logistics

MDPI

Article

Exploring Applications and Practical Examples by Streamlining Material Requirements Planning (MRP) with Python

João Reis

Industrial Engineering and Management, Faculty of Engineering, Lusófona University, 1749-024 Lisbon, Portugal; joao.reis@ulusofona.pt

Abstract: *Background*: Material Requirements Planning (MRP) is critical in Supply Chain Management (SCM), facilitating effective inventory management and meeting production demands in the manufacturing sector. Despite the potential benefits of automating the MRP tasks to meet the demand for expedited and efficient management, the field appears to be lagging behind in harnessing the advancements offered by Artificial Intelligence (AI) and sophisticated programming languages. Consequently, this study aims to address this gap by exploring the applications of Python in simplifying the MRP processes. *Methods*: This article offers a twofold approach: firstly, it conducts research to uncover the potential applications of the Python code in streamlining the MRP operations, and the practical examples serve as evidence of Python's efficacy in simplifying the MRP tasks; secondly, this article introduces a conceptual framework that showcases the Python ecosystem, highlighting libraries and structures that enable efficient data manipulation, analysis, and optimization techniques. *Results*: This study presents a versatile framework that integrates a variety of Python tools, including but not limited to Pandas, Matplotlib, and Plotly, to streamline and actualize an 8-step MRP process. Additionally, it offers preliminary insights into the integration of the Python-based MRP solution (MRP.py) with Enterprise Resource Planning (ERP) systems. *Conclusions*: While the article focuses on demonstrating the practicality of Python in MRP, future endeavors will entail empirically integrating MRP.py with the ERP systems in small- and medium-sized companies. This integration will establish real-time data synchronization between the Python and ERP systems, leading to accurate MRP calculations and enhanced decision-making processes.

Keywords: data analysis; decision-making process; enterprise resource planning; inventory management; material requirements planning; Python; real-time data synchronization; supply chain management

Citation: Reis, J. Exploring Applications and Practical Examples by Streamlining Material Requirements Planning (MRP) with Python. *Logistics* **2023**, *7*, 91. https://doi.org/10.3390/logistics7040091

Academic Editor: Robert Handfield

Received: 17 September 2023
Revised: 1 November 2023
Accepted: 27 November 2023
Published: 1 December 2023

1. Introduction

Material Requirements Planning (MRP) plays a critical role in Supply Chain Management (SCM) by facilitating efficient inventory management and meeting production demands in the manufacturing sector. Amidst the significant expansion of the SCM in recent years, attributed to the swift technological advancements within the manufacturing sector, a noteworthy impact at the MRP would be anticipated. However, software employing object-oriented dynamic programming languages such as Python appears to be lagging in adequately addressing this exigency within the MRP. Henceforth, the landscape presents an evident scarcity in academia, where the focus is directed toward the MRP to harness the capabilities of Python. To reinforce the previous argument, a rapid inquiry was conducted using Elsevier's Scopus database as of 26 August 2023. The search was confined to Titles–Abstracts–Keywords, deploying the search terms "Material Requirements Planning" and "Python". Only two documents resulted from the search, effectively accentuating the noteworthy scarcity of comprehensive publications within this field. Notably, the work authored by Nandhakumar et al. [1] emerged as a relevant contender, centering its attention on harnessing Python's potential for precise cost estimation during the preliminary phases

of production. Through a comparative analysis between the traditional analytical costing approach and the software-driven cost estimation, a congruity is revealed. This empirical validation serves to accentuate the software's precision in handling accurate outcomes, consequently cementing its status as an instrument for preemptively evaluating costs before the actual commencement of production. This, in turn, empowers enterprises to chart their downstream production strategies. The subsequent investigation, authored by Odedairo and Ladokun [2], aimed to craft an independent lot-sizing module (LST-MOD) complete with a user-friendly graphical interface (GUI). The module's purpose was to establish the optimal ordering policy, a crucial facet of lot sizing, which forms one of the triad input components for the MRP. In practical scenarios, the application of lot sizing models (LSM) to address real-world situations is infrequent due to disparate assumptions, thresholds, and the varying extent of their applicability. Recognizing that lot sizing decisions are exogenous, the lot sizing facet within the MRP framework was disentangled. This endeavor culminated in the creation of a stand-alone module using the Python programming language. With a primary product and its sub-items as the focal point, the LST-MOD was employed to compute the comprehensive inventory costs. When experimenting with distinct lot-sizing models, those exhibiting heightened flexibility showcased superior performance over their less-flexible counterparts. This pioneering approach underscores the feasibility of constructing an in-house lot-sizing module, especially pertinent for organizations with limited financial prowess.

Considering the mentioned points, we have identified the following gaps: (1) The theoretical gap pertains to the misalignment between the advancements offered by Python and the need to automate the MRP tasks in the field of supply chain management. Despite Python being widely recognized as a powerful language with extensive capabilities, its applications in streamlining the MRP processes remain largely unexplored and underutilized. Thus, there is a theoretical gap in comprehending and harnessing the potential of the Python code in optimizing the MRP operations. This argument is justified by our mentioned preliminary search on Elsevier Scopus, while this search yielded only two relevant documents. Consequently, contrary to our anticipation of a substantial body of literature, our findings revealed an unexpectedly scant amount of relevant material. While there exists, solid theoretical knowledge regarding the MRP and Python resources individually, the practical knowledge integrating both domains, is considerably limited; and (2) the practical gap relates to the scarcity of practical examples and real-world implementation of the Python code in the MRP processes. In other words, despite the anticipated advantages that the automation of the MRP through Python is poised to bring in terms of swiftness, efficiency, and cost savings for the implementing enterprises [3], our research has revealed a conspicuous absence of reports on the subject from renowned consulting entities, such as the McKinsey Global Institute, Deloitte, among others. Furthermore, it has become increasingly apparent that organizations are progressively seeking bespoke software and AI solutions tailored to their specific requirements [4,5], while also emphasizing cost-effectiveness. This stands in contrast to the use of generic MRP software, which often proves to be a costly investment and may not adequately cater to the nuanced demands of these organizations.

Consequently, there is a need to develop scenarios that provide tangible evidence of the effectiveness of the Python code in simplifying the MRP tasks and enhancing decision-making processes. Bridging this practical gap necessitates presenting practical use cases that demonstrate the advantages of employing the Python code in the MRP calculations, inventory management, and other relevant aspects of supply chain management. In sum, the problem identified in this study is the underutilization of the capabilities of Python programming in streamlining the MRP tasks within the SCM domain. Despite the recognition of Python as a powerful language with extensive capabilities [6,7], its potential in optimizing the MRP processes remains largely unexplored. This study recognizes the lack of comprehensive literature and practical examples integrating Python and MRP, thereby highlighting the need to address this gap. By addressing these gaps, this article not

only highlights the viability of Python in the MRP but also lays the groundwork for future advancements in empirically integrating these systems with Enterprise Resource Planning (ERP), thereby enabling real-time supply chain management.

The primary objectives of this study are: (1) to explore the potential applications in streamlining the MRP operations and provide practical examples demonstrating the efficacy of Python in simplifying the MRP tasks; (2) to introduce a conceptual framework that showcases the integration of Python within the MRP ecosystem, emphasizing the use of specific libraries and structures for efficient data manipulation, analysis and optimization techniques; (3) to present a versatile framework that integrates various Python tools, such as Pandas, Matplotlib, and Plotly, to streamline an 8-step MRP process and provide preliminary insights into integrating the MRP.py with ERP systems; and (4) to lay the groundwork for future empirical integration of the MRP.py with ERP systems in small- and medium-sized companies, enabling real-time data synchronization and improved decision-making processes in supply chain management. Following the mentioned objectives, we drafted the following research question: How can the applications of Python be effectively utilized to automate and simplify the MRP processes in supply chain management?

To address the research question, this article initiates by establishing a foundational conceptual framework. Herein, we expound upon the fundamental principles around MRP, concurrently delineating the integration of Python's computational capabilities into the MRP operations. Subsequently, our focus goes toward the employed research methodology, presenting a succinct yet comprehensive delineation of the article configuration. Successive sections of the manuscript progressively explore a comprehensive discussion, systematically unraveling an 8-step framework that constitutes the basis of the Python-enabled MRP solution. The discussion is complemented by a well-defined conceptual framework and empirical validation through real-world commercial scenarios. In the final section, the conclusion includes the article's key theoretical and managerial contributions. Concurrently, it acknowledges the research's limitations and future endeavors.

2. Literature Review

This section comprehensively discusses the fundamental principles that form the basis of the MRP. Moreover, it undertakes an exploration of Python's computational capabilities about the diverse tasks and operations of the MRP.

