Graduating Project

Programme Grande École - 3rd year (PGE3)

MSc in Data and Business Analytics (MSc. DBAN)

How can advanced analytics and data science be applied to optimize sustainable supply chains for enhanced environmental and financial performance?

Tom OLEJNICZAK

Supervisor: SADOGHI Amirhossein

Abstract

This study aims to understand how companies can optimize their supply chain process for both environmental and financial benefits using the data that is available to them. This will be done using a public dataset, made available by the Brunel University of London. Based on this data, an optimization method will be implemented to minimize the sum of the warehousing cost and transportation cost. A Linear Programming (LP) model will be used to solve this problem. After attempting to establish the best model to solve this problem, the objective function will be displayed and the economic implications expressed. The results should underline that implementing a LP model, and overall an optimization model applied to supply chain management (SCM), should benefit companies by both reducing cost and carbon footprint, by using the most optimal transportation routings.

This graduating project, by finding an optimal solution to the problem, demonstrates how optimization models can significantly reduce total supply chain costs, while also reducing carbon footprint. In a practical context, this result is crucial and should not only be expressed to data professionals, hence the importance of using data analytics to communicate results to non-technical teams as well. It should however be noted that the model relies on simplified assumptions, which may not take into account the full complexity of the supply chain. Additionally, the data is a sample from a single day of operations: the model would increase in complexity as the size of the data increases as well.

Acknowledgments

I want to thank Sir Amirhossein SADOGHI for all the guidance throughout the elaboration of this graduating project. I also want to thank Nadjib BRAHIMI for his knowledge and work as the director of the MSc Data and Business Analytics.

Additionally, I would like to thank Jean-Damien LEROY and Bruno PENNA for their contributions throughout my academic journey. Their influence and support have been greatly appreciated.

Table of Contents

Table of Figures

1. Introduction: Research Question & Justification

In 2023, Splunk Inc. demonstrated in a study that the more mature a company is when it comes to managing its data, the more financial benefits they gain from their practices and the more subject to innovation they become. In this domain, French companies tend to lag behind, and are struggling to capitalize on all their data assets.

Integrating data science into company practices has had an undeniable impact on companies' economic growth. The analysis of big data can help companies improve their market trends prediction, operations optimization, while also tailoring the customer experience. All these serve the similar goal of improving profits. Despite these advancements, a seemingly underexploited use of data science is to serve the purpose of enhancing the quality of companies' environmental practices.

In this paper, we will attempt to evaluate to which extent the use of data science in companies can be used for not only financial gains, such as cost reduction or increase in profitability, but also to improve the environmental impact of the supply chain of the companies. We will not be seeking to diminish the importance of data science on financial gains, but to expand its horizon to merge these financial objectives with environmental sustainability. With the likely administration of stricter environmental constraints on companies (leading potentially to financial losses), data science can play a crucial role by enabling firms to spot and implement more efficient & less resource-demanding processes that not only reduce environmental impact but also lower cost, thus integrating financial performances with ecological management.

Therefore, this research will investigate how advanced analytics and data science can be applied to optimize sustainable supply chains, with the objective of enhancing environmental and financial performance?

To answer this question, we will now focus first on the available research papers online, to understand what findings have already been made on this topic, but also what the limits of the literature are to understand what goals our research paper will have to achieve. We will then cover a methodology proposal to understand how to achieve this goal.

1.2. Literature Review: Learnings & Critique of Related Research

In this section, we will review available research that focuses on the use of data science within the supply chain, how it is used for financial benefits and how companies are starting to use it for environmental benefits as well. Our goal is to understand the current state of data analytics in this context, what are the current obstacles and what is needed to establish a clear framework to leverage the use of data science for environmental and financial benefits. We will learn from available research, but also understand their limitations and how our research could contribute to this topic.

Merlino M. and Sproge I. (2017) underlined that Supply Chain operations should take a new direction, integrating new technologies such as artificial intelligence (AI), Big Data Analytics (BDA), in order to become more sustainable facing environmental changes. Authors point out how digital technology was already enhancing collaboration, back in the year of their study, and they understand that BDA is improving supply chain visibility, optimizing inventory management, assessing risk while also improving customer satisfaction.

Our paper will explore how integrating data science can extend beyond financial advantages to significantly enhance the environmental sustainability of companies' supply chains, merging economic and ecological objectives in the face of potential stricter environmental regulations. Modern companies, if they consider data analytics in their strategies, often lack structure in today's Big Data Analytics (BDA) environment (Raeesi, R., Sahebjamnia, N., & Mansouri, S. A, 2023). This would mean potential improvements in the value of BDA if the management of the data improves as companies gain importance of consciousness of the importance of BDA.

Sustainable Supply Chain Management (SSCM) and its development has been studied by Stefan Seuring & al (2022), and how it transitioned from a niche topic of interest to a point of focus from various companies across many industries. This research paper also underlines the modern points of concern for the evolution of SSCM, such as stakeholder management problems or risk management. Importantly, this research explains how there remain many points of improvement for SSCM, where environmental and social points remain to be explored and implemented in SSCM. Here, our research could offer a framework on how BDA and data management can be used to explore the potential of environmental and social benefits within SSCM. Furthermore, research emphasized the importance of digital transformation as a factor of sustainable improvement in the supply chain, and data analytics is at the core of numerous numerical transformations projects. However, we may be cautious when drawing conclusions from this

research paper, as it focuses on selective literature from the authors, which can be responsible for an unconscious bias when making conclusions. This paper also struggles to offer solutions to tackle the issues faced by modern SSCM, while we will be aiming to offer a framework in our research, especially while focusing on digital transformation. Finally, this paper mentions digital transformation, which while important for our research specifically, we will work on Data Analytics, which takes part in digital transformations but is not the main component of it.

Taghikhah F. Daniel J., Mooney G. (2017), in their research, aim to demonstrate the crucial role of BDA in enhancing the sustainability of the supply chain. The 3 areas of research they focus on are configuration, implementation, and evaluation. Their methodology is to use a total of 311 papers, from 2012 to 2016. After reviewing these papers, authors suggest that decision-makers are looking for actionable insights to optimize sustainability outcomes. However, authors identify a need for more dynamic, predictive, and behaviorally informed models to make progress in terms of SSCM. They believe predictive analytics could forecast the impact of sustainability initiatives, whereas behavioral data models could give companies better insights into human factors affecting Sustainable Supply Chain performances. While the insight from their research is important for our topic, we need to consider the fact that they focus on papers that sometimes are a decade old, and that they may omit recent advancements and trends in the field of both supply chain and data analysis.

Mageto J. (2021) also focuses in their study on the link between BDA and SSCM. They focused on the context of the manufacturing industry and pointed out the need for sustainability within the supply chain to align with the United Nation's sustainability goals. Mageto J. provides us a clear understanding of how BDA can drive SSCM in the manufacturing sector: there is a clear need of real-time, heterogeneous data analysis to create transparency across the supply chain, therefore promoting open and accountable practices. BDA can also provide insights to promote a sustainability culture within a company, and its insights can detect early on supply chain risks. This research paper Toulmin's Argumentation Model is a framework for analyzing and constructing arguments, breaking them down into six components: claim, grounds, warrant, backing, qualifier, and rebuttal, to understand the structure and logic of persuasive communication. It emphasizes the practical connections between an argument's main point, the evidence supporting it, and the logical reasoning that links them. This research paper uses the Toulmin Argumentation Model: by breaking down arguments into claim, grounds, warrant, backing, qualifier and rebuttal, this framework for constructing arguments helps understand the connection between an argument's

principal point, its evidence and the logical reasoning behind it. While this is very insightful for our research, it lacks empirical data to support the arguments that are being made.

Rakesh R. Menon and V. Ravi (2021) went further into analyzing the limitations of implementing initiatives within SSCM, by focusing on the Indian electronics industry. They identified various barriers, categorized into Policy, Human Resources and Technological barriers. The main barriers identified are lack of commitment from top management, lack of new technologies or of awareness of the benefits of sustainability, or even resistance to change. BDA could have a positive impact on abolishing technological barriers, but also to overcome another mentioned burden, it being the lack of performance metrics/evaluation standards on sustainability. The Goal of BDA is to deliver insights on the data that is analyzed and could create valid metrics to measure the effects of positive SSCM practices. Overall, this paper tends to underline that the lack of awareness of the benefits of SSCM practices is a massive burden in its development, which data analytics could solve. It is insightful information to keep in mind, and it's even more valuable since this research is based on a specific context in the form of the Indian electronics industry. The study directly relates to our topic on optimizing sustainable supply chains with advanced analytics but does not explicitly offer solutions to overcome the mentioned barriers.

Arunachalam, Kumar, and Kawalek (2018) evaluate how BDA and its capabilities are evolving constantly. They explain how BDA may enhance Supply Chain Management (SCM) and suggest a framework to display how it does so. BDA capabilities can be demonstrated by the range of purposes it can serve, from the initial data generation to advanced data visualization to tell a story to various stakeholders. This could explain why, according to their research, BDA is starting to play a massive role in SCM. This can be done for companies of various sizes and with various data maturity. They present various BDA capabilities dimensions: importance of quality data and keeping the data structured, data integration capabilities to collect, integrate, transform and store data, the application of sophisticated analytical techniques or even the importance of a data-driven culture. They also offer a framework to emphasize the importance of data maturity, claiming some companies are Data and Analytics Poor, while others are Data and Analytics Rich and how they should tend towards that goal. However, introducing BDA to SCM may lead companies to face various challenges, such as privacy concerns, lack of skills from the Supply Chain (SC) management team, or even the cost or issues linked to data scalability. Authors present the need for a strategic approach to overcome these various challenges, putting the accent on the role of leadership, skill development and cultural change. Overall, this research paper points out how BDA plays a massive role in contributing to the improvement of SCM, and how predominant data is in the organization

of the modern supply chain. It provides us with a clear framework and understanding of how BDA plays a massive role in modern SCM, even offering us insights from a strategic point of view. It goes further to mention the importance of the quality of data, by underlining how BDA lacks quality if it misses crucial organizational and structural aspects. However, it seems to only focus on financial gain, and generating financial insights to stakeholders. Sustainability is mentioned but not as important as the financial gain in this model. This paper may provide us with a clear framework moving forward.

Weersink and al. (2018) attempted to apply those principles to the agricultural sector, explaining how BDA can improve agricultural productivity and environmental sustainability. For this specific sector, the insights from data analytics aim to improve precision agriculture and traceability, being an example of how BDA can not only be linked to financial gain, here by improving efficiency therefore reducing cost, but also being at the origin of better environmental practices. While demonstrating a clear link between BDA and optimizing sustainable supply chain in the sector it focuses on, it also showcases how challenging it may be to evaluate a clear return on investment for farmers that decide to increase the value they allocate to data analytics. Our study will aim to overcome challenges mentioned by Weersink and al., such as displaying a clear return on investment, or other challenges mentioned by authors such as stricter environmental policies, skill gaps between the stakeholders of environmentally friendly practices and data professionals, or the importance of a clear data governance. Weersink & al. studied a sector where environmental policies could clearly be considered, which is not the case for every industry. However, even then, the industry faces technological and organizational challenges, displaying how challenging it could be to make the role of data analytics vary based on different industries.

Container terminal operations (CTOs) play a major role in supply chain as a series of planning tasks performed between a container terminal: Raeesi, Sahebjamnia, and Mansouri (2023) focused in their research on how BDA can improve environmental sustainability within those container terminal operations. They outlined how the use of BDA, here focusing on more sustainable CTOs, is for the most part unexplored. In the sector of focus, here container terminal operational issues, environmental improvement could include the decarbonization of a major part of CTOs. However, it is explained how BDA is barely involved in that process, focusing rather on operational efficiency. The main reason for this limitation of involvement of BDA in environmental improvement is the lack of a guiding framework to accomplish exactly this without compromising the financial gain and optimization of the supply chain. Our study will aim to evaluate how a

guideline could be provided to merge both supply chain optimization and environmental considerations.

Researchers found that the introduction of a green supply chain can enhance financial performances (Kong, T., Feng, T. and Huo, B., 2021). They demonstrated how green supplier introductions facilitate information sharing, while not negatively affecting financial performances. This paper also does not exclude a positive correlation between sustainable practices in supply chain and financial performance. Leveraging data science for environmental benefits and using the insights from BDA to improve the ecological impact of the supply chain is therefore not only viable but also profitable. The relevance of enlarging the use of BDA for environmental benefits in the supply chain is important in the eventuality of strict environmental regulations that may be applied to companies in the future. Rugman and Verbeke (1998) demonstrated before these policies seemed unavoidable how strict environmental policies can influence firms' strategic decisions, suggesting that multinational companies can leverage environmental regulation to gain competitive advantage. In today's context, this said leverage could be done by generating environmental insights from BDA.

Available literature clearly demonstrates the potential of BDA to make supply chains both economically efficient and environmentally sustainable. While this is a goal for companies, the research we focused on demonstrated several limits for changes, such as data management, privacy concerns or even skill gaps. A framework is needed to assess environmental and social benefits as key performance indicators.

While helping us understand the issue, few available literatures suggest a framework to solve these. Our goal will be to establish a framework to understand how companies can integrate BDA within SCM to be more sustainable without getting rid of financial gains. It is important to note that we will continue doing research and that additional literature may contribute to establishing this framework.

1.3. Proposed Methodology

To answer the question "How can advanced analytics and data science be applied to optimize sustainable supply chains for enhanced environmental and financial performance?", supply chain data will be needed. First, the data that will be used will be presented. The data will focus on supply chain operations, and the link with the research topic will be illustrated. Then, a data analysis step, consisting of data cleaning, statistical description, and data visualization will be achieved. This will

enable a broad understanding of the dataset to then build the model as efficiently as possible. A first optimization model will then be built. This will only be possible after the data analysis steps provide a clear understanding of the metrics to then establish the objective function, and the associated constraints. This model will provide an objective function, representing the minimized optimal cost of operating the supply chain. Based on the performance of the model, both in the quality of the returned objective function and the time taken to solve the problem, it could be subject to improvements. Once the model has proven to provide an optimal value for the objective function in a prompt amount of time, the result of the objective function will be compared to the cost of operating the supply chain without the optimization model, underlining the benefits of implementing the model. The output should provide insights from both financials and environmental aspects: the solution will return optimized routing, meaning the transportation and warehousing costs will be reduced, and as the routing options are optimized, less but more efficient achieved routing would mean a decrease in the company's carbon footprint. The optimization model should provide actionable insights into improving the supply chain process, and how it can contribute to reducing costs and environmental impact. However, these results may be limited by assumptions within the model. The limitations of the analysis should be examined by evaluating how the model would perform under different business cases, and what are the limits of implementing such models in the supply chain process.

1.4. Sample and data description

The dataset we will be using is a public dataset from the Brunel University of London, focusing on supply chain logistics, designed to address a real-world optimization problem. Optimizing this given problem would allow a company to help reduce environmental impact by minimizing transportation distances, leading to lower carbon emissions. From a financial standpoint, improving route efficiency would be crucial to cut costs. This dataset is divided into seven tables, containing either data linked to operations, or constraints linked to the problem. The main table is *OrderList*, which contains the history of orders that need to be assigned a route. The file contains 6 other files. First, *FreightRates* represents every available courier, the specific weight constraints for each route, and the cost of each routing option. The *PlantPort* table displays the available links between different warehouses and shipping ports. The *ProductsPerPlant* table lists every supported warehouse-product combination. *VmiCustomers* lists all cases where the warehouse is only allowed to support specific customers. When information is not listed in this table, any warehouses can supply any customers. Finally, *WhCapacities* lists warehouse capacities measured by the number of

orders that can be handled per day, and *WHCosts* gives us information on the cost of storing products in each warehouse. The following analysis will be conducted.

For the tools used, the analysis will be done using Python and Power BI. This will enable to use both a data visualization tool and a coding language to display as much information as possible on the dataset.

For the analysis, the dataset will be subject to data cleaning, which will consist of the handling of missing or inconsistent values. The, a data summary such as the mean, median and standard deviation for efficiency metrics, and trend analysis for time-series data, if applicable, will be done. A summary of key metrics should be considered, to display relevant information before establishing the model, which can be done with data visualization. Field of focus for the data analysis could include the distribution orders by origin and destination port, carriers, and other key operational factors that is included within the dataset. Considering the multiple tables available, merging the datasets may be included in the data analysis part, but could also be part of the analysis

The next step will consist of an optimization model aimed at minimizing the supply chain cost and carbon footprint. This process will involve setting up an objective function with the related constraints, such as the warehousing capacity, transportation limit, and available routing options. The model will operate under a set of the constraints. Based on the constraints, multiple models could be considered. For example, if the constraints follow the form of a linear equation, a linear programming model should be considered. By implementing a real operational approach with the related constraints, the model should produce actionable insights that can be directly applicable to a business, using operational data available to most companies.

2. Optimizing Sustainable Supply Chains: a practical approach

2.1 Introducing the dataset: link between the dataset and sustainable supply chain optimization

Using real-world data enables the analysis of sustainable supply chain optimization under practical conditions. This ensures the model considers operational challenges like fluctuating demand, transportation limits, and warehousing constraints that companies regularly face. By working with actual data, the analysis provides solutions that are not only theoretically sound but also applicable in real business contexts. This approach helps to ensure that the optimization results are both practical and relevant for financial gains and reducing environmental impact.

First, a data analysis set, with both description and visualization, will allow the understanding of the given dataset in terms of specific information included, but also on the size and formats of the given information. After understanding the data at stake, the information from the data analysis step will allow for the formulation of the optimization model. Considering the given information so far, the problem is a minimization of the supply chain cost. The results of a first model will be taken into consideration, and used to refine our formulation of the problem and build a second model that returns a value for the objective function. The end goal is to elect one final model that will be considered as the best at displaying the result of the objective function.

2.2 Data Analysis

2.2.1 Data Description

While most of the data analysis part will be done using Python, a Power BI report will first be created to get a first understanding of the dataset at stake. In a business context, this would allow stakeholders to see a narrative from the data, highlighting key insights in a clear and accessible format. This dashboard will provide an overview of the data for the given day that is accessible to us.

