University of Stavanger

Master of Science in Industrial Asset Management

Master's Thesis

The development of a demand forecasting web application for Spare Parts Management using Bootstrapping Method

Author:
Pimprapa Poolsawas
(261817)

Supervisor:
Idriss El-Thalji
Jawad Raza

June 15, 2022

ACKNOWLEDGEMENTS

First of all, I would like to express my greatest gratitude to my professors, Idriss El-Thalji and Jawad Raza, who pushed my knowledge to new boundaries of learning and helped me throughout the project with the report.

I would to show special regards to my partner, Denys Chaikovskyi, who guide me through the development of the web application. This challenge could not have been overcome without his help and direction.

I would also like to express my indebtedness to my parents who although are far away I could not have completed this work without their constant support.

I acknowledge that without their support, this thesis would have not been completed as it is now. And for that, I am very thankful.

Abstract

The forecast has been recognized as one of the most essential parts of spare part management, which often impacts inventory costs and performance to a large degree. In the case study, the demand pattern of spare parts from a subsea maintenance service provider is characterized as intermittent demand. This demand pattern is common among spare parts which accounts for a large portion of inventory costs. As a means of predicting its intermittent demand, there is a method called WSS bootstrapping method, described in the literature. We developed a web application based on the WSS bootstrapping model by Willemain et al. (2004) and an adapted jittering method by Rego and Mesqutia (2015). The computational results show that the results of the bootstrapping model often contain true value when using the 99% confidence interval. The error of the model is lower than our analysis criterion. Therefore, the performance of the bootstrapping model is satisfying.

Contents

Chapter 1

Introduction

1.1 Background and motivation

For operation and maintenance service providers, high reliability and availability of spare part are essential. This is because the lack of spare part can lead to downtime in the overall process of operation and maintenance and result in the risk of economic loss and a negative company's reputation. In order to avoid the costs of downtime, spare parts should be timely available in stock. On the other hand, over-stock of spare parts can also be expensive due to inventory holding costs. In this context, spare part inventory management plays a crucial role in optimizing spare part stock levels to target the availability of spare parts at minimal inventory investment.

The present study is motivated by an operation and maintenance service provider for industrial markets, who is facing these challenges. The company aims to increase spare part availability and a high service level without overstocking undue spare parts in the inventory. This can be challenging since the consumption of spare parts often fluctuates. There are many periods of zero consumption of spare parts, and sudden spikes in demand. This makes it is very difficult to predict the demands of spare parts. As consequence, spare parts are not available in stock timely and the service company is unable to perform operations and maintenance on time, affecting clients' productivity and causing time delays and high costs.

Several models and methods have been developed with the purpose to predict demand of spare parts. However, regular forecasting techniques generate inaccurate estimates when dealing with intermittent demand. One way to improve the forecast accuracy is to develop smart prediction models that has potential to capture this intermittent pattern of spare parts needs.

Many research studies have analyzed the bootstrap methodology (e.g. Smith and Babai, 2011; Willemain et al., 2004; Porras and Dekker, 2008; Syntetos et al., 2015). The results show the bootstrap model outperforms parametric methods with greater inventory cost saving.

In this case study, the WSS Boostraping model with adapted jittering process will be used to obtained predicted value of yearly demand. The algorithms of the model are written by Ruby Programming Language from scratch.

1.2 Objectives of Research

The main objective of this research is to develop a forecasting web application to predict demand levels of spare parts. Thus, the main objective will be reached through the following sub-objectives:

- To identify the demand pattern of spare parts;
- To gain a better understanding of a systematic theory for a predictive model for forecasting spare parts demand;
- To establish competence in predicting spare parts needs and industry best practices in the area;
- To develop a web application to forecast demand of spare parts in order to ensure availability of spare parts.

1.3 Thesis methodology

The increase of the advent of technology allows us to develop a web application as an automated solution for material requirement planning. In this study, the main focus will be the development of a demand forecasting web application. The core algorithms of the application will consist of the adapted WSS Boostraping model closely following guidelines suggested by Willemain et al. (2004) and Rego and Mesqutia (2015) as reference materials.

The methodology will discuss more about data preparation, classification of demand pattern, and finally the bootstrapping model. To assess accuracy in forecasting, the true demand value for 2021 will be used to compare with predicted demand value.

The programming language used for bootstrapping algorithms is Ruby. For the user interface, we use HTML and CSS to provides the structure and style the web pages.

1.4 Limitation in the study

The limitation is mainly the lack of historical data available for usage (e.g. monthly demand time series and inventory cost). An assumption of monthly demand need to be made. More details about the assumption will be discuss further in Chapter 3, Section 3.2.1.

Chapter 2

Research Background

2.1 Related research

2.1.1 Demand patterns classification

The categorization of demand patterns is an essential element that helps facilitates the selection of a forecasting method. Syntetos et al (2005) proposed a method of demand pattern categorization based on a modification of Williams' work. The method considers both the squared coefficient of variation of demand sizes (CV^2) and average time interval factors (ADI). The study has been shown that these two parameters are very important from a forecasting perspective [10].

The CV^2 is defined as the square of the coefficient of variation:

$$CV^2 = \left(\frac{\text{Standard deviation of a population}}{\text{Average value of a population}}\right)^2 \tag{2.1}$$

The Average Demand Interval (ADI) measures the demand regularity in time by computing the average interval between two demands:

$$ADI = \left(\frac{\text{Total number of periods}}{\text{Number of demand buckets}}\right) \tag{2.2}$$

Based on the parameters, the demand pattern is classified into four different categories [15]:

- Smooth demand - the regular demand in time and in quantity.
- Intermittent demand - the demand with very little variation in demand quantity but a high variation in the interval between two demands.

- Erratic demand - the demand with regular occurrences in time and high quantity variations.
- Lumpy demand - the demand with a large variation in quantity and in time.

Figure 2.1 presents these four categorization schemes and their breakpoint values.

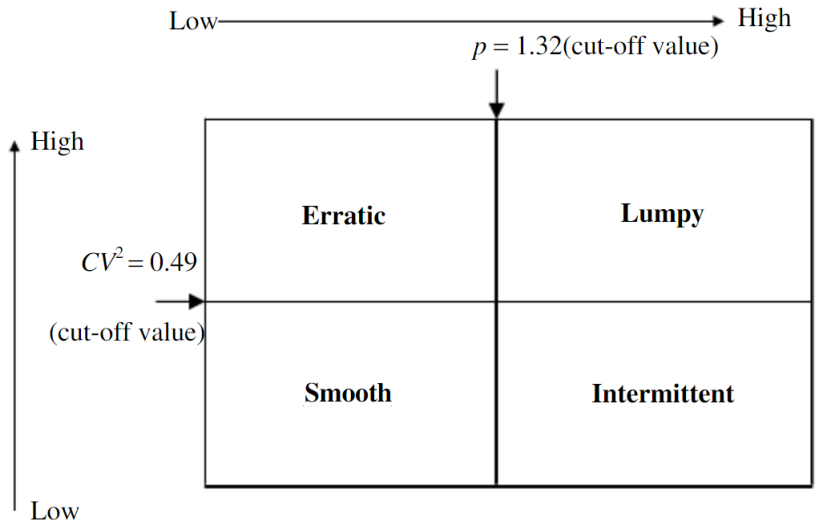


Figure 2.1: Demand pattern categorization (after Syntetos et al, 2005).

2.1.2 Forecasting intermittent demand

Literature shows several studies focusing on different aspects of forecasting methods for intermittent demand. Croston's method is the standard parametric method and the first method to forecast intermittent demand items [4]. In this method, demand probability and demand size are estimated separately. If a positive demand occurs in period t, the estimates of demand intervals and sizes are updated. Otherwise, if the demand is zero, none of the parameters are updated. The model can be summarized as the following:

$$\text{If } X_t \neq 0 \text{ then } \begin{cases} Z_{t+1} = \alpha X_t + (1-\alpha) Z_t \\ V_{t+1} = \alpha q + (1-\alpha) V_t \\ Y_{t+1} = \dfrac{Z_{t+1}}{V_{t+1}} \end{cases} \tag{2.3}$$

$$\text{If } X_t = 0 \text{ then } \begin{cases} Z_{t+1} = Z_t \\ V_{t+1} = V_t \\ Y_{t+1} = Y_t \end{cases} \tag{2.4}$$

Where Z_t is the estimate of mean non-zero demand size at time t, V_t is the estimate of mean interval size between non-zero demands at time t, Y_t denotes an estimate of mean demand size at time t, X_t denotes actual demand observed at time t, and and q denotes the current number of consecutive zero-demand periods.

