Optimizing Biotech operations through Database Management & Data Analysis

Applied Project Final Report

Carlos Olivella

Spring 2023

A paper submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Management and Systems

at the

Division of Programs in Business

School of Professional Studies

New York University

Table of Contents

1. Declaration

I grant powers of discretion to the Department, SPS, and NYU to allow this paper to be copied in part or in whole without further reference to me. This permission covers only copies made for study purposes or for inclusion in Department, SPS, and NYU research publications, subject to normal conditions of acknowledgement.

2. Business Background

BioSec LLC (hereafter, the "Client" or "BioSec") is a life-sciences-focused global developer and supplier of a complete line of Biosecurity and Biotechnology products for a variety of industries, including agriculture, biosecurity, and waste management. It has over 30 years of experience in the biological solutions industry, and an IP portfolio focused on adapted toxic resistant microorganisms that are highly efficient in biodegradation processes.

Among the products it sells, the Client offers different solutions that can be applied to surfaces and waste in order to clean them or remove odors. It also offers shoe stations that are used to disinfect employees' shoes once they enter a premise. Finally, it offers incinerators that are utilized to eliminate waste effectively.

As expected, the sale of industrial products involves large quantities of orders in addition to a large quantity of items per order. Currently, the Client is experiencing some difficulties managing, storing, and analyzing all the data that is generated by these sales. The data is being stored in different file formats managed by the Client's employees in different computers across the organization. This makes it difficult to access and analyze.

The Client is also worried that its current setup may lead to loss of data at some point given that these files can be corrupted or lost if the computer is damaged or stolen. As such, the Client is looking for a service that can store the large amounts of data it holds in a safe and centralized location where it can access and analyze it easily.

The Client's employees will benefit from this project because they will be able to access and analyze their data from a centralized location. Thus, the importance of this project is that it will enable the Client to gain meaningful insights from its sales, as opposed to just having a record of them.

3. Project Overview

This project will have two main sections. In first place, this project aims to create a proof of concept of a functional database using one of the main relational database management systems (RDBMS) used by companies throughout the world. These industry-leading RDBMS provide robust platforms that store data in servers in different regions of the world, providing solutions against any losses generated by single points of failure. This means that the database will be able to store the Client's sales data in a safe and easily-accessible manner.

In second place, this project aims to create a proof of concept of a dashboard using Python. This dashboard will enable the Client to easily visualize and analyze its data, allowing it to make more informed decisions about its sales efforts, and to identify trends and patterns in its data. A major part of this section will be to include a regression model within the dashboard that allows the Client to predict the next year's sales based on past data.

In the end, the Client will be able to determine what products and customers it should focus on more by: (i) importing its sales data into the database; and (ii) initializing the dashboard in order to visualize different sales metrics.

4. Project Charter

Project Manager: Carlos Olivella

Date Prepared: February 15, 2023

Name and Location of Client Organization: BioSec LLC, Miami, FL

4.1 Business Problem/Opportunity:

Currently, the Client is experiencing some difficulties managing, storing, and analyzing their sales data. The Client stores its sales data in various formats and locations, which makes it difficult to access and analyze. It is also worried that its current setup may lead to loss of data at some point. As such, the Client is looking for a service that can store the large amounts of data they hold in a safe and centralized location where it can access and analyze it easily.

4.2 Project Goal:

The goal of this project is to help the Client increase sales. The creation of a proof of concept of a functional database and dashboard that allows the Client to easily access, update, and analyze their sales data will allow it to make more informed decisions about their sales and marketing efforts, and to identify trends and patterns in their data. In the end, the Client will be able to determine what products and clients they should focus on more.

4.3 Project Description:

This project aims to address the business problem by providing the Client with a proof of concept of the following items:

- A centralized database (using a tool such as Oracle's relational database management system) to safely store the data that the Client currently possesses, as well as any future data that may be generated.
- A dashboard using tools such as HTML, CSS, Python, and Plotly Dash that will allow the company to view and analyze the data in a more user-friendly way using descriptive statistics.
- A machine learning model that will allow the Client to predict future sales and display it on the dashboard.

4.4 Project Sponsor:

Sebastian Lora, Director, Life Sciences

4.5 Project Objectives:

The objectives of this project are as follows:

- Set up a centralized database to store the sales data provided by the company, within 3 weeks of the project commencement.
- Develop a dashboard using HTML, CSS, Python, and Plotly Dash that allows the company to view and analyze the sales data in a user-friendly way, within 7 weeks of the project commencement.
- Build a regression model to be included in the dashboard to forecast future sales based on the past data, within 10 weeks of the project commencement.
- Test and refine the database, dashboard, and machine learning model to ensure that they are functioning as intended, within 12 weeks of the project commencement.

4.6 Project Scope

4.6.1 In-Scope Activities:

- Set up a centralized relational database that allows the Client to insert, update, and delete its sales data.

- Develop a dashboard using HTML, CSS, Python, and Plotly Dash that allows the Client to view and analyze the sales data in a user-friendly way. This dashboard must integrate with the relational database and receive data from it.
- Build a regression model to be included in the dashboard to forecast future sales using Python.

4.6.2 Out-of-Scope Activities:

- Collection and use of data other than the Client's sales data
- User authentication and access control features
- Use of programming languages that are different to the ones included in this project charter.
- Integration with other systems used or owned by the Client or provided by third parties.
- Machine learning or artificial intelligence models beyond the aforementioned regression model

4.7 Risks and Mitigation Strategies

4.7.1 Risk:

Data quality issues, such as missing or inconsistent data, could affect the accuracy of the dashboard and regression model.

4.7.2 Mitigation:

Implement data quality checks and validation rules during the data storage process to identify and correct errors. Perform exploratory data analysis to identify and handle any outliers or anomalies in the data. Any data quality issues that cannot be solved without the Client's intervention will require the Client's intervention. If this is not possible, the Project Manager will not be responsible for these data quality issues.

4.7.3 Risk:

The Client decides to change the project scope.

4.7.4 Mitigation:

The activities carried out in this project have been stated clearly in this project charter. As such, no changes can be made.

4.7.5 Risk:

Technical issues, such as software bugs or infrastructure failures, could cause downtime and delays in delivering the dashboard and regression model.

4.7.6 Mitigation:

Regularly test the dashboard and regression model to identify and fix any bugs or issues.

4.7.7 Risk:

The Project Manager is sick or incapacitated and, as such, cannot carry out the activities included in this project.

4.7.8 Mitigation:

Develop a contingency plan in close coordinator with the Client in order to modify the project plan once the Project Manager is available once again.

4.7.9 Risk:

The software and tools used throughout the project are incapable of sustaining the project's activities, or their free versions do not possess the required capabilities.

4.7.10 Mitigation:

The Project Manager will explore different software options and will coordinate with the Client to determine if these options can be used.

4.8 Communication Plan

- Frequency: Bi-weekly
- Method: Email
- Content: The Project Manager will send the Client status reports through email. These will include major activities carried out, as well as any issues that might exist.

4.9 Schedule Overview

- Project Start Date: February 24, 2023
- Estimated Project Completion Date: May 4, 2023
- Major Milestones:
- Database goes live – March 9, 2023
- Status report A is submitted – March 9, 2023
- Dashboard goes live – March 30, 2023
- Status report B is submitted – April 6, 2023
- Draft report is submitted – April 20, 2023
- Final report is submitted – April 27, 2023
- Client presentation – May 4, 2023

4.10 Impact of Late Delivery:

Late delivery of the project's deliverables will impact the Project Manager's grade. However, the Project Manager will still be responsible for the project's completion in a timely manner.

4.11 Project Plan

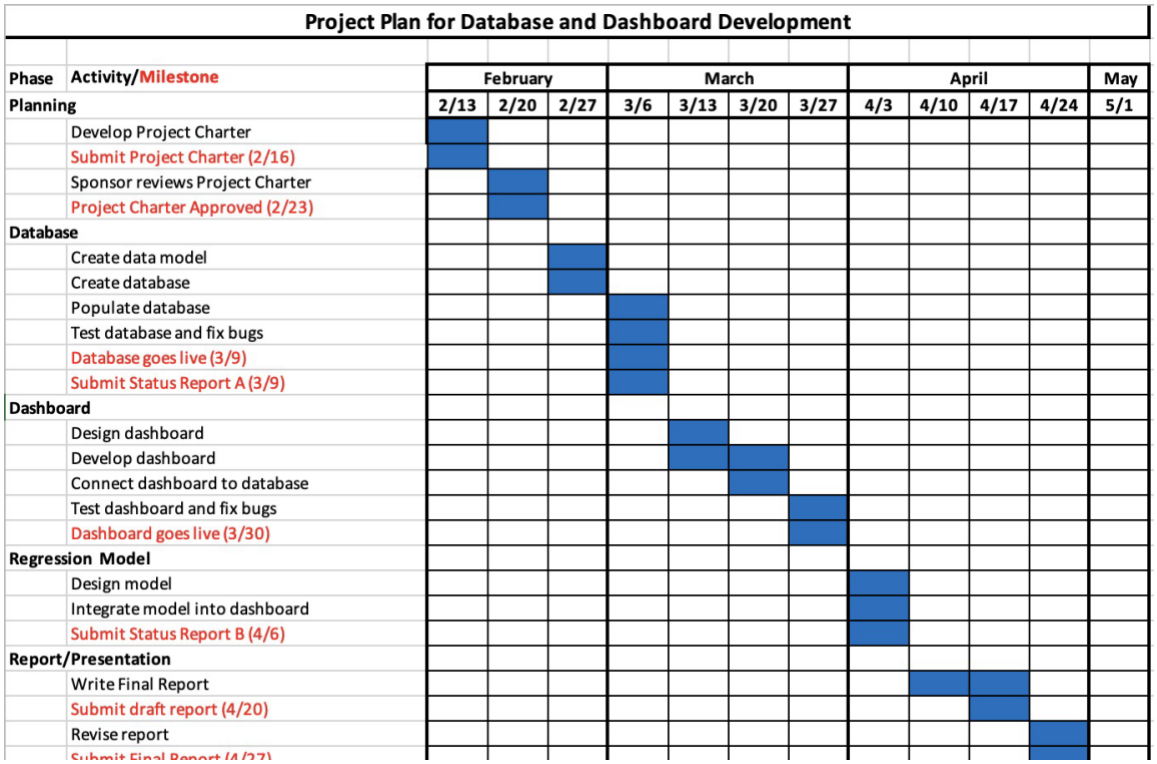


Fig. 4-1

5. Project Plan

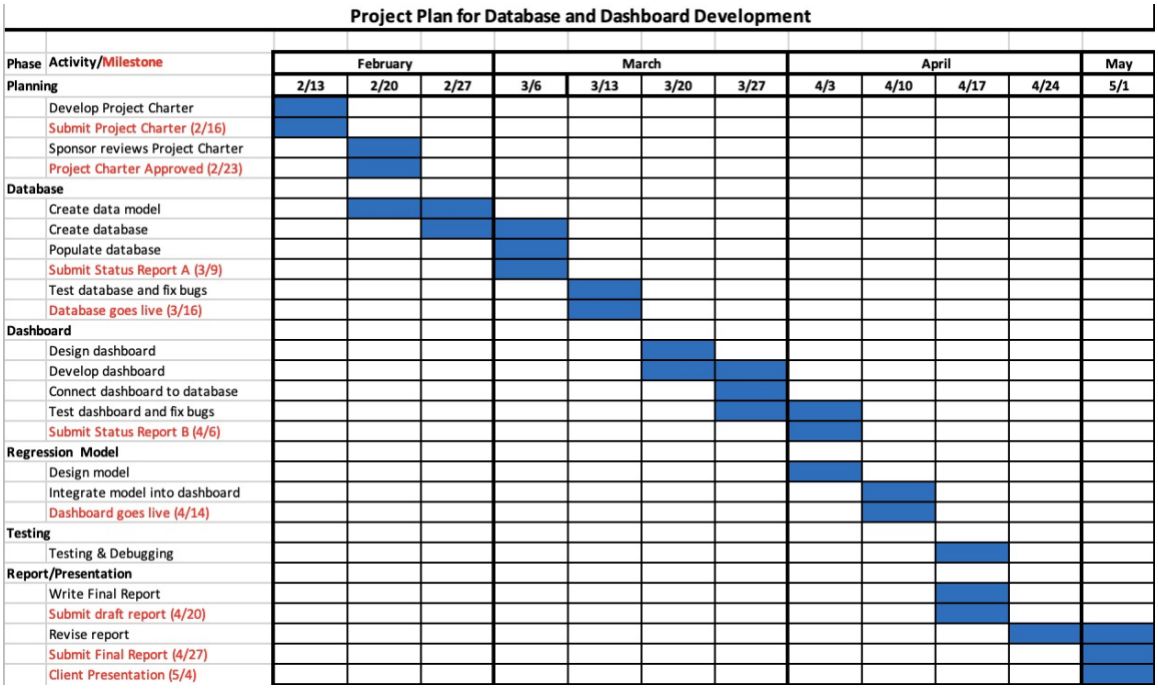


Fig. 5-4-1

Element Name	Description	Due By
Develop Project Charter	This document must include information about the company and sponsor, the project timeline, and objectives, as well as deliverables.	2/16
Sponsor Reviews Project Charter	The Sponsor reviews and approves the project charter.	2/22

Submit Project Charter	The Project Charter must be submitted to Brightspace.	2/23
Create Data Model	The layout of the database must be designed, along with the relationships between tables, primary keys, and foreign keys.	3/5
Submit Status Report A	The report must contain information about the milestones that have been completed, and whether or not there have been issues in the execution of the tasks.	3/9
Create Database	Using the layout created previously, the database must be created by running the corresponding SQL script.	3/12
Populate Database	The database must be populated with the dummy data. The data must be separated into each of the tables within the database using SQL.	3/12

Test Database and Fix Bugs	Any bug or errors that arise in the database must be fixed in this phase.	3/15
Database goes live	The database will be fully operational on this date, meaning that the tables have been created, the relationships between them work correctly, and data can be imported, modified, and deleted from the database without errors.	3/16
Design dashboard	This involves deciding which metrics and graphs will be displayed on the dashboard, how the layout will be organized, and how the overall aesthetic will look like.	3/26
Develop dashboard	The dashboard will be developed using Python and Plotly Dash. This involves writing the Python script to create the layout and colors, as well as the graphs. The graphs will be empty at this point, as the data will	4/2

	be fed into the dashboard in the next phase.	
Connect dashboard to database	The dashboard will be connected to the database so that the correct data is fed into each of the graphs and dropdowns included in the dashboard.	4/2
Submit Status Report B	The report must contain information about the milestones that have been completed, and whether or not there have been issues in the execution of the tasks.	4/6
Test Dashboard and fix bugs	Any bug or errors that arise in the dashboard must be fixed in this phase.	4/9
Design Regression model	The regression model will be designed using Python. This will involve deciding what type of regression to use and writing the corresponding script.	4/9

Integrate model into Dashboard	Once the script is validated and is working correctly, the predicted data must be integrated into the dashboard.	4/13
Dashboard goes live	The dashboard will be fully operational on this date, meaning that the layout, colors, graphs, metrics, dropdowns, and data are all correct and working properly. The dashboard must allow the user to modify the data inputted into the dashboard using the dropdowns, and must refresh in real time to display the correct data.	4/14
Testing and debugging	Any bug or errors that arise in the overall project must be fixed in this phase.	4/18
Submit Draft report	A draft of the final report must be submitted on this date.	4/20

Submit Final Report	The final report must be submitted on this date.	4/27
Final Presentation	The project will be presented to the professor and the project sponsor.	5/4

6. Literature Review

6.1 Introduction

Data is becoming increasingly important for companies and executives when making decisions. Estimates state that the global data analytics market accounted for USD 31.8 billion in 2021 and is projected to increase to USD 329.8 billion by 2030 (Johnson, 2022). This constitutes an impressive Compound Annual Growth Rate (CAGR) of 29.9% from 2022 to 2030. Companies see how useful data analysis can be for their businesses. However, managing data can be difficult. As we will see, two of the biggest problems that companies currently face regarding data are: (i) data storage; (ii) data management; and (iii) data analysis & visualization.

6.2 Data Storage

Data storage poses a problem to companies for several reasons. Companies that use local storage methods instead of cloud storage can face risks such as data loss due to critical failures, theft, or human error (Davin, 2018). This is why services like cloud storage have become so popular in recent years. A 2016 survey projected that Cloud computing and storage services would increase by 50%, while software and infrastructure by 33%, and would eventually overshadow the in-house hardware and software (Odun-Ayo et al., 2017).

Cloud computing is defined as “a parallel and distributed computing system consisting of a pool of interconnected and virtualized computers that are dynamically provisioned and presented a single computing resource to the users based on pre-agreed Service Level Agreements (SLA)” (Odun-Ayo et al., 2017). More specifically, cloud storage consists of storing data in third-party services instead of storing it in dedicated servers owned by the owner of the data which, in turn, has to build systems and architecture to store said data (Odun-Ayo et al., 2017).

Cloud storage provides significant benefits to its client. First, the client does not need to invest any capital in software and hardware used to store data; the cloud service provider does this for them. Second, the client does not need any technical expertise to maintain storage, backup, replication, and disaster management; it can contact the provider directly, whose employees will solve the issue. Finally, the client's data can be accessed from virtually any computer and by a person with the correct credentials (Arokia Paul Rajan & Shanmugapriyaa, 2012).

6.3 Data Management

Data management poses another problem for companies due to its difficulty. The sheer volume of data that a company possesses can inhibit a company from generating value from it, whether this is due to improper data categorization, storing data in multiple

locations and systems, having outdated or incorrect data, or not being able to integrate data with different tools (Malak, 2023).

The purpose of data integration is to query a variety of different sources by providing a "unified object model interface to meet the needs of the user's query" (HongJu et al., 2017). Company data is often stored in different locations and sources. However, companies that use the right tools can easily integrate and unify these different sources. On the contrary, companies that store their data locally on files distributed across different devices have a hard time doing this. Integrating numerous CSV files can be difficult without the correct tools. This is where database management systems come into play.

Database management systems are "programs that manage the database structure and control access to the data stored in the database" (Coronel & Morris, 2017). Among their benefits, database management systems can help a company improve data sharing, security, integration, and access (Coronel & Morris, 2017).

Numerous cloud service providers worldwide can provide cloud storage and management through their database management systems. However, companies like Amazon, Oracle, and Microsoft have led this industry for years. In my specific case, NYU licenses Oracle's SQL Developer platform for its students to use. Considering the limitations on spending

imposed on capstone projects, my project will use this platform exclusively for providing my Client with a database. Currently, my Client does not use cloud storage services. It stores its data locally. This presents an important opportunity to provide the Client with a database stored in the cloud that can help it avoid possible data loss, access the data from different locations, and integrate its data with other tools.

6.4 Data Analysis & Visualization

Storing and managing your data can only get you so far. It is extremely difficult to obtain valuable insights from a million rows of data at a glance. This data must be processed, analyzed, and, often, visualized for the user to obtain meaning from it. Dashboards are a great way of achieving this. Dashboards are a "visual display of data used to monitor conditions and/or facilitate understanding" (Wexler et al., 2017). Visualizing data is important because it allows us humans to identify relationships and trends between data. As such, dashboards can help companies obtain information about an array of different subjects and measures at a glance. They are also interactive, meaning that the user can change certain inputs in the dashboard, and it will produce the corresponding outputs.

