



Design and Implementation of Data Architecture and Data Processing Pipelines

Assignment 2 Report

Data Engineering Course

Assignment Github Repository:
<https://github.com/jads-group8/de2>

Berkay Kulak, Diogo Rio, Doruk Karakas, Krishna Teja Atluri

and Mikolaj Hilgert

Group 8 | Dec 1, 2024

Table of contents

1. Overview of the Data Pipelines	1
1.1 Problem Overview	1
1.2 Data Pipeline goals and data description	1
2. Design and Implementation of Data Architecture	2
3. Design and Implementation of Data Pipelines	4
4. Reflection on Design and Implementation of Data Architecture and Data Pipelines	5
4.1 Alternative design discussion	5
4.2 Implementation reflection	5
5. Individual Contributions of Students	6
6. References	7
7. Appendix	8
Appendix 1. Dashboard images	8
Appendix 2. CronTab Setup	9

1. Overview of the Data Pipelines

1.1 Problem Overview

As e-commerce expands amid fierce competition and rising consumer expectations, businesses struggle to manage vast data effectively. Without actionable insights, they miss trends, optimise inventory poorly, and make slower decisions, leading to inefficiencies and lost revenue (Shopify, 2023). Moreover, ineffective data processing and visualisation risks leaving businesses behind competitors (Waite, 2024).

1.2 Data Pipeline goals and data description

The dataset used consists of a transactional star schema of e-commerce stores (Islam, 2024), with a fact table capturing key metrics like quantity, price, and total sales, linked to five dimension tables: customers, items, stores, time, and payments. Each dimension table provides detailed contextual data, enabling analysis across customer demographics, product details, store locations, time periods, and payment methods.

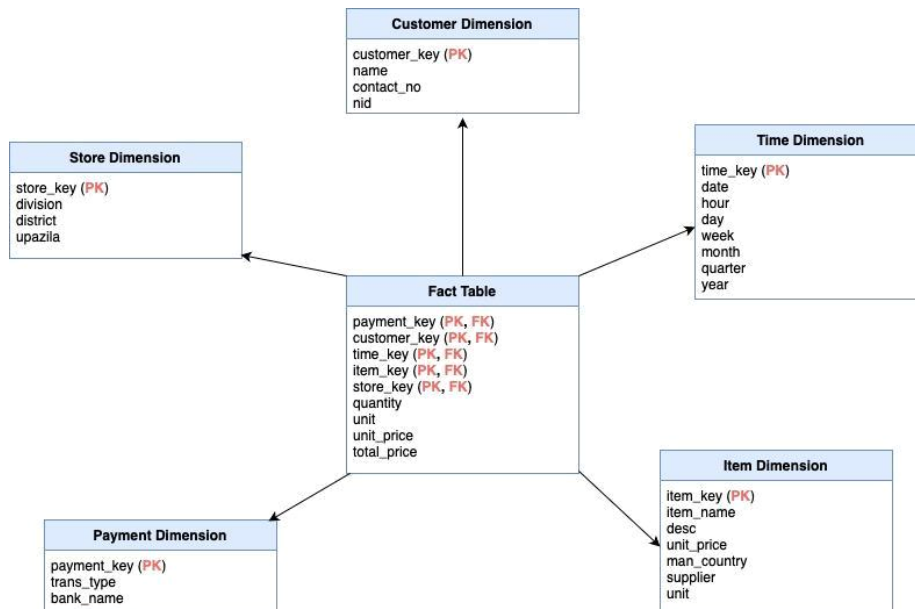


Figure 1: Star schema of e-commerce database

The goal of this project was to develop three interactive dashboards to provide stakeholders with actionable insights into real-time store sales performance, best product categories, and high-performing e-commerce stores. These dashboards leverage streaming and batch data pipelines, integrating transactional data with auxiliary dimensions to enrich insights. Below is an overview of the dashboards and their implementation:

1. Real-Time Sales Dashboard

This dashboard delivers a live view of sales trends, payment methods, and performance metrics, enabling stakeholders to monitor business activity in real-time. This streaming pipeline consumes transactional data from a Kafka stream, parses and enriches it with item, store, and time dimensions, and writes the processed data to a BigQuery table for visualisation with data updates every 1 minute (Looker Studio constraint).