Syntetos and Boylan (2001) identified a bias in Croston's method since $E\left[\bar{X}_t\right] = E\left[\frac{Z_t}{V_t}\right] \neq E\left[Z_t\right] \frac{1}{E[V_t]}$. In order to Croston's forecast X_t with a bias, the authors in [12] modified Croston's method, known as the Syntetos-Boylan Approximation (SBA), by multiplying the Croston estimate by a factor of $\left(1 - \frac{\alpha}{2}\right)$ and the expected estimate of demand per period will be:

$$E\left(Y_t'\right) = E\left(\frac{z_t'}{p_t'}\right) = \frac{E\left(z_t'\right)}{E\left(p_t'\right)} = \frac{\mu}{p} \tag{2.5}$$

However, Shenstone Hyndman (2005) showed that Croston's method and three related methods (including SBA) can be inconsistent with the properties of intermittent demand data [8]. In addition, there are some limitations as to the degree of lumpiness that can be effectively handled by a parametric distribution. For this reason, non-parametric bootstrapping approaches have a greater potential to provide further improvements in this area since it does not rely upon any underlying distributional assumptions. Moreover, the bootstrapping method can be useful when lacking high-quality data or historical data [2].

Alternative bootstrapping methods are available in the academic literature, namely Efron (1979), Snyder (2002), Willemain et al. (2004), Porras and Dekker (2008), Teunter and Duncan (2009), Zhou and Viswanathan (2011), and Snyder et al. (2012). The Willemain–Smart–Schwarz model (WSS) by Willemain et al. (2004) appears to be the most robust bootstrapping model, with its authors [13] and many empirical studies claiming significant improvements in forecasting accuracy over parametric approaches.

2.2 Model: Bootstrap method

In the 1970s, bootstrap method was first introduced by Brad Efron to estimate the sampling distribution of an observed sample. Bootstrap samples are obtained by randomly sampling with replacements from original data points.

This simple bootstrapping would ignore autocorrelation in the demand sequence and produce only the previous demand history numbers as forecast values. Willemain et al. (2004) proposed an adapted bootstrap method, called the WSS method. The method comes with two extensions to overcome the limitations of the original bootstrapping, including Discrete-Time Morkov Chain and Jittering.

2.2.1 Discrete-Time Markov Chains

A Discrete-Time Markov Chain is a system that describes transitions from one state to another state over discrete periods of time with a certain probability, also known as a stochastic process.

Definition

Let $\{X_t, t \in T\}$ be a stochastic process and $x_{[t_0,t]} = \{x_0, x_1, \ldots, x_i\}$ be the sequence of values assumed by the random variables X_t at $t_0 < t_1 < \cdots < t_i$.

The process $\{X_t, t \in T\}$ is a Markov Process if the transition probability to move at time $t + dt$ to the state x_j after the given evolution $x_{[0,t]} = \{x_0, x_1, \ldots, x_i\}$ depends only by the present state $x(t) = x_i$ and not by its whole history:

$$\begin{aligned} \Pr\left(x(t+dt) = x_j \mid x_{[0,t]}\right) &= \Pr\left(x(t+dt) = x_j \mid x(t) = x_i\right) \\ &= p_{ij} \end{aligned} \tag{2.6}$$

p_{ij} denotes is called the transition probability from state x_i to state x_j

The use of Discrete-Time Markov Chains

In the WSS bootstrapping method, Willemain et al. (2004) modeled a two-state Markov chain for autocorrelation in order to solve the problem of potential dependencies between demand occurrences. The process are divided into two stages:

- Stage 1: Estimating the occurrence (or non-occurrence) of demand in the next period, by taking the occurrence (or non-occurrence) of demand in the current period.
- Stage 2: Resampling the demand size if the forecast of the demand occurrence appears to be nonzero.

With Markov Chains, we take into account the probability of state transitions rather than simply randomly sample one of the previous periods.

2.2.2 Jittering

Willemain et al (2004) first proposed a modification to the sample bootstrapping method, called "Jittering". This technique prevents over-plotting in ordinal data and addresses some possibilities for observing demand values in the future that have never been observed before.

Let X* be a non-zero demand value that is randomly selected, and let Z be a standard normal random deviate. The jittering technique works as follows:

$$\text{Jittered value} = 1 + \text{integer}\left(X^* + Z\sqrt{X^*}\right) \tag{2.7}$$

$$\text{If jittered value} \leq 0, \text{ then jittered value} = X^*$$

However, the jittering technique creates a bias in the forecast of demand size. The jittering Equation (2.7) has an expected value of $E(J) = E(X) + 0.5$. This is because when taking the integer part of $X + Z\sqrt{X}$, we reduce it by an amount between 0 and 1 which gives an average reduction of 0.5 (assuming the reduction is uniformly distributed between 0 and 1). Therefore, $1 + INT(X + Z\sqrt{X})$ ends up giving an average increase of 0.5. Furthermore, the retention of the original demand value when the jittering process generates a negative demand value introduces a second bias as well. This step is sampling from a truncated normal distribution of Z with a highly negative of Z (which creates a negative value of J) being replaced by zero. It creates the bias since the mean of Z value is no longer zero but greater than zero.

To overcome the mentioned biases, Rego and Mesquita (2015) proposed an adaptation of the jittering process as follows:

$$\text{Jittered value} = \text{integer}\left(0.5 + X^* + Z\sqrt{X^*}\right) \tag{2.8}$$

$$\text{If jittered value} \leq 0, \text{ then jittered value} = 1$$

This simple modification eliminates the upward bias of 0.5, as well as reduce the second bias in the jittering process.

2.2.3 Bootstrapping procedure

In this study, we refer to the WSS bootstrapping method by Willemain et al. (2004) with an adapted jittering method by Rego and Mesqutia (2015). The method works according to the following steps [13, 17]:

- Step 1: Obtain historical demand data (including demand size and interarrival times) according to the chosen time bucket.

- Step 2: Estimate transition probabilities for two-state (zero vs. nonzero) Markov model.
- Step 3: Conditional on last observed demand, use Markov model to generate a sequence of zero/nonzero values over forecast horizon.
- Step 3: Replace nonzero state markers with a numerical value sampled at random with replacement from the set of observed nonzero demands.
- Step 4: Jitter the nonzero demand values.
- Step 5: Sum the forecast values over the horizon to get one predicted value.
- Step 6: Repeat steps 2 –5 many times.
- Step 7: Sort and use the resulting distribution of predicted values.

2.3 Forecasting accuracy measures

Many studies have adopted various accuracy metrics as evaluation criteria in the recent decades to evaluate the performance of forecasting methods. Shcherbakov et al. (2013) shows that every accuracy measure has its drawbacks and there is no perfect single measure that can be used universally. For this reason, the forecasting accuracy will be analyzed in a combination of both percentage error-based and absolute error-based measures. Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are the most frequently used criteria to evaluate the performance of the forecasting models.

2.3.1 Root Mean Square Error

Root Mean Square Error (RMSE) is commonly used as a general purpose error metric. It assess how well the predicted values from the model fit the actual values in the dataset. The formula to find the root mean square error is defined as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\frac{A_t - F_t}{A_t}\right)^2} \tag{2.9}$$

2.3.2 Mean Absolute Error

Mean Absolute Error (MAE) measures the absolute distance between the original values and predicted values. It provides an indication of the forecast accuracy by averaging the error term in the forecast for the whole time series. The formula is defined as follows:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|A_t - F_t| \tag{2.10}$$

2.3.3 Mean Absolute Percentage Error

Mean Absolute Percentage Error (MAPE) is the most widely used measure for checking forecast accuracy. It is calculated by dividing the absolute forecast error in each period by the actual value in that period and then averaging those fixed percentages. This can be expressed as a percentage given as:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right| \times 100 \tag{2.11}$$

Where:

- n is the number of observations
- A_t is the actual value at time t
- F_t is the forecast value at time t

Chapter 3

Research methodology and design

The methodology within this paper is based on the WSS bootstrapping method with the adapted jittering process. In this section, we will describe the main characteristics of the data sets and further highlight the data preparation, the classification of demand pattern and the use of the bootstrapping method to estimate the yearly demand for spare parts.

3.1 Industrial data sets

To illustrate the effectiveness of the model, we investigated a subsea operation and maintenance service provider in Norway. The company has more than 1600 types of spare parts, of which only about 10 percent of them have available historical demand data in both 2020 and 2021. Some spare parts that occurrences of nonzero demand in 2020 and 2021 less than once are excluded since these items lacked any basis for the model.

The historical data contain yearly consumption of spare parts, price per unit, stock quantity, stock value, criticality index, plant, storage location, and document year. In general, each item was reviewed and given a criticality score between 0 and 10 (0 = unimportant and 10 = absolutely critical). As representatives in this case study, we consider high-priority spare parts with criticality scores of 10 and 9. We sort the spare parts by their values and select the top 40 items for analysis.

3.2 Experimental Design

We intend to predict a yearly demand for 2021 by using yearly demand from 2020 and the earlier years (depending on data availability) as inputs. In order to assess the accuracy of the forecasting method, we compare the true value with the predicted value. Figure 3.1 represents the main process steps. More detail and an illustrative example will be provided for better understanding.

