently by the Brazilian Air Force (FAB). The research implements an empirical, comparative, and quantitative research design (Creswell & Creswell, 2018; Ragin, 2014) with real operational data from FAB's Materiel and Services Logistics Integrated System (SILOMS) to evaluate forecasting methods in a realistic defense logistics situation.

The study compares two forecasting approaches:

(1) Traditional statistical models, univariate forecasting methods that rely only on historical demand data, such as Simple Moving Average (SMA), Single Exponential Smoothing (SES), and Croston's method with its Syntetos–Boylan Approximation (SBA) variant, and

(2) Machine learning models, multivariate forecasting methods that utilizes of nonlinear relationships between multiple variables. The focus is mainly on XGBoost and artificial neural networks, selected for their strong performance in recent forecasting research (Boylan & Syntetos, 2021;Douaioui et al., 2024).

Quantitative analysis is used to measure and compare the models using error metrics such as Mean Absolute Scaled Error (MASE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

B. DATA DESCRIPTION

The data used in this study were extracted from the SILOMS, which is the ERP system that serves as the main logistics management platform for the Brazilian Air Force (FAB). SILOMS contains information related to logistics, including inventory levels, procurement, transportation, and spare part demand. For this research, the focus was the historical demand data of aircraft spare parts, which is the main element in the FAB's current forecasting model, and incurs in inventory stockouts and impacts readiness.

The dataset contains monthly demand records for all part numbers used in FAB T-27 Tucano (EMB-312) aircraft fleet, totaling 6,648 unique PNs. The selected data covers a period of 124 months (approximately 10 years), from January 2015 to April 2025, and each observation corresponds to the quantity of a specific part consumed during a given month.

Before proceeding to modeling, some preprocessing steps were conducted to ensure the integrity and usability of the data. First, the dataset was filtered to include only spare parts with at least one recorded demand during the analyzed period, as items with continuous zero demand provide no variation to forecast. Next, since the dataset only contains rows for months that had demand, missing months were identified and filled with zero-demand values to maintain a consistent time index through the dataset, which would be later necessary to correctly train both traditional and machine learning models. Additional procedures included data type standardization and aggregation of demand per part number to eliminate duplicates and ensure accuracy.

The resulting dataset includes 2,428 unique items presenting frequent zero-demand periods and occasional demand, which will be called spikes, aligning with the study's focus on evaluating forecasting models with intermittent demand behavior.

Given the sensitivity of the data and its association with defense operations, all identifiers related to specific aircraft systems, part descriptions, or suppliers were removed. The dataset was anonymized to preserve confidentiality while retaining the statistical and temporal characteristics necessary for analysis.

C. DEMAND CATEGORIZATION

In order to separate items into intermittent vs. non-intermittent demand, rather than applying theoretical classification metrics such as the Average inter-demand interval and the squared coefficient of variation (CV^2) (Syntetos et al., 2005), this study employed a rule-based filtering approach grounded in observed demand behavior within the dataset.

For each PN, the total number of months in which the demand quantity was greater than zero, referred to as spikes, was calculated across the ten-year period. Only the items

that presented the following number of spikes were considered as having a realistic level of intermittency:

- *Minimum spikes* = 4
- *Maximum spikes* = 60

Items with fewer than four spikes were not considered due to insufficient data for model training. On the other hand, items that presented more than sixty spikes were considered too regular to represent intermittent demand behavior.

This categorization was implemented in Python (Python Software Foundation, 2024) using the pandas library (McKinney, 2010). The algorithm counted the number of months with spikes per part number, applied the thresholds, and then produced a filtered dataset containing only the qualified items, totaling 2,428 part numbers.

This categorization method ensured that the final dataset reflected the operational reality of the Brazilian Air Force (FAB), where most spare parts are neither completely inactive nor regularly consumed but instead display intermittent usage over time.

D. DATA TRIMMING EARLY OF ZERO-DEMAND MODELS

Next, the dataset was submitted to additional filtering to trim long zero-demand periods at the beginning of each part's time series, since many part numbers exhibited several years of inactivity before the first recorded demand event. Keeping those long initial zero sequences would bias both traditional and machine-learning models, artificially inflating the number of non-demand periods.

The data trimming algorithm was developed in Python to automatically detect and remove irrelevant pre-demand periods. The cleaning process followed these rules:

(1) For each part number, the algorithm identified the first and second months with nonzero demand ("spikes").

(2) It measured the gap (in months) between the two spikes.

(3) It trimmed the dataset to start with the same number of zero-demand months before and after the first spike, excluding any earlier all-zero months.

After applying this algorithm, the dataset size was reduced from 6,648 part numbers to 2,428, representing only those with sufficient demand activity and valid intermittent behavior.

E. FORECASTING MODELS

This research compares two different forecasting approaches: univariate traditional statistical models and multivariate machine learning models. The main objective is to evaluate their performance when predicting intermittent demand for aircraft spare parts for FAB's T-27 TUCANO fleet. All models were trained and tested on the same dataset, using identical validation procedures.

1. Traditional Forecasting Methods

a. Simple moving average (SMA)

The Simple Moving Average (SMA) model estimates future demand as the arithmetic mean of the most recent T observations. Let X_t denote the actual demand observed in period t, and let $\hat{y}_{t+1}$ represent the forecast for the next period. According to Makridakis et al. (1983) SMA is calculated by:

$$\hat{y}_{t+1} = \frac{1}{T} \sum_{i=t-T+1}^{t} X_i . \qquad (1)$$

This approach smooths the variations during the observed period and assumes that the demand levels of the T observations are representative of the near future. Due to its characteristic of assigning equal weight to all past observations, SMA is more effective for relatively stable series. However, for intermittent demand behaviors, it performs poorly, usually underpredicting in the presence of sudden spikes or prolonged zero-demand

intervals. This method is currently used by the Brazilian Air Force (FAB), with a *T* of 24 months, and so it will be used as a benchmark for the comparison with other forecasting methods.

b. Simple exponential smoothing (SES)

Single Exponential Smoothing introduces a weighting scheme that decays exponentially with time, giving greater influence on more recent observations. According to Makridakis et al. (1983) describes SES as

$$\hat{y}_{t+1} = \alpha X_t + (1-\alpha) F_t, \qquad (2)$$

where $0 < \alpha \leq 1$, the smoothing parameter controls the outliers' impact on the forecast. Typically, the smoothing parameter is close to 0, ensuring that historic values outweigh recent observations.

Although SES adapts faster than SMA to changes in the mean level, it still assumes a continuous demand with no intermittency. Therefore, when long sequences of zeros occur, its forecasts approach zero, and similarly to SMA, SES will often underforecast subsequent spikes.

c. Croston's Method (CR)

Croston (1972) proposed a formulation explicitly designed for intermittent demand. Instead of modeling the whole time series, it focuses on the two main characteristics of an intermittent series and separately forecasts them:

- the demand size when demand is different from zero and
- the inter-demand interval (time between non-zero demands).

If $y_t = 0$, $\hat{q}_t = \hat{q}_{t-1}$ and $\hat{p}_t = \hat{p}_{t-1}$ and $\hat{y}_t = \hat{y}_{t-1}$. For each non-zero observation q_t $(q_t \neq 0)$ and interval p_t:

$$\hat{q} = \alpha q_t + (1-\alpha)\hat{q}_{t-1}$$
$$\hat{p} = \alpha p_t + (1-\alpha)\hat{p}_{t-1}, \quad (3)$$

and the demand forecast is given by

$$\hat{y}_{t+1}^{CR} = \frac{\hat{q}_t}{\hat{p}_t}. \quad (4)$$

This method was the first one created exclusively dedicated to forecasting intermittent demand behavior, and it was a huge contribution in the area. This model allows the update only when a non-zero demand is observed, thus avoiding distortion from consecutive zeros. However, it underpredicts when the intermittent series is composed of a long period with zero demand.

d. Syntetos-Boylan approximation (SBA)

Syntetos and Boylan (2005) introduced a bias correction to Croston's method, yielding the Syntetos–Boylan Approximation (SBA):

$$\hat{y}_{t+1}^{SBA} = \left(1 - \frac{\alpha}{2}\right)\frac{\hat{q}_t}{\hat{p}_t}. \quad (5)$$

The adjustment slightly reduces the forecast magnitude and generally produces unbiased results under a wide range of intermittency conditions. Because of its simplicity and robustness, SBA is widely regarded as the benchmark model for intermittent demand forecasting (Xu et al., 2012) and serves as a reference point in this study.

2. Machine learning methods

Unlike traditional forecasting methods that use predefined expressions to calculate their parameters, machine learning models learn the mapping between inputs and outputs algorithmically from the training data. In this study, two supervised learning models were tested: Extreme Gradient Boosting (XGBoost) and an artificial neural network (NN). Both

models accept multiple variables and are capable of modeling nonlinear dependencies and complex interactions that traditional models cannot easily express.

a. Extreme Gradient Boosting (XGBoost)

According to Chen & Guestrin (2016), XGBoost is a "scalable tree boosting" algorithm popular among data scientists, known for delivering cutting-edge results on numerous problems. It is a supervised, tree-based ensemble method that builds a sequence of small decision trees, where each new tree focuses on the residual errors left by the previous ones (Douaioui et al., 2024), as shown in Figure 7.

Figure 7. Decision tree example

In supervised learning, the model is trained on inputs (features) and outputs (the target variable), learning how to generalize this relationship to new, unseen data. In practical terms, the model learns complex, nonlinear relationships by iteratively correcting itself, using decision trees as weak learners, i.e., simple rule-based models that recursively split the data based on threshold conditions.

While performing these corrections, a built-in regularization scheme controls model complexity to reduce overfitting. Regularization refers to penalties imposed on overly complex models, for example, limiting the number of leaves in each tree or shrinking leaf weights, to ensure that the model does not simply memorize noise.

According to Ji et al. (2019), XGBoost models reliably deliver greater accuracy and quicker computation times compared to other ensemble methods in demand forecasting tasks.

For intermittent demand, XGBoost has attractive attributes since it can aggregate a large number of features, such as lagged demand, rolling statistics, and calendar indicators, and discover interactions among them without requiring the forecaster to specify a rigid functional form. Features are the input variables used by the model, and XGBoost can automatically learn nonlinear interactions among them without requiring manual specification of relationships. It is also computationally efficient, which is important when training across many part numbers simultaneously, as in the dataset used in this study. When trained in this way, the method functions as a global model, meaning it learns patterns across multiple items rather than fitting a separate model for each time series. This allows for the capture of shared structures, such as similar demand patterns or seasonal effects, across part numbers. Figure 8 illustrates how the model works.

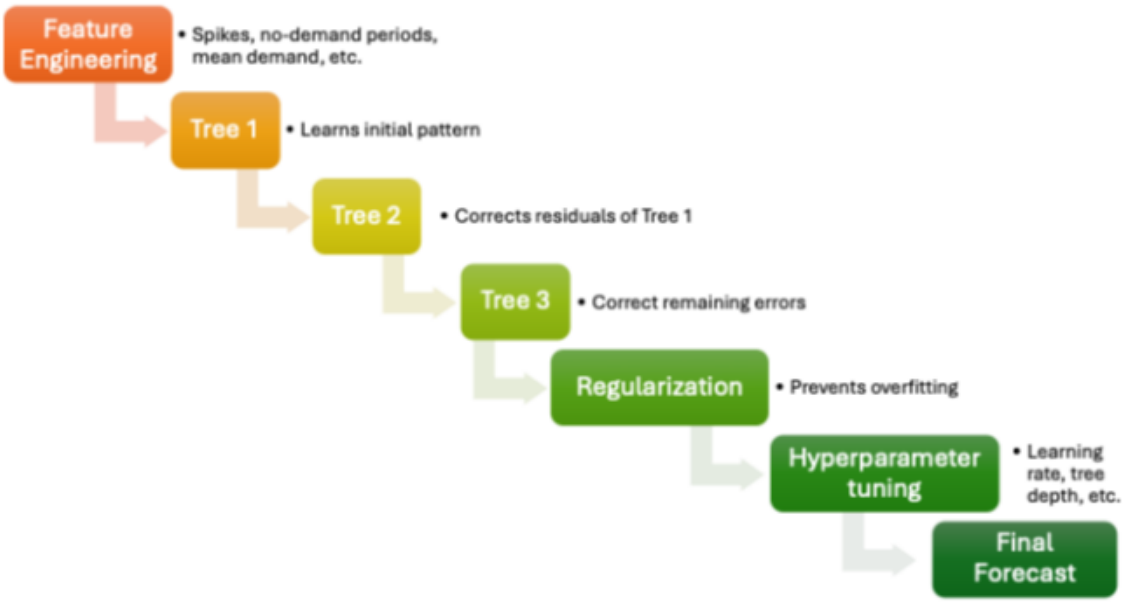


Figure 8. XGBoost Overview

b. Artificial Neural Networks (NN)

The artificial neural network implemented in this study is a supervised, fully connected architecture (multi-layer perceptron) trained globally across all part numbers. A multi-layer perceptron (MLP) is a neural network composed of layers of interconnected "neurons," where each neuron applies a weighted sum of the inputs followed by a nonlinear activation function. These stacked layers enable the model to learn complex patterns that simple linear methods cannot capture. The network maps a fixed-length (24 months) input to output a single-step demand prediction, capturing nonlinear patterns and dynamic effects without relying on additional engineered or categorical features. Each input neuron corresponds to one month of past demand, and the hidden neurons transform these inputs before producing the final forecast. A single neural network is trained and used for all PNs: the training set is constructed from 25-month rolling windows across the entire panel, where the first 24 months of each window serve as input and the 25th month as output. After training on these windows from all parts, the same global network is applied to generate one-step-ahead forecasts for every PN.

The network was implemented using the *scikit-learn* library in Python and configured to remain compatible with the available data volume. Early stopping was implemented to avoid overfitting during training. Also, hyperparameters such as the number of hidden layers, number of neurons, and learning rate were used to tune the model. In this architecture, the hidden layers allow the network to detect nonlinearities in the demand history, such as long quiet periods followed by sudden movements, while the output layer produces a single numeric forecast.

However, neural networks generally provide limited interpretability, meaning it is difficult to trace how specific inputs or features contribute to the final prediction. For this reason, a compact architecture with few layers and fewer neurons was chosen, prioritizing stability, reproducibility, and easier diagnostic analysis. Figure 9 represents a schematic of the neural network that transforms the 24-month demand into a forecast.

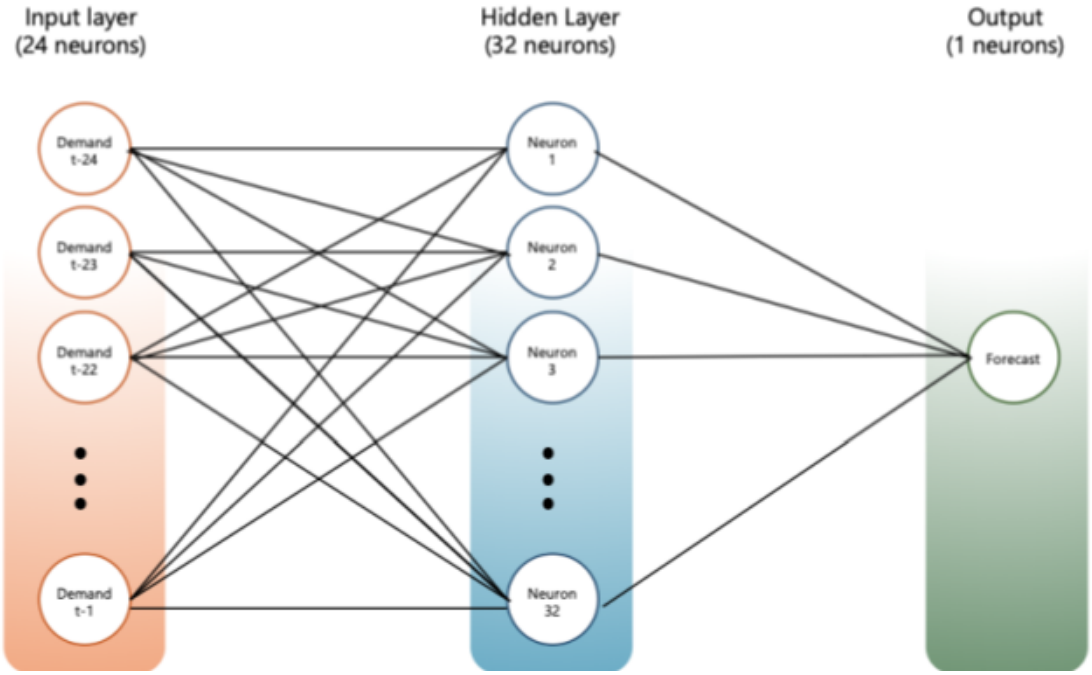


Figure 9. Neural Network diagram

3. Comparative considerations

The four traditional and two machine learning models represent a variety of complexities, from simple averaging rules such as SMA, currently used by FAB, to more robust and complex models such as XGBoost. At the same time, the methods also represent a spectrum of cost and implementation, from traditional approaches that offer transparency and low computational cost but rely on strong assumptions about demand stability, to Machine-learning models, which, by contrast, are more flexible, imposing fewer structural assumptions and can exploit auxiliary information through feature engineering (defined below), but are more computationally exigent. The comparison in this study will reveal whether the added flexibility of machine learning justifies the higher effort required.

F. FEATURE ENGINEERING FOR MACHINE LEARNING MODELS

Feature engineering is a crucial step in preparing data for machine learning models. Unlike traditional statistical methods that use only historical demand data, machine

learning algorithms require a structured set of input variables (features) to identify temporal patterns and nonlinear relationships.

Many researchers have tried to identify what characteristics define when an intermittent demand occurs. According to Zhuang et al. (2022, p. 48), scholar proposed features as "average demand interval (Levén and Segerstedt, 2004), demand frequency (Gutierrez et al., 2008), demand interval distribution (Croston, 1972), demand size (Nasiri Pour et al., 2008), demand distribution (Zotteri, 2000), early data generated by users during the purchase process (Verganti, 1997), promotion or holiday information (Hyndman, 2006), demand autocorrelation (Willemain et al., 2004), product transaction patterns (Syntetos, 2001)." These characteristics help determine whether a series exhibits the sporadic, zero-inflated behavior typically associated with intermittent demand, and they form the basis for various classification and forecasting approaches in the literature.

Defining the appropriate features to use in the models is essential for obtaining accurate results. If features are weak or poorly engineered, the model can overfit to noise or memorize rare spikes. On the other hand, point forecasts may be conservative after long zero runs unless features explicitly signal upcoming spikes. To mitigate these issues and based on the characteristics defined when an intermittent demand occurs, I: (i) standardized a consistent feature set across parts, (ii) used early stopping to avoid overfitting, and (iii) evaluated performance with a scale-free performance metric (MASE) to ensure no bias in zero-inflated series.

To create appropriate features, the starting point was the average monthly demand of each spare part, covering all months available for each Part Number after data trimming. Each record contained the observed demand quantity for a given month. Based on that, new explanatory variables were derived to capture temporal structure, historical dependencies, and potential seasonal effects that might influence demand behavior.

1. Lag features

Lag features were constructed to represent the most recent demand observations. For each month t, lagged values such as $y_{t-1}, y_{t-2}, \ldots, y_{t-12}$ were included as predictors. These features allow the model to learn short-term dependencies, such as persistence or

decay following a demand event. Additional lag variables were created to capture relative changes in recent periods, helping with the model's identification of upward or downward shifts in demand.

2. Demand activity indicators

Several indicators were constructed from rolling windows to summarize recent demand behavior over multiple time horizons. The model computed the cumulative demand over the past 3, 6, and 12 months and also the proportion of months with non-zero usage for each window. Together, these indicators capture the more important characteristics of an intermittent demand behavior, inspired by Croston (1972), and provide a compact representation of demand activity, frequency, and intensity. For example, a high rolling sum combined with a low non-zero rate may indicate intermittent large spikes, whereas a moderate sum with a high non-zero rate suggests steady demand.

3. Temporal and calendar features

Calendar variables were added to account for potential temporal effects. Month, quarter, and an elapsed-time index were used to represent the chronological order of each observation. These features enable the model to detect cyclical or gradual structural patterns.

4. Zero sequences

The primary variable for characterizing intermittency measures the number of months since the last non-zero demand, effectively encoding the length of zero sequences for each part. This "zero-sequence" feature (implemented as the *recency_nz* variable in the code) records how long the item has remained without demand up to month t, an important characteristic of intermittent demand series. Therefore, this feature enables the model to infer the likelihood of a new demand spike based on how long the part has remained without demand. Additionally, the lag variables are combined with it, so that the model can distinguish whether the most recent observations correspond to zero demand or to non-zero demand and how large those non-zero demands were.

5. Encoding

Categorical features like month and quarter were kept as ordinal integers because XGBoost can process these directly through split-based learning.

G. MODEL TRAINING AND VALIDATION

In the Brazilian Air Force (FAB), new demand information becomes available each month, and so the SMA for 24 months must be continuously updated. To replicate this environment, a rolling validation was employed, incrementally advancing new monthly demand over time, retraining models on past observations, and producing forecasts for the next period.

1. Rolling validation

All models were trained on all observations within the current window and used to forecast demand for the next month (t+1). After that, the actual demand for t+1 was incorporated into the training data, and the process continued sequentially. This procedure was applied to mirror FAB's monthly forecasting workflow, in which SMA is continuously updated as new demand occurs.