2. Top Product Categories Dashboard

This dashboard identifies high-performing product categories across all of the e-commerce stores, providing insights into sales contributions and category-level performance for inventory and marketing optimisation. Using a batch pipeline, it aggregates sales and quantities at the category level, calculates the average price per item, and determines category contributions to total store sales. This allows any new stores to act on the data-driven insights gathered by our dashboard, as it extracts the top-performing category for each store - and consequently form an understanding of what likely sells well.

3. Top 100 E-commerce Stores Dashboard

This dashboard highlights the top 100 stores based on total sales, along with their best-selling products. A batch pipeline processes historical sales data, ranks stores by sales, and identifies the highest-grossing products within each store. By focusing on key revenue drivers, this dashboard helps stakeholders identify successful products.

These dashboards can all be accessed through this single Looker studio [link](#), or are available in [Appendix 1](#).

2. Design and Implementation of Data Architecture

The architecture implementation diagram is presented below:

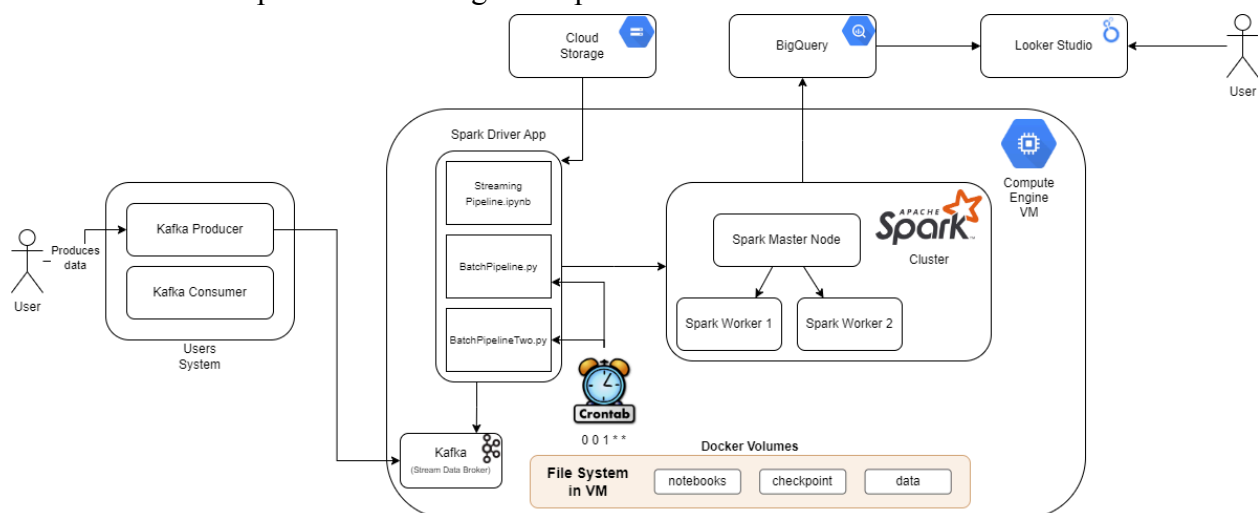


Figure 2: Data architecture implementation overview

The data architecture outlined in Figure 2, is one that is based on Microsoft Big Data Architecture from the labs, and is designed to integrate real-time streaming and batch processing workflows. The implementation integrates tools such as Apache Kafka, Apache Spark, Google Cloud Storage, BigQuery, and Looker Studio to provide an automated and efficient solution to derive business insights from raw data.

The real-time streaming component utilises Apache Kafka as the stream data message broker. Transaction data, which has been transformed to a JSON format, is continuously sent by the producer component and published as events to the relevant Kafka topic. These Kafka events are consumed by the real-time pipeline (*StreamingPipeline.ipynb*), which uses Apache Spark's streaming capabilities to process the incoming data. The pipeline applies data transformations and combines it with relevant auxiliary dimensions, fetched from Google Cloud Storage. This transformed data is then stored in BigQuery, where it is used for the real-time sales dashboard.

In addition to real-time processing, the architecture includes batch data pipelines to handle historical data. These pipelines are also implemented using Apache Spark. Two distinct batch pipelines have been created: one for aggregating sales data at the category level to generate insights for the Top Product Categories Dashboard, and another for processing historical sales data to identify the Top 100 E-commerce Stores and their best-selling products. These pipelines load the raw *csv* data from Google Cloud Storage, process it in Spark, and save the results into BigQuery. Automation is also a key feature of this setup. The use of Crontab ensures that batch pipelines run at scheduled intervals, of once a month without manual intervention, enabling the system to process new data automatically. The setup screenshot can be found in [Appendix 2](#).