In this case, we will be creating a dashboard for the Client. This dashboard will allow the Client to identify different relationships and trends concerning its sales data and play around with different measures. It must connect to the database that we create to obtain the

necessary data for visualization. To create the dashboard, I will use Python. This is due to its popularity in data analysis and data science, the ease with which applications can be developed in comparison with other languages, and the vast number of libraries available for use (Sutchenkov & Tikhonov, 2020).

However, several Python libraries can be used to create dashboards. Among these are Voila, Plotly Dash, and Panel, among others. Hereinafter, we will describe these libraries.

6.4.1 Voila

A Jupyter Notebook extension, Voila allows its user to create dashboards within Jupyter notebooks easily but also run them separately.

6.4.2 Panel

Like Voila, Panel also allows the user to create dashboards in and out of Jupyter notebooks. The user can add controls and layouts wherever they want within the notebook.

6.4.3 Plotly Dash

The creators of Plotly Dash intended this tool to have more of a focus on the enterprise level of creating dashboards. As such, it is more advanced than some other tools and offers more functionality. One advantage of Dash is that despite being fully written in Python, it

provides wrappers for HTML tags for Python. However, this also makes it more complicated, as the user must have some basic knowledge of HTML and CSS (Panel, 2023).

In addition to this, interactivity is achieved through what is known are callback functions. "These allow for reading the values of inputs in the Dash app (e.g., text inputs, dropdowns, and sliders), which can subsequently be used to compute the value of one or more "outputs", i.e., properties of other components in the app. The function that computes the outputs is wrapped in a decorator that specifies the aforementioned inputs and outputs; together, they form a callback. The callback is triggered whenever one of the specified inputs changes in value" (Hossain, 2019).

Due to the possibility of including more functionality in our dashboard, and allowing for use of HTML and CSS, I have chosen Plotly Dash as the Python library that I will use during this project.

6.5 Conclusion

As we have seen, data storage, management, analysis, and visualization all pose different issues for companies that want to manage and understand their data. The transition from local storage to cloud storage, as well as the use of a database management system, can be

extremely helpful for companies as it allows them to shift their focus towards gaining meaningful insight from data, and away from worrying about losing it or integrating it. Regarding data analysis and visualization, dashboards constitute a useful tool for companies to analyze their data at a glance and play around with different metrics. As such, my project will employ the use of Oracle's database management system to provide my Client with a database, and the use of Python and Plotly Dash to create a capable and interactive dashboard for the company to analyze its data.

6.6 References

Arokia Paul Rajan, R., & Shanmugapriyaa, S. (2012). Evolution of Cloud Storage as Cloud Computing Infrastructure Service. IOSR Journal of Computer Engineering (IOSRJCE), 1(1), 38–45.

Coronel, C., & Morris, S. (2017). Database Systems: Design, Implementation, and Management (13th ed.). Cengage Learning, Inc.

Davin, C. (2018, August 16). The Risks of Saving Data Locally. Toolkit by Davin Tech Group. https://davintechgroup.com/toolkit/the-risks-of-saving-data-locally/

HongJu, X., Fei, W., FenMei, W., & Xiuzhen, W. (2017). Some Key Problems of Data Management in Army Data Engineering Based on Big Data. 2017 IEEE 2nd International Conference on Big Data Analysis. https://ieeexplore-ieee-org.proxy.library.nyu.edu/stamp/stamp.jsp?tp=&arnumber=8078796

Hossain, S. (2019). Visualization of Bioinformatics Data with Dash Bio. 126–133. https://doi.org/10.25080/Majora-7ddc1dd1-012

Johnson, R. (2022, December 16). Data Analytics Market Size Set to Achieve USD 329.8 Billion by 2030 growing at 29.9% CAGR - Exclusive Report by Acumen Research and Consulting. https://www.globenewswire.com/news-release/2022/12/16/2575454/0/en/Data-Analytics-Market-Size-Set-to-Achieve-USD-329-8-Billion-by-2030-growing-at-29-9-CAGR-Exclusive-Report-by-Acumen-Research-and-Consulting.html#:~:text=LOS%20ANGELES%2C%20Dec.,29.9%25%20from%202022%20to%202030.

Malak, H. A. (2023, January 21). 11 Data Management Challenges and Solutions. The ECM Consultant. https://theecmconsultant.com/data-management-challenges/

Panel. (2023). Comparisons. https://panel.holoviz.org/about/comparisons.html

Sutchenkov, Anton. A., & Tikhonov, A. I. (2020). Embedding Interactive Python Web Applications into Electronic Textbooks. 2020 V International Conference on Information Technologies in Engineering Education (Inforino), 1–4. https://doi.org/10.1109/Inforino48376.2020.9111663

Wexler, S., Shaffer, J., & Cotgreave, A. (2017). The Big Book of Dashboards. John Wiley & Sons, Inc.

7. Lessons Learned

As is expected in a project of this size, not everything went according to plan. Several issues arose during the execution of this project. I will explain what occurred in detail in Section 9 of this report, but for the purposes of this section, I want to highlight the main takeaways I gained during the execution of this project with regards to planning, meeting deadlines, and finding alternate solutions.

7.1 Underestimating/overestimating difficulty

When creating the plan for this project, I think I did a good job of anticipating which tasks I would have to complete and in which order. However, I have learned that it is more difficult to anticipate how difficult a specific task will be. For example, one of the first tasks I had to complete during this project was creating the dummy data that would be used in both the database and the dashboard. Initially, I believed that this would be very easy. What I failed to anticipate was that there is no straight-forward way of randomly generating sales data which is believable and resembles real sales data. As such, this task took significantly longer than expected. I believe that this can be solved in future projects by doing more research beforehand on each specific task that will be completed.

7.2 Things outside of your control

Given that I had used Oracle's relational database management system extensively throughout the MASY program, I decided that I would build the database using this platform. However, I came across an issue where, due to my Apple computer's architecture, I could not use the specific Python package that is used to connect Oracle databases using Python. I did not anticipate this obstacle when planning this project, and it was completely outside of my control. As I will explain in Section 9 of this report, I had to find alternative solutions. Even though this worked out perfectly in the end, it could have set me back weeks, or could have completely halted the execution of this project. I think this can also be solved in the future by doing more research, as well as allocating more time for tasks as essential as this one.

7.3 Learning on the go

Even though I had learned how to create web applications during the MASY program, I did not have prior experience with Plotly Dash. Nonetheless, after doing some research, I decided that creating the dashboard using this tool would be achievable and would be an excellent skill to learn. As I expected, the tool was intuitive enough for me to learn how to build great dashboards within a relatively short period of time, and there is a plethora of videos, guides, and books on the internet which made this task possible. However, in future

projects I think I should be more careful when deciding to learn a new skill on the go in projects as large as this one.

8. Sponsor Acceptance Form

Sponsor Acceptance Form
Fall 2022

The purpose of this document is to acknowledge completion of the student's project on behalf of the client and sponsor.

Project: Optimizing Biotech operations through Database Management & Data Analysis

To be completed by the student:

Name of Student: Carlos Olivella

I have submitted my completed project to the sponsor and reviewed with him.

Signature of student: Carlos Olivella

To be completed by the sponsor:

Name of sponsor: Sebastian Lora

Please check one of the following:

✓ The project is completed to my satisfaction
_____ The project is not fully complete, but it is sufficient for our purpose
_____ The project has a lot of incomplete work and/or is not usable for me

Signature of sponsor: Sebastian Lora

9. Project Chronology

This section will include a detailed description of the tasks completed during this project, any obstacles or issues that arose, along with the insights that I gained.

9.1 Generating the dummy data

Due to the confidential nature of its data, the Client was only able to provide a CSV file containing five (5) rows and thirty-three (33) columns of sales data including values such as text, numbers, dates, and null values. As such, I had to generate an additional 4,995 rows of dummy data to be used throughout the project (hereafter, 'Dummy Data' or 'DD') using several of Microsoft Excel's built-in functions.

Before generating the Dummy Data, I decided that I would create a data dictionary to describe the contents of each column. These descriptions would include any guidelines communicated by the Client as to what each column should and should not include. Each column would be of a specific data type.

- Columns that contain text values will be characterized as strings.
- Columns that contain whole numbers will be characterized as integers.
- Columns that contain decimal values will be characterized as floats.

9.1.1 Data Dictionary

The data dictionary contains three elements: (i) the column name; (ii) the column's datatype in parenthesis; and (iii) a description of the data that the column will contain and whether or not the column can be null.

- Order ID (integer): A unique number that identifies each order made by a customer. In the event that a customer orders more than one product in a given order, each line item in that order is considered an order itself. As such, no Order ID is repeated. Cannot be null.
- Order Date (date): The date on which the order was made by the customer. Cannot be null.
- Shipping Date (date): The date on which the order was shipped by the Client to the customer. Can be null when Status is 'cancelled'.
- Fulfilled Date (date): The date on which the order was delivered to the customer. Can be null when Status is 'cancelled'.
- Status (string): The status of the order made by a customer. Can correspond to 'fulfilled', 'cancelled', or 'refunded'. Cannot be null.
- Product ID (integer): A unique number that identified each product sold by the Client. Cannot be null.

- Category (string): The type of product ordered by the customer. Can correspond to 'incinerators', 'shoe stations', and 'solutions'. Cannot be null.
- Product SKU (string): The name that is given to a specific product sold by the Client. Cannot be null.
- Unit Cost (float): The cost of producing a single unit of the specific product. Cannot be null.
- Price (float): The price at which the specific product is sold by the Client. Cannot be null.
- Quantity Ordered (integer): The quantity of the specific product ordered by the customer. Cannot be null.
- Total Amount (float): The amount specified in the Price column times the Quantity Ordered. Cannot be null.
- Discount Percent (number): The percent discount applied to the order. Cannot be null.
- Discount Amount (float): The Total Amount times the Discount Percent. Cannot be null.
- Invoiced Amount (float): The Total Amount minus the Discount Amount. Cannot be null.
- Payment (string): The payment method used by the customer. Cannot be null.

- Customer ID (integer): A unique number that identifies the customer that made the order. Cannot be null.
- Customer Type (string): The type of customer that made the order. Can correspond to 'entity' or 'individual'. Cannot be null.
- Corporate Name (string): The customer's corporate name. Must be null if Customer Type is 'individual'.
- First Name (string): The customer's first name. Legal representatives of an entity can be included here. Can be null if Customer Type is 'entity'.
- Middle Initial (string): The customer's middle initial. Can be null.
- Last Name (string): The customer's last name. Can be null if Customer Type is 'entity'.
- Gender (string): The customer's gender. Can be null if Customer Type is 'entity'.
- Email (string): The customer's email. Cannot be null.
- Customer Since (date): The date on which the customer first made an order. Cannot be null.
- SSN (string): The customer's Social Security Number. Can be null.
- Phone (string): The customer's phone number. Cannot be null.
- Street (string): The customer's street address. Cannot be null.
- City (string): The city in which the customer is located. Cannot be null.
- State (string): The state in which the customer is located. Cannot be null.

- Country (string): The country in which the customer is located. Cannot be null.
- Zip Code (string): The zip code in which the customer is located. Cannot be null.
- Region (string): The region in which the customer is located. Can be null if Country is not 'USA'.

9.1.2 Generating the data

As mentioned above, the dummy data was created using some of Microsoft Excel's built-in functions. These were very helpful in allowing me to establish certain parameters within which the data would be randomly generated. The process that I decided to use for each column is described as follows:

- Order ID: This is a sequence of numbers from 1 to 5000.
- Order Date: The first Order Date in this column was '2015-01-01'. To generate the rest of the Order Date column, I used Excel's RANDBETWEEN() function. This function generates a random number between the two numbers specified by the user inside the parenthesis. First, created a separate column where I generated a random number between 0 and 1. This number was then added to the first Order Date, and this step was repeated for all 5000 rows. This ensured that some days had several orders, while other days might have only 1 order.

- Shipping Date: To generate the Shipping Date, I used the same process that I used for the Order Date. The only difference is that I generated a random number between 0 and 2, given that the Client can take between 0 and 2 days to ship a product.
- Fulfilled Date: To generate the Fulfilled Date, I used the same process used in the Shipping Date column. The only difference is that the random number was between 2 and 7. This is because, according to the Client, this is the average period of days that an order takes to be delivered.
- Status: The same method used for the date columns could not be used here because the RANDBETWEEN function generates a relatively equal distribution of the values included in it. Therefore, approximately 33.3% of rows would be 'fulfilled', 33.3% would be 'cancelled', and 33.3% would be 'refunded'. This is not realistic. As such, to generate these values I decided to generate a random number between 1 and 5000 in each row. Then I used Excel's INDEX and MATCH functions to distribute the three status categories among the rows, based on specific weights that I inputted into the function. I decided that 88% (4400) of the rows would be 'fulfilled', 7% (750) would be 'cancelled', and 5% (50) would be 'refunded'. The resulting function was the following:

$$= INDEX(\{"fulfilled","cancelled","refunded"\}, MATCH(B2,\{1,4401,4751\},1))$$

In reality, generating the values in this way had no real impact given that I will be working with dummy data. However, my intention was to make the data resemble reality as much as I could.

- Product ID: I used a list of ten items found on the Client's (www.biosecag.com) website as products in this case.

item_id	item_category	item_name	unit_cost	price	weights
1	shoe stations	uvo3 shoe station	754.44	1599.99	0.12
2	solutions	quartzseal	7.82	32.99	0.23
3	solutions	biologic sr2	121.55	199.99	0.18
4	solutions	ceramyc guard	15.76	49.99	0.21
5	incinerators	ais 33 cyclone	4550.00	6990.90	0.12
6	incinerators	ais 40 cyclone	6780.32	8799.99	0.05
7	incinerators	ais 55 cyclone	8450.00	11990.90	0.04
8	incinerators	ais 132 cyclone	10500.00	12499.99	0.02
9	incinerators	ais 1250 cyclone	15760.00	18799.99	0.02
10	incinerators	ais 3860 cyclone	18540.00	22499.99	0.01

Fig. 9-1

To distribute all ten products among the 5000 rows, I used the same method used for the Status column. In this case, the weights used can be seen in the image above. I used the following formula to generate the values:

$$= INDEX(\{1,2,3,4,5,6,7,8,9,10\}, MATCH(F2, \{1,601,1751,2651,3701,4301,4551,4751,4851,4951\}, 1))$$

- Category: These values were generated using a simple VLOOKUP() function that looks up the ID in the first column (image above) that corresponds with the ID in the Product ID column of the Dummy Data, and returns the value in the item_category column.
- Product SKU: These values were generated using a simple VLOOKUP() function that looks up the ID in the first column (image above) that corresponds with the ID in the Product ID column of the Dummy Data, and returns the value in the item_name column.
- Unit Cost: These values were generated using a simple VLOOKUP() function that looks up the ID in the first column (image above) that corresponds with the ID in the Product ID column of the Dummy Data, and returns the value in the unit_cost column.
- Price: These values were generated using a simple VLOOKUP() function that looks up the ID in the first column (image above) that corresponds with the ID in the Product ID column of the Dummy Data, and returns the value in the price column.
- Quantity Ordered: I used the RANDBETWEEN() function to generate random values for the quantities. However, it is not realistic for 300 of the highest-priced incinerators to be ordered in a single order, or for a customer to order a single unit of the cheapest product, given that these are industrial products which are normally

sold in bulk. As such, I created specific ranges to include in the function, depending on the product. The ranges are as follows:

  - Product 1: 1 to 15
  - Product 2: 3 to 300
  - Product 3: 5 to 50
  - Product 4: 2 to 200
  - Product 5: 1 to 6
  - Product 6: 1 to 6
  - Product 7: 1 to 5
  - Product 8: 1 to 4
  - Product 9: 1 to 3
  - Product 10: 1 to 2

- Total Amount: This column was generated by multiplying the Price times the Quantity Ordered.
- Discount Percent: I used a combination of the IF() function and different ranges to produce the discount values. The ranges are:

- 1 to 4 units: 0%
- 5 to 10 units: 3%
- 11 to 50 units: 5%
- 51 to 100 units: 7%
- 101 to 200 units: 8.5%
- 201+ units: 10%

The final function was the following:

$$= \text{IF}(\text{AND}(\text{K2} \geq 1, \text{K2} < 5),0, \text{IF}(\text{AND}(\text{K2} \geq 5, \text{K2} < 11), .03, \text{IF}(\text{AND}(\text{K2} \geq 11, \text{K2} < 51), .05, \text{IF}(\text{AND}(\text{K2} >= 51, \text{K2} < 101), .07, \text{IF}(\text{AND}(\text{K2} >= 101, \text{K2} < 201), .085, .1)))))$$

- Discount Amount: These values are the result of multiplying the Total Amount by the Discount Percent.
- Invoiced Amount: These values are the result of subtracting the Discount Amount from the Total Amount.
- Payment: The same method used for Status and Product ID was used in this case.
- Customer ID: I created a list of ten hypothetical customers based on the list of 40 Pork Powerhouses across the United States.

cust_id	cust_type	corporate_name	first_name	middle_initial	last_name	weights
1	entity	Smithfield Foods				0.10
2	entity	Holden Farms				0.32
3	individual		John		Atkins	0.03
4	entity	Prestage Farms				0.07
5	entity	Kalmback Feeds				0.08
6	individual		Chris	E.	Smalling	0.13
7	entity	Schwartz Farms				0.04
8	entity	Seaboard Foods				0.10
9	entity	Carthage System				0.05
10	entity	VMC Management				0.08

Fig. 9-2

The process used for this column is the same as the one that I used for Status and Product ID. The formula is:

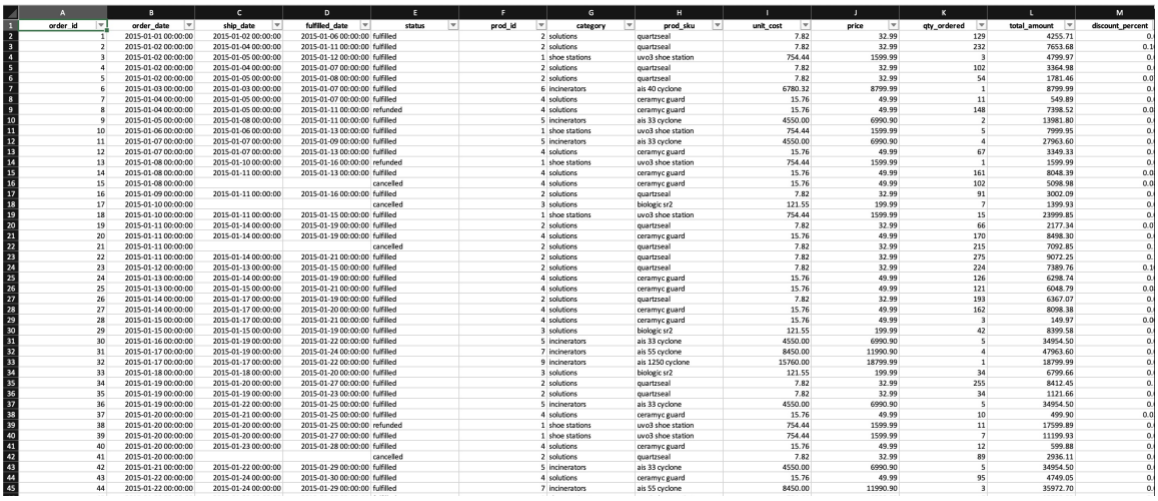
$$= \text{INDEX}(\{1,2,3,4,5,6,7,8,9,10\},$$

$$\text{MATCH}(\text{R2}, \{1,601,1751,2651,3701,4301,4551,4751,4851,4951\},1))$$

- Customer Type: To generate these values I used a VLOOKUP() function that looks up the Customer ID and returns the Customer Type.
- Corporate Name: The same process used for Customer Type was used here.
- First Name: The same process used for Customer Type was used here.
- Middle Initial: The same process used for Customer Type was used here.
- Last Name: The same process used for Customer Type was used here.
- Gender: The same process used for Customer Type was used here.