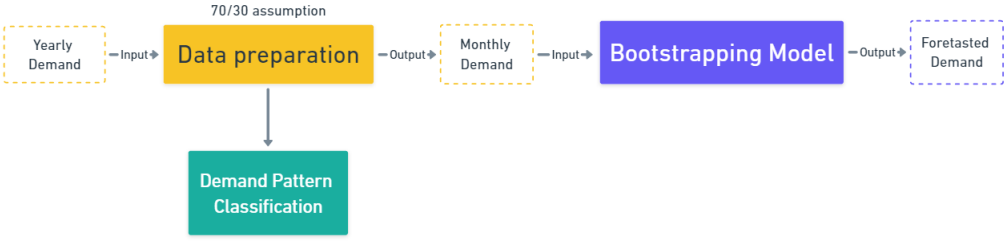


Figure 3.1: Process Mapping Diagram

3.2.1 Data preparation

Due to the unavailability of monthly demand, an assumption on how the annual demand is distributed throughout the year needs to be made. Repairs and maintenance work activities in the north sea are typically carried out in the summer months when the weather is favorable. Thus, we simulate monthly demand assuming 70 percent of the demands occur in the summer months and 30 percent in the non-summer month (hereafter referred to as "the 70/30 assumption").

Example: SKU-000001000000031794

To further illustrate the process of data preparation, we take the SKU-000001000000031794 as an example. The item has demands of 14 units in 2019, 12 units in 2020 and 10 units in 2021. We split the 3 years into two periods: the first period (2019-2020) is used to establish the relationship about the explanatory variables and estimate transition probabilities; the second period (2021) is held out for the purpose of assessing accuracy of the forecasting methods.

The monthly demand of the first period is simulated based on the yearly demand and the 70/30 assumption, obtaining a time series made up of 24

time points (from x_1 to x_{24}) as shown in Table 3.1. Figure 3.1 and 3.2 represent the simulated demand distributions throughout the year 2019 and 2020, respectively.

Table 3.1: SKU-000001000000031794 dataset

	Demand (EA)	The 70/30 assumption
2019	14	[1, 1, 1, 0, 1, 3, 3, 3, 0, 1, 0, 0]
2020	12	[0, 0, 1, 1, 1, 3, 3, 3, 0, 0, 0, 0]
2021	10	-

The simulated demand series is used as input data for the bootstrap simulation. For the second period (2021), the demand is held to assess the accuracy of the forecasting method.

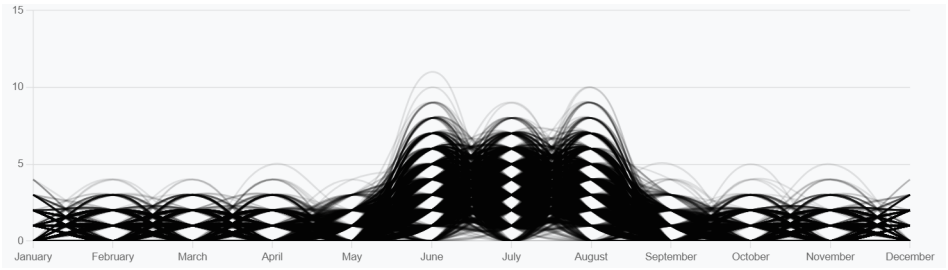


Figure 3.2: The simulation of monthly demand in 2019

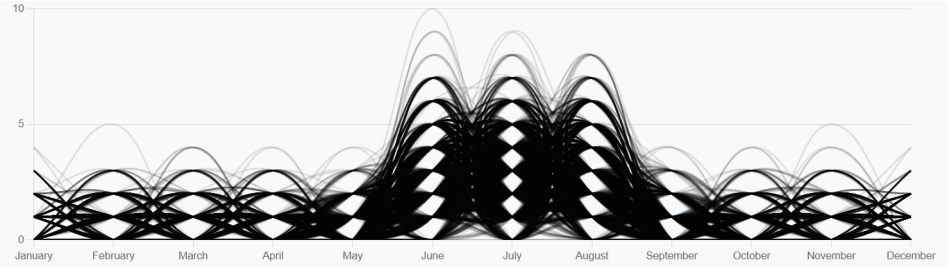


Figure 3.3: The simulation of monthly demand in 2020

3.2.2 Demand pattern of spare parts

To determine forecastability of spare parts, we classify the demand profiles based on the two parameters in the literature: the Average Demand Interval (ADI) and the square of the Coefficient of Variation (CV^2) [10].

Example: SKU-0000010000000031794

According to the data preparation process, the monthly demand time series that we simulated in the data preparation process are shown in the Table 3.2.

Table 3.2: Demand history over 24 months:

Period	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Demand Quantity	1	1	1	0	1	3	3	3	0	1	0	0
Period	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Demand Quantity	0	0	1	1	1	3	3	3	0	0	0	0

The Average Demand Interval (ADI) is equal to the average interval between two demands:

$$ADI = \left(\frac{\text{Total number of periods}}{\text{Number of demand buckets}}\right) = \frac{24}{14} = 1.7143$$

To compute the square of the Coefficient of Variation (CV^2), we consider only the non-zero values of the demand history.

$$CV^2 = \left(\frac{\text{Standard deviation of a population}}{\text{Average value of a population}}\right)^2 = \left(\frac{0.9897}{1.8571}\right)^2 = 0.2840$$

Therefore, this item is an **intermittent** demand pattern.

3.2.3 Implementation of Bootstrapping

For bootstrapping models, a first-order two-state Markov chain and the modification of jittering are used in the model to estimate demand, as suggested by Willemain et al. (2004) and Rego and Mesqutia (2015). The number of bootstrap forecasts is set 10,000 for sufficient estimation.

Example: SKU-0000010000000031794

Stage 1: Markov Chain

In the first stage, we resample occurrence or non-occurrence of demand, considering the occurrence or non-occurrence of demand in the previous period.

1. Transform the demand time series d_t (t = 1, 2, ..., n) into a binary time series y_t (t = 1, 2, ..., n). We record a day with a demand occurrence as '1' and a day without a demand occurrence as '0'.

   The actual demand time series:

   [1, 1, 1, 0, 1, 3, 3, 3, 0, 1, 0, 0, 0, 0, 1, 1, 1, 3, 3, 3, 0, 0, 0, 0]

   The binary time series:

   [1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

2. Estimate transition probabilities for two-state (zero vs. nonzero) Markov model. The four conditional probabilities that are relevant in this case are shown in Table 3.2.

   Table 3.3: Conditional probabilities of demand occurrences

	Demand today	**No demand today**
Demand yesterday	10/14 = 0.7143	4/14 = 0.2857
No demand yesterday	3/9 = 0.3333	6/9 = 0.6667

   Out of the 14 days with demand, there were 10 occasions when the next day also showed a demand, 4 when not. Of the 10 days without demand, the last observation must be discounted since we do not know what happened on the next day. Of the remaining 9 days without demand, 3 were followed by demand and 6 were not.

3. Generate a random number from a uniform distribution between 0 and 1 or X $\sim$ U(0, 1).

4. Check the most recent period of the demand history to see if there was a demand or not.

   If there was a demand, then compare the random number to the probability of demand today given that there was demand yesterday. If the random number is lower, record a value of 1 for demand occurrence; else, if it is higher, record a value of 0.

If there was no demand, then compare the random number to the probability of demand today given that there was no demand yesterday. If the random number is lower, record a value of 1 for demand occurrence; else, if it is higher, record a value of 0.

Table 3.4: Conditional probabilities of demand occurrences

	Demand Occurence			
	Period 25		**Period 26**	
Rep	**Rand**	**Occurs**	**Rand**	**Occurs**
1	0.2454	1	0.1455	1
2	0.4454	0	0.5657	0
3	0.9787	0	0.2777	1

The step 3 and 4 are presented in Table 3.3. For the first resampling, the conditional probability of demand occurrence in Period 25 is 0.3333, because there was no demand in Period 24. As the random number (Rand = 0.2454) is in the range from 0 to 0.333, this corresponds to a demand occurrence, indicated by a '1' in the 'Occurs' column. For Period 26, the conditional probability of demand occurrence is 0.7143 because we have just simulated a demand occurrence in Period 25. As the next random number (Rand = 0.1455) is in the range from 0 to 0.7143, this again corresponds to a demand occurrence. Demand occurrences are simulated in exactly the same way for the second, and third, replications.

Stage 2: Jittering

For stage 2, we resample the size of demand if stage 1 yields demand occurrence. The modification of the jittering process works as follows:

1. Resample a (positive) demand value, X, from the past periods.
2. Generate a standard normally distributed random variable, Z, with mean = 0 and variance = 1.
3. Calculate a jittering value, J, using Equation (2.8). If this value is less than or equal to 0, replace demand size by 1; otherwise, replace demand size by the jittered value.

Table 3.5: Jittering demand sizes

	Period 25				Period 26			
Rep	**X**	**Z**	**J**	**Demand Size**	**X**	**Z**	**J**	**Demand Size**
1	1	0.6545	2	2	1	-1.5225	0	1
2		-	-	-	-	-	-	-
3		-	-	-	3	0.4225	4	4

According to Table 3.3, there were three demand occurrences. The demand sizes are jittered and shown in Table 3.4. The demand sizes of the first replication in Period 25 and the third replication in Period 26 are replaced by jittered values, 2 and 4, respectively. For the first replication in Period 26, the demand size is replaced by 1 due to the fact that the jittered value is equal to 0.

4. Sum the forecast values over the horizon to get one predicted value.

The bootstrapping process is repeated until we have enough bootstrap forecasts (1000 BS in this case) for estimating the entire distribution of yearly demand.

Chapter 4

Application framework

This section will briefly introduce the main components of the application and the main algorithms that we use for constructing the bootstrapping model.