All pipelines in this architecture write their results to BigQuery, which serves as the central sink. As Google's native data warehouse solution, BigQuery is highly adaptable and supports a wide range of use cases. In this instance, we leverage it to use it as a data source for dashboards built in Looker Studio. This integration is particularly effective because BigQuery's native compatibility with Looker Studio ensures smooth data visualisation and near real-time insights.

The entire system runs on Google Cloud's Compute Engine. Running everything there, apart from the Kafka producer and consumer, simplifies the setup since all existing firewall rules and API accesses are already configured. Kafka and Spark are run using Docker Compose, further simplifying setup. The system utilises two Spark workers. This means that the streaming pipeline and one batch job can run simultaneously. Thanks to Spark's orchestration capabilities, this is not an issue- once the first batch pipeline has completed, the second batch can run on the second worker, even if the streaming worker remains continuously occupied.

3. Design and Implementation of Data Pipelines

Figure 3 provides a detailed illustration of the implementation of three pipelines, each designed to handle different transformations to deliver the desired results. Each rounded rectangle specifies the steps taken within the PySpark program, and the arrows show the order of execution. Each pipeline includes a step to pull relevant data dimension files stored in a Google Cloud Storage bucket. The streaming pipeline is the only one that is designed to consume real-time transaction Kafka events. The processed data is then saved as separate tables in the BigQuery data warehouse. The dashboards in Looker Studio dynamically fetch the data from BigQuery.

