Simple Retail Store Analytics Dashboard

Reece Calvin, Andrew Enelow, Thomas McDonagh, Nicholas Perrotta, Cam Plume

DS 4300: Final Project

Abstract:

For our DS 4300 Final Project, we developed a user-friendly online retail analytics dashboard geared towards small growing businesses, utilizing the "Online Retail" dataset from UCI. By utilizing programming languages and database platforms learned throughout our data science classes, such as Python, MongoDB, and JavaScript, alongside libraries such as Flask and React, we designed visualizations including spending trends by country, amount spent over time, and best-selling products. Additionally, the visualizations are not static as the user can interact with the application's drop-down menu to create a visualization for their specific needs. Overall, our dashboard democratizes access to analytics tools, allowing businesses of all sizes to make informed decisions to grow in the competitive retail space.

Current Industry Models:

Current models that exist to fill this area of the market mainly include SaaS (Software as a Service) solutions such as Tableau, Power BI (Microsoft), and Looker (Google).

Tableau is a stand-alone software that is renowned for its user-friendly interface and the ability to create advanced visualizations without database knowledge. However, it comes at a steep cost to businesses, ($50-75/user) and can be very complicated to connect to data sources to the software.

Power BI is a software among the Microsoft software collection that makes it easy for businesses already integrated within the Microsoft ecosystem. Power BI’s advantage comes from its low cost (option for a free version as well) and seamless integration with Excel and specific AI tools. However, it lacks the customization that Tableau or a custom-designed software would have which can limit users.

Looker is a software recently bought by Google that takes a different approach from the other two mentioned above; the key distinction is that it uses its unique LookML data modeling language which allows for far more customization, but it requires knowledge of the language to take advantage all the features Looker has to offer.

Introduction:

In today’s day and age, data and analytics have become an indispensable tool for businesses. In fact, according to a recent survey done by the Centre for Economics and Business

Research, about 80% (of companies surveyed in the study) have seen their revenues increase after implementing real-time analytics (1). Traditionally, supporting data initiatives has been costly and hard to maintain. Thus, only large companies with more money and resources have been able to leverage their data to analyze sales trends, understand customer behavior, and make data-driven business decisions. But times are changing. With our project, we wanted to test our hand in developing a simple sales retail dashboard and prove that with modern programming languages, database platforms, and a few YouTube videos, the power of data analytics can be brought to any local small business or mom-and-pop shop. We sourced our data from UCI, leveraging a dataset called "Online Retail" to mimic transactions for a small business. We chose this dataset in particular because it came from a reputable source and we felt it was a good representation of what small business data could look like. From there, we decided to implement our backend using Python and MongoDB. We chose to work with MongoDB because we find it simple to interact with programmatically and we thought that the document-store functionality was perfect for our data. As for the dashboard itself, we used Javascript, Flask, and Native to create a fully functioning, dynamic front end that allows users to conduct advanced analysis in mere seconds. We hope that our project emphasizes that the future is now, and even local brick-and-mortar stores will soon be able to leverage data insights and compete with contemporary titans of industry.

Instructions To Run a Development Version of our Application:

Before attempting to run our project, please make sure you have installed the following central pieces of software. There are extensive instructions available on the pages linked below:

- NodeJS: https://nodejs.org/en
- Python 3+: https://www.python.org/downloads/
- MongoDB: https://www.mongodb.com/try/download/community

Next, install the following web development dependencies using the commands given:

- Flask: pip install flask
- React: npm install -g create-react-app
- CORS: npm install cors -> This may c

Third, install necessary data science dependencies

- pip install pymongo
- pip install matplotlib
- pip install pandas
- pip install numpy
- pip install scikit-learn

Now that we have all our dependencies installed, start your MongoDB server. There are many different approaches to this, but the easiest is as follows:

- Add MongoDB to your system PATH variables
- Execute terminal command "mongod"

Next, clone our project repo from the following repository (https://github.com/CamPlume1/online_retail_dashboard), or download it from our gradescope submission.
Load the dataset to your MongoDB instance through the following commands:

- cd data_upload
- python main.py

In order to run our flask backend, run the following commands in a new terminal

- cd backend
- Flask run

Finally, in order to run the application, in a new terminal, run the following terminal commands in a new terminal

- Cd frontend
- Npm start

The application should now be available for viewing and interaction on any browser at localhost:3000

Project Structure

There are two central pieces of our program, a backend and frontend.