2.1. Fundamentals of Material Requirement Planning

MRP is a methodology employed to plan the production of assembled products, ranging from computers to automobiles and a diverse array of other products that are assembled. In this dynamic process, certain items undergo repetitive production cycles, whereas others are crafted in well-defined batches. This systematic approach commences with the formulation of a master schedule (Figure 1), which outlines the required quantity and the timeline for each assembled product, often denoted as the end item. Subsequently, the MRP expands by generating a comprehensive production plan tailored to the end item. This plan delineates the optimal quantities and the synchronized timing of subassemblies, essential parts, and raw materials, essential to the assembly of the final product. According to Stevenson [8] and Lou et al. [9], the MRP includes three principal inputs. First, the BOM (Bill of Materials) is a guide detailing the composition of the final product [10]. This is combined with the "master schedule", a strategic document outlining the desired quantum of finished products and when. Third, the "inventory record file", tells the existing inventory on hand and pending orders [11]. The planner analyzes the provided information to determine the net requirements for every interval within the designated planning horizon. Process outputs include planned order schedules, order releases, changes, performance tracking reports, planning reports, and exception reports [8].

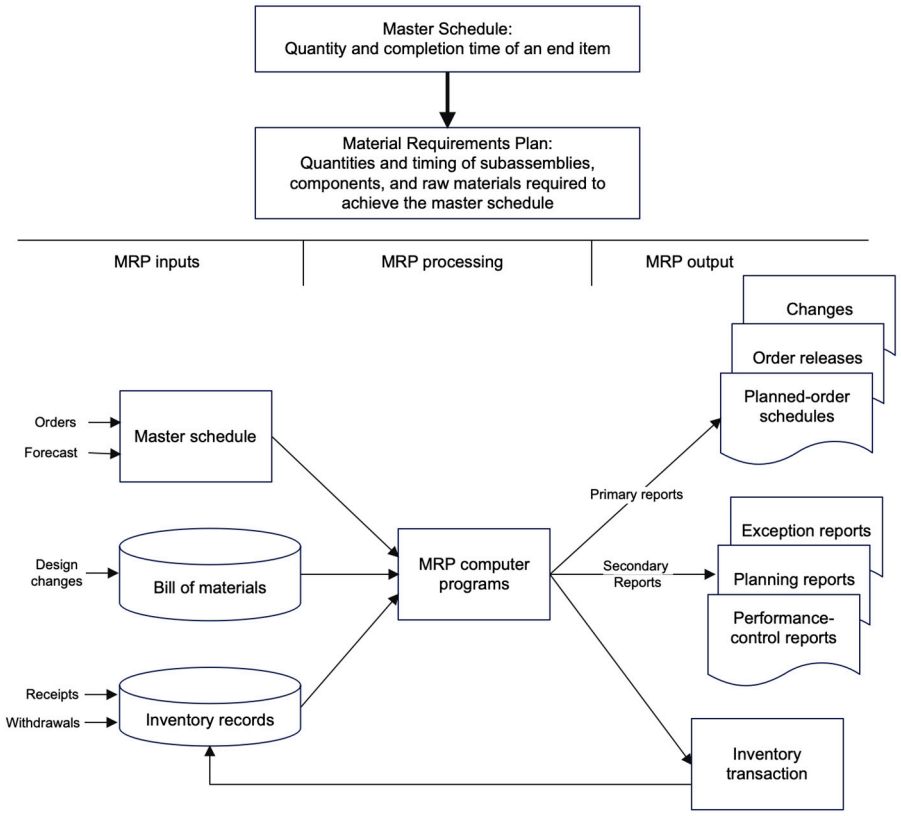

Figure 1. MRP inputs, processing, and output [8].

Examining Figure 1, we encounter the Master Schedule, otherwise referred to as the Master Production Schedule (MPS). This schedule plays the role of a compass, indicating the specific end items for production, their deadlines, and the corresponding quantities needed [12]. The quantities on the MPS come from various origins, including customer orders, predictive forecasts, and warehouse requisitions designed to build seasonal inventory reserves. The MPS effectively divides the planning horizon into discrete periods or time intervals, frequently delineated in weeks. However, these temporal divisions do not need to be uniform. While the immediate short-term section of the MPS may be structured in weekly units, subsequent periods might be organized in monthly or quarterly increments. Although the MPS does not adhere to a predefined time frame, managers generally chart its course to encompass a sufficiently distant future, allowing for a preliminary grasp of anticipated demands in the upcoming stretch. Equally vital is the inclusion of stacked or cumulative lead times essential for end-item production. Another essential input is the BOM, which encompasses a comprehensive roster of assemblies, sub-assemblies, individual parts, associated costs, and raw materials indispensable to produce a single unit of the final product. Consequently, each final product has its own dedicated BOM. This listing within the BOM adheres to a hierarchical structure, detailing the requisite quantities of each element necessary to complete a solitary unit of its parent item. The inherent nature of this facet within a BOM becomes clear upon exploring a product structure tree—a graphical diagram of the subassemblies and constituents needed for the assembly of a product. The top of the tree features the final product, followed by the subsequent strata subassemblies or significant components needed for crafting the final product. Beneath each major component are the corresponding minor elements. Progressing down the tiers of the tree reveals the components (parts, materials) necessary for manufacturing a single unit of the higher-tier item. A product structure tree serves as a tool for illustrating how the bill of materials is wielded to ascertain the quantities of each constituent (requirements) mandated to attain a desired volume of end items. Components situated at the lowermost levels of the tree typically encompass raw materials or procured parts, whereas those positioned

at higher tiers predominantly constitute assemblies or subassemblies. For suppliers, the product structure trees about items at lower tiers present pertinent considerations. On the bottom left in Figure 1, we find the inventory records. These encompass recorded data on the condition of each item over designated time intervals, referred to as time intervals. This repository encompasses vital aspects such as current stock levels and outstanding orders. It further encompasses additional particulars concerning each item, such as supplier details, lead times, and batch size policies. The file logs all movements due to stock receipts, withdrawals, canceled orders, and analogous occurrences. Like the precision demanded for the bill of materials, the accuracy of inventory records is paramount. Any inaccuracies concerning requirements or lead times hold the potential to cast adverse reverberations on the MRP process.

The MRP processing (vide Figure 1) involves taking the end item requirements stipulated by the master schedule into a phased arrangement of requisites for assemblies, parts, and raw materials. The MRP process execution included the synchronization of time phases with a sequence termed an "explosion", which is partitioned into distinct sections within a spreadsheet framework. Key spreadsheet terminologies are delineated as follows [8,13]: (1) Gross Requirements: This signifies the total expected demand for an item or raw material during each period, irrespective of the quantity already available; (2) Scheduled Receipts: These denote open orders, representing orders that have been placed and are anticipated to arrive mainly from suppliers; (3) Projected on hand: This metric reflects the anticipated inventory level at the start of each period; (4) Net Requirements: These represent the actual quantity required for each period, calculated by considering demand and subtracting the available inventory and scheduled receipts; (5) Planned Order Receipt: This denotes the expected quantity to be received by the beginning of the specific period it pertains to; (6) Planned Order Releases: These entries signify planned order quantities for each period. The indispensability of the computerization of the MRP comes to the fore when one ponders over the fact that a typical company deals with numerous end items, each necessitating its unique material requirements plan replete with individual components. The ongoing maintenance of on-hand and on-demand inventory levels, schedules, order releases, and other relevant factors necessitates constant updates due to changes and rescheduling. In the absence of computerized aid, this task would be overwhelmingly arduous; however, the prowess of programming languages empowers planners to undertake these responsibilities with significantly reduced complexity.

The MRP systems possess the capacity to provide management with a wide range of primary and secondary reports. Within primary reports, which encompass production and inventory planning and control, the following components are commonly included: Planned Orders, a schedule that indicates the quantity and timing of forthcoming orders. Order Releases, authorizing the execution of planned orders. Lastly in Figure 1, Changes to Planned Orders [14], which incorporate alterations to planned orders, ranging from adjustments to due dates and order quantities to outright cancellations. The secondary reports encompass performance tracking, scheduling, and exceptions, the following aspects hold significance. In this regard, we highlight the Performance Control Reports that assess the operation of system operations. Planning Reports that are anticipating future inventory requisites. Lastly, the Exception Reports highlight significant discrepancies such as late and overdue orders, warranting immediate attention. In summary, the MRP systems deliver a spectrum of critical and supplementary outcomes and reports that collectively equip management with the requisite tools for informed decision-making and effective operational oversight.