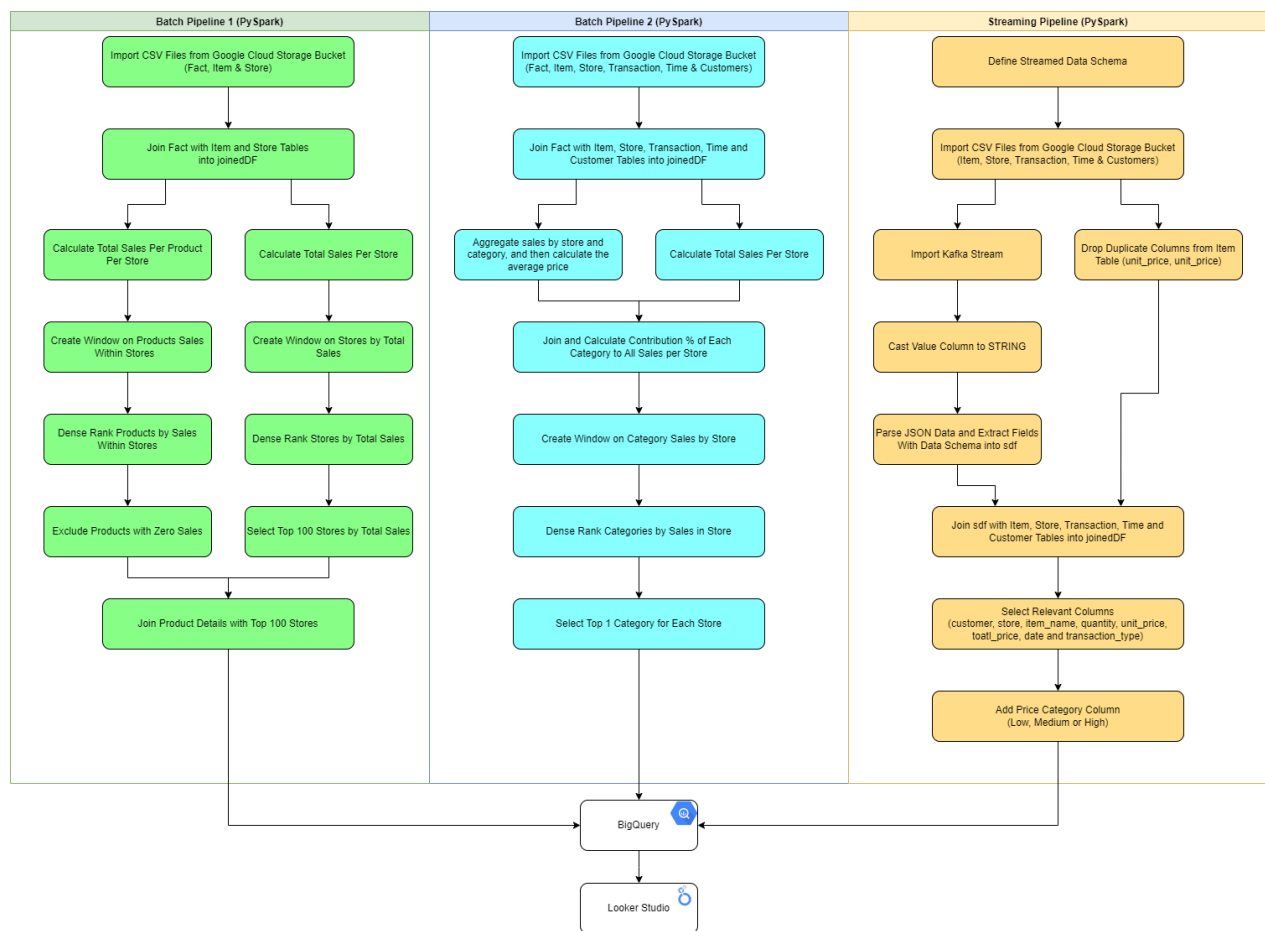


Figure 3: Pipeline Flowchart overview

4. Reflection on Design and Implementation of Data Architecture and Data Pipelines

4.1 Alternative design discussion

Looking at the problem statement, we can consider potentially making changes to our design or implementing other architectures that would also fit the business goals.

Currently, the design has limitations because we "mock" incoming transactions by sending events from the user's computer. Instead, the e-commerce stores should directly publish these events to the Kafka topic. Moreover, hosting the Kafka instance and running Spark through a notebook within a VM is not ideal for production workloads. A more robust approach would be to deploy Spark on a distributed cluster, such as Kubernetes, or a managed Spark service (e.g., Databricks). This separation would ensure higher scalability and fault tolerance.

Considering an alternative architecture, the Lambda architecture could address some limitations of the current design. Currently, we integrate real-time and batch processing within the same shared infrastructure, but utilise two different data sources. In contrast, a Lambda architecture separates these workflows into distinct layers while allowing the results from the speed layer (streaming) to drive the batch processing. The batch layer would handle large-scale processing of our immutable historical data, and the speed layer processes real-time data to provide immediate insights. A dedicated serving layer then unifies the outputs from both layers, enabling dashboards or queries to combine the results comprehensively. This differs from our current approach, where each pipeline writes directly to the distinct tables in BigQuery.

This sort of separation would offer more modularity and therefore fault tolerance. By storing raw and immutable data in the batch layer, the system can reprocess the historical data efficiently. In general, transforming the current architecture to this one would introduce complexity but it would result in a more robust system.

4.2 Implementation reflection

The implementation itself went quite smoothly, especially for the batch pipelines, due to the fact that the only dependency is on Google Cloud Storage buckets and writing to BigQuery. Given that we pull all the data at once, conceptually, this is much easier to do, as there is less need to integrate with other services (such as Kafka). Moreover, there is a lot of information available on how to use PySpark, which made it easier to implement all the transformations that we wanted. Given that the dataset is quite extensive, it also made it relatively easy to create some complex manipulations.

We ultimately ended up with three pipelines due to limitations in Spark operations within the streaming pipeline. This was because more advanced operations were illogical for the use case of a live sales dashboard. To achieve the necessary complexity for the assignment, we therefore decided to create an additional batch pipeline. Moreover, despite its challenge, setting up the streaming pipeline provided a valuable learning opportunity. One notable issue we encountered was forgetting to clear the topics, which caused confusion when unexpectedly large numbers of events were processed instead of the expected small batches. To debug this issue we made use of the recommended Kafkacat program to read the topic and noticed the discrepancy in incoming data.

For everyone in the group, all of these technologies were new and had their associated learning curves, we are however pleased with the final result.

5. Individual Contributions of Students

Mikolaj: I again worked as the lead for this project. I was in charge of maintaining and overseeing the development, progress and quality of the code and report. I worked with the group to develop the batch pipelines and created the streaming pipeline. I also contributed to the discussion sections of this report.

Krishna Teja: I contributed to defining the top product category batch pipeline by collaborating with the team to outline key metrics and potential processing steps. This included designing the workflow to aggregate sales data as well as calculate category-level contributions. Additionally, I helped in the crontab setup.