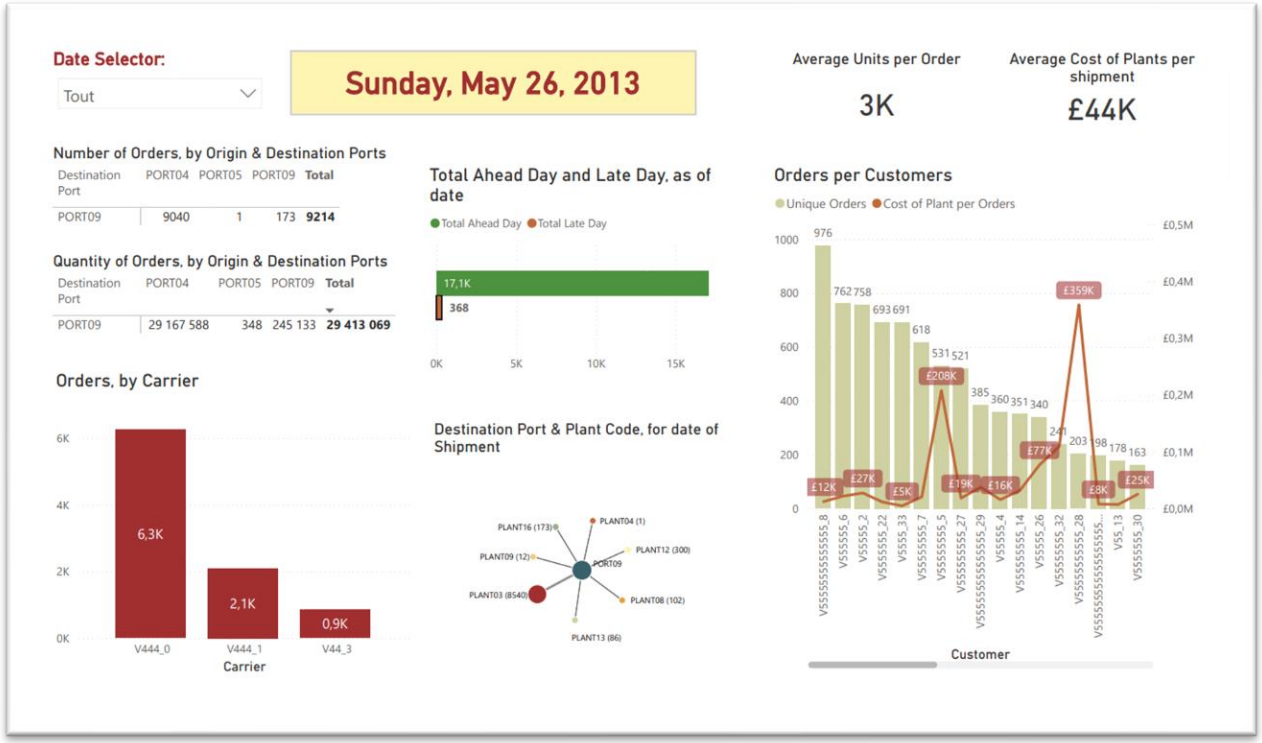


Figure 1:Power BI: operational data dashboard

This dashboard, using a daily granularity, would display several pieces of information to stakeholders. First, the Power BI user will have available a date filter, to display and monitor data from any given day. Our sample contains information for one date: May 26, 2013, which is displayed as a header of our dashboard. For this given date, the number of orders by origin and destination port is displayed, to evaluate which destinations are the most frequent. The most frequent route for the given day is to have origin port 4, 5 and 9, and destination port 9. Underneath, the quantity ordered for these orders is displayed: most of the orders go from port 4 to port 9. The total ahead day and late day display, as of May 26, the number of ahead days, but most importantly the number of late days, allowing business teams to monitor our on-time performance. A bar chart will display the volume of orders by carriers, and a journey chart represents the plant code of items for the destination port. Two date carts display the average unit per orders, and the average cost of plants per shipment per day. Finally, the end user can explore, for each customer, the number of items ordered, and the cost associated with each order. The cost is calculated by multiplying the cost per unit from the *WhCosts* table with the unit quantity of every different item from *OrderList*.

This dashboard will be useful to display information to stakeholders, however, it is not sufficient to get a complete understanding of our dataset.

First, relevant python libraries are imported. A color list is created to maintain uniformity across all graphs, ensuring cohesive presentation coherent with the standards expected in professional reporting. The dataset is imported, making sure we take into consideration every sheet of our file.

The data description section clarifies the content of the excel file, and what information is within the various sheets. Description will include numbers of rows, numbers of columns and the name of columns. Our main table, *OrderList*, consists of 9215 rows and 14 columns. *FreightRates* has 1540 rows, and 11 columns. Other tables have 2 columns and a few numbers of rows to store the constraints linked to our problem. This confirms that *OrderList* is our main table, containing what we are assuming to be logs of orders. Other sheets, due to the lack of various columns and lines, are either additional information, or constraints that we could be using in the optimization. This is why the statistical description only involves *OrderList*.

From the statistical description, the data analysis part will display statistical information from unit quantity (μ=3202.75, min = 235, Q1=330, Q2=477, Q3=1275, max = 561847, σ=15965.62), transportation day count which is related to estimated shipping time (TPT) (μ=1.717743, min = 0, Q1=1, Q2=2, Q3=2, max = 4, σ=0.6305), or the weight of the transportations (μ=19.87, min =

0, Q1=1.4, Q2=4.44, Q3=13.32, max = 2338.40, σ=66.56). This statistical description provides insights into the distribution of the elected variables. These insights are beneficial for understanding the range and central tendencies of these variables: the main aspects to consider include the high variability in the unit quantities within each shipment, which could impact the shipping optimization. The information regarding the day count distribution shows most shipments occur within one or two days, which could be crucial information in lead time forecasting scenarios. Furthermore, the wide range in transportation weight suggests that shipping strategies may need to take into account the diversity of load sizes on a goal to optimize costs.

Addressing null and duplicate values is an essential step moving forward, as it will ensure the data accuracy, especially for our optimization model, moving forward. Null values would lead to incomplete calculations and distortions in the insights provided, while duplicate values can inflate the significance of the duplicated metrics. If any component of the dataset contains either null or duplicate values, a data cleaning step will be added. The *check_duplicate_values* function will identify duplicate entries within the datasets, using *df.duplicated()*. The sum of these duplicate values is then reported. When duplicate values are found, *df.drop_duplicates()* will be applied ensuring that only unique values are retained within the dataset. After checking for null or duplicate values, the output returns that no tables have null values, only *FreightRates* have duplicate values: those will be dropped. The output returns that there are no remaining duplicate values in *FreightRates*, meaning that the dataset is cleaned and can be used for further analysis.

2.2.2. Data Visualization

From the description of *OrderList*, and from the Power BI report, it seemed that the dataset would contain dates from a singular day. A query is established to return unique date values, and confirms that the dataset contains data from the 26/05/2013, as mentioned previously. This is crucial to establish that no time series data visualization techniques, which rely on trends across multiple time periods, are applicable in this case. This implies that the analysis will focus on understanding the static snapshot of supply chain operations for a single day. Insights will remain contextually relevant for that specific day, underlining the need to focus on operational data insights for the data analysis step, rather than trend analysis.

The Orders by Origin Port graph, similar to the one displayed in the Power BI dashboard, provides a view of the distribution of orders by origin port. This is a valuable insight within the supply chain process to understand the most important carriers used. A pie chart will display that the carrier distribution from our dataset is the following: 23% of transportations is done with carrier *V444_1*,

68% with *V444_0*, and the remaining 9% with *V44_3*. In a business context, this information would be important for route planning: there is a heavy reliance on carrier *V444_0* in transportation planning. In a risk management context, this would underline a carrier dependency, indicating a need to diversify the transportation methods to avoid supply chain distributions if that carrier is subject to delays or issues. In a business context, this graph would allow decision-makers to visualize this information and take informed decisions to potentially balance carrier load.

Going beyond the Power BI report, specifically the graph displaying the day ahead and late day count, the On-time delivery performance by carrier is calculated. The on-time delivery performance by carrier provides an analysis on each carrier's ability to meet delivery deadlines. This is done by creating a new column, called *on_time_delivery*, within *OrderList*, taking into consideration when *ship_late_day_count* is greater than 0, then grouping by carriers. With all carriers seemingly having spot-on performance, having an on-time delivery rate greater than 97%, this graph demonstrates a high level of reliability across the supply chain process, with *V444_1* (99,57%) and *V44_3* (100%) reaching perfect on-time performance. In time-sensitive logistics operations, the analysis is essential for evaluating high on-time delivery rates.

A 3D graph, built using matplotlib (a python library designed for visualization), visualizes both minimum and maximum weight quantity and the freight rates charged by each carrier. In a business context, this would be particularly useful in comparing the pricing strategy of various carriers based on the weight of the shipment they can handle. The scatter plot loops through unique carriers in the *FreightRates* table of our dataset, returning their corresponding minimum and maximum weight and rates. This would provide a clear understanding of how carriers price their services for different weight ranges. The graph displays patterns based on the shipment weight: notably, it reveals lower rates for larger shipments, probably due to economies of scales, or specific weight thresholds that would lead to price changes. Seemingly, most carriers have pricing concentrated around lower weight quantities, indicating that most shipments should fall within a small weight range. However, outliers are present within the data: some carriers offer services for significantly heavier shipments, at higher rates, such as *V444_2*. These different price structures and options should be taken into consideration to elect carriers based on weight-specific pricing strategies, improving cost efficiency. However, a challenge with Jupyter Notebooks is the lack of options to zoom within the 3D plot, making it challenging to identify the values for every specific data point. However, on any other IDE, this would be practical to clearly see the rate of each minimum and maximum weight quantity associated with each carrier. The clear distribution of freight rates in relation to minimum and maximum shipment weight, and the pricing variation between carriers, remains a crucial takeaway

from this visualization, allowing in a business context to allow proper decision making regarding which carrier to select for specific shipment in regard to their sizes.

By merging *OrderList* and *WhCapacities* on the *plant_code*, the order capacity can be displayed for each plants. *daily_capacity_per_plant* can be defined by dropping duplicate values, to get a single capacity value per plant code. The result can then be plotted. *PLANT03* has the highest daily capacity with more than 1000 units and has the best warehousing option in terms of capacity volume. *PLANT16* is the second largest in terms of capacity. In comparison, *PLANT08* and *PLANT09* are smaller warehousing capacity, and might be underutilized because of this in the supply chain process.

Moving forward, the datasets will be merged to calculate the cost. To calculate the operational costs, multiple datasets are merged. The *OrderList* table is merged with *FreightRates* using the *carrier*, *origin_port* and *destination_port* as keys for the merging operation. The *OrderList* is then merged with *WhCosts*, using *plant_code* as a key to add the warehousing cost to the total cost. Following the merging operations, the column *cost* is created to compute the total operational cost, calculated by multiplying the unit quantity with the associated unit costs for both transportation and warehousing. Any rows containing missing values are removed using the *dropna()* function. These merge operations will allow further analysis based on the cost, such as understanding whether the operational costs scales proportionally with the weight being transported.

The following graph, displaying the average cost by weight category and carrier, shows a clear correlation between total cost and weight, with the total cost rising as weight increases. The matrix allows for a comparison of costs based on weight categories (Low, Medium, High, and Very High) between *V44_0* and *V44_1*, the two main carriers in the data. According to this graph, for shipments in the Low and Medium weight categories, carrier *V44_0* offers a more cost-effective solution. However, as shipment weight increases into the high and very high categories, Carrier *V44_1* becomes a more preferable option, with drastically lower costs for heavier shipments. In a business context, this would display to companies the option to adopt a differentiated carrier strategy based on shipment weight, with *V44_0* as an priority option for lighter shipments, and heavier shipments allocated to *V44_1*. Overall, based on the information displayed, actionable insights can be learned from this graph, to allow for better-informed transportation method selection.

To gain a better understanding of how the newly created cost metric interacts with the other metrics from the dataset, a correlation matrix will be used. A correlation matrix is a table that

displays the correlation coefficients between multiple variables in a dataset. Each cell in the matrix shows the correlation between two variables, with values ranging from -1 to 1. 1 indicates a perfect positive correlation; as one variable increases, the other also increases proportionally. 0 displays no correlation; changes in one variable do not predict changes in the other. -1 represents a perfect negative correlation; as one variable increases, the other decreases proportionally. By displaying a correlation matrix, it becomes possible to identify how variables from the dataset are related to each other, and if any pattern exists in the dataset. In a business context, this correlation matrix could help explain a trend from a specific metric, identifying links with another metric.

For instance *ship_late_day_count* and *on-time-delivery* have a correlation of -0.86 between these variables shows a strong negative correlation. This makes sense, as late shipments would likely lead to poor on-time delivery performance. Conversely, the relationship between *unit_quantity* and *cost* exhibits an exceptionally strong positive correlation of 0.99, indicating a nearly proportional increase in cost as the unit quantity rises. The weight has a correlation of 0.43 with the cost associated with shipments, underlining how the weight is correlated positively to the cost of the shipment: this makes sense as the heavier a shipment is, the more likely it is to impact the cost of said shipment. All of these assessments are logical within the supply chain context. Most of the other metrics seem to have no correlation with one another, with correlations between -0.10 and 0.10.

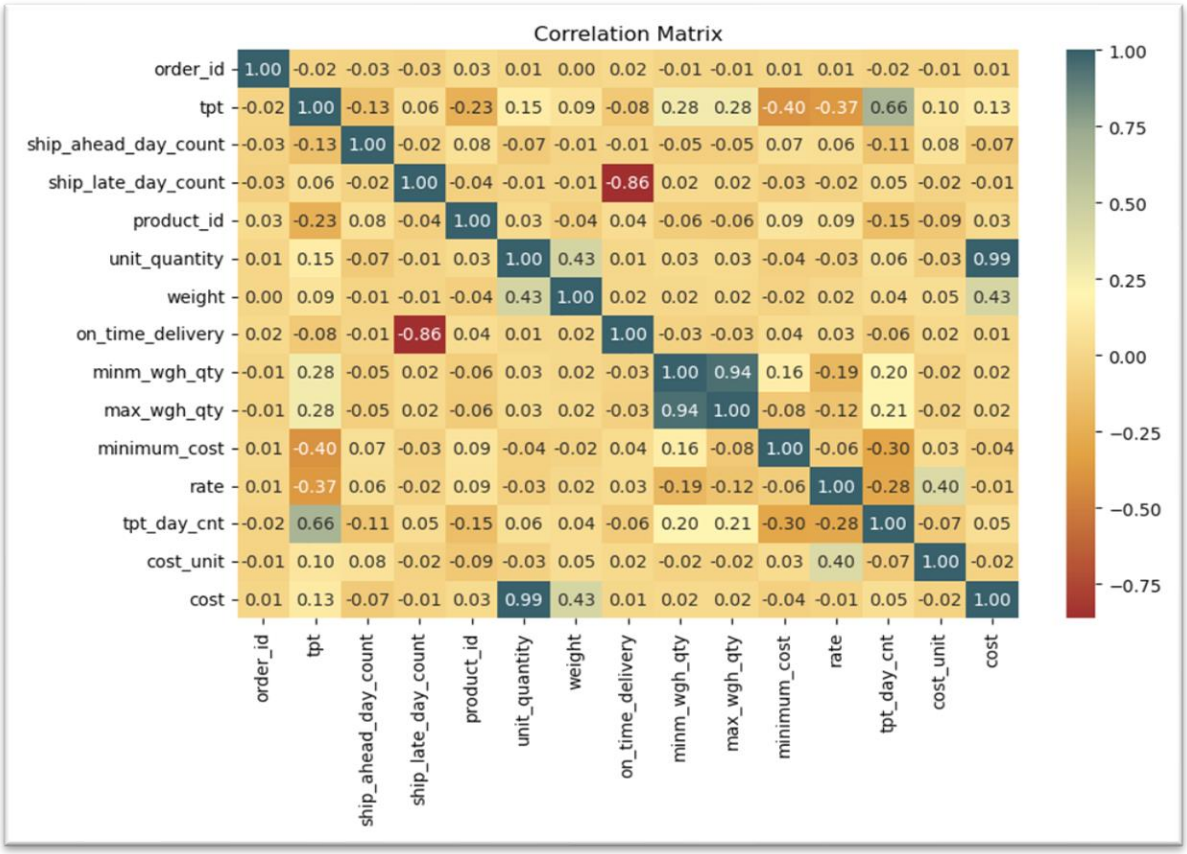


Figure 2: Correlation Matrix

Before optimizing the routing process, the available connection from the data between plant and port will be displayed. This information is crucial as it will be part of the constraints for the optimization model. First, all possible connections from table *PlantPort* will be displayed, representing every available option. Then, the displayed routes will be the one used in our *OrderList*.

The following graph is displaying all the possible routing combinations from the data and is part of the constraints: only these routes can be considered in the optimization process.

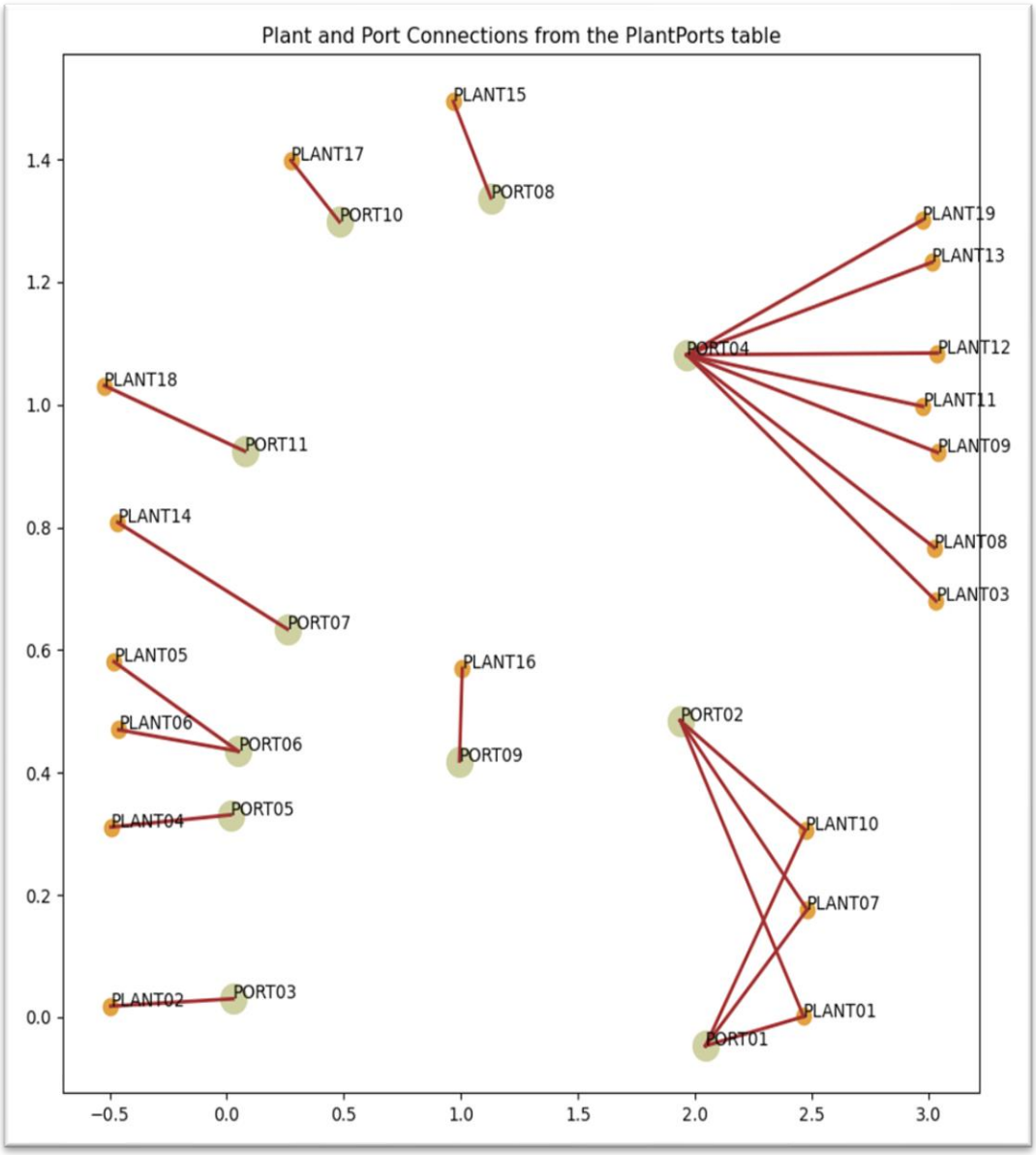


Figure 3: Plant and Port Connections from the PlantPorts table

Here, all of the options of plants to port assignments are represented by a red line. Multiple plants can be assigned to certain ports, such as *PORT04* and *PORT02* both having the option of serving a large cluster of plants. *PORT04* has the option to connect to *PLANT19*, *PLANT12*, *PLANT11*, *PLANT13*, *PLANT09*, and *PLANT03*, showing that it could be a major origin port from our data. On the other hand, *PLANT17* and *PLANT09* are both connected to a single port. Seemingly, *PORT04* is the most important port from our data. This is confirmed when displaying a similar graph from the *OrderList* table, displaying all the routing options that are actually used in the baseline data, referencing to the data from the daily supply chain optimization:

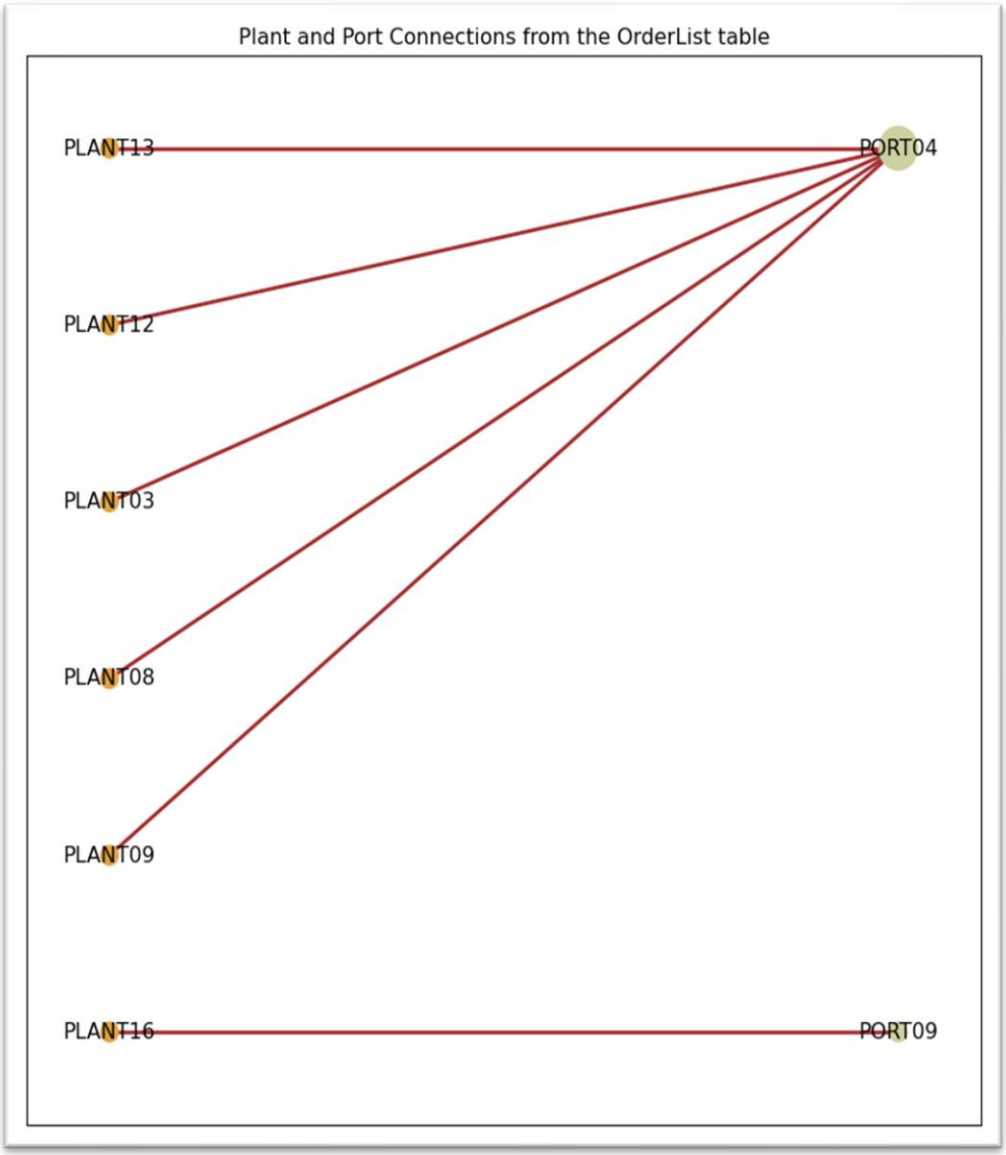


Figure 4: Plant and Port Connections from the OrderList table

This confirms that *PORT04* plays an important role within the supply chain, as it is the main used port in the actual data. *PLANT13*, *PLANT12*, *PLANT03*, *PLANT08*, and *PLANT09* are all directly linked to *PORT04*. Still from the *OrderList* table, *PORT09* is connected to only one plant, PLANT16. According to these graphs, there are 7 unique routing options in *OrderList*, and 22 from *PlantPorts*. These will be taken into consideration in the optimization model.

3. Optimization Model

3.1. Objective function and constraints

After gaining a better understanding of the available data, the upcoming goal is to optimize the routing based on a set of constraints that will be defined.. Optimization can be referred to as the

process of finding the optimal solution to a problem, or optimum, while satisfying all constraints and minimizing the objective (Foulds, L. R. 2012). In the given case study, the goal is to minimize the total costs of operating the supply chain, representing the sum of both warehousing and transportation cost, by finding the optimal routing options and resource allocation in the supply chain process. Based on the daily data available, the model will be built to make optimal adjustments of the variables. Firstly, this is done by defining the objective function, which represents the total cost to be minimized. This said function will operate under a set of constraints available in the dataset.

The objective function to minimize can be displayed as:

$$\text{Minimize } \boldsymbol{WC} + \boldsymbol{TC}$$
$$WC = \sum cWC_{kijp} X_{kijp}$$
$$TC = \sum cTC_{kijp} X_{kijp}$$

Having $cWC_{kijp}X_{kijp}$ the warehousing cost for orders *k* at plant *p*, and $cTC_{kijp}X_{kijp}$ the transportation cost for orders *k* from port *i* to port *j*.

To solve this optimization problem, we have multiple various constraints to take into consideration.

From the *WhCapacity* table: $\sum X_{kipj} \leq C_i$, having: X_{kipj} a Boolean variable equal to 1 if order *k* is shipped from warehouse *i* and 0 if not, C_i the maximum number of orders that warehouse *i* can handle per day.

From the *FreightRates* table: $\sum W_{kijcstm} \leq \max\,(F_{pjcstm})$, having: $w_{kijcstm}$ the weight in kilograms for order *k* shipped from *i* to customer port *j* via courier *c*, with delivery time *t* and transportation mode *m*. $\max\,(F_{pjcstm})$ being the maximum weight limit for shipments from port *i* to port *j* via courier *c*, using service level *s*, delivery time *t* and transportation mode *m*.

Additionally, one route cannot serve itself: an additional constraint to apply a non-self-routing system is established: having $X_{kipj} = 0$, for all *i* = *j,* a decision variable that indicates whether a route from port i to port *j* is selected (1 if the route is chosen, 0 otherwise). This ensures that *i* is never equal to *j*, therefore no orders is shipped from a port to a same port.

The goal is to minimize the sum of warehousing cost and transportation cost, under the given set of constraints. The constraints are linear: the variables from these constraints are not multiplied by each other. In other words, all of the constraints are formulated under linear equations, defining the feasibility region of the problem. For this reason, a linear programming model (LP) will be built to solve this problem.

3.2. Linear Programming Model: effectiveness, relevance and applications

Linear programming (LP) is a "mathematical tool for optimizing an outcome through a mathematical model, […] extensively used in the planning of different real-life applications such as agriculture, management, business, industry, transportation, telecommunication, engineering, and so on." (Kunwar, R., & Sapkota, H. P., 2022). A LP model aims to establish a model in which the linear function is maximized or minimized when subjected to various constraints. The goal is to find out a routing solution that minimizes the freight cost, while taking into consideration the constraints that we have established.

The effectiveness of LP models applies to the field of logistics, which is why it is a relevant model in our case. LP is also a powerful model for fields such as finance, production planning and energy management. Grothey, A. & McKinnon, K. (2023) apply LP techniques to solve pooling problems within supply chain: resource distribution needs to be optimized for cost efficiency while maintaining quality constraints. While LP algorithms like Simplex Models are pointed out mainly for their effectiveness, others, like Interior-Point methods, would rather be used for their short time of completion as the complexity of the problem scales up. The time to solve a problem with the Interior-Point method grows at a manageable pace, as the size of the optimization problem increases, whereas the time of completion of a Simplex algorithm increases at a similar rate than the complexity of the problem. LP algorithms can be used in complex problems containing uncertainty or unknown variables, which makes it a useful model for business context, improving the quality of the decision-making process (Silva, C., Ribeiro, R., Gomes, P., 2024). However, as mentioned previously, LP can be limited as the size and the complexity of the problem increases. In a business context, a large and complex problem would involve increasing computational costs, which could defeat the point of establishing this model initially.

To optimize sustainable supply chains, the following Linear Programming (LP) model is designed to minimize transportation costs between supply nodes and demand nodes. The code will be using the PuLP library, an open source LP modeler to create and solve optimization problems. The objective of the model is to minimize (*LpMinimize*) transportation costs, taking into consideration

our constraints mentioned previously. The time module will also be used, to track the time taken to solve the LP model. Before applying the model, a timer is initialized. The list of supply nodes is extracted from *WhCosts*, and the list of unique demand nodes from *OrderList*. Then, the supply dictionary is created, as *supply_dict*, mapping the supply nodes and the daily capacity from *WhCapacities*. The demand dictionary is also initialized, associating each demand node with the total quantity of units required.

For costs, for every order in OrderList, the code matches the corresponding freight rates based on carrier, *orig_port_cd*, *dest_port_cd*, *minm_wgh_qty* and *max_wgh_qty*. Every matching rate will then be added to the cost list created before. *Cost_dict* is created where each supply node maps to each destination node, associating these nodes to the transportation cost. With all of those criteria established, *prob* is defined as the LP problem under the name *MaterialSupplyProblem*. A list of tuples is created, representing all possible routes between each supply and demand nodes, and *LpVariables* will group each route and the corresponding variables representing the transported quantity.

The model will then need an objective function and the constraints linked to this function. The first constraints refer to the supply constraints: the sum of products transported from each warehouse should not limit the capacity established. The second constraint ensures that the sum of products transported to each destination port meets the demand. Finally, the self-transportation constraint prevent warehouses from shipping products to themselves by setting the associated variables to 0: when *w* (warehouse) = *b* (port), *vars[w][b]* is set to 0, preventing any transportation from occurring from a warehouse to itself, subtracting the self-routing routes from consideration in the model. Using the *CPLEX_CMD* solver, it will either solve the problem, or point out that the problem is infeasible with the given constraints. Once this is done, the timer is stopped, and the time will be formatted into minutes and seconds. The output will display the decision variables, the minimized value of the objective function, and the time taken to solve the problem.

In 11 minutes and 24 seconds, the model has established the results. All transportation routes returned the value "None": this also means that the objective function is also "None". From this, we can conclude that the model, as it is now, is infeasible. Infeasibility implies that it is impossible to allocate the available supply to meet the demand, taking into account the given constraints, costs, and warehousing capacities. A supply chain analyst should interpret this outcome as a proof that the configuration should be revised, or that the data should be adjusted: the infeasibility of the problem still remains a crucial learning from a business standpoint. In this case, we will attempt to relax one of the constraints, to obtain an objective function that is not "None".

3.3. Linear Programming: an alternative approach using PuLP

Regarding the result from the previous Linear Programming model, in this practical approach, a second LP model will be established.

The time of completion of the previous model indicates that there is a need to simplify the model and the data. Firstly, the *order_id* of Orders is changed to a simpler format from 1 to *order_no*. The useful datasets from our tables are retrieved, and reorganized. The linked tables are rearranged: Plants are connected to each Port, Available Carriers for Transportation between nodes, Capacity of Plants to execute orders, Vmi Customers and Plants. The output returns the number of orders, the list of available ports, the list of available plants, the list of Order IDs, and the list of demand ports, which are all to *PORT09*. The returned carriers info are the options of carriers from a port to another one, including the minimum and maximum capacity. As mentioned, since the constraints need to be relaxed, the maximum capacity is multiplied by 2 to have a feasible problem. The numbers of orders that each plant can manage are also listed. For the plants, the capacity is multiplied by 7, also to relax constraints.

To further simplify the data used, the *VmiCustomer* data is retrieved, selecting only the plants which can serve specific customers. An array is created, displaying every product that can be stored in each Plants. A list is also created to show which port can connect with which plant. Finally, to improve the model to display the connections between plant and port, random positions are allocated to both plants and ports.

The previous LP model displayed that the data cannot be handled as it is, and that it should be subject to preprocessing before creating the model. Knowing that the model returned an infeasible solution, the preprocessing part will ignore the impossible solutions: for example, if Port A does not store the Product J, then it is impossible to have such routing in the final solution. After removing the routings that are impossible, the LP model will attempt to find the optimal solution from these possible routings. To achieve this, first, dictionaries and counters are initialized. *possible_routes* stores information about each possible route for an order, including transportation and warehousing cost. order_chances keeps track of every potential shipping option, for each order. Then, the model loops through orders from *OrdersList*, to extract all of the needed order-related data. *is_there_source* will track whether a valid plant is found for the order, and *order_chances* will, for each order, initialize an empty list, where all possible routes will be stored. Within this loop, another loop will check each available port to see if it is a valid route for the order or not. *(port, dest_port)* in *carriers_info* makes sure that the combination of the source port and destination

port is feasible. If a feasible source is found, *is_there_source* is equal to "True". Finally, the model will establish the cost calculation. The transportation cost is calculated based on the weight of the order and the rate of shipping between the origin port and the destination port (if the service type is CRF, transportation port is 0, since no additional freight cost is charged). As for the warehousing cost, it is calculated by multiplying the quantity ordered by the warehousing cost per unit for each plant. These costs are then stored in *possible_routes*. If there is no feasible source found after checking all possible routings, "*No source available for product_id*" is printed. This preprocessed information will then be used to select optimal routes for each order.

This preprocessing step allows for a reformulation of the problem: in *possible_routings*, we have a list of all of the possible transportation, meaning the non-self-routing constraint should not be part of the constraint anymore. As the other constraints are met in *possible_routings*, we can reformulate the problem and the corresponding constraints. The following is defined:

$$\text{Minimize } \boldsymbol{WC} + \boldsymbol{TC}$$

$$WC = \sum cWC_{kijp} X_{kijp}$$

$$TC = \sum cTC_{kijp} X_{kijp}$$

Where $cWC_{kijp} X_{kijp}$ is the warehousing cost for orders *k* at plant *p*, and $cTC_{kijp} X_{kijp}$ is the transportation cost for orders *k* from port *i* to port *j*.

Subject to the following constraints: serve all demand ($\sum_{i,j,p} X_{kijp} = 1$), limitation to the capacity of the plant ($\sum_{k,i,j} X_{kijp} \leq P_p$), and the weight capacity limit of each carriers ($\sum_{k,p} w_k X_{kijp} \leq \max(C_{ij})$).

The decision variable is the same as in the previous model, as the goal remains to minimize the total cost. The constraints ensure that the solution remains feasible, and is the component that is evolving within this second model. Based on the same variables as defined previously, the first constraint establishes that all orders must be served, the second limits the amount of product handled by a plant, and the third constraint imposes a restriction on the weight capacity of the carriers, using the same variables as defined previously.

This formulation allows for the building of the second LP model, using PuLP. Like for the previous model, a timer is initialized to track the time taken to solve the problem, and the objective is set to minimize the total cost. A list, *customer_plant*, is created to list all unique combinations. Then, the decision variable X is created. This decision variable represents whether or not a particular order

is shipped from an origin port to a destination port, while being supplied by a specific plant. If *X[order, orig_port, dest_port, plant] = 1*, the order is assigned to this route and plant. This refers in the problem as $X_{kijp} = 1$ meaning that the order *k*, from port *i* to port *j* and supplied at plant *p*, is shipped.

One of the constraints is that all orders must be processed. This constraint makes sure that every order is fulfilled exactly once. Each order must be assigned to exactly one route and one plant: the sum of X for all possible routes and plants must be equal to 1. This is done with a loop ensuring that for each *order_id* in *orders*, exactly one combination of origin port, destination port and plant is chosen, as the binary variable should be exactly equal to 1.

The second constraint, regarding the plant capacity limitation, makes sure that the total number of orders assigned to a plant does not exceed its capacity. In other words, the sum of all orders assigned to a specific plant must be less than or equal to the capacity of the plant. For each *plant* in *plants*, the code sums over all potential routings options, and checking if the *supply_plant* within the *possible_routes* matches *plant*. This constraint ensures that the total amount of these selected routes is less than, or equal to seven times the capacity of the plant. The multiplication by seven refers to the relaxing of the initial constraint.

The third and last constraint, regarding carrier capacity between ports, ensures that the total weight of orders does not exceed the capacity of the carrier. The sum of the weights of all orders shipped, multiplied by the binary variable X, should be less than or equal to the maximum carrier capacity. For each origin port to destination port, the loop sums the weight of all the routes. The constraint makes sure that the total transported weight between these two ports is less than, or equal to, 2 times the maximum capacity. For this constraint, 2 is the chosen multiplication to relax this constraint, to make this problem feasible. Both constraints have been relaxed for the feasibility of the problem, but one could adapt the multiplication factors to adapt the strictness of these new constraints.

WC represents the total warehousing cost, and TC the transportation costs. *prob.setObjective(WC + TC)* sets the objective function to minimize the sum of *WC* and *TC*. The problem is solved, the timer is ended, and the results are printed.

In a time of 3.27 seconds, the output indicates a successful solution to the problem (as the status is optimal), meaning the solver did find a feasible solution that minimizes the total costs while satisfying all constraints. The objective function, i.e. the total cost of fulfilling an order (including

both warehousing cost and transportation cost), is indicated to be at $17,074,339.45. This amount represents the cost of managing the entire supply chain, within the day, and under the established conditions. Overall, it reflects the financial impact of the current supply chain setup and route choices. This is what an optimized, cost-effective scenario would look like to fulfill all orders. The running and solution time indicates that the model is efficient and can be solved quickly in regards to the actual formulation and constraints. Overall, the results suggest that the optimal total cost for managing the supply chain is around $17 million, and that given the quick running time, the model could be scaled to larger datasets or a more complex supply chain structure. The model output can be implemented in the real-world supply chain, after focusing on the chosen routes and plant allocations.

However, it is crucial to see whether the run time was exceptionally low or high compared to the average running time, and if the result of the objective function could be another result. This would suggest the possibility of multiple optimal results. By running 10 iterations of the model, *run_times* is created, stocking all of the run times of the 10 iterations. Then, the average running time will be displayed, effectively displaying how low should the model take on average to run. This is done by dividing the sum of run times by the length of run time, or the number of total run times. Additionally, if average objective value is equal to the objective value after resolving the model once, it could be assumed that this is the only available option for the model. The iterations returns an average objective function of $17,074,339.45, indicating that the first iteration of the model returned the only possible. Additionally, the code returned an average running time of 3.10 seconds, meaning the first iteration of the model was slightly slower than the average running time, but only marginally.

The next part will display the result of the model, both graphically and in text format. The decision variable is processed, extracting the chosen routes and reorganizing them within the dictionary *final_routes*. This dictionary will store the final selected routes based on the optimal solution from the LP model, having origin port and destination port. A loop will then iterate over each decision variable. Regarding V, if it is greater than 0, the specific route was chosen, as the binary decision variable is equal to 1. However, if it is equal to 0, the route was not selected. Then, the loop checks if the route exists within *final_routes*. If the origin and destination ports do not exist within *final_routes*, a new entry is created for the route. If the route exists within *final_routes*, the code checks if the plant is listed or not under this exact route. If the plant does not exist, a new entry is created with the quantity of 1. If both the routes and the plant appear within *final_routes*, the loop will update the quantity. Finally, the *final_routes* is outputted, representing each entry within the

dictionary with the associated plant and quantity of goods. With this, a structured representation of all the selected routes from the LP model are displayed, providing information on which plants are supplying along which routes, and in what quantity. After this, a simple loop will print the name of values of every instance where the decision variable is equal to 1. By checking if *v.varValue* is not equal to 0, this loop will inspect and display the selected decision variables from the model, therefore each individual chosen routes and plants, with the variable names associated.