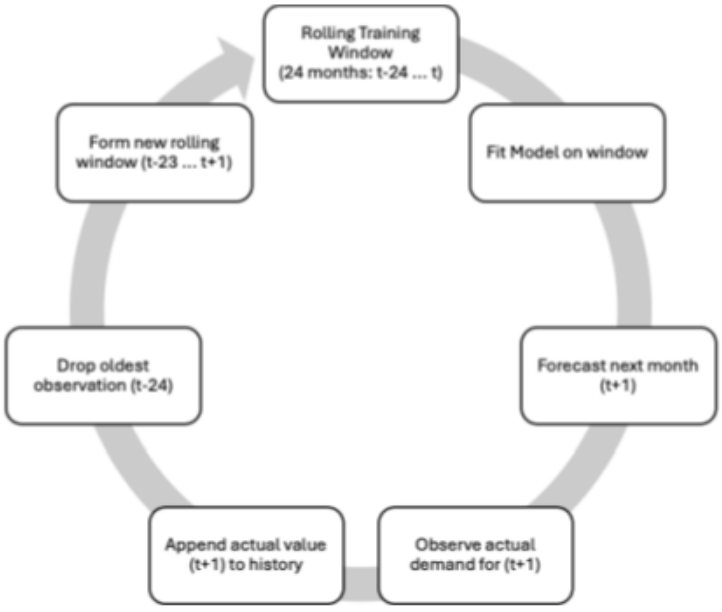


Figure 10. Rolling validation diagram

2. Data splitting

This study adopted a dynamic evaluation, in which each part number was processed individually using a rolling training window of fixed length of 36 months. The model was then fitted on this moving window and used to predict demand for the subsequent month. After the true demand became known, that observation was appended to the time series, and the oldest month was dropped from the window. This procedure is repeated sequentially until the end of the dataset. When sufficient data were available, the most recent 10–15% of the training window was internally reserved as a validation segment for hyperparameter tuning and early stopping.

3. Training procedure

For the traditional statistical models (SMA, SES and Croston-SBA), forecasting parameters were selected to optimize performance metrics within each rolling window- in other words, smoothing parameters (α's) were specific to each PN and could change over the time period studied. All implementations were developed in Python using the *statsmodels* library for exponential smoothing and custom-coded functions Croston-SBA based on their original formulations.

For machine learning models, were trained globally using all PNs and followed a supervised approach built on the feature set previously described:

XGBoost was implemented as a two-stage model, combining a binary classifier to predict whether demand would occur (non-zero) and a regressor to estimate its magnitude if it does. Both components used gradient-boosted decision trees trained with logistic loss and squared-error loss functions, respectively. The classifier (XGBClassifier) was trained to estimate the probability that demand in month t is non-zero, while the regressor (XGBRegressor) estimated the demand size given that a demand occurs. This modelling aligns with the intermittent demand, since the model to learns the timing of demand and its magnitude. The complete set of hyperparameters was fixed and applied globally to all part numbers as given below, together with their functional roles within the boosting algorithm.

Classifier hyperparameters (for whether demand will be non-zero):

- *n_estimators* = 150 – number of boosted trees; higher values allow the ensemble to learn more complex patterns but increase computation and risk of overfitting.
- *max_depth* = 4 – maximum depth of each decision tree; controls how complex each tree can become and how many interactions between features can be captured.
- *learning_rate* = 0.08 – shrinkage factor; each tree contributes only a small correction, stabilizing training and reducing overfitting.
- *subsample* = 0.9 – fraction of rows sampled for each tree; adds randomness to reduce overfitting.
- *colsample_bytree* = 0.9 – fraction of features sampled for each tree; further reduces overfitting and improves generalization.
- *objective* = *"binary:logistic"* – logistic loss used to predict $P(y_t>0)$.

Regressor hyperparameters (for demand magnitude):

- *n_estimators* = 250 – larger ensemble to estimate continuous values; more trees help capture nonlinear patterns in positive demands.
- *max_depth* = 4 – same complexity constraints as the classifier, balancing expressiveness with generalization.
- *learning_rate* = 0.05 – smaller step size to improve stability when fitting continuous demand magnitudes.
- *subsample* = 0.9 – *colsample_bytree* = 0.9 – same as classifier.
- *objective* = *"reg:squarederror"* – standard squared-error loss for continuous regression.

The Artificial Neural Network was implemented as a compact, fully connected MLPRegressor using the scikit-learn framework. It was trained globally across all parts using only recent lagged demand values as inputs. Each training consisted of a 24-month demand window (lags $t-1$ to $t-24$), which the network mapped to a single next-month

prediction. The architecture contained one hidden layer with 32 neurons and *ReLU* activation, followed by a linear output neuron that produces the forecast. The complete set of hyperparameters was fixed across all rolling windows and applied globally to all part numbers. Their final values were:

Neural Network hyperparameters:

- *hidden_layer_sizes* = (32,) – one hidden layer with 32 neurons; the number of neurons controls the model's capacity to learn nonlinear relationships.
- *activation = "relu"* – the Rectified Linear Unit activation function introduces nonlinearity and helps the network detect changes in demand.
- *solver = "adam"* – a stochastic gradient-based optimizer that adapts learning rates during training.
- *learning_rate_init* = 0.001 – initial learning rate; determines how quickly the network's weights update in response to errors. Lower values stabilize training but require more iterations.
- *max_iter* = 500 – maximum number of training iterations.
- *random_state* = 10 – ensures reproducibility of weight initialization and training.

While each traditional model was trained independently for each part number, the machine learning methods used demand for all items as predictors to enable cross-series learning, facilitating consistent performance comparisons.

H. EVALUATION METRICS

After generating the forecasts, the performance of each model's forecasts was evaluated using statistical error measures (1) Mean Absolute Scaled Error (MASE); (2) Root Mean Squared Error (RMSE); and (3) Mean Absolute Error (MAE).

According to Hyndman (2006), some accuracy metrics are not suited for assessing intermittent demand forecasts due to their sparse demand patterns. He also categorizes four

types of forecast error metrics, of which this study employs a combination of scale-dependent and scale-free metrics to provide a comprehensive assessment of model performance.

1. Scale-dependent metrics

Scale-dependent metrics are accuracy measures whose numerical values depend on the absolute magnitude of the data being forecasted. This means that two series with identical relative errors can produce very different metric values simply because one series has larger demand levels than the other (Hyndman, 2006). As a result, scale-dependent metrics cannot be used to compare accuracy across items with different units or different demand magnitudes, which is particularly important in this study because the dataset contains many part numbers with widely varying demand levels (Hyndman, 2006).

a. Mean Absolute Error (MAE)

The Mean Absolute Error measures the average magnitude of forecasting errors without considering their direction. Considering that $e_i = X_i - F_i$, it is computed as the mean of the absolute differences between actual and predicted demand values (Makridakis et al., 1983):

$$MAE = \sum_{i=1}^{n} \frac{|e_i|}{n} \tag{6}$$

MAE is straightforward to interpret since it expresses the typical deviation between forecasted and observed demand in the same units as the data, being the easiest to understand and compute (Hyndman, 2006). However, due to its dependency on scale, MAE cannot be compared between series. Also, in zero-heavy intermittent demand datasets, it tends to favor forecasting methods that will keep values close to zero but unable to predict a spike. For example, it penalizes ten errors of one unit the same as one error of ten units, even though missing a spike is critical for an intermittent behavior.

b. Root Mean Squared Error (RMSE)

The Root Mean Squared Error penalizes larger errors more heavily by squaring deviations before averaging, being calculated as (Makridakis et al., 1983):

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{e_i^2}{n}} \tag{7}$$

RMSE emphasizes severe forecast misses, such as unexpected spikes, and is particularly informative when evaluating models' responsiveness to extreme events. However, because it is sensitive to outliers, RMSE should be interpreted in conjunction with other measures.

2. Scale-free error metric

a. Mean Absolute Scaled Error (MASE)

According to Hyndman (2006), MASE is the more appropriate metric for intermittent demand data, making it possible to use it as a standard metric and to compare models across parts with different demand magnitudes. MASE is scale-independent and allows direct comparison between series. It is defined as the ratio between the model's MAE and the MAE of a naïve benchmark (Hyndman, 2006).

The naïve benchmark used in MASE assumes that the prediction for period t is simply the observed demand at $t - 1$. In other words, the naïve forecast is $\hat{y}_t naive = X_{t-1}$. The scaling factor in the MASE denominator is the in-sample Mean Absolute Error of this naïve method, computed as the average absolute difference between X_t and X_{t-1} over the historical series:

$$MASE = \frac{\sum_{i=1}^{n} \frac{|e_i|}{n}}{\sum_{i=2}^{n} \frac{|X_i - X_{i-1}|}{n-1}} \quad (8)$$

A MASE value lower than 1 indicates that the evaluated model outperforms the naïve forecast, while values above 1 suggest inferior performance (Hyndman & Koehler, 2006). Due to its robust application to intermittent data, MASE was used as the primary comparison metric.

3. Evaluation framework

The combination of these metrics offers a balanced perspective on forecasting performance:

- MAE and RMSE capture absolute and squared deviations, highlighting general accuracy and the impact of large errors.
- MASE allows scale-free comparison across parts with vastly different demand patterns, serving as the main quantitative indicator. In practice, models were compared primarily using MASE.

I. TOOLS AND IMPLEMENTATION

1. Python

The Python programming language was chosen for its versatility, open-source nature, and the extensive ecosystem of libraries supporting time series analysis and machine learning. All scripts were written in Python 3.11 and executed on macOS, with data and code organized under the *Thesis_Forecasting* GitHub repository to ensure reproducibility and traceability of results, and can be consulted in this thesis's Appendix. The following libraries were used:

- *pandas and NumPy* (Harris et al., 2020) – for data manipulation, numerical operations, and time series structuring;
- *statsmodels* (Seabold & Perktold, 2010) – for implementing statistical forecasting models such as Exponential Smoothing;
- *scikit-learn* (Buitinck et al., 2013) – for preprocessing utilities, model evaluation functions, and neural network implementation;
- *xgboost* (Chen & Guestrin, 2016) – for the Extreme Gradient Boosting model;
- *matplotlib* (Hunter, 2007) and *seaborn* (Waskom, 2021) – for graphical visualization of trends, errors, and comparative results.

Each script was designed with modularity in mind, allowing independent execution of key steps such as data cleaning, demand categorization, feature engineering, model training, and evaluation.

2. Microsoft Excel

Microsoft Excel was used as a supporting analytical tool during exploratory data analysis, allowing for visual inspection of demand series, aggregation of monthly statistics, and preliminary plotting of demand behavior across part numbers. Moreover, it also facilitated early verification of data integrity after preprocessing (e.g., checking for missing months, duplicated records, or extreme values), serving as an effective validation layer to ensure that the processed data were consistent with operational expectations.

3. Computational workflow and reproducibility

The research workflow followed a linear, reproducible structure organized into four main stages:

1. Data Preparation – Extraction from SILOMS, cleaning, zero-filling, and filtering based on spike thresholds
2. Feature Engineering – Generation of lagged variables, rolling statistics, calendar indicators, and spike-related features

3. Model Training and Validation – Implementation of traditional and machine learning models

4. Performance Evaluation and Reporting – Computation of statistical metrics and storage of outputs for analysis.

Finally, using GitHub version control ensured transparency in code evolution, while detailed commit messages documented adjustments made during model optimization.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS AND ANALYSIS

A. OVERVIEW

This chapter presents and interprets the forecasting results obtained from the models described in Chapter 3. The objective is comparing and evaluating the performance between traditional statistical methods and modern machine-learning approaches in predicting intermittent demand for aircraft spare parts used by the Brazilian Air Force (FAB).

Six models were compared:

(1) Moving Average (MA 24)

(2) Single Exponential Smoothing (SES)

(3) Croston-Syntetos-Boylan Approximation (SBA)

(4) Artificial Neural Network (NN)

(5) Extreme Gradient Boosting (XGBoost)

Each model performance was measured using the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Scaled Error (MASE), with the last one being the main metric, since it is the most appropriate to intermittent demand according to Hyndman (2006).

B. STATISTICAL ACCURACY COMPARISON

According to Makridakis & Hibon (2000), when performing error measurements across a relevant number of different series, the best approach is to calculate the errors for all aggregated values instead of averaging the errors for each individual series. This method prevents series with more extreme error values from influencing the final result.

For the comparison, all observations with a forecasted value were kept. This decision enables us to compare the number of observations required to train each model, while also simulating a real-world scenario where each model is trained and evaluated using the available data at that time. Each forecasting model in this study has different

minimum data requirements. Methods such as MA (24) require at least 24 historical months before they can generate a valid forecast, whereas SES, Croston/SBA and XGBoost have different requirements such as last spike or enough training window. For computational consistency and reproducibility, NN evaluation is restricted to a 24-month forecast across all items. Restricting evaluation to only the months where all models overlap would discard a large portion of the available information and artificially disadvantage models that are able to operate effectively with shorter histories.

For this reason, the evaluation uses all months in which each model legitimately produces a forecast, reflecting each method's actual behavior. Figure 11 shows how many observations each PN contributes to the evaluation, reflecting differences in data availability and model data requirements.

.

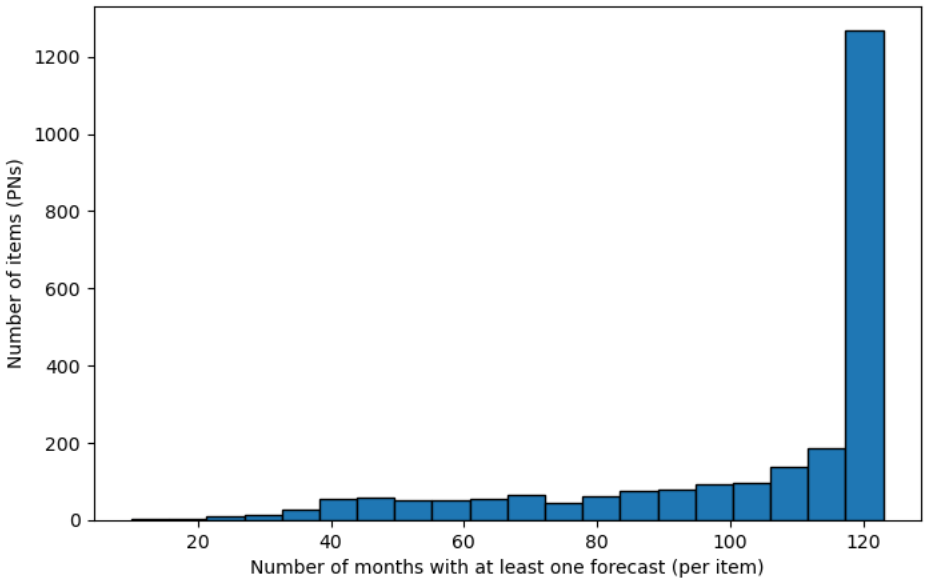


Figure 11. Distribution of the number of months per part number (PN) in which at least one forecasting model produced a valid forecast.

Therefore, Table 1 summarizes the aggregate results across all 2,428 evaluated part numbers (PNs).

Table 1. Model Performance Comparison (2,428 PNs)

Model	number of forecasts	MAE	RMSE	MASE
MA 24	196,553	3.97	46.75	1.00
SES	252,348	4.25	55.98	0.99
Croston–SBA	236,482	6.81	70.69	1.55
Neural Network	57,939	3.53	48.01	1.04
XGBoost	167,661	3.60	50.67	0.89

The XGBoost model achieved the lowest MASE value in the comparison, as shown in Table 1. The value of 0.89 means that the model achieved an average error approximately 11% lower compared to a naïve forecast, i.e., repeating the last month's demand as the forecasted value for the next period. Moreover, XGBoost reduced the mean absolute scaled error by approximately 10 % compared to the best traditional statistical model result (SES with MASE of 0.99), showing a clear improvement in predictive accuracy.

The statistical models, MA and SES have accuracy similar to the achieved naïve model (MASE ≈ 1), showing that they are not appropriate for an intermittent pattern. Additionally, the impossibility of adjustments in classical models prevents them from improving and adapting to the behavior of these series.

Surprisingly, the Croston-SBA method, despite being designed for intermittent demand, exhibited the highest errors, reflecting its sensitivity to long zero-demand stretches common in the FAB's dataset.

In addition to global averages, the analysis also examined accuracy at the individual part numbers level in Table 2. For each PN, the model with the lowest MASE was identified. XGBoost achieved the best result for approximately 55.6 % of all evaluated parts, demonstrating not only an overall but a clear advantage across different intermittent demand behaviors.

Table 2. Individual performance comparison on individual PNs

Model	Lowest MASE by PN	%
MA 24	144	5.93%
SES	644	26.52%
Croston–SBA	210	8.65%
Neural Network	79	3.25%
XGBoost	1351	55.64%

To examine whether model performance varies with demand volume, parts were classified into low, medium, and high-demand categories based on their average monthly demand. Table 3 shows the distribution of mean monthly demand across items was divided into tertiles: items in the lowest third were labeled low-demand, those in the middle third medium-demand, and those in the top third high-demand.

Table 3. Individual performance comparison by demand volume

	Lowest MSE by PN					
Model	Low demand	%	Medium demand	%	High demand	%
MA 24	18	2.2%	66	8.2%	60	7.3%
SES	159	19.9%	249	31.1%	236	28.6%
Croston–SBA	29	3.6%	72	9.0%	109	13.2%
Neural Network	0	0.0%	0	0.0%	79	9.6%
XGBoost	595	74.3%	414	51.7%	342	41.4%

This result indicates that XGBoost model is appropriate for a wide variety of intermittent demand patterns (low-demand, medium-demand, or high-demand).

The dispersion of the Mean Absolute Scaled Error (MASE) for each forecasting model was also analyzed. Table 4 shows the median and Interquartile Range (IQR) of MASE across all PNs.

Table 4. MASE dispersion across all PNs by model

Model	25th percentile	Median	75th percentile
MA 24	0.95	1.03	1.22
SES	0.93	0.97	1.04
Croston–SBA	0.96	1.31	2.02
Neural Network	1.18	1.97	4.07
XGBoost	0.50	0.71	1.03

XGBoost achieved the lowest median MASE (0.71) with an IQR of 0.50-1.04, demonstrating stability and consistent improvements relative to the naïve forecast in most cases, which is essential for reproducibility. The SES model presented the best result among the statistical methods, with a stable median of 0.97 and a narrow interquartile range (IQR) of 0.93–1.04 but showed only low accuracy gains. MA produced medians close to the naïve benchmark, while the Croston–SBA method presented the highest median (1.32) and wider dispersion, reaffirming its limited suitability for the long zero-demand sequences present in this dataset. Finally, the neural network model presented the largest median error (1.97) and the widest IQR (1.18–4.08), consistent with its sensitivity to limited data per series.

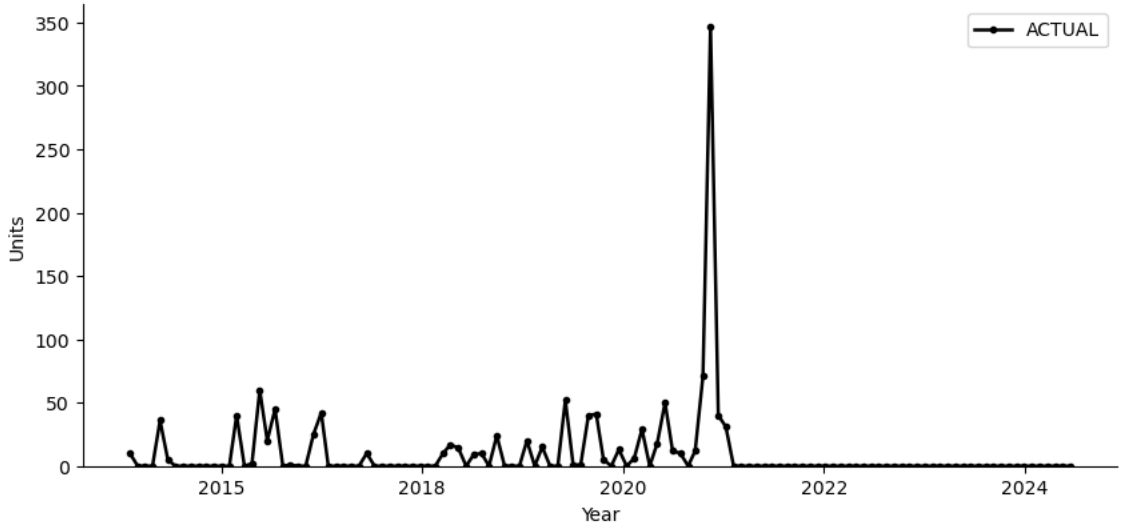
Lastly, the results obtained in Tables 1, 2, 3 and 4 provide robust comparisons of models accuracy in intermittent demand contexts. While the aggregated results (Table 1) capture the general model performance across the dataset, weighted by demand volume, the individual performance (Tables 2 and 3) and the median and IQR (Table 4) show how each model performs for each specific PN. In this context, the quartile values presented in Table 4 help interpret how errors are distributed across items. For example, the first quartile indicates that 25% of the parts achieved a MASE $\leq$ 0.95 using the MA (24) method, whereas, for XGBoost, 25% of the parts reached a much lower MASE $\leq$ 0.50.

If analyzed by column, Table 4 shows that the median represents the regular performance for each model. At the same time, the third quartile shows the point below which 75% of the parts fall. Therefore, it becomes clear that XGBoost has consistently lower MASE than other models. Together, these complementary results demonstrate that XGBoost consistently delivers the best balance between accuracy and robustness across

the fleet of parts, while statistical baselines, such as SES, remain close to the naïve benchmark.