- Email: The same process used for Customer Type was used here.
- Customer Since: The same process used for Customer Type was used here.
- SSN: The same process used for Customer Type was used here.
- Phone: The same process used for Customer Type was used here.
- Street: The same process used for Customer Type was used here.
- City: The same process used for Customer Type was used here.
- State: The same process used for Customer Type was used here.
- Country: The same process used for Customer Type was used here.
- Zip Code: The same process used for Customer Type was used here.
- Region: The same process used for Customer Type was used here.

As a result, the final dummy data with 5000 rows and 33 columns looks like this:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	order_id	order_date	ship_date	fulfilled_date	status	prod_id	category	prod_sku	unit_cost	price	qty_ordered	total_amount	discount_percent
2	1	2015-01-01 00:00:00	2015-01-02 00:00:00	2015-01-06 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	129	4255.71	0.
3	2	2015-01-02 00:00:00	2015-01-04 00:00:00	2015-01-11 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	232	7653.68	0.1
4	3	2015-01-02 00:00:00	2015-01-05 00:00:00	2015-01-12 00:00:00	fulfilled	1	shoe stations	uvo3 shoe station	754.44	1599.99	3	4799.97	0.
5	4	2015-01-02 00:00:00	2015-01-04 00:00:00	2015-01-07 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	102	3364.98	0.
6	5	2015-01-02 00:00:00	2015-01-05 00:00:00	2015-01-08 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	54	1781.46	0.0
7	6	2015-01-03 00:00:00	2015-01-03 00:00:00	2015-01-07 00:00:00	fulfilled	6	incinerators	ais 40 cyclone	6780.32	8799.99	1	8799.99	0.
8	7	2015-01-04 00:00:00	2015-01-05 00:00:00	2015-01-07 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	11	549.89	0.
9	8	2015-01-04 00:00:00	2015-01-05 00:00:00	2015-01-11 00:00:00	refunded	4	solutions	ceramyc guard	15.76	49.99	148	7398.52	0.0
10	9	2015-01-05 00:00:00	2015-01-08 00:00:00	2015-01-11 00:00:00	fulfilled	5	incinerators	ais 33 cyclone	4550.00	6990.90	2	13981.80	0.
11	10	2015-01-06 00:00:00	2015-01-06 00:00:00	2015-01-13 00:00:00	fulfilled	1	shoe stations	uvo3 shoe station	754.44	1599.99	5	7999.95	0.
12	11	2015-01-07 00:00:00	2015-01-07 00:00:00	2015-01-09 00:00:00	fulfilled	5	incinerators	ais 33 cyclone	4550.00	6990.90	4	27963.60	0.
13	12	2015-01-07 00:00:00	2015-01-07 00:00:00	2015-01-13 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	67	3349.33	0.
14	13	2015-01-08 00:00:00	2015-01-10 00:00:00	2015-01-16 00:00:00	refunded	1	shoe stations	uvo3 shoe station	754.44	1599.99	1	1599.99	0.
15	14	2015-01-08 00:00:00	2015-01-11 00:00:00	2015-01-13 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	161	8048.39	0.0
16	15	2015-01-08 00:00:00			cancelled	4	solutions	ceramyc guard	15.76	49.99	102	5098.98	0.0
17	16	2015-01-09 00:00:00	2015-01-11 00:00:00	2015-01-16 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	91	3002.09	0.
18	17	2015-01-10 00:00:00			cancelled	3	solutions	biologic sr2	121.55	199.99	7	1399.93	0.
19	18	2015-01-10 00:00:00	2015-01-11 00:00:00	2015-01-15 00:00:00	fulfilled	1	shoe stations	uvo3 shoe station	754.44	1599.99	15	23999.85	0.
20	19	2015-01-11 00:00:00	2015-01-14 00:00:00	2015-01-19 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	66	2177.34	0.0
21	20	2015-01-11 00:00:00	2015-01-14 00:00:00	2015-01-19 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	170	8498.30	0.
22	21	2015-01-11 00:00:00			cancelled	2	solutions	quartzseal	7.82	32.99	215	7092.85	0.
23	22	2015-01-11 00:00:00	2015-01-14 00:00:00	2015-01-21 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	275	9072.25	0.
24	23	2015-01-12 00:00:00	2015-01-13 00:00:00	2015-01-15 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	224	7389.76	0.1
25	24	2015-01-13 00:00:00	2015-01-14 00:00:00	2015-01-19 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	126	6298.74	0.
26	25	2015-01-13 00:00:00	2015-01-15 00:00:00	2015-01-21 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	121	6048.79	0.0
27	26	2015-01-14 00:00:00	2015-01-17 00:00:00	2015-01-19 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	193	6367.07	0.
28	27	2015-01-14 00:00:00	2015-01-17 00:00:00	2015-01-20 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	162	8098.38	0.
29	28	2015-01-15 00:00:00	2015-01-17 00:00:00	2015-01-21 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	3	149.97	0.0
30	29	2015-01-15 00:00:00	2015-01-15 00:00:00	2015-01-19 00:00:00	fulfilled	3	solutions	biologic sr2	121.55	199.99	42	8399.58	0.
31	30	2015-01-16 00:00:00	2015-01-19 00:00:00	2015-01-22 00:00:00	fulfilled	5	incinerators	ais 33 cyclone	4550.00	6990.90	5	34954.50	0.
32	31	2015-01-17 00:00:00	2015-01-19 00:00:00	2015-01-24 00:00:00	fulfilled	7	incinerators	ais 55 cyclone	8450.00	11990.90	4	47963.60	0.
33	32	2015-01-17 00:00:00	2015-01-17 00:00:00	2015-01-22 00:00:00	fulfilled	9	incinerators	ais 1250 cyclone	15760.00	18799.99	1	18799.99	0.
34	33	2015-01-18 00:00:00	2015-01-18 00:00:00	2015-01-20 00:00:00	fulfilled	3	solutions	biologic sr2	121.55	199.99	34	6799.66	0.
35	34	2015-01-19 00:00:00	2015-01-20 00:00:00	2015-01-27 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	255	8412.45	0.
36	35	2015-01-19 00:00:00	2015-01-19 00:00:00	2015-01-23 00:00:00	fulfilled	2	solutions	quartzseal	7.82	32.99	34	1121.66	0.
37	36	2015-01-19 00:00:00	2015-01-22 00:00:00	2015-01-25 00:00:00	fulfilled	5	incinerators	ais 33 cyclone	4550.00	6990.90	5	34954.50	0.
38	37	2015-01-20 00:00:00	2015-01-21 00:00:00	2015-01-25 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	10	499.90	0.0
39	38	2015-01-20 00:00:00	2015-01-20 00:00:00	2015-01-25 00:00:00	refunded	1	shoe stations	uvo3 shoe station	754.44	1599.99	11	17599.89	0.
40	39	2015-01-20 00:00:00	2015-01-20 00:00:00	2015-01-27 00:00:00	fulfilled	1	shoe stations	uvo3 shoe station	754.44	1599.99	7	11199.93	0.
41	40	2015-01-20 00:00:00	2015-01-23 00:00:00	2015-01-28 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	12	599.88	0.
42	41	2015-01-20 00:00:00			cancelled	2	solutions	quartzseal	7.82	32.99	89	2936.11	0.
43	42	2015-01-21 00:00:00	2015-01-22 00:00:00	2015-01-29 00:00:00	fulfilled	5	incinerators	ais 33 cyclone	4550.00	6990.90	5	34954.50	0.
44	43	2015-01-22 00:00:00	2015-01-24 00:00:00	2015-01-30 00:00:00	fulfilled	4	solutions	ceramyc guard	15.76	49.99	95	4749.05	0.
45	44	2015-01-22 00:00:00	2015-01-24 00:00:00	2015-01-29 00:00:00	fulfilled	7	incinerators	ais 55 cyclone	8450.00	11990.90	3	35972.70	0.

Fig. 9-3

9.2 Creating the data model

Once I generated the Dummy Data, I moved on to creating the data model which I would later use to create the database. In order to proceed with creating the data model, I contacted the Client so that we could agree upon the final columns that the data model would include. The data that is imported into the data model originates from the Client's sales data. Therefore, it was important to ask if they foresaw that the data they had sent me would continue to include the same columns in the future, or if they believed any changes would be made to this data in upcoming periods. The Client informed me that they did not expect any changes to be made to their sales data in upcoming months and, as such, I could proceed with building the data model as is.

Based on the columns in the Dummy Data, this was the proposed database schema:

ORDER	PRODUCT	CUSTOMER	STATUS	CATEGORY	PAYMENT_METHOD	CUSTOMER_TYPE
order_pk	prod_pk	cust_pk	status_pk	category_pk	payment_pk	cust_type_pk
order_date	prod_sku	corp_name	status	category	payment	cust_type
ship_date	unit_cost	first_name				
fulfilled_date	price	middle_initial				
qty_ordered	category_fk	last_name				
total_amount		email				
discount_percent		cust_since				
discount_amount		ssn				
invoiced_amount		phone				
prod_fk		cust_type_fk				
cust_fk		gender_fk				
payment_fk						
address_fk						
status_fk						

GENDER	ADDRESS	ZIP_CODE	CITY	STATE	REGION	COUNTRY
gender_pk	address_pk	zip_code_pk	city_pk	state_pk	region_pk	country_pk
gender	street	zip_code	city	state	region	country
	zip_code_fk	city_fk	state_fk	region_fk	country_fk	

Fig. 9-4

Each cell highlighted in yellow represents the name of an entity (or table) within the database. Each cell below these is an attribute of that particular entity. Cells highlighted in green represent primary keys, while cells highlighted in red represent foreign keys.

Once this proposal was finalized, I created the data model using Oracle Data Modeler. The final data model can be seen below:

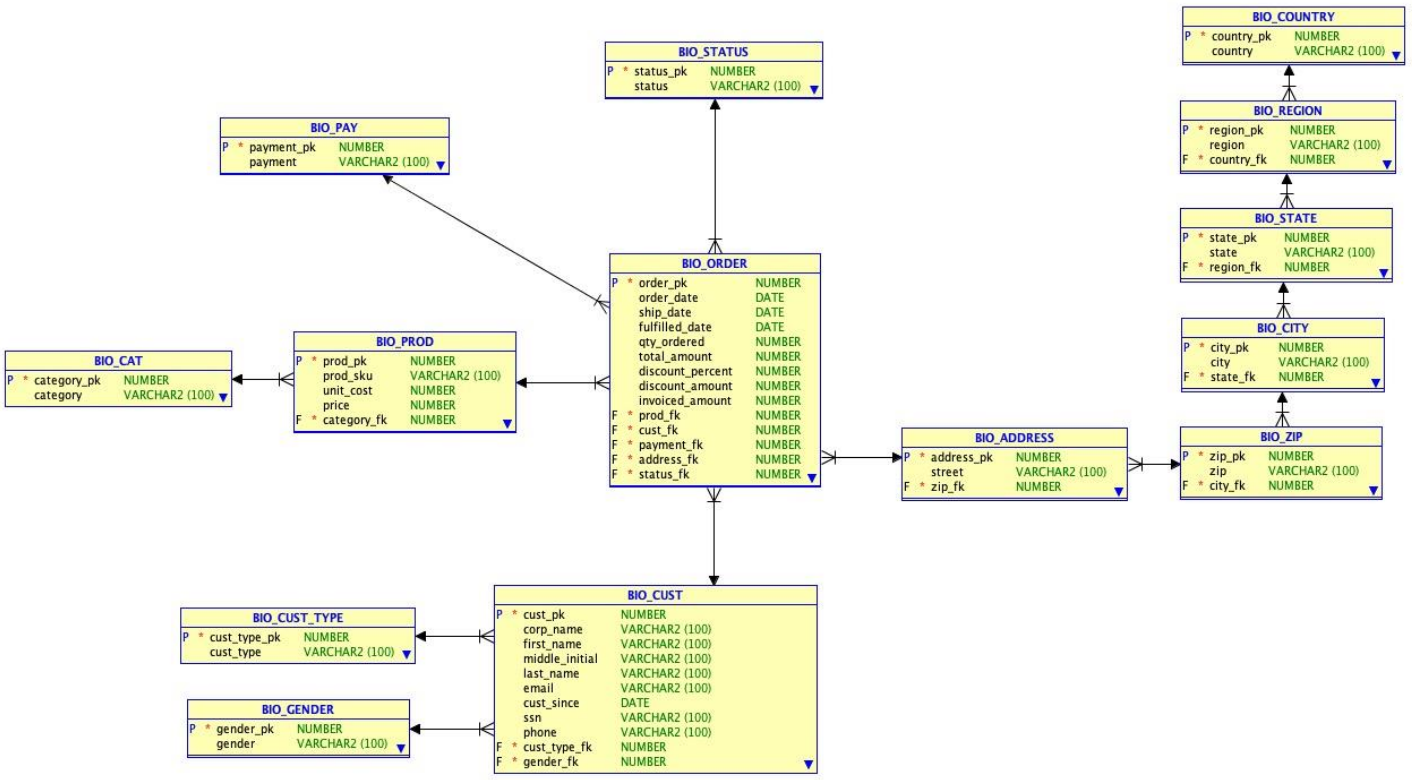


Fig. 9-5

9.3 Creating the database

Once the data model was finalized, I moved on to one of the essential tasks within the project: creating the database. As mentioned in Section 7, this proved to be more difficult than I initially thought. The plan was to use Oracle's relational database management system to create, store, and manage the database. However, there were several issues when connecting to Oracle using Python.

9.3.1 Issues with cx_Oracle

First, the Python library that is used to connect to Oracle is called cx_Oracle. The connection to a database such as Oracle is usually straightforward, as long as the user has the correct credentials. However, in my case, the use of this library proved to be an obstacle due to the specific computer I used to complete this project . The cx_Oracle library requires the use of a computer that has a 'x86_64' architecture. The 2021 M1 Apple Macbook Air that I used is built on a different architecture, therefore, when running a script that uses cx_Oracle in it, the script would not run. Normally, this can be solved by running the script as follows:

$$arch\ -x86\_64\ python3\ \{script\ name\}$$

Nonetheless, in this case, this created a different problem. Running a script with the inclusion of 'arch -x86_64' in the command does not allow the script to run if the Pandas library is imported inside the script. These two problems together made it impossible to create an app as I intended, given that my intention was for the app to use both Pandas and cx_Oracle within the same script. For this reason, I was obligated to find an alternative to Oracle.

9.3.2 MySQL

After doing some research, along with the help of my professor, I identified that MySQL could be an excellent alternative. MySQL is an *"Open-Source SQL database management system developed, distributed, and supported by Oracle Corporation"*[1].

One of the disadvantages of using Oracle SQL Developer was that I needed to install a client on my computer, which would then connect to Oracler's servers. The main function of the cx_Oracle Python library was, indeed, to be the bridge between the client I downloaded and Oracle's servers. Because I was not able to use cx_Oracle, I therefore was not able to connect to Oracle's servers and databases. The use of MySQL is beneficial in this sense because it does not require the user to install a client to connect to a server. Instead, the user's own computer can act as the server, meaning that MySQL can run easily on a laptop.

Another benefit of MySQL is that it does not require the 'x86_64' architecture, meaning that I was able to connect to the database and use Pandas in the same script. The following screenshot demonstrates a successful connection to the database:

---

[1] MySQL (2023). 1.2.1 What is MySQL?. MySQL 8.0 Reference Manual. Retrieved on April 19, 2023, from: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html.

```
import mysql.connector as connection

mydb = connection.connect(
  host="localhost",
  user="root",
  password="Alexandra2005"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT 'The connection has been successful'")

for _ in mycursor:
    print(_)
```

```
('The connection has been successful',)
```

Fig. 9-6

In order to proceed with this change, I asked the Client if they had any preference regarding the RDMBS platform used in the project. The Client informed me that their knowledge of RDMBS platforms was very limited. Once I explained the issue that had occurred, along with the benefits of using MySQL, the Client was happy to agree to the modification.

9.3.3 Importing the data

The next step was to import the dummy data into a stage table in MySQL. There is no reason to do this using Python, as MySQL has a Table Data Import Wizard which makes the process very simple.

When importing the dummy data, I encountered two issues. First, the way dates were initially formatted in the dummy data was not compatible with MySQL dates and, as such, would not import correctly. Because of this, I had to change the format from "YYYY-MM-DD" to "YYYY-MM-DD HH:MM:SS", to include hours, minutes, and seconds. After doing so, the import was successful. The second issue I had was that, for some reason, trying to query or preview the new table would cause MySQL to crash. After doing some research, I came across a forum[2] where several individuals state that they experienced the same issue. Their solution was to downgrade MySQL Workbench to version 8.0.31. This proved to be correct, and once I downgraded the version, the program did not crash again.

Querying the data also proved to be an issue. One of the libraries that is required to connect MySQL to Python and query data is the Python-MySQL Connector. I was able to import

[2] Anshgupta (2023). *MySql workbench is crashing*. Apple Developer Forums. Retrieved on April 19, 2023, from: https://developer.apple.com/forums/thread/724378.

this library successfully and query the data, but I could not convert the data into a Pandas dataframe. After doing some research, I found a forum on StackOverflow mentioning that SQLAlchemy was needed to execute the queries, read the SQL, and convert the results to a Pandas dataframe.

9.3.4 SQLAlchemy

SQLAlchemy *"is a library that facilitates the communication between Python programs and databases"*[3]. In basic terms, SQLAlchemy acts as a translator, converting Python classes and function calls into SQL statements that are sent to a database.

Once I switched to using SQLAlchemy, querying the data was significantly easier. However, there were still several ways of doing so.

```
# Method 1 - longer, and doesn't have the 'with' clause that automatically closes the connection

from sqlalchemy import create_engine, text
import pymysql
import pandas as pd


sqlEngine = create_engine('mysql+pymysql://root:Alexandra2005@localhost/bio')

dbConnection = sqlEngine.connect()

frame           = pd.read_sql(text("select * from bio_stage limit 5"), dbConnection)

pd.set_option('display.expand_frame_repr', False)

dbConnection.close()

frame
```

Fig. 9-7

[3] Krebs, B. (2017, Nov 09). *SQLAlchemy ORM Tutorial for Python Developers*. Auth0 Blog. Retrieved on April 19, 2023, from: https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/

Fig. 9-8

As we can see in the images above, method 1 required more lines of code to query the stage table. It also did not close the database connection automatically. This might not be a significant issue in my case, given that my database is small. However, in corporate applications, failing to close a connection to the database can create high costs for the company.

9.3.5 Creating the database

Once the stage table was imported successfully and I was able to query the data, it was time to create the normalized tables that comprise the final database. This was achieved by running a single SQL script (see Section **Error! Reference source not found.**). The following image show that the creation of the transactional database was successful, along with the tables that were created on the tab on the left.