2.2. Integrating Python's Capability with MRP Tasks

In recent years, the field of Supply Chain Analysis (SCA) has undergone a remarkable transformation, driven by the rapid advances in Information and Communication Technologies (ICT) in the manufacturing industry. This evolution has led to a growing demand for skilled professionals who can explore the complexities of the SCA and management.

However, despite this increased interest, there remains a critical shortage of experts in this field [15,16].

Recognizing the urgency of this skills gap, several leading global institutions have taken steps to incorporate Supply Chain Analytics Courses into their academic curricula. These efforts, whether as mandatory or optional subjects, are valuable, as they represent important first steps towards addressing the pressing need for supply chain expertise [17].

However, it is important to acknowledge that these initiatives, while commendable, are not sufficient to bridge the existing skill gap. A study by Kumar et al. [17] highlights this point by noting that the use of practical tools such as R-studio and Python-based exercises in the SCA courses is still limited to a small number of prestigious higher education institutions, such as the Massachusetts Institute of Technology and Rutgers University.

The scarcity of such practical tools in supply chain education is a concerning trend, as it leaves a significant portion of aspiring professionals without exposure to the real-world tools and technologies they need to succeed in this field. The demand for skilled supply chain analysts is not limited to elite institutions; it spans a wide range of industries and regions, making it essential that education in this area becomes more accessible and widespread.

To truly prepare the next generation of professionals for success in the SCA, there is a need for a more comprehensive and inclusive approach. By integrating Python into the MRP tasks, educational institutions can empower students with a versatile and in-demand skillset that is aligned with the dynamic nature of modern supply chains. This approach not only equips students with practical skills but also fosters innovation and problem-solving abilities, which are essential in an ever-evolving industry.

Furthermore, Python's open-source nature ensures that access to these tools is not limited by financial constraints, democratizing the learning process, and providing opportunities for aspiring professionals from all backgrounds to enter the field.

In conclusion, the integration of Python's capabilities with the MRP tasks presents a promising solution to address the shortage of skilled professionals in the SCA. By expanding the reach of practical tools and technology-enabled learning to a broader spectrum of educational institutions and aspiring analysts, we can make a significant contribution to the growth and sustainability of the global supply chain ecosystem.

3. Materials and Methods

Originally, we planned to conduct a Systematic Literature Review (SLR) by using a PRISMA statement (Preferred Reporting Items for Systematic Reviews and Meta-Analyses). The SLR stands out as an optimal methodology, characterized by a systematic approach encompassing the detection, selection, categorization, and analysis of pertinent articles pertaining to a defined topic [18]. This method is executed with a rigorous, transparent, and replicable procedure, fostering a comprehensive and meticulous appraisal of the research subject, ultimately yielding an extensive, high-quality review [19]. Notwithstanding, as detailed in the introduction section, after conducting an initial inquiry on Elsevier Scopus using the specified search terms "Material Requirements Planning" and "Python", which yielded a meager two relevant documents, our attention pivoted towards the construction of a theoretical framework based on the existing MRP model [8]. Thus, unable to conduct an SLR, due to the scarcity of peer-reviewed articles, we chose to ensure transparency by adhering to an academic guide widely utilized by scholars globally. The academic manual [8] that served as the basis for the preexisting MRP model is widely recognized as a foundational text in the field of Operations Management (OM) and Production Planning and Control (PPC). The manual has been adopted and adapted by many scholars for educational and scientific purposes.

Thus, the purpose of this article is to address a gap in the current literature by exploring the various applications of Python in streamlining the MRP processes. We aimed to demonstrate the tangible benefits of Python implementation through practical illustrations. To achieve this, we adopted a twofold approach. First, we conducted conventional research

using a variety of sources, including scientific-academic manuals, blinded peer-reviewed articles, conference proceedings, and the Python official website [20]. The selection of manuals and peer-reviewed journal articles substantiates our decision, as they provide invaluable and credible insights into the understanding of a specific discipline [21]. This allowed us to uncover the many applications of Python programming in simplifying MRP operations. The practical examples we provide (Section 4.1.4) serve as empirical evidence of Python's effectiveness in enhancing the MRP tasks. Second, we introduce a conceptual framework [22] that provides an in-depth overview of the Python ecosystem. This framework highlights key libraries and frameworks that facilitate seamless data manipulation, advanced analysis, and optimization techniques. These components collectively contribute to the refinement of the MRP processes. Finally, we present an exploratory showcase of real-world business cases (Section 4.3) and additional exploratory case research (Section 4.4). While real-world cases only partially validate the proposed framework model, it underscores the potential advantages of using Python and paves the way for future empirical validation. It is important to note that the adoption of a high-level, general-purpose programming language like Python [23] to enhance the MRP systems and ERP is still relatively unexplored by companies. However, our work lays the groundwork for understanding the potential of Python in this domain.

4. Results

In this section, we aim to elaborate on how the Python programming optimizes the MRP operations. Presented below is an 8-step application guide, accompanied by a variety of applications that exemplify Python's effectiveness in simplifying the MRP tasks. Furthermore, we also provide a foundational code snippet for one of the 8 steps in the Python-based MRP solution (MRP.py). Through this article, our goal is to encourage fellow scholars to pursue similar endeavors. Such endeavors collectively contribute to strengthening or improving the conceptual framework presented later in this section.

4.1. Uncover the Potential Applications of Python in Streamlining MRP Operations

The versatile potential of Python in building tailored MRP systems or enhancing pre-existing ones involves coupling Python's expansive libraries and its integration with diverse technologies. These factors collectively underscore Python's aptitude for sculpting the MRP solutions attuned to individual business exigencies. The utilization of Python to streamline the MRP operations can be divided into an 8-step framework:

4.1.1. Data Aggregation

Python efficiently aggregates data from various sources, such as sales orders, stock levels, production schedules, and supplier delivery times. For instance, it can be used to integrate sales order data, inventory levels, production schedules, and supplier lead times from various sources into a single, consistent dataset. Preprocessing is often a prerequisite before conducting the MRP calculations. This preprocessing involves tasks such as handling missing values, accomplished through either estimation or statistical methods; eliminating duplicates by identifying and removing redundant records; transforming data types, which entails converting data from one type to another, like converting string values to numeric; and, addressing outliers, encompassing the identification and elimination of data points that deviate significantly from the norm. Hence, as a fundamental principle, the MRP demands the integration of data from diverse sources to ensure precise calculations. The adaptability offered by the Python libraries, exemplified by Pandas, proves instrumental in refining and preprocessing data, preparing it for comprehensive analysis. The code first imports the Pandas library, a Python package providing fast, flexible, and expressive data structures designed to work with relational or labeled data [24,25]. Subsequently, a Pandas DataFrame is established to store the data [26]. The code proceeds with utilizing the dropna () method to remove missing values from the DataFrame [27], followed by drop_duplicates () to remove duplicate records [28] and astype () to facilitate data type conversion [29]

within the DataFrame. Finally, the script prints the DataFrame to exhibit the outcomes of preprocessing. Having aggregated and cleansed the data, Python stands primed for the ensuing MRP calculations, as elaborated in the following steps.

4.1.2. Demand Prediction

Employing data analytics and Python's Machine Learning (ML) libraries facilitates demand prognostication [30]. This dynamic procedure converges historical sales data, seasonal patterns, and other relevant variables to yield forecasted demand–a fundamental input for the MRP calculations [31]. However, demand forecasting presents some complexity, as some machine learning algorithms are diverse and more complex than others, requiring more data to be trained. Additionally, the availability of data can be a limiting factor, as some ML algorithms require a large amount of data for training. For instance, the Generative Pre-trained Transformer 3 (GPT-3) by OpenAI is trained on hundreds of billions of words [32]. The number of computational resources required can also vary, with some algorithms placing greater demands on computing power. Once an ML library has been chosen, the next step is to collect the data that will be used to train the model. This dataset should include historical sales data, seasonal patterns, and any other relevant variables that could affect demand. This data then becomes the foundation for training the ML model. The training phase can take a significant amount of time, depending on the complexity of the model and the volume of data. Once the model is trained, it can be used to make predictions about future demand [33]. These forecasts can then be used to make informed decisions about inventory management, production scheduling, and marketing strategies. In the context of the MRP, demand forecasting is essential for determining the optimal quantities of raw materials and finished goods to produce to meet future demand. This helps to prevent supply-demand imbalances and ensures that the company has sufficient inventory to meet demand without overstocking. This section aligns with Elbegzaya's thesis [34], emphasizing the increasing significance of seamlessly integrating and orchestrating the complete spectrum of end-to-end supply chain processes in the face of market complexities and uncertain demand. Elbegzaya [34] highlights the growing importance of data sourcing, management, and manipulation as a key advantage for businesses. Notably, leading organizations have extensively explored the boundaries of ML and AI to enhance operational excellence. The widespread use of AI is particularly evident in comprehensive computational modeling for tasks such as reasoning, pattern recognition, extensive calculations, experiential learning, and adept comprehension to cater to diverse requirements. Despite its prominence in demand planning and forecasting, the comprehensive integration of AI and/or machine learning within other supply chain sub-applications, including the MRP and MPS, remains an area of ongoing focus within this discourse.