The results can also be displayed visually to get a better vision of the results within *final_routes*, as such:

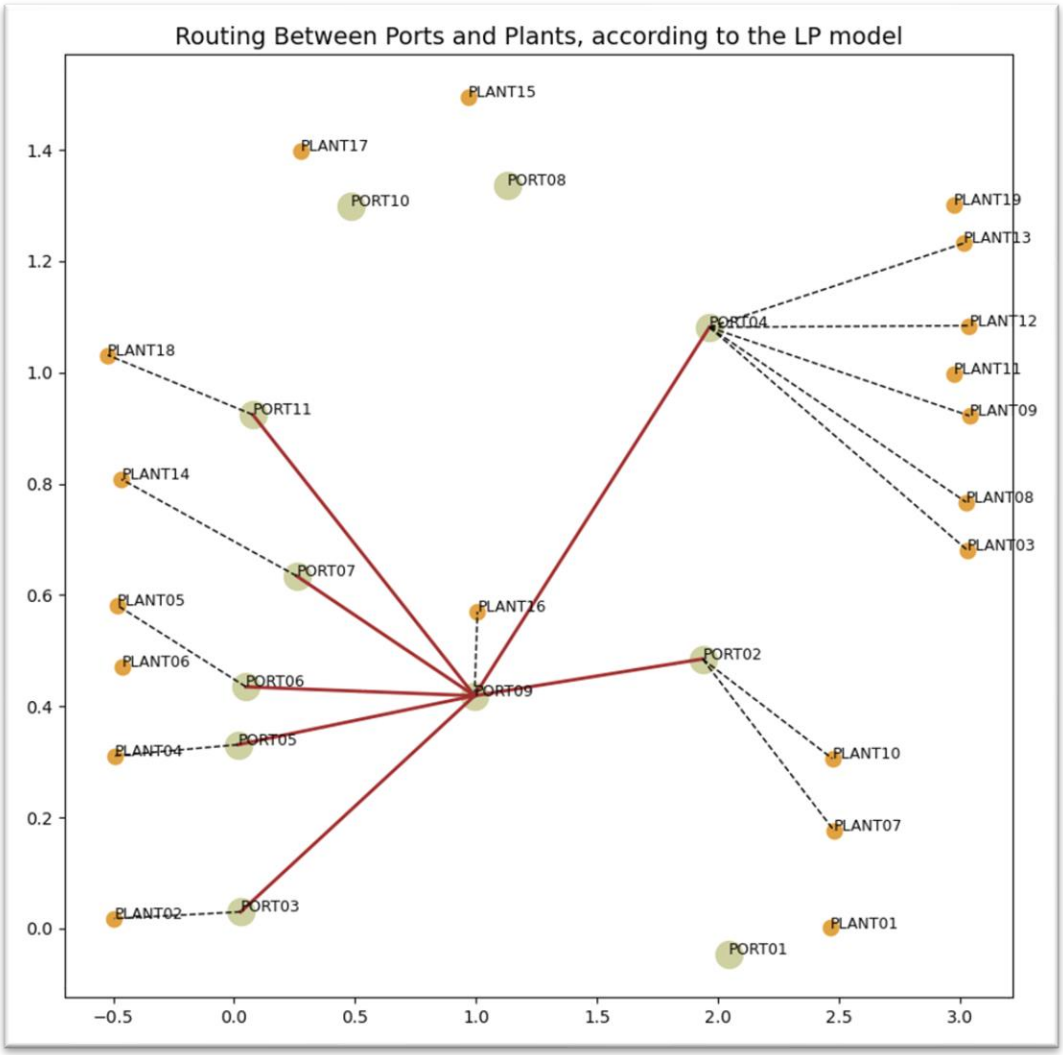


Figure 5: Routing between Ports and Plants, according to the LP model

The red lines represent the selected routes between origin ports and destination ports based on the solution, while the dashed lines represent the routing options from *port_plants_items*. When comparing this graph to the graph representing the routing options from the *PlantPorts* table, it

confirms that the solution elected routing options that all fall in the set of possible routing options from the problem. Points are a representation of the locations of ports and plants. This is achieved by looping over the routes in *final_routes*, plotting the routes between ports (and purposely making the line thicker for the elected connections) and the connections between ports and plants, displaying the ports and plants as scatter points and annotating the port names on the plot. Overall, this visualization allows for easy interpretation of the selected routes and plant locations, to better understand the output of the result.

Based on this visualization, the model identifies primary and secondary routing options selected by the optimization model. As expected *PORT09* is displayed as the main option of the supply chain process, serving multiple plants, like *PLANT16*, *PLANT06* and *PLANT05*. Some ports, like *PORT10* and *PORT08*, on the other hand, handle close to no traffic in this solution, indicating these ports may not be required for the current set of orders. In the optimized model, *PORT09* plays a crucial role in the network. While some underutilization of certain ports is observed, the model has returned an optimal solution, meaning that the overall routing plan is efficient as it is displayed. Increasing the volume of the dataset would probably allow to balance the use of each port, or each point overall, to increase the flexibility in the model. However, it would also increase the time of completion of the model, and the cost for a company to compute the model.

This model is a demonstration of how data science can be used to optimize supply chains for both financial and environmental performance. Applying the LP model, using the available data, the model has been able to minimize total costs, directly responsible for financial benefits. This has been done with concrete and actionable output, with a clear understanding of the routes to be chosen, having a minimized total cost of $17,074,339.45. From an environmental standpoint, this model has displayed the most efficient routes and the consolidation of the shipment. Using the routing from the LP model will consolidate shipments and reduce routing options, which would then lower the number of vehicle trips needed. The use of fewer, shorter routes, reducing the total kilometers traveled by freights, is directly linked to cutting fuel consumption and lower greenhouse gas emission. This reduces not only transportation cost, but also the fuel consumption and greenhouse gas emission. By reducing the unnecessary routing options, referring to the routing options that are not part of a non-optimal model, vehicle trips and carbon emissions are minimized. This approach aligns the establishment and implementation of the LP model with sustainability goals. With visualization of the results, data analytics help involved stakeholders take informed actions: the result output can either be used for further data manipulation (with the detailed text output), or used for visualization to represent the optimal routing options.

3.4. Linear Programming: Google OR-Tools and CBC

Using the same problem formulation and constraints, the model will be built under a different solver. The goal is to determine whether another model finds a different result for the objective function, and if not, if it finds the same result in a shorter amount of time. This is important to make sure that the time of completion of the model is not multiplied when the complexity of the model is increased. This model will be solved using Google OR-Tools, and the COIN-OR Branch-and-cut (CBC) solver, which is an open-source mixed-integer programming solver. OR-Tools is an open-source suite of optimization tools designed for large-scale, combinatorial optimization problems, like the one at stake here.

Instead of using *LpProblem* from PuLP, *pywraplp.Solver* is used. With this method change, *LpVariable()* becomes *BoolVar()*: this creates a binary decision variable for each possible route. Constraints are stocked into *solved.Add()*: the order assignment constraint to one route only is established, then the plant capacity constraint, and finally, the carrier capacity. The warehousing cost is defined as WC, the transportation cost as TC. With Google OR-Tools *solver.Minimize()* sets the objective function. By adapting the syntax of the previous model, the OR-Tools model will check if the solution is optimal and will output the result. This performance will be compared to the performance of the previous model. Once the OR-Tools solver (pywraplp.Solver) is executed using the CBC algorithm, the solution is found in less than 3 second, and the result matches the PuLP model's objective value of $17,074,339.45, confirming that both solvers yield the same optimal solution.

3.5 Comparing the results of Linear Programming models, and economic implications

3.5.1 Comparing PuLP and CBC models

The two models (PuLP and OR-Tools using CBC), returned the same results for the objective function: $17,074,339.45. With the given set of constraints, both models computed the constraints effectively and provided the same solution, which confirms that it is the optimal one. However, the OR-Tools model solved the problem slightly faster than the PuLP model and, even if the gain in terms of seconds is marginal here, this means that this model should be elected to solve this problem: as the problem size increases, this time performance disparity between the two models is likely to be increased. The OR-Tools model should be chosen for solving the problem, or similar problems with a larger dataset, as it has a better scalability potential due to faster time of completion. In a business context, this would have important implications, as the size of the data

would increase with each day of operations leading to a greater size of data made available. In this case, using the OR-Tools model with the CBC solver, companies would be able to solve complex optimization problems faster, like the one at stake here, faster.

3.5.2. Economic interpretation and limit of the result

The goal of the model was to reduce the total warehousing and transportation costs, to benefit companies financially by reducing the total operating costs, and to reduce their carbon footprint, by optimizing and reducing the number of routes taken. The various models have been achieving those goals with the given set of constraints, and the following part will evaluate how does the output of these models reflect the reality of the baseline operations. The baseline operations refers to the supply chain operation as they are, without the addition of the optimization process.

The first model returned that the problem is infeasible, meaning that with the given constraints and conditions, the model could not be solved. In more than 3 seconds, after relaxing the constraints and reformulating the model, the objective function is now equal to $17,074,339.45. A third model using OR-Tools found the same solution in less than 3 seconds.

For each *order_id* in the table orders, the code will retrieve all of the potential routes from *order_chances*, defined previously in the LP model. The first route is reelected and the order, origin port, destination port is extracted. With this set of information, the corresponding warehousing costs and transportation costs are retrieved from *possible_routes*. *baseline_warehousing_cost* and *baseline_transportation_cost* are defined based on this data, and refers to the costs when the supply chain process is not subject to the optimization model. For every route within the objective function X, the result of our optimization model, the code will check whether the route is within the optimized solution or not, i.e. when *var.varValue* == 1, or not. These results are accumulated in both *optimized_warehousing_costs* and *optimized_transportation_costs*, referring to the operating cost within the optimization model. From there, the *baseline_total_cost* and *optimized_total_cost* are calculated, by summing the respective transportation and warehousing cost. The cost saved with the optimization model is finally calculated, also indicating the share of cost saved with the model. A small tolerance for floating point precision between the objective function and the total optimized cost is specified in the code.

For the warehousing costs, the baseline warehousing cost is indicated to be $15,586,233.34 for operating on the specific day. The solution retrieved for the LP model returns a cost of $15,501,599.52. This indicates a potential saving of $84,633.82, which refers to 0.54%. For the

transportation costs, the baseline transportation cost is indicated to be $2,640,491.23 for operating on the specific day. The solution retrieved for the LP model returns a cost of $1,572,739.93. This indicates a potential saving of $1,067,751.30 (approximately 40%). For the total costs, the baseline total cost is indicated to be $18,226,724.57 for operating on the specific day. The solution retrieved for the LP model returns a cost of $17,074,339.45 (i.e. the objective function). This indicates a total potential cost saving up to $1,152,385.13 (approximately 6%). Seemingly, the most notable saving potential comes from the transportation cost, where the optimization model could lead to savings of over $2.1 million on operational costs. This result reveals that transportation is a key area of inefficiency within the supply chain, and could be solved with an optimized route planning. The significant cost cut potential, especially in transportation, demonstrates overall that the supply chain would operate more efficiently with an optimized supply chain model. Additionally, while not directly displayed in the results, this also means that reducing the transportation costs may likely also reduce the carbon footprint of the supply chain if a lower total distance is traveled and an optimized routing scheme is planned. From a business standpoint, this model manifests a direct impact on competitiveness. The advanced optimization techniques grant the concerned company major cost reduction and lower carbon emission.

Based on the results of the optimization, while significant cost savings could be achieved in both warehousing and transportation, these outcomes are tied to a static set of constraints and assumptions: the LP models have been applied to a linear dataset. This model could be a starting point to validate a model, but would increase in complexity as the size of the dataset and the associated constraints increase as well. However, variables such as capacity, production levels or cost per unit can evolve as the market dynamics, or companies make operational adjustments. A static approach relying on fixed constraints will display an optimized routing at a given point, but the models should evolve at the same pace as constraints. In a practical case, stakeholders should be able to implement evolution in the constraints (for example, an increase in productivity, marginal profits reducing the cost per unit), and all else being equal, being provided with a routing option taking into consideration the given changes. Additionally, the computing cost of the model should not be a problem in this case, as the dataset is not of a hefty size, and the model is ran only once. However, in the context of big data analytics, and if the model is computed at a daily or weekly frequency, it may be costly to actually compute the model. It is crucial to understand this to then consider whether it is cost-efficient to establish the model or not: if the cost of computing the model is higher than the actual benefit from displaying the optimal routing solution, the model should then be optimized further.

4. Conclusion

4.1. Final Considerations and findings

In the context of supply chain, the optimization of routing problems using data science could play a pivotal role for companies with an operational process, from both a financial and environmental aspect. The optimization of the company's transportations routing allows them to reduce their carbon footprint, by reducing their fuel consumption for example, by reducing the routing to only what is optimal, and cutting what is unnecessary in terms of transportation. By minimizing transportation costs, companies are able to reduce their overall expenditures. Reducing cost allows companies to make other investments, displaying financial health and environmental stewardship displays corporate efficiency. Therefore, businesses could benefit from understanding optimization models and investing in the technical knowledge linked to these models, especially with growing environmental constraints. Investing in optimization could be a solution to potential environmental constraints.

For this reason, a practical case is used to display how, from basic operational data, it is possible to limit the carbon footprint and transportation costs. A data analysis step before the optimization process is crucial for the operational team to be sure that they have to their hands clean and well-organized data that accurately reflects their supply chain activity. Without this, companies would be making decisions that do not directly reflect the reality of their business. With data analysis and data visualization, both the data direction team and non-technical teams from a company could grasp the key metrics used in decision-making. All teams can understand how the data they collect and manage is used in the optimization process, such as routing improvement in this case. The role of data visualization also tightens the gap between technical and non-technical teams, by displaying clearly the current state of the supply chain to individuals without a data background: this underlines the importance of using tools like PowerBI. Overall, data analysis and data visualization helps companies correct decision-making and in helping companies understand the data that is used in decision-making.

Several models have been developed, as a practical case, with the goal to optimize routing processes. The first Linear Programming model established that the model is infeasible as it is, in a time of 11 minutes and 24 seconds. This conclusion in a business context allows us to understand that the constraints and the data need to be revised to solve the problem. For this reason, a second model should be built, redefining the problem and the constraints to find a feasible solution to the problem. After adapting the constraints and making the model evolve, in 5 seconds, the model

returned a feasible solution, with an objective value of $17,074,339.45, with the corresponding routing displayed. This model has been challenged by another solver to understand if using another solution impacts both the objective function and the time of completion. These models provided actionable insights that could drive operational improvement. The implementation of the model would lead to a saving of $1,152,385.13 (6% of total operational costs). The model revealed that transportation accounts for the most cost cut potential when comparing the baseline routing and the optimized routing.

4.2. Limitations and further research

In this case of routing optimization, an LP model has been built, using PuLP, then OR-Tools. While effective, limiting the analysis on these solvers represents one approach, amongst many other available. The use of linear constraints simplifies the problem, allowing solutions to be found in less than a second. However it also means that the model will not take into consideration more complex factors. Other solver options, such as Gurobi or CPLEX, could manage non-linear constraints, larger datasets and more intricate variables. With additional data, fluctuation in demand could have been analyzed, but also variable transportation costs, and mixed-integer solvers can deal with those nonlinear constraints. Building these could have provided a scalable model for the problem, and not only a static one.

Optimizing routing problems will both reduce transportation costs and carbon footprint, with insights accessible to both technical and non-technical teams, with the use of data science. The models implemented offer a result that could be applied to a business context, with the evaluated results. However, these models, and the principle of using a model can be limited. The built models assume linear demand and transportation cost, without acknowledging external complexity within the supply chain process. These models have been built on data that is assumed to be reliable and accurate: while data can be cleaned, and its accuracy can be confirmed or refuted, wrong inputs in the data can lead to discrepancies that would lead to stakeholders making the wrong decisions. Additionally, this could be responsible for leading to a Linear Programming model to be infeasible. Although these models are a powerful tool for optimization, it remains a simplification of reality. As such, it has intrinsic limitations, not least the fact that it cannot be directly applied to all real-life business situations without specific adjustments. The underlying assumptions and simplifications required for modeling must always be taken into account when interpreting results and applying them in practical contexts.

Going beyond these limitations requires companies to always refine their data collection, to make it as representative of reality as possible. Additionally, they should go beyond by exploring other optimization techniques, based on the data available. Heuristic approaches could be considered for more flexibility and adaptation and machine learning (ML) could allow the implementation of predictive analytics. Nevertheless, the effectiveness of these models would heavily depend on the part of data in the company's cultures. The optimization process may lose its value without a commitment to cultivating data-driven practices. This implies firstly that stakeholders should input accurate and representative data, making sure that the inputs used in the models reflect the reality of the company's operations. There should also be an understanding across the organization that model results are simplifications, and while these models provide details on effective cost reduction strategies to implement, their outputs are bound by the assumptions and constraints under which they operate. A strong data culture is crucial for both interpreting the insights from LP models, and by extension from other models, but also for acting effectively and with good data-driven practices in mind, upon these insights.

5. Appendix

APPENDIX B: TITLE PAGE

GRADUATING PROJECT

Submitted in fulfillment of requirements for completion of MCs Data and Business Analytics

OLEJNICZAK Tom

(Final submission date)

Graduating Project supervisor

(SADOGHI Amirhossein)

APPENDIX C: Declaration of Academic Integrity and Personal Work (to be submitted with the GP submission)

Declaration of Academic Integrity and Personal Work

"I, the undersigned, OLEJNICZAK Tom, declare on my honor that this submitted Graduating Project "How can advanced analytics and data science be applied to optimize sustainable supply chains for enhanced environmental and financial performance?", is my own work. No part of this research has been submitted in the past for publication or for degree purposes.

I also confirm that,

☐ I did not use generative AI

■ I used generative AI for (in details):

Use of AI to assist with code troubleshooting and debugging, and for refining the optimization models built.

I am aware of and understand Rennes SB ACADEMIC INTEGRITY POLICY on Academic Dishonesty and its consequences. I am fully responsible for the truthfulness of this declaration.

Date: 27/10/2024,

In Bordeaux

Signature:

APPENDIX D: CONFIDENTIALITY AGREEMENT

Confidentiality Agreement Of Graduating Project

Family name: OLEJNICZAK

First name: Tom

Program attended at RENNES School of Business: MCs Data and Business Analytics

TITLE OF THE GRADUATING PROJECT**:**

"How can advanced analytics and data science be applied to optimize sustainable supply chains for enhanced environmental and financial performance?"

Date of submission: 27/10/2024

Appendices:..... YES ☐ NO ☐

Confidentiality: YES .....☐ duration: .....

No .....☐

Comments:.....

.....

.....

....

.....

....

.....