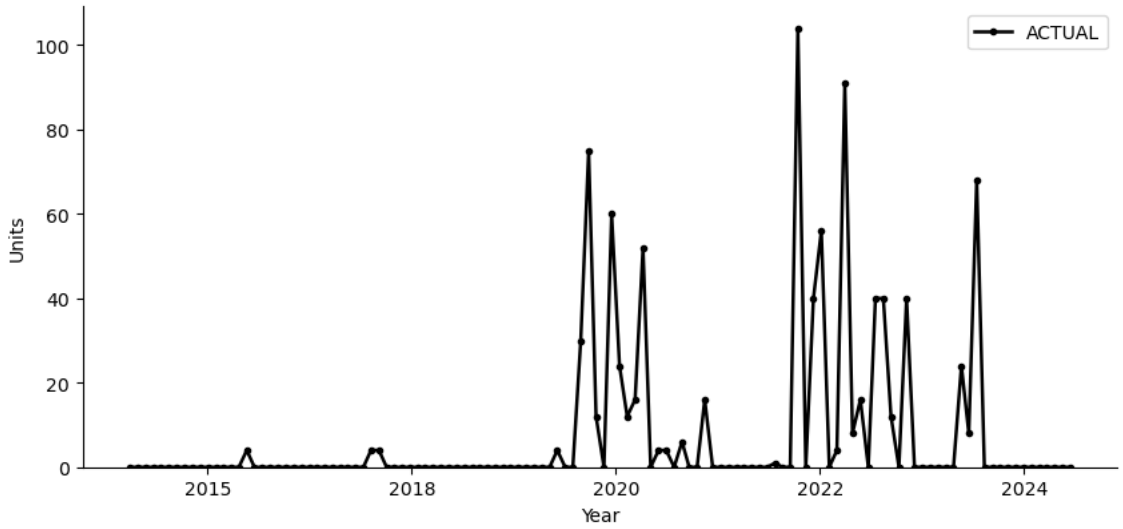
C. PATTERNS IN FORECAST ERRORS

While the error measures provide summary metrics to evaluate forecast accuracy, this section visualizes and analyzes how each model performs for specific demand patterns and examines the reasons behind the results presented in Tables 1, 2, 3 and 4. To illustrate the main behaviors observed, five representative PNs were chosen, introducing the following patterns: rare extreme spikes, intermittent bursts, lightly intermittent demand, high-variance series, and declining demand, shown in Figures 12 to 16. These categories were identified through a systematic visual inspection of all time series, looking for recurrent structural characteristics such as long sequences of zeros, isolated large spikes, alternating periods of bursts and inactivity, sustained volatility, or clear downward trends. This exploratory review allowed the selection of PNs that exemplify the dominant shapes present in the dataset. They will be referred to as PN A, B, C, D, and E to preserve sensitive information.



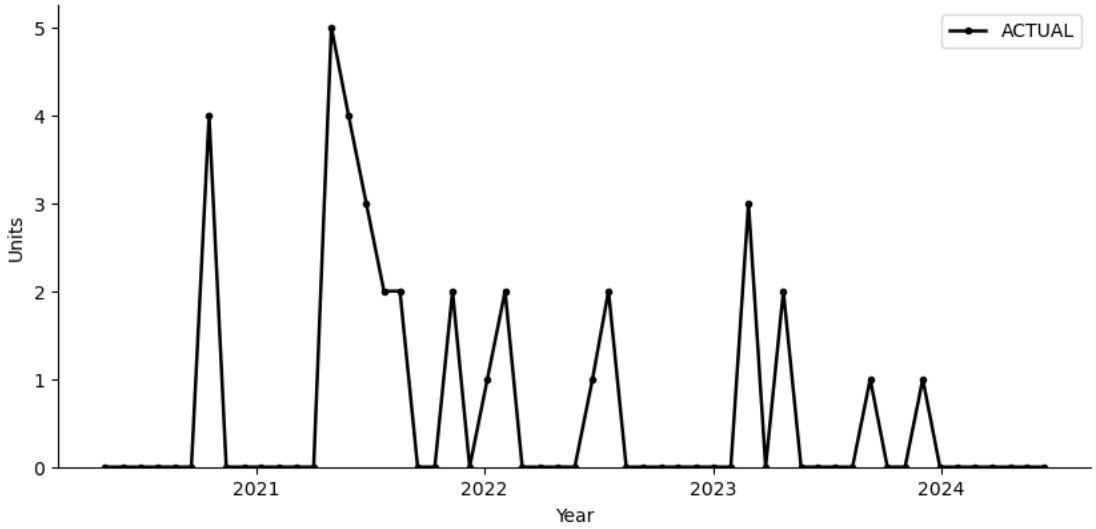
Isolated, High-volume event followed by long inactivity

Figure 12. PN A – Rare extreme Spike



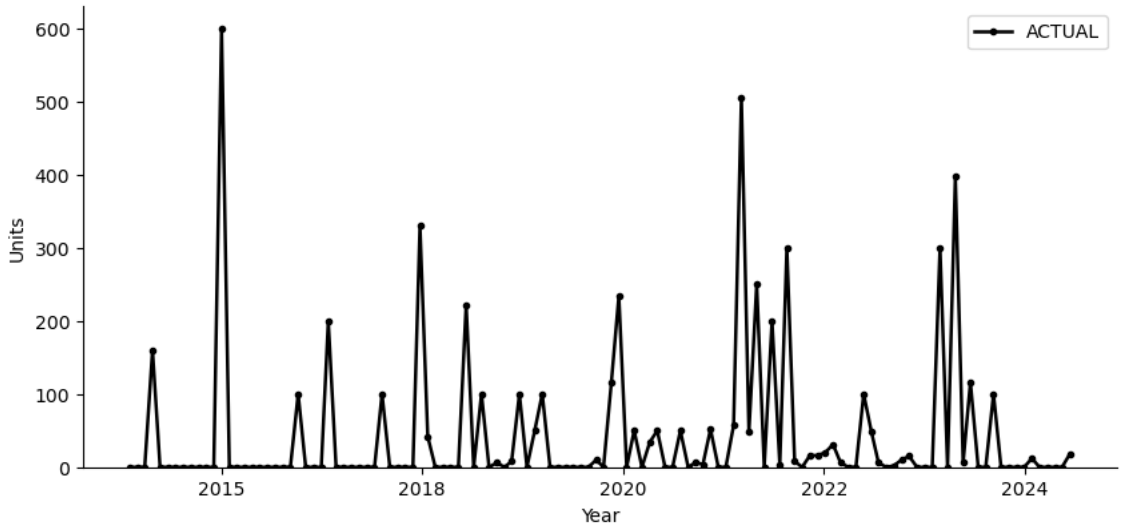
Clustered short episodes of demand separated by gaps

Figure 13. PN B – Intermittent bursts



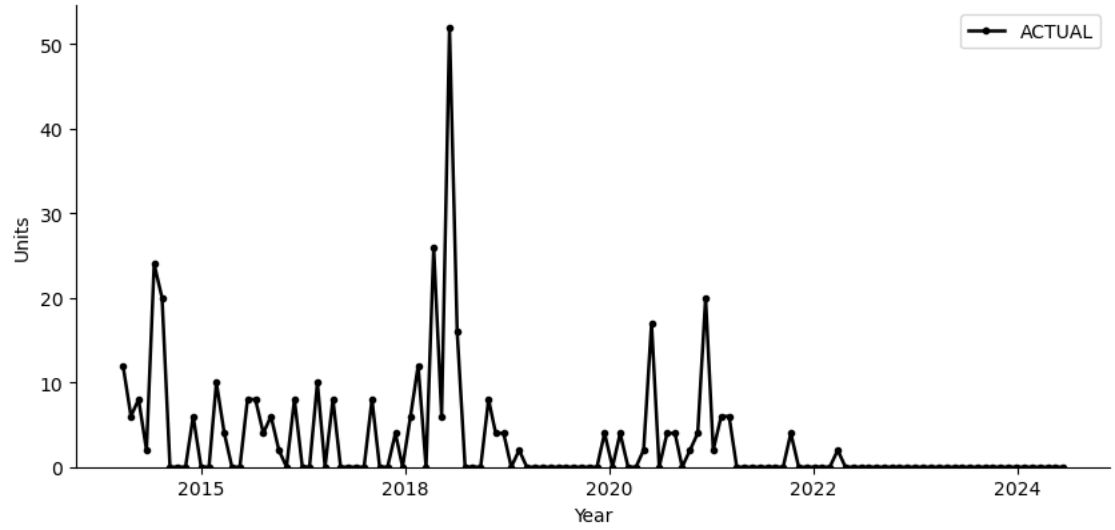
Irregular but moderately frequent demand

Figure 14. PN C – lightly intermittent demand



Continuous activity with strong amplitude fluctuations

Figure 15. PN D – High-variance series



Decreasing demand over time until obsolescence

Figure 16. PN E – Declining demand

1. Moving average

The moving average for the 24-month model used by the Brazilian Air Force (FAB) is constrained by its long averaging window, causing it to have slow reaction to changes in the demand pattern, producing a forecast that stays almost constant during the whole period. This behavior is expected because all past observations are weighted equally, diluting the effect of sudden variations.

As a result, while MA provides stable forecasts, it reacts too slowly to sudden demand surges, as occurs in the rare spike pattern and intermittent burst (Figures 17 and 18).

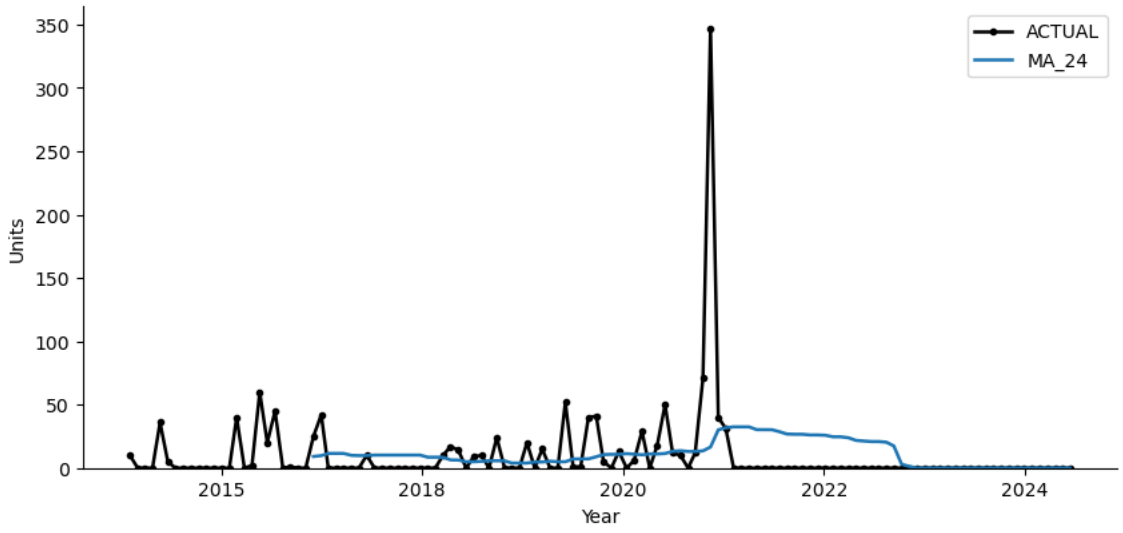


Figure 17. Moving average forecasting for demand with rare extreme spikes

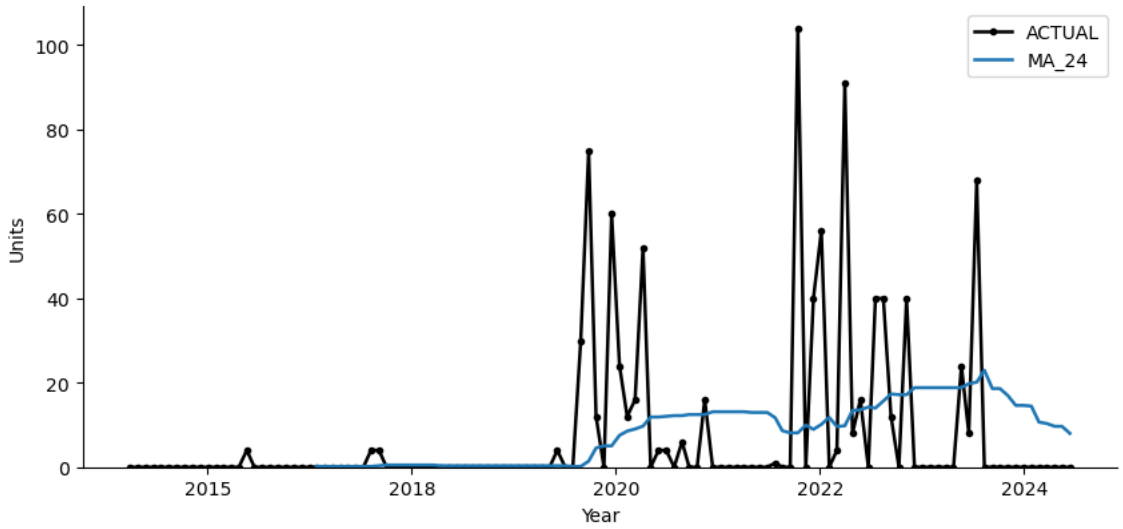


Figure 18. Moving average forecasting for demand with intermittent burst pattern

This same slow reaction results in overestimation in lightly intermittent and declining series (Figures 19 and 21), since it prolongs elevated historic demand.

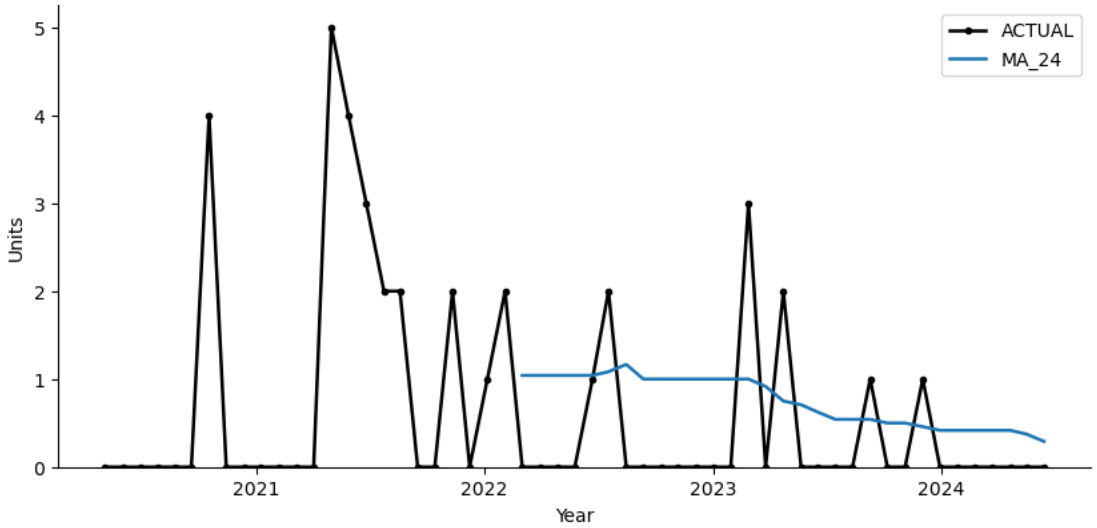


Figure 19. Moving average forecasting for demand with lightly intermittent pattern

Finally, for high-variance scenarios (Figure 20), successful smooths extreme fluctuations, however it cannot predict actual spikes.

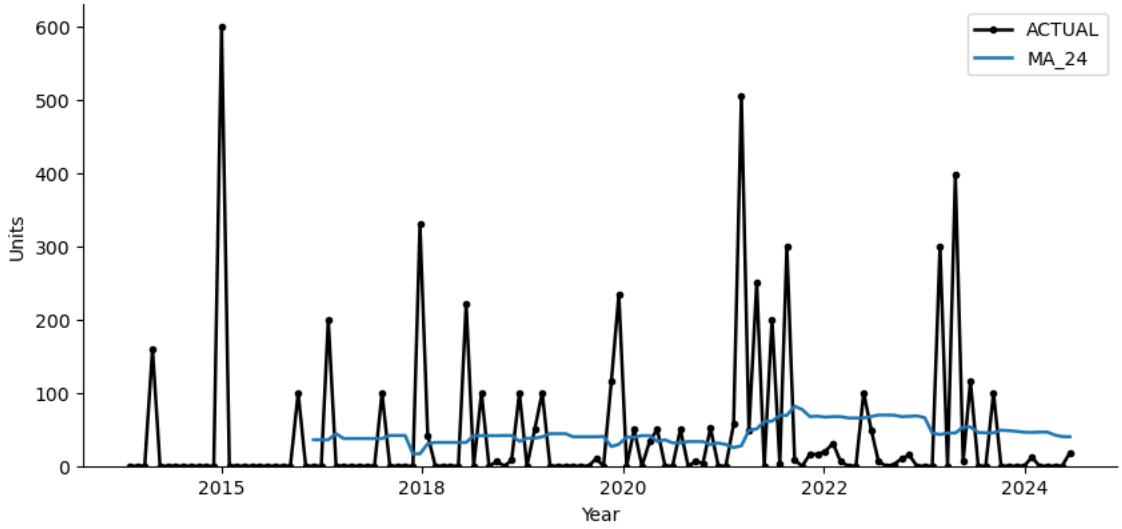


Figure 20. Moving average forecasting for demand with high variance

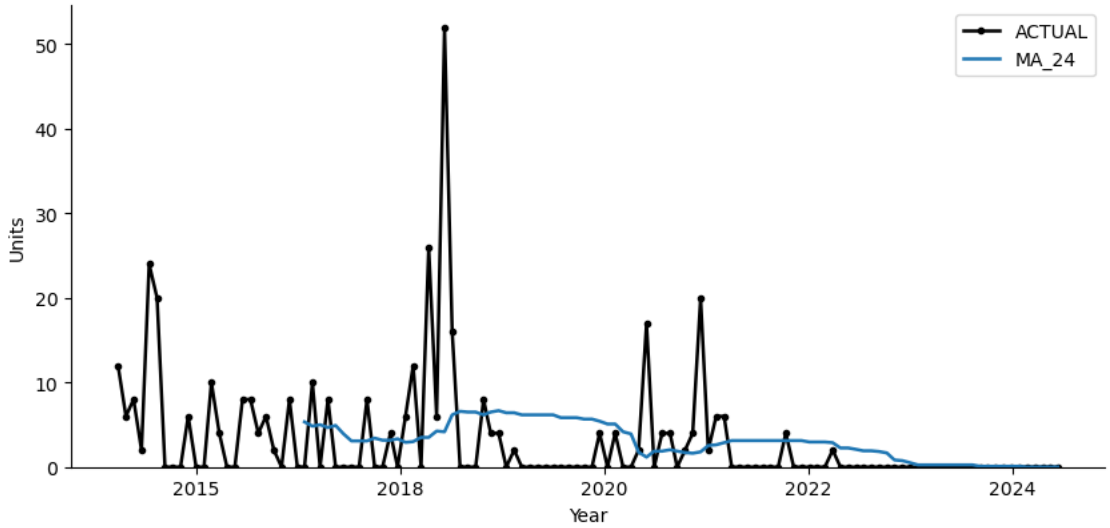


Figure 21. Moving average forecasting for declining demand

Since forecasting accuracy directly affects spare parts availability, these wrong predictions may translate into operational consequences, such as the temporary grounding

of aircraft, delayed maintenance actions, and reductions in fleet readiness levels. Because the FAB relies on timely replenishment to support both scheduled and unscheduled maintenance, a method that reacts too slowly to demand changes increases the likelihood of stockouts during sudden demand surges.

2. Simple exponential smoothing

The SES model responds faster than the moving average, as it applies exponentially decreasing weights to older observations, reacting more to recent data. However, even with the smoothing parameter optimized for each individual series, staying between 0.1 and 0.3, this model still suffers from the same difficulties as the MA in capturing sudden demand.

For rare extreme spikes and intermittent bursts (Figure 22 and 23), although it captures the increase caused by the abrupt demand, it tends to generate a smoothed, partial response rather than a true spike, and this response typically comes after the spike has already occurred, meaning the model is essentially forecasting yesterday's shock instead of anticipating tomorrow's.

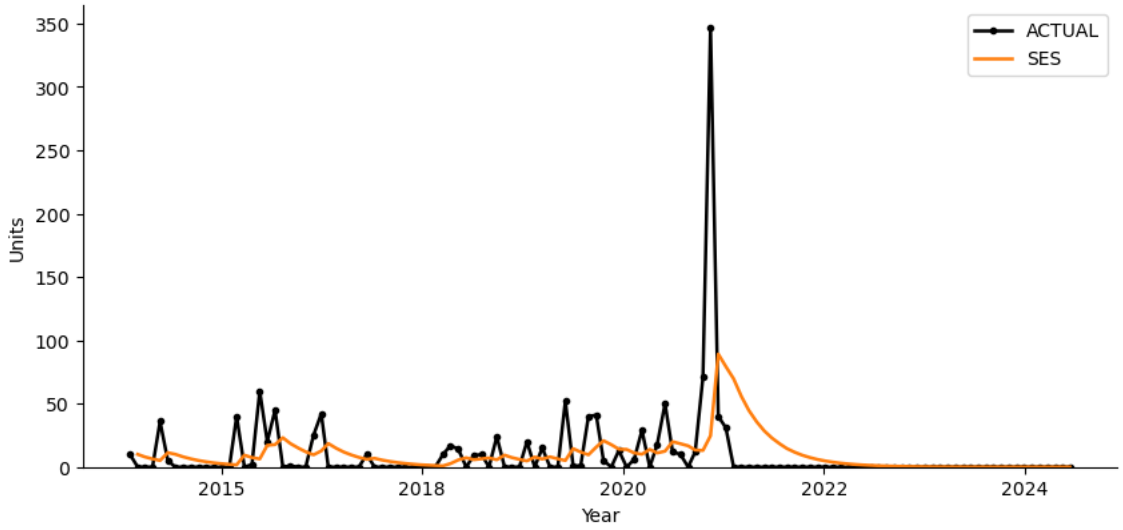


Figure 22. Simple exponential smoothing forecasting for demand with rare extreme spikes

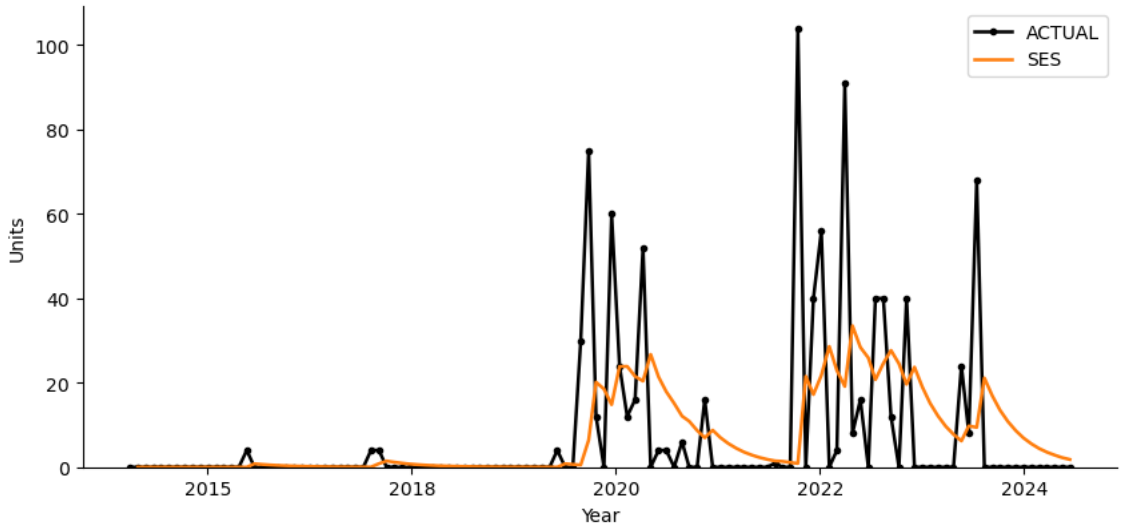


Figure 23. Simple exponential smoothing forecasting for demand with intermittent burst pattern

On the other hand, for lightly intermittent and declining demand behaviors (Figures 24 and 26), it produces a more realistic output than MA, decreasing faster towards zero, even though persisting forecasting demand during a period.