Backend: We developed our backend using Flask, following a REST API structure. Our backend supports the following API Calls. API calls should be made using HTTP.

```
# Gets graphic depicting customer profiles by product
@app.route('/api/cam_graphic/', methods=['GET'])

# API Root function for testing, returns “Hello World”
@app.route('/')

# Gets all the unique years in the dataset
@app.route('/years/')

# Gets line chart comparing annual sales between countries
@app.route('/api/nick_graphic/', methods=['GET'])

# Gets graphic displaying monthly sales and trends by country
@app.route('/tom_graphic/', methods=['GET'])

# Gets graphic displaying top products by revenue
@app.route('/api/reece_graphic/', methods=['GET'])
```

```
# Gets all the unique countries in the dataset
@app.route('/api/countries')

# Gets the number of unique countries in the dataset
@app.route('/api/countries_count')
# Gets the total number of transactions in the database
@app.route('/api/total_transactions')

# gets the total sales in British Pounds Sterling from the database
@app.route('/api/total_sales')

# Gets all unique product descriptions
@app.route('/api/unique_descriptions')

# gets the total number of units sold
@app.route('/api/total_units'))

# Gets the unit purchased most (by quantity)
@app.route('/api/top_unit')

# Gets the description of the product that generates the most revenue
@app.route('/api/top_unit_rev')

# Gets andrews graphic-> Alternate view of monthly sales without trend line
@app.route('/api/andrew_graphic/')
```

Frontend: We developed a frontend for our application using Javascript and React. Importantly, none of us had any meaningful experience in web or application development, and as such there are certain to be design, style, and potentially functionality flaws in this application. While we may have bitten off more than we could chew, we were very happy with the result.

There were five key components of our frontend, described below.

Stats: Our first section is a statistics window that displays live, non-interactive statistics about the paired dataset as a whole. We developed this section in a component, located at frontend/src/components/stats.js. This component functions by making an API call for each statistic, stored in state variables, and displayed in text components

ImageContainers: The next four components are very similar, and follow nearly identical designs. First, there is an interactive feature, allowing users to make selections. The selection

options are made by making backend API calls. User selections are then stored in state variables, as well as ImageComponent dependency arrays. User selections trigger an API call to the backend, retrieving a parameterized graphic. These components in particular could have been abstracted in a much more intelligent manner. Our lack of previous experience, or even working knowledge with JavaScript prevented us from using proper patterns of inheritance or abstraction.

App: Our central app framework is contained in frontend/src/App.js

The implementation and result of our application is discussed in more detail below.