4.1.3. Inventory Optimization

Inventory optimization is the process of finding the right balance between having enough inventory to meet customer demand and not having too much inventory that costs money to store. This is a complex task that considers factors such as product demand, lead time, holding cost, and purchase cost. The goal of inventory optimization is to determine the optimal reorder points and order quantities for different products.

Python scripts can be used to automate the calculation of optimal reorder points and order quantities. The programming language is well-suited for this task, as it can handle the complex calculations and considerations involved in inventory optimization efficiently and accurately. Zara is a well-known case study in inventory optimization. The company is known for its quick response, high-fashion products, competitive prices, and many stores. To strike a good balance between keeping inventory on display to drive sales and eliminating the impact of excessive shipping orders, Zara focuses on improving its global distribution process. In mid-2005, Zara created a new process that has shipped quantities as the main decision variables and the maximization of global sales as its objective [35].

The new process consisted of two models: the prediction model and the optimization model [36]. To this end, Zara used commercial Mixed Integer Programming (MIP) software. However, there is no record of using a high-level, general-purpose programming language such as Python [23].

4.1.4. BOM Oversight

As mentioned earlier, a BOM is an exhaustive list of all materials, components, subassemblies, and their corresponding quantities required to manufacture a given product (Figure 2). In manufacturing, particularly in sectors such as the automotive industry, production efficiency depends on BOM oversight. This implies vigilant monitoring of BOM components and their availability throughout the production continuum, ensuring compliance with production deadlines and preventing potential problems related to shortages. In this regard, Python can be leveraged as a powerful tool for BOM oversight. Python's versatile capabilities can be harnessed to guarantee the accessibility of essential components and harmonize their synchronization with production schedules. For example, Python can be used to create scripts (the Python script below as an example) or software tools that automatically synchronize the BOM with production schedules. This ensures that the materials and components needed for each production phase are available on time, mitigating delays and bottlenecks. Additionally, Python can be integrated with various data sources and sensors to enable real-time monitoring of the status of materials and components. This provides manufacturers with timely updates on inventory levels, usage trends, and potential shortages. By capitalizing on Python's capabilities, manufacturers can obtain a comprehensive and up-to-date understanding of their BOMs, which can lead to improved operational efficiency, productivity, and profitability. To provide a clear industrial example, we used a simple instance from Stevenson [8]. This simple example illustrates the bill of materials (BOM) for product X, which requires the calculation of the quantities of components B, C, D, E, and F needed to assemble a single unit of X.

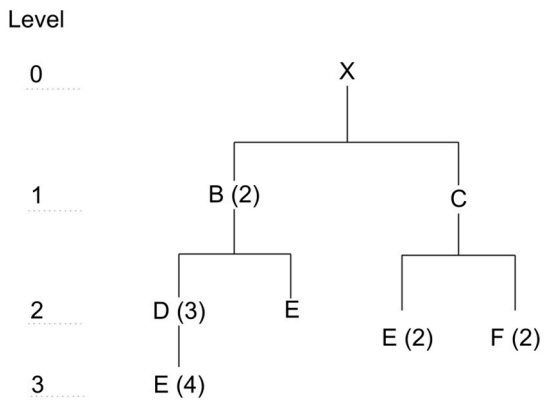

Figure 2. Bill of Materials (BOM) of product X (adapted from Stevenson [8]).

According to Figure 3, product X will require 2 units of B, 1 unit of C, 6 units of D, 28 units of E (note that E occurs in three places, with requirements of 24 + 2 + 2 = 28), and 2 units of F. Stevenson's work [8] underscores that the process of calculating total requirements is more complex than what Figure 2 might initially convey. This complexity arises from various factors. Firstly, many products boast a significantly greater number of components than illustrated. Additionally, the factor of timing proves crucial—determining when components need to be ordered or produced—and this temporal aspect must factor into the analysis. Lastly, several components or subassemblies could already be in stock, creating a need to account for these existing quantities in the calculations. The accuracy of the BOM is of paramount importance as it dictates the composition of a product. This significance stems from the fact that errors introduced at one level can be magnified through the multiplication process used to determine quantity requirements.

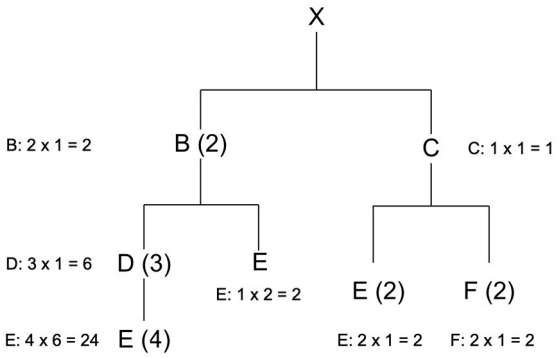

Figure 3. Materials needed to assemble X (adapted from Stevenson [8]).

While most companies have specialized software applications, the manufacturing sector is increasingly realizing the need for skilled industrial engineers. These experts can design custom software solutions that meet the unique needs of their industries. This can help to prevent errors that can accumulate over time. In this context, Python software can give a company a competitive advantage. As demonstrated by the example from Stevenson's [3] scholarly guide, we have developed a concise set of code lines that can automatically execute the task. While this program is designed for a straightforward educational and scientific purpose and can be used by a wide range of readers, including beginners and experts, the code can be further enhanced to be more complex, like the examples available on platforms like pyBOM [37].

The Python code in Figure 4 calculates the quantities of components needed to assemble a single unit of the top-level item 'X', based on the BOM shown in Figure 2. The code has the following key aspects: The BOM is represented as a dictionary, where each key is a component, and the corresponding value is a dictionary of the required sub-components to assemble that component along with their quantities. For example, to assemble 'X', we need 2 units of 'B' and 1 unit of 'C'. The function 'calculate_quantities' takes three parameters–the BOM, the current item being processed, and the quantity of the current item. It returns a dictionary of the quantities of all components needed to assemble the given item. If the current 'item' is not found in the BOM (i.e., it is a basic component), the function returns a dictionary with just the current component and its quantity. If the current 'item' is in the 'BOM', the function first creates a dictionary to store the component's quantities. Then, it loops through each sub-component of the current 'item' in the BOM. For each sub-component, its function recursively calls 'calculate_quantities' with the sub-component as the new 'item', the quantity required for the sub-component (calculated by multiplying the 'qty_needed' from the BOM by the current 'quantity'), and the 'available_quantity' dictionary. The quantities are then updated. The calculated 'quantities' from the sub-components are used to update the 'quantities' dictionary. If a component already exists in the 'quantities' dictionary, the calculated quantity is added to the existing quantity. Otherwise, a new entry is created with the calculated quantity. The code starts by calling 'calculate_quantities' for the top-level item 'X' with a quantity of 1. These initiatives are a recursive calculation of the quantities needed to assemble 'X', resulting in a dictionary containing these quantities. Finally, the final quantities for each component are printed using a simple loop that iterates through the dictionary returned by 'calculate_quantities'. By recursively traversing the BOM and calculating quantities for each component, the code accurately determines the quantities of 'B', 'C', 'D', 'E', and 'F' needed to assemble one single unit of 'X'. In other words, the code prints the final quantities for each component, and by employing this recursive approach to step through the BOM and calculate quantities for each component, the code accurately determines the quantities of 'B', 'C', 'D', 'E' and 'F' that is needed to construct a single unit of 'X'. The recursive approach handles the multi-level structure of the BOM, ensuring that all necessary quantities are appropriately aggregated and computed. This enables a comprehensive understanding of

the quantities required to produce the top-level item, considering all its sub-components and their respective quantities.

```
# Start # Python lines of code #

def calculate_quantities(bom, item, quantity=1):
    if item not in bom:
        return {item: quantity}

    quantities = {item: quantity}
    for component, qty_needed in bom[item].items():
        component_quantities = calculate_quantities(bom, component, qty_needed * quantity)
        for comp, qty in component_quantities.items():
            if comp in quantities:
                quantities[comp] += qty
            else:
                quantities[comp] = qty

    return quantities

bom = {
    'X': {'B': 2, 'C': 1},
    'B': {'D': 3, 'E': 1},
    'D': {'E': 4},
    'C': {'E': 2, 'F': 2},
}

final_quantities = calculate_quantities(bom, 'X')

for component, quantity in final_quantities.items():
    print(f"{component}: {quantity}")

# End # Python lines of code #
```

Figure 4. Foundational code snippet for one of the 8 steps in the Python-based MRP solution (MRP.py).