Berkay: I contributed in designing and refining the top products batch pipeline and wrote the first sections of the report and created the star schema data diagram.

Diogo: Helped with the development of the top products batch pipeline and with the writing of its section. Also conducted overall report quality control.

Doruk: I've contributed in the design of the batch data pipeline and problem definition of the project. I've taken part in the discussion in determining our approach and the documentation of the projects' Problem Overview and Design and Implementation of Data Pipelines.

6. References

Waite, R. (2024, April 17). E-Commerce Analytics: Making Data-Driven Decisions For Growth. <https://www.robinwaite.com/blog/e-commerce-analytics-making-data-driven-decisions-for-growth>

Shopify. (2023, June 29). Ecommerce Data Management: Types and Best Practices. <https://www.shopify.com/blog/ecommerce-data-management>

Islam, M. M. (2024). ECommerce Data Analysis. Retrieved December 1, 2024, from Kaggle.com: <https://www.kaggle.com/datasets/mmohaiminulislam/ecommerce-data-analysis>

7. Appendix

Appendix 1. Dashboard images

Top 100 Store Dashboard

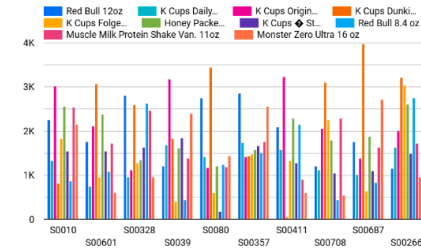
Top 100 Stores by total sales

store_key	top_store_rank
1. S0010	1
2. S00601	2
3. S00328	3
4. S0039	4
5. S0080	5
6. S00357	6
7. S00411	7
8. S00708	8
9. S00687	9
10. S00266	10

Total

productLsales
15.33M €

Top 10 Best-Selling Products by Sales Contribution (€)



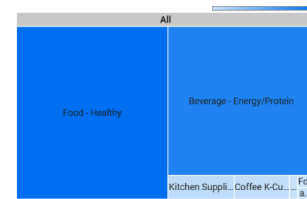
Top Product Categories Dashboard

store_key	desc	category_sales	avg_price_per_item	contribution_pct
1. S00530	Beverage - Energy/Prot...	20765	35	13.94
2. S0062	Beverage - Energy/Prot...	20405	35	13.76
3. S00243	Beverage - Energy/Prot...	19151	34	13.19
4. S00633	Food - Healthy	17698.5	17	13.07
5. S00492	Beverage - Energy/Prot...	19401	37	13.01
6. S00422	Beverage - Energy/Prot...	18111	37	12.97
7. S00582	Beverage - Energy/Prot...	18692	34	12.93
8. S00262	Beverage - Energy/Prot...	18861	36	12.81
9. S00576	Beverage - Energy/Prot...	19537	38	12.78
10. S00621	Beverage - Energy/Prot...	18872	39	12.77
11. S00684	Beverage - Energy/Prot...	18208	36	12.76
12. S0015	Beverage - Energy/Prot...	19298	36	12.76
13. S00604	Food - Healthy	18332	16	12.75
14. S00367	Beverage - Energy/Prot...	18584	36	12.71
15. S00277	Beverage - Energy/Prot...	17952	34	12.69

Store count

Record Count
726

Most common top categories



Real Time Sales Dashboard

Total Sales

total_price
343.8K

Average

total_price
106.75

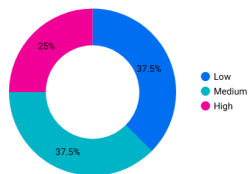
Min

total_price
6

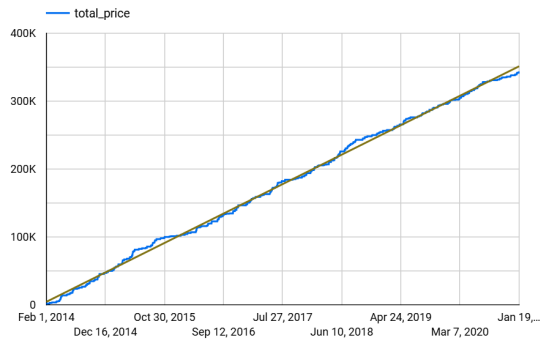
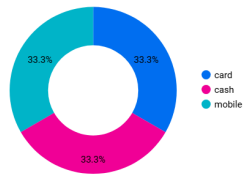
Max