....

https://github.com/tomolejniczak/GP/blob/main/gp_dashboard.pbix

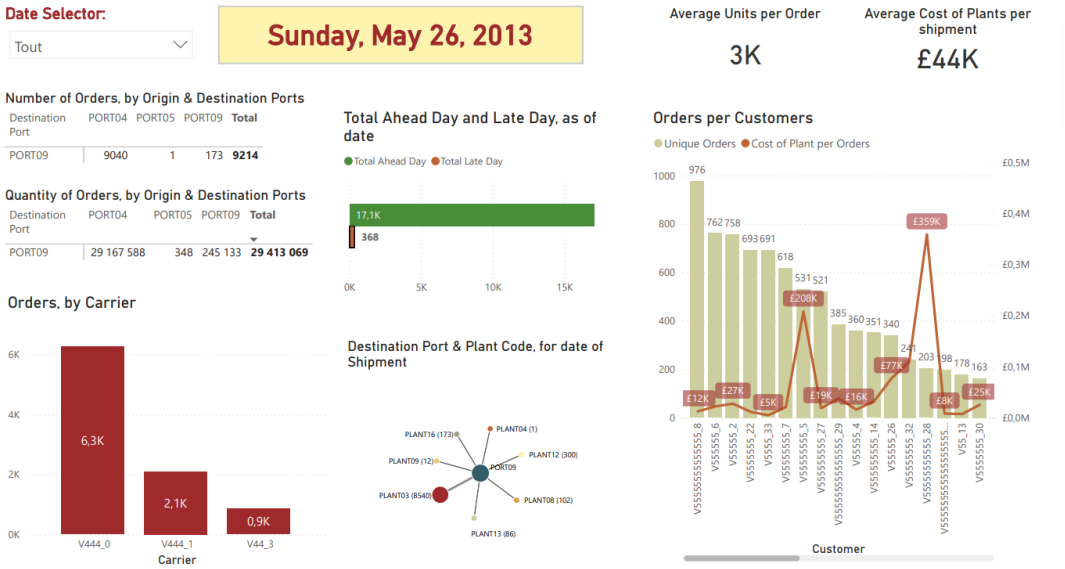


Figure 3:Power BI: operational data dashboard

https://github.com/tomolejniczak/GP/blob/main/GP%20-%20OLEJNICZAK%20Tom.ipynb

[30]:
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import numpy as np
import seaborn as sns
import plotly.graph_objs as go
from tqdm import tqdm
from mpl_toolkits.mplot3d import Axes3D
import time
import plotly.graph_objects as go
from pulp import LpMinimize, LpProblem, LpVariable, lpSum, LpInteger, LpStatus,␣
 ↪value, CPLEX_CMD
import networkx as nx
from ortools.constraint_solver import pywrapcp, routing_enums_pb2, pywraplp
import itertools


colors = ['#9E2A2B', '#bf6535', '#e09f3e', '#f0c977', '#fff3b0', '#ccce9e',␣
 ↪'#99a88c', '#335c67']
```

[4]:
```
file = pd.ExcelFile(r'C:\Users\tomol\OneDrive\Desktop\Supply chain logisitcs␣
 ↪problem.xlsx')
df_dict = {}
for names in file.sheet_names:
    globals()[names] = file.parse(names)
    df_dict[names] = globals()[names]
print(df_dict.keys())
```

```
dict_keys(['OrderList', 'FreightRates', 'WhCosts', 'WhCapacities',
'ProductsPerPlant', 'VmiCustomers', 'PlantPorts'])
```

1 1. Data description

We will start by understanding what is within the excel file and what information are in the various sheets.

[5]:
```
def description(df_name, df):
    print(f"{df_name}:\n")
    print(f"Rows: {df.shape[0]}")
    print(f"Columns: {df.shape[1]}")
    print("Column names:")
    print(df.columns.tolist())
    print("\n" + "-"*80 + "\n")
for df_name, df in df_dict.items():
    description(df_name, df)
```

```
OrderList:

Rows: 9215
Columns: 14
Column names:
['Order ID', 'Order Date', 'Origin Port', 'Carrier', 'TPT', 'Service Level',
'Ship ahead day count', 'Ship Late Day count', 'Customer', 'Product ID', 'Plant
Code', 'Destination Port', 'Unit quantity', 'Weight']

--------------------------------------------------------------------------------

FreightRates:

Rows: 1540
Columns: 11
Column names:
['Carrier', 'orig_port_cd', 'dest_port_cd', 'minm_wgh_qty', 'max_wgh_qty',
'svc_cd', 'minimum cost', 'rate', 'mode_dsc', 'tpt_day_cnt', 'Carrier type']

--------------------------------------------------------------------------------

WhCosts:

Rows: 19
Columns: 2
Column names:
['WH', 'Cost/unit']

--------------------------------------------------------------------------------

WhCapacities:

Rows: 19
Columns: 2
Column names:
['Plant ID', 'Daily Capacity ']
```

```
--------------------------------------------------------------------------------

ProductsPerPlant:

Rows: 2036
Columns: 2
Column names:
['Plant Code', 'Product ID']

--------------------------------------------------------------------------------

VmiCustomers:

Rows: 14
Columns: 2
Column names:
['Plant Code', 'Customers']

--------------------------------------------------------------------------------

PlantPorts:

Rows: 22
Columns: 2
Column names:
['Plant Code', 'Port']

--------------------------------------------------------------------------------
```

1.0.1 Statistical Description

[6]:
```
print("Statistical Description for OrderList:\n")
print(OrderList.describe())
print("\n" + "-"*80 + "\n")
```

```
Statistical Description for OrderList:

           Order ID           Order Date          TPT  Ship ahead day count  \
count  9.215000e+03                 9215  9215.000000           9215.000000
mean   1.447274e+09  2013-05-26 00:00:00     1.717743              1.852306
min    1.447126e+09  2013-05-26 00:00:00     0.000000              0.000000
25%    1.447197e+09  2013-05-26 00:00:00     1.000000              0.000000
50%    1.447276e+09  2013-05-26 00:00:00     2.000000              3.000000
75%    1.447346e+09  2013-05-26 00:00:00     2.000000              3.000000
max    1.447425e+09  2013-05-26 00:00:00     4.000000              6.000000
std    8.381629e+04                  NaN     0.630500              1.922302
```

```
       Ship Late Day count    Product ID  Unit quantity       Weight
count          9215.000000  9.215000e+03    9215.000000  9215.000000
mean              0.039935  1.680536e+06    3202.747151    19.871688
min               0.000000  1.613321e+06     235.000000     0.000000
25%               0.000000  1.669702e+06     330.000000     1.407430
50%               0.000000  1.683636e+06     477.000000     4.440000
75%               0.000000  1.689554e+06    1275.500000    13.325673
max               6.000000  1.702654e+06  561847.000000  2338.405126
std               0.319625  1.526593e+04   15965.622260    66.569064
```

Check for Duplicates & Null Values

[7]:
```python
def check_duplicate_values(df_name, df):
    duplicate_count = df.duplicated().sum()
    if duplicate_count > 0:
        print(f"Duplicate Values in {df_name}: {duplicate_count}")
    print("\n" + "-"*80 + "\n")

def check_missing_values(df_name, df):
    print(f"Missing Values in {df_name}:\n")
    print(df.isnull().sum())
    print("\n" + "-"*80 + "\n")

for df_name, df in df_dict.items():
    check_missing_values(df_name, df)
    check_duplicate_values(df_name, df)
```

```
Missing Values in OrderList:

Order ID               0
Order Date             0
Origin Port            0
Carrier                0
TPT                    0
Service Level          0
Ship ahead day count   0
Ship Late Day count    0
Customer               0
Product ID             0
Plant Code             0
Destination Port       0
Unit quantity          0
Weight                 0
dtype: int64
```

```
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------

Missing Values in FreightRates:

Carrier         0
orig_port_cd    0
dest_port_cd    0
minm_wgh_qty    0
max_wgh_qty     0
svc_cd          0
minimum cost    0
rate            0
mode_dsc        0
tpt_day_cnt     0
Carrier type    0
dtype: int64

--------------------------------------------------------------------------------

Duplicate Values in FreightRates: 3

--------------------------------------------------------------------------------

Missing Values in WhCosts:

WH           0
Cost/unit    0
dtype: int64

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------

Missing Values in WhCapacities:

Plant ID           0
Daily Capacity     0
dtype: int64

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
```

```
Missing Values in ProductsPerPlant:

Plant Code    0
Product ID    0
dtype: int64

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------

Missing Values in VmiCustomers:

Plant Code    0
Customers     0
dtype: int64

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------

Missing Values in PlantPorts:

Plant Code    0
Port          0
dtype: int64

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
```

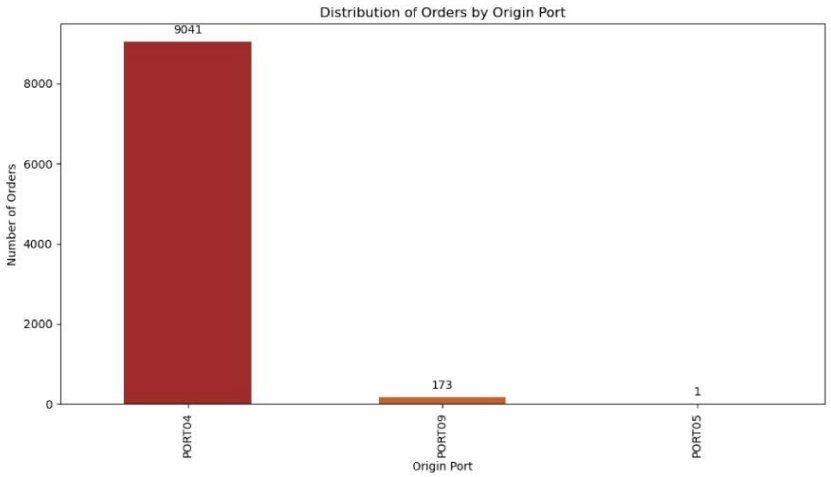
[8]:
```
for df_name, df in df_dict.items():
    df.columns = [col.strip().replace(' ', '_').replace('/', '_').lower() for␣
 ↪col in df.columns]
```

2 2. Data Visualisation

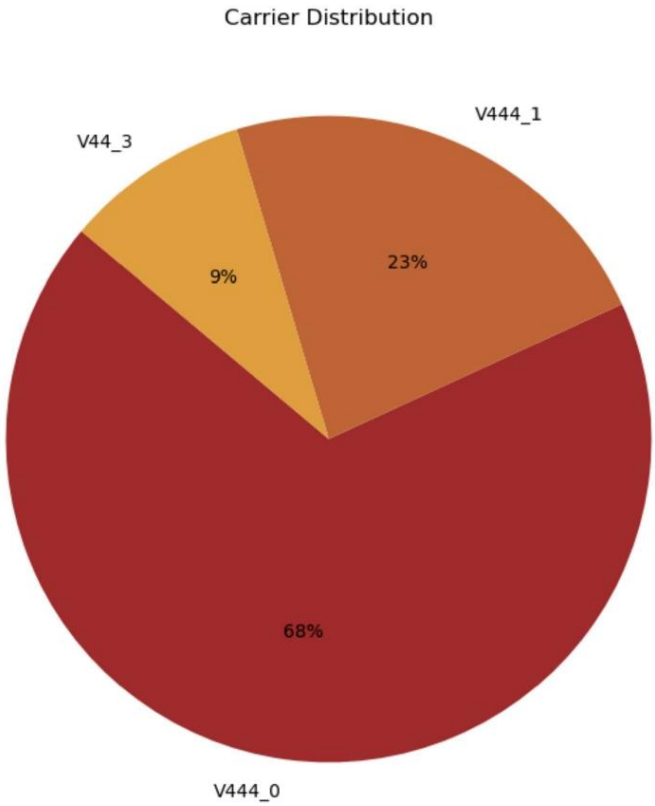
[9]:
```
OrderList['order_date'] = pd.to_datetime(OrderList['order_date'])
unique_date = OrderList['order_date'].unique()
if len(unique_date) == 1:
    print(f"We will be working with data from a specific day: {unique_date[0]}")
else:
    print(f"We will be working with data across multiple days: {unique_date}")
```

```
We will be working with data from a specific day: 2013-05-26 00:00:00
```

[10]:
```
plt.figure(figsize=(12, 6))
ax = OrderList['origin_port'].value_counts().plot(kind='bar', color=colors)
plt.title('Distribution of Orders by Origin Port')
plt.xlabel('Origin Port')
plt.ylabel('Number of Orders')
plt.xticks(rotation=90)
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 10),
                textcoords='offset points')
plt.show()
```



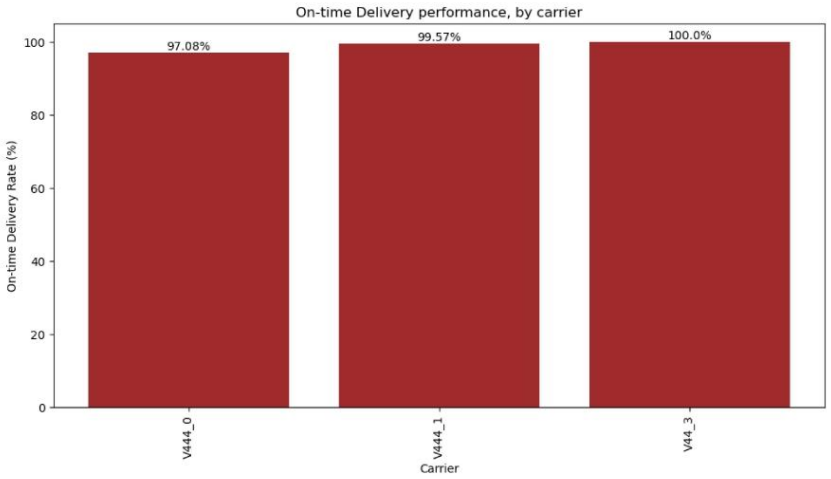
[11]:
```
plt.figure(figsize=(8, 8))
OrderList['carrier'].value_counts().plot(
    kind='pie',
    autopct=lambda p: f'{p:.0f}%',
    startangle=140,
    colors=colors)
plt.title('Carrier Distribution')
plt.ylabel('')
plt.show()
```



[12]:
```
OrderList['on_time_delivery'] = OrderList['ship_late_day_count'] <= 0
carrier_performance = OrderList.groupby('carrier')['on_time_delivery'].mean().
 ↪reset_index()
carrier_performance['on_time_delivery'] =␣
 ↪carrier_performance['on_time_delivery'] * 100
plt.figure(figsize=(12, 6))
bars = plt.bar(carrier_performance['carrier'],␣
 ↪carrier_performance['on_time_delivery'], color=colors[0])
plt.title('On-time Delivery performance, by carrier')
plt.xlabel('Carrier')
plt.ylabel('On-time Delivery Rate (%)')
plt.xticks(rotation=90)
```

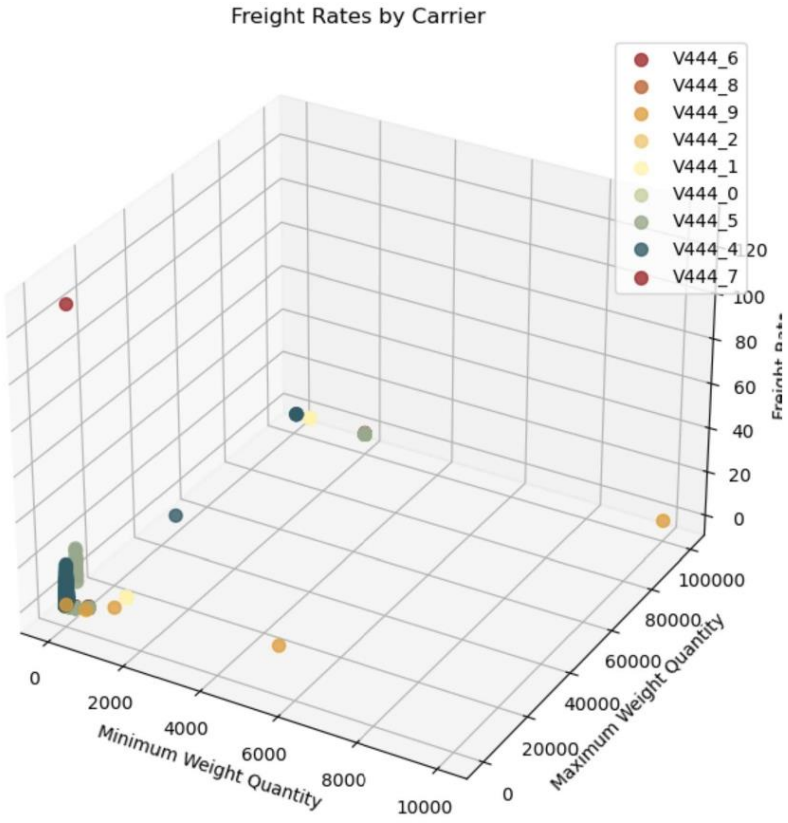
8

```
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{round(yval, 2)}%',␣
 ↪ha='center', va='bottom')
plt.show()
```



[13]:
```
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

carriers = FreightRates['carrier'].unique()
for i, carrier in enumerate(carriers):
    carrier_data = FreightRates[FreightRates['carrier'] == carrier]
    ax.scatter(
        carrier_data['minm_wgh_qty'],
        carrier_data['max_wgh_qty'],
        carrier_data['rate'],
        color=colors[i % len(colors)],
        s=50,
        alpha=0.8,
        label=carrier)
ax.set_title('Freight Rates by Carrier')
ax.set_xlabel('Minimum Weight Quantity')
ax.set_ylabel('Maximum Weight Quantity')
ax.set_zlabel('Freight Rate')

ax.legend()
```

```
plt.show()
```

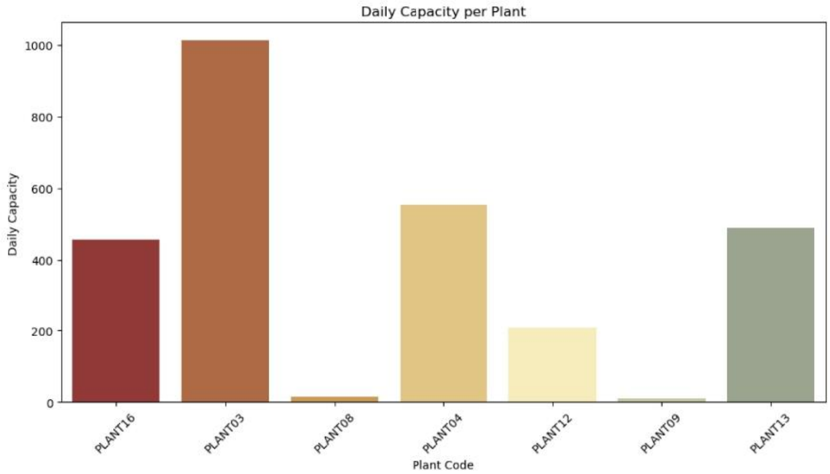


NOTE: On jupyter notebook, there is no option to zoom into the graph. However, on any other IDE, this would be practical to see clearly the rate of each min. and max. weight quantity associated to each carrier.

[14]:
```
order_capacity = OrderList.merge(WhCapacities, left_on='plant_code',␣
 ↪right_on='plant_id', how='left')
daily_capacity_per_plant = order_capacity[['plant_code', 'daily_capacity']].
 ↪drop_duplicates()

plt.figure(figsize=(12, 6))
```

```
sns.barplot(x='plant_code', y='daily_capacity', data=daily_capacity_per_plant,␣
 ↪palette=colors)
plt.title('Daily Capacity per Plant')
plt.xlabel('Plant Code')
plt.ylabel('Daily Capacity')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\tomol\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\tomol\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\tomol\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Merging *OrderList* with *FreightRates* and *df_WhCosts*, and calculating the Total Cost

[15]:
```
OrderList = df_dict['OrderList']
```

11

```
OrderList = OrderList.merge(df_dict['FreightRates'], left_on=['carrier',␣
 ↪'origin_port', 'destination_port'],
                            right_on=['carrier', 'orig_port_cd',␣
 ↪'dest_port_cd'], how='left')
OrderList = OrderList.merge(df_dict['WhCosts'], left_on='plant_code',␣
 ↪right_on='wh', how='left')

OrderList['cost'] = (OrderList['unit_quantity'] * OrderList['rate']) +␣
 ↪(OrderList['unit_quantity'] * OrderList['cost_unit'])

OrderList = OrderList.dropna()

ProductsPerPlant = df_dict['ProductsPerPlant']
ProductsPerPlant = ProductsPerPlant.dropna()

df_dict['OrderList'] = OrderList
df_dict['ProductsPerPlant'] = ProductsPerPlant

OrderList
```