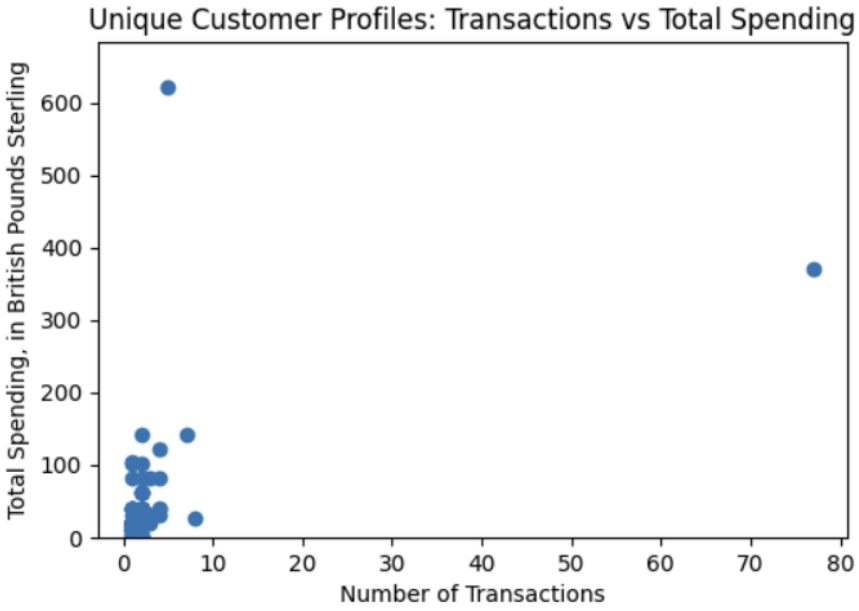
Methodology and Feature Implementation

Stats:
When a user opens our application, they are greeted by a statistics window, showing some descriptive statistics of their retail data.

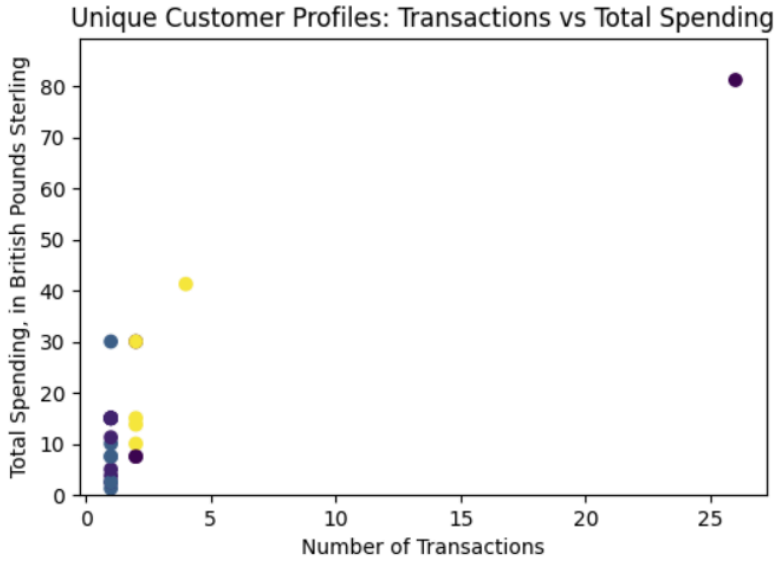
While this section is not interactive, it allows users to understand some overall insights into their business, including top products, sales, and operating markets. This feature was implemented using six API calls, implemented in component_calls.py.

Customer Profiles:
Our intention for the customer profiles feature was for it to be dual purpose. Users should be able to inspect the trends of their business at large, in addition to inspecting subgroups of products to see how those groups are performing.
We implemented this by creating a pipeline that grouped transactions based on customer id, summed the number of transactions, and summed the value of each transaction.
When no products are selected, the aggregation is performed on the entire dataset.
When more than one product is selected, color encoding is added to the visualization.

Example of single product customer profiles:



Example of multiple product customer profiles:



Note: Product labeling is handled in the frontend.

Amount Spent by Country:

For one of our visualizations on our dashboard, we wanted to visualize, over time, the amount spent on online retail by country. For the backend, the API script, “mongo_api.py” has a method, “spending_by_month_and_country” that uses a cypher query to access the spending for each country for each month as seen below.

```
# Cypher Query
pipeline = [
    {"$project": {
        "Month": {"$month": "$InvoiceDate"},
        "Country": 1,
        "TotalAmount": {"$multiply": ["$Quantity", "$UnitPrice"]}
    }},
    {"$group": {
        "_id": {"Month": "$Month", "Country": "$Country"},
        "TotalAmount": {"$sum": "$TotalAmount"}
    }}
]

# Runs cypher query
results = collection.aggregate(pipeline)
```