4.1.5. MRP Processing

The MRP processing is a part of the PPC that ensures timely and efficient order fulfillment while minimizing resource wastage and delays [9]. Python can be used to automate and optimize orders to create material timetables by accounting for lead times in procurement, calculating reorder points, and triggering timely alerts. It can also evaluate resource availability by considering current inventory and incoming orders, preventing production disruptions from material shortages. Python can optimize order quantities and reorder points to reduce costs while ensuring material availability. It can also generate reordering cues based on demand fluctuations in the MRP. Finally, it manages materials efficiently by curbing excess inventory and stockouts. Overall, Python can be a valuable tool for improving the efficiency and effectiveness of order scheduling.

4.1.6. Vendor Collaboration

Collaboration with suppliers is essential for a smooth flow of materials and information in the supply chain and it was one of the main issues identified in our readings of the literature [38]. Python can play a valuable role in this collaboration by automating the creation and submission of purchase orders (POs). POs are formal documents that a buyer issues to a supplier, specifying the items to be purchased, quantities, prices, and delivery conditions. Python can be used to automate the creation of POs by integrating with a company's MRP system. Once the POs have been created, Python can be used to submit them to suppliers in a variety of ways. For example, POs can be sent via email, where Python can automate the email generation process and attach the relevant documents. Alternatively, Python can be used to facilitate the use of EDI (Electronic Data Interchange), a standardized format for exchanging business documents electronically [39]. EDI eliminates the need for manual data entry and reduces errors, which can further streamline the collaboration process [40].

In summary, Python's capabilities in automating processes, performing computations, and facilitating communication make it a valuable tool for improving collaboration with suppliers. By automating the creation and submission of POs, Python can help to improve procurement efficiency, accuracy, and collaboration.

4.1.7. Visual Insights and Reporting

One of the most important outputs of the MRP is the data it generates (vide Figure 1). In today's data-driven world, the ability to effectively convey insights from complex datasets is crucial [41]. Python offers powerful visualization libraries such as Matplotlib [42] and Plotly [43] that can help us do just that. Matplotlib and Plotly are industry-standard libraries that are known for their versatility in generating a wide variety of visualizations [42,43]. Matplotlib is a good choice for creating static graphs, while Plotly offers interactivity and dynamic features that allow users to explore data interactively. Visualizations can help to reveal hidden patterns and trends in data that would otherwise be difficult to see, it also improves the communication of complex insights to stakeholders and enables data-driven decisions.

In conclusion, using Matplotlib and Plotly to visualize the MRP data is an essential skill for modern decision-makers. The insights gained from these visualizations can help organizations make informed decisions, optimize processes, and stay ahead of the competition.

4.1.8. ERP Fusion

The last phase we identified was the merger of MRP with the Enterprise Resource Planning (ERP) systems. ERP application is an enterprise-wide package that plays a key role in managing various business processes, including purchasing [44], inventory management [45], and production scheduling [46], just to name a few. However, even the most robust ERP systems may not provide all the advanced visualization and analytical tools needed for in-depth analysis of the MRP data. This is where Python´s fusion with the ERP systems comes into play. By integrating Python with the ERP systems, a wide range of tools can be leveraged to perform complex data transformations, predictive modeling, and advanced analytics on the MRP-related data. While ERP systems offer standardized reporting and analytics, they can fall short when it comes to specific, customized analytics that meet unique business needs [47]. In that regard, Python can help to develop custom analysis workflows that align with each organization´s specific MRP requirements. Overall, the integration of Python with the ERP systems can provide organizations with several benefits that can help them improve their production processes and achieve their business goals.

4.2. Conceptual Framework

In Figure 5, we provide an overview of the key concepts discussed in Section 4. The below figure visually incorporates the transformation of the conceptual model depicted in Figure 1 into a practical Python implementation, seamlessly integrated with the ERP systems. As previously highlighted, Python empowers the integration of innovative techniques for information retrieval and data aggregation. This, in turn, leads to highly efficient data manipulation, analysis, and optimization, all of which significantly enhance the MRP (Materials Requirement Planning) processes. Additionally, the Python's capabilities facilitate the generation of both primary and secondary reports, offering insightful visualizations that empower stakeholders to make well-informed decisions.

Significant enhancements in Figure 1 primarily go around data aggregation facilitated by Python. While there is a lack of literature that specifically outlines a conceptual framework for the MRP utilizing Python, recent publications by reputable scholars [48] have emphasized the integration of programming languages within the PPC. Notably, Esteso et al. [48] underscores the shift from conventional programming languages like C++, MATLAB, JAVA, Visual Basic.NET, and Delphi, alongside simulation tools such as ARENA, to contemporary ML platforms like TensorFlow and PyTorch, which are referenced in this article. The accessibility of Python's APIs within these platforms has subsequently

positioned Python as the dominant language for agent development. This development landscape serves as the impetus for our present study. In consideration of the framework depicted in Figure 1, Python effectively integrates data from various sources, encompassing, yet not limited to, sales orders and inventory levels. We emphasized the role of the Pandas library in this context, offering swift, adaptable, and expressive data structures tailored for handling relational or labeled data. This capability has facilitated efficient data analysis and seamless handling of complex data sets, fostering a more streamlined and comprehensive approach to data management. Furthermore, we have preserved the integrity of the Bill of Materials (BOM) and inventory optimization components, albeit with requisite technological refinements. By incorporating the latest technological advancements, we have managed to streamline and refine the process of handling the BOMs and inventory, enhancing the overall efficiency and productivity of the system–this includes the foundational code snippet for one of the 8-steps in the Python-based MRP solution. Regarding reporting, the principal alterations focus on leveraging visual insights tools like Matplotlib and Plotly. The integration of these visualization tools has not only enhanced the interpretability of complex data sets but has also provided a user-friendly interface, enabling stakeholders to comprehend and interpret data effortlessly, thereby enhancing the decision-making process. Overall, these tools empower stakeholders with data-driven insights, facilitating informed decision-making. Lastly, a crucial highlight centers on the symbiotic relationship between the MRP and ERP fostered through the Python's integration. This fusion of Python is arguably one of the most relevant facets of our system, bringing cohesion and synergy to the overall framework.

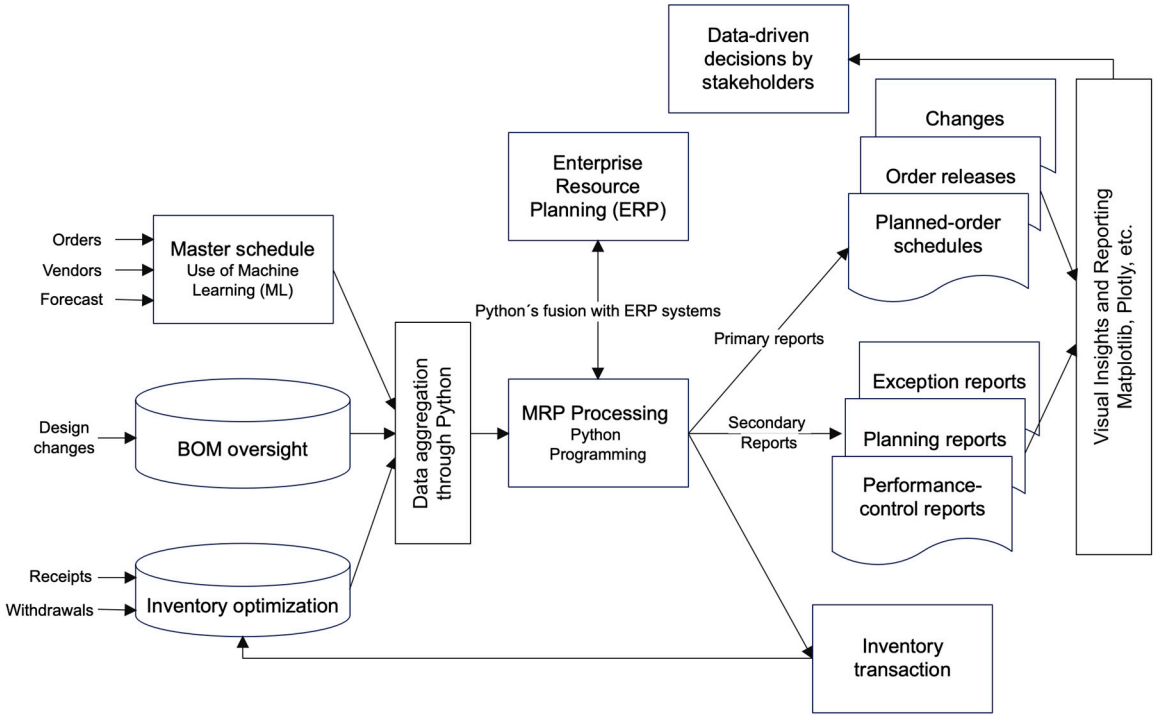

Figure 5. Conceptual framework on Material Requirements Planning (MRP) with Python.