[15]:

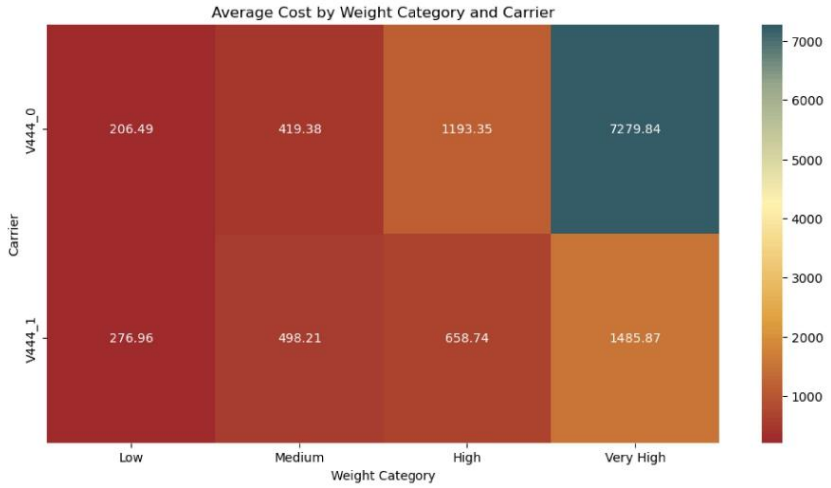
	order_id	order_date	origin_port	carrier	tpt	service_level	\
854	1.447385e+09	2013-05-26	PORT09	V444_0	0	DTP	
855	1.447385e+09	2013-05-26	PORT09	V444_0	0	DTP	
856	1.447338e+09	2013-05-26	PORT09	V444_0	0	DTP	
857	1.447338e+09	2013-05-26	PORT09	V444_0	0	DTP	
858	1.447407e+09	2013-05-26	PORT09	V444_0	0	DTP	
…	…	…	…	… …	…		
209397	1.447328e+09	2013-05-26	PORT04	V444_1	1	DTD	
209398	1.447328e+09	2013-05-26	PORT04	V444_1	1	DTD	
209399	1.447328e+09	2013-05-26	PORT04	V444_1	1	DTD	
209400	1.447328e+09	2013-05-26	PORT04	V444_1	1	DTD	
209401	1.447328e+09	2013-05-26	PORT04	V444_1	1	DTD	

	ship_ahead_day_count	ship_late_day_count	customer	\
854	0	0	V55555_4	
855	0	0	V55555_4	
856	0	0	V55555_4	
857	0	0	V55555_4	
858	3	0	V555555_6	
…	…	…	…	
209397	5	0	V55555555555555_8	
209398	5	0	V55555555555555_8	
209399	5	0	V55555555555555_8	
209400	5	0	V55555555555555_8	
209401	5	0	V55555555555555_8	

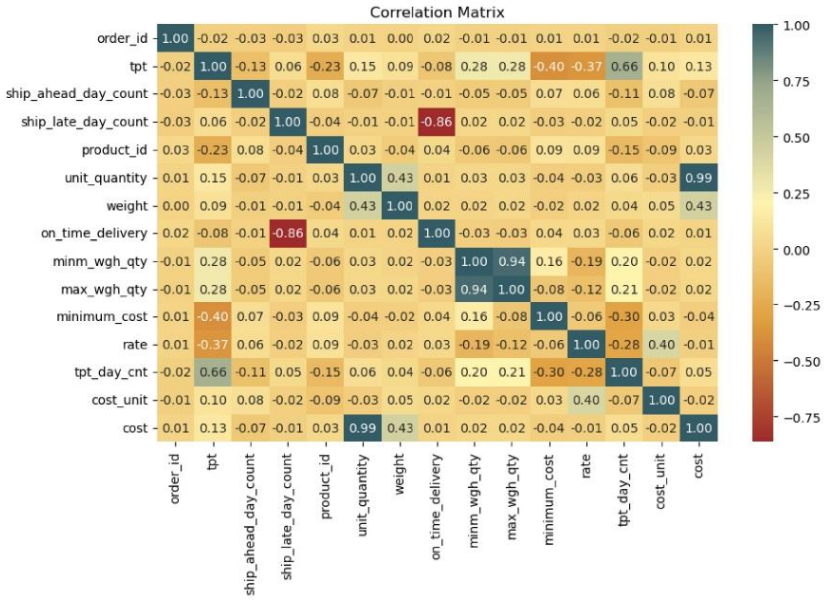
	product_id	…	max_wgh_qty	svc_cd	minimum_cost	rate	mode_dsc	\

```
854        1692724  …     5000.00    DTP       31.2784  12.2784    GROUND
855        1692724  …     5000.00    DTD       31.2784  13.2784    GROUND
856        1692724  …     5000.00    DTP       31.2784  12.2784    GROUND
857        1692724  …     5000.00    DTD       31.2784  13.2784    GROUND
858        1692722  …     5000.00    DTP       31.2784  12.2784    GROUND
…              …  …           …     …      …        …        …
209397     1683424  …       99.99    DTD        7.8044   0.0804    AIR
209398     1683424  …      299.99    DTD       11.2272   0.0792    AIR
209399     1683424  …      499.99    DTD       31.2672   0.0780    AIR
209400     1683424  …      999.99    DTD       49.4272   0.0760    AIR
209401     1683424  …     9999.99    DTD       95.2272   0.0744    AIR

       tpt_day_cnt  carrier_type       wh  cost_unit         cost
854            0.0   V88888888_0  PLANT16   1.919808  4713.804893
855            0.0   V88888888_0  PLANT16   1.919808  5045.804893
856            0.0   V88888888_0  PLANT16   1.919808  5082.958288
857            0.0   V88888888_0  PLANT16   1.919808  5440.958288
858            0.0   V88888888_0  PLANT16   1.919808  5807.066871
…              …           …       …         …          …
209397         1.0  V888888883_1  PLANT03   0.517502   189.534900
209398         1.0  V888888883_1  PLANT03   0.517502   189.154500
209399         1.0  V888888883_1  PLANT03   0.517502   188.774100
209400         1.0  V888888883_1  PLANT03   0.517502   188.140100
209401         1.0  V888888883_1  PLANT03   0.517502   187.632900

[208548 rows x 28 columns]
```

[16]:
```
cmap = LinearSegmentedColormap.from_list("custom_cmap", colors, N=100)

OrderList['weight_category'] = pd.qcut(OrderList['weight'], q=4, labels=['Low',␣
 ↪'Medium', 'High', 'Very High'])

heatmap_data = OrderList.pivot_table(index='carrier',␣
 ↪columns='weight_category', values='cost', aggfunc='mean')

plt.figure(figsize=(12, 6))
sns.heatmap(heatmap_data, annot=True, fmt=".2f", cmap=cmap)
plt.title('Average Cost by Weight Category and Carrier')
plt.xlabel('Weight Category')
plt.ylabel('Carrier')
plt.show()
```



[17]:
```
custom_cmap = LinearSegmentedColormap.from_list("custom_cmap", colors)
correlation_matrix = df_dict['OrderList'].corr(numeric_only=True)
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=correlation_matrix.round(2),␣
↪cmap=custom_cmap, fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



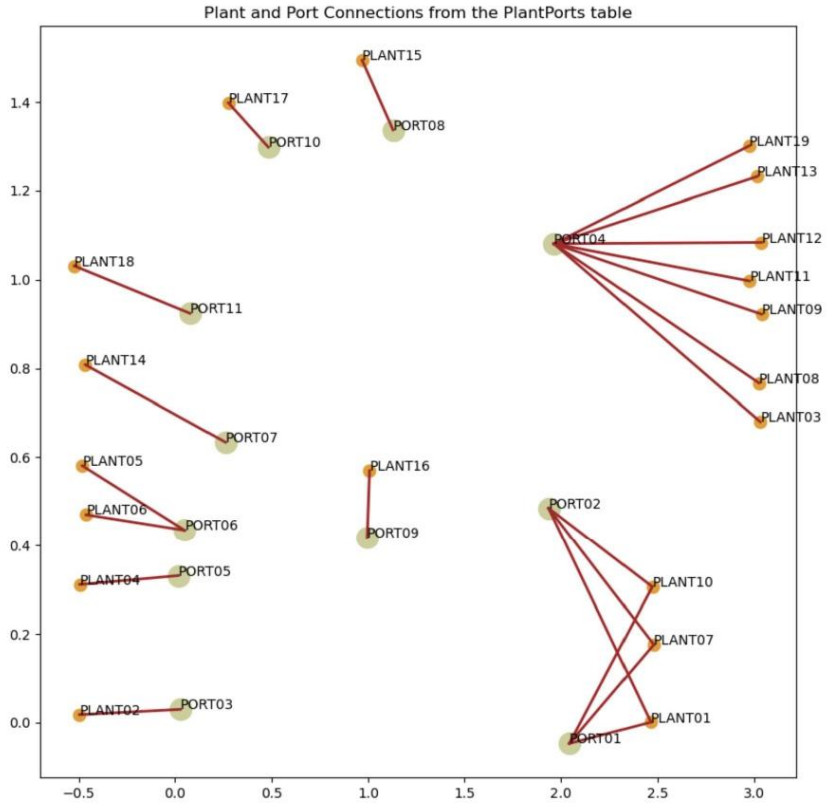
3 Displaying Network

Available connection from our data

[18]:
```
PortsPos = {
    'PORT01': (2.045, -0.047),
    'PORT02': (1.937, 0.485),
    'PORT03': (0.027, 0.03),
    'PORT04': (1.965, 1.081),
    'PORT05': (0.016, 0.331),
    'PORT06': (0.049, 0.435),
    'PORT07': (0.259, 0.634),
    'PORT08': (1.128, 1.337),
    'PORT09': (0.994, 0.419),
    'PORT10': (0.481, 1.298),
    'PORT11': (0.077, 0.924)
}

PlantsPos = {
    'PLANT01': (2.466, 0.001),
    'PLANT02': (-0.496, 0.018),
    'PLANT03': (3.033, 0.68),
```

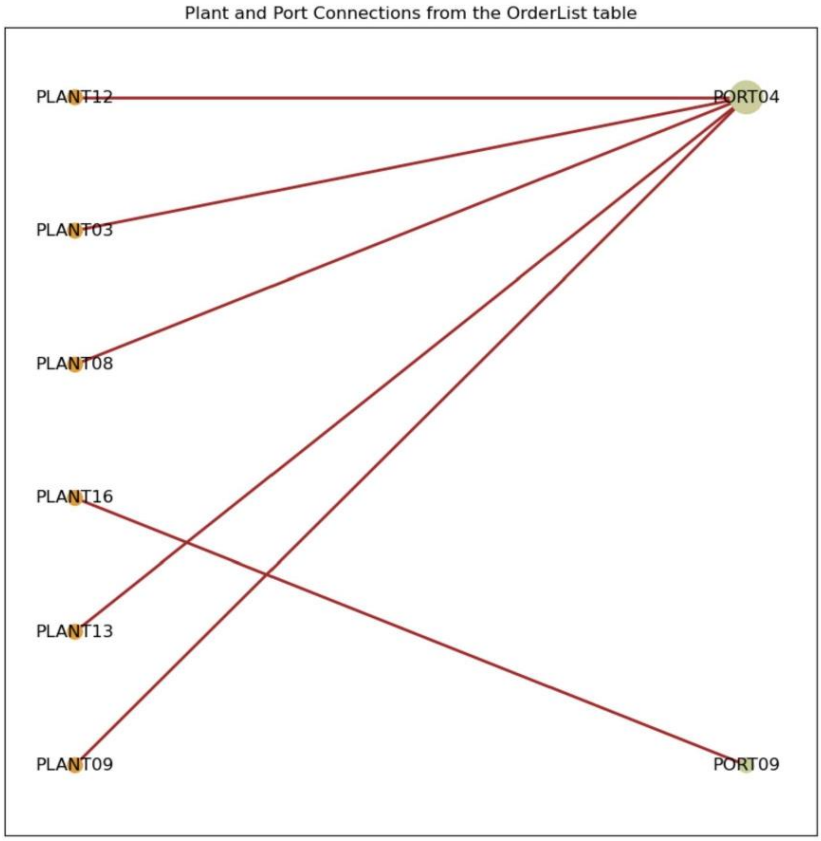
```
    'PLANT04': (-0.495, 0.311),
    'PLANT05': (-0.482, 0.581),
    'PLANT06': (-0.462, 0.47),
    'PLANT07': (2.481, 0.176),
    'PLANT08': (3.026, 0.767),
    'PLANT09': (3.04, 0.922),
    'PLANT10': (2.473, 0.306),
    'PLANT11': (2.977, 0.997),
    'PLANT12': (3.039, 1.084),
    'PLANT13': (3.017, 1.232),
    'PLANT14': (-0.466, 0.808),
    'PLANT15': (0.968, 1.494),
    'PLANT16': (1.006, 0.57),
    'PLANT17': (0.274, 1.398),
    'PLANT18': (-0.523, 1.031),
    'PLANT19': (2.976, 1.301)
}

plants = PlantPorts['plant_code'].unique()
ports = PlantPorts['port'].unique()

port_plants = {port: PlantPorts[PlantPorts['port'] == port]['plant_code'].
 ↪tolist() for port in ports}

line_color = colors[0]
port_dot_color = colors[5]
plant_dot_color = colors[2]

plt.figure(figsize=(10, 10))

for port in ports:
    if len(port_plants[port]) > 0:
        for plant in port_plants[port]:
            plt.plot([PortsPos[port][0], PlantsPos[plant][0]],
                     [PortsPos[port][1], PlantsPos[plant][1]],
                     linewidth=2, c=line_color)

plt.scatter([pos[0] for _, pos in PortsPos.items()],
            [pos[1] for _, pos in PortsPos.items()],
            s=250, c=port_dot_color)

for port in ports:
    plt.annotate(port, PortsPos[port], fontsize=10)

plt.scatter([pos[0] for _, pos in PlantsPos.items()],
            [pos[1] for _, pos in PlantsPos.items()],
            s=80, c=plant_dot_color)
```

```
for plant in plants:
    plt.annotate(plant, PlantsPos[plant], fontsize=10)

plt.title('Plant and Port Connections from the PlantPorts table')
plt.show()
```



[19]:
```
unique_routes = OrderList[['plant_code', 'origin_port']].drop_duplicates()
plant_ports_graph = nx.Graph()

plants = unique_routes['plant_code'].unique()
ports = unique_routes['origin_port'].unique()
```

```
plant_ports_graph.add_nodes_from(plants, bipartite=0)
plant_ports_graph.add_nodes_from(ports, bipartite=1)

for _, row in unique_routes.iterrows():
    plant_ports_graph.add_edge(row['plant_code'], row['origin_port'])

layout = nx.bipartite_layout(plant_ports_graph, plants)

for node in layout:
    if node in plants:
        layout[node][0] -= 0.1
    else:
        layout[node][0] += 0.1

degrees = dict(plant_ports_graph.degree)

plant_sizes = [degrees[plant] * 100 for plant in plants]
port_sizes = [degrees[port] * 100 for port in ports]

line_color = colors[0]
port_dot_color = colors[5]
plant_dot_color = colors[2]

plt.figure(figsize=(10, 10))

nx.draw_networkx_edges(plant_ports_graph, layout, edge_color=line_color,␣
 ↪width=2)

nx.draw_networkx_nodes(plant_ports_graph, layout, nodelist=ports,␣
 ↪node_size=port_sizes, node_color=port_dot_color)

nx.draw_networkx_nodes(plant_ports_graph, layout, nodelist=plants,␣
 ↪node_size=plant_sizes, node_color=plant_dot_color)

nx.draw_networkx_labels(plant_ports_graph, layout)

plt.title('Plant and Port Connections from the OrderList table')
plt.show()
```



[20]:
```
unique_order_routes = OrderList[['plant_code', 'origin_port']].drop_duplicates()
print(f"Number of unique routing options in OrderList: {unique_order_routes.
  ↪shape[0]}")
unique_plantport_routes = PlantPorts[['plant_code', 'port']].drop_duplicates()
print(f"Number of unique routing options in PlantPorts:␣
  ↪{unique_plantport_routes.shape[0]}")
```

```
Number of unique routing options in OrderList: 6
Number of unique routing options in PlantPorts: 22
```

4 II. Optimization

4.1 1. Constraints

Objective Function The objective function to minimize can be displayed as the following:

$$Minimize : \sum (WC_{ki} + TC_{kpi})$$

Having:

$$WC = \sum_{kijp} c^{WC}_{kijp} X_{kijp}$$
$$TC = \sum_{kijp} c^{TC}_{kijp} X_{kijp}$$

Constraints

From the WhCapacity table:

$$\sum O_{ki} \leq C_i$$

where:

- $ O_{ki} $ is a Boolean variable equal to **1** if order $ k $ is shipped from warehouse $ i $ and **0** if not.
- $ C_i $ is the maximum number of orders that warehouse $ i $ can handle per day.

From the FreightRates table:

$$\sum w_{kpjcstm} \leq \max(F_{pjcstm})$$

where:

- $ w_{kpjcstm} $ is the weight in kilograms for order $ k $
- $ F_{pjcstm} $ is the maximum weight limit for shipments from port $ p $ to customer port $ j $ via courier $ c $, using service level $ s $, delivery time $ t $, and transportation mode $ m $.

No self-routing constraint:

4.2 2. Linear Programming (LP): CPLEX_CMD

[137]:
```
from pulp import LpMinimize, LpProblem, LpVariable, lpSum, LpInteger, value,␣
↪CPLEX_CMD
import time
```

```
start_time = time.time()

supply_nodes = list(df_dict['WhCosts']['wh'])

supply_dict = {}
for node in supply_nodes:
    total_capacity =␣
 ↪sum(df_dict['WhCapacities']['daily_capacity'][df_dict['WhCapacities']['plant_id']␣
 ↪== node])
    supply_dict[node] = total_capacity

demand_nodes = list(df_dict['OrderList']['destination_port'].unique())

demand_dict = {}
for index, row in df_dict['OrderList'].iterrows():
    dest_port = row['destination_port']
    unit_quantity = row['unit_quantity']
    if dest_port in demand_dict:
        demand_dict[dest_port] += unit_quantity
    else:
        demand_dict[dest_port] = unit_quantity

costs = []
for index, row in df_dict['OrderList'].iterrows():
    carrier = row['carrier']
    orig_port = row['orig_port_cd']
    dest_port = row['dest_port_cd']
    weight = row['weight']

    matching_rates =␣
 ↪df_dict['FreightRates'][(df_dict['FreightRates']['carrier'] == carrier) &
                                (df_dict['FreightRates']['orig_port_cd'] ==␣
 ↪orig_port) &
                                (df_dict['FreightRates']['dest_port_cd'] ==␣
 ↪dest_port) &
                                (df_dict['FreightRates']['minm_wgh_qty'] <=␣
 ↪weight) &
                                (df_dict['FreightRates']['max_wgh_qty'] >=␣
 ↪weight)]['rate']

    rate = matching_rates.values[0] if not matching_rates.empty else None
    costs.append(rate)

cost_dict = {}
for supply_node in supply_nodes:
    cost_dict[supply_node] = {}
```

```
    for demand_node in demand_nodes:
        cost_dict[supply_node][demand_node] = costs.pop(0)

prob = LpProblem("MaterialSupplyProblem", LpMinimize)