9.4 Populating the database

The next step in the process was to populate the database by running a SQL script to transfer the data from the stage table into each of the normalized tables created in Section 9.3.

```
#####################################################################

##### POPULATING COUNTRY TABLE #####

INSERT INTO bio_country(country)
SELECT DISTINCT country
FROM bio_stage
WHERE   bio_stage.country IS NOT NULL
        AND NOT EXISTS (SELECT country FROM bio_country);

# SELECT * FROM bio_country;

#####################################################################

##### POPULATING REGION TABLE #####

INSERT INTO bio_region(region, country_fk)
SELECT DISTINCT s.region region, c.country_pk country_fk
FROM bio_stage s, bio_country c
WHERE   s.country = c.country
        AND s.region IS NOT NULL
        AND NOT EXISTS (SELECT region FROM bio_region);

# SELECT * FROM bio_region;

#####################################################################

##### POPULATING STATE TABLE #####

INSERT INTO bio_state(state, region_fk)
SELECT DISTINCT s.state state, r.region_pk region_fk
FROM bio_stage s, bio_region r
WHERE   s.region = r.region
        AND s.state IS NOT NULL
        AND NOT EXISTS (SELECT state FROM bio_state);

# SELECT * FROM bio_state;
```

Fig. 9-9

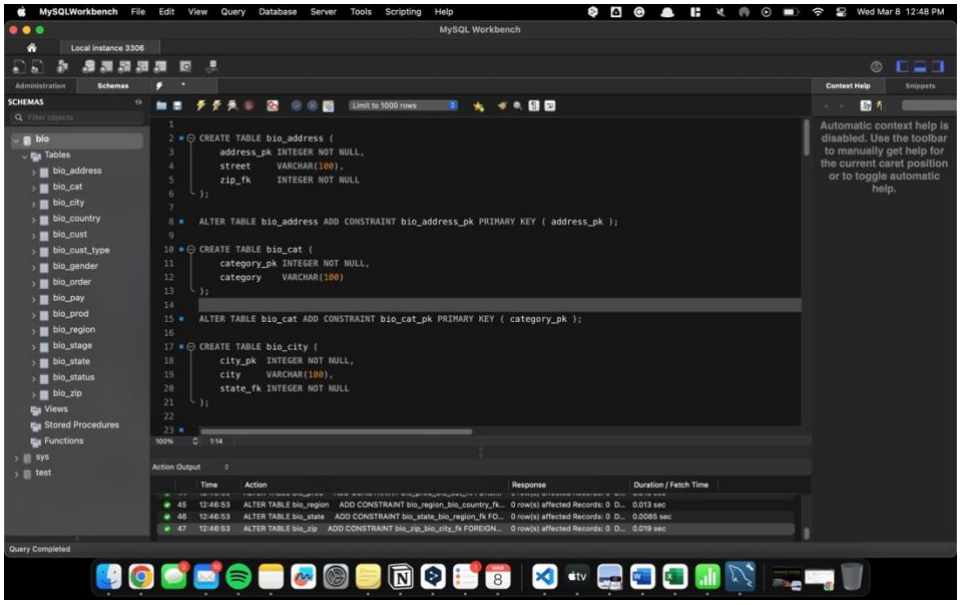


Fig. 9-10

This process was successful, and no issues arose during its execution. The full script that I used can be seen in Section 0 of this report.

9.5 Client Approval

Once I had finished creating the database, I had a brief meeting with the Client where I demonstrated how it worked and the value that it could provide to them. The process consisted of going through a test run where, after I had deleted all the data from the database, I imported the Dummy Data once again, populated the normalized tables, and demonstrated the ease with which the Client could make changes to the database through the use of SQL. I also explained the benefits of using a platform such as MySQL, with all of the security benefits that it possesses. The Client was very impressed with the database and mentioned that this would provide a great deal of relief to their team given that they

would no long have to worry about where they should store their data, along with the possibility of losing it. As such, they informed me that I could move on to the next phase of the project.

9.6 Creating the dashboard

At this point, the database was finalized, and all bugs were solved. Therefore, the next step was to create the dashboard which would display the data contained in the database. To create the dashboard, I decided to use the Plotly Dash toolkit. I will not go into the reasons why I chose Plotly Dash over other options, as this explanation can be seen in Section 6.4 of this report.

9.6.1 HTML

However, it is important to mention that Plotly Dash utilizes a wrapper for HTML so that apps can be built in Python. This allows the user to benefit from the modularity and specificity that HTML offers when creating apps. Specifically, it is very helpful in creating custom layouts, color schemes, and graphs. The following image is a preview of the code I wrote to create the layout (the complete script can be seen in Section 10.3 of this report):

```
# App Layout
app.layout = dbc.Container(
    html.Div([
    dbc.Card(
        dbc.CardBody([
            # 1st Row
            dbc.Row(
                # Title
                dbc.Col(
                    html.Div([
                        dbc.Card(
                            dbc.CardBody([
                                html.Div([
                                    html.H1("BIOSEC SALES",
                                            style={"font-weight": "bold", "color": "white"})
                                ], style={'textAlign': 'center'})
                            ])
                        ),
                    ]),
                    width={'size':6,'offset':3},
                    align='center'
                ),
                align='center'
            ),
            html.Br(),
            # 2nd Row
            dbc.Row([
                # Year Dropdown
                dbc.Col(
                    dcc.Dropdown(id='year_dropdown',
                                 options=[{'label':year,'value':year} for year in dropdown_years]),
                    width={'size':2,'offset':4}
                ),
                # Quarter Dropdown
                dbc.Col(
                    dcc.Dropdown(id='quarter_dropdown',
                                 options=[{'label':quarter,'value':quarter} for quarter in dropdown_quarters]),
                    width=2
                ),
```

Fig. 9-11

9.6.2 Bootstrap

One of the most important tools I used when creating the dashboard was Bootstrap, *"a free, open-source front-end development framework for the creation of websites and web apps"*[4]. Bootstrap is built on HTML, CSS, and JavaScript, and includes the functionality

[4] Zola, A. (2023). *What is Bootstrap*. WhatIs.com. Retrieved on April 19, 2023, from: https://www.techtarget.com/whatis/definition/bootstrap.

that is required for apps to be fully responsive depending on whether the user is accessing them through a desktop, mobile phone, or tablet.

However, perhaps the most helpful aspect of Bootstrap is its grid system. This separates the window into 12 columns which can be modified to whatever layout the creator desires. As such, the grid system allows the creator to go very deep into customization when building an app. In my case, this functionality helped me greatly in designing my dashboard because I was able to make changes to the layout with only a few modifications to the code.

9.6.3 Choosing which metrics to include

The first step in designing the dashboard was to decide on which metrics and graphs to include. Given that I was working with sales data, there were a lot of options to choose from, including lag times, revenue breakdowns, pie charts, and map charts. The Client's input during this phase was essential. In the end, we decided to go with the following metrics:

- Dropdowns: Given that the idea was for this dashboard to be interactive, and considering that the sales data contains date parameters, I decided to include some dropdowns so that the user can modify the data that they want to see. After speaking with the Client and consulting their needs, we decided that the most appropriate

breakdown for the dashboard was for it to include two dropdowns: (i) the year; and (ii) the quarter. Including the year and quarter is important because it allows the user to limit the data that it sees to specific period of time. Even though the specific quarters were not included in the sales data provided by the Client, they mentioned that quarters are frequently used in corporate settings. As such, I used SQL to extract them from the order date:

```
with sqlEngine.connect() as con:
    dropdown_df1 = pd.DataFrame(con.execute(text("SELECT DISTINCT YEAR(order_date) col FROM bio_order;")).fetchall())
    dropdown_df2 = pd.DataFrame(con.execute(text("SELECT DISTINCT QUARTER(order_date) col FROM bio_order;")).fetchall())
    dropdown_df3 = pd.DataFrame(con.execute(text("SELECT DISTINCT UPPER(prod_sku) col FROM bio_prod ORDER BY UPPER(prod_sku);")).fetchall())

    dropdown_years = list(dropdown_df1['col'])
    dropdown_quarters = list(dropdown_df2['col'])
    dropdown_products = list(dropdown_df3['col'])
```

Fig. 9-12

The final result was as follows:

Fig. 9-13

The user can choose the specific year along with the specific quarter, and the dashboard will update in real time once the user clicks the submit button.

- Shipping Time: This metric represents the average number of days that it takes for an order to be shipped to the client. In other words, it is the average number of days

between the order date and the shipping date. The Client required this metric in their dashboard because logistics is one of their main priorities which, in their view, gives them a competitive advantage over its competitors. The Client mentioned that being able to see how long an order takes to be shipped, along with how this number fluctuates through time, allows them to make changes to their business model in hopes of optimizing shipping times.

We decided that this metric would be displayed as a simple decimal number representing the number of days. However, I also included some conditional formatting within this metric so that the number would be displayed in green if the number was less than 2, and red if the number was equal to or greater than 2. The following image is a preview of the metric:

Fig. 9-14

- Delivery Time: This metric represents the average number of days that it takes for an order to be fulfilled. In other words, it is the average number of days between the shipping date and the fulfillment date. The Client mentioned that this metric was key to them because it was their way of auditing their shipping service providers. The Client does not deliver its product itself. Instead, they hire a third-party to deliver the products to the end consumer. As such, this metric allows them to see the speed with which their products are being delivered. This is important because, no matter how efficient the Client is at producing and shipping the product, this can all be negated by the slow performance of a third-party provider.

  We decided that this metric would also be displayed as a simple decimal number representing the number of days. However, I also included some conditional formatting within this metric so that the number would be displayed in green if the number was less than 3, and red if the number was equal to or greater than 3. The following image is a preview of the metric:

Fig. 9-15

- Percentage of Cancelled Orders: The Client requested this metric because they wanted to visualize how many of its orders were actually being fulfilled. We decided that this metric would also be displayed as a decimal number which represents the percentage of orders that are cancelled by customers. Conditional formatting was also applied to this metric so that it is displayed in green when the percentage is less than 5%, and displayed in red when it is equal to or greater than 5%.

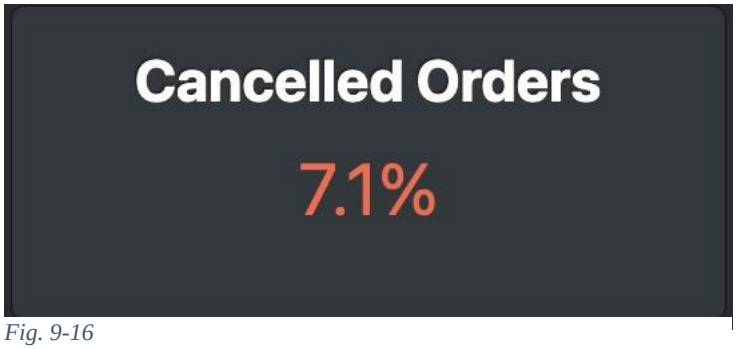


Fig. 9-16

- Percentage of Refunded Orders: The Client requested this metric because it would allow them to gauge how satisfied their customers were with their products. We decided that this metric would also be displayed as a decimal number which represents the percentage of orders that are refunded. Conditional formatting was also applied to this metric so that it is displayed in green when the percentage is less than 5%, and displayed in red when it is equal to or greater than 5%.

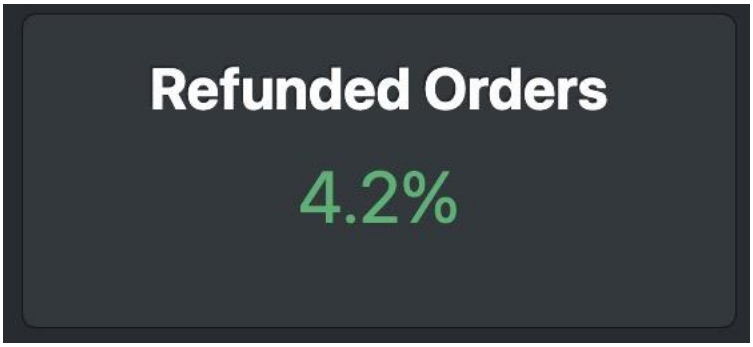


Fig. 9-17

- Revenue Breakdown per Quarter: The Client requested this metric because it would allow them to gauge how well their business is doing, and what their profit margins are. Initially, this metric would only include to total revenue. However, we decided that displaying the total revenue by itself was not enough. As such, we decided that this metric would be in the form of a stacked bar chart that displays the total revenue, along with a separation between the cost and the profit obtained in each quarter.

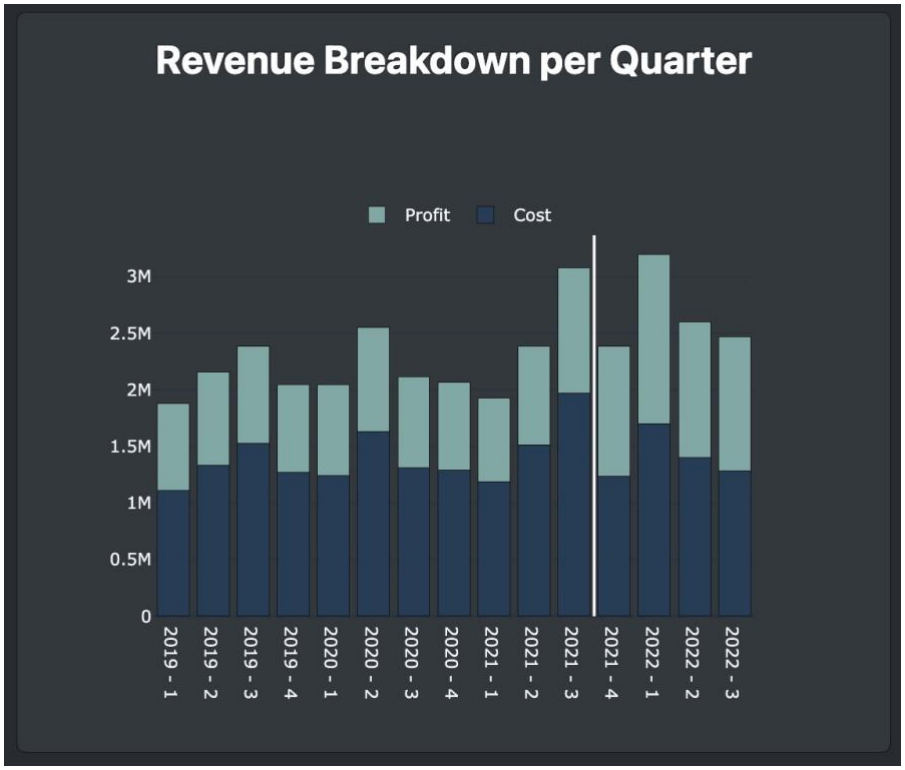


Fig. 9-18

As we can see, the chart displays each the total revenue obtained in each quarter, but it also displays the cost (in dark blue) and the profit (in light blue). The chart also displays a white line separating the last four quarters. I will go into detail about this when I explain the regression model.

- Number of Orders per Product: This metric displays the total number of orders per product in the form of a horizontal bar chart. It is a relatively simple chart. The

Client wanted to be able to visualize how well their products were selling, and how some products compared to others. I also included some conditional formatting for the color of the bars, depending on what category each product is in.

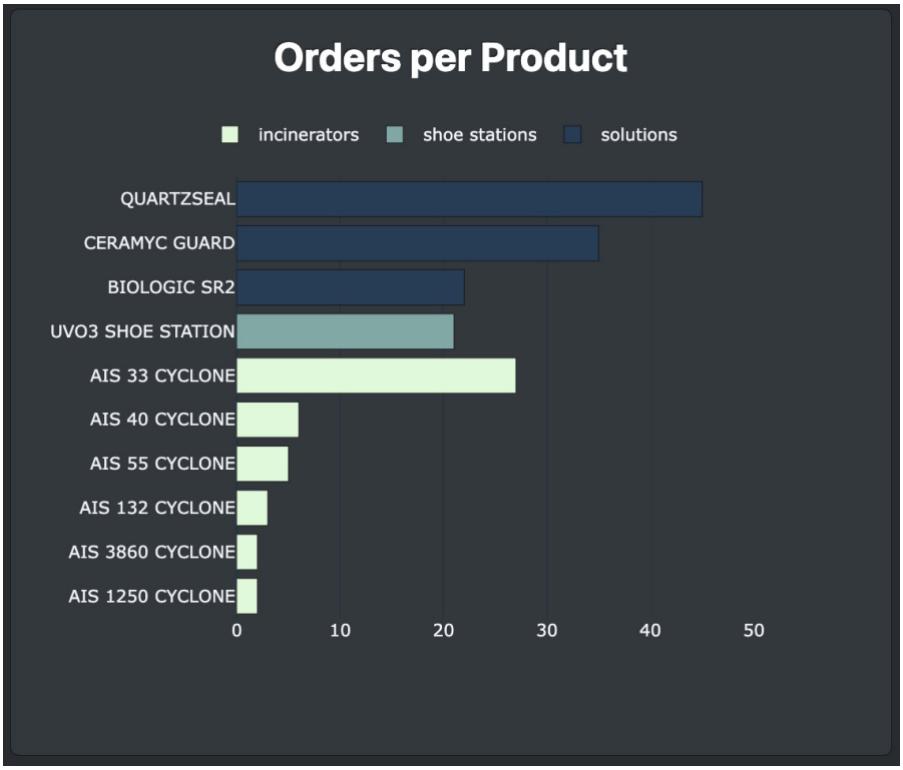


Fig. 9-19

- Distribution of Payment Methods: This chart displays the percentage of orders that were made with each of the payment methods offered by the Client in the form of a pie chart. The Client believes that this is important because it allows them to analyze if they should prioritize certain payment methods over others. For example,

card providers charge certain fees on transactions. The Client would prefer that its sales be paid through bank transfers, but most clients decide to pay with debit cards, as we can see in the graph.

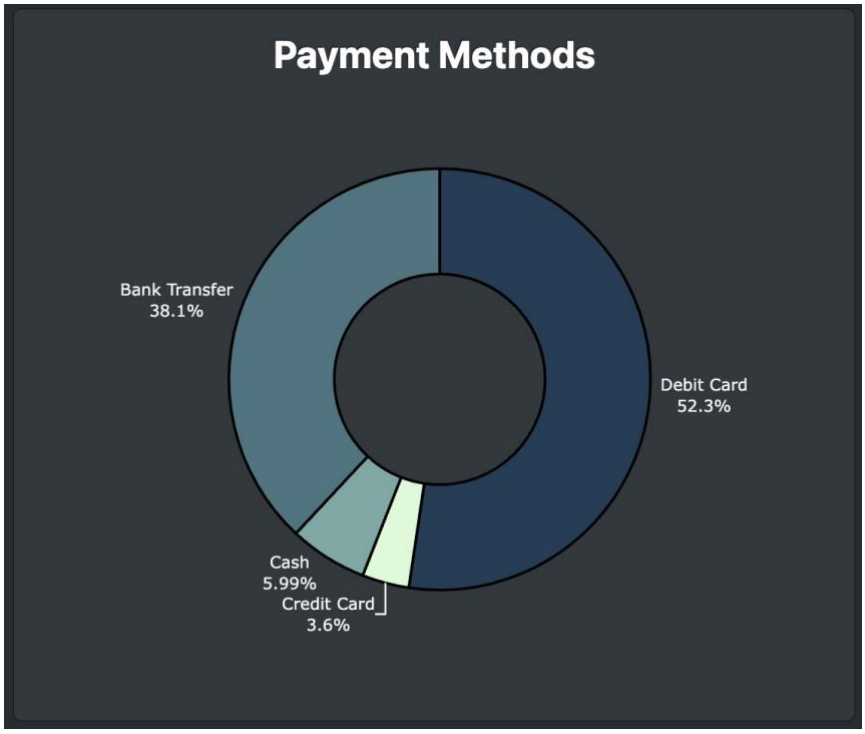


Fig. 9-20

- Distribution of Product Categories: This chart displays the percentage of products that were ordered within each of the product categories offered by the Client in the form of a pie chart. Similarly to the Orders per Product graph, the Client wanted to visualize how well each product category sells in order to determine what should be prioritized.