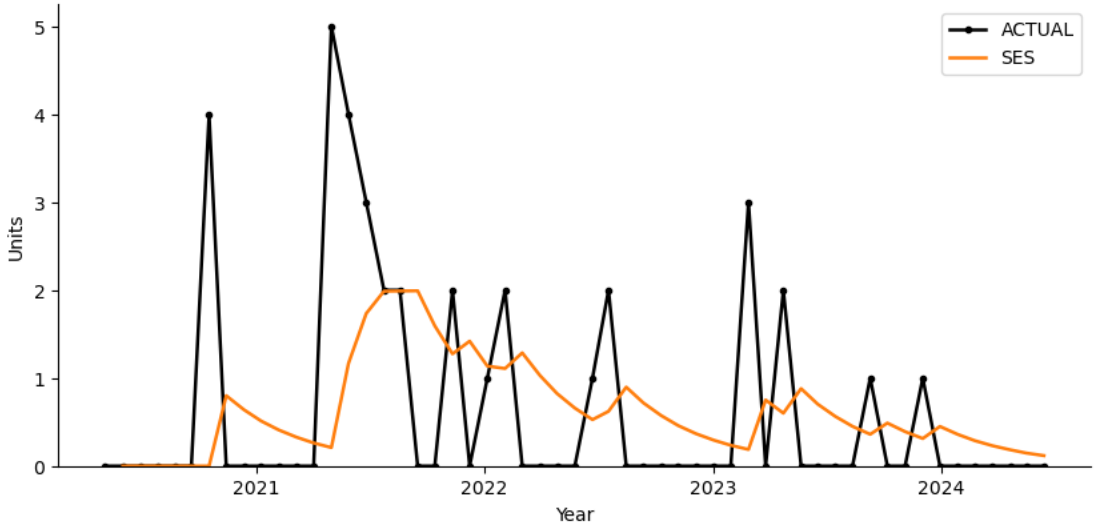


Figure 24. Simple exponential smoothing forecasting for demand with lightly intermittent pattern

For high-variance series (Figure 25), SES smooths the noise but fails in capturing the fast oscillations in the pattern.

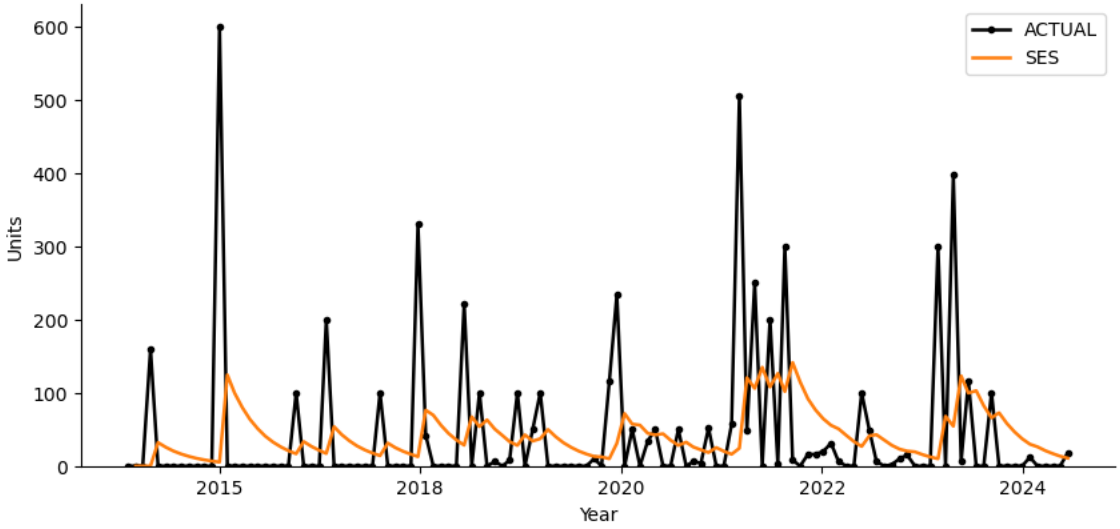


Figure 25. Simple exponential smoothing forecasting for demand with high variance

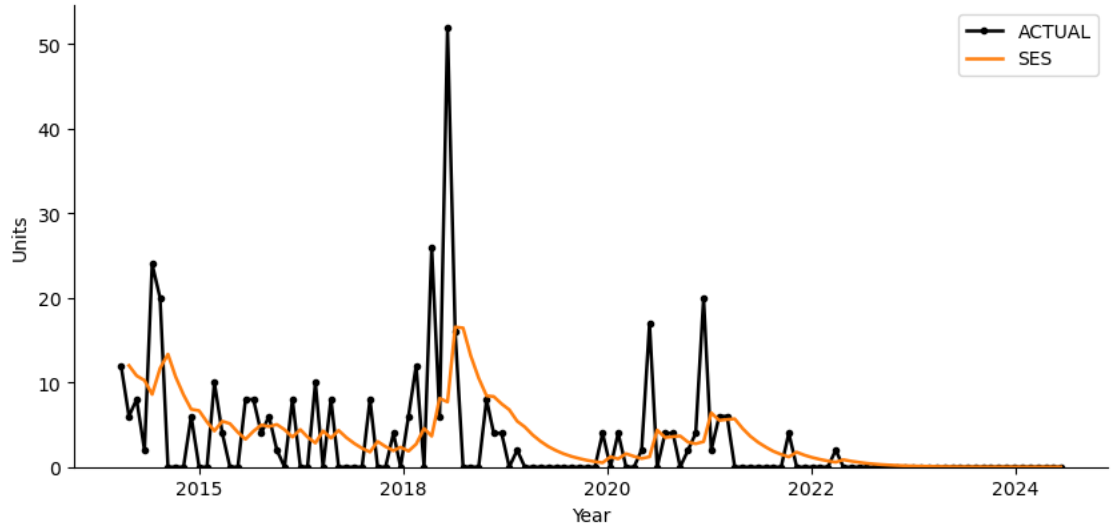


Figure 26. Simple exponential smoothing forecasting for demand with declining demand

Overall, SES produces forecasts balanced between reactiveness and smoothness, while struggling with sparse or volatile data.

3. Croston Syntetos-Boylan Approximation (Croston SBA)

Although Croston's method with Syntetos–Boylan Approximation (SBA) is a classical approach for intermittent series, in this dataset, this model behaved overly rigidly. Across all five patterns (Figures 27 to 31), the forecasts converge to an almost constant level, insensitive to sudden spikes or declines. Especially for declining demand series (Figure 31), since the new forecast is only calculated upon the occurrence of a new spike, Croston SBA continues projecting positive demand long after the last spike. This demonstrates how the Croston SBA method is more appropriate for stationary intermittent series with no irregularities.

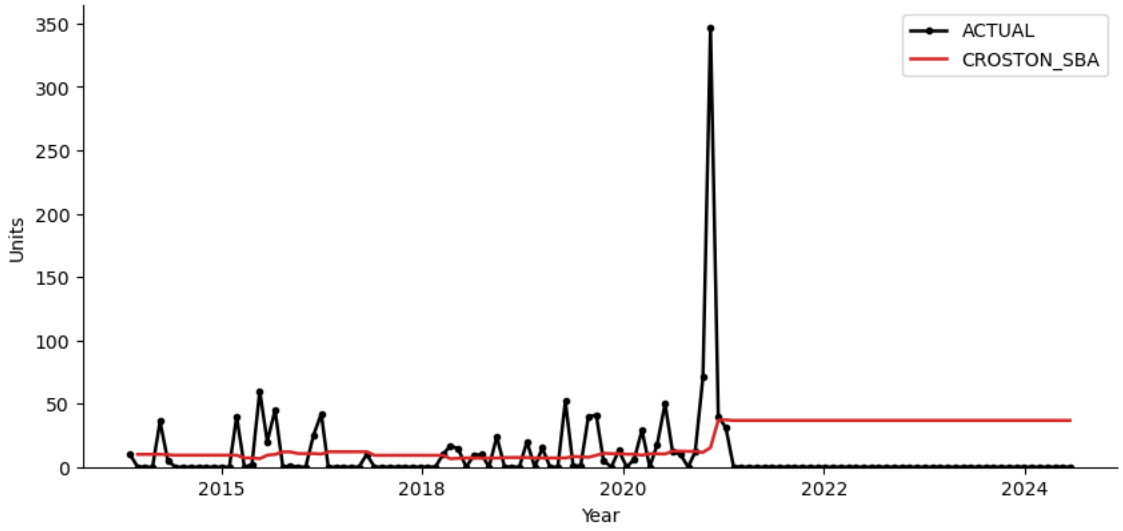


Figure 27. Croston SBA forecasting for demand with rare extreme spikes

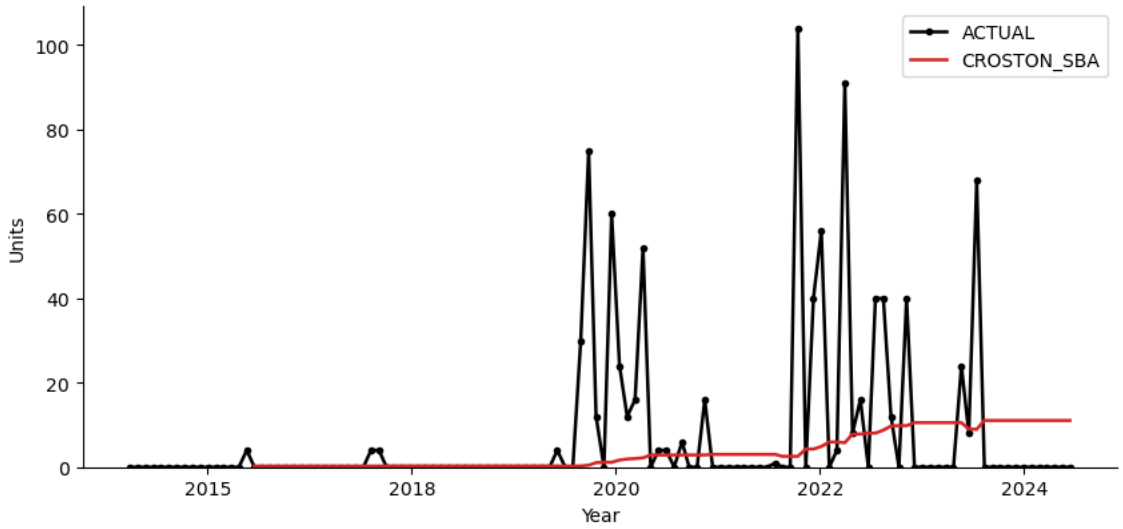


Figure 28. Croston SBA forecasting for demand with intermittent bursts pattern

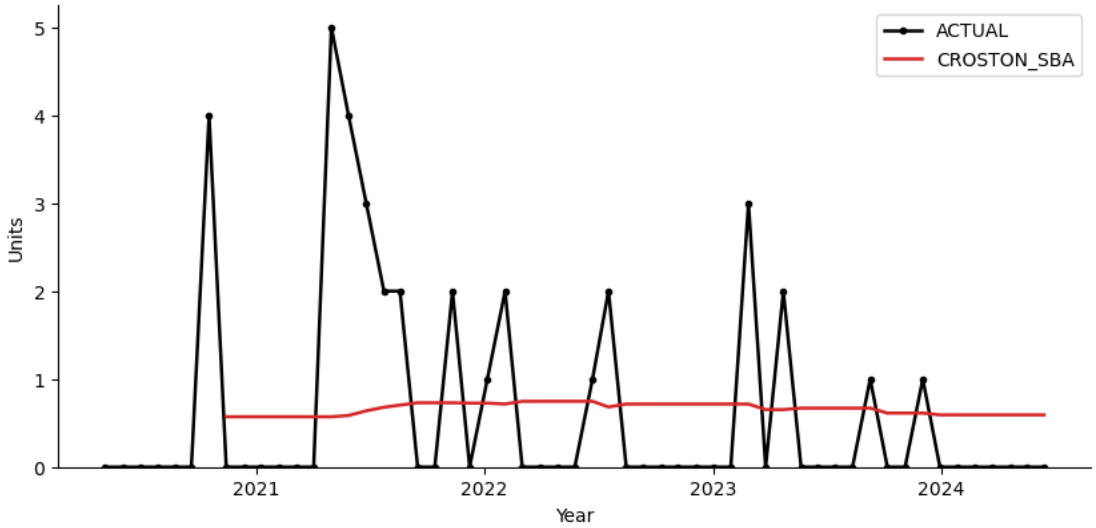


Figure 29. Croston SBA forecasting for demand with lightly intermittent pattern

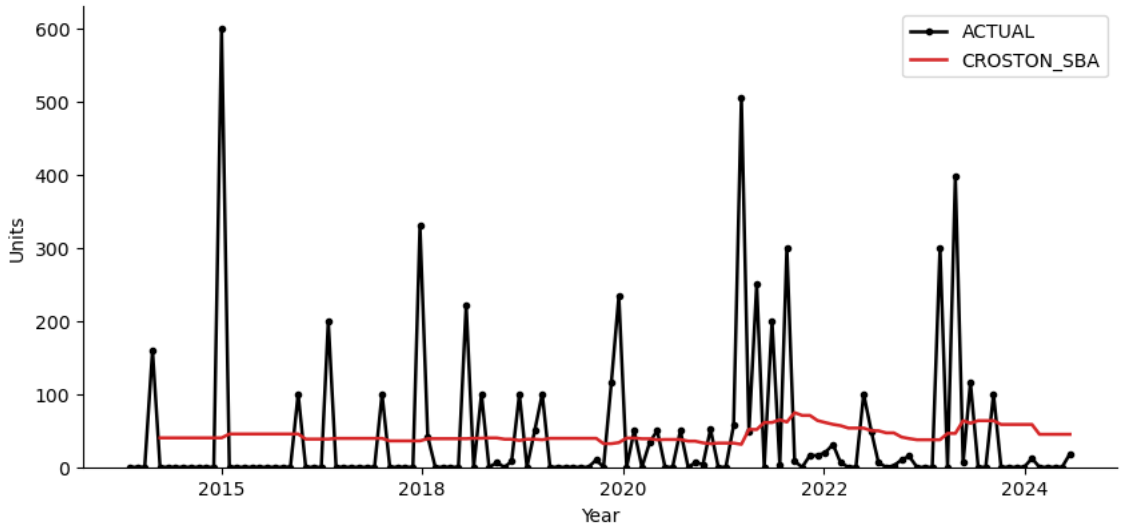


Figure 30. Croston SBA forecasting for demand with high variance

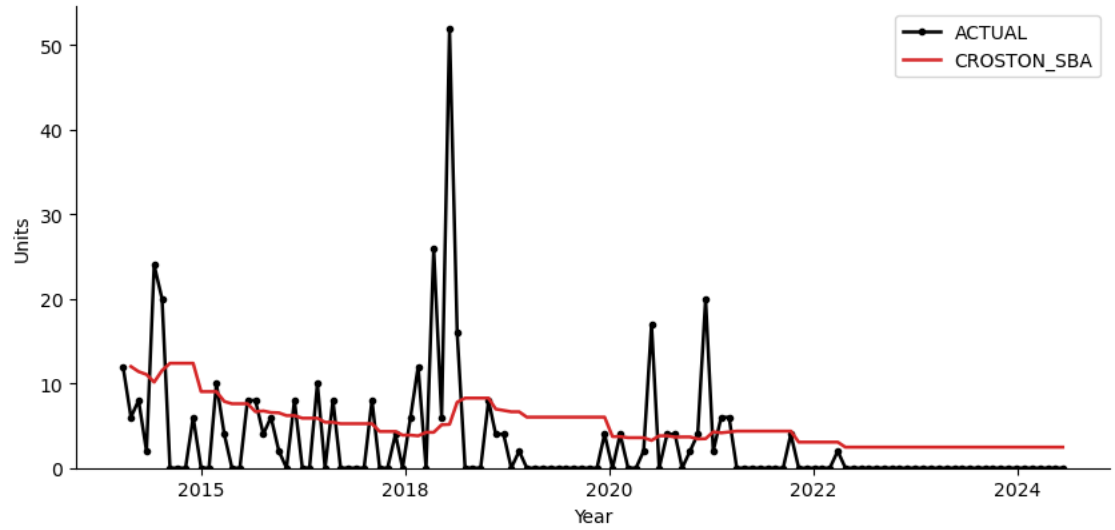
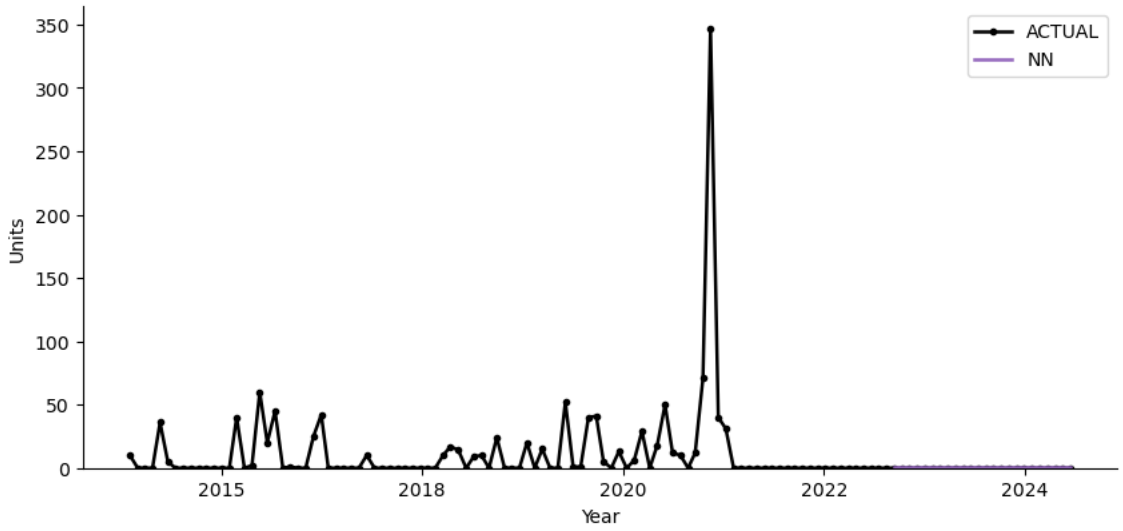


Figure 31. Croston SBA forecasting for declining demand

4. Artificial Neural Network (NN)

The artificial neural network forecasts show low reactions across all demand patterns. This may reflect the global training the model received, where it learned the dominant feature of the dataset: long sequences of zeros. Therefore, in the rare extreme and declining consumption series (Figures 32 and 36), it predicts demand near zero, failing to recognize variations in the data. For the other patterns (Figure 32, 33, and 34), NN

generalizes forecasting around a global mean, failing to predict spikes and periods of zero demand.