Routes = [(w, b) for w in supply_nodes for b in demand_nodes]
vars = LpVariable.dicts("Route", (supply_nodes, demand_nodes), 0, None,␣
 ↪LpInteger)

prob += lpSum([vars[w][b] * cost_dict[w][b] for (w, b) in Routes]),␣
 ↪"Sum_of_Transporting_Costs"

for w in supply_nodes:
    prob += (
        lpSum([vars[w][b] for b in demand_nodes]) <= supply_dict[w],
        "Sum_of_Products_out_of_warehouses_%s" % w,
    )

for b in demand_nodes:
    prob += (
        lpSum([vars[w][b] for w in supply_nodes]) >= demand_dict[b],
        "Sum_of_Products_into_projects%s" % b,
    )

for w in supply_nodes:
    for b in demand_nodes:
        if w != b:
            prob += (vars[w][b] >= 0, "No_Self_Transportation_%s_%s" % (w, b))
        else:
            prob += (vars[w][b] == 0, "No_Self_Transportation_%s_%s" % (w, b))
            vars[w][b].lowBound = 0

cplex_solver = CPLEX_CMD(msg=True)
prob.solve(cplex_solver)


end_time = time.time()

elapsed_time = end_time - start_time

if elapsed_time >= 60:
    minutes = int(elapsed_time // 60)
    seconds = int(elapsed_time % 60)
    time_display = f"{minutes} minutes and {seconds} seconds"
else:
    seconds = int(elapsed_time)
    time_display = f"{seconds} seconds"
```

```
for v in prob.variables():
    print(v.name, "=", v.varValue)

print("Value of Objective Function = ", value(prob.objective))
print("Time taken to solve the model: ", time_display)
```

```
Route_PLANT01_PORT09 = None
Route_PLANT02_PORT09 = None
Route_PLANT03_PORT09 = None
Route_PLANT04_PORT09 = None
Route_PLANT05_PORT09 = None
Route_PLANT06_PORT09 = None
Route_PLANT07_PORT09 = None
Route_PLANT08_PORT09 = None
Route_PLANT09_PORT09 = None
Route_PLANT10_PORT09 = None
Route_PLANT11_PORT09 = None
Route_PLANT12_PORT09 = None
Route_PLANT13_PORT09 = None
Route_PLANT14_PORT09 = None
Route_PLANT15_PORT09 = None
Route_PLANT16_PORT09 = None
Route_PLANT17_PORT09 = None
Route_PLANT18_PORT09 = None
Route_PLANT19_PORT09 = None
Value of Objective Function =  None
Time taken to solve the model:  11 minutes and 24 seconds
```

4.3 3. Linear Programming (LP): PuLP

The objective function and constraints associated will first be redefined. The objective function to minimize can be expressed as follows:

$$Minimize : \sum (WC_{ki} + TC_{kpi})$$

Having:

$$WC = \sum_{kijp} c_{kijp}^{WC} X_{kijp}$$

$$TC = \sum_{kijp} c_{kijp}^{TC} X_{kijp}$$

- c_{kijp}^{WC}: The warehousing cost for order k at plant p.
- c_{kijp}^{TC}: The transportation cost of order k from port i to port j.

To solve the optimization problem, here are how the new constraints will be displayed.

1. Serve All Demands Each order must be fully served from a single route:

$$\sum_{i,j,p} X_{kijp} = 1 \ \forall k \in K$$

2. Plant Capacity The total number of orders handled by each plant must not exceed the plant's capacity:

$$\sum_{k,i,j} X_{kijp} \leq P_p^{cap} \ \forall p \in P$$

Where: - P_p^{cap} is the maximum capacity of plant p.

3. Weight Capacity of Carriers The total weight of the orders transported between ports must not exceed the carrier's weight limit:

$$\sum_{k,p} w_k X_{kijp} \leq C_{ij}^{max} \ \forall i \in I \text{ and } j \in J$$

Where: - w_k is the weight of order k. - C_{ij}^{max} is the maximum carrying capacity between ports i and j.

As **possible_routing** will contain only the available routing options, constraints like no self-transportation constraints are not needed.

[125]:
```
file = pd.ExcelFile(r'C:\Users\tomol\OneDrive\Desktop\Supply chain logisitcs␣
↪problem.xlsx')
df_dict = {}
for names in file.sheet_names:
    globals()[names] = file.parse(names)
    df_dict[names] = globals()[names]
print(df_dict.keys())
for df_name, df in df_dict.items():
    df.columns = [col.strip().replace(' ', '_').replace('/', '_').lower() for␣
↪col in df.columns]
```

```
dict_keys(['OrderList', 'FreightRates', 'WhCosts', 'WhCapacities',
'ProductsPerPlant', 'VmiCustomers', 'PlantPorts'])
```

[126]:
```
df_dict['OrderList']['order_id'] = [i+1 for i in␣
↪range(len(df_dict['OrderList']))]

order_no = len(df_dict['OrderList'])
ports = list(df_dict['PlantPorts']['port'].unique()); ports.sort()
plants = list(df_dict['WhCosts']['wh'].unique()); plants.sort()
orders = df_dict['OrderList']['order_id'].tolist()[:order_no]
```

```
demand_ports = list(df_dict['OrderList']['destination_port'].unique());␣
 ↪demand_ports.sort()

carriers_info = {}
for port in ports:
  for demand_port in demand_ports:
    matches_df =␣
 ↪df_dict['FreightRates'][(df_dict['FreightRates']['orig_port_cd'] == port) &␣
 ↪(df_dict['FreightRates']['dest_port_cd'] == demand_port)]
    if matches_df.empty:
        continue
    max_qty = matches_df['max_wgh_qty'].max()
    min_qty = matches_df['minm_wgh_qty'].min()
    min_rate = matches_df['rate'].min()
    rate = matches_df['rate'].max()
    carriers_info[(port, demand_port)] = {"min": min_qty, "max": 2 * max_qty,␣
 ↪"fixed_cost":min_rate, "rate": rate}
carriers_info
```

[126]:
```
{('PORT02', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.0484,
  'rate': 7.0608},
 ('PORT03', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.11560000000000001,
  'rate': 128.02720000000002},
 ('PORT04', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.0424,
  'rate': 7.664},
 ('PORT05', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.07200000000000001,
  'rate': 12.072799999999999},
 ('PORT06', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.07400000000000001,
  'rate': 7.1628},
 ('PORT07', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.1448,
  'rate': 0.2772},
 ('PORT08', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.1016,
  'rate': 1.8272},
```

```
 ('PORT09', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.0332,
  'rate': 21.278399999999998},
 ('PORT10', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.0964,
  'rate': 18.963054247870332},
 ('PORT11', 'PORT09'): {'min': 0.0,
  'max': 199999.98,
  'fixed_cost': 0.0684,
  'rate': 0.34480000000000005}}
```

[127]:
```
plants_cap = {plant: 7 *␣
 ↪df_dict['WhCapacities'][df_dict['WhCapacities']['plant_id'] ==␣
 ↪plant]['daily_capacity'].values[0] for plant in plants}
plants_cap
```

[127]:
```
{'PLANT01': 7490,
 'PLANT02': 966,
 'PLANT03': 7091,
 'PLANT04': 3878,
 'PLANT05': 2695,
 'PLANT06': 343,
 'PLANT07': 1855,
 'PLANT08': 98,
 'PLANT09': 77,
 'PLANT10': 826,
 'PLANT11': 2324,
 'PLANT12': 1463,
 'PLANT13': 3430,
 'PLANT14': 3843,
 'PLANT15': 77,
 'PLANT16': 3199,
 'PLANT17': 56,
 'PLANT18': 777,
 'PLANT19': 49}
```

[128]:
```
VmiPlants = df_dict['VmiCustomers']['plant_code'].unique().tolist()
VmiCustomers = df_dict['VmiCustomers']['customers'].unique().tolist()
VmiPlantCustomers = {plant:␣
 ↪df_dict['VmiCustomers'][df_dict['VmiCustomers']['plant_code'] ==␣
 ↪plant]['customers'].tolist() for plant in VmiPlants}
print("VmiPlants = ", VmiPlants)
FreePlants = [plant for plant in plants if plant not in VmiPlants]
print("FreePlants = ", FreePlants)
print("VmiCustomers = ", VmiCustomers)
```

```
print("VmiPlantCustomers = ", VmiPlantCustomers)
```

```
VmiPlants =  ['PLANT02', 'PLANT06', 'PLANT10', 'PLANT11']
FreePlants =  ['PLANT01', 'PLANT03', 'PLANT04', 'PLANT05', 'PLANT07', 'PLANT08',
'PLANT09', 'PLANT12', 'PLANT13', 'PLANT14', 'PLANT15', 'PLANT16', 'PLANT17',
'PLANT18', 'PLANT19']
VmiCustomers =  ['V5555555555555_16', 'V555555555555555_29', 'V555555555_3',
'V555555555555555_8', 'V55555555_9', 'V55555_10', 'V55555555_5',
'V5555555555555555_18', 'V555555_34', 'V55555555555555555_54']
VmiPlantCustomers =  {'PLANT02': ['V5555555555555_16', 'V555555555555555_29',
'V555555555_3', 'V555555555555555_8', 'V55555555_9', 'V55555_10', 'V55555555_5'],
'PLANT06': ['V5555555555555555_18', 'V55555_10'], 'PLANT10':
['V555555555555555_29', 'V555555_34', 'V55555555555555555_54', 'V55555_10'],
'PLANT11': ['V555555555555555555_54']}
```

[129]:
```
plant_products = {plant:␣
↪df_dict['ProductsPerPlant'][df_dict['ProductsPerPlant']['plant_code'] ==␣
↪plant]['product_id'].tolist() for plant in plants}
port_plants = {port: df_dict['PlantPorts'][df_dict['PlantPorts']['port'] ==␣
↪port]['plant_code'].tolist() for port in ports}
```

[130]:
```
possible_routes = {}
order_chances = {}
counter = 0
for index, order in df_dict['OrderList'].iterrows():
    if counter >= order_no:
        break
    counter += 1
    order_id = order['order_id']
    dest_port = order['destination_port']
    qty = order['unit_quantity']
    weight = order['weight']
    customer_id = order['customer']
    product_id = order['product_id']
    service_type = order['service_level']
    is_there_source = False
    order_chances[order_id] = []
    for port in ports:
        if (port, dest_port) in carriers_info:
          for plant in port_plants[port]:
            if product_id in plant_products[plant]:
              if (plant not in VmiPlants) or (plant in VmiPlants and␣
↪customer_id in VmiPlantCustomers[plant]):
                is_there_source = True

                rate = carriers_info[(port, dest_port)]['rate'] if service_type␣
↪!= 'CRF' else 0.0
```

```
                transportation_cost = weight * rate      # weight * rate
                cost_per_unit    = df_dict['WhCosts'][df_dict['WhCosts']['wh']␣
↪== plant]['cost_unit'].values[0]
                warehousing_cost = qty * cost_per_unit   # quantity *␣
↪cost_per_unit
                possible_routes[order_id, port, dest_port, plant] =␣
↪{"transportation_cost": transportation_cost, "warehousing_cost":␣
↪warehousing_cost, "qty": qty, "weight": weight}
                order_chances[order_id].append((order_id, port, dest_port,␣
↪plant))


    if not is_there_source:
      print("No source available for product_id: ", product_id)
```

[144]:
```
start_time = time.time()


prob = LpProblem("MaterialSupplyProblem", LpMinimize)

customer_plant = []
for (order,orig_port,dest_port, plant) in possible_routes.keys():
  if (order, plant) not in customer_plant:
    customer_plant.append((order, plant))

X = {}


for (order,orig_port,dest_port, plant) in possible_routes.keys():
  X[order,orig_port,dest_port, plant] = LpVariable("X({0},{1},{2},{3})".
↪format(order,orig_port,dest_port, plant), cat = "Binary")


for order_id in orders:
  prob += ( lpSum([X[order,orig_port,dest_port, plant] for␣
↪order,orig_port,dest_port, plant in order_chances[order_id]]) == 1)

for plant in plants:
  prob += (lpSum([X[order,orig_port, port, supply_plant] for (order, orig_port,␣
↪port, supply_plant) in possible_routes.keys() if supply_plant == plant]) <=␣
↪plants_cap[plant])

for from_port, to_port in carriers_info.keys():
```

```
  prob += (lpSum([X[order,orig_port,dest_port, plant] *␣
 ↪possible_routes[order,orig_port,dest_port, plant]['weight'] for␣
 ↪(order,orig_port,dest_port, plant) in possible_routes.keys() if orig_port ==␣
 ↪from_port and dest_port == to_port]) <= carriers_info[from_port,␣
 ↪to_port]['max'])

WC = lpSum([X[order,orig_port,dest_port, plant] *␣
 ↪possible_routes[order,orig_port,dest_port, plant]['warehousing_cost'] for␣
 ↪(order,orig_port,dest_port, plant) in possible_routes.keys()])
TC = lpSum([X[order,orig_port,dest_port, plant] *␣
 ↪possible_routes[order,orig_port,dest_port, plant]['transportation_cost'] for␣
 ↪(order,orig_port,dest_port, plant) in possible_routes.keys()])
prob.setObjective(WC + TC)

prob.solve()
end_time = time.time()

elapsed_time = end_time - start_time

print("Status : ",LpStatus[prob.status])
if LpStatus[prob.status] == "Optimal":
  print("****************************************")
  obj = value(prob.objective)
  print("Objective Function: ${:,.2f}".format(obj))
  print("Running Time: {:.2f} sec".format(elapsed_time))
  print("Solution Time: {:.2f} sec".format(prob.solutionTime))
```

```
Status :  Optimal
****************************************
Objective Function: $17,074,339.45
Running Time: 3.27 sec
Solution Time: 0.52 sec
```

[110]:
```
num_runs = 10

total_time = 0
total_objective_value = 0

for i in range(num_runs):
    start_time = time.time()

    prob = LpProblem("MaterialSupplyProblem", LpMinimize)

    customer_plant = []
    for (order, orig_port, dest_port, plant) in possible_routes.keys():
        if (order, plant) not in customer_plant:
            customer_plant.append((order, plant))
```

```
    X = {}

    for (order, orig_port, dest_port, plant) in possible_routes.keys():
        X[order, orig_port, dest_port, plant] = LpVariable("X({0},{1},{2},{3})".
↪format(order, orig_port, dest_port, plant), cat="Binary")

    for order_id in orders:
        prob += (lpSum([X[order, orig_port, dest_port, plant] for order,␣
↪orig_port, dest_port, plant in order_chances[order_id]]) == 1)

    for plant in plants:
        prob += (lpSum([X[order, orig_port, port, supply_plant] for (order,␣
↪orig_port, port, supply_plant) in possible_routes.keys() if supply_plant ==␣
↪plant]) <= plants_cap[plant])

    for from_port, to_port in carriers_info.keys():
        prob += (lpSum([X[order, orig_port, dest_port, plant] *␣
↪possible_routes[order, orig_port, dest_port, plant]['weight'] for (order,␣
↪orig_port, dest_port, plant) in possible_routes.keys() if orig_port ==␣
↪from_port and dest_port == to_port]) <= carriers_info[from_port,␣
↪to_port]['max'])

    WC = lpSum([X[order, orig_port, dest_port, plant] * possible_routes[order,␣
↪orig_port, dest_port, plant]['warehousing_cost'] for (order, orig_port,␣
↪dest_port, plant) in possible_routes.keys()])
    TC = lpSum([X[order, orig_port, dest_port, plant] * possible_routes[order,␣
↪orig_port, dest_port, plant]['transportation_cost'] for (order, orig_port,␣
↪dest_port, plant) in possible_routes.keys()])
    prob.setObjective(WC + TC)

    prob.solve()

    end_time = time.time()

    elapsed_time = end_time - start_time
    total_time += elapsed_time

    if LpStatus[prob.status] == "Optimal":
        obj = value(prob.objective)
        total_objective_value += obj

average_time = total_time / num_runs
average_objective_value = total_objective_value / num_runs

print(f"Average Running Time: {average_time:.2f} sec")
```

```
print(f"Average Objective Function: ${average_objective_value:,.2f}")
```

```
Average Running Time: 3.10 sec
Average Objective Function: $17,074,339.45
```

[115]:
```
final_routes = {}
for key, v in X.items():
  if v.varValue > 0:
    if (key[1], key[2]) not in final_routes:
      final_routes[(key[1], key[2])] = {key[3]: {'qty': v.varValue}}
    else:
      if key[3] not in final_routes[(key[1], key[2])]:
        final_routes[(key[1], key[2])][key[3]] = {'qty': v.varValue}
      else:
        final_routes[(key[1], key[2])][key[3]]['qty'] += v.varValue
final_routes
```

[115]:
```
{('PORT09', 'PORT09'): {'PLANT16': {'qty': 173.0}},
 ('PORT04', 'PORT09'): {'PLANT03': {'qty': 7091.0},
  'PLANT09': {'qty': 74.0},
  'PLANT12': {'qty': 280.0},
  'PLANT13': {'qty': 96.0},
  'PLANT08': {'qty': 1.0}},
 ('PORT05', 'PORT09'): {'PLANT04': {'qty': 387.0}},
 ('PORT06', 'PORT09'): {'PLANT05': {'qty': 180.0}},
 ('PORT02', 'PORT09'): {'PLANT07': {'qty': 102.0}, 'PLANT10': {'qty': 183.0}},
 ('PORT11', 'PORT09'): {'PLANT18': {'qty': 9.0}},
 ('PORT03', 'PORT09'): {'PLANT02': {'qty': 638.0}},
 ('PORT07', 'PORT09'): {'PLANT14': {'qty': 1.0}}}
```

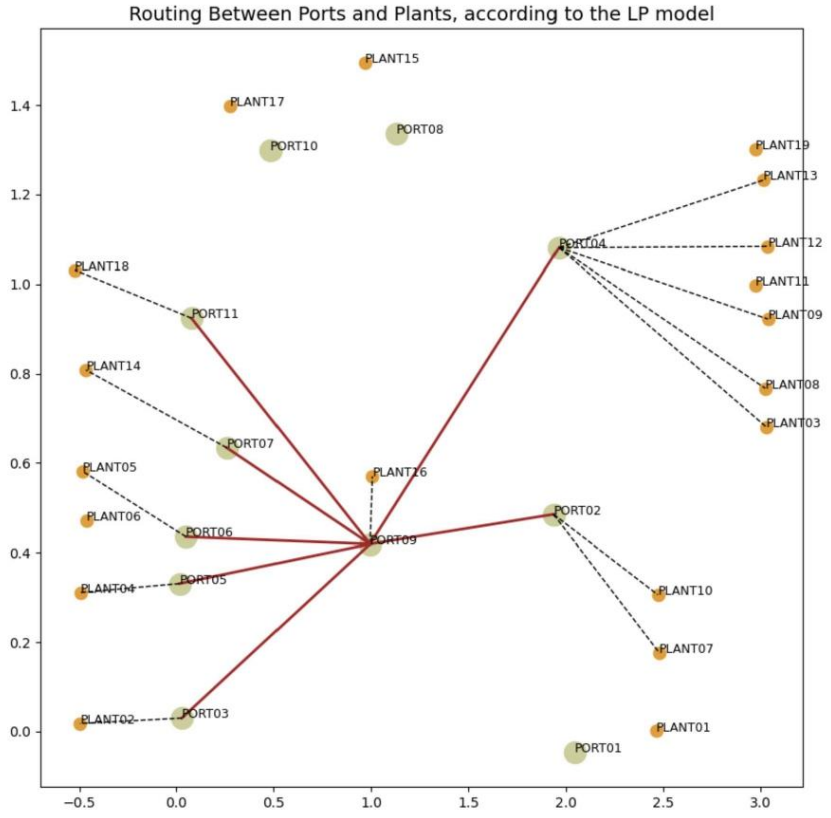
[116]:
```
for key, v in X.items():
  if v.varValue:
    print(v.name, "=", v.varValue)
```