4.3. Laying down the Foundations for Integrating MRP.py into Existing ERP Systems

This section seeks to achieve a balance between generalization and practical application, providing a foundational framework for the seamless integration of the MRP.py into the established ERP systems. While the focus of this article remains on a generalizable model applicable to researchers at varying levels of expertise in MRP, ERP, and Python, we have also sought to present some further details about how to integrate the MRP.py into the ERP systems.

Presently, our research efforts are concentrated within two distinct teams, one operating within the Portuguese electronic sector and the other within the automotive industry. Our commitment lies in the empirical validation of the fundamental groundwork. While the integration of the MRP.py into dedicated ERP systems is still in the exploratory phase, this section presents preliminary empirical findings derived from our ongoing research endeavor. Specifically, through collaboration with industrial and computer engineers from the mentioned sectors, we formulated comprehensive integration guidelines for the MRP.py within the ERP systems. The guidelines will be empirically explored in the upcoming months, yielding fresh insights, and reinforcing the ongoing research by supporting the development of specific models within the mentioned organizations.

During the current phase, the integration of the MRP.py code into the ERP systems required collaborative brainstorming among key individuals in the companies, led by engineers with extensive experience in both the MRP and ERP systems. As such, the generic guidelines presented herein are a collective effort between the two teams, from the electronic and automotive industries. To ensure clarity and systematic implementation, we have categorized the guidelines into four primary phases (p-phases) and an additional four secondary phases (s-phases), for effectively integrating the MRP.py into the ERP systems.

Initiating the integration process, we initially underscored the critical requirement for system analysis and compatibility verification (p-phase I), requiring an in-depth understanding of the existing ERP system's architecture and its compatibility with the developed MRP.py. Subsequently, we emphasized the integration of API and data exchange protocols (p-phase II), exploring the use of application programming interfaces (APIs) and data exchange protocols to streamline communication and data transfer between the MRP.py and the ERP system. For smaller enterprises, the integration of the MRP.py through the Microsoft Excel document outputs from the ERP system might offer a simpler data processing solution. Nevertheless, we anticipate the development of more robust and secure data transfer protocols, especially for larger corporations. In such cases, implementing a data mapping strategy becomes essential, ensuring seamless data transformation processes that accurately interpret and process data from both systems. This becomes particularly crucial as the MRP systems often incorporate external information such as supplier deliveries, requiring an efficient data conversion process.

Furthermore, it is imperative to establish event-driven integration mechanisms (p-phase III) that facilitate real-time updates and notifications between the MRP.py code and the ERP system, ensuring continuous system synchronization. To prevent unwarranted discrepancies, configuring event triggers and automated workflows is necessary, guaranteeing that any changes or updates within one system are immediately reflected in the other, thereby maintaining data consistency and accuracy. Finally, an effective error-handling strategy (p-phase IV) should be devised to address data inconsistencies, processing errors, and system failures during the integration process. Implementing a robust logging and monitoring mechanism is crucial, enabling the capture and analysis of errors, facilitating effective troubleshooting, and preventing future discrepancies.

In conjunction with the primary measures, we delineate four pivotal secondary phases. First and foremost, ensuring robust security protocols and access control measures (s-phase I) is imperative to safeguard the integrity and confidentiality of all data, especially sensitive information pertaining to employees and stakeholders. This involves assigning specific user roles and permissions, ensuring that employees can only access the data relevant to their roles within the organization. Given the escalating prominence of cybersecurity threats, the implementation of encryption techniques and secure authentication mechanisms (s-phase II) is paramount to fortify the system against potential security breaches. The secure authentication mechanisms phase can include methods such as multi-factor authentication, biometric authentication, or other secure login protocols that verify the identity of users before granting access. Regular testing and validation of procedures are also necessary to uphold the reliability of the integrated MRP.py and ERP system. In this regard, scenario-based testing plays a pivotal role in identifying and resolving potential

issues or discrepancies, ensuring the continuous accuracy and functionality of the systems over time. This involves carrying out security audits, penetration tests, and vulnerability assessments to ensure that the security measures in place are effective and up to date.

Additionally, safeguarding the knowledge acquired through the integration process is critical (s-phase III). Thorough documentation of all developments and the creation of comprehensive training materials are essential to facilitate users' and stakeholders' understanding of the integrated MRP.py and ERP system. Lastly, the implementation of continuous monitoring and maintenance practices (s-phase IV) is vital for overseeing the performance, scalability, and sustainability of the integrated system. Employing monitoring tools such as dashboards and routine maintenance procedures, including timely software updates, is instrumental in ensuring the seamless integration of the entire process.

By adhering to these guidelines, organizations can seamlessly integrate the MRP.py code into their existing ERP systems, thereby optimizing their manufacturing processes. As underscored earlier, the current guidelines are of a preliminary nature, thereby necessitating a more comprehensive elucidation. Specifically, a thorough investigation is imperative to discern the most appropriate encryption techniques and secure authentication mechanisms tailored to various industries. For instance, an in-depth analysis is required to determine the compatibility of multi-factor authentication within the automotive sector, or whether a dynamic and expedient protocol, capable of authenticating users prior to granting access, would be more efficacious. This inquiry represents merely a fraction of the manifold queries that underscore the persistent relevance of this subject, underscoring its exigency for further scholarly inquiry.

4.4. Preliminary Empirical Validation–Practical Examples from Real-Life (Business Cases)

While the utilization of Python in the MRP may not yet be widely recognized, there are compelling instances that underscore its value within this framework. We would like to elucidate two cases. The first instance revolves around the Python software known as "openerp-mrp" [49], specifically focusing on the manufacturing module. This module seamlessly handles critical aspects of the manufacturing process, including planning, ordering, inventory management, and the production or assembly of products using raw materials and components. Some features of this solution include versatile capabilities such as Make to Stock/Make to Order and support for multi-level Bills of material, without imposing any limits. The second case centers on the evolution of open-source ERP systems. Through an in-depth comparative analysis between OpenERP (leveraging open-source technologies) and other prominent ERP systems, it becomes clear that OpenERP surpasses expectations when pitted against its competitors. The introduction of Odoo/OpenERP marked a pivotal moment in ERP systems [50], as it introduced elements such as a Content Management System (CMS), an e-commerce system, and integrated Business Intelligence (BI) capabilities into the system. However, this expanded functionality necessitates a thorough examination and adaptation to address the increased system complexity. A real-world illustration of Python's efficacy in this context can be seen in the case of Made.com [51], a substantial online furniture retailer boasting a vast array of products. Collaborating directly with designers and manufacturers, they have steadily expanded their network of partners (vendors). Faced with the need for ambitious warehouse integrations, Made.com embarked on a journey to reconstruct its ERP system incrementally, adopting a microservices architecture developed in Python and interconnected through Event Sourcing. The choice of Python in this endeavor not only mitigated risks but also facilitated rapid software development—a testament to Python's agility and effectiveness. In summary, Python's relevance in the MRP-ERP is underscored by these instances, which highlight its versatility, adaptability, and efficiency in addressing the evolving demands of modern manufacturing processes.

4.5. Summary

Although Python is a versatile tool, implementing an MRP system remains complex due to several salient factors. Primarily, building custom MRP systems or amplifying extant

ones necessitates multidisciplinary competencies, encompassing not only programming skills but also an in-depth comprehension of supply chains and business processes. Consequently, organizations should recruit professionals such as industrial engineers, individuals equipped not only with programming proficiency but also profound insights into supply chains and logistics. Furthermore, the construction of personalized MRP systems through Python mandates the fragmentation of the mentioned 8-step framework into multiple projects, given the size of the task and its complexity. Lastly, the narrative of real-time data synchronization, scalability, security fortification, and continuous upkeep should also find resonance during the Python-driven MRP system's journey.