4.1 Components

The application database works through PostgreSQL, an open-source relational database management system emphasizing extensibility and SQL compliance. It compiles to regular HTML, CSS and Ruby.

Figure 4.1 illustrates the Model-View-Controller (MVC) architecture of the web application. The principle of information flows of the system is implemented in accordance with MVC. Consider the example:

1. The client goes to the website URL / items (items list).
2. The browser sends a request to the server.
3. On the server, the request is accepted by the Rails router (routes.rb) and determines which method of the controller to transfer control. In this case it is the index method.
4. The index method of the items_controller takes control. It sends a query to the Item model, which in turn using ActiveRecord technology generates a query in SQL for the database used and in accordance with the parameters, generates a set of data from the database and sends back them to the controller method.
5. The index controller method processes the data and sends it to the Index view, which generates (renders) the HTML code (list of items) and returns it to the controller.

6. The controller performs a final check and sends the generated page to the client's browser via HTTP server (Puma in this case).

7. The browser receives the page and displays it to the user.

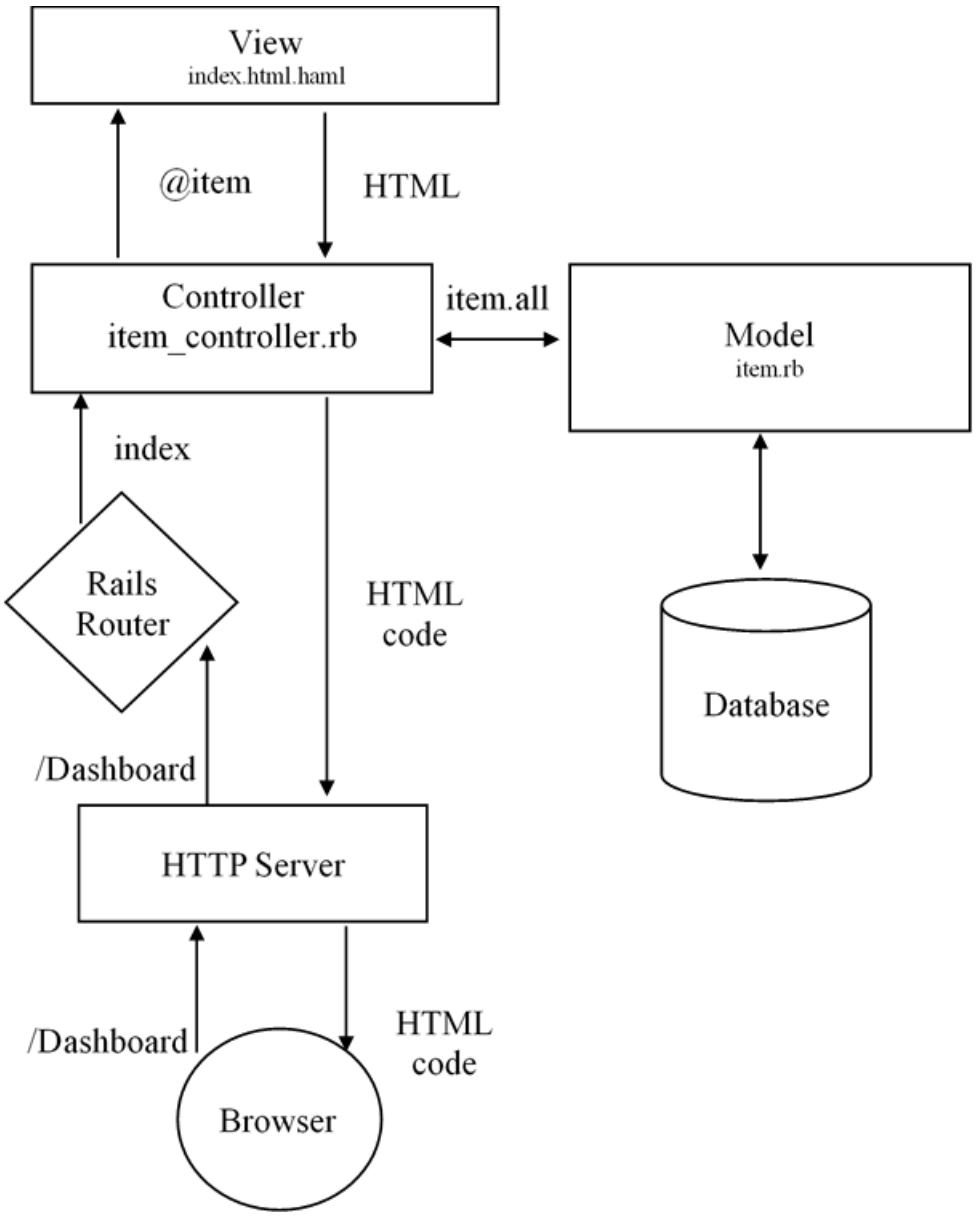


Figure 4.1: The Model-View-Controller (MVC) architecture

4.2 Algorithms

The algorithms of data preparation, demand pattern of spare parts, and bootstrapping are written in Ruby Programming Language.

In Ruby, we use *initialize* method to set the initial values (input data) when the object of the class is instantiated and assign it to the instance variables of the class. The *call* method is a publicly available method that performs the calculations of the class and controls the flow of the algorithm. It's main method that should be used outside of the class and it shapes the class to follow single responsibility principle.

According to a class that handles end-user requests and generates HTML page with the results of calculations (*ItemsController*), more details related to the algorithms of *ItemsController* will be provided in Appendix A.

4.2.1 Data preparation

In this this step, the logic behind the algorithm is based on the 70/30 assumption.

Algorithm 1: Data preparation

```
class GenerateDataService
  attr_reader :predicted_demand
  attr_reader :prediction_result

  def initialize(consumed_qty:)
    @consumed_qty = consumed_qty
  end

  def call
    @predicted_demand = generate_predicted_demand
    @mean = calculate_mean_by_month
    @prediction_result = round_mean_prediction
    @prediction_result
  end

  private
  def generate_predicted_demand
    pseudo_random_number = []
    predicted_demand = []
    occurence = Array.new(12, 0)
```

```
    1000.times do
      pseudo_random_number = Array.new(@consumed_qty) { r.rand() }
      @occurence_of_demand_MC = occurence.dup
      pseudo_random_number.each do |elements|
        perform_monte_carlo(elements)
      end
      predicted_demand << @occurence_of_demand_MC
    end
    predicted_demand
  end

  def perform_monte_carlo(elements)
    case elements
    when 0 .. 0.03333333333 # January
        @occurence_of_demand_MC[0] += 1
    when 0.03333333334 .. 0.06666666666 #February
        @occurence_of_demand_MC[1] += 1
    when 0.06666666667 .. 0.09999999999 #March
        @occurence_of_demand_MC[2] += 1
    when 0.10000000000 .. 0.13333333332 #April
        @occurence_of_demand_MC[3] += 1
    when 0.13333333333 .. 0.16666666665 #May
        @occurence_of_demand_MC[4] += 1
    when 0.16666666666 .. 0.39999999998 #June
        @occurence_of_demand_MC[5] += 1
    when 0.39999999999 .. 0.63333333331 #July
        @occurence_of_demand_MC[6] += 1
    when 0.63333333332 .. 0.86666666664 #August
        @occurence_of_demand_MC[7] += 1
    when 0.86666666665 .. 0.89999999997 #September
        @occurence_of_demand_MC[8] += 1
    when 0.89999999998 .. 0.9333333333 #October
        @occurence_of_demand_MC[9] += 1
    when 0.9333333333 .. 0.96666666663 #November
        @occurence_of_demand_MC[10] += 1
    else #December
        @occurence_of_demand_MC[11] += 1
    end
  end
```

```
  def calculate_mean_by_month
    @predicted_demand.transpose.map do |demands_by_month|
      demands_by_month.sum(0.0) / demands_by_month.size
    end
  end

  def round_mean_prediction
    result = Array.new(12, 0)
    while result.sum < @consumed_qty
      max_index = @mean.each_with_index.max.last
      if @mean[max_index] > 1.0
        result[max_index] = @mean[max_index].round
      else
        result[max_index] = 1
      end
      @mean[max_index] = 0
    end
    result
  end
end
```

Algorithm 1 explanation

initialize :	Define @*consumed_qty* as an input
call :	Call the methods: *generate_predicted_demand*, *calculate_mean_by_month*, and *round_mean_prediction*
generate_predicted_demand :	Generate 1000 pseudo random numbers to perform Monte Carlo.
perform_monte_carlo :	Iterate over the array of pseudo-random number. If the random number equals 0 to 0.03333333333, mark as "1 unit of spare parts was used in January" and so forth.
calculate_mean_by_month :	Average the demand size each month.
round_mean_prediction :	Choose maximum numbers of mean and round it to a nearest number until it hits the yearly demand of spare parts. The higher value of mean is, the more likely that demand would occur.

4.2.2 Demand pattern of spare parts

In order to classify demand pattern, we need to calculate ADI and CV^2 then compare the ADI and CV^2 values with the cutoffs.

