Project 3 (Business Insights Assessment)

Data Downloading

First of all, I downloaded all three data from Google Drive manually to do some exoploratory analysis on them in my local machine. The fils are 2 fact tables and 1 dim table:

order_items.csv
order_item_options.csv
date_dim.csv

Fortunately, I can read them using Pandas library without in the problem or file integrity issues. Then I tried to analyze deeply these files, like duplication rows or columns, missing NA values, histogram of all numerical columns, and calculating outliers. I will do all these activities in the transformation layer automatically later.

Data Duplication:

After checking duplicate rows and columns in these 3 files, I found that there are no duplicate rows and columns in two files: **order_items.csv**, **date_dim.csv**.

```
No duplicate rows or columns found
```

However, in the file **order_item_options.csv** the are 2299 records are duplicated in 6 columns. I removed them successfully.

```
Duplicate rows:
                        ORDER_ID              LINEITEM_ID OPTION_GROUP_NAME OPTION_NAME  OPTION_PRICE  OPTION_QUANTITY
1970    5f19da32505ee9e8467b2403  5f19da33505ee9e8467b2405       Vegetables    Escarole           0.0                1
2035    5f5bd259535ee95811972b8a  5f5bd26a535ee9de14972b10            Bread          No           0.0                1
2036    5f5bd259535ee95811972b8a  5f5bd26a535ee9de14972b10            Bread          No           0.0                1
2037    5f5bd259535ee95811972b8a  5f5bd26a535ee9de14972b10            Bread          No           0.0                1
2038    5f5bd259535ee95811972b8a  5f5bd26a535ee9de14972b10            Bread          No           0.0                1
...                          ...                       ...              ...         ...           ...              ...
192581  5f71b48b505ee9e10fa8bc2d  5f71b4a0505ee9170ea8bc4b      Milk Options          No           0.0                1
192765  5f71c1e04f5ee93a5f72a46f  5f71c20c505ee9c721a8bc3e      Milk Options          No           0.0                1
192766  5f71c1e04f5ee93a5f72a46f  5f71c20c505ee9c721a8bc3e      Milk Options          No           0.0                1
192767  5f71c1e04f5ee93a5f72a46f  5f71c20c505ee9c721a8bc3e      Milk Options          No           0.0                1
192768  5f71c1e04f5ee93a5f72a46f  5f71c20c505ee9c721a8bc3e      Milk Options          No           0.0                1

[2299 rows x 6 columns]
```

Missing Values:

Also, after checking missing values in these files, I found that the file **order_item_options.csv** has no missing values:

```
Is there missing values in the master_dataset?  False

ORDER_ID            0
LINEITEM_ID         0
OPTION_GROUP_NAME   0
OPTION_NAME         0
OPTION_PRICE        0
OPTION_QUANTITY     0
dtype: int64
```

The file **date_dim.csv** has some reasonable missing values for the column **holiday_name**:

```
Is there missing values in the master_dataset?  True

date_key          0
year              0
month             0
week              0
day_of_week       0
is_weekend        0
is_holiday        0
holiday_name    353
dtype: int64
```

But there are some reasonable and three major missing value for columsn of **LINEITEM_ID**, **ITEM_CATEGORY**, **ITEM_NAME** in the file **order_items.csv**:

```
Is there missing values in the master_dataset?  True

APP_NAME                  0
RESTAURANT_ID             0
CREATION_TIME_UTC         0
ORDER_ID                  0
USER_ID               17808
PRINTED_CARD_NUMBER  157435
IS_LOYALTY                0
CURRENCY                  0
LINEITEM_ID               1
ITEM_CATEGORY             1
ITEM_NAME                 1
ITEM_PRICE                0
ITEM_QUANTITY             0
dtype: int64
```

After exploratory analysis, I found that all three NaN values columns related to only one record:

```
missing_rows = df[df[['LINEITEM_ID', 'ITEM_CATEGORY', 'ITEM_NAME']].isna().any(axis=1)]
display(missing_rows)
```