5. Conclusions

This article's findings indicate that leveraging Python in the MRP simplifies the planning process, reduces human error, increases efficiency, and enables organizations to make data-driven decisions. Automating the MRP tasks allows supply chain professionals to focus on strategic activities and proactively address potential disruptions.

5.1. Theoretical Contributions

Our article introduces an innovative conceptual framework (Figure 5) to investigate the MRP method using Python. While the foundational procedures of the MRP are not novel, the application of a high-level, general-purpose programming language such as Python represents a distinct and inventive approach [23]. The framework integrates a variety of Python tools, including Pandas, Matplotlib, and Plotly, to facilitate the operationalization of an 8-step MRP process. The framework differs from prevailing standardized corporate solutions in two ways: it uses a comprehensive set of tools that can be adapted to the dynamic complexity of different organizational environments; and, in addition, it supports a multidimensional analysis of complexity, suggesting the development of MRP.py into discrete projects. This allows the use of varied tools and a more nuanced understanding of the complex dynamics of the MRP systems (e.g., exhaustive BOMs). In such cases, a stratagem involving modular designs, while seemingly ostensibly simplistic, may be considered an appropriate course of action. By partitioning the implementation into discrete projects, the assimilation of varied techniques becomes not only sustainable but also instrumental. This approach generates a more nuanced understanding of the complex dynamics inherent in these systems.

In short, the framework elaborated herein, although apparently simple, serves as a complex combination of techniques–a characteristic shared with other scholarly authors [8]. Through this multifaceted stance, the framework enriches our discernment of the relationship between the MRP and ERP, particularly in industries that manage a wide range of materials.

5.2. Managerial Contributions

This article makes several contributions to the field of management. First, it recognizes that each company has unique MRP practical requirements. For example, some companies may need to focus on improving their supplier relationships, while others may need to focus on managing the complex BOM. As a result, different parts of the conceptual framework (Figure 5) will need to be developed to different degrees. This article encourages industrial engineers to empirically validate the conceptual framework presented here within the context of their own organizations. Once the conceptual framework has been validated and/or adapted, engineers can develop separate Python programs for each component of the framework, treating each component as a distinct and independent project. The goal is to integrate these projects into a unified whole, which we call the MRP.py, and deploy it within the ERP system.

Second, this article also highlights the importance of the Python developer community. The community is large and active, and it is constantly releasing new libraries and tools. This makes it possible for industrial engineers to flexibly adapt and refine their

reporting and analytical solutions in response to changing business needs. By combining the MRP, ERP, and Python, companies can transform ordinary processes into strategic assets. Python's analytical capabilities can release new insights, agility, and optimization opportunities that go beyond the capabilities of traditional ERP systems. This integration can help companies extract more value from their ERP investments and remain adaptable in a dynamic business environment. In that regard, this article provides practical examples, including the case of OpenERP and the evolution of open-source ERP systems like Odoo/OpenERP, which serve as real-life testaments to the efficacy and adaptability of Python in addressing the complexities of modern manufacturing and SCM. Odoo/OpenERP marked a pivotal moment in ERP systems, as it introduced elements such as CMS, an e-commerce system, and integrated BI capabilities into the system. Moreover, the operational framework of Made.com serves as an additional tangible exemplification of Python's efficacy. Made.com strategically undertook a phased modernization of its ERP system, implementing a microservices architecture founded on the Python programming language. The application of Python not only reduced potential risks but also sped up software development, thus reaffirming the language's exceptional adaptability and proficiency. Projections indicate the imminent emergence of noteworthy cases showcasing the integration of Python, thereby justifying its relevance for in-depth academic analysis.

5.3. Research Limitations

This article outlines several research limitations that warrant careful consideration. We aim to highlight three key issues that, from our perspective, hold significant importance.

First, the conceptual framework is limited in its relevance over time. It is based on existing literature, but the field of AI and computer science is constantly evolving. As a result, the framework may become outdated as new technologies and techniques emerge. Thus, the framework may become outdated as new technologies and techniques emerge. For example, the reference to the Python libraries in this article may need to be updated to reflect the latest developments. Second, the framework is also designed to be generic, so it may not apply to all specific business domains. For example, a company in the automotive industry may have different needs than a company in the electronic industry. Third, the identified 8-step framework needs to be disaggregated into discrete projects. This is because managing a complex technological initiative is challenging and requires careful planning and execution. Breaking the initiative down into smaller, more manageable projects can help mitigate risks and ensure that the initiative is successful.

By consciously acknowledging and proactively mitigating these research limitations, we can formulate a resilient and adaptable approach to the dynamic technological landscape.

5.4. Future Endeavours

While the focus of this article is to showcase the practical applications of Python in the MRP, upcoming endeavors will revolve around the empirical incorporation of Python-based MRP solutions (referred to as MRP.py) into Enterprise Resource Planning (ERP) systems utilized by small and medium-sized enterprises. This integration aims to establish seamless real-time data synchronization between the Python and ERP systems, culminating in precise MRP computations and elevated decision-making processes. By bridging the divide between Python programming and MRP automation, this article not only underscores the effectiveness of Python code in the MRP but also paves the way for future innovations in harmonizing the MRP.py with the ERP systems. This progress lays the foundation for a more efficient and accurate management of the supply chain.

Funding: This research received no external funding.