Algorithm 2: Demand pattern of spare parts

```
# Calculate ADI
demand_buckets = []

@prediction_result.each do |element|
    if element > 0
        demand_buckets << element
    end
end

adi = @prediction_result.count.to_f/demand_buckets.count.to_f

# Calculate CV^2
average_value_of_pop = demand_buckets.sum(0.0)/demand_buckets.count

x_bar = @prediction_result.sum(0.0)/@prediction_result.count

sum = demand_buckets.sum(0.0) { |demand| (demand -
    average_value_of_pop) ** 2 }

standard_deviation = Math.sqrt(sum / (demand_buckets.size.to_f))

square_coefficient_of_variation =
    (standard_deviation/average_value_of_pop)**2

# Compare with cutoffs
def perform_demand_category(adi, square_coefficient_of_variation)
    if adi <= 1.34 && square_coefficient_of_variation <= 0.49
        p "Smooth"
    elsif adi >= 1.34 && square_coefficient_of_variation >= 0.49
        p "Lumpy"
    elsif adi < 1.34 && square_coefficient_of_variation > 0.49
        p "Erratic"
    elsif adi > 1.34 && square_coefficient_of_variation < 0.49
        p "Intermittent"
    end
end
```

Algorithm 2 explanation

$@prediction\_result.count.to\_f$:	Count the number of elements in $@prediction\_result$ and convert the values of the numbers as float.
$demand\_buckets.count.to\_f$:	Count the number of elements in $demand\_buckets.count.to\_f$ (how many non-zero demand occurrence in demand time series) and convert the values of the numbers as float.
adi :	Calculate adi by using Equation (2.1).
$average\_value\_of\_pop$:	Average value of $demand\_buckets$.
x_bar :	Average value of $@prediction\_result$.
$standard\_deviation$:	Calculate standard deviation of $@demand_buckets$.
$square\_coefficient\_of\_variation$:	Calculate the square of the CV.
$perform\_demand\_category$:	Compare with cutoffs ADI and CV^2 with the cutoffs.

4.2.3 Implementation of Bootstrapping

In this process, we estimate the occurrence (or non-occurrence) of demand by using Markov Chains and resample the demand size if the forecast of the demand occurrence appears to be nonzero by using the adapted jittering process.

Algorithm 3: Initialize inputs and call methods

```
class PredictNextMonthService
  def initialize(new_demands:, original_demand:)
    @new_demand = nil
    @all_demands = original_demand + new_demands
    @original_demand = original_demand
  end

  def call
    set_occurence_of_demand
    calculate_month_with_without_demands
    calculate_4_conditional_probabilities
    set_rand_uniform_distribution
    add_new_occurance
    add_forecast_value_to_demand
    @new_demand
  end
```

Algorithm 3 explanation

initialize :	Define *new_demand* and *original_demand* as inputs
call :	Call the methods: *set_occurence_of_demand*, *calculate_month_with_without_demands*, *calculate_4_conditional_probabilities*, *set_rand_uniform_distribution*, *add_new_occurance*, and *add_forecast_value_to_demand*. Return the variable: @*new_demand*.

Algorithm 4: Markov Chains

```
private
  def set_occurence_of_demand
    @occurrence_of_demand = @all_demands.map do |d|
      d > 0 ? 1 : d
    end
  end

  def calculate_month_with_without_demands
    months_with_demands = @occurrence_of_demand.count(1)
    months_without_demands = @occurrence_of_demand.count(0)

    @number_of_months_with_demands = months_with_demands
    @number_of_months_without_demands = months_without_demands

    if @occurrence_of_demand.last == 0
      @number_of_months_without_demands -= 1
    elsif @occurrence_of_demand.last == 1
      @number_of_months_with_demands -= 1
    end
  end

  def calculate_4_conditional_probabilities

    demand_and_demand = 0
    demand_and_no_demand = 0
    no_demand_and_demand = 0
    no_demand_and_no_demand = 0

    @occurrence_of_demand.each_cons(2) do |occurance_a, occurance_b|
      if occurance_a == 1 && occurance_b == 1
        demand_and_demand += 1
      end
      if occurance_a == 1 && occurance_b == 0
        demand_and_no_demand += 1
      end
      if occurance_a == 0 && occurance_b == 1
        no_demand_and_demand += 1
      end
      if occurance_a == 0 && occurance_b == 0
```

```
      no_demand_and_no_demand += 1
    end
  end

  @probability_of_demand_and_demand = demand_and_demand.to_f /
      @number_of_months_with_demands.to_f

  @probability_of_demand_and_no_demand =
      demand_and_no_demand.to_f /
      @number_of_months_with_demands.to_f

  @probability_of_no_demand_and_demand =
      no_demand_and_demand.to_f /
      @number_of_months_without_demands.to_f

  @probability_of_no_demand_and_no_demand =
      no_demand_and_no_demand.to_f /
      @number_of_months_without_demands.to_f
end

def set_rand_uniform_distribution
  @rand_uniform_distribution = Random.rand()
end

def add_new_occurance
  @occurrence_of_demand << decide_if_demand_occurs
end

def decide_if_demand_occurs
  if @occurrence_of_demand.last == 1
    @rand_uniform_distribution <
        @probability_of_demand_and_demand ? 1 : 0
  else
    @rand_uniform_distribution <
        @probability_of_no_demand_and_demand ? 1 : 0
  end
end

def add_forecast_value_to_demand
  if @occurrence_of_demand.last == 1
    @new_demand = perform_jittering
  else
    @new_demand = 0
```

```
    end
  end

  def perform_jittering
    Jittering.new(original_demand: @original_demand).call
  end
end
```

Algorithm 4 explanation

set_occurence_of_demand :	Record a month with a demand occurrence as '1' and a month without a demand occurrence as '0' into @*occurrence_of_demand*.
calculate_month_with_without_demand :	Count how many months with demand and without demand. Discount the last observation.
calculate_4_conditional_probabilities :	Calculate 4 conditional probabilities: @*probability_of_demand_and_demand*, @*probability_of_demand_and_no_demand*, @*probability_of_no_demand_and_demand*, @*probability_of_no_demand_and_no_demand*.
set_rand_uniform_distribution :	Generate uniform random numbers between 0 and 1.
add_new_occurance :	Append occurrences of demand into @*occurrence_of_demand* by using results from *decide_if_demand_occurs*.
decide_if_demand_occurs :	Compare @*rand_uniform_distribution* with @*probability_of_demand_and_demand* or @*probability_of_no_demand_and_demand* (depending on current demand).
add_forecast_value_to_demand	Perform *perform_jittering* and replace demand size with jitterd value if there is a demand occurrence.
perform_jittering	Call class Jittering to perform jittering process.

Algorithm 5: Jittering Process

```
class Jittering
  def initialize(original_demand:)
    @demand = original_demand
  end

  def call
    pick_demand
    generate_guassian_random_numbers
    jittering
  end

  private

  def pick_demand
    picked_demands = []
    @demand.each do |d|
      picked_demands << d if d > 0
    end

    @demand_sample = picked_demands.sample
  end

  def generate_guassian_random_numbers
    normal_distribution_generrator = RandomGaussian.new(0, 1)
    @rand_normal_distribution = normal_distribution_generrator.rand
  end

  def jittering
    jittered_demand = nil
    value = 0.5 + @demand_sample + (@rand_normal_distribution *
        Math.sqrt(@demand_sample))
    if value > 0
      jittered_demand = value.to_i
    else
      jittered_demand = 1
    end
    jittered_demand
  end
end
```

Algorithm 5 explanation

initialize	Define *original_demand* as an input
call	Call the methods: *pick_demand*, *generate_guassian_random_numbers*, *jittering*.
pick_demand	Randomly pick non-zero demand from original demand and save into @*demand_sample*.
generate_guassian_random_numbers	Generate Gaussian distributed numbers with mean = 0 and variance = 1.
jittering	Calculate jittered value by using Equation (2.8). If jittered value is more than 0, replace demand size with the jittered value, otherwise replace it with 1.

Algorithm 6: RandomGaussian

```
class RandomGaussian
  def initialize(mean, stddev, rand_helper = lambda { Kernel.rand })
    @rand_helper = rand_helper
    @mean = mean
    @stddev = stddev
    @valid = false
    @next = 0
  end

  def rand
    if @valid then
      @valid = false
      return @next
    else
      @valid = true
      x, y = self.class.gaussian(@mean, @stddev, @rand_helper)
      @next = y
      return x
    end
  end

  private
  def self.gaussian(mean, stddev, rand)
    theta = 2 * Math::PI * rand.call
    rho = Math.sqrt(-2 * Math.log(1 - rand.call))
    scale = stddev * rho
    x = mean + scale * Math.cos(theta)
    y = mean + scale * Math.sin(theta)
    return x, y
  end
end
```

Algorithm 6 explanation

initialize	Define *mean*, *stddev*, and *rand_helper* as inputs.
rand and *self.gaussian*	Use the Box-Muller transform method to generate uniformly distributed random numbers.

According to a class that handles end-user requests and generates HTML page with the results of calculations (*ItemsController*), more details related to the algorithms of *ItemsController* will be provided in Appendix A.