NN forecasted 0 demand for the period due to the long zero demand period

Figure 32. NN forecasting for demand with rare extreme spikes

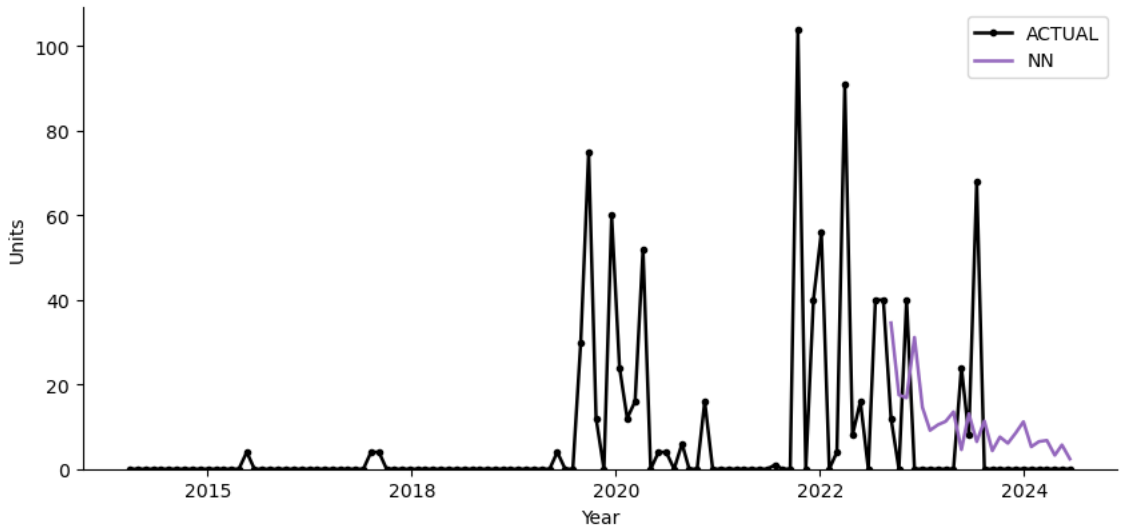


Figure 33. NN forecasting for demand with intermittent bursts pattern

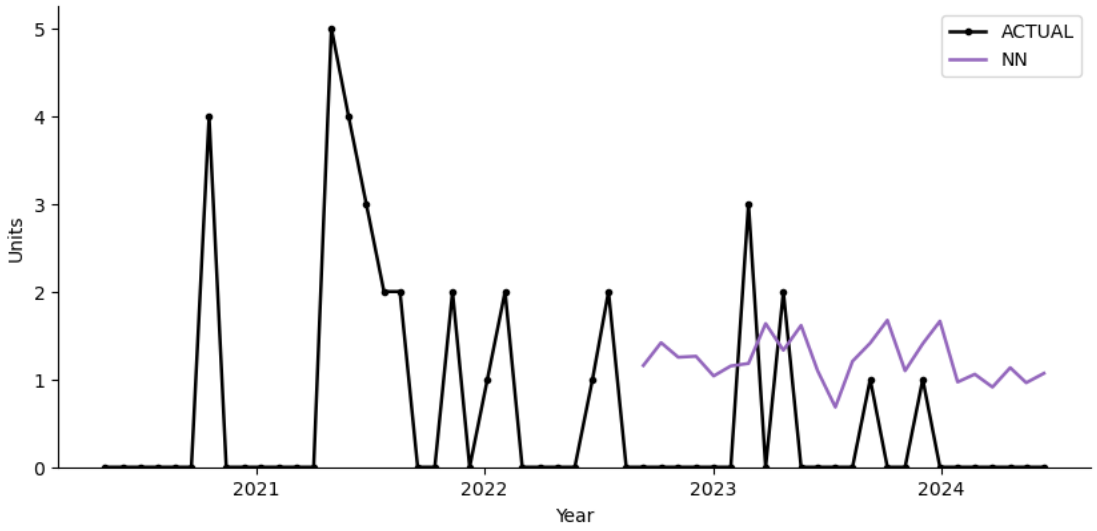


Figure 34. NN forecasting for demand with lightly intermittent pattern

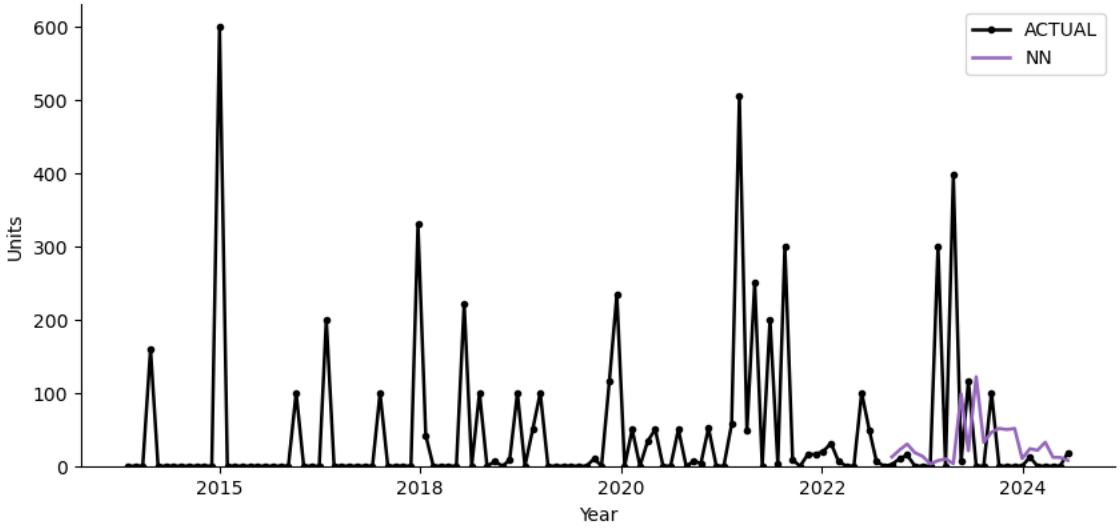


Figure 35. NN forecasting for demand with high variance

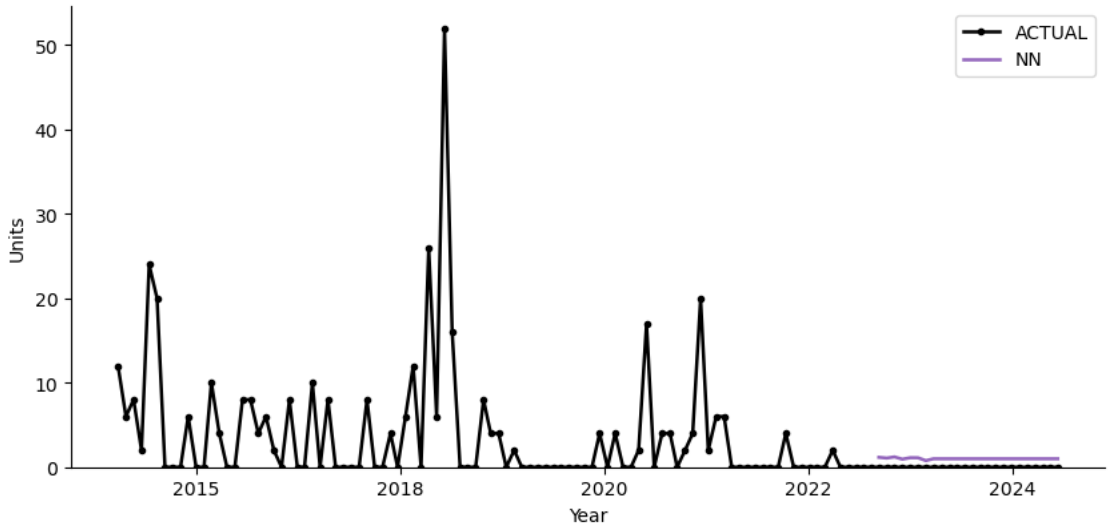


Figure 36. NN forecasting for declining demand

5. XGBoost

The XGBoost model shows the highest adaptability across all five patterns. It is possible to see how the model, based on the routine learned during training, attempts to predict the next spike during long zero-demand periods and, at the same time, quickly reverts to zero, avoiding over forecasting.

In the rare spike series (Figure 37), it detects spikes and forecasts peaks. At the same time, although it fails to detect the sudden spike in time, it rapidly reverts the forecast back to zero.

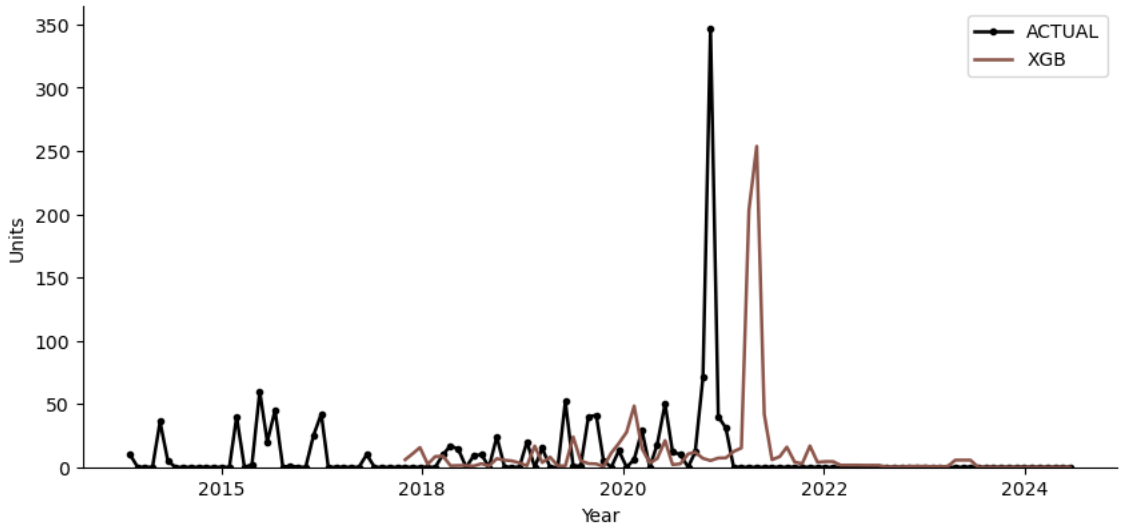


Figure 37. XGBoost forecasting for demand with rare extreme spikes

Similarly for the remaining patterns (Figures 38 to 41), XGBoost succeeds in following demand surges, downward trends and even sudden changes in volume.

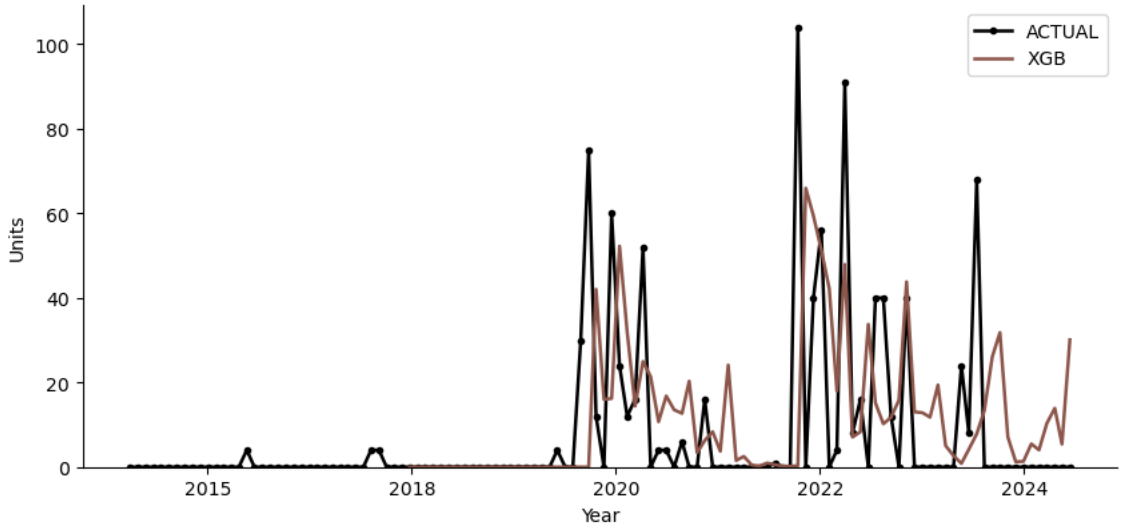


Figure 38. XGBoost forecasting for demand with intermittent bursts pattern

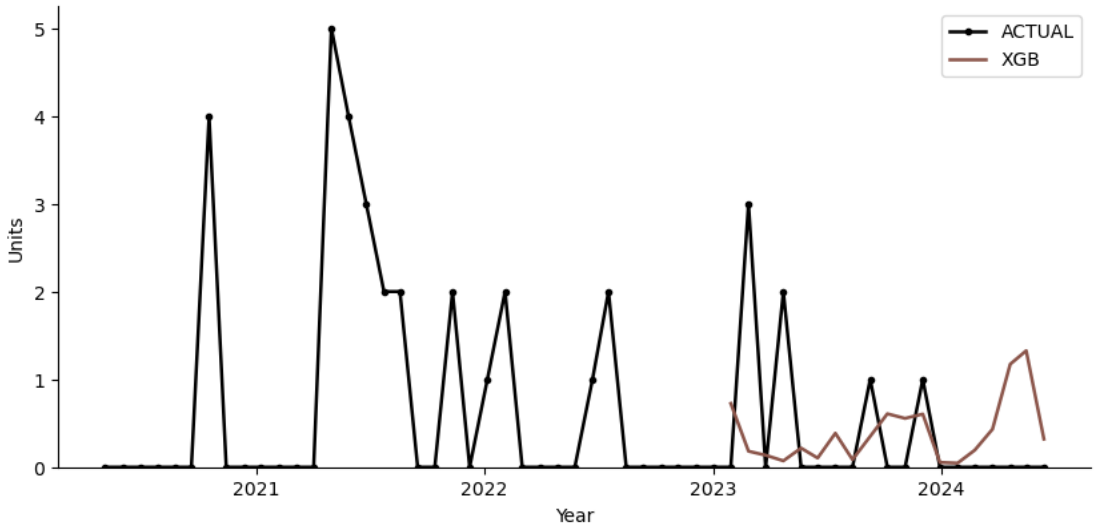


Figure 39. XGBoost forecasting for demand with lightly intermittent pattern

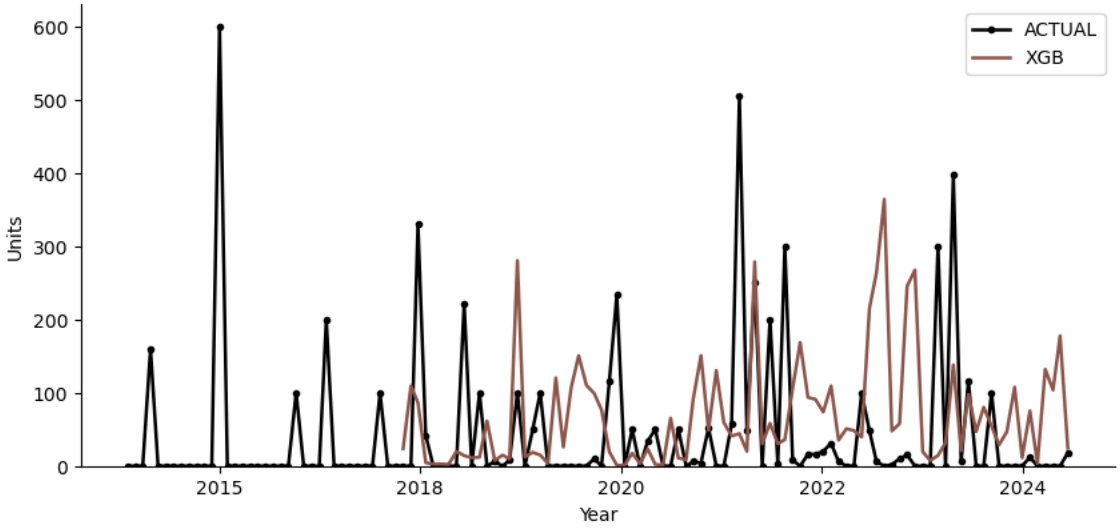


Figure 40. XGBoost forecasting for demand with high variance

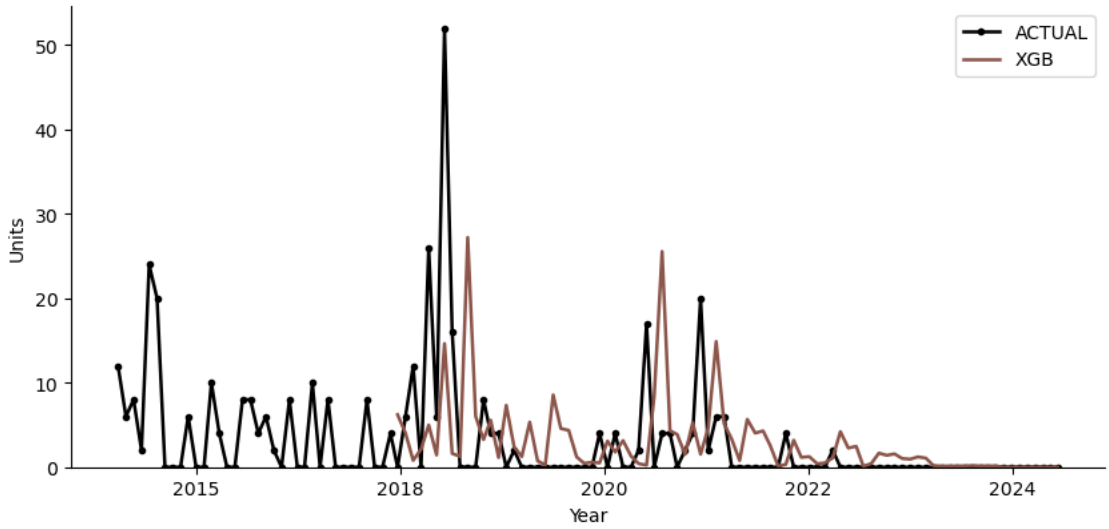


Figure 41. XGBoost forecasting for declining demand

A characteristic of the XGBoost model is its ability to predict spikes even when no recent spike has occurred. This behavior comes from the model's capacity to identify combinations of lag patterns that historically preceded abrupt increases somewhere else in the dataset. When these same conditions reappear for a given PN, the model can anticipate a spike before it happens, rather than reacting only after the spike has already occurred. This makes XGBoost different from the statistical benchmarks, which are designed to smooth shocks rather than project them forward.

This mechanism helps explain why XGBoost was the only method that projected a spike in 2024. While this flexibility gives XGBoost the ability to anticipate spikes, it also may cause over-forecasting if the spike does not materialize. Even so, the overall results show that this capacity to capture patterns provides meaningful forecasting gains in an environment dominated by intermittent demand.

6. Model Comparison

The comparative visualization (Figures 42 to 46) highlights the strengths and weaknesses of each model. MA and SES are stable but react slowly to the irregularities in the series; Croston–SBA forecasts a practically constant demand, and the neural network fails to adapt to non-recurring events, resulting in recurrent underprediction. In contrast,

XGBoost balances responsiveness and stability, producing forecasts that follow actual demand dynamics more closely while maintaining zero forecasts during inactivity.

These visual findings reinforce the quantitative results from Tables 1 to 4, where XGBoost achieved the lowest overall MASE, median MASE and outperformed classical baselines in approximately 55 % of the evaluated part numbers.

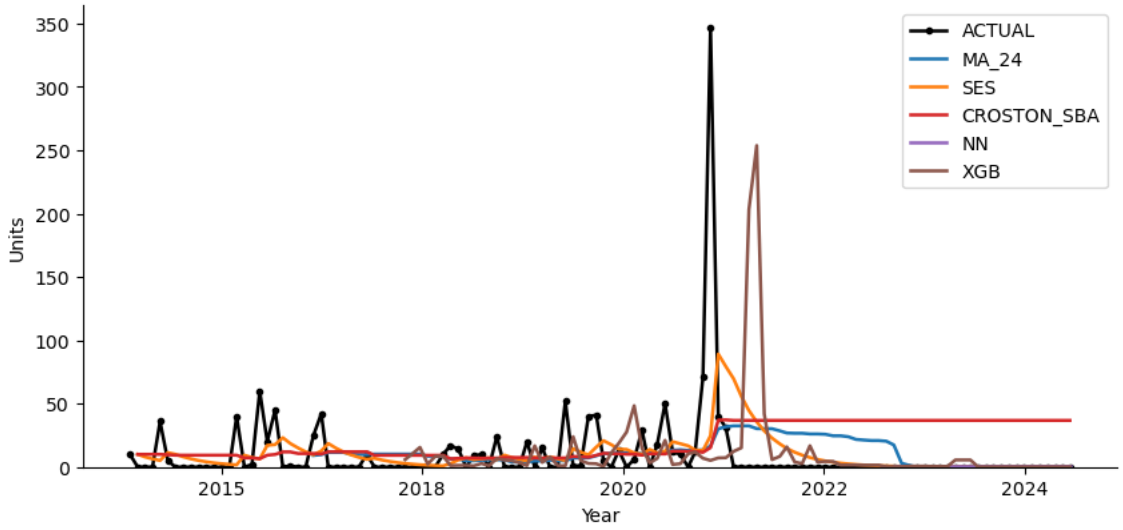


Figure 42. Models forecasting comparison for demand with rare extreme spikes

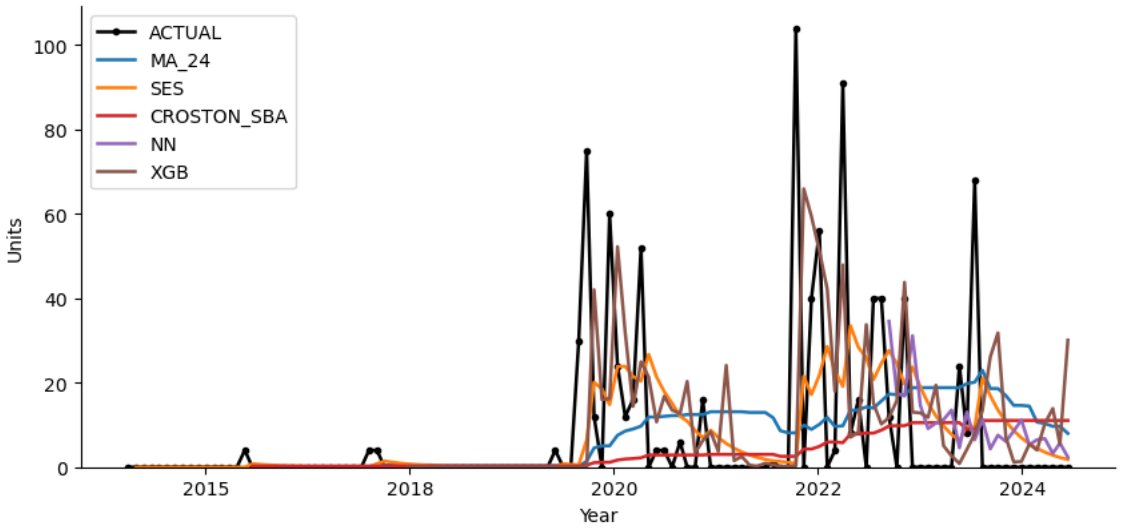


Figure 43. Models forecasting comparison for intermittent bursts pattern

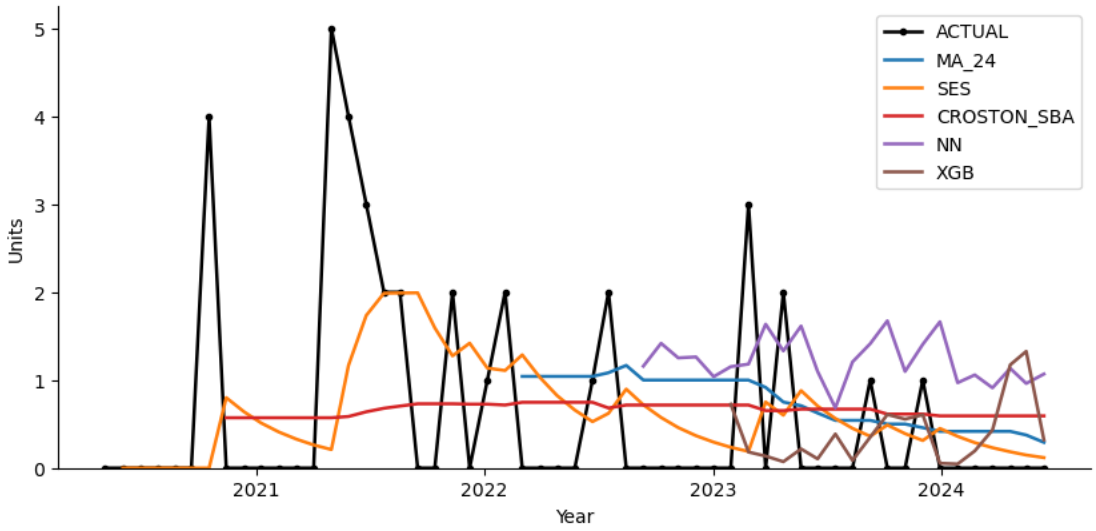


Figure 44. Models forecasting comparison for demand with lightly intermittent pattern