The method will return the queried data as a dictionary where each month is a key and each country's spending for that month is the value. Storing the data in a dictionary will allow a visualization to be easily produced in another method. The "plot_country_spending (specified_countries)" method visualizes each country's spending by month. The parameter "specified_countries" will only visualize the countries that are specified in the list. This will allow for functionality on the dashboard as the user can select the countries from a dropdown menu that the user wishes to see. This method is called "spending_by_month_and_country" as seen below to have the data in a dictionary needed to create the visualization.

```
# Given the selected countires, vizualizes their spending by month
def plot_country_spending(specified_countries):

    # Gets cypher queried data as a dictionary and coverst to a data frame
    data = spending_by_month_and_country()
    df = pd.DataFrame(data).T
```

Spending Trends by Country:

For our next visualization, we wanted to drill down a level and visualize the spending trends of user-selected countries in 2011. We decided to just visualize 2011 data due to the nature of our dataset, as it only had sales data for December of 2010. Thus, given the context of our project, we found it redundant to include 2010 data and instead decided to just give users a comprehensive look at the year 2011. Furthermore, we thought it would be nice to include a trendline, which we could implement through the machine learning library sci-kit learn. Regarding the code itself, we first used Pymongo and a simple Python script to import our sales data to a MongoDB instance. From there, we used the "get_data" function in the "mongo_api.py" script to get our data. The function "get_data" uses Pymongo and a cypher query to query the dataset and return how much was spent in a respective country, grouped by month. Below you can see our logic for pulling the data.

```
pipeline = [
    {
        '$match': {
        'Country': Country,
        '$expr': { '$eq': [{ '$year': "$InvoiceDate" }, 2011] }
        }
    },
    {
        '$project': {
        'month': { '$month': "$InvoiceDate" },
        'revenue': { '$multiply': ["$UnitPrice", "$Quantity"] }
        }
    },
    {
        '$group': {
        '_id': "$month" ,
        'totalRevenue': { '$sum': "$revenue" }
        }
    },
    {
        '$sort': { "_id": 1 }
    }
    ]
```

Then, we converted the resulting query into a data frame for easy analysis and visualization, and added a column to that data frame where we converted numerical months (i.e, 1,2,3), returned by the cypher query, into corresponding month names (i.e., January, February, March) using a helper function called “number to month”. Once we had the data in a format we liked, the function “gen_country_graphic” in the “mongo_api.py” script used the pulled data to create a bar chart. The function also uses a helper method called “linear regression” to compute the trendline. Regarding the visualization itself and its functionality, the analysis section will look more closely at what can be produced from the aforementioned functions.

Total Quantity Sold by Year by Country:

*Note: This is not in the final version but we took the time to make it and it exists in the code

Another visualization we wanted to create was to visualize how the quantity of products sold varied by month. This can be filtered by year and by country to better understand purchasing trends in across years and by differing countries. The code itself uses Pymongo and a cypher query to get quantity data from a start time and end time determined by the year inputted and filters by the country inputted. This data is then saved in a pandas dataframe to store the data in memory. Another method then takes this data and uses matplotlib to vizualize it accordingly.

```
pipeline = [
    {'$match': {'Country': country, 'InvoiceDate': {'$gte': start_date, '$lte': end_date}}},
    {'$project': {'YearMonth': {'$dateToString': {'format': "%Y-%m", 'date': "$InvoiceDate"}},
    'Quantity': "$Quantity"}},
    {'$group': {'_id': "$YearMonth", 'TotalQuantity': {'$sum': "$Quantity"}}},
    {'$sort': {'_id': 1}}
]
```

Best Selling Products:

Another visualization that we wanted to create was to visualize the best-selling products in 2011. We used the "best_selling_products" method in the "mongo_api.py" script to do this. best_selling_products takes in a year as a parameter, for this, we used 2011. This method uses a cipher query to access the data for the best-selling products in 2011. The method will return the queried data as a dictionary where the key is the product description and the value is the total quantity sold. The data is then stored in a pandas DataFrame and visualized using matplotlib.