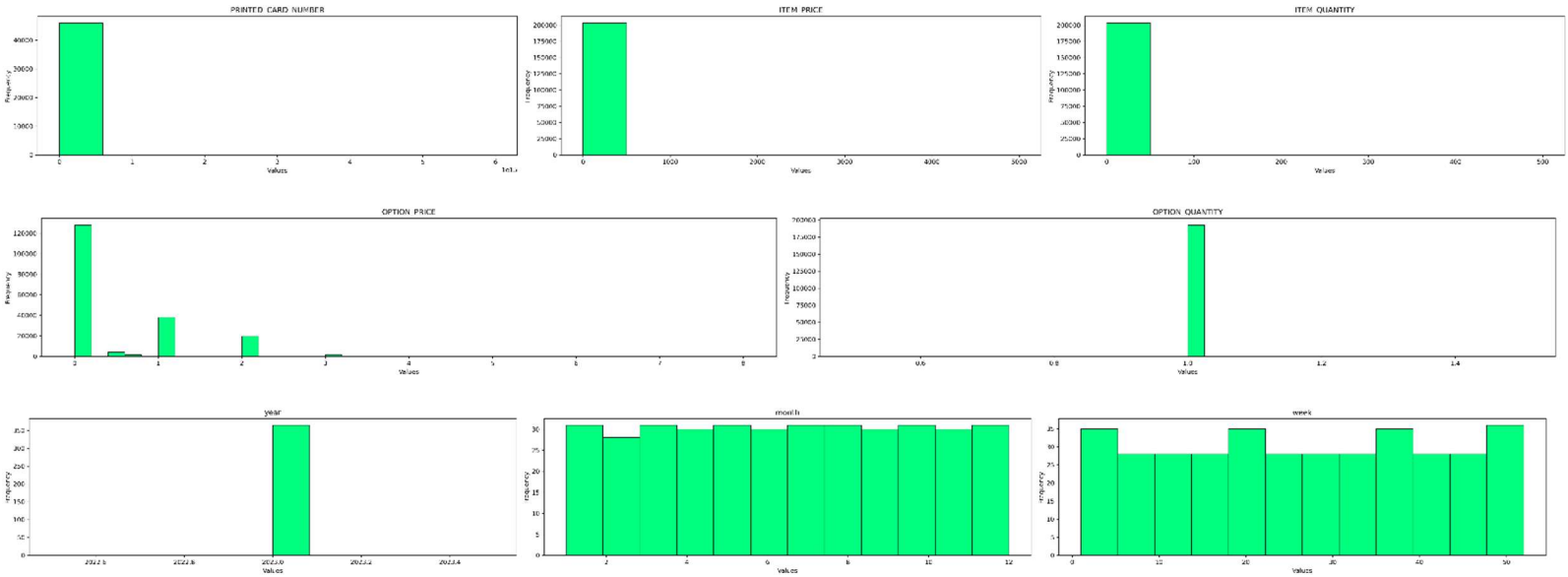
0.0s Python

	APP_NAME	RESTAURANT_ID	CREATION_TIME_UTC	ORDER_ID	USER_ID	PRINTED_CARD_NUMBER	IS_LOYALTY	CURRENCY	LINEITEM_ID	ITEM_CATEGORY	ITEM_NAME	ITEM_PRICE	ITEM_Q
16529	Alltown Fresh	5e7e35ed902ad5ac017b242b	2021-12-07T16:23:30.917Z	61af8a82ffd125453e62ae37	609d681462e498b356e72a6d	NaN	False	USD	NaN	NaN	NaN	4.39	

As I want to use **LINEITEM_ID** column as primary key for this file, then, I removed this record from the data.

Histogram Analysis:

Here is the histogram for only numerical columns:



Almost good and reasonable histograms!

Primary Key:

Then, I decided to consider the primary key for all three files, which will be unique for them.

I considered **LINEITEM_ID** for **order_items.csv**:

```
unique_key_dups = df['LINEITEM_ID'].duplicated().any()
unique_key_dups
```

✓ 0.0s

```
np.False_
```

a combination of 3 columns of **['LINEITEM_ID', 'OPTION_GROUP_NAME', 'OPTION_NAME']** for **order_item_options.csv**:

```
print(df_clean[df_clean.duplicated(subset=['LINEITEM_ID', 'OPTION_GROUP_NAME', 'OPTION_NAME'])])
```

✓ 0.0s

```
Empty DataFrame
Columns: [ORDER_ID, LINEITEM_ID, OPTION_GROUP_NAME, OPTION_NAME, OPTION_PRICE, OPTION_QUANTITY]
Index: []
```