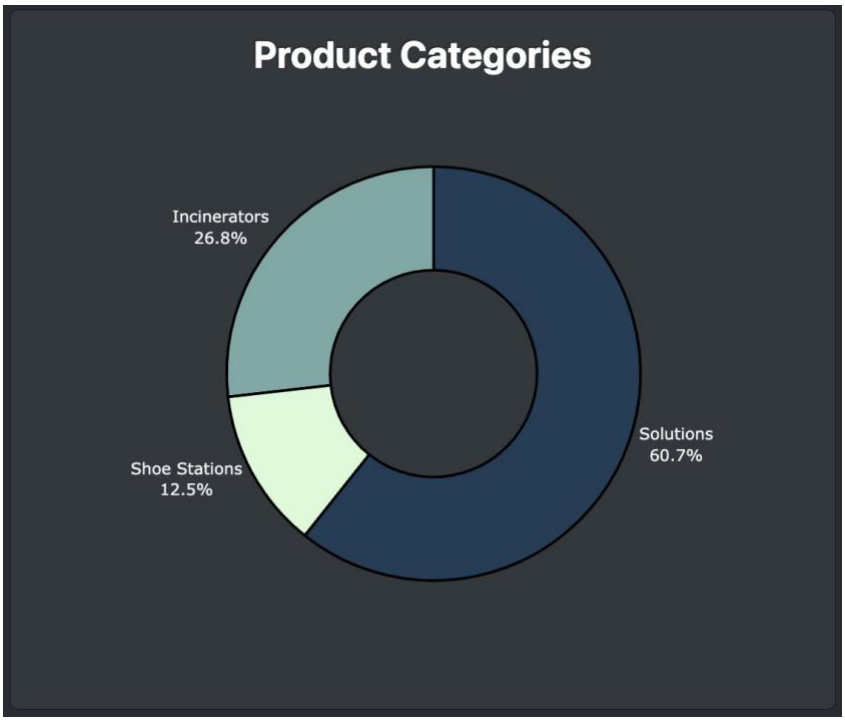


Fig. 9-21

- Number of Orders per State: This metric displays the number of orders made by customers distributed by state. The Client sells its products to customers all over the Unites States of America and wishes to visualize the distribution of sales maong states. We decided to go with a heatmap chart as it is the most aesthetically pleasing in my opinion and does a good job of showing the differences between states.

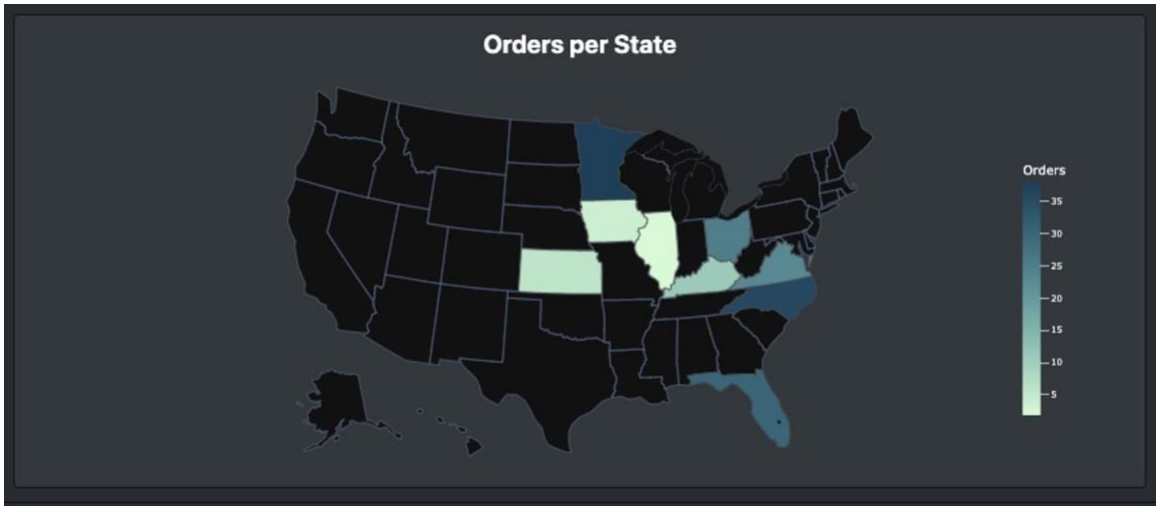


Fig. 9-22

9.6.4 Choosing the layout

Choosing the layout of the dashboard is an essential step in the process because it determines what the user will see first, along with the size of each chart and graph. Initially, I had decided that I was going to use the following layout:

Fig. 9-23

As I moved along in the process, I realized a few things that were wrong with this layout. The sections designated for Cancelled Orders and Refunded Orders were too large to simply include a percentage. This meant that there was less space for the other charts to the right which actually display graphs. Specifically, the map chart was too small for the user to be able to see the entire map, but also drill down into each state. For these reasons, I decided to modify the layout as follows:

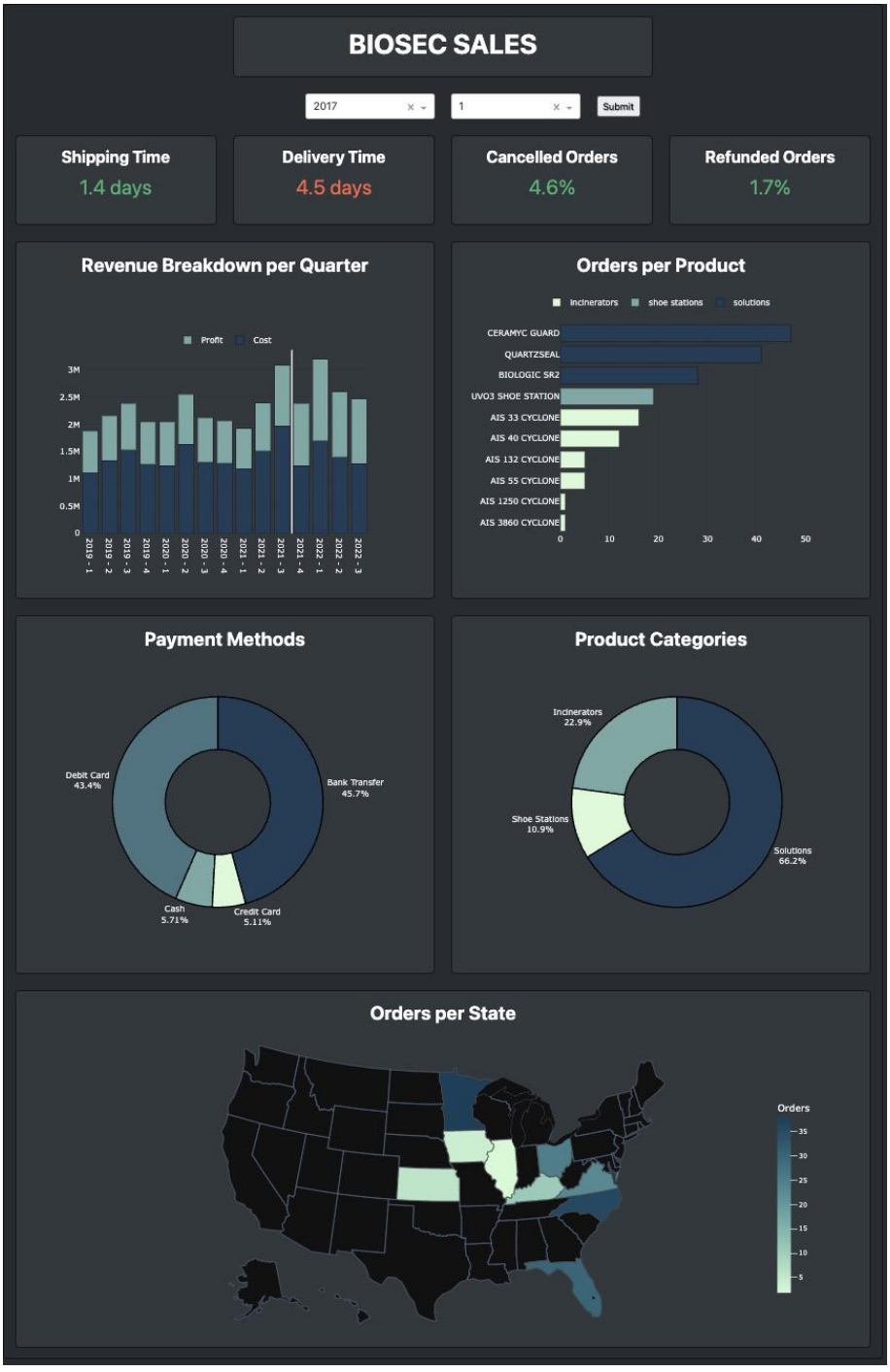

BIOSEC SALES
2017
1
Submit
Shipping Time
1.4 days
Delivery Time
4.5 days
Cancelled Orders
4.6%
Refunded Orders
1.7%
Revenue Breakdown per Quarter
Profit
Cost
3M
2.5M
2M
1.5M
1M
0.5M
0
2019 - 1
2019 - 2
2019 - 3
2019 - 4
2020 - 1
2020 - 2
2020 - 3
2020 - 4
2021 - 1
2021 - 2
2021 - 3
2021 - 4
2022 - 1
2022 - 2
2022 - 3
Orders per Product
Incinerators
shoe stations
solutions
CERAMYC GUARD
QUARTZSEAL
BIOLOGIC SR2
UVO3 SHOE STATION
AIS 33 CYCLONE
AIS 40 CYCLONE
AIS 132 CYCLONE
AIS 55 CYCLONE
AIS 1250 CYCLONE
AIS 3860 CYCLONE
0
10
20
30
40
50
Payment Methods
Debit Card
43.4%
Bank Transfer
45.7%
Cash
5.71%
Credit Card
5.11%
Product Categories
Incinerators
22.9%
Shoe Stations
10.9%
Solutions
66.2%
Orders per State
Orders
35
30
25
20
15
10
5

Fig. 9-24

As we can see, this layout moves the four text-based metrics to the top, taking up a single row of space within the dashboard, without reducing the horizontal space used by the other larger graphs. The map chart was also moved to its own row at the bottom so that it can take as much horizontal space as possible.

9.6.5 Connecting the dashboard to the database

Once the layout was determined, the next step was to connect the dashboard to the database so that data could be fed into each of the charts. I will include a preview of the code for the purposes of explaining each section. However, the full script can be seen in Section 10.3.

```
# Function: Shipping Time
@app.callback(Output('shipping_time','children'),
              Input('submit-button-state','n_clicks'),
              State('year_dropdown','value'),
              State('quarter_dropdown','value'))
def shipping_time(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                    SELECT ROUND(AVG(DATEDIFF(ship_date, order_date)),1)
                    FROM bio_order
                    WHERE YEAR(order_date) = :y
                        AND QUARTER(order_date) = :q;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df1 = pd.DataFrame(statement)
            value = df1.iloc[0,0]

        return html.Div([
            dbc.Card(
                dbc.CardBody([
                    html.Div([
                        html.H4("Shipping Time",
                                style={"font-weight": "bold", "color": "white"}),
    html.P(str(value) + ' days', style={'color': 'MediumSeaGreen' if value < 2 else 'Tomato', 'font-size':'30px'})
                    ], style={'textAlign': 'center'})
                ])
            ),
        ])
```

Fig. 9-25

The image above shows the code that was required to make the Shipping Time chart work correctly. It has several sections which I will explain.

- Callback Function: The first four lines of code correspond to the callback function. These functions *"are automatically called by Dash whenever an input component's property changes, in order to update some property in another component (the output)"*[5]. In this specific case, I set the input to be the submit button that is placed next to the Year and Quarter dropdowns. This input takes in the two values set by the user for Year and Quarter. Then, according to this input, the function sends the two values to shipping_time function in line 6 of the code. This updates the Shipping Time chart to reflect the data that corresponds to these two values. Callback functions are an essential part of Dash, as they are what make an app interactive.
- SQL Query: The second part of the code corresponds to the SQL query. Once the main function receives the two values from the callback function, it connects to the database, executes the SQL query, and creates a Pandas dataframe with the results. It is important to mention that this would not be possible without bind variables.

[5] Plotly (2023). *Basic Dash Callbacks*. Retrieved on April 19, 2023, from: https://dash.plotly.com/basic-callbacks

```
query = text("""
        SELECT ROUND(AVG(DATEDIFF(ship_date, order_date)),1)
        FROM bio_order
        WHERE YEAR(order_date) = :y
            AND QUARTER(order_date) = :q;
""")
parameters = {'y':year,'q':quarter}
```

Fig. 9-26

As we can see in the image above, the Year is set to :y and the Quarter is set to :q. These are not values in the sales data. Instead, they serve as placeholders for the values that are received from the callback function. As stated in the last line of code, y is set to the Year received from the callback function and, accordingly, q is set to the Quarter. As such, whenever the user modifies the values in the dropdowns, this also modifies the SQL query, which allows the correct data to be retrieved and displayed in the graph.

- Return Value: Finally, a section of HTML code is returned by the main function, which is then included in the overall layout of the dashboard.

During the completion of this task, I experienced some issues:

- While building the Orders per Product graph, I kept receiving an error saying that the Orders column I was using for the x-axis of the graph didn't exist. This took me a while to solve because I didn't understand the problem. Eventually, after doing

some research online I found that the problem lied within the callback function. The ‘State’ component of the callback function doesn´t send the values to the main function until the ‘Input’ component is executed. Given that my ‘Input’ component was a Submit button, the values were not sent to the main function until the user clicked on this button. This meant that until this button was clicked, the aforementioned bind variables within the SQL query were empty, resulting in an empty dataframe, thus being unable to send any data to the graph. This resulted in an empty graph being displayed on the dashboard initially. I thought this looked unprofessional, so I decided that once the dashboard was initially opened, it should be completely empty and should not display anything except for the title and the dropdowns. As such, only when the dropdown values were chosen and the Submit button was clicked, would the dashboard populate with all of the graphs and metrics. This issue taught me a lot about analyzing the flow of data within these apps and the importance of analyzing each step of the process in order to identify where the data is and how each component works.

- The second issue I experienced also occurred while I was building the Orders per Product graph. The problem lied in that when I used the product category as the color component for the bars in the chart, it would order the data differently for some reason. To state it in SQL terms, my intention was to order the bars as "ORDER BY product", but when I introduced the color component it would order

the data as "ORDER BY category, product". I consider this one of the few failures of this project because, after extensive research into the issue, I was unable to find a solution to this issue.

9.7 Forecasting sales

As mentioned in Section 9.6.3 of this report, one of the graphs displayed in the dashboard shows the breakdown of revenue per quarter, along with the profit and cost. The following image shows the graph:

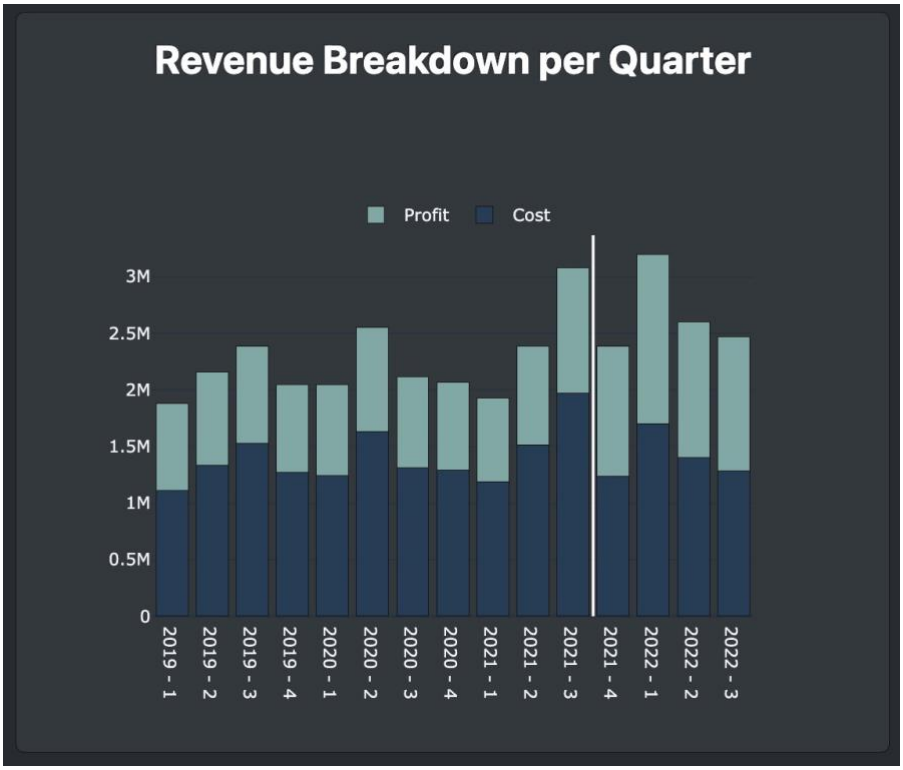


Fig. 9-27

The white line displayed in the graph separated the last four bars from the rest. These last four bars do not include data from the dummy data. Instead, they constitute the predictions made by the regression model I created.

```
#import MinMaxScaler and create a new dataframe for LSTM model
from sklearn.preprocessing import MinMaxScaler
df_model = df_supervised.drop(['cost','date'],axis=1)
#split train and test set
train_set, test_set = df_model[0:-4].values, df_model[-4:].values
```

```
#apply Min Max Scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(train_set)
# reshape training set
train_set = train_set.reshape(train_set.shape[0], train_set.shape[1])
train_set_scaled = scaler.transform(train_set)
# reshape test set
test_set = test_set.reshape(test_set.shape[0], test_set.shape[1])
test_set_scaled = scaler.transform(test_set)
```