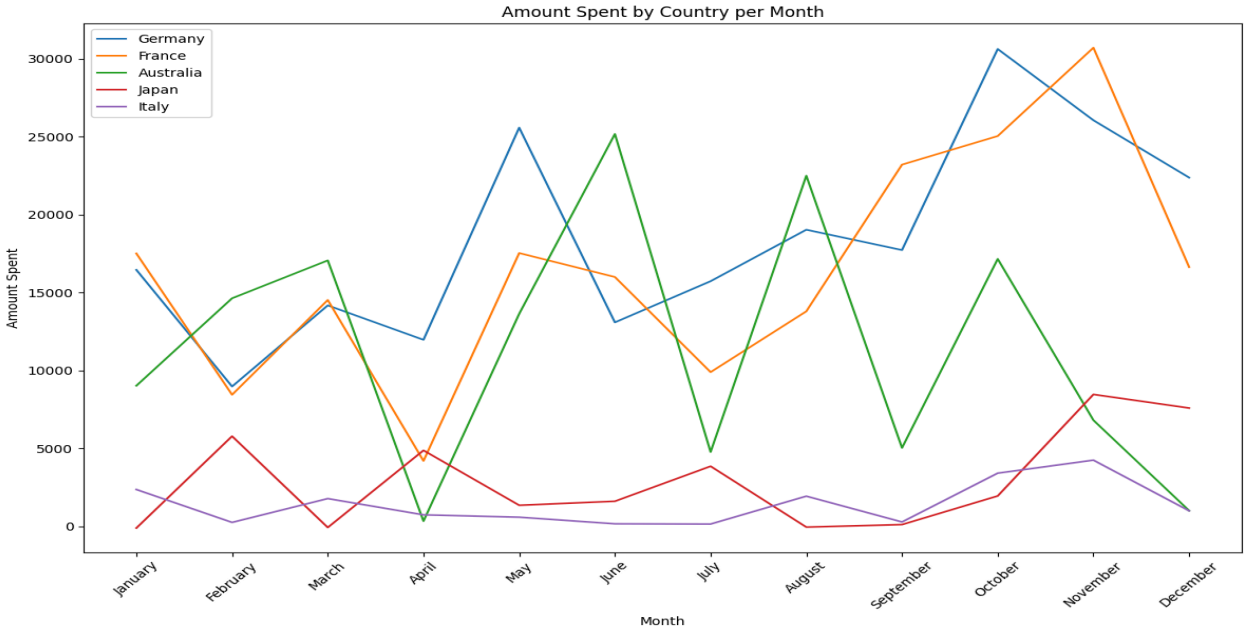
```
pipeline = [
{'$match': {'$expr': {'$eq': [{'$year': "$InvoiceDate"}, year]}}},
{'$project': {'Description': 1, 'Quantity': 1}},
{'$group': {'_id': "$Description", 'TotalQuantity': {'$sum': "$Quantity"}}},
{'$sort': {"TotalQuantity": -1}},
{'$limit': 10}
]

results = collection.aggregate(pipeline)
```