Chapter 5

Analysis and Results

5.1 Classification of demand pattern

According to Syntetos et al. (2005) proposal, we classify 40 items by using monthly demand from the year 2020. The coefficient of variation (CV^2) and the average demand interval (ADI) are computed by Equation (2.1) and (2.2) respectively, resulting in Table 4.1. Figure 4.1 displays the results in a scatter plot to identify general characteristics. Most of the spare part (95%) have very low size variability but high average demand interval, being classified as "Intermittent". The rest of them (5%) are classified as "Erratic" because of their low average demand interval and high size variability.

The bootstrap methodology may be beneficial in this case study since many researches show that it has been proved to be appropriate and outperform other methods when dealing with intermittent demand items.

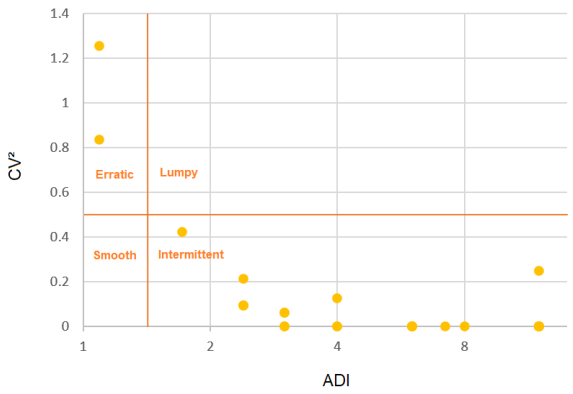


Figure 5.1: The simulation of monthly demand in 2019

Table 5.1: Demand pattern classification

Material	ADI	The square of CV	Category
83915	4.00	0.13	Intermittent
80053	2.40	0.09	Intermittent
86181	6.00	0.00	Intermittent
87754	4.00	0.00	Intermittent
84270	1.09	1.25	Erratic
86023	6.00	0.00	Intermittent
13572	2.40	0.21	Intermittent
86866	12.00	0.00	Intermittent
75807	6.00	0.00	Intermittent
84267	1.09	0.84	Erratic
88362	3.00	0.06	Intermittent
88363	8.00	0.00	Intermittent
62191	6.00	0.00	Intermittent
07176	7.20	0.00	Intermittent
77504	12.00	0.00	Intermittent
80885	3.00	0.00	Intermittent
65586	1.71	0.42	Intermittent
85995	6.00	0.00	Intermittent
83453	3.00	0.06	Intermittent
76831	12.00	0.00	Intermittent
90471	12.00	0.00	Intermittent
86927	12.00	0.00	Intermittent
86926	12.00	0.00	Intermittent
71042	12.00	0.00	Intermittent
64729	6.00	0.00	Intermittent
87399	12.00	0.00	Intermittent
87291	12.00	0.00	Intermittent
84164	3.00	0.00	Intermittent
64728	12.00	0.00	Intermittent
64703	12.00	0.00	Intermittent
80344	12.00	0.00	Intermittent
84275	12.00	0.25	Intermittent
80813	12.00	0.00	Intermittent
53180	4.00	0.00	Intermittent
84186	12.00	0.00	Intermittent
07202	12.00	0.00	Intermittent
53188	12.00	0.00	Intermittent
75070	2.40	0.09	Intermittent
53187	12.00	0.00	Intermittent
56728	2.40	0.09	Intermittent

5.2 Validation of Bootstrapping Model

After all the computational works, basic statistical analysis results of the forecast are obtained and listed in Table 5.2. It becomes clear from the mean of the forecast value (F_t), that this model's accuracy oscillates between under-predicting and over-predicting future demand. Some examples of the bootstrapping results discussed in this chapter will be presented in the Appendix B.

In Table 5.3, the 90%, 95%, and 99% forecast intervals for each item the results are reported. When the actual demand (A_t) is inside the interval, it is highlighted in **bold**. It can be seen that there are 31 of 40 items whose the actual demand is inside the 99% confidence interval. Whereas, the 90% and 95% bootstrapped confidence interval do not often contain the actual demand. Of 40 spare parts, there are 21 and 24 items that contain the actual demand respectively for the 90% and 95% confidence interval. Hence, the 99% confidence interval of the mean of the predicted demand seems to be the best compromise.

Considering the performance of the forecasting model, measurement statistics are calculated further by using the actual demand (A_t) in 2021 and the mean of the forecast value (F_t). The proper measures of fit are the Mean absolute error (MEA) = 0.2735 and the Root Mean Square Error (RMSE) = 0.4349, calculated from Equation (2.9) and Equation (2.10) respectively. Based on the minimum RMSE required (≤ 1) [3], the actual demand fits the mean of the predicted demand quite well. Mean absolute percentage error (MAPE) is used as a relative error measure. According to Equation 2.11, MAPE is equivalent to 12.63% which is recognized as good forecasting [1].

In the context of the entire experiment, the model gives fairly good result and the accuracy and error rate are satisfied. We can conclude the WSS bootstrapping methods with the adapted jittering process appears to be a good compromise for obtaining demand forecast intervals. Statistical analysis indicated acceptance of model simulations.

Table 5.2: Statistical analysis results

Material	**Actual Demand (A_t)**	**Mean of Forecast Value (F_t)**	**S.D.**
83915	3	3.3060	4.5844
80053	8	7.7270	5.8012
86181	1	1.3940	1.7694
87754	3	3.1220	2.8315
84270	36	35.3410	14.5913
86023	2	2.0140	2.8976
13572	5	5.6530	5.1500
86866	1	1.1090	1.6053
75807	2	2.0300	3.0158
84267	52	51.8190	16.7117
88362	8	7.6500	5.8302
88363	2	1.5370	2.1542
62191	2	1.9690	2.8025
7176	2	1.6750	2.3159
77504	1	1.1010	1.4237
80885	3	2.8880	3.7799
65586	15	14.7340	8.5413
85995	1	1.7380	2.5106
83453	4	3.9710	4.9952
76831	1	1.0390	1.5269
90471	1	1.0560	1.4159
86927	1	1.0790	1.5195
86926	1	1.1120	1.5529
71042	1	1.0480	1.4972
64729	2	1.9820	2.9332
87399	1	1.1010	1.5617
87291	1	1.0630	1.5281
84164	2	2.1460	2.8870
64728	2	1.2590	1.6346
64703	1	1.0740	1.5134
80344	2	1.2100	1.6745
84275	11	11.2560	7.4930
80813	1	1.1280	1.6400
53180	2	2.2120	3.2136
84186	3	1.2050	1.6826
7202	2	1.2170	1.6637
53188	1	1.0380	1.4740
75070	4	4.1600	4.9659
53187	1	1.0880	1.5602
56728	6	5.9420	6.5225

Table 5.3: Confidence interval of 90%, 95% and 99%

Material	$\mathbf{A}_t$	**Confidence Interval**		
		90%	**95%**	**99%**
83915	3	3.0675 : 3.5445	3.0219 : 3.5901	**2.9326 : 3.6794**
80053	8	**7.4252 : 8.0288**	**7.3674 : 8.0866**	**7.2544 : 8.1996**
86181	1	1.3020 : 1.4860	1.2843 : 1.5037	1.2499 : 1.5381
87754	3	**2.9747 : 3.2693**	**2.9465 : 3.2975**	**2.8913 : 3.3527**
84270	36	**34.5820 : 36.1000**	**34.4366 : 36.2454**	**34.1524 : 36.5296**
86023	2	**1.8633 : 2.1647**	**1.8344 : 2.1936**	**1.7780 : 2.2500**
13572	5	5.3851 : 5.9209	5.3338 : 5.9722	5.2335 : 6.0725
86866	1	1.0255 : 1.1925	1.0095 : 1.2085	**0.9782 : 1.2398**
75807	2	**1.8731 : 2.1869**	**1.8431 : 2.2169**	**1.7843 : 2.2757**
84267	52	**50.9497 : 52.6883**	**50.7832 : 52.8548**	**50.4577 : 53.1803**
88362	8	7.3467 : 7.9533	**7.2886 : 8.0114**	**7.1751 : 8.1249**
88363	2	1.4249 : 1.6491	1.4035 : 1.6705	1.3615 : 1.7125
62191	2	**1.8232 : 2.1148**	**1.7953 : 2.1427**	**1.7407 : 2.1973**
07176	2	1.5545 : 1.7955	1.5315 : 1.8185	1.4863 : 1.8637
77504	1	1.0269 : 1.1751	1.0128 : 1.1892	**0.9850 : 1.2170**
80885	3	**2.6914 : 3.0846**	**2.6537 : 3.1223**	**2.5801 : 3.1959**
65586	15	**14.2897 : 15.1783**	**14.2046 : 15.2634**	**14.0382 : 15.4298**
85995	1	1.6074 : 1.8686	1.5824 : 1.8936	1.5335 : 1.9425
83453	4	**3.7112 : 4.2308**	**3.6614 : 4.2806**	**3.5641 : 4.3779**
76831	1	**0.9596 : 1.1184**	**0.9444 : 1.1336**	**0.9146 : 1.1634**
90471	1	**0.9823 : 1.1297**	**0.9682 : 1.1438**	**0.9407 : 1.1713**
86927	1	1.0000 : 1.1580	**0.9848 : 1.1732**	**0.9552 : 1.2028**
86926	1	1.0312 : 1.1928	1.0158 : 1.2082	**0.9855 : 1.2385**
71042	1	**0.9701 :1.1259**	**0.9552 : 1.1408**	**0.9260 : 1.1700**
64729	2	**1.8294 : 2.1346**	**1.8002 : 2.1638**	**1.7431 : 2.2209**
87399	1	1.0198 : 1.1822	1.0042 : 1.1978	**0.9738 : 1.2282**
87291	1	**0.9835 : 1.1425**	**0.9683 : 1.1577**	**0.9385 : 1.1875**
84164	2	**1.9958 : 2.2962**	**1.9671 : 2.3249**	**1.9108 : 2.3812**
64728	2	1.1740 : 1.3440	1.1577 : 1.3603	1.1258 : 1.3922
64703	1	**0.9953 : 1.1527**	**0.9802 : 1.1678**	**0.9507 : 1.1973**
80344	2	1.1229 : 1.2971	1.1062 : 1.3138	1.0736 : 1.3464
84275	11	**10.8662 : 11.6458**	**10.7916 : 11.7204**	**10.6456 : 11.8664**
80813	1	1.0427 : 1.2133	1.0264 : 1.2296	**0.9944 : 1.2616**
53180	2	2.0448 : 2.3792	2.0128 : 2.4112	**1.9502 : 2.4738**
84186	3	1.1175 : 1.2925	1.1007 : 1.3093	1.0679 : 1.3421
07202	2	1.1305 : 1.3035	1.1139 : 1.3201	1.0815 : 1.3525
53188	1	**0.9613 : 1.1147**	**0.9466 : 1.1294**	**0.9179 : 1.1581**
75070	4	**3.9017 : 4.4183**	**3.8522 : 4.4678**	**3.7555 : 4.5645**
53187	1	1.0068 : 1.1692	**0.9913 : 1.1847**	**0.9609 : 1.2151**
56728	6	**5.6027 : 6.2813**	**5.5377 : 6.3463**	**5.4107 : 6.4733**