+ Code + Markdown

```
X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

```
# Import statsmodels.formula.api
import statsmodels.formula.api as smf

model = Sequential()
model.add(LSTM(4, batch_input_shape=(1, X_train.shape[1], X_train.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, batch_size=1, verbose=1, shuffle=False)
```

Fig. 9-28

The image above shows a snippet of the code I used. The full script can be seen in Section 10.5 of this report. Even though the model ended up working, I encountered several obstacles along the way.

9.7.1 Installing packages

Similar to the issue I encountered when trying to use cx_Oracle, I also had a problem installing the packages that I intended to use to create the regression model. Initially, I had decided to use Python's Prophet package created by Facebook. By doing some research on this package, I was able to conclude that it was an intuitive and comprehensive package that would allow me to create the model with relative ease. However, I was unable to install it due to conflicts between different packages installed on my computer. Online research was insufficient to overcome this obstacle, reason why I decided to use a different package.

When trying to install the Statsmodels package, I came across the same issues regarding conflicts between packages. This forced me to create a separate Python file to create the model. This allowed me to install the required packages without any conflicts arising with the other installed packages. In the end, I decided to use the SciKit-Learn and Keras packages to create the model.

Because of this issue, I was also unable to simply append the results from the regression model to the dataframe that would be later used to populate the graph, as these results were in a different Python file. I tried importing variables from one file to the other but was unable to. This forced me to have to manually copy the model's predictions into the main dashboard file, which is not ideal.

I believe that this issue was beyond my control to a certain extent. My abilities in programming and computer science did not allow me to solve the issue. However, I could have done more research during the planning phase to perhaps anticipate this obstacle and find an alternative solution.

9.7.2 Dummy data was inadequate

Despite my efforts in creating the dummy data in a way that was as realistic as possible, this proved to be insufficient. Usually, sales data has some kind of trend within it. This trend can be cyclical, seasonal, or linear. A good example of this is that every year companies experience a significant increase in sales around holidays such as Black Friday and Christmas. This creates certain trends that are replicated throughout the data and help the corresponding regression models make more accurate predictions. This was not the case in my dummy data. Even though I had modified the data to resemble certain amounts of orders, distribution of clients, among other things, I did not foresee that the data would

have no identifiable trends. This makes it very difficult for a model to make accurate predictions.

This constitutes an example of what I mentioned in Section 7.3 with regards to learning on the go. I had very little prior experience with building regression models before beginning this project. This made it very difficult to anticipate issues such as this one. For upcoming projects, I believe that I will have to prepare more diligently when deciding to embark on a new journey such as building a regression model.

9.8 Final Dashboard

In the end, this was the final dashboard that I created (see following image). Overall, the dashboard met the expectations that I set for myself initially. Clearly, I had some setbacks, and it is not perfect. After speaking with the Client, they informed me that they were very happy with the projects' result. They stated that even though the product currently included Dummy Data that does not necessarily represent the reality of their sales, their intention throughout this project was not to receive a perfect product but, instead, to see first-hand the potential of using a database and a dashboard such as the ones I built for them. All things considered, I believe that this project was a success. I learned a great deal about a plethora of topics along the way, and I believe that this dashboard can provide great value to the Client.

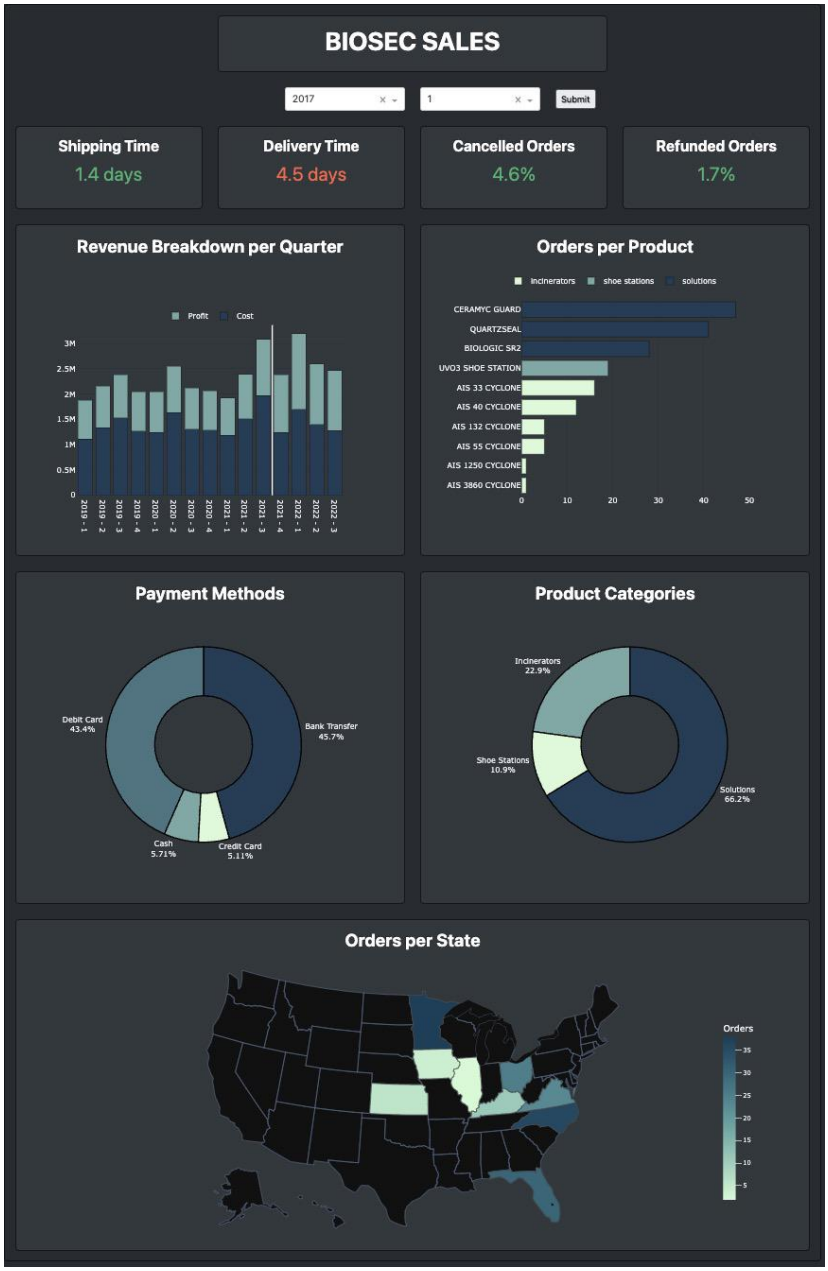

BIOSEC SALES
2017
1
Submit
Shipping Time
1.4 days
Delivery Time
4.5 days
Cancelled Orders
4.6%
Refunded Orders
1.7%
Revenue Breakdown per Quarter
Profit
Cost
3M
2.5M
2M
1.5M
1M
0.5M
0
2019 - 1
2019 - 2
2019 - 3
2019 - 4
2020 - 1
2020 - 2
2020 - 3
2020 - 4
2021 - 1
2021 - 2
2021 - 3
2021 - 4
2022 - 1
2022 - 2
2022 - 3
Orders per Product
Incinerators
shoe stations
solutions
CERAMYC GUARD
QUARTZSEAL
BIOLOGIC SR2
UVO3 SHOE STATION
AIS 33 CYCLONE
AIS 40 CYCLONE
AIS 132 CYCLONE
AIS 55 CYCLONE
AIS 1250 CYCLONE
AIS 3860 CYCLONE
0
10
20
30
40
50
Payment Methods
Debit Card
43.4%
Bank Transfer
45.7%
Cash
5.71%
Credit Card
5.11%
Product Categories
Incinerators
22.9%
Shoe Stations
10.9%
Solutions
66.2%
Orders per State
Orders
35
30
25
20
15
10
5

Fig. 9-29

10. Supporting Materials

10.1 SQL script used to create normalized tables

```
CREATE TABLE bio_address (
   address_pk INT NOT NULL AUTO_INCREMENT,
   street     VARCHAR(100),
   zip_fk     INT NOT NULL,
   PRIMARY KEY (address_pk)
);

CREATE TABLE bio_cat (
   category_pk INT NOT NULL AUTO_INCREMENT,
   category    VARCHAR(100),
   PRIMARY KEY (category_pk)
);

CREATE TABLE bio_city (
   city_pk  INT NOT NULL AUTO_INCREMENT,
   city     VARCHAR(100),
   state_fk INT NOT NULL,
   PRIMARY KEY (city_pk)
);

CREATE TABLE bio_country (
   country_pk INT NOT NULL AUTO_INCREMENT,
   country    VARCHAR(100),
   PRIMARY KEY (country_pk)
);

CREATE TABLE bio_cust (
   cust_pk        INT NOT NULL AUTO_INCREMENT,
   corp_name      VARCHAR(100),
   first_name     VARCHAR(100),
   middle_initial VARCHAR(100),
   last_name      VARCHAR(100),
   email          VARCHAR(100),
   cust_since     DATETIME,
   ssn            VARCHAR(100),
   phone          VARCHAR(100),
   cust_type_fk   INT NOT NULL,
   gender_fk      INT NOT NULL,
   PRIMARY KEY (cust_pk)
```

```
);

CREATE TABLE bio_cust_type (
   cust_type_pk INT NOT NULL AUTO_INCREMENT,
   cust_type    VARCHAR(100),
   PRIMARY KEY (cust_type_pk)
);

CREATE TABLE bio_gender (
   gender_pk INT NOT NULL AUTO_INCREMENT,
   gender    VARCHAR(100),
   PRIMARY KEY (gender_pk)
);

CREATE TABLE bio_order (
   order_pk         INT NOT NULL AUTO_INCREMENT,
   order_date       DATETIME,
   ship_date        DATETIME,
   fulfilled_date   DATETIME,
   qty_ordered      INT,
   total_amount     DOUBLE,
   discount_percent DOUBLE,
   discount_amount  DOUBLE,
   invoiced_amount  DOUBLE,
   prod_fk          INT NOT NULL,
   cust_fk          INT NOT NULL,
   payment_fk       INT NOT NULL,
   address_fk       INT NOT NULL,
   status_fk        INT NOT NULL,
   PRIMARY KEY (order_pk)
);

CREATE TABLE bio_pay (
   payment_pk INT NOT NULL AUTO_INCREMENT,
   payment    VARCHAR(100),
   PRIMARY KEY (payment_pk)
);

CREATE TABLE bio_prod (
   prod_pk     INT NOT NULL AUTO_INCREMENT,
   prod_sku    VARCHAR(100),
   unit_cost   DOUBLE,
   price       DOUBLE,
   category_fk INT NOT NULL,
   PRIMARY KEY (prod_pk)
);

CREATE TABLE bio_region (
   region_pk  INT NOT NULL AUTO_INCREMENT,
```

```
   region     VARCHAR(100),
   country_fk INT NOT NULL,
   PRIMARY KEY (region_pk)
);

CREATE TABLE bio_state (
   state_pk  INT NOT NULL AUTO_INCREMENT,
   state     VARCHAR(100),
   region_fk INT NOT NULL,
   PRIMARY KEY (state_pk)
);

CREATE TABLE bio_status (
   status_pk INT NOT NULL AUTO_INCREMENT,
   status    VARCHAR(100),
   PRIMARY KEY (status_pk)
);

CREATE TABLE bio_zip (
   zip_pk  INT NOT NULL AUTO_INCREMENT,
   zip     VARCHAR(100),
   city_fk INT NOT NULL,
   PRIMARY KEY (zip_pk)
);

ALTER TABLE bio_address
   ADD CONSTRAINT bio_address_bio_zip_fk FOREIGN KEY ( zip_fk )
      REFERENCES bio_zip ( zip_pk );

ALTER TABLE bio_city
   ADD CONSTRAINT bio_city_bio_state_fk FOREIGN KEY ( state_fk )
      REFERENCES bio_state ( state_pk );

ALTER TABLE bio_cust
   ADD CONSTRAINT bio_cust_bio_cust_type_fk FOREIGN KEY ( cust_type_fk )
      REFERENCES bio_cust_type ( cust_type_pk );

ALTER TABLE bio_cust
   ADD CONSTRAINT bio_cust_bio_gender_fk FOREIGN KEY ( gender_fk )
      REFERENCES bio_gender ( gender_pk );

ALTER TABLE bio_order
   ADD CONSTRAINT bio_order_bio_address_fk FOREIGN KEY ( address_fk )
      REFERENCES bio_address ( address_pk );

ALTER TABLE bio_order
   ADD CONSTRAINT bio_order_bio_cust_fk FOREIGN KEY ( cust_fk )
      REFERENCES bio_cust ( cust_pk );
```

```
ALTER TABLE bio_order
   ADD CONSTRAINT bio_order_bio_pay_fk FOREIGN KEY ( payment_fk )
      REFERENCES bio_pay ( payment_pk );

ALTER TABLE bio_order
   ADD CONSTRAINT bio_order_bio_prod_fk FOREIGN KEY ( prod_fk )
      REFERENCES bio_prod ( prod_pk );

ALTER TABLE bio_order
   ADD CONSTRAINT bio_order_bio_status_fk FOREIGN KEY ( status_fk )
      REFERENCES bio_status ( status_pk );

ALTER TABLE bio_prod
   ADD CONSTRAINT bio_prod_bio_cat_fk FOREIGN KEY ( category_fk )
      REFERENCES bio_cat ( category_pk );

ALTER TABLE bio_region
   ADD CONSTRAINT bio_region_bio_country_fk FOREIGN KEY ( country_fk )
      REFERENCES bio_country ( country_pk );

ALTER TABLE bio_state
   ADD CONSTRAINT bio_state_bio_region_fk FOREIGN KEY ( region_fk )
      REFERENCES bio_region ( region_pk );

ALTER TABLE bio_zip
   ADD CONSTRAINT bio_zip_bio_city_fk FOREIGN KEY ( city_fk )
      REFERENCES bio_city ( city_pk );
```

10.2 SQL script used to populate normalized tables

```
DELETE FROM bio_order;
DELETE FROM bio_cust;
DELETE FROM bio_gender;
DELETE FROM bio_cust_type;
DELETE FROM bio_prod;
DELETE FROM bio_cat;
DELETE FROM bio_pay;
DELETE FROM bio_status;
DELETE FROM bio_address;
DELETE FROM bio_zip;
DELETE FROM bio_city;
DELETE FROM bio_state;
DELETE FROM bio_region;
DELETE FROM bio_country;
DELETE FROM bio_stage;

ALTER TABLE bio_order AUTO_INCREMENT = 1;
```

```
ALTER TABLE bio_cust AUTO_INCREMENT = 1;
ALTER TABLE bio_gender AUTO_INCREMENT = 1;
ALTER TABLE bio_cust_type AUTO_INCREMENT = 1;
ALTER TABLE bio_prod AUTO_INCREMENT = 1;
ALTER TABLE bio_cat AUTO_INCREMENT = 1;
ALTER TABLE bio_pay AUTO_INCREMENT = 1;
ALTER TABLE bio_status AUTO_INCREMENT = 1;
ALTER TABLE bio_address AUTO_INCREMENT = 1;
ALTER TABLE bio_zip AUTO_INCREMENT = 1;
ALTER TABLE bio_city AUTO_INCREMENT = 1;
ALTER TABLE bio_state AUTO_INCREMENT = 1;
ALTER TABLE bio_region AUTO_INCREMENT = 1;
ALTER TABLE bio_country AUTO_INCREMENT = 1;

##############################################################################

##### POPULATING COUNTRY TABLE #####

INSERT INTO bio_country(country)
SELECT DISTINCT country
FROM bio_stage
WHEREbio_stage.country IS NOT NULL
                AND NOT EXISTS (SELECT country FROM bio_country);

# SELECT * FROM bio_country;

##############################################################################

##### POPULATING REGION TABLE #####

INSERT INTO bio_region(region, country_fk)
SELECT DISTINCT s.region region, c.country_pk country_fk
FROM bio_stage s, bio_country c
WHERES.country = c.country
                AND s.region IS NOT NULL
    AND NOT EXISTS (SELECT region FROM bio_region);

# SELECT * FROM bio_region;

##############################################################################

##### POPULATING STATE TABLE #####

INSERT INTO bio_state(state, region_fk)
SELECT DISTINCT s.state state, r.region_pk region_fk
FROM bio_stage s, bio_region r
WHERES.region = r.region
                AND s.state IS NOT NULL
    AND NOT EXISTS (SELECT state FROM bio_state);
```

```
# SELECT * FROM bio_state;

#############################################################################

##### POPULATING CITY TABLE #####

INSERT INTO bio_city(city, state_fk)
SELECT DISTINCT bio_stage.city city, bio_state.state_pk state_fk
FROM bio_stage, bio_state
WHEREbio_stage.state = bio_state.state
                    AND bio_stage.city IS NOT NULL
     AND NOT EXISTS (SELECT city FROM bio_city);

# SELECT * FROM bio_city;

#############################################################################

##### POPULATING ZIP TABLE #####

INSERT INTO bio_zip(zip, city_fk)
SELECT DISTINCT s.zip zip, c.city_pk city_fk
FROM bio_stage s, bio_city c
WHEREs.city = c.city
                    AND s.zip IS NOT NULL
     AND NOT EXISTS (SELECT zip FROM bio_zip);

# SELECT * FROM bio_zip;

#############################################################################

##### POPULATING ADDRESS TABLE #####

INSERT INTO bio_address(street, zip_fk)
SELECT DISTINCT s.street street, z.zip_pk zip_fk
FROM bio_stage s, bio_zip z
WHEREs.zip = z.zip
                    AND s.street IS NOT NULL
     AND NOT EXISTS (SELECT street FROM bio_address);

# SELECT * FROM bio_address;

#############################################################################

##### POPULATING STATUS TABLE #####

INSERT INTO bio_status(status)
SELECT DISTINCT bio_stage.status
FROM bio_stage
```

```
WHEREbio_stage.status IS NOT NULL
                AND NOT EXISTS (SELECT status FROM bio_status);

# SELECT * FROM bio_status;

##########################################################################

##### POPULATING PAYMENT TABLE #####

INSERT INTO bio_pay(payment)
SELECT DISTINCT bio_stage.payment
FROM bio_stage
WHEREbio_stage.payment IS NOT NULL
                AND NOT EXISTS (SELECT payment FROM bio_pay);

# SELECT * FROM bio_pay;

##########################################################################

##### POPULATING CATEGORY TABLE #####

INSERT INTO bio_cat(category)
SELECT DISTINCT bio_stage.category
FROM bio_stage
WHEREbio_stage.category IS NOT NULL
                AND NOT EXISTS (SELECT category FROM bio_cat);

# SELECT * FROM bio_cat;

##########################################################################

##### POPULATING PRODUCT TABLE #####

INSERT INTO bio_prod(prod_sku, unit_cost, price, category_fk)
SELECT DISTINCT s.prod_sku prod_sku,
                                s.unit_cost unit_cost,
        s.price price,
        c.category_pk category_fk
FROM bio_stage s, bio_cat c
WHERES.category = c.category
                AND s.prod_sku IS NOT NULL
    AND s.unit_cost IS NOT NULL
    AND s.price IS NOT NULL
    AND NOT EXISTS (SELECT prod_sku FROM bio_prod);

# SELECT * FROM bio_prod;

##########################################################################
```

```
##### POPULATING CUSTOMER TYPE TABLE #####

INSERT INTO bio_cust_type(cust_type)
SELECT DISTINCT cust_type
FROM bio_stage
WHERE NOT EXISTS (SELECT cust_type FROM bio_cust_type);

# SELECT * FROM bio_cust_type;

###############################################################################

##### POPULATING GENDER TABLE #####

INSERT INTO bio_gender(gender)
SELECT DISTINCT gender
FROM bio_stage
WHEREbio_stage.gender IS NOT NULL
                AND NOT EXISTS (SELECT gender FROM bio_gender);

# SELECT * FROM bio_gender;

###############################################################################

##### POPULATING CUSTOMER TABLE #####

INSERT INTO bio_cust(corp_name, first_name, middle_initial, last_name, email, cust_since, ssn, phone,
cust_type_fk, gender_fk)
SELECT DISTINCT s.corp_name corp_name,
        s.first_name first_name,
        s.middle_initial middle_initial,
        s.last_name last_name,
        s.email email,
        s.cust_since cust_since,
        s.ssn ssn,
        s.phone phone,
        c.cust_type_pk cust_type_fk,
        g.gender_pk gender_fk
FROM bio_stage s
LEFT JOIN bio_cust_type c ON s.cust_type = c.cust_type
LEFT JOIN bio_gender g ON s.gender = g.gender
WHERE s.email IS NOT NULL
         AND s.cust_since IS NOT NULL
    AND s.phone IS NOT NULL
    AND s.cust_type IS NOT NULL
    AND NOT EXISTS (SELECT corp_name, first_name, middle_initial, last_name FROM bio_cust);

# SELECT * FROM bio_cust;

###############################################################################
```

```
##### POPULATING ORDER TABLE #####

INSERT INTO bio_order(order_date, ship_date, fulfilled_date, qty_ordered, total_amount,
discount_percent, discount_amount, invoiced_amount, prod_fk, cust_fk, payment_fk, address_fk,
status_fk)
SELECT DISTINCT sg.order_date order_date,
            sg.ship_date ship_date,
            sg.fulfilled_date fulfilled_date,
            sg.qty_ordered qty_ordered,
            sg.total_amount total_amount,
            sg.discount_percent discount_percent,
            sg.discount_amount discount_amount,
            sg.invoiced_amount invoiced_amount,
            pr.prod_pk prod_fk,
            c.cust_pk cust_fk,
            p.payment_pk payment_fk,
             a.address_pk address_fk,
            st.status_pk status_fk
FROM bio_stage sg
LEFT JOIN bio_cust c ON sg.corp_name = c.corp_name
                                                OR sg.first_name = c.first_name
                                                OR sg.middle_initial = c.middle_initial
                                                OR sg.last_name = c.last_name
LEFT JOIN bio_prod pr ON sg.prod_sku = pr.prod_sku
LEFT JOIN bio_pay p ON sg.payment = p.payment
LEFT JOIN bio_address a ON sg.street = a.street
LEFT JOIN bio_status st ON sg.status = st.status;

# SELECT * FROM bio_order;
```

10.3 Complete database script

```
### DATABASE APP

# This app will be used to import a csv file containing the Client's sales data,
# transfer the file's contents into the bio_stage table in the MySQL database,
# and populate the corresponding normalized tables in the transactional database.