total_price
605

Product price category



Payment type

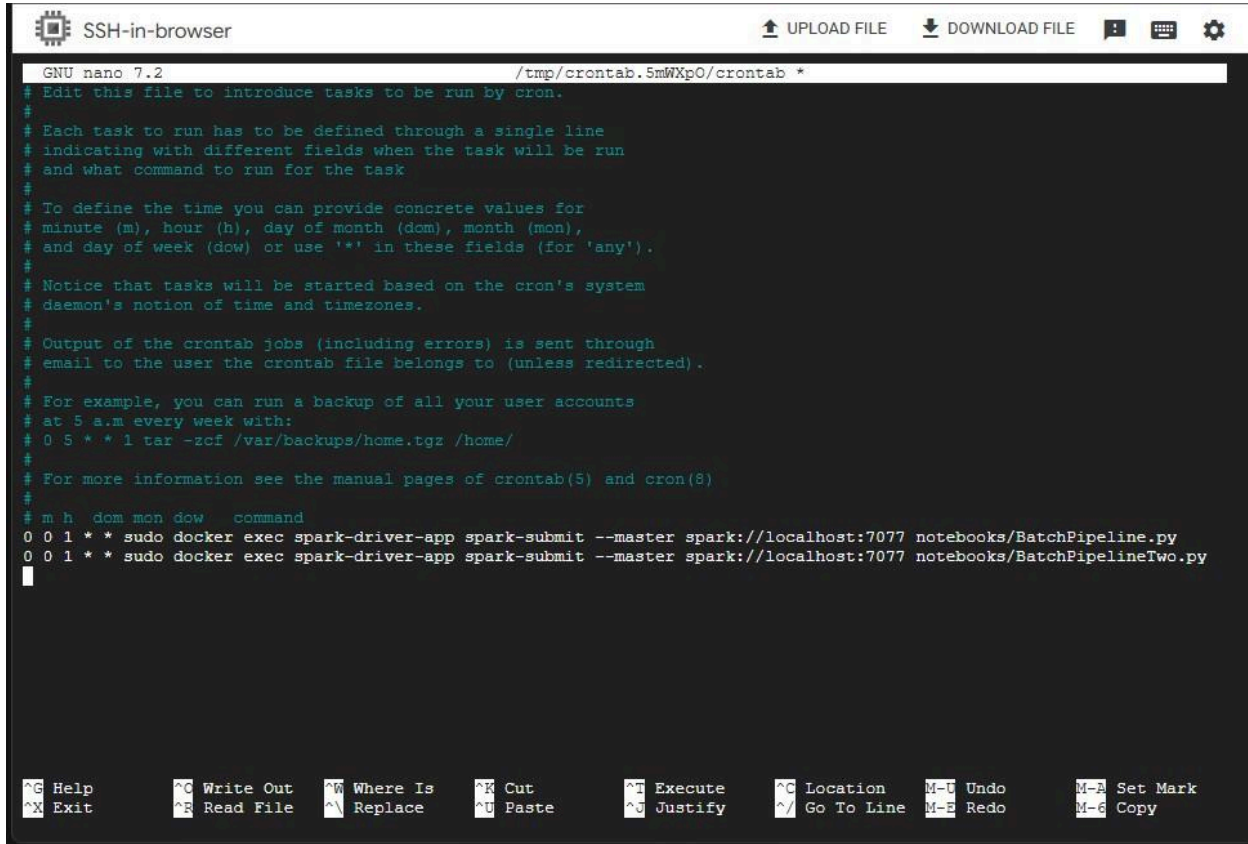


Select store

store

Select date range

Appendix 2. CronTab Setup



```
GNU nano 7.2 /tmp/crontab.5mWXP0/crontab *
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m. every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
0 0 1 * * sudo docker exec spark-driver-app spark-submit --master spark://localhost:7077 notebooks/BatchPipeline.py
0 0 1 * * sudo docker exec spark-driver-app spark-submit --master spark://localhost:7077 notebooks/BatchPipelineTwo.py
```

^G Help ^C Write Out ^W Where Is ^X Cut ^T Execute ^O Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^_ Replace ^U Paste ^J Justify ^_/ Go To Line M-B Redo M-C Copy