Chapter 6

Discussion

The results from 1000 replicates for each item support the expected hypothesis. The findings show that the modified WSS bootstrap method performs very well. According to the evaluation metrics, Root Mean Square Error (RMSE), Mean Absolute Error (MEA), and Mean Absolute Percentage Error (MAPE) are in the acceptable accuracy ranges. The proposed bootstrapping model can therefore be considered as a good forecaster for a subsea operation and maintenance service provider in the case study. The 99% confidence interval often contain the true value of demand.

The main issue of this thesis is an unavailability of historical data. Most often, the model requires historical data of monthly demand for greater than 2 year to detect general trends. In fact, there are 4 out of 40 spare parts that have data available for at least two years and the data are reported in yearly consumption. Using only 1 year historical data and the 70/30 assumption to obtain monthly demand, may induce bias and weak estimation of the model.

Another interesting point is the performance measurement of the model. RMSE, MEA, and MAPE can be used for general purposes but they are not suitable for the application of certain data. The mentioned metrics disregard economical aspects, while it is very important for spare parts management to consider stock holding cost and stockout cost. These traditional metrics may not represent the actual performance of the model and lead to a possibly biased evaluation criterion. This bias can be eliminated by using a new performance measurement for intermittent demand forecasts proposed by Martin et al (2020), called Stock-keeping-oriented Prediction Error Costs (SPEC). But due to the unavailability of data in terms of cost factors, we were not able to implement this metric to measure the performance of the model.

Several improvements can be done if high-quality historical data are available and accessible. We therefore recommend to collect more data to improve the accuracy of the model and use the SPEC metric in future studies to measure the actual quality of the model.

Appendix A

Algorithm Code

Algorithm 7: ItemsController

```
require "histogram/array"

class ItemsController < ApplicationController
  before_action :set_item, only: %i[ edit update destroy ]

  def index
    @items = Item.all
  end

  def import
    counter = Item.import params[:file]
    redirect_to items_path, notice: "Imported #{counter} items"
  end

  def show
    @item = Item.friendly.find(params[:id])
    perform_bootstrapping
  end

  def new
    @item = Item.new
  end

  def edit
  end
```

```
def create
  @item = Item.new(item_params)

  if @item.save
    redirect_to item_url(@item), notice: "Item was successfully
        created."
  else
    render :new, status: :unprocessable_entity
  end
end

def update
  if @item.update(item_params)
    redirect_to item_url(@item), notice: "Item was successfully
        updated."
  else
    render :edit, status: :unprocessable_entity
  end
end

def destroy
  @item.destroy

  redirect_to items_url, notice: "Item was successfully
      destroyed."
end

private

def set_item
  @item = Item.find(params[:id])
end

def item_params
  params.require(:item).permit(:name, :demand_time_series)
end

def perform_monte_carlo
  service = GenerateDataService.new(consumed_qty: 14)
  service.call
  @predicted_demand = service.predicted_demand.map.with_index do
      |monte_result,index|
    {
```

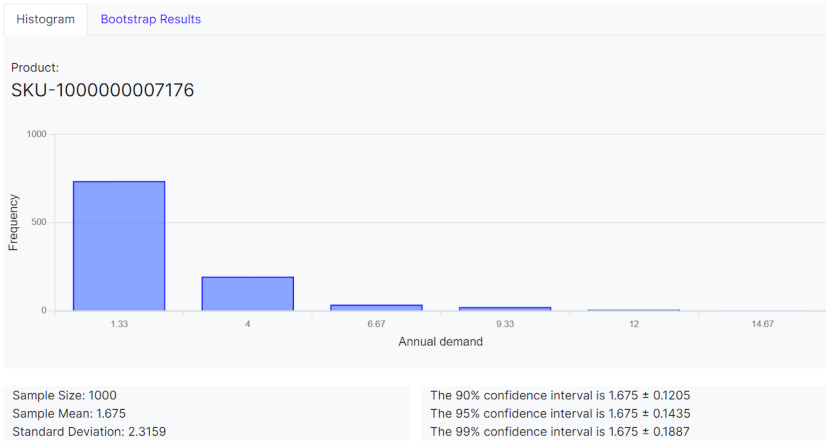
```
      name: index,
      data: monte_result.map.with_index {|mb,index|
          [Date::MONTHNAMES[index+1],mb]}
    }
  end

  @prediction_result = service.prediction_result
  @random_seed = service.random_seed
end

def perform_bootstrapping
  original_demand = @item.demand_time_series

  number_of_orginal_demand = original_demand.size
  @sum_original_demand = original_demand.sum(0.0)

  @predictions = demand_forecast_values(original_demand)
  @graphable_demand_forecast_values =
      graphable_demand_forecast_values(@predictions,
      original_demand)
  @graphable_colors = ["#0d6efd", "#0d6efd"] +
      Array.new(@graphable_demand_forecast_values.count - 2,
      "#212529")
  @graphable_histogram = @predictions.flatten.histogram.transpose

  @sum_of_demand_in_each_array = @predictions.map do |demands|
      @sum_of_demand_in_each_array = demands.sum(0.0)
  end

  @graphable_sum_of_demand_in_each_array =
      @sum_of_demand_in_each_array.histogram(6)

  @graphable_sum_of_demand_in_each_array[0] =
      @graphable_sum_of_demand_in_each_array[0].map {|el|
      el.round(2)}

  @graphable_sum_of_demand_in_each_array =
      @graphable_sum_of_demand_in_each_array.transpose

  @mean_of_all_array = @sum_of_demand_in_each_array.sum(0.0) /
      @sum_of_demand_in_each_array.size

  sum = @sum_of_demand_in_each_array.sum(0.0) { |demand| (demand
```

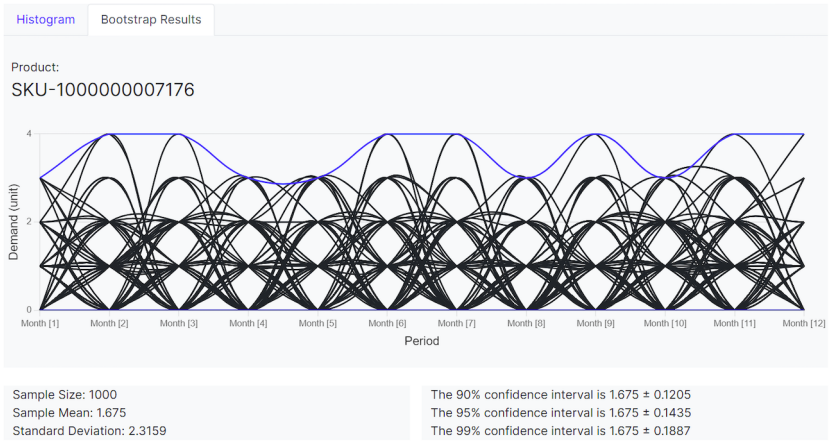
```
      - @mean_of_all_array) ** 2 }

  @standard_deviation = Math.sqrt(sum /
      (@sum_of_demand_in_each_array.size))

  @confidence_interval_90 =
      1.645*(@standard_deviation/Math.sqrt(@sum_of_demand_in_each_array.size))

  @confidence_interval_95 =
      1.960*(@standard_deviation/Math.sqrt(@sum_of_demand_in_each_array.size))

  @confidence_interval_99 =
      2.576*(@standard_deviation/Math.sqrt(@sum_of_demand_in_each_array.size))

  @confidence_interval_99_99 =
      3.291*(@standard_deviation/Math.sqrt(@sum_of_demand_in_each_array.size))
end

def demand_forecast_values(original_demand)
  predictions = []
  @number_of_samples = 1000
  number_of_forecasted_period = 12

  @number_of_samples.times do
    demands = []
    number_of_forecasted_period.times do
      demands << PredictNextMonthService.new(new_demands:
          demands, original_demand: original_demand).call
    end
    predictions << demands
  end
  predictions
end

def graphable_demand_forecast_values(predictions, original_demand)
  predictions_with_min_max = predictions.dup
  max_numbers = predictions_with_min_max.transpose.map(&:max)
  min_numbers = predictions_with_min_max.transpose.map(&:min)

  predictions_with_min_max =
      predictions_with_min_max.unshift(max_numbers, min_numbers)
  graphable_data = predictions_with_min_max.map.with_index do
      |demands, index|
```