##############################################################################################
#####################################################################

# STEP 1: IMPORTING THE NECESSARY LIBRARIES
```

```
import pandas as pd
import os
import mysql.connector
import pymysql
import sqlalchemy
from sqlalchemy import create_engine, text

################################################################################################
#########################################################################

# STEP 2: IMPORTING THE CSV FILE INTO PYTHON

# Create a string with the current folder we are in. The csv file must be in the current working directory for this to work.
pwd = os.getcwd()

# Create a string of the entire file path.
filepath = pwd + '/data/dummy_data.csv'

# Import the file and assign it to 'df' dataframe
df = pd.read_csv(filepath)
print("File imported successfuly.")

# We preview the file's shape
print("The dataframe has the following shape:", df.shape)

################################################################################################
#########################################################################

# STEP 3: CONNECTING TO THE DATABASE

# Creating the engine with sqlalchemy
sqlEngine = create_engine('mysql+pymysql://root:Alexandra2005@localhost/bio')
print("Engine created.")

################################################################################################
#########################################################################

# STEP 4: POPULATING STAGE TABLE

with sqlEngine.connect() as con:

    table = "bio_stage"

    try:
        df.to_sql(table, con=sqlEngine, if_exists='append', index=False)

    except ValueError as vx:
        print(vx)
```

```
    except Exception as ex:
        print(ex)

    else:
        print("Table %s populated successfully."%table);

    finally:
        con.close()

##########################################################################################
#########################################################################

# STEP 5: POPULATING NORMALIZED TABLES

with sqlEngine.connect() as con:

    country_query = text("""

    INSERT INTO bio_country(country)
    SELECT DISTINCT country
    FROM bio_stage
    WHERE       bio_stage.country IS NOT NULL
            AND NOT EXISTS (SELECT country FROM bio_country);

    """)

    region_query = text("""

    INSERT INTO bio_region(region, country_fk)
    SELECT DISTINCT s.region region, c.country_pk country_fk
    FROM bio_stage s, bio_country c
    WHERE       s.country = c.country
                    AND s.region IS NOT NULL
            AND NOT EXISTS (SELECT region FROM bio_region);

    """)

    state_query = text("""

    INSERT INTO bio_state(state, region_fk)
    SELECT DISTINCT s.state state, r.region_pk region_fk
    FROM bio_stage s, bio_region r
    WHERE       s.region = r.region
            AND s.state IS NOT NULL
            AND NOT EXISTS (SELECT state FROM bio_state);

    """)
```

```
city_query = text("""

INSERT INTO bio_city(city, state_fk)
SELECT DISTINCT bio_stage.city city, bio_state.state_pk state_fk
FROM bio_stage, bio_state
WHERE       bio_stage.state = bio_state.state
      AND bio_stage.city IS NOT NULL
      AND NOT EXISTS (SELECT city FROM bio_city);

""")

zip_query = text("""

INSERT INTO bio_zip(zip, city_fk)
SELECT DISTINCT s.zip zip, c.city_pk city_fk
FROM bio_stage s, bio_city c
WHERE       s.city = c.city
      AND s.zip IS NOT NULL
      AND NOT EXISTS (SELECT zip FROM bio_zip);

""")

address_query = text("""

INSERT INTO bio_address(street, zip_fk)
SELECT DISTINCT s.street street, z.zip_pk zip_fk
FROM bio_stage s, bio_zip z
WHERE       s.zip = z.zip
      AND s.street IS NOT NULL
      AND NOT EXISTS (SELECT street FROM bio_address);

""")

status_query = text("""

INSERT INTO bio_status(status)
SELECT DISTINCT bio_stage.status
FROM bio_stage
WHERE       bio_stage.status IS NOT NULL
      AND NOT EXISTS (SELECT status FROM bio_status);

""")

payment_query = text("""

INSERT INTO bio_pay(payment)
SELECT DISTINCT bio_stage.payment
FROM bio_stage
WHERE       bio_stage.payment IS NOT NULL
```

```
        AND NOT EXISTS (SELECT payment FROM bio_pay);

    """)

    category_query = text("""

    INSERT INTO bio_cat(category)
    SELECT DISTINCT bio_stage.category
    FROM bio_stage
    WHERE       bio_stage.category IS NOT NULL
        AND NOT EXISTS (SELECT category FROM bio_cat);

    """)

    product_query = text("""

    INSERT INTO bio_prod(prod_sku, unit_cost, price, category_fk)
    SELECT DISTINCT s.prod_sku prod_sku,
                s.unit_cost unit_cost,
                s.price price,
                c.category_pk category_fk
    FROM bio_stage s, bio_cat c
    WHERE       s.category = c.category
        AND s.prod_sku IS NOT NULL
        AND s.unit_cost IS NOT NULL
        AND s.price IS NOT NULL
        AND NOT EXISTS (SELECT prod_sku FROM bio_prod);

    """)

    cust_type_query = text("""

    INSERT INTO bio_cust_type(cust_type)
    SELECT DISTINCT cust_type
    FROM bio_stage
    WHERE NOT EXISTS (SELECT cust_type FROM bio_cust_type);

    """)

    gender_query = text("""

    INSERT INTO bio_gender(gender)
    SELECT DISTINCT gender
    FROM bio_stage
    WHERE       bio_stage.gender IS NOT NULL
        AND NOT EXISTS (SELECT gender FROM bio_gender);

    """)
```

```
    customer_query = text("""

    INSERT INTO bio_cust(corp_name, first_name, middle_initial, last_name, email, cust_since, ssn,
phone, cust_type_fk, gender_fk)
    SELECT DISTINCT s.corp_name corp_name,
                s.first_name first_name,
                s.middle_initial middle_initial,
                s.last_name last_name,
                s.email email,
                s.cust_since cust_since,
                s.ssn ssn,
                s.phone phone,
                c.cust_type_pk cust_type_fk,
                g.gender_pk gender_fk
    FROM bio_stage s
    LEFT JOIN bio_cust_type c ON s.cust_type = c.cust_type
    LEFT JOIN bio_gender g ON s.gender = g.gender
    WHERE s.email IS NOT NULL
       AND s.cust_since IS NOT NULL
       AND s.phone IS NOT NULL
       AND s.cust_type IS NOT NULL
       AND NOT EXISTS (SELECT corp_name, first_name, middle_initial, last_name FROM bio_cust);

    """)

    order_query = text("""

    INSERT INTO bio_order(order_date, ship_date, fulfilled_date, qty_ordered, total_amount,
discount_percent, discount_amount, invoiced_amount, prod_fk, cust_fk, payment_fk, address_fk,
status_fk)
    SELECT DISTINCT sg.order_date order_date,
                sg.ship_date ship_date,
                sg.fulfilled_date fulfilled_date,
                sg.qty_ordered qty_ordered,
                sg.total_amount total_amount,
                sg.discount_percent discount_percent,
                sg.discount_amount discount_amount,
                sg.invoiced_amount invoiced_amount,
                pr.prod_pk prod_fk,
                c.cust_pk cust_fk,
                p.payment_pk payment_fk,
                a.address_pk address_fk,
                st.status_pk status_fk
    FROM bio_stage sg
    LEFT JOIN bio_cust c ON sg.corp_name = c.corp_name
                    OR sg.first_name = c.first_name
                    OR sg.middle_initial = c.middle_initial
                    OR sg.last_name = c.last_name
    LEFT JOIN bio_prod pr ON sg.prod_sku = pr.prod_sku
```

```
LEFT JOIN bio_pay p ON sg.payment = p.payment
LEFT JOIN bio_address a ON sg.street = a.street
LEFT JOIN bio_status st ON sg.status = st.status;

""")

con.execute(country_query)
print("bio_country - success")
con.execute(region_query)
print("bio_region - success")
con.execute(state_query)
print("bio_state - success")
con.execute(city_query)
print("bio_city - success")
con.execute(zip_query)
print("bio_zip - success")
con.execute(address_query)
print("bio_address - success")
con.execute(status_query)
print("bio_status - success")
con.execute(payment_query)
print("bio_payment - success")
con.execute(category_query)
print("bio_category - success")
con.execute(product_query)
print("bio_product - success")
con.execute(cust_type_query)
print("bio_cust_type - success")
con.execute(gender_query)
print("bio_gender - success")
con.execute(customer_query)
print("bio_customer - success")
con.execute(order_query)
print("bio_order - success")

con.execute(text("DELETE FROM bio_stage"))

con.commit()
```

10.4 Complete dashboard script

SETUP

```
# importing libraries
import dash
from dash import dcc
from dash import html
import dash_bootstrap_components as dbc
from jupyter_dash import JupyterDash
from dash.dependencies import Output, Input, State
import plotly.graph_objects as go
import plotly.express as px
import plotly.colors as pc
import os
import pandas as pd
import mysql.connector
import pymysql
import sqlalchemy
from sqlalchemy import create_engine, text

# creating sqlalchemy engine
sqlEngine = create_engine('mysql+pymysql://root:Alexandra2005@localhost/bio')

# running the dashboard
app = JupyterDash(__name__, external_stylesheets=[dbc.themes.SLATE])
```

DROPDOWNS

```
with sqlEngine.connect() as con:
    dropdown_df1 = pd.DataFrame(con.execute(text("SELECT DISTINCT YEAR(order_date) col FROM
bio_order;")).fetchall())
    dropdown_df2 = pd.DataFrame(con.execute(text("SELECT DISTINCT QUARTER(order_date) col
FROM bio_order;")).fetchall())
    dropdown_df3 = pd.DataFrame(con.execute(text("SELECT DISTINCT UPPER(prod_sku) col FROM
bio_prod ORDER BY UPPER(prod_sku);")).fetchall())

    dropdown_years = list(dropdown_df1['col'])
    dropdown_quarters = list(dropdown_df2['col'])
    dropdown_products = list(dropdown_df3['col'])
```

CALLBACK FUNCTIONS

```
# Text field
def drawText():
```

```
    return html.Div([
        dbc.Card(
            dbc.CardBody([
                html.Div([
                    html.H2("Text"),
                ], style={'textAlign': 'center'})
            ])
        ),
    ])

# Function: Shipping Time
@app.callback(Output('shipping_time','children'),
              Input('submit-button-state','n_clicks'),
              State('year_dropdown','value'),
              State('quarter_dropdown','value'))
def shipping_time(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                  SELECT ROUND(AVG(DATEDIFF(ship_date, order_date)),1)
                  FROM bio_order
                  WHERE YEAR(order_date) = :y
                      AND QUARTER(order_date) = :q;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df1 = pd.DataFrame(statement)
            value = df1.iloc[0,0]

        return html.Div([
            dbc.Card(
                dbc.CardBody([
                    html.Div([
                        html.H4("Shipping Time",
                                style={"font-weight": "bold", "color": "white"}),
    html.P(str(value) + ' days', style={'color': 'MediumSeaGreen' if value < 2 else 'Tomato', 'font-
size':'30px'})
                    ], style={'textAlign': 'center'})
                ])
            ),
        ])

# Function: Delivery Time
@app.callback(Output('delivery_time','children'),
              Input('submit-button-state','n_clicks'),
              State('year_dropdown','value'),
              State('quarter_dropdown','value'))
```

```
def delivery_time(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                    SELECT ROUND(AVG(DATEDIFF(fulfilled_date, ship_date)),1)
                    FROM bio_order
                    WHERE YEAR(order_date) = :y
                        AND QUARTER(order_date) = :q;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df2 = pd.DataFrame(statement)
            value = df2.iloc[0,0]

        return html.Div([
            dbc.Card(
                dbc.CardBody([
                    html.Div([
                        html.H4("Delivery Time",
                                style={"font-weight": "bold", "color": "white"}),
                        html.P(str(value) + ' days', style={'color': 'MediumSeaGreen' if value < 3 else 'Tomato',
'font-size':'30px'})
                    ], style={'textAlign': 'center'})
                ])
            ),
        ])

# Function: Percentage of Cancelled Orders
@app.callback(Output('cancelled_orders','children'),
              Input('submit-button-state','n_clicks'),
              State('year_dropdown','value'),
              State('quarter_dropdown','value'))
def cancelled_orders(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                    SELECT ROUND(((COUNT(*) / (SELECT COUNT(*) FROM bio_order WHERE
YEAR(order_date) = :y AND QUARTER(order_date) = :q)) * 100),1) AS 'col'
                    FROM bio_order
                    LEFT JOIN bio_status ON status_pk = status_fk
                    WHERE status = 'cancelled'
                        AND YEAR(order_date) = :y
                        AND QUARTER(order_date) = :q;
            """)
            parameters = {'y':year,'q':quarter}
```

```
        statement = con.execute(query, parameters).fetchall()
        df3 = pd.DataFrame(statement)
        value = df3.iloc[0,0]

    return html.Div([
        dbc.Card(
            dbc.CardBody([
                html.Div([
                    html.H4("Cancelled Orders",
                            style={"font-weight": "bold", "color": "white"}),
                    html.P(str(value) + '%', style={'color': 'MediumSeaGreen' if value < 5.0 else 'Tomato', 'font-
size':'30px'})
                ], style={'textAlign': 'center'})
            ])
        ),
    ])

# Function: Percentage of Refunded Orders
@app.callback(Output('refunded_orders','children'),
              Input('submit-button-state','n_clicks'),
              State('year_dropdown','value'),
              State('quarter_dropdown','value'))
def refunded_orders(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                    SELECT ROUND(((COUNT(*) / (SELECT COUNT(*) FROM bio_order WHERE
YEAR(order_date) = :y AND QUARTER(order_date) = :q)) * 100),1) AS 'col'
                    FROM bio_order
                    LEFT JOIN bio_status ON status_pk = status_fk
                    WHERE status = 'refunded'
                        AND YEAR(order_date) = :y
                        AND QUARTER(order_date) = :q;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df4 = pd.DataFrame(statement)
            value = df4.iloc[0,0]

    return html.Div([
        dbc.Card(
            dbc.CardBody([
                html.Div([
                    html.H4("Refunded Orders",
                            style={"font-weight": "bold", "color": "white"}),
                    html.P(str(value) + '%', style={'color': 'MediumSeaGreen' if value < 5.0 else 'Tomato', 'font-
size':'30px'})
```

```
                ], style={'textAlign': 'center'}
                )
            ])
        ),
    ])

# Function: Number of Orders per Product
@app.callback(Output('orders_per_product','children'),
            Input('submit-button-state','n_clicks'),
            State('year_dropdown','value'),
            State('quarter_dropdown','value'))
def orders_per_product(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                SELECT UPPER(prod_sku) product, category, COUNT(*) orders
                FROM bio_order
                LEFT JOIN bio_prod ON prod_pk = prod_fk
                LEFT JOIN bio_cat ON category_pk = category_fk
                WHERE YEAR(order_date) = :y
                    AND QUARTER(order_date) = :q
                GROUP BY prod_sku, category
                ORDER BY orders ASC;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df5 = pd.DataFrame(statement)
        return html.Div([
                dbc.Card(
                    dbc.CardBody([
                        html.Div([
                            html.H3("Orders per Product",
                                style={"font-weight": "bold", "color": "white"}),
                            dcc.Graph(
                                figure=px.bar(data_frame=df5,
                                        x='orders',
                                        y='product',
                                        color='category',

color_discrete_sequence=['rgba(218,250,215,1)','rgba(119,169,166,1)','rgba(31,62,88,1)']
                                        ).update_layout(template='plotly_dark',
                                                plot_bgcolor= 'rgba(0, 0, 0, 0)',
                                                paper_bgcolor= 'rgba(0, 0, 0, 0)',
                                                yaxis_title=None,
                                                xaxis_title=None,
                                                xaxis_range=[0,50],
                                                legend=dict(
```

```
                                        orientation='h',
                                        title=None,
                                        xanchor='center',
                                        x=0.4,
                                        yanchor='bottom',
                                        y=1.05
                                        )
                                    ),
                            config={
                                'displayModeBar': False
                            }, style={'width':'100%'}
                        )
                    ],
                    style={'textAlign': 'center'})
                ])
            ),
        ])

# Function: Payment Methods
@app.callback(Output('payment_methods','children'),
          Input('submit-button-state','n_clicks'),
          State('year_dropdown','value'),
          State('quarter_dropdown','value'))
def payment_methods(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                SELECT  payment,
                    COUNT(*) orders,
                    ROUND(((COUNT(*)/(SELECT COUNT(*) FROM bio_order WHERE
YEAR(order_date) = :y AND QUARTER(order_date) = :q))*100),1) percentage
                FROM bio_order
                LEFT JOIN bio_pay ON payment_pk = payment_fk
                WHERE YEAR(order_date) = :y
                    AND QUARTER(order_date) = :q
                GROUP BY payment
                ORDER BY payment;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df6 = pd.DataFrame(statement)
        return html.Div([
                dbc.Card(
                    dbc.CardBody([
                        html.Div([
                            html.H3("Payment Methods",
                                style={"font-weight": "bold", "color": "white"}),
```

```
                        dcc.Graph(
                            figure = px.pie(
                                    df6,
                                    values='percentage',
                                    names='payment',
                                    hole=0.5,
                                    hover_data=['orders'],

color_discrete_sequence=['rgba(31,62,88,1)','rgba(71,117,131,1)','rgba(119,169,166,1)','rgba(218,250,215,1
)']
                            ).update_traces(
                                    marker=dict(line=dict(color='#000000', width=2)),
                                    textposition='outside',
                                    text=['Bank Transfer','Cash','Credit Card','Debit Card'],
                                    showlegend=False
                            ).update_layout(template='plotly_dark',
                                        plot_bgcolor= 'rgba(0, 0, 0, 0)',
                                        paper_bgcolor= 'rgba(0, 0, 0, 0)'
                            ),
                            config={
                                'displayModeBar': False
                            },
                            style={'width':'100%'}
                        )
                    ],
                    style={'textAlign': 'center'})
                ])
            ),
        ])