```
X(1,PORT09,PORT09,PLANT16) = 1.0
X(2,PORT09,PORT09,PLANT16) = 1.0
X(3,PORT09,PORT09,PLANT16) = 1.0
X(4,PORT09,PORT09,PLANT16) = 1.0
X(5,PORT09,PORT09,PLANT16) = 1.0
X(6,PORT09,PORT09,PLANT16) = 1.0
X(7,PORT09,PORT09,PLANT16) = 1.0
X(8,PORT09,PORT09,PLANT16) = 1.0
X(9,PORT09,PORT09,PLANT16) = 1.0
X(10,PORT09,PORT09,PLANT16) = 1.0
X(11,PORT09,PORT09,PLANT16) = 1.0
X(12,PORT09,PORT09,PLANT16) = 1.0
X(13,PORT09,PORT09,PLANT16) = 1.0
X(14,PORT09,PORT09,PLANT16) = 1.0
```

```
X(9183,PORT03,PORT09,PLANT02) = 1.0
X(9184,PORT03,PORT09,PLANT02) = 1.0
X(9185,PORT03,PORT09,PLANT02) = 1.0
X(9186,PORT03,PORT09,PLANT02) = 1.0
X(9187,PORT03,PORT09,PLANT02) = 1.0
X(9188,PORT03,PORT09,PLANT02) = 1.0
X(9189,PORT03,PORT09,PLANT02) = 1.0
X(9190,PORT03,PORT09,PLANT02) = 1.0
X(9191,PORT03,PORT09,PLANT02) = 1.0
X(9192,PORT03,PORT09,PLANT02) = 1.0
X(9193,PORT03,PORT09,PLANT02) = 1.0
X(9194,PORT03,PORT09,PLANT02) = 1.0
X(9195,PORT03,PORT09,PLANT02) = 1.0
X(9196,PORT03,PORT09,PLANT02) = 1.0
X(9197,PORT03,PORT09,PLANT02) = 1.0
X(9198,PORT03,PORT09,PLANT02) = 1.0
X(9199,PORT03,PORT09,PLANT02) = 1.0
X(9200,PORT03,PORT09,PLANT02) = 1.0
X(9201,PORT03,PORT09,PLANT02) = 1.0
X(9202,PORT04,PORT09,PLANT03) = 1.0
X(9203,PORT04,PORT09,PLANT03) = 1.0
X(9204,PORT04,PORT09,PLANT03) = 1.0
X(9205,PORT04,PORT09,PLANT03) = 1.0
X(9206,PORT04,PORT09,PLANT03) = 1.0
X(9207,PORT04,PORT09,PLANT03) = 1.0
X(9208,PORT04,PORT09,PLANT03) = 1.0
X(9209,PORT04,PORT09,PLANT03) = 1.0
X(9210,PORT04,PORT09,PLANT03) = 1.0
X(9211,PORT04,PORT09,PLANT03) = 1.0
X(9212,PORT04,PORT09,PLANT03) = 1.0
X(9213,PORT04,PORT09,PLANT03) = 1.0
X(9214,PORT04,PORT09,PLANT03) = 1.0
X(9215,PORT04,PORT09,PLANT03) = 1.0
```

[117]:
```
line_color = colors[0]
port_dot_color = colors[5]
plant_dot_color = colors[2]

plt.figure(figsize=(10, 10))

for (fr, to) in final_routes.keys():
    plt.plot([PortsPos[fr][0], PortsPos[to][0]], [PortsPos[fr][1],␣
 ↪PortsPos[to][1]],
             linewidth=2, c=line_color)
    port_plants_items = final_routes[(fr, to)]

    for plant in port_plants_items.keys():
```

```
        plt.plot([PortsPos[fr][0], PlantsPos[plant][0]], [PortsPos[fr][1],␣
 ↪PlantsPos[plant][1]],
                 'k--', linewidth=1)

plt.scatter([pos[0] for _, pos in PortsPos.items()], [pos[1] for _, pos in␣
 ↪PortsPos.items()],
            s=250, c=port_dot_color)
for port in ports:
    plt.annotate(port, PortsPos[port], fontsize=9)

plt.scatter([pos[0] for _, pos in PlantsPos.items()], [pos[1] for _, pos in␣
 ↪PlantsPos.items()],
            s=80, c=plant_dot_color)
for plant in plants:
    plt.annotate(plant, PlantsPos[plant], fontsize=9)
plt.title("Routing Between Ports and Plants, according to the LP model",␣
 ↪fontsize=14)
plt.show()
```



4.4 4. Linear Programming (LP): OR-Tools, CBC

[118]:
```
file = pd.ExcelFile(r'C:\Users\tomol\OneDrive\Desktop\Supply chain logisitcs␣
↪problem.xlsx')
df_dict = {}
for names in file.sheet_names:
    globals()[names] = file.parse(names)
    df_dict[names] = globals()[names]
print(df_dict.keys())

for df_name, df in df_dict.items():
    df.columns = [col.strip().replace(' ', '_').replace('/', '_').lower() for␣
↪col in df.columns]
```

```
dict_keys(['OrderList', 'FreightRates', 'WhCosts', 'WhCapacities',
'ProductsPerPlant', 'VmiCustomers', 'PlantPorts'])
```

[119]:
```
order_no = len(df_dict['OrderList'])
ports = list(df_dict['PlantPorts']['port'].unique()); ports.sort()
plants = list(df_dict['WhCosts']['wh'].unique()); plants.sort()
orders = df_dict['OrderList']['order_id'].tolist()[:order_no]

demand_ports = list(df_dict['OrderList']['destination_port'].unique());␣
 ↪demand_ports.sort()
carriers_info = {}
for port in ports:
  for demand_port in demand_ports:
    matches_df =␣
 ↪df_dict['FreightRates'][(df_dict['FreightRates']['orig_port_cd'] == port) &␣
 ↪(df_dict['FreightRates']['dest_port_cd'] == demand_port)]
    if matches_df.empty:
        continue
    max_qty = matches_df['max_wgh_qty'].max()
    min_qty = matches_df['minm_wgh_qty'].min()
    min_rate = matches_df['rate'].min()
    rate = matches_df['rate'].max()
    carriers_info[(port, demand_port)] = {"min": min_qty, "max": max_qty,␣
 ↪"fixed_cost":min_rate, "rate": rate}

plants_cap = {plant:␣
 ↪df_dict['WhCapacities'][df_dict['WhCapacities']['plant_id'] ==␣
 ↪plant]['daily_capacity'].values[0] for plant in plants}

VmiPlants = df_dict['VmiCustomers']['plant_code'].unique().tolist()
VmiCustomers = df_dict['VmiCustomers']['customers'].unique().tolist()
VmiPlantCustomers = {plant:␣
 ↪df_dict['VmiCustomers'][df_dict['VmiCustomers']['plant_code'] ==␣
 ↪plant]['customers'].tolist() for plant in VmiPlants}
FreePlants = [plant for plant in plants if plant not in VmiPlants]

plant_products = {plant:␣
 ↪df_dict['ProductsPerPlant'][df_dict['ProductsPerPlant']['plant_code'] ==␣
 ↪plant]['product_id'].tolist() for plant in plants}
port_plants = {port: df_dict['PlantPorts'][df_dict['PlantPorts']['port'] ==␣
 ↪port]['plant_code'].tolist() for port in ports}



possible_routes = {}
order_chances = {}
counter = 0
```

```
for index, order in df_dict['OrderList'].iterrows():
    if counter >= order_no:
        break
    counter += 1
    order_id = order['order_id']
    dest_port = order['destination_port']
    qty = order['unit_quantity']
    weight = order['weight']
    customer_id = order['customer']
    product_id = order['product_id']
    service_type = order['service_level']
    is_there_source = False
    order_chances[order_id] = []
    for port in ports:
        if (port, dest_port) in carriers_info:
          for plant in port_plants[port]:
            if product_id in plant_products[plant]:
              if (plant not in VmiPlants) or (plant in VmiPlants and␣
 ↪customer_id in VmiPlantCustomers[plant]):
                is_there_source = True
                rate = carriers_info[(port, dest_port)]['rate'] if service_type␣
 ↪!= 'CRF' else 0.0
                transportation_cost = weight * rate      # weight * rate
                cost_per_unit   = df_dict['WhCosts'][df_dict['WhCosts']['wh']␣
 ↪== plant]['cost_unit'].values[0]
                warehousing_cost = qty * cost_per_unit   # quantity *␣
 ↪cost_per_unit
                possible_routes[order_id, port, dest_port, plant] =␣
 ↪{"transportation_cost": transportation_cost, "warehousing_cost":␣
 ↪warehousing_cost, "qty": qty, "weight": weight}
                order_chances[order_id].append((order_id, port, dest_port,␣
 ↪plant))

    if not is_there_source:
      print("No source available for product_id: ", product_id)
```

[135]:
```
from ortools.linear_solver import pywraplp

start_time_ortools = time.time()

solver = pywraplp.Solver.CreateSolver('CBC')

customer_plant_ortools = []
X_ortools, O_ortools = {}, {}

for (order, orig_port, dest_port, plant) in possible_routes.keys():
```

```
    if (order, plant) not in customer_plant_ortools:
        customer_plant_ortools.append((order, plant))

for (order, orig_port, dest_port, plant) in possible_routes.keys():
    X_ortools[(order, orig_port, dest_port, plant)] = solver.
 ↪BoolVar("X_ortools({},{},{},{})".format(order, orig_port, dest_port, plant))

for (order, plant) in customer_plant_ortools:
    O_ortools[(order, plant)] = solver.BoolVar("O_ortools({},{})".format(order,␣
 ↪plant))

for (order, orig_port, dest_port, plant) in possible_routes.keys():
    solver.Add(X_ortools[(order, orig_port, dest_port, plant)] <=␣
 ↪O_ortools[(order, plant)])

for order_id in orders:
    solver.Add(solver.Sum(X_ortools[(order, orig_port, dest_port, plant)] for␣
 ↪(order, orig_port, dest_port, plant) in order_chances[order_id]) == 1)

for plant in plants:
    solver.Add(solver.Sum(X_ortools[(order, orig_port, port, supply_plant)] for␣
 ↪(order, orig_port, port, supply_plant) in possible_routes.keys() if␣
 ↪supply_plant == plant) <= 7 * plants_cap[plant])

for from_port, to_port in carriers_info.keys():
    solver.Add(solver.Sum(X_ortools[(order, orig_port, dest_port, plant)] *␣
 ↪possible_routes[(order, orig_port, dest_port, plant)]['weight'] for (order,␣
 ↪orig_port, dest_port, plant) in possible_routes.keys() if orig_port ==␣
 ↪from_port and dest_port == to_port) <= 2 * carriers_info[(from_port,␣
 ↪to_port)]['max'])



WC_ortools = solver.Sum(X_ortools[(order, orig_port, dest_port, plant)] *␣
 ↪possible_routes[(order, orig_port, dest_port, plant)]['warehousing_cost']␣
 ↪for (order, orig_port, dest_port, plant) in possible_routes.keys())
TC_ortools = solver.Sum(X_ortools[(order, orig_port, dest_port, plant)] *␣
 ↪possible_routes[(order, orig_port, dest_port, plant)]['transportation_cost']␣
 ↪for (order, orig_port, dest_port, plant) in possible_routes.keys())
solver.Minimize(WC_ortools + TC_ortools)

status_ortools = solver.Solve()

end_time_ortools = time.time()
elapsed_time_ortools = end_time_ortools - start_time_ortools
```

```
if status_ortools == pywraplp.Solver.OPTIMAL:
    print("*****************************************")
    objective_ortools = solver.Objective().Value()
    print(f"OR-Tools Objective Function: ${objective_ortools:,.2f}")
    print(f"OR-Tools Running Time: {elapsed_time_ortools:.2f} sec")
```

```
*****************************************
OR-Tools Objective Function: $16,712,493.00
OR-Tools Running Time: 2.98 sec
```

[136]:
```
results = {
    'Model': ['PuLP', 'OR-Tools'],
    'Objective Function Result': [obj, objective_ortools],
    'Time Taken (seconds)': [elapsed_time, elapsed_time_ortools]
}
df_results = pd.DataFrame(results)
print(df_results)
```

```
      Model  Objective Function Result  Time Taken (seconds)
0      PuLP               1.707434e+07              3.219297
1  OR-Tools               1.671249e+07              2.982937
```

4.5 5. Economic implications

[134]:
```
baseline_warehousing_cost = 0
baseline_transportation_cost = 0
optimized_warehousing_cost = 0
optimized_transportation_cost = 0

for order_id in orders:
    possible_order_routes = order_chances[order_id]

    first_route = possible_order_routes[0]
    order, orig_port, dest_port, plant = first_route

    warehousing_cost = possible_routes[order, orig_port, dest_port,␣
 ↪plant]['warehousing_cost']
    transportation_cost = possible_routes[order, orig_port, dest_port,␣
 ↪plant]['transportation_cost']

    baseline_warehousing_cost += warehousing_cost
    baseline_transportation_cost += transportation_cost

for (order, orig_port, dest_port, plant), var in X.items():
    if var.varValue == 1:
        warehousing_cost = possible_routes[order, orig_port, dest_port,␣
 ↪plant]['warehousing_cost']
```

```
        transportation_cost = possible_routes[order, orig_port, dest_port,␣
 ↪plant]['transportation_cost']

        optimized_warehousing_cost += warehousing_cost
        optimized_transportation_cost += transportation_cost

baseline_total_cost = baseline_warehousing_cost + baseline_transportation_cost
optimized_total_cost = optimized_warehousing_cost +␣
 ↪optimized_transportation_cost

objective_value = value(prob.objective)
if abs(optimized_total_cost - objective_value) < 1e-6:
    print("\nThe optimized total cost matches the objective function.")
else:
    print("\nOptimized total cost does not match the objective function.")

warehousing_savings = baseline_warehousing_cost - optimized_warehousing_cost
percentage_warehousing_savings = (warehousing_savings /␣
 ↪baseline_warehousing_cost) * 100 if baseline_warehousing_cost != 0 else 0

transportation_savings = baseline_transportation_cost -␣
 ↪optimized_transportation_cost
percentage_transportation_savings = (transportation_savings /␣
 ↪baseline_transportation_cost) * 100 if baseline_transportation_cost != 0␣
 ↪else 0

total_savings = baseline_total_cost - optimized_total_cost
percentage_total_savings = (total_savings / baseline_total_cost) * 100 if␣
 ↪baseline_total_cost != 0 else 0

print("\nComparison of Baseline vs Optimized Costs")

print("\n-- Warehousing Costs --")
print("Baseline Warehousing Cost: ${:,.2f}".format(baseline_warehousing_cost))
print("Optimized Warehousing Cost: ${:,.2f}".format(optimized_warehousing_cost))
print("Warehousing Cost saved: ${:,.2f} ({:.2f}%)".format(warehousing_savings,␣
 ↪percentage_warehousing_savings))

print("\n-- Transportation Costs --")
print("Baseline Transportation Cost: ${:,.2f}".
 ↪format(baseline_transportation_cost))
print("Optimized Transportation Cost: ${:,.2f}".
 ↪format(optimized_transportation_cost))
print("Transportation Cost saved: ${:,.2f} ({:.2f}%)".
 ↪format(transportation_savings, percentage_transportation_savings))
```

```
print("\n-- Total Costs --")
print("Baseline Total Cost: ${:,.2f}".format(baseline_total_cost))
print("Optimized Total Cost: ${:,.2f}".format(optimized_total_cost))
print("Total Cost saved: ${:,.2f} ({:.2f}%)".format(total_savings,␣
  ↪percentage_total_savings))
```

```
The optimized total cost matches the objective function.

Comparison of Baseline vs Optimized Costs

-- Warehousing Costs --
Baseline Warehousing Cost: $15,586,233.34
Optimized Warehousing Cost: $15,501,599.52
Warehousing Cost saved: $84,633.82 (0.54%)

-- Transportation Costs --
Baseline Transportation Cost: $2,640,491.23
Optimized Transportation Cost: $1,572,739.93
Transportation Cost saved: $1,067,751.30 (40.44%)

-- Total Costs --
Baseline Total Cost: $18,226,724.57
Optimized Total Cost: $17,074,339.45
Total Cost saved: $1,152,385.13 (6.32%)
```

6. Reference List

"The Economic Impact of Data Innovation 2023: Global research: 8 key strategies for disrupting competitors, building resilience and gaining a 9.5% profit edge", Splunk, 2023. https://www.splunk.com/en_us/campaigns/economic-impact-of-data.html

Raeesi, R., Sahebjamnia, N. & Mansouri, S.A., 2023. The synergistic effect of operational research and big data analytics in greening container terminal operations: A review and future directions. European Journal of Operational Research. https://doi.org/10.1016/j.ejor.2022.11.054

Kong, T., Feng, T. & Huo, B., 2021. Green supply chain integration and financial performance: a social contagion and information sharing perspective. Business Strategy and the Environment. https://doi-org.rennes-sb.idm.oclc.org/10.1002/bse.2745

Rugman, A.M. & Verbeke, A., 1998. Corporate Strategies and Environmental Regulations: An Organizing Framework. JSTOR. https://www.jstor.org/stable/3094073

Weersink, A., Fraser, E., Pannell, D., Duncan, E. & Rotz, S., 2018. Opportunities and Challenges for Big Data in Agricultural and Environmental Analysis. Annual Review of Resource Economics. https://www.annualreviews.org/doi/pdf/10.1146/annurev-resource-100516-053654

Arunachalam, D., Kumar, N. & Kawalek, J.P., 2017. Understanding big data analytics capabilities in supply chain management: Unravelling the issues, challenges and implications for practice. Transportation Research Part E: Logistics and Transportation Review. https://doi.org/10.1016/j.tre.2017.04.001

Seuring, S., Aman, S., Hettiarachchi, B., de Lima, A., Schilling, L. & Sudusinghe, J., 2022. Reflecting on theory development in sustainable supply chain management. Current Opinion in Green and Sustainable Chemistry. https://doi.org/10.1016/j.clscn.2021.100016

Menon, R. & Ravi, V., 2021. Analysis of barriers of sustainable supply chain management in electronics industry: An interpretive structural modelling approach. Circular Economy and Sustainability. https://doi.org/10.1016/j.clrc.2021.100026

Taghikhah, F., Daniel, J. & Mooney, G., 2017. Sustainable Supply Chain Analytics: Grand Challenges and Future Opportunities. SSRN. https://ssrn.com/abstract=3733718

Mageto, J., 2021. Big Data Analytics in Sustainable Supply Chain Management: A Focus on Manufacturing Supply Chains. Sustainability. https://doi-org.rennes-sb.idm.oclc.org/10.3390/su13137101

Merlino, M. & Sproge, I., 2017. The Augmented Supply Chain. Procedia Engineering. https://doi.org/10.1016/j.proeng.2017.01.053

Foulds, L. R., 2012. Optimization techniques: an introduction. Springer Science & Business Media. https://doi.org/10.1007/978-1-4613-9458-7

Kunwar, R., & Sapkota, H. P., 2022. An Introduction to Linear Programming Problems with Some Real-Life Applications. European Journal of Mathematics and Statistics. https://doi.org/10.24018/ejmath.2022.3.2.108

Grothey, A., McKinnon, K. (2023). On the effectiveness of sequential linear programming for the pooling problem. *Ann Oper Res* **322**, 691–711 https://doi.org/10.1007/s10479-022-05156-7

Silva, C., Ribeiro, R., Gomes, P. (2024). Algorithmic Optimization Techniques for Operations Research Problems. In: Silhavy, R., Silhavy, P. (eds) Data Analytics in System Engineering. CoMeSySo 2023. Lecture Notes in Networks and Systems, vol 935. Springer, Cham. https://doi.org/10.1007/978-3-031-54820-8_26