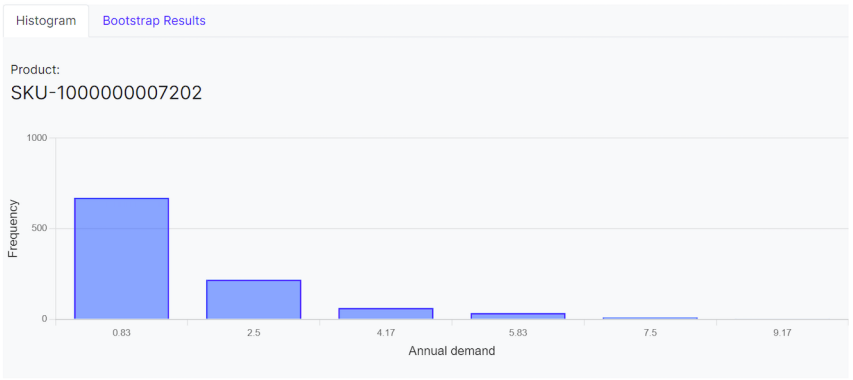
```
      {
        name: index,
        data: demands.map.with_index {|pr,index| ["Month
            #{[index+1]}",pr]}
      }
    end
  end
end
```

Appendix B

Examples of result on UI

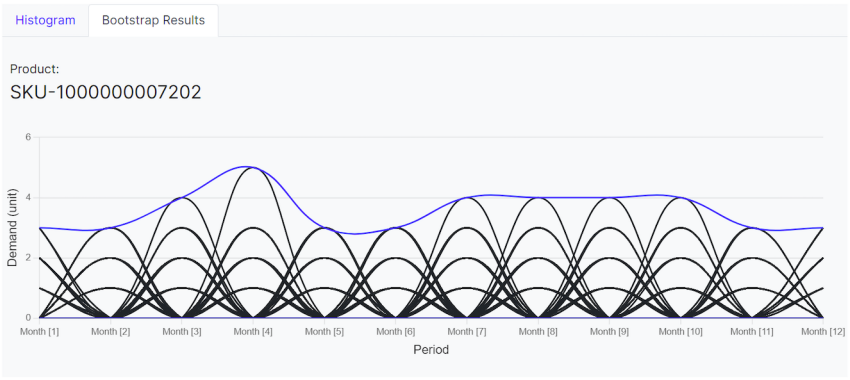






Sample Size: 1000
Sample Mean: 1.217
Standard Deviation: 1.6637

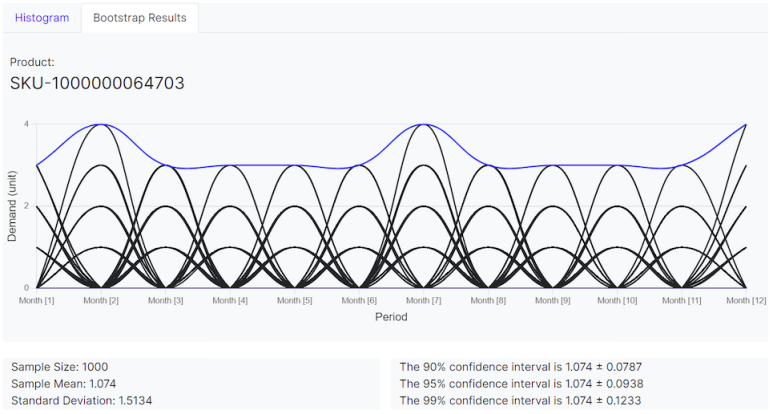
The 90% confidence interval is 1.217 ± 0.0865
The 95% confidence interval is 1.217 ± 0.1031
The 99% confidence interval is 1.217 ± 0.1355



Sample Size: 1000
Sample Mean: 1.217
Standard Deviation: 1.6637

The 90% confidence interval is 1.217 ± 0.0865
The 95% confidence interval is 1.217 ± 0.1031
The 99% confidence interval is 1.217 ± 0.1355


Histogram
Bootstrap Results
Product:
SKU-1000000064703
1000
500
0
Frequency
0.75
2.25
3.75
5.25
6.75
8.25
Annual demand
Sample Size: 1000
Sample Mean: 1.074
Standard Deviation: 1.5134
The 90% confidence interval is 1.074 ± 0.0787
The 95% confidence interval is 1.074 ± 0.0938
The 99% confidence interval is 1.074 ± 0.1233


Histogram
Bootstrap Results
Product:
SKU-1000000064703
4
2
0
Demand (unit)
Month [1]
Month [2]
Month [3]
Month [4]
Month [5]
Month [6]
Month [7]
Month [8]
Month [9]
Month [10]
Month [11]
Month [12]
Period
Sample Size: 1000
Sample Mean: 1.074
Standard Deviation: 1.5134
The 90% confidence interval is 1.074 ± 0.0787
The 95% confidence interval is 1.074 ± 0.0938
The 99% confidence interval is 1.074 ± 0.1233

Bibliography

[1] Alayed, E., O'Hegarty, R., Kinnane, O. (2022). Solutions for Exposed Structural Concrete Bridged Elements for a More Sustainable Concrete Construction in Hot Climates. Buildings, 12(2), 176. https://doi.org/10.3390/buildings12020176

[2] Armstrong, J. S. (2001). Judgmental Bootstrapping: Inferring Experts' Rules for Forecasting. International Series in Operations Research Management Science, 171–192. https://doi.org/10.1007/978-0-306-47630-3_9

[3] Barnston, A. G. (1992). Correspondence among the Correlation, RMSE, and Heidke Forecast Verification Measures; Refinement of the Heidke Score. Weather and Forecasting, 7(4), 699–709. https://doi.org/10.1175/1520-0434(1992)007

[4] Croston, J. D. (1972). Forecasting and Stock Control for Intermittent Demands. Operational Research Quarterly (1970–1977), 23(3), 289. https://doi.org/10.2307/3007885

[5] Martin, D., Spitzer, P., Kühl, N. (2020). A New Metric for Lumpy and Intermittent Demand Forecasts: Stock-keeping-oriented Prediction Error Costs. Proceedings of the Annual Hawaii International Conference on System Sciences. https://doi.org/10.24251/hicss.2020.121

[6] Porras, E., Dekker, R. (2008). An inventory control system for spare parts at a refinery: An empirical comparison of different re-order point methods. European Journal of Operational Research, 184(1), 101–132. https://doi.org/10.1016/j.ejor.2006.11.008

[7] Rego, J. R. D., Mesquita, M. A. D. (2015). Demand forecasting and inventory control: A simulation study on automotive spare parts. International Journal of Production Economics, 161, 1–16. https://doi.org/10.1016/j.ijpe.2014.11.009

[8] Shenstone, L., Hyndman, R. J. (2005). Stochastic models underlying Croston's method for intermittent demand forecasting. Journal of Forecasting, 24(6), 389–402. https://doi.org/10.1002/for.963

[9] Smith, M., Babai, M. Z. (2011). A Review of Bootstrapping for Spare Parts Forecasting. Service Parts Management, 125–141. https://doi.org/10.1007/978-0-85729-039-7_6

[10] Syntetos, A. A., Boylan, J. E., Croston, J. D. (2005). On the categorization of demand patterns. Journal of the Operational Research Society, 56(5), 495–503. https://doi.org/10.1057/palgrave.jors.2601841

[11] Syntetos, A. A., Zied Babai, M., Gardner, E. S. (2015). Forecasting intermittent inventory demands: simple parametric methods vs. bootstrapping. Journal of Business Research, 68(8), 1746–1752. https://doi.org/10.1016/j.jbusres.2015.03.034

[12] Syntetos, A., Boylan, J. (2001). On the bias of intermittent demand estimates. International Journal of Production Economics, 71(1–3), 457–466. https://doi.org/10.1016/s0925-5273(00)00143-2

[13] 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

[14] Williams, T. M. (1984). Stock Control with Sporadic and Slow-Moving Demand. Journal of the Operational Research Society, 35(10), 939–948. https://doi.org/10.1057/jors.1984.185