Feature Analysis

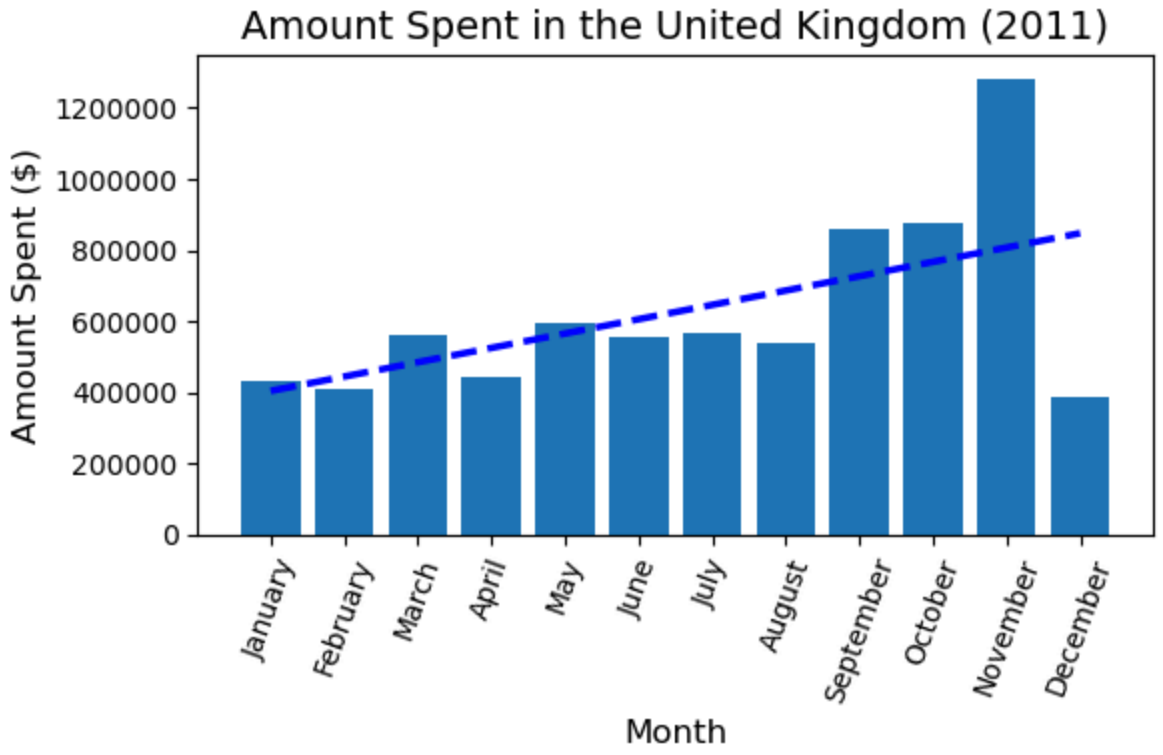
Amount Spent by Country:

As seen by the graph below, the method"plot_country_spending(specified_countries)" produces a time series line graph showing Germany's, France's, Australia's, Japan's, and Italy's online retail spending each month. As previously mentioned, each country was selected from the dashboard's drop-down menu to include it in the graph. Each country is clearly distinguished via a legend to allow for easy analysis. This graph is useful as a user can look at a specific country's online retail spending trends. For example, this graph shows how Germany spends more on online retail than Italy. Additionally one can observe the overall online retail spending trends by month. For example, one could deduce from the provided countries that online retail spending increases from September to October. This could produce valuable insight for online retail stores as they now know that they will need to make sure to increase their inventory for October as sales are typically larger than sales in September.



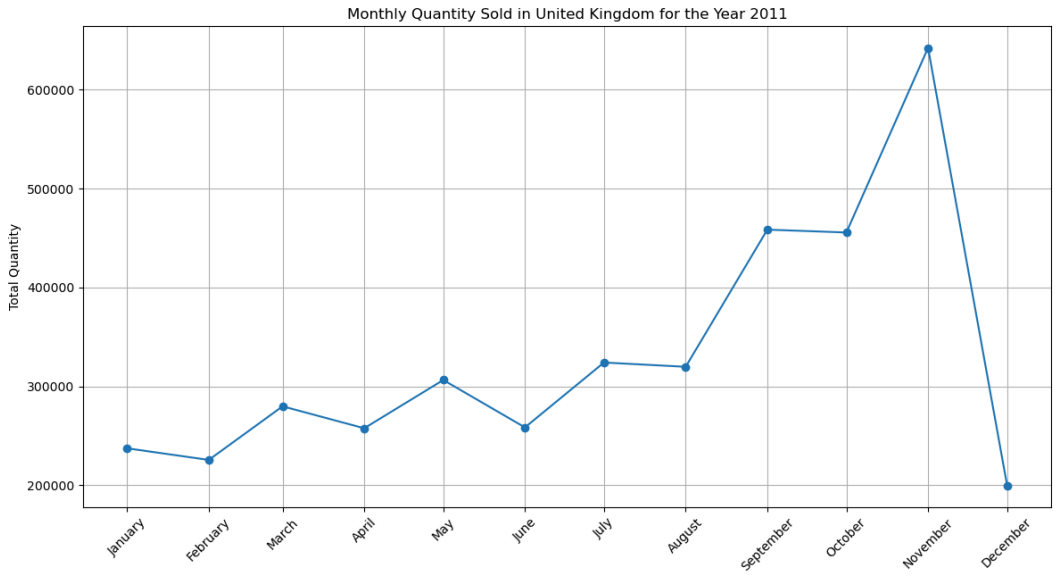
Spending Trends by Country:

The graph below depicts an example visualization of the United Kingdom, created using the method “gen_country_graphic” in our “mongo_api.py” script. It produces a bar chart of month-over-month expenditure, with a dotted trendline to indicate the direction of overall spending for the user-selected country. To select a country, users will select from a dropdown menu in our frontend, and a country-specific graph will be displayed. We thought the visualization would be useful to add because it works nicely in tandem with the “Amount Spent by Country” visualization seen above. For example, users can identify a specific country of interest using the “Amount spent by Country” visualization, and then drill down using the “Spending Trends by Country” to get a holistic picture of how one country might compare to another, as well as individual trends. Furthermore, the chart is useful in identifying seasonality trends, overall spending direction, and months in which overall expenditure is inconsistent. For example, in the United Kingdom, a user might see that overall expenditure has been steadily increasing from January 2011 to November 2011, but then drops way off in December. Using this information, they can delve into whether there were any supply chain disruptions, business strategies, etc. leading to the massive fall-off in spending.



Monthly Quantity Sold by Year by Country:

The graph below depicts the quantity of products sold by month for the example variables chosen. For this specific visualization, we chose to pick the year 2011 and the United Kingdom. We chose to depict this data as a line chart because it shows the rate of change between months very well. In the user-interface, a dropdown menu can be used to select the country and year to filter the data to the user's needs. The data shows an increasing correlation between the changing months and the quantity sold. There is a slow steady increase in the winter to summer months and then a sharp increase in the fall with a big decline in december. This information can be useful to a user of the system because it can show how much inventory should be stored on a monthly bases based on historical data.



Best Selling Products:

This graph shows the bestselling products, in terms of units sold, for a given year. This can be helpful when businesses are reviewing last year's sales and can use the outputted data to estimate future demand. For example, in 2011, the World War 2 Gliders were the best-selling item with over 53k units sold. A company selling the gliders would be best to prioritize it when restocking for the next year.

Why MongoDB & Conclusion:

Our choice of MongoDB ensured us flexibility, scalability, and ease of use as businesses are ever-changing and we felt we created a dashboard that is mutable to fit up-to-date trends. MongoDB's flexible schema allows us to integrate many different platforms easily as features can be added later without the need for large migrations. The aggregation features allow low-latency queries for the analytics we made. Additionally, MongoDB's excellent scalability features empower growing businesses to handle expanding data pipelines with ease. However, it's essential to note potential downsides. A lack of joins in MongoDB can make analytics across document types complex and slow, which limits the possibility of integration of other types of data. Lastly, while SQL is widely understood, individuals lacking data engineering skills may encounter difficulties in understanding and extending our platform.

In conclusion, our work highlights the massive upsides businesses of all sizes have in leveraging data to help in decision-making. For example, German and French stores can see that they will fall behind if they do not adapt to the online market, in the winter months, online sales soar through the roof, and Japan sees a disproportionate lack of online sales. Businesses can use this information to know how much they should commit to selling online, which products to stock up on, and when to expect increased traffic.

References:

1. KX News. “Real-Time, Real Value: 80% of Businesses See Revenue Increases Thanks to Real-Time Data.” *Www.prnewswire.com*, 9 May 2022, www.prnewswire.com/de/pressemitteilungen/real-time-real-value-80-of-businesses-see-revenue-increases-thanks-to-real-time-data-870705221.html. Accessed 8 Apr. 2024.
2. Chen,Daqing. (2015). Online Retail. UCI Machine Learning Repository. https://doi.org/10.24432/C5BW33.