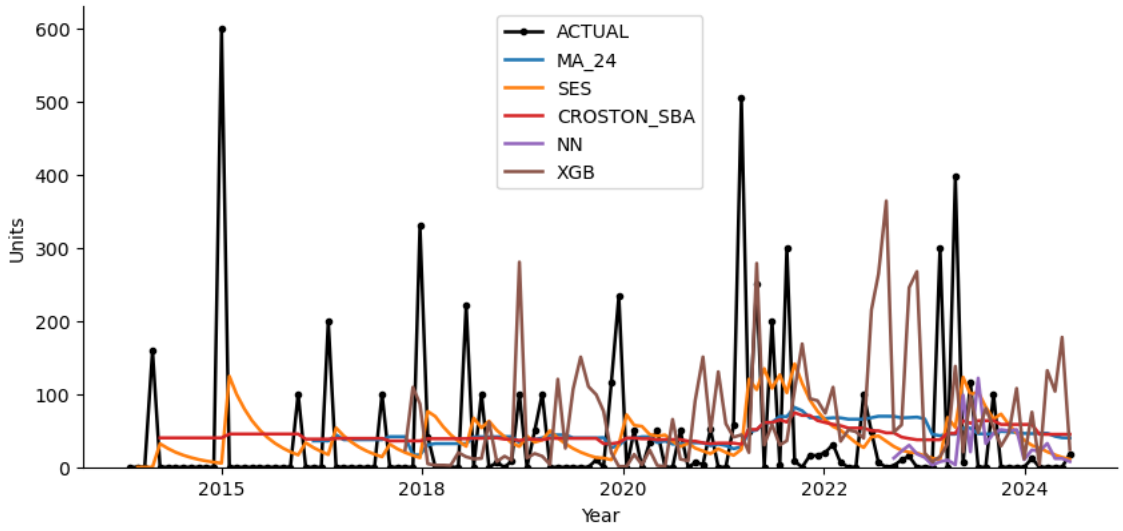


Figure 45. Models forecasting comparison for demand with high variance

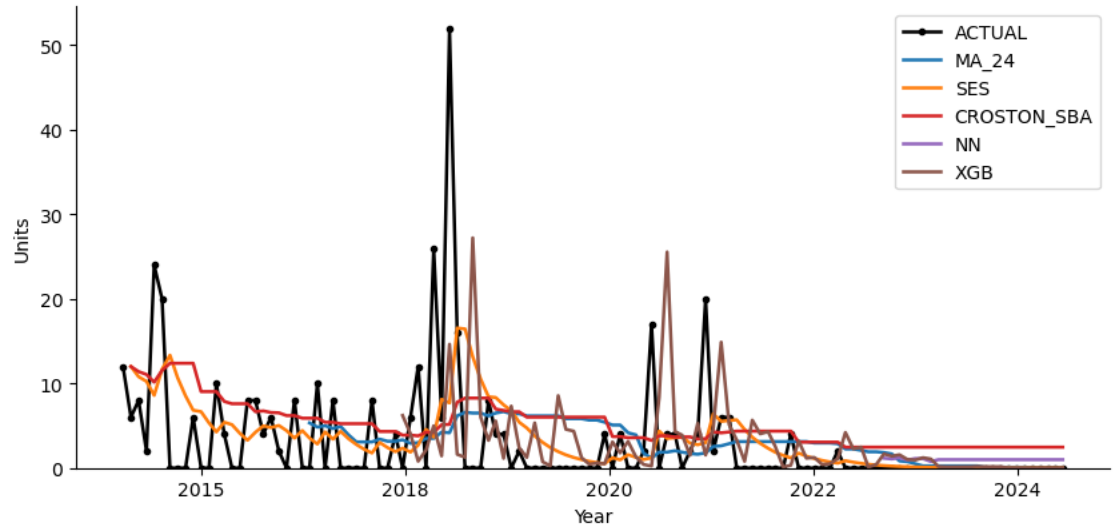


Figure 46. Models forecasting comparison for declining demand

D. SUMMARY

Across all evaluated configurations:

- XGBoost achieved the lowest global errors (MASE = 0.89),
- Outperformed traditional statistical models by roughly 11 %, and
- Provided the best forecast for 55.6 % of parts.

These results reaffirm the effectiveness of feature-based machine learning for intermittent demand forecasting in defense logistics and highlight its potential to enhance data-driven decision-making within FAB's inventory management processes.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

A. SUMMARY OF FINDINGS

This study evaluated five forecasting models, Moving Average (MA 24), Single Exponential Smoothing (SES), Croston-Syntetos-Boylan Approximation (SBA), Artificial Neural Network (NN), and Extreme Gradient Boosting (XGBoost), to predict the intermittent demand of T-27 TUCANO aircraft fleet spare parts within the Brazilian Air Force (FAB).

Among the tested approaches, XGBoost achieved the lowest mean and median and outperformed the baseline methods in approximately 55.6% of all evaluated parts.

Its multivariate structure, ability to capture nonlinear dependencies, and robustness under data sparsity allowed it to model both the occurrence and magnitude of sporadic demands more effectively than classical univariate techniques. These results confirm the suitability of machine-learning approaches for forecasting intermittent demand in defense logistics.

B. THEORETICAL CONTRIBUTIONS

From a theoretical perspective, this research contributes to the growing body of work that extends intermittent demand forecasting beyond the traditional Croston family of models.

The findings reinforce that:

- Univariate statistical models (e.g., MA, SES) are limited when the stochastic structure of demand violates continuity and stationarity assumptions.
- Feature-based machine learning provides a more general framework, where predictors such as recency, variability, and seasonality are explicitly modeled.

- The two-stage XGBoost architecture (classification + regression) effectively decomposes the problem into "demand occurrence" and "demand size," conceptually similar to the theoretical split proposed by Croston (1972) but implemented through data-driven learning.

The study also illustrates how pooled error metrics (MAE, RMSE, and MASE) can be used to compare series, aligning with the methodology of Hyndman (2006) and the M-Competitions among forecasting methods reported by Makridakis & Hibon (2000).

C. OPERATIONAL IMPLICATIONS

1. Impact on Stockouts and Readiness

The improved accuracy provided by XGBoost model can prevent FAB from experiencing stockouts, which delay maintenance activities, ground aircraft, and reduce the effective operational availability of the fleet. Even moderate reductions in forecast error can translate into significant operational benefits when scaled across thousands of components and maintenance actions.

2. Impact on Inventory Costs

Similarly, forecast overestimation leads to immobilized inventory, tying up capital and increasing storage and obsolescence costs. In the Brazilian context, a single percentage point reduction in average inventory levels can represent millions of *reais* (Reals, Brazilian national currency) in freed working capital, enabling more agile resource reallocation.

D. MANAGERIAL AND POLICY IMPLICATIONS

The results of this research have direct managerial implications for FAB's supply and maintenance planning:

1. Forecast Integration

The XGBoost model can be integrated into existing systems as a monthly forecasting module, automatically updated with new demand data. Its computational requirements are modest, enabling near-real-time to use in operational environments.

2. Decision Support for Procurement

The improved accuracy of forecasts can enhance procurement planning, particularly for parts with medium or high consumption variability, and so reduce emergency orders.

3. Scalable Frameworks for Other Fleets

The same forecasting framework can be adapted to other FAB platforms beyond the T-27, providing a standardized, scalable approach to inventory prediction. Although this study focused specifically on intermittent demand, the overall structure of the framework suggests that it could also perform well for parts with more regular or continuous consumption patterns. In non-intermittent settings, the richer data availability may even enhance model stability and accuracy, as the model would be able to learn smoother trends and more consistent relationships. While this was not evaluated in the present research, the operational logic indicates that the approach could extend naturally to mixed-demand environments.

E. LIMITATIONS AND FUTURE RESEARCH

Although the results are promising, several limitations should be acknowledged:

- Data size constraints – Some parts have limited historical records, which restricts the training capacity of complex models.
- Integration with inventory optimization – Forecasts were evaluated using statistical accuracy; Operational validation (e.g., service levels, stockout costs, holding costs) remains an important next step.

Future studies could extend this research by:

- Coupling machine-learning forecasts with inventory control policies.
- Estimating the economic impact of forecast errors through simulation of stockout costs, readiness rates, and flight-hour costs.
- Comparing feature-based boosting models with hybrid or deep-learning methods using larger datasets.

Exploring cross-fleet generalization, which enables knowledge transfer between similar aircraft systems, such as the C-97 BANDEIRANTES and the C-97 BRASÍLIA.

F. FINAL REMARKS

This thesis demonstrated that integrating machine-learning forecasting into FAB's logistics processes can achieve substantial operational benefits.

By reducing forecast errors, XGBoost provides a quantitative foundation for more reliable maintenance planning, higher fleet readiness, and lower procurement and inventory costs.

The approach represents a transition from purely reactive logistics toward predictive and data-driven sustainment, consistent with the modernization goals of the Brazilian Air Force.

APPENDIX A. FILLING MONTH ROWS CODE

```
"""
fill_missing_T27.py

This script reads the raw T-27 demand CSV,
auto-detects the delimiter and parses with a tolerant engine,
fills missing months (Jan 2015 – Apr 2025) with zero demand,
and writes the zero-filled series to the specified output.
"""
import os
import csv
import pandas as pd

# === Configuration ===
INPUT_FILE = 'path'
OUTPUT_DIR = "path"
OUTPUT_FILE = os.path.join(OUTPUT_DIR, 'T27_filled.csv')
START_DATE = '2015-01-01'
END_DATE   = '2025-04-01'


def main():
  # Ensure output directory exists
  os.makedirs(OUTPUT_DIR, exist_ok=True)

  # 1) Detect delimiter from a sample of the raw file
  with open(INPUT_FILE, 'r', newline='', encoding='utf-8') as f:
    sample = f.read(4096)
  dialect = csv.Sniffer().sniff(sample, delimiters=[',',';','\t','|'])
  sep = dialect.delimiter
  print(f"Detected delimiter: '{sep}'")

  # 2) Read using the more forgiving Python engine
  df = pd.read_csv(
    INPUT_FILE,
    sep=sep,
    engine='python',
    on_bad_lines='warn',
    quotechar='"',
    skipinitialspace=True,
    encoding='utf-8'
  )

  # 3) Clean column names: strip BOM, quotes, whitespace
```

```
df.columns = (
    df.columns
      .str.lstrip('\ufeff')
      .str.replace("", '', regex=False)
      .str.strip()
)
# 4) Clean string cells: strip stray quotes/whitespace
for col in df.select_dtypes(include='object').columns:
    df[col] = (
        df[col]
          .astype(str)
          .str.replace("", '', regex=False)
          .str.strip()
    )

# 5) Build datetime column from year/month
if {'ANO_COMPETENCIA','MES_COMPETENCIA'}.issubset(df.columns):
    df['DT_COMPETENCIA'] = pd.to_datetime(
        df['ANO_COMPETENCIA'].astype(int).astype(str)
        + '-' +
        df['MES_COMPETENCIA'].astype(int).astype(str)
        + '-01',
        errors='coerce'
    )
else:
    raise KeyError(f"Missing date columns. Found: {df.columns.tolist()}")

# 6) Identify demand column (e.g. QT_CONSUMO or QTD_CONSUMO)
cand = [
    c for c in df.columns
    if c.replace('_','').upper().startswith('QT') and 'CONSUMO' in c.upper()
]
if not cand:
    raise KeyError(f"Demand column not found. Available: {df.columns.tolist()}")
demand_col = cand[0]
df = df.rename(columns={demand_col: 'DEMAND'})
df['DEMAND'] = pd.to_numeric(df['DEMAND'], errors='coerce').fillna(0)

# 7) Prepare full monthly index
full_idx = pd.date_range(start=START_DATE, end=END_DATE, freq='MS')

# 8) For each part number (NR_PN), sum by month and fill zeros
rows = []
if 'NR_PN' in df.columns:
    group_key = 'NR_PN'
else:
```

```
        group_key = None

    # If multiple parts exist, we'll keep their static info; otherwise brute force
    groups = df.groupby(group_key) if group_key else [('ALL', df)]
    for key, grp in groups:
        # static columns are all except date, demand, ano/mes
        static_cols = [
            c for c in grp.columns
            if c not in ('DT_COMPETENCIA','DEMAND','MES_COMPETENCIA','ANO_COMPETENCIA')
        ]
        static_vals = grp.iloc[0][static_cols].to_dict()

        # sum same-month demand
        ts = grp.set_index('DT_COMPETENCIA')['DEMAND']
        monthly = ts.resample('MS').sum().reindex(full_idx, fill_value=0)

        # generate output rows
        for date, val in monthly.items():
            row = static_vals.copy()
            row['DT_COMPETENCIA']  = date
            row['MES_COMPETENCIA'] = date.month
            row['ANO_COMPETENCIA'] = date.year
            row['DEMAND']          = int(val)
            if group_key:
                row[group_key] = key
            rows.append(row)

    # 9) Build DataFrame and rename demand back
    result = pd.DataFrame(rows)
    result = result.rename(columns={'DEMAND': demand_col})

    # 10) Order columns (static + mes, ano, dt, demand)
    ordered = static_cols + (
        [group_key] if group_key else []
    ) + [
        'MES_COMPETENCIA','ANO_COMPETENCIA','DT_COMPETENCIA', demand_col
    ]
    result = result[[c for c in ordered if c in result.columns]]

    # 11) Save to CSV
    result.to_csv(OUTPUT_FILE, index=False)
    print(f"Written zero-filled T-27 series to: {OUTPUT_FILE}")


if __name__ == '__main__':
    main()
```

APPENDIX B. FILTERING INTERMITTENT PATTERN CODE

```
"""
filter_by_spike_count.py

Keeps only those NR_PN whose total non-zero-demand months (spikes)
fall between MIN_SPIKES and MAX_SPIKES (inclusive).
Maintains the full monthly structure (one row per month per PN).
"""
import os
import pandas as pd

# ——— Configuration ———
INPUT_FILE  = 'path'
OUTPUT_DIR  = 'path'
FILTERED_FILE = os.path.join(OUTPUT_DIR, 'T27_intermittent_PNs_clean.csv')
MIN_SPIKES    = 4
MAX_SPIKES    = 60


def main():
  os.makedirs(OUTPUT_DIR, exist_ok=True)

  # Load the zero-filled monthly dataset
  df = pd.read_csv(INPUT_FILE, parse_dates=['DT_COMPETENCIA'])

  # Count unique PNs before filtering
  total_before = df['NR_PN'].nunique()
  print(f"Unique PNs before filtering: {total_before}")

  # Compute spike counts (months with demand > 0) per PN
  spike_counts = (
    df.groupby('NR_PN')['QT_CONSUMO']
      .apply(lambda s: (s > 0).sum())
      .reset_index(name='spike_count')
  )

  # Identify valid PNs
  valid_pns = spike_counts [
    (spike_counts ['spike_count'] >= MIN_SPIKES) &
    (spike_counts ['spike_count'] <= MAX_SPIKES)
  ]['NR_PN']
  total_after = valid_pns.nunique()
  print(f"Unique PNs after filtering (spikes between {MIN_SPIKES} and {MAX_SPIKES}): {total_after}")
```

```
    # Filter and save full monthly structure
    df_filtered = df[df['NR_PN'].isin(valid_pns)].copy()
    df_filtered.to_csv(FILTERED_FILE, index=False)
    print(f"Filtered data written to {FILTERED_FILE} ({len(df_filtered)} rows)")

if __name__ == '__main__':
    main()


# trim_by_first_gap.py
# Keeps, for each PN, data from (first_spike – gap_between_first_two_spikes) onward.
# If only one spike exists -> keep fallback_pre months before first spike.
# If no spikes -> drop PN when --drop_all_zero is set.

import argparse
import numpy as np
import pandas as pd
from pathlib import Path

def months_between(d1: pd.Timestamp, d2: pd.Timestamp) -> int:
    return (d2.year – d1.year) * 12 + (d2.month – d1.month)

def trim_group(g: pd.DataFrame, date_col: str, qty_col: str,
               fallback_pre: int, drop_all_zero: bool, verbose: bool=False):
    g = g.sort_values(date_col).copy()
    vals = pd.to_numeric(g[qty_col], errors="coerce").fillna(0).to_numpy()

    pos_idx = np.flatnonzero(vals > 0)

    if len(pos_idx) == 0:
        if drop_all_zero:
            if verbose:
                print(f" -> no spikes: DROPPED")
            return None, dict(reason="all_zero_dropped")
        else:
            if verbose:
                print(f" -> no spikes: KEPT (no trim)")
            return g, dict(reason="all_zero_kept," start_date=g[date_col].min(),
                           first_spike=None, second_spike=None, gap_m=None)

    i1 = int(pos_idx[0]); d1 = pd.to_datetime(g.iloc[i1][date_col])
    if len(pos_idx) >= 2:
        i2 = int(pos_idx[1]); d2 = pd.to_datetime(g.iloc[i2][date_col])
        gap_m = max(months_between(d1, d2) – 1, 0)  # exclusive gap
        pre_m = gap_m
```

```
        reason = "two_spikes_gap"
    else:
        gap_m = None
        pre_m = max(int(fallback_pre), 0)
        d2 = None
        reason = "single_spike_fallback"

    d1p = pd.Period(d1, freq="M")
    start_date = (d1p - pre_m).to_timestamp(how="start")
    earliest = pd.to_datetime(g[date_col]).min()
    if start_date < earliest:
        start_date = earliest

    trimmed = g[g[date_col] >= start_date].copy()
    meta = dict(reason=reason, start_date=start_date, first_spike=d1,
                second_spike=d2, gap_m=gap_m, kept_rows=len(trimmed))
    if verbose:
        if reason == "two_spikes_gap":
            print(f"  -> first={d1.date()}, second={d2.date()}, gap={gap_m}m, start={start_date.date()},
kept={len(trimmed)}")
        else:
            print(f"  -> first={d1.date()}, only one spike, fallback_pre={pre_m}m, start={start_date.date()},
kept={len(trimmed)}")
    return trimmed, meta

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--input," required=True)
    ap.add_argument("--output," required=True)
    ap.add_argument("--pn_col," default="NR_PN")
    ap.add_argument("--date_col," default="DT_COMPETENCIA")
    ap.add_argument("--qty_col," default="QT_CONSUMO")
    ap.add_argument("--fallback_pre," type=int, default=3)
    ap.add_argument("--drop_all_zero," action="store_true")
    ap.add_argument("--save_log," default="trim_log.csv")
    ap.add_argument("--verbose," action="store_true")
    ap.add_argument("--preview," type=int, default=5, help="print first N PN summaries")
    args = ap.parse_args()

    inp = Path(args.input); outp = Path(args.output); logp = Path(args.save_log)
    if args.verbose:
        print(f"[INFO] Reading {inp.resolve()}")

    df = pd.read_csv(inp)
    if args.date_col not in df.columns:
        raise SystemExit(f"[ERR] date_col '{args.date_col}' not in CSV. Got: {list(df.columns)[:12]}")
```

```
df[args.date_col] = pd.to_datetime(df[args.date_col], errors="coerce")
if df[args.date_col].isna().all():
    raise SystemExit("[ERR] All dates failed to parse. Check DT_COMPETENCIA format.")

df[args.qty_col] = pd.to_numeric(df[args.qty_col], errors="coerce").fillna(0)

if args.verbose:
    print(f"[INFO] Rows={len(df):,}, PNs={df[args.pn_col].nunique():,}, "
          f"Date range={df[args.date_col].min().date()}→{df[args.date_col].max().date()}")

trimmed_blocks = []
log_rows = []
for k, (pn, g) in enumerate(df.groupby(args.pn_col, sort=False), start=1):
    if args.verbose and k <= args.preview:
        print(f"[PN {k}] {pn} ({len(g)} rows)")
        trimmed, meta = trim_group(g, args.date_col, args.qty_col,
                                   args.fallback_pre, args.drop_all_zero, verbose=True)
    else:
        trimmed, meta = trim_group(g, args.date_col, args.qty_col,
                                   args.fallback_pre, args.drop_all_zero, verbose=False)

    if trimmed is not None and not trimmed.empty:
        trimmed_blocks.append(trimmed)
        meta["NR_PN"] = pn
        log_rows.append(meta)
    else:
        log_rows.append(dict(NR_PN=pn, reason=meta["reason"], kept_rows=0,
                             start_date=None, first_spike=None, second_spike=None, gap_m=None))

if len(trimmed_blocks) == 0:
    raise SystemExit("[WARN] No PNs kept. Likely all-zero and --drop_all_zero was used.")

out_df = pd.concat(trimmed_blocks, ignore_index=True).sort_values([args.pn_col, args.date_col])
out_df.to_csv(outp, index=False)

log_df = pd.DataFrame(log_rows)
cols = ["NR_PN","reason","start_date","first_spike","second_spike","gap_m","kept_rows"]
for c in cols:
    if c not in log_df.columns:
        log_df[c] = None
log_df = log_df[cols]
log_df.to_csv(logp, index=False)

print(f"[OK] Saved trimmed dataset -> {outp.resolve()}")
print(f"[OK] Saved trimming log    -> {logp.resolve()}")
```

```
    print("[OK] Reasons: two_spikes_gap = used gap between first two spikes; "
          "single_spike_fallback = used fallback_pre; all_zero_dropped = PN dropped")

if __name__ == "__main__":
    main()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. MOVING AVERAGE CODE