# Function: Product Categories
@app.callback(Output('product_categories','children'),
          Input('submit-button-state','n_clicks'),
          State('year_dropdown','value'),
          State('quarter_dropdown','value'))
def product_categories(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                SELECT  category,
                    COUNT(*) orders,
                    ROUND(((COUNT(*)/(SELECT COUNT(*) FROM bio_order WHERE
YEAR(order_date) = :y AND QUARTER(order_date) = :q))*100),1) percentage
                FROM bio_order
                LEFT JOIN bio_prod ON prod_pk = prod_fk
                LEFT JOIN bio_cat ON category_pk = category_fk
                WHERE YEAR(order_date) = :y
```

```
                    AND QUARTER(order_date) = :q
                GROUP BY category
                ORDER BY category;
            """)
            parameters = {'y':year,'q':quarter}
            statement = con.execute(query, parameters).fetchall()
            df7 = pd.DataFrame(statement)
        return html.Div([
                dbc.Card(
                    dbc.CardBody([
                        html.Div([
                            html.H3("Product Categories",
                                style={"font-weight": "bold", "color": "white"}),
                            dcc.Graph(
                                figure = px.pie(
                                        df7,
                                        values='percentage',
                                        names='category',
                                        hole=0.5,
                                        hover_data=['orders'],

color_discrete_sequence=['rgba(31,62,88,1)','rgba(119,169,166,1)','rgba(218,250,215,1)']
                                    ).update_traces(
                                        marker=dict(line=dict(color='#000000', width=2)),
                                        textposition='outside',
                                        text=['Incinerators','Shoe Stations','Solutions'],
                                        showlegend=False
                                    ).update_layout(template='plotly_dark',
                                            plot_bgcolor= 'rgba(0, 0, 0, 0)',
                                            paper_bgcolor= 'rgba(0, 0, 0, 0)'
                                    ),
                                    config={
                                        'displayModeBar': False
                                    },
                                    style={'width':'100%'}
                                )
                            ],
                            style={'textAlign': 'center'})
                        ])
                    ),
                ])

# Function: Map Chart
@app.callback(Output('map_chart','children'),
              Input('submit-button-state','n_clicks'),
              State('year_dropdown','value'),
              State('quarter_dropdown','value'))
def orders_per_state(n_clicks, year, quarter):
    if not year or not quarter:
```

```
    return None
else:
    with sqlEngine.connect() as con:
        query = text("""
            SELECT state, COUNT(*) orders
            FROM bio_order
            LEFT JOIN bio_address ON address_pk = address_fk
            LEFT JOIN bio_zip ON zip_pk = zip_fk
            LEFT JOIN bio_city ON city_pk = city_fk
            LEFT JOIN bio_state ON state_pk = state_fk
            WHERE YEAR(order_date) = :y
                AND QUARTER(order_date) = :q
            GROUP BY state;
        """)
        parameters = {'y':year,'q':quarter}
        statement = con.execute(query, parameters).fetchall()
        df8 = pd.DataFrame(statement)
    return html.Div([
            dbc.Card(
                dbc.CardBody([
                    html.Div([
                        html.H3("Orders per State",
                                style={"font-weight": "bold", "color": "white"}),
                        dcc.Graph(
                            figure=px.choropleth(df8,
                                                 locations=df8['state'],
                                                 locationmode="USA-states",
                                                 color=df8['orders'],
                                                 scope="usa",
                                                 color_continuous_scale="darkmint"
                            )
                            .update_layout(template='plotly_dark',
                                           geo_bgcolor="rgba(0,0,0,0)",
                                           paper_bgcolor='rgba(0,0,0,0)',
                                           coloraxis_colorbar=dict(
                                               thicknessmode="pixels", thickness=20,
                                               lenmode="pixels", len=300,
                                               yanchor="top", y=0.8,
                                               xanchor='left', x=0.9,
                                               ticks="outside", ticklen=10,
                                               tickcolor="#ffffff",
                                               title='Orders',
                                               title_font=dict(size=14),
                                               tickfont=dict(size=10),
                                           ),
                                           margin=dict(l=10, r=10, t=10, b=10)
                            ),
                            config={
                                'displayModeBar': False
```

```
                    },
                    style={'width':'100%'}
                )
            ],
            style={'textAlign': 'center'})
        ])
    ),
])


# Function: Revenue Breakdown Chart
@app.callback(Output('revenue_profit','children'),
          Input('submit-button-state','n_clicks'),
          State('year_dropdown','value'),
          State('quarter_dropdown','value'))
def revenue_profit(n_clicks, year, quarter):
    if not year or not quarter:
        return None
    else:
        with sqlEngine.connect() as con:
            query = text("""
                SELECT          YEAR(order_date) year,
                    QUARTER(order_date) qtr,
                    CONCAT(YEAR(order_date), ' - ', QUARTER(order_date)) qtryear,
                    ROUND(SUM(invoiced_amount),0) revenue,
                    ROUND(SUM(qty_ordered * unit_cost),0) cost,
                    ROUND(((SUM(qty_ordered * unit_cost)/SUM(invoiced_amount))*100),1) cost_percent,
                    ROUND(SUM(invoiced_amount) - (SUM(qty_ordered * unit_cost)),0) profit,
                    ROUND(((SUM(invoiced_amount) - (SUM(qty_ordered *
unit_cost)))/SUM(invoiced_amount))*100,1) profit_percent
                FROM bio_order
                LEFT JOIN bio_prod ON prod_pk = prod_fk
                WHERE YEAR(order_date) BETWEEN (SELECT MAX(YEAR(order_date)) FROM
bio_order)-2 AND (SELECT MAX(YEAR(order_date)) FROM bio_order)
                GROUP BY YEAR(order_date), QUARTER(order_date), CONCAT(YEAR(order_date), ' - ',
QUARTER(order_date));
            """)
            statement = con.execute(query).fetchall()
            df9 = pd.DataFrame(statement)
            df9['qtr'] = df9['qtr'].astype(str)
            df9 = df9.drop(11)
            # Appending rows for next 4 quarters based on predicted costs and profits.
            df_append = pd.DataFrame([
                [2021,'4','2021 - 4',0.0, 1238972, 0.0, 1146284, 0.0],
                [2022,'1','2022 - 1',0.0, 1701269, 0.0, 1496533, 0.0],
                [2022,'2','2022 - 2',0.0, 1399340, 0.0, 1202617, 0.0],
                [2022,'3','2022 - 3',0.0, 1282957, 0.0, 1187395, 0.0],
                ], columns=['year','qtr','qtryear','revenue','cost','cost_percent','profit','profit_percent'])
            df9 = df9.append(df_append, ignore_index=True)
```

```
    with sqlEngine.connect() as con:
        query = text("""
          SELECT    CONCAT(YEAR(order_date), ' - ', QUARTER(order_date)) date,
              ROUND(SUM(invoiced_amount),0) sales
          FROM bio_order
          GROUP BY YEAR(order_date), QUARTER(order_date), CONCAT(YEAR(order_date), ' -
', QUARTER(order_date));
        """)
        statement = con.execute(query).fetchall()
        df_sales = pd.DataFrame(statement)

    return html.Div([
        dbc.Card(
          dbc.CardBody([
            html.Div([
              html.H3("Revenue Breakdown per Quarter",
                  style={"font-weight": "bold", "color": "white"}),
              dcc.Graph(
                figure = go.Figure(
                  data=[
                    go.Bar(name='Cost',
                      x=df9['qtryear'],
                      y=df9['cost'],
                      hovertext=df9['cost_percent'].apply(lambda x: "{:.1f}%".format(x)),
                      textposition='auto',
                      marker=dict(color='rgba(31,62,88,1)')
                    ),
                    go.Bar(name='Profit',
                      x=df9['qtryear'],
                      y=df9['profit'],
                      hovertext=df9['profit_percent'].apply(lambda x: "{:.1f}%".format(x)),
                      textposition='auto',
                      marker=dict(color='rgba(119,169,166,1)')
                    )
                  ]
                )
                .update_layout(barmode='stack',
                      template='plotly_dark',
                      plot_bgcolor= 'rgba(0, 0, 0, 0)',
                      paper_bgcolor= 'rgba(0, 0, 0, 0)',
                      legend=dict(
                          orientation='h',
                          title=None,
                          xanchor='center',
                          x=0.5,
                          yanchor='bottom',
                          y=1
                      ),
```

```
                                shapes=[dict(
                                    type='line',
                                    xref='x',
                                    yref='paper',
                                    x0=len(df9) - 4.5, # x position of line
                                    y0=0,
                                    x1=len(df9) - 4.5, # x position of line
                                    y1=1,
                                    line=dict(
                                        color='white',
                                        width=2
                                    )
                                )]

                            ),
                            config={'displayModeBar': False},
                            style={'width':'100%'}
                        )
                    ], style={'textAlign': 'center'}
                    )
                ]),
            )
        ])
```

APP LAYOUT

```
# App Layout
app.layout = dbc.Container(
    html.Div([
    dbc.Card(
        dbc.CardBody([
            # 1st Row
            dbc.Row(
                # Title
                dbc.Col(
                    html.Div([
                        dbc.Card(
                            dbc.CardBody([
                                html.Div([
                                    html.H1("BIOSEC SALES",
                                            style={"font-weight": "bold", "color": "white"})
                                ], style={'textAlign': 'center'})
                            ])
                        ),
                    ]),
                    width={'size':6,'offset':3},
                    align='center'
                ),
```

```
            align='center'
        ),
        html.Br(),
        # 2nd Row
        dbc.Row([
            # Year Dropdown
            dbc.Col(
                dcc.Dropdown(id='year_dropdown',
                             options=[{'label':year,'value':year} for year in dropdown_years]),
                width={'size':2,'offset':4}
            ),
            # Quarter Dropdown
            dbc.Col(
                dcc.Dropdown(id='quarter_dropdown',
                             options=[{'label':quarter,'value':quarter} for quarter in dropdown_quarters]),
                width=2
            ),
            # Submit Button
            dbc.Col(
                html.Button(id='submit-button-state', className='btn btn-light btn-sm', n_clicks=0,
children='Submit'),
                width=2
            )
        ], align='center'),
        html.Br(),
        # 3rd Row
        dbc.Row([
          # Average Lag (Order to Ship)
            dbc.Col(id='shipping_time',
                 align='center'
            ),
            dbc.Col(id='delivery_time',
                 align='center'
            ),
            dbc.Col(id='cancelled_orders',
                 align='center'
            ),
            dbc.Col(id='refunded_orders',
                 align='center'
            )
        ], align='center'),
        html.Br(),
        # 4th Row
        dbc.Row([
            # Revenue/Profit per Month
            dbc.Col(
                id='revenue_profit'
            ),
            # No. of Orders per Product
```

```
            dbc.Col(id='orders_per_product'
            )
        ]),
        html.Br(),
        # 5th Row
        dbc.Row([
            # Payment Methods
            dbc.Col(id='payment_methods',
                width=6
            ),
            # Product Categories
            dbc.Col(id='product_categories',
                width=6
            )
        ], align='center'
        ),
        html.Br(),
        # 6th Row
        dbc.Row(
            id='map_chart'
        )
    ]), color='dark', style={'height':'100%'}
  )
], style={'height':'100%'}))

if __name__ == '__main__':
  app.run_server(mode='inline')
```

10.5 Complete regression model script

```
#from dashboard import df_sales
import pandas as pd
from sklearn.model_selection import KFold, cross_val_score, train_test_split
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
from keras.layers import LSTM
import numpy as np

############################### PREDICTING COST ##############################
```

```
quarters = [
    '2015-03-31',
    '2015-06-30',
    '2015-09-30',
    '2015-12-31',
    '2016-03-31',
    '2016-06-30',
    '2016-09-30',
    '2016-12-31',
    '2017-03-31',
    '2017-06-30',
    '2017-09-30',
    '2017-12-31',
    '2018-03-31',
    '2018-06-30',
    '2018-09-30',
    '2018-12-31',
    '2019-03-31',
    '2019-06-30',
    '2019-09-30',
    '2019-12-31',
    '2020-03-31',
    '2020-06-30',
    '2020-09-30',
    '2020-12-31',
    '2021-03-31',
    '2021-06-30',
    '2021-09-30',
    '2021-12-31',
]

cost = [
    1255939.0,
    1521707.0,
    1456474.0,
    1493557.0,
    1502370.0,
    1035276.0,
    1145478.0,
    1492789.0,
    1235215.0,
    1386963.0,
    1456880.0,
    1586875.0,
    1199241.0,
    1284900.0,
    1259664.0,
    1416687.0,
```

```
    1110132.0,
    1333515.0,
    1527325.0,
    1273479.0,
    1241770.0,
    1631571.0,
    1310647.0,
    1290604.0,
    1188644.0,
    1516557.0,
    1972898.0,
    769849.0
]
df_cost = pd.DataFrame(columns={'date':quarters,'cost':cost})
df_cost['date'] = quarters
df_cost['cost'] = cost
df_cost
#represent month in date field as its first day
df_cost['date'] = pd.to_datetime(df_cost['date'])
df_cost = df_cost.drop([27])
#create a new dataframe to model the difference
df_diff = df_cost.copy()
#add previous sales to the next row
df_diff['prev_cost'] = df_diff['cost'].shift(1)
#drop the null values and calculate the difference
df_diff = df_diff.dropna()
df_diff['diff'] = (df_diff['cost'] - df_diff['prev_cost'])
df_diff
#create dataframe for transformation from time series to supervised
df_supervised = df_diff.drop(['prev_cost'],axis=1)
#adding lags
for inc in range(1,5):
    field_name = 'lag_' + str(inc)
    df_supervised[field_name] = df_supervised['diff'].shift(inc)
#drop null values
df_supervised = df_supervised.dropna().reset_index(drop=True)
#import MinMaxScaler and create a new dataframe for LSTM model
from sklearn.preprocessing import MinMaxScaler
df_model = df_supervised.drop(['cost','date'],axis=1)
#split train and test set
train_set, test_set = df_model[0:-4].values, df_model[-4:].values
#apply Min Max Scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(train_set)
# reshape training set
train_set = train_set.reshape(train_set.shape[0], train_set.shape[1])
train_set_scaled = scaler.transform(train_set)
# reshape test set
test_set = test_set.reshape(test_set.shape[0], test_set.shape[1])
```

```
test_set_scaled = scaler.transform(test_set)
X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
# Import statsmodels.formula.api
import statsmodels.formula.api as smf

model = Sequential()
model.add(LSTM(4, batch_input_shape=(1, X_train.shape[1], X_train.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, batch_size=1, verbose=1, shuffle=False)
y_pred = model.predict(X_test,batch_size=1)
#for multistep prediction, you need to replace X_test values with the predictions coming from t-1
y_pred
y_test
#reshape y_pred
y_pred = y_pred.reshape(y_pred.shape[0], 1, y_pred.shape[1])
#rebuild test set for inverse transform
pred_test_set = []
for index in range(0,len(y_pred)):
    print(np.concatenate([y_pred[index],X_test[index]],axis=1))
    pred_test_set.append(np.concatenate([y_pred[index],X_test[index]],axis=1))
#reshape pred_test_set
pred_test_set = np.array(pred_test_set)
pred_test_set = pred_test_set.reshape(pred_test_set.shape[0], pred_test_set.shape[2])
#inverse transform
pred_test_set_inverted = scaler.inverse_transform(pred_test_set)
#create dataframe that shows the predicted sales
result_list = []
sales_dates = list(df_cost[-7:].date)
act_sales = list(df_cost[-7:].cost)
for index in range(0,len(pred_test_set_inverted)):
    result_dict = {}
    result_dict['pred_value'] = int(pred_test_set_inverted[index][0] + act_sales[index])
    result_list.append(result_dict)
df_result_cost = pd.DataFrame(result_list)
#for multistep prediction, replace act_sales with the predicted sales
df_result_cost

############################### PREDICTING PROFIT ##############################

quarters = [
    '2015-03-31',
    '2015-06-30',
    '2015-09-30',
    '2015-12-31',
    '2016-03-31',
```

```
    '2016-06-30',
    '2016-09-30',
    '2016-12-31',
    '2017-03-31',
    '2017-06-30',
    '2017-09-30',
    '2017-12-31',
    '2018-03-31',
    '2018-06-30',
    '2018-09-30',
    '2018-12-31',
    '2019-03-31',
    '2019-06-30',
    '2019-09-30',
    '2019-12-31',
    '2020-03-31',
    '2020-06-30',
    '2020-09-30',
    '2020-12-31',
    '2021-03-31',
    '2021-06-30',
    '2021-09-30',
    '2021-12-31',
]

profit = [
    819512.0,
    907284.0,
    813413.0,
    800590.0,
    939577.0,
    663429.0,
    754774.0,
    881857.0,
    766385.0,
    788253.0,
    862276.0,
    890476.0,
    756649.0,
    755641.0,
    818690.0,
    846724.0,
    769633.0,
    825536.0,
    856835.0,
    775210.0,
    804575.0,
    918589.0,
    808254.0,
```

```
    774728.0,
    737457.0,
    871686.0,
    1108329.0,
    493734.0
]
df_profit = pd.DataFrame(columns={'date':quarters,'profit':profit})
df_profit['date'] = quarters
df_profit['profit'] = profit
df_profit
#represent month in date field as its first day
df_profit['date'] = pd.to_datetime(df_profit['date'])
df_profit = df_profit.drop([27])
#create a new dataframe to model the difference
df_diff = df_profit.copy()
#add previous sales to the next row
df_diff['prev_profit'] = df_diff['profit'].shift(1)
#drop the null values and calculate the difference
df_diff = df_diff.dropna()
df_diff['diff'] = (df_diff['profit'] - df_diff['prev_profit'])
df_diff
#create dataframe for transformation from time series to supervised
df_supervised = df_diff.drop(['prev_profit'],axis=1)
#adding lags
for inc in range(1,5):
    field_name = 'lag_' + str(inc)
    df_supervised[field_name] = df_supervised['diff'].shift(inc)
#drop null values
df_supervised = df_supervised.dropna().reset_index(drop=True)
#import MinMaxScaler and create a new dataframe for LSTM model
from sklearn.preprocessing import MinMaxScaler
df_model = df_supervised.drop(['profit','date'],axis=1)
#split train and test set
train_set, test_set = df_model[0:-4].values, df_model[-4:].values
X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
# Import statsmodels.formula.api
import statsmodels.formula.api as smf

model = Sequential()
model.add(LSTM(4, batch_input_shape=(1, X_train.shape[1], X_train.shape[2]), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, batch_size=1, verbose=1, shuffle=False)
y_pred = model.predict(X_test,batch_size=1)
#for multistep prediction, you need to replace X_test values with the predictions coming from t-1
y_pred
```

```
y_test
#reshape y_pred
y_pred = y_pred.reshape(y_pred.shape[0], 1, y_pred.shape[1])
#rebuild test set for inverse transform
pred_test_set = []
for index in range(0,len(y_pred)):
    print(np.concatenate([y_pred[index],X_test[index]],axis=1))
    pred_test_set.append(np.concatenate([y_pred[index],X_test[index]],axis=1))
#reshape pred_test_set
pred_test_set = np.array(pred_test_set)
pred_test_set = pred_test_set.reshape(pred_test_set.shape[0], pred_test_set.shape[2])
#inverse transform
pred_test_set_inverted = scaler.inverse_transform(pred_test_set)
#create dataframe that shows the predicted sales
result_list = []
sales_dates = list(df_cost[-7:].date)
act_sales = list(df_cost[-7:].cost)
for index in range(0,len(pred_test_set_inverted)):
    result_dict = {}
    result_dict['pred_value'] = int(pred_test_set_inverted[index][0] + act_sales[index])
    result_list.append(result_dict)
df_result_cost = pd.DataFrame(result_list)
#for multistep prediction, replace act_sales with the predicted sales
df_result_cost
```