Data Availability Statement: All data is contained in the article.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Nandhakumar, S.; Thirumalai, R.; Viswaaswaran, J.; Senthil, T.; Vishnuvardhan, V. Investigation of Production Costs in Manufacturing Environment Using Innovative Tools. *Mater. Today Proc.* **2021**, *37*, 1235–1238. [CrossRef]
2. Babatunde, O.; Demola, L. (Eds.) *Varying Lot-Sizing Models for Optimum Quantity-Determination in Material Requirement Planning System*; IAENG: London, UK, 2018.
3. Rozario, D. Can Machine Learning Optimize the Efficiency of the Operating Room in the Era of COVID-19? *Can. J. Surg.* **2020**, *63*, E527–E529. [CrossRef] [PubMed]
4. Wiegers, K.E.; Beatty, J. *Software Requirements*; Pearson Education: Redmond, WA, USA, 2013; ISBN 978-0-7356-7962-7.
5. Vial, G.; Cameron, A.-F.; Giannelia, T.; Jiang, J. Managing Artificial Intelligence Projects: Key Insights from an AI Consulting Firm. *Inf. Syst. J.* **2023**, *33*, 669–691. [CrossRef]
6. Martelli, A.; Ravenscroft, A.M.; Holden, S.; McGuire, P. *Python in a Nutshell*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2023; ISBN 978-1-09-811351-3.
7. Rajamani, S.K.; Iyer, R.S. Machine Learning-Based Mobile Applications Using Python and Scikit-Learn. In *Designing and Developing Innovative Mobile Applications*; IGI Global: Hershey, PA, USA, 2023; pp. 282–306. ISBN 978-1-66848-582-8.
8. Stevenson, W.J. *Operations Management*, 13th ed.; The McGraw-Hill series in operations and decision sciences; McGraw-Hill Education: New York, NY, USA, 2018; ISBN 978-1-259-66747-3.
9. Luo, D.; Thevenin, S.; Dolgui, A. A State-of-the-Art on Production Planning in Industry 4.0. *Int. J. Prod. Res.* **2023**, *61*, 6602–6632. [CrossRef]
10. Kashkoush, M.; ElMaraghy, H. Product Family Formation by Matching Bill-of-Materials Trees. *CIRP J. Manuf. Sci. Technol.* **2016**, *12*, 1–13. [CrossRef]
11. Hasanudin, M.; Andwiyan, D.; Yuliana, K.; Tarmizi, R.; Haris; Nugroho, A. E-SCM Based on Material Inventory Management Uses the Material Requirements Planning Method. *J. Phys. Conf. Ser.* **2020**, *1477*, 052006. [CrossRef]
12. Tobon-Valencia, E.; Lamouri, S.; Pellerin, R.; Moeuf, A. Modeling of the Master Production Schedule for the Digital Transition of Manufacturing SMEs in the Context of Industry 4.0. *Sustainability* **2022**, *14*, 12562. [CrossRef]
13. Heizer, J.; Render, B.; Munson, C. *Operations Management: Sustainability and Supply Chain Management*, 12th ed.; Pearson: Boston, MA, USA, 2017; ISBN 978-0-13-413042-2.
14. Magad, E.L.; Amos, J.M. The Impact of Material Requirements Planning and Distribution Requirements Planning on Materials Management. In *Total Materials Management*; Springer: Boston, MA, USA, 1989; pp. 188–221. ISBN 978-1-4684-6568-6.
15. Bowers, M.R.; Camm, J.D.; Chakraborty, G. The Evolution of Analytics and Implications for Industry and Academic Programs. *Interfaces* **2018**, *48*, 487–499. [CrossRef]
16. Mehta, B.S.; Awasthi, I.C. Industry 4.0 and Future of Work in India. *FIIB Bus. Rev.* **2019**, *8*, 9–16. [CrossRef]
17. Kumar, A.; Shrivastav, S.K.; Oberoi, S.S. Application of Analytics in Supply Chain Management from Industry and Academic Perspective. *FIIB Bus. Rev.* **2021**, 231971452110280. [CrossRef]
18. Rana, R.L.; Adamashvili, N.; Tricase, C. The Impact of Blockchain Technology Adoption on Tourism Industry: A Systematic Literature Review. *Sustainability* **2022**, *14*, 7383. [CrossRef]
19. Christofi, M.; Vrontis, D.; Thrassou, A.; Shams, S.M.R. Triggering Technological Innovation through Cross-Border Mergers and Acquisitions: A Micro-Foundational Perspective. *Technol. Forecast. Soc. Chang.* **2019**, *146*, 148–166. [CrossRef]
20. Python Welcome to Python.Org. Available online: https://www.python.org/ (accessed on 29 October 2023).
21. Mills, A.; Durepos, G.; Wiebe, E. *Encyclopedia of Case Study Research*; SAGE Publications, Inc.: Thousand Oaks, CA, USA, 2010; ISBN 978-1-4129-5670-3.
22. Rocco, T.S.; Plakhotnik, M.S. Literature Reviews, Conceptual Frameworks, and Theoretical Frameworks: Terms, Functions, and Distinctions. *Hum. Resour. Dev. Rev.* **2009**, *8*, 120–130. [CrossRef]
23. Mohamed, K.S. IoT Physical Layer: Sensors, Actuators, Controllers and Programming. In *The Era of Internet of Things: Towards a Smart World*; Mohamed, K.S., Ed.; Springer International Publishing: Cham, Switzerland, 2019; pp. 21–47. ISBN 978-3-030-18133-8.
24. Team, T.P.D. Pandas: Powerful Data Structures for Data Analysis, Time Series, and Statistics. Available online: https://pypi.org/project/pandas/ (accessed on 27 August 2023).
25. McKinney, W. Pandas: A Foundational Python Library for Data Analysis and Statistics. *Python High Perform. Sci. Comput.* **2011**, *14*, 1–9.
26. Pajankar, A.; Joshi, A. Introduction to Pandas. In *Hands-on Machine Learning with Python*; Apress: Berkeley, CA, USA, 2022; pp. 45–61. ISBN 978-1-4842-7920-5.
27. Mishra, V.K.; Sebastian, S.; Iqbal, M.; Anand, Y. Dealing with Missing Values in a Relation Dataset Using the DROPNA Function in Python. In *Mathematics and Computer Science Volume 1*; Ghosh, S., Niranjanamurthy, M., Deyasi, K., Mallik, B.B., Das, S., Eds.; Wiley: Hoboken, NJ, USA, 2023; pp. 463–470. ISBN 978-1-119-87967-1.
28. Mumtaz, R.; Amin, A.; Khan, M.A.; Asif, M.D.A.; Anwar, Z.; Bashir, M.J. Impact of Green Energy Transportation Systems on Urban Air Quality: A Predictive Analysis Using Spatiotemporal Deep Learning Techniques. *Energies* **2023**, *16*, 6087. [CrossRef]
29. Team, T.P.D. Pandas.DataFrame.Astype—Pandas 2.0.3 Documentation. Available online: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.astype.html (accessed on 27 August 2023).
30. Chelliah, B.J.; Latchoumi, T.P.; Senthilselvi, A. Analysis of Demand Forecasting of Agriculture Using Machine Learning Algorithm. *Environ. Dev. Sustain.* **2022**, 1–17. [CrossRef]

31. Duhem, L.; Benali, M.; Martin, G. Parametrization of a Demand-Driven Operating Model Using Reinforcement Learning. *Comput. Ind.* **2023**, *147*, 103874. [CrossRef]
32. Phoon, K.-K.; Zhang, W. Future of Machine Learning in Geotechnics. *Georisk Assess. Manag. Risk Eng. Syst. Geohazards* **2023**, *17*, 7–22. [CrossRef]
33. Khan, M.A.; Saqib, S.; Alyas, T.; Ur Rehman, A.; Saeed, Y.; Zeb, A.; Zareei, M.; Mohamed, E.M. Effective Demand Forecasting Model Using Business Intelligence Empowered with Machine Learning. *IEEE Access* **2020**, *8*, 116013–116023. [CrossRef]
34. Elbegzaya, T. Application AI in Traditional Supply Chain Management Decision-Making. Available online: http://dspace.unive.it/handle/10579/17733 (accessed on 1 November 2023).
35. Caro, F.; Gallien, J.; Díaz, M.; García, J.; Corredoira, J.M.; Montes, M.; Ramos, J.A.; Correa, J. Zara Uses Operations Research to Reengineer Its Global Distribution Process. *Interfaces* **2010**, *40*, 71–84. [CrossRef]
36. Caro, F.; Gallien, J. Inventory Management of a Fast-Fashion Retail Network. *Oper. Res.* **2010**, *58*, 257–273. [CrossRef]
37. Siegwart, R. PyBOM. Available online: https://github.com/robsiegwart/python-BOM (accessed on 29 August 2023).
38. Prajogo, D.; Olhager, J. Supply Chain Integration and Performance: The Effects of Long-Term Relationships, Information Technology and Sharing, and Logistics Integration. *Int. J. Prod. Econ.* **2012**, *135*, 514–522. [CrossRef]
39. Klapita, V. Implementation of Electronic Data Interchange as a Method of Communication Between Customers and Transport Company. *Transp. Res. Procedia* **2021**, *53*, 174–179. [CrossRef]
40. Scala, S.; McGrath, R. Advantages and Disadvantages of Electronic Data Interchange an Industry Perspective. *Inf. Manag.* **1993**, *25*, 85–91. [CrossRef]
41. Basole, R.C. Accelerating Digital Transformation: Visual Insights from the API Ecosystem. *IT Prof.* **2016**, *18*, 20–25. [CrossRef]
42. Matplotlib, T. Matplotlib—Visualization with Python. Available online: https://matplotlib.org/ (accessed on 31 August 2023).
43. Plotly, T. Plotly: Low-Code Data App Development. Available online: https://plotly.com/ (accessed on 31 August 2023).
44. Lee, Z.; Lee, J. An ERP Implementation Case Study from a Knowledge Transfer Perspective. In *Second-Wave Enterprise Resource Planning Systems*; Shanks, G., Seddon, P.B., Willcocks, L.P., Eds.; Cambridge University Press: Cambridge, UK, 2003; pp. 335–350. ISBN 978-0-521-81902-2.
45. Gupta, M.; Kohli, A. Enterprise Resource Planning Systems and Its Implications for Operations Function. *Technovation* **2006**, *26*, 687–696. [CrossRef]
46. Harjunkoski, I.; Maravelias, C.T.; Bongers, P.; Castro, P.M.; Engell, S.; Grossmann, I.E.; Hooker, J.; Méndez, C.; Sand, G.; Wassick, J. Scope for Industrial Applications of Production Scheduling Models and Solution Methods. *Comput. Chem. Eng.* **2014**, *62*, 161–193. [CrossRef]
47. Parthasarathy, S.; Daneva, M. Customer Requirements Based ERP Customization Using AHP Technique. *Bus. Process Manag. J.* **2014**, *20*, 730–751. [CrossRef]
48. Esteso, A.; Peidro, D.; Mula, J.; Díaz-Madroñero, M. Reinforcement Learning Applied to Production Planning and Control. *Int. J. Prod. Res.* **2023**, *61*, 5772–5789. [CrossRef]
49. SA, O. Openerp-Mrp: MRP. 2014. Available online: https://pypi.org/project/openerp-mrp/ (accessed on 27 August 2023).
50. Ganesh, A.; Shanil, K.N.; Sunitha, C.; Midhundas, A.M. OpenERP/Odoo—An Open Source Concept to ERP Solution. In Proceedings of the 2016 IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, India, 27–28 February 2016; IEEE: Bhimavaram, India, 2016; pp. 112–116.
51. Jay.devs Top 30 Companies That Use Python for Success and Profit. Available online: https://jaydevs.com/top-companies-that-use-python/ (accessed on 1 September 2023).