```
#
_____
# File: moving_average_24.py
# Purpose: Forecast h steps using the mean of the last 24 observations.
# Usage:
# from moving_average_24 import ma24
# yhat = ma24(series, h=12)
#
_____
import pandas as pd


def ma24(y):
    """
    Rolling 24-month moving average, one-step-ahead.
    Uses the previous 24 months to predict the current month.
    First 24 rows are NaN.
    """
    y = pd.Series(y)
    return y.rolling(24).mean().shift(1)


def attach_ma24_forecast(df):
    """
    Expect columns: 'NR_PN', 'DT_COMPETENCIA', 'QT_CONSUMO'.
    Adds 'forecast' = rolling 24-month mean shifted by 1 (one-step-ahead), per PN.
    Returns a new DataFrame with all original columns + 'forecast'.
    """
    out = df.copy()
    out ["DT_COMPETENCIA"] = pd.to_datetime(out ["DT_COMPETENCIA"])
    out = out.sort_values(["NR_PN," "DT_COMPETENCIA"])
    out ["MA_24"] = (
        out.groupby("NR_PN," group_keys=False)["QT_CONSUMO"]
          .transform(lambda s: s.rolling(24).mean().shift(1))
    )
    return out
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. SIMPLE EXPONENTIAL SMOOTHING CODE

```
import pandas as pd
import numpy as np

def ses(y, alpha: float = 0.2) -> pd.Series:
    """
    Simple Exponential Smoothing (SES), 1-step-ahead line aligned to y.
    For time t, forecast uses data up to t-1.
    First value is NaN by design.
    """
    y = pd.Series(y).astype(float)
    fc = pd.Series(np.nan, index=y.index)
    if len(y) == 0:
        return fc

    level = y.iloc[0]  # init with first value
    for i in range(1, len(y)):
        fc.iloc[i] = level                  # forecast for time i is level at i-1
        level = alpha * y.iloc[i] + (1-alpha) * level  # update level with y[i]

    return fc.clip(lower=0)

def attach_ses(df, alpha: float = 0.2, out_col: str = "ses"):
    """
    Expects columns: 'NR_PN', 'DT_COMPETENCIA', 'QT_CONSUMO'.
    Adds a 1-step-ahead SES column per PN.
    """
    out = df.copy()
    out["DT_COMPETENCIA"] = pd.to_datetime(out["DT_COMPETENCIA"])
    out = out.sort_values(["NR_PN","DT_COMPETENCIA"])
    out[out_col] = (
        out.groupby("NR_PN", group_keys=False)["QT_CONSUMO"]
           .apply(lambda s: ses(s, alpha=alpha))
    )
    return out
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. CROSTON SBA CODE

```
#
_____
_____
# File: croston.py
# Purpose: calculate one-step-ahead Croston forecasts for intermittent demand series.
# Usage:
# from croston import croston
#
_____
_____
import numpy as np
import pandas as pd

def croston(y: pd.Series, alpha: float, adjust: bool = False) -> pd.Series:
    """
    One-step-ahead Croston line aligned to y.
    At each t, forecast uses data up to t-1.
    """
    y = pd.Series(y).astype(float)
    fc = pd.Series(np.nan, index=y.index)

    q = None  # smoothed demand size
    p = None  # smoothed inter-demand interval
    z = 0     # periods since last demand

    for i, val in enumerate(y.values):
        # 1-step-ahead forecast BEFORE updating with y[t]
        if q is not None and p not in (None, 0):
            r = q / p
            if adjust:
                r *= (1 - alpha / 2.0)  # SBA bias correction
            fc.iloc[i] = r

        # Update state WITH current observation
        z += 1
        if val > 0:
            if q is None:
                q, p = val, z
            else:
                q = q + alpha * (val - q)
                p = p + alpha * (z - p)
            z = 0
```

```
    return fc

def croston_opt(y, alpha_grid=None, adjust: bool = False):
    """
    Choose α by minimizing in-sample RMSE of the ONE-STEP-AHEAD rolling Croston line.
    Returns: (forecast_series, best_alpha)
    """
    y = pd.Series(y).astype(float)
    if (y < 0).any():
        raise ValueError("Croston requires nonnegative data.")
    if alpha_grid is None:
        # conservative small alphas commonly used for intermittent monthly demand
        alpha_grid = np.linspace(0.05, 0.30, 6)  # 0.05, 0.10, ..., 0.30

    best_alpha, best_rmse = None, float("inf")
    best_fc = None

    for a in alpha_grid:
        fc = croston(y, alpha=a, adjust=adjust)
        mask = fc.notna()
        if mask.sum() == 0:
            continue
        rmse = float(np.sqrt(np.mean((y[mask].values - fc[mask].values) ** 2)))
        if rmse < best_rmse:
            best_rmse, best_alpha, best_fc = rmse, float(a), fc

    # Fallback if never initialized (all zeros)
    if best_fc is None:
        best_alpha = float(alpha_grid[0])
        best_fc = pd.Series(0.0, index=y.index)

        return best_fc, best_alpha

def attach_croston(df, alpha: float = 0.1, adjust: bool = False, out_col: str = "Croston-SBA"):
        """
        Expects columns: 'NR_PN', 'DT_COMPETENCIA', 'QT_CONSUMO'.
        Adds a one-step-ahead Croston forecast column per PN.
        """
        out = df.copy()
        out["DT_COMPETENCIA"] = pd.to_datetime(out["DT_COMPETENCIA"])
        out = out.sort_values(["NR_PN","DT_COMPETENCIA"])
        out[out_col] = (
            out.groupby("NR_PN", group_keys=False)["QT_CONSUMO"]
               .apply(lambda s: croston(s, alpha=alpha, adjust=adjust))
        )
```

APPENDIX F. NEURAL NETWORK CODE

```
# nn.py
# A neural net forecaster (scikit-learn MLP) for intermittent monthly demand.
# – Inputs: last L months (lags only)
# – Predicts next month; rolled recursively for H months
# – One model for all PNs
# – Leakage-safe: for each PN, last VAL_LAST samples are NOT used for training.

import numpy as np
import pandas as pd
from sklearn.neural_network import MLPRegressor

# ====== CONFIG ======
CSV_PATH   = "path"  # <-- set this to your file path
BACKTEST_OUT = "path"
FUTURE_OUT   = "path"

L         = 24   # lookback months
H         = 3    # horizon months
VAL_LAST  = 24   # number of tail months per PN to backtest (and exclude from training)
SEED      = 10

# ======= HELPERS =======
def make_continuous_monthly(df: pd.DataFrame) -> pd.DataFrame:
    out = []
    for pn, g in df.groupby("NR_PN," sort=False):
        g = g.sort_values("DT_COMPETENCIA")
        idx = pd.date_range(g ["DT_COMPETENCIA"].min(), g ["DT_COMPETENCIA"].max(), freq="MS")
        gg = g.set_index("DT_COMPETENCIA").reindex(idx)
        gg ["NR_PN"] = pn
        gg ["QT_CONSUMO"] = pd.to_numeric(gg ["QT_CONSUMO"], errors="coerce").fillna(0.0)
        gg.index.name = "DT_COMPETENCIA"
        out.append(gg.reset_index())
    return pd.concat(out, ignore_index=True)

def build_windows(y: np.ndarray, L: int):
    X, Y = [], []
    for t in range(L, len(y)):
        X.append(y [t-L:t])
        Y.append(y [t])
    if not X:
        return np.zeros((0, L), dtype=np.float32), np.zeros((0,), dtype=np.float32)
    return np.array(X, dtype=np.float32), np.array(Y, dtype=np.float32)
```

```
# ====== MAIN ======
def main():
  # Load & clean
  df = pd.read_csv(CSV_PATH)
  df =
df.rename(columns={"DT_COMPETENCIA":"DT_COMPETENCIA","NR_PN":"NR_PN","QT_CONSUMO":"Q
T_CONSUMO"})
  df["DT_COMPETENCIA"] = pd.to_datetime(df["DT_COMPETENCIA"])
  df["NR_PN"] = df["NR_PN"].astype(str)
  df["QT_CONSUMO"] = pd.to_numeric(df["QT_CONSUMO"], errors="coerce").fillna(0.0)

  # Monthly continuity per PN
  panel = make_continuous_monthly(df)

  # ---- Build global TRAIN set (exclude tail windows per PN) ----
  X_train, y_train = [], []
  backtest_index = []  # to know what to predict later (per PN, time index)
  meta = []            # store (pn, dates, y, X_all) to re-use

  for pn, g in panel.groupby("NR_PN," sort=False):
    g = g.sort_values("DT_COMPETENCIA")
    dates = g["DT_COMPETENCIA"].values
    y = g["QT_CONSUMO"].astype(np.float32).values

    X_all, Y_all = build_windows(y, L)
    n = len(X_all)
    if n == 0:
      continue

    v = min(VAL_LAST, n)   # hold out at least 1, ~20% if big
    tr = max(1, n – v)

    # add training part to global set
    X_train.append(X_all[:tr])
    y_train.append(Y_all[:tr])

    # remember which windows are held out for backtest for this PN
    # window t predicts y at index t (0-based on windows) -> original series index = L + t
    held_start_t = tr  # first held-out window
    held_idxs = np.arange(held_start_t, n) + L  # target indices in original series
    backtest_index.append(pd.DataFrame({
      "NR_PN": pn,
      "TARGET_POS": held_idxs
    }))
```

```
    meta.append((pn, dates, y, X_all, Y_all, tr))

if not X_train:
    raise RuntimeError("Not enough data to train.")

X_train = np.concatenate(X_train, axis=0)
y_train = np.concatenate(y_train, axis=0)

# ---- Fit tiny MLP once ----
model = MLPRegressor(
    hidden_layer_sizes=(32,),
    activation="relu",
    solver="adam",
    learning_rate_init=1e-3,
    max_iter=500,
    random_state=SEED,
    verbose=False
)
model.fit(X_train, y_train)

# ---- Backtest predictions (monthly line over held-out tail) ----
bt_rows = []
for (pn, dates, y, X_all, Y_all, tr) in meta:
    # Predict only held-out windows for this PN
    if tr < len(X_all):
        X_hold = X_all[tr:]
        yhat_hold = model.predict(X_hold).astype(float)
        # clamp to >= 0 (counts)
        yhat_hold = np.maximum(yhat_hold, 0.0)

        target_pos = np.arange(tr, len(X_all)) + L
        for pos, yh in zip(target_pos, yhat_hold):
            bt_rows.append({
                "NR_PN": pn,
                "DT_COMPETENCIA": pd.Timestamp(dates[pos]),
                "actual": float(y[pos]),
                "yhat": float(yh)
            })

bt_df = pd.DataFrame(bt_rows).sort_values(["NR_PN","DT_COMPETENCIA"])
bt_df.to_csv(BACKTEST_OUT, index=False)

# ---- Future forecasts (H months ahead per PN) ----
fut_rows = []
for pn, g in panel.groupby("NR_PN", sort=False):
    g = g.sort_values("DT_COMPETENCIA")
```

```
        y = g ["QT_CONSUMO"].astype(np.float32).values

        last_date = g ["DT_COMPETENCIA"].max()
        future_dates = pd.date_range(last_date + pd.offsets.MonthBegin(1), periods=H, freq="MS")

        if len(y) < L:
            base = float(y [-1]) if len(y) else 0.0
            fc = [base]*H
        else:
            hist = list(y [-L:])
            fc = []
            for _ in range(H):
                x = np.array(hist [-L:], dtype=np.float32)[None, :]
                yhat = float(model.predict(x)[0])
                yhat = max(0.0, yhat)
                fc.append(yhat)
                hist.append(yhat)

        for i, d in enumerate(future_dates, 1):
            fut_rows.append({"NR_PN": pn, "DT_FORECAST": d, "h": i, "yhat": float(fc [i-1])})

    fut_df = pd.DataFrame(fut_rows).sort_values(["NR_PN","DT_FORECAST","h"])
    fut_df.to_csv(FUTURE_OUT, index=False)

    print(f"Backtest saved: {BACKTEST_OUT}")
    print(f"Future saved:   {FUTURE_OUT}")

if __name__ == "__main__":
    main()
```

APPENDIX G. XGBOOST CODE

```
"""
Fast rolling XGB (two-stage: classifier + regressor) + metrics for intermittent demand.
"""

import os, sys, time, math, argparse, warnings
from typing import List, Tuple, Optional

import numpy as np
import pandas as pd
from xgboost import XGBClassifier, XGBRegressor

warnings.filterwarnings("ignore," category=FutureWarning)
pd.set_option("display.max_columns," 120)

# ============================
# CONFIG: put your paths here
# ============================
INPUT_CSV_DEFAULT      = "path"              # <-- change me
OUTPUT_PREDS_DEFAULT   = "path"                 # <-- change me
OUTPUT_METRICS_DEFAULT  = "path"                  # <-- change me
# ============================


def month_start(d: pd.Timestamp) -> pd.Timestamp:
  return pd.Timestamp(d.year, d.month, 1)

def ensure_monthly_continuity(g: pd.DataFrame) -> pd.DataFrame:
  g = g.sort_values("DT_COMPETENCIA").copy()
  idx = pd.period_range(g ["DT_COMPETENCIA"].min(), g ["DT_COMPETENCIA"].max(),
freq="M").to_timestamp(how="start")
  g = g.set_index("DT_COMPETENCIA").reindex(idx)
  g = g.rename_axis("DT_COMPETENCIA").reset_index()
  g ["NR_PN"] = g ["NR_PN"].ffill().bfill()
  g ["QT_CONSUMO"] = pd.to_numeric(g ["QT_CONSUMO"], errors="coerce").fillna(0.0)
  return g

def recency_since_last_nonzero(values: np.ndarray) -> np.ndarray:
  out = np.zeros_like(values, dtype=np.float32)
  since = math.inf
  for i, v in enumerate(values):
    if v > 0:
      since = 0
```

```
        else:
            since = since + 1 if since != math.inf else since
        out [i] = 0.0 if since == math.inf else float(since)
    return np.clip(out, 0, 120)

def rolling_nonzero_rate(x: np.ndarray, w: int) -> np.ndarray:
    out = np.zeros_like(x, dtype=np.float32)
    cumsum_nz = np.cumsum((x > 0).astype(np.int32))
    for i in range(len(x)):
        j0 = max(0, i – w)
        total = i – j0
        if total <= 0:
            out [i] = 0.0
        else:
            nz = cumsum_nz [i-1] – (cumsum_nz [j0-1] if j0 > 0 else 0)
            out [i] = float(nz) / float(total)
    return out

def rolling_mean_nonzero(x: np.ndarray, w: int) -> np.ndarray:
    out = np.zeros_like(x, dtype=np.float32)
    for i in range(len(x)):
        j0 = max(0, i – w)
        window = x [j0:i]
        window = window [window > 0]
        out [i] = float(window.mean()) if len(window) else 0.0
    return out

def add_features(df: pd.DataFrame) -> Tuple [pd.DataFrame, List [str]]:
    assert df [“NR_PN”].nunique() == 1
    df = df.sort_values(“DT_COMPETENCIA”).copy()
    y = df [“QT_CONSUMO”].astype(float).values

    # --- SAFE ---
    for k in range(1, 13):
        df [f”lag_{k}”] = pd.Series(y).shift(k).fillna(0.0).values

    df [“delta_1”] = df [“lag_1”] – df [“lag_2”]
    df [“delta_3”] = df [“lag_3”] – df [“lag_6”]
    df [“delta_6”] = df [“lag_6”] – df [“lag_12”]

    # --- SAFE RECENCY: “months since last nonzero” computed using ONLY y up to t-1 ---
    def recency_since_last_nonzero_safe(values: np.ndarray) -> np.ndarray:
        out = np.zeros_like(values, dtype=np.float32)
        since = math.inf # no nonzero seen yet
        for i in range(len(values)):
            # update ‘since’ based on previous month only
```

```
        if i > 0 and values [i-1] > 0:
            since = 0
        elif since != math.inf:
            since += 1
        # if we have never seen nonzero, keep 0.0 (same convention as before)
        out [i] = 0.0 if since == math.inf else float(since)
    return np.clip(out, 0, 120)

df ["recency_nz"] = recency_since_last_nonzero_safe(y)

# --- SAFE ROLLING NONZERO RATE & MEAN: your functions already exclude current (window = x
[j0:i]) ---
for w in (3, 6, 12):
    df [f"nz_rate_{w}"] = rolling_nonzero_rate(y, w)   # uses y up to t-1 internally
    df [f"mean_nz_{w}"] = rolling_mean_nonzero(y, w)   # uses y up to t-1 internally

# --- SAFE ROLLING SUMS: exclude current month by shifting 1 ---
s = pd.Series(y)
df ["roll_sum_3"]  = s.shift(1).rolling(3).sum().fillna(0.0).values  # sum(y [t-3:t])
df ["roll_sum_6"]  = s.shift(1).rolling(6).sum().fillna(0.0).values
df ["roll_sum_12"] = s.shift(1).rolling(12).sum().fillna(0.0).values

# --- CALENDAR / INDEX (safe) ---
dt = pd.to_datetime(df ["DT_COMPETENCIA"])
df ["month"]   = dt.dt.month.astype(np.int16)
df ["quarter"] = dt.dt.quarter.astype(np.int16)
df ["t_idx"]   = np.arange(len(df), dtype=np.int32)

# --- TARGETS ---
df ["y_clf"] = (df ["QT_CONSUMO"] > 0).astype(np.int8)
df ["y_reg"] = df ["QT_CONSUMO"].astype(float)

Xcols = [c for c in df.columns if c not in ("NR_PN","DT_COMPETENCIA","QT_CONSUMO","y_clf","y_reg")]
return df, Xcols

def fit_two_stage_no_es(tr_full: pd.DataFrame, Xcols: List [str],
                        clf_params: dict, reg_params: dict,
                        threads: int, seed: int):
    if len(tr_full) >= 40:
        cut = int(len(tr_full) * 0.85)
        tr = tr_full.iloc [:cut]
        va = tr_full.iloc [cut:]
    else:
        tr = tr_full
        va = None
```

```
y_clf_tr = tr ["y_clf"].astype(int).values
uniq = np.unique(y_clf_tr)
if len(uniq) == 1:
    clf, const_p = None, float(uniq [0])
else:
    const_p = None
    clf = XGBClassifier(
        n_estimators=clf_params ["n_estimators"],
        max_depth=clf_params ["max_depth"],
        learning_rate=clf_params ["learning_rate"],
        subsample=0.9, colsample_bytree=0.9,
        reg_lambda=1.0, min_child_weight=1.0,
        objective="binary:logistic",
        n_jobs=threads, tree_method="hist",
        random_state=seed
    )
    eval_set = None
    if va is not None and len(va) > 0:
        eval_set = [(va [Xcols].values, va ["y_clf"].astype(int).values)]
    clf.fit(tr [Xcols].values, y_clf_tr, eval_set=eval_set, verbose=False)

mask_nz_tr = (tr ["y_reg"] > 0)
if mask_nz_tr.sum() < 5:
    reg = None
else:
    Xr = tr.loc [mask_nz_tr, Xcols].values
    yr = tr.loc [mask_nz_tr, "y_reg"].values
    w  = np.clip(yr, 1.0, None)
    reg = XGBRegressor(
        n_estimators=reg_params ["n_estimators"],
        max_depth=reg_params ["max_depth"],
        learning_rate=reg_params ["learning_rate"],
        subsample=0.9, colsample_bytree=0.9,
        reg_lambda=1.0, min_child_weight=1.0,
        objective="reg:squarederror",
        n_jobs=threads, tree_method="hist",
        random_state=seed
    )
    if va is not None and len(va) > 0:
        mask_nz_va = (va ["y_reg"] > 0)
        if mask_nz_va.sum() > 0:
            reg.fit(Xr, yr, sample_weight=w,
                    eval_set=[(va.loc [mask_nz_va, Xcols].values,
                               va.loc [mask_nz_va, "y_reg"].values)],
                    verbose=False)
        else:
```

```
            reg.fit(Xr, yr, sample_weight=w)
        else:
            reg.fit(Xr, yr, sample_weight=w)

    unique = (const_p is not None)
    return clf, reg, unique, const_p

def predict_two_stage(clf, reg, unique: bool, const_p: Optional [float],
                      last_row: pd.DataFrame, Xcols: List [str]) -> Tuple [float,float,float]:
    if unique:
        p = float(const_p)
    else:
        p = float(np.asarray(clf.predict_proba(last_row [Xcols].values)[:,1]).ravel()[0])
    if reg is None:
        size = 0.0
    else:
        size = float(max(0.0, np.asarray(reg.predict(last_row [Xcols].values)).ravel()[0]))
    return p, size, p*size

def rolling_backtest_one_pn(g: pd.DataFrame,
                            train_window: int,
                            refit_every: int,
                            clf_params: dict, reg_params: dict,
                            threads: int, seed: int) -> pd.DataFrame:
    g = ensure_monthly_continuity(g)
    feat_all, Xcols = add_features(g)

    rows = []
    clf = reg = None
    unique = False
    const_p = None

    for i in range(train_window, len(feat_all)):
        if (i - train_window) % refit_every == 0 or clf is None:
            win = feat_all.iloc [i - train_window:i].copy()
            clf, reg, unique, const_p = fit_two_stage_no_es(win, Xcols, clf_params, reg_params, threads, seed)

        last_row = feat_all.iloc [[i]].copy()
        p, size, yhat = predict_two_stage(clf, reg, unique, const_p, last_row, Xcols)

        rows.append({
            "NR_PN":          g ["NR_PN"].iloc [0],
            "DT_COMPETENCIA": feat_all ["DT_COMPETENCIA"].iloc [i],
            "actual":         float(feat_all ["QT_CONSUMO"].iloc [i]),
            "lag_1":          float(feat_all ["lag_1"].iloc [i]),
            "p_hat":          float(p),
```

```
            "size_hat":       float(size),
            "yhat_xgb":       float(yhat),
        })

    return pd.DataFrame(rows)

# ---------- Metrics ----------
def mase_per_pn(df_pn: pd.DataFrame) -> float:
    ae = np.abs(df_pn ["actual"] - df_pn ["yhat_xgb"])
    denom = np.abs(df_pn ["actual"] - df_pn ["lag_1"])
    d = denom.mean()
    return (ae.mean() / d) if d and not np.isclose(d, 0.0) else np.nan

def spike_metrics_per_pn(df_pn: pd.DataFrame, spike_q: float, tau: float) -> Tuple [float,float,int,int,int]:
    if df_pn.empty:
        return np.nan, np.nan, 0, 0, 0
    spike_cut = df_pn ["actual"].quantile(spike_q)
    actual_spike = df_pn ["actual"] >= spike_cut
    pred_spike   = (df_pn ["p_hat"] >= tau) & (df_pn ["yhat_xgb"] >= spike_cut)
    tp = int((actual_spike & pred_spike).sum())
    n_actual = int(actual_spike.sum())
    n_pred   = int(pred_spike.sum())
    recall = (tp / n_actual) if n_actual > 0 else np.nan
    precision = (tp / n_pred) if n_pred > 0 else np.nan
    return float(recall), float(precision), n_actual, n_pred, tp

def undershare_per_pn(df_pn: pd.DataFrame, undershare_q: float) -> float:
    y = df_pn ["actual"].values
    yhat = df_pn ["yhat_xgb"].values
    eps = 1e-9
    share_miss = np.maximum(0.0, (y - yhat) / np.maximum(y, eps))
    return float(np.quantile(share_miss, undershare_q)) if len(share_miss) else np.nan

def compute_metrics(preds: pd.DataFrame, spike_q: float, tau: float, undershare_q: float) -> pd.DataFrame:
    out = []
    for pn, g in preds.groupby("NR_PN"):
        m = {
            "NR_PN": pn,
            "n_obs": len(g),
            "MASE":  mase_per_pn(g),
            "denom_lag1_meanAE": float(np.abs(g ["actual"] - g ["lag_1"]).mean())
        }
        rec, prec, n_a, n_p, tp = spike_metrics_per_pn(g, spike_q, tau)
        m.update({
            "spike_q": spike_q,
            "tau": tau,
```

```
            "spike_recall": rec,
            "spike_precision": prec,
            "n_actual_spikes": n_a,
            "n_pred_spikes": n_p,
            "n_tp": tp
        })
        m ["undershare_pctl"] = undershare_per_pn(g, undershare_q)
        out.append(m)
    return pd.DataFrame(out).sort_values("NR_PN").reset_index(drop=True)

def panel_summary(metrics: pd.DataFrame):
    valid = metrics.loc [metrics ["n_obs"] > 0].copy()
    if valid.empty:
        print("No metrics to summarize.")
        return
    mase_vals = valid ["MASE"].dropna()
    pct_mase_lt_1 = (mase_vals < 1.0).mean() if len(mase_vals) else np.nan
    print("\n=== Panel summary ===")
    print(f"PNs evaluated:          {len(valid)}")
    print(f"Median MASE:            {np.nanmedian(mase_vals):.3f}")
    print(f"%PNs with MASE < 1:     {pct_mase_lt_1*100:.1f}%")
    print(f"MASE p95:               {np.nanpercentile(mase_vals,95):.3f}")
    rec = valid ["spike_recall"].dropna()
    prec = valid ["spike_precision"].dropna()
    print(f"Spike recall (median):   {np.nanmedian(rec):.3f}" if len(rec) else "Spike recall (median):   n/a")
    print(f"Spike precision (median):{np.nanmedian(prec):.3f}" if len(prec) else "Spike precision (median): n/a")
    und = valid ["undershare_pctl"].dropna()
    print(f"Undershare pctl median:  {np.nanmedian(und):.3f}" if len(und) else "Undershare pctl median:  n/a")
    print("======================\n")

def main():
    ap = argparse.ArgumentParser()
    # IO (CLI overrides CONFIG defaults)
    ap.add_argument("--input_csv,"   type=str, default=INPUT_CSV_DEFAULT, help="Input CSV with
DT_COMPETENCIA,NR_PN,QT_CONSUMO")
    ap.add_argument("--output_preds," type=str, default=OUTPUT_PREDS_DEFAULT, help="Row-level
rolling predictions CSV")
    ap.add_argument("--output_metrics," type=str, default=OUTPUT_METRICS_DEFAULT, help="Per-PN
metrics CSV")
    ap.add_argument("--pn_limit," type=int, default=None)

    # Rolling control
    ap.add_argument("--train_window," type=int, default=36)
    ap.add_argument("--refit_every,"  type=int, default=3)
    ap.add_argument("--threads,"      type=int, default=4)
    ap.add_argument("--seed,"         type=int, default=42)
```

```
    # Model sizes
    ap.add_argument("--clf_n_estimators," type=int, default=150)
    ap.add_argument("--clf_max_depth,"   type=int, default=4)
    ap.add_argument("--clf_lr,"          type=float, default=0.08)
    ap.add_argument("--reg_n_estimators," type=int, default=250)
    ap.add_argument("--reg_max_depth,"   type=int, default=4)
    ap.add_argument("--reg_lr,"          type=float, default=0.05)

    # Metrics thresholds
    ap.add_argument("--spike_q,"     type=float, default=0.95)
    ap.add_argument("--tau,"         type=float, default=0.50)
    ap.add_argument("--undershare_q," type=float, default=0.95)

    args = ap.parse_args()

    if not os.path.exists(args.input_csv):
        print(f"ERROR: input not found: {args.input_csv}")
        sys.exit(1)

    df = pd.read_csv(args.input_csv)
    need = {"DT_COMPETENCIA","NR_PN","QT_CONSUMO"}
    if not need.issubset(df.columns):
        print(f"ERROR: input CSV must have columns {need}")
        sys.exit(1)
    df["DT_COMPETENCIA"] = pd.to_datetime(df["DT_COMPETENCIA"]).map(month_start)
    df["NR_PN"] = df["NR_PN"].astype(str)
    df["QT_CONSUMO"] = pd.to_numeric(df["QT_CONSUMO"], errors="coerce").fillna(0.0)
    df = df.sort_values(["NR_PN","DT_COMPETENCIA"]).reset_index(drop=True)

    pns = df["NR_PN"].unique().tolist()
    if args.pn_limit is not None:
        pns = pns[:args.pn_limit]

    clf_params = dict(n_estimators=args.clf_n_estimators, max_depth=args.clf_max_depth, learning_rate=
args.clf_lr)
    reg_params = dict(n_estimators=args.reg_n_estimators, max_depth=args.reg_max_depth, learning_rate=
args.reg_lr)

    print(f"Starting FAST rolling backtest for {len(pns)} PNs (TRAIN_WINDOW={args.train_window},
REFIT_EVERY={args.refit_every})")

    all_rows = []
    t0 = time.time()
    for i, pn in enumerate(pns, start=1):
        g = df.loc[df["NR_PN"] == pn, ["NR_PN","DT_COMPETENCIA","QT_CONSUMO"]].copy()
```

```
        if g.shape [0] < max(args.train_window, 6):
            continue
        if i == 1 or i % 50 == 0:
            elapsed = (time.time() – t0) / 60.0
            print(f"[{i}/{len(pns)}] PN {pn} | elapsed {elapsed:.1f} min")

        try:
            fc = rolling_backtest_one_pn(
                g,
                train_window=args.train_window,
                refit_every=args.refit_every,
                clf_params=clf_params,
                reg_params=reg_params,
                threads=args.threads,
                seed=args.seed
            )
            all_rows.append(fc)
        except Exception as e:
            print(f" -> PN {pn} failed with error: {e}")

    if not all_rows:
        print("No predictions produced.")
        sys.exit(0)

    preds = pd.concat(all_rows, ignore_index=True)
    preds = preds.sort_values(["NR_PN","DT_COMPETENCIA"]).reset_index(drop=True)
    preds.to_csv(args.output_preds, index=False)
    print(f"Saved rolling predictions → {args.output_preds}")

    metrics = compute_metrics(preds, spike_q=args.spike_q, tau=args.tau, undershare_q=args.undershare_q)
    metrics.to_csv(args.output_metrics, index=False)
    print(f"Saved per-PN metrics     → {args.output_metrics}")

    panel_summary(metrics)

if __name__ == "__main__":
    main()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. ERROR MEASURES (MAE, RMSE AND MASE) CODE

```
"""
Evaluation script for computing MAE, RMSE, and MASE
for each forecasting model in a merged forecasts dataset.

Outputs:
 1) per_pn_metrics.csv     – MAE, RMSE, MASE, n_obs per (NR_PN, model)
 2) per_model_summary.csv  – MAE, RMSE, MASE, n_obs per model (aggregated)

Just edit the CONFIG SECTION below before running.
"""

import pandas as pd
import numpy as np
from pathlib import Path

# ============================================================
# CONFIG SECTION — EDIT THESE VALUES ONLY
# ============================================================
CSV_IN = "path"     # input file
OUT_PER_PN = "path"               # output 1
OUT_PER_MODEL = "path"     # output 2
LAST_N = None                          # e.g., 24 for last 24 months, or None for all
MODELS = ["xgb," "nn," "ma_24," "ses," "arima," "croston_sba"]  # leave [] to auto-detect
# ============================================================


# ---------- helpers ----------
def ensure_actual(df):
  if "actual" not in df.columns:
    if "QT_CONSUMO" in df.columns:
      df = df.rename(columns={"QT_CONSUMO": "actual"})
    else:
      raise ValueError("Missing 'actual' or 'QT_CONSUMO' column.")
  return df

def lastN(g, n):
  if n is None:
    return g
  g = g.sort_values("DT_COMPETENCIA")
  return g.tail(n).copy() if len(g) >= n else pd.DataFrame(columns=g.columns)

def compute_metrics_block(g, pred_col):
```

```
    g = g.sort_values("DT_COMPETENCIA").copy()
    g ["lag1"] = g ["actual"].shift(1)
    g = g.dropna(subset=["lag1," pred_col])
    if g.empty:
        return pd.Series({"n_obs": 0, "MAE": np.nan, "RMSE": np.nan, "MASE": np.nan})
    e = g ["actual"] – g [pred_col]
    mae = e.abs().mean()
    rmse = np.sqrt((e**2).mean())
    denom = (g ["actual"] – g ["lag1"]).abs().mean()
    mase = mae / denom if denom and not np.isclose(denom, 0.0) else np.nan
    return pd.Series({"n_obs": len(g), "MAE": mae, "RMSE": rmse, "MASE": mase})

# ---------- main ----------
print("Loading data...")
df = pd.read_csv(CSV_IN)
if "DT_COMPETENCIA" in df.columns:
    df ["DT_COMPETENCIA"] = pd.to_datetime(df ["DT_COMPETENCIA"], errors="coerce")
df ["NR_PN"] = df ["NR_PN"].astype(str)
df = ensure_actual(df)

id_cols = {"NR_PN," "DT_COMPETENCIA," "actual," "QT_CONSUMO"}

# detect models if not manually set
if not MODELS:
    MODELS = [c for c in df.columns if c not in id_cols and df [c].notna().any()]
print(f"Models detected: {MODELS}")

for m in MODELS:
    df [m] = pd.to_numeric(df [m], errors="coerce")

# limit to last N months if set
if LAST_N is not None:
    df = (
        df.sort_values(["NR_PN," "DT_COMPETENCIA"])
          .groupby("NR_PN," as_index=False)
          .apply(lambda g: lastN(g, LAST_N))
          .reset_index(drop=True)
    )
    print(f"→ Limited to last {LAST_N} months per PN.")

# ---------- per-PN metrics ----------
per_pn_rows = []
for m in MODELS:
    gpn = df.groupby("NR_PN," sort=False).apply(lambda g: compute_metrics_block(g, m)).reset_index()
    gpn.insert(1, "model," m)
    per_pn_rows.append(gpn)
```

```
per_pn_df = pd.concat(per_pn_rows, ignore_index=True)
per_pn_df = per_pn_df[["NR_PN", "model", "n_obs", "MAE", "RMSE", "MASE"]]

# ---------- per-model summary ----------
agg_rows = []
for m in MODELS:
    tmp = df[["NR_PN", "DT_COMPETENCIA", "actual", m]].copy()
    tmp["lag1"] = tmp.groupby("NR_PN")["actual"].shift(1)
    tmp = tmp.dropna(subset=["lag1", m])
    if tmp.empty:
        agg_rows.append({"model": m, "n_obs": 0, "MAE": np.nan, "RMSE": np.nan, "MASE": np.nan})
        continue
    e = tmp["actual"] - tmp[m]
    mae = e.abs().mean()
    rmse = np.sqrt((e**2).mean())
    denom = (tmp["actual"] - tmp["lag1"]).abs().mean()
    mase = mae / denom if denom and not np.isclose(denom, 0.0) else np.nan
    agg_rows.append({"model": m, "n_obs": len(tmp), "MAE": mae, "RMSE": rmse, "MASE": mase})

per_model_df = pd.DataFrame(agg_rows).sort_values("MASE")

# ---------- save ----------
Path(OUT_PER_PN).parent.mkdir(parents=True, exist_ok=True)
per_pn_df.to_csv(OUT_PER_PN, index=False)
per_model_df.to_csv(OUT_PER_MODEL, index=False)

print("\n=== Per-model summary ===")
print(per_model_df.to_string(index=False))
print(f"\nSaved per-PN metrics → {OUT_PER_PN}")
print(f"Saved per-model summary → {OUT_PER_MODEL}")
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Altay, N., Rudisill, F., & Litteral, L. A. (2008). Adapting Wright's modification of Holt's method to forecasting intermittent demand. *International Journal of Production Economics*, *111*(2), 389–408. https://doi.org/10.1016/j.ijpe.2007.01.009

Amosu, O., Kumar, P., Ogunsuji, Y., Oni, S., & Faworaja, O. (2024). AI-driven demand forecasting: Enhancing inventory management and customer satisfaction. *World Journal of Advanced Research and Reviews*, *23*, 708–719. https://doi.org/10.30574/wjarr.2024.23.2.2394

Blanchard, B. S., & Fabrycky, W. J. (2023). *Systems Engineering and Analysis* (5th ed.). Pearson.

Boylan, J. E., & Syntetos, A. A. (2021). *Intermittent demand forecasting: Context, methods and applications* (1st ed.). John Wiley & Sons, Inc. https://doi.org/10.1002/9781119135289

Brazilian Air Force. (2025, January 20). *Ministério da Aeronáutica: Um Marco na História do Brasil*. Força Aérea Brasileira. Retrieved May 27, 2025, from https://www.fab.mil.br/noticias/mostra/43670

Brazilian Air Force. (2024, August 23). *Intendência da Aeronáutica celebra os seus 79 anos de criação*. Força Aérea Brasileira. Retrieved May 27, 2025, from https://www.fab.mil.br/noticias/mostra/43016

Brazilian Air Force. (2023, November 21). *Força Aérea Brasileira comemora 40 anos da aeronave T-27 Tucano*. Força Aérea Brasileira. https://www.fab.mil.br/noticias/mostra/41856

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Müller, A. C., Grisel, O., … Duchesnay, É. (2013). API design for machine learning software: Experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (pp. 108–122).

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches* (5th ed.). SAGE Publications.

Croston, J. D. (1972). Forecasting and stock control for intermittent demands. *Operational Research Quarterly*, 23(3), 289–303. https://doi.org/10.2307/3007885

Douaioui, K., Oucheikh, R., Othmane, B., & Mabrouki, C. (2024). Machine Learning and Deep Learning Models for Demand Forecasting in Supply Chain Management: A Critical Review. *Applied System Innovation*, *7*, 93. https://doi.org/10.3390/asi7050093

Embraer. (2025, May 27). *Embraer: History of Embraer*. Retrieved May 27, 2025, from https://historicalcenter.embraer.com/global/en/history

Gutierrez, R. S., Solis, A. O., & Mukhopadhyay, S. (2008). Lumpy demand forecasting using neural networks. *International Journal of Production Economics*, *111*(2), 409–420. https://doi.org/10.1016/j.ijpe.2007.01.007

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hua, Z. S., Zhang, B., Yang, J., & Tan, D. S. (2007). A new approach of forecasting intermittent demand for spare parts inventories in the process industries. *Journal of the Operational Research Society*, *58*(1), 52–61. https://doi.org/10.1057/palgrave.jors.2602119

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2nd ed.). OTexts. https://otexts.com/fpp2/

Hyndman, R. J. (2006) Another look at forecast accuracy metrics for intermittent demand *Foresight*: *International Journal of Applied Forecasting, 4(1) (2006),* 43–46

Ji, S.; Wang, X.; Zhao, W.; Guo, D. An Application of a Three-Stage XGboost-Based Model to Sales Forecasting of a Cross-Border e-Commerce Enterprise. Math. Probl. Eng. 2019, 2019, 8503252.

Klaus, H., Rosemann, M., & Gable, G. G. (2000). What is ERP? *Information Systems Frontiers*, *2*(2), 141–162.

Levén, E., & Segerstedt, A. (2004). Inventory control with a modified Croston procedure and Erlang distribution. *International Journal of Production Economics*, *90*(3), 361–367. https://doi.org/10.1016/S0925-5273(03)00053-7

Makridakis, S., & Hibon, M. (2000). The M3-Competition: Results, conclusions and implications. *International Journal of Forecasting*, *16*(4), 451–476. https://doi.org/10.1016/S0169-2070(00)00057-1

Makridakis, S., Wheelwright, S. C., & McGee, V. E. (1983). *Forecasting, methods and applications* (2nd ed.). John Wiley & Sons, Inc.

Montgomery, D. C., & Johnson, L. A. (1976). *Forecasting and time series analysis* (1st ed.). McGraw-Hill, Inc.

McKinney, W. (2010). *Data structures for statistical computing in Python.* In *Proceedings of the 9th Python in Science Conference* (pp. 51–56).

Nasiri Pour, A., Rostami-Tabar, B., & Rahimzadeh, A. (2008). A hybrid neural network and traditional approach for forecasting lumpy demand. *Proceedings of the World Academy of Science, Engineering and Technology*, *2*(4). http://waset.org/publications/10793/a-hybrid-neural-network-and-traditional-approach-for-forecasting-lumpy-demand

Preez, J., & Witt, S. (2003). Univariate versus multivariate time series forecasting: An application to international tourism demand. *International Journal of Forecasting*, *19*, 435–451. https://doi.org/10.1016/S0169-2070(02)00057-2

Python Software Foundation. (2024). *Python (Version 3.)* [Computer software]. https://www.python.org/

Ragin, C. C. (2014). *The comparative method: Moving beyond qualitative and quantitative strategies* (Updated ed.). University of California Press.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

Saunders, Mark & Lewis, P. & Thornhill, A.. (2023). *Research Methods for Business Students*. Pearson.

Seabold, S., & Perktold, J. (2010). *Statsmodels: Econometric and statistical modeling with Python.* In *Proceedings of the 9th Python in Science Conference* (pp. 57–61).

Syntetos, A. (2001). *Forecasting of intermittent demand* [Doctoral dissertation, Brunel University]. Buckinghamshire New University Repository. https://bnu.repository.guildhe.ac.uk/id/eprint/9960/1/A%20%20Syntetos%20(PhD%20thesis%202001).pdf

Government Accountability Office. (2003). *Preliminary observations on the effectiveness of logistics support during Operation Iraqi Freedom* (GAO-03-924). U.S. Government Accountability Office. https://www.gao.gov/assets/gao-04-305r.pdf

U.S. Government Accountability Office. (2015). *Defense inventory: Services generally have reduced their excess items and disposed of those no longer needed* (GAO-15-350). https://www.gao.gov/assets/gao-15-350.pdf

Van Creveld, M. (1977). *Supplying War: Logistics from Wallenstein to Patton.* Cambridge University Press.

Verganti, R. (1997). Order overplanning with uncertain lumpy demand: A simplified theory. *International Journal of Production Research*, *35*(12), 3229–3248. https://doi.org/10.1080/002075497194057

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software, 6*(60), 3021. https://doi.org/10.21105/joss.03021

Willemain, T. R., Smart, C. N., & Schwarz, H. F. (2004). A new approach to forecasting intermittent demand for service parts inventories. *International Journal of Forecasting*, *20*(3), 375–387. https://doi.org/10.1016/S0169-2070(03)00013-X

Xu, Q., Wang, N., & Shi, H. (2012). A Review of Croston's method for intermittent demand forecasting. *Proceedings—2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2012*. https://doi.org/10.1109/FSKD.2012.6234258

Xu, J., Brand, J. E., & Koch, B. (2020). Machine learning. In *SAGE Research Methods Foundations*. SAGE Publications Ltd. https://doi.org/10.4135/9781526421036883644

Zhou, Z.-H. (2020). *Machine learning*. Springer. https://doi.org/10.1007/978-981-15-1967-3

Zotteri, G. (2000). The impact of distributions of uncertain lumpy demand on inventories. *Production Planning & Control*, *11*(1), 32–43. https://doi.org/10.1080/095372800232469

Zhuang, X., Yu, Y., & Chen, A. (2022). A combined forecasting method for intermittent demand using the automotive aftermarket data. *Data Science and Management*, *5*(2), 43–56. https://doi.org/10.1016/j.dsm.2022.04.001

Acquisition Research Program
Naval Postgraduate School
555 Dyer Road, Ingersoll Hall
Monterey, CA 93943

www.acquisitionresearch.net