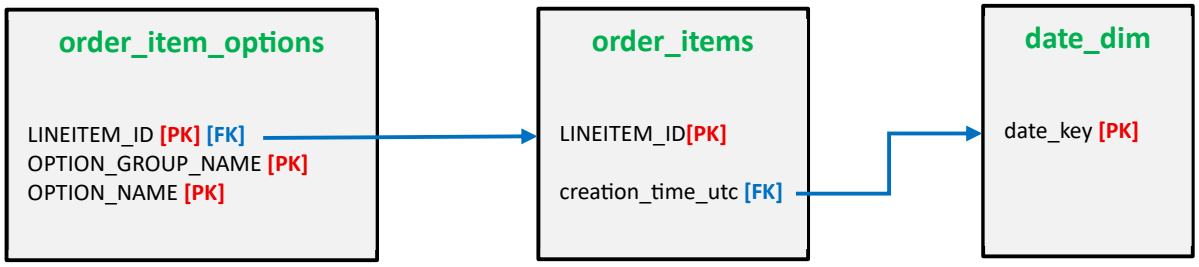
```
unique_key_dups = df_clean[['LINEITEM_ID', 'OPTION_GROUP_NAME', 'OPTION_NAME']].duplicated().any()
unique_key_dups
```

✓ 0.0s

```
np.False_
```

and **'date_key'** for **date_dim.csv**:

```
unique_key_dups = df[['date_key']].duplicated().any()
unique_key_dups
✓ 0.0s
np.False_
```



Statistical Analysis:

I did some statistical analysis for these datasets. Here is some information

for **order_items.csv**:

```
There are 203518      LineItem_ID in the dataset
There are 131328      Order_ID in the dataset
There are 20174       users in the dataset
There are 431         item_names in the dataset
There are 45          item_categories in the dataset
There are 28          restaurants in the dataset
There are 3           applications in the dataset
There are 1           Currency in the dataset
```

for **order_item_options.csv**:

```
There are 102712      unique LineItem_ID in the dataset
There are 78614       Order_ID in the dataset
There are 637         option_names in the dataset
There are 133         option_group_names in the dataset
```

and for **date_dim.csv**:

```
There are 365   unique date_key in the dataset
There are 1     year in the dataset
There are 12    month in the dataset
There are 52    week in the dataset
There are 7     day_of_week in the dataset
There are 12    holiday_name in the dataset
```

Proposal:

After completing exploratory data analysis on the three source datasets, the following proposal outlines the design and implementation of a production-ready data pipeline to support business insights, customer analytics, and reporting requirements.

Step 1 – Data Ingestion (S3 Bronze Layer)

Objective

The objective of this step is to ingest raw transactional and dimensional data from a SQL Server database into an Amazon S3–based data lake, forming the Bronze layer. This layer will store raw, immutable data to support auditability, reprocessing, and downstream transformations.

Design Choice

To comply with the project requirement that all transformation logic be implemented using Spark, AWS Glue ETL jobs with JDBC connectivity will be used for data ingestion instead of AWS Lambda. This approach ensures that the ingestion process is both scalable and aligned with Spark-based processing standards. AWS Glue provides a distributed Spark environment that is well suited for:

- Reading relational data using JDBC
- Handling incremental and batch ingestion
- Scaling efficiently for future data growth
- Maintaining consistency across ingestion and transformation layers

Security and Connectivity

The following AWS services will be used to enable secure and reliable ingestion:

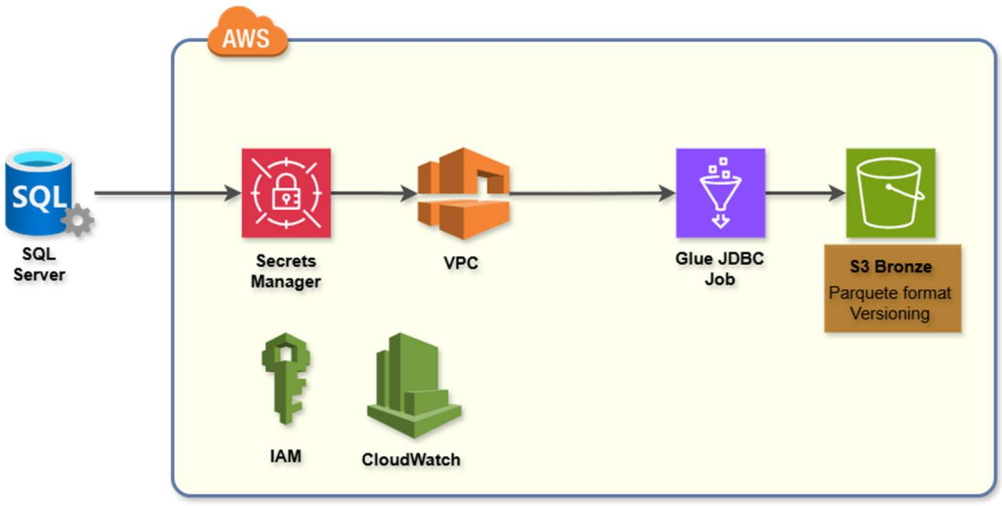
- **AWS Secrets Manager:** Stores SQL Server credentials securely and prevents hardcoding sensitive information in Glue job scripts.
- **Amazon VPC:** Provides secure network connectivity between AWS Glue and the SQL Server database, ensuring that data transfer occurs within a controlled private network.
- **AWS IAM:** Enforces least-privilege access policies, allowing Glue jobs to read secrets and write data to Amazon S3 securely.

Target Storage (Bronze Layer)

Raw data will be written separately to the following S3 bucket: **s3://project3-bronze-bucket/**

Implementation Approach

- AWS Glue ETL jobs will connect to SQL Server using JDBC.
- Source tables (**order_items**, **order_item_options**, and **date_dim**) will be extracted in batch mode.
- Data will be written to Amazon S3 in **Parquet** format to optimize storage efficiency and downstream processing.
- No transformations will be applied at this stage to preserve data fidelity.
- Enable S3 bucket versioning to retain all versions of the raw CSV data.



Step 2 – Transformation (S3 Silver Layer)

Objective

The objective of the Silver layer is to transform raw ingested data into **clean**, **standardized**, and **analytically usable datasets**. This step focuses on improving data quality by resolving **missing values**, **removing duplicates**, **enforcing schemas**, and **preparing fact and dimension tables** for downstream analytics and metric computation.

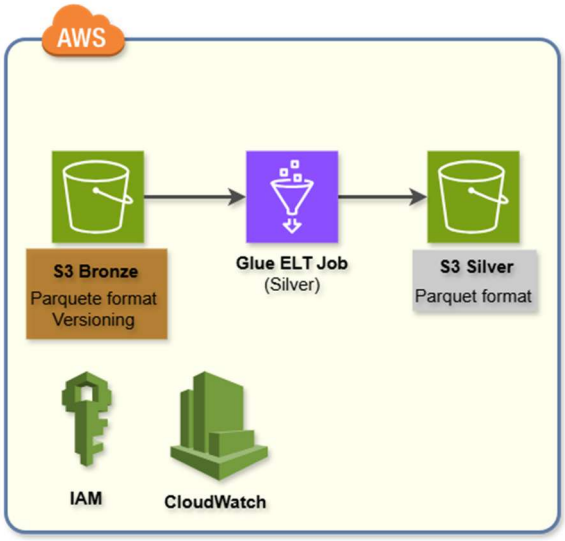
Design and Storage

Data in this layer will be structured into fact and dimension–oriented datasets, enabling efficient joins and aggregations in subsequent processing stages. Transformed data will be stored in a dedicated S3 bucket: **s3://project3-silver-bucket/**

Implementation Approach

An AWS Glue PySpark ETL job, named **project3-silver-glue**, will be executed to perform the following transformations:

- **Schema Enforcement**
  - Explicitly define schemas for fact and dimension tables to prevent schema drift.
  - Ensure consistent column ordering and naming conventions.
- **Data Type Standardization**
  - Cast columns to appropriate data types (e.g., timestamps, numeric fields, boolean flags).
  - Ensure consistency across datasets for join keys and metrics.
- **Duplicate Handling**
  - Identify and remove duplicate records based on defined primary or composite keys.
  - Preserve only the most recent or valid records where applicable.
- **Missing Value Management**
  - Remove records with critical missing identifiers (e.g., primary keys).
  - Retain non-critical nulls where they are analytically meaningful (e.g., holiday names).
- **Column Normalization**
  - Standardize column names (e.g., lowercase, snake_case) for consistency and usability.
  - Align naming conventions across all datasets.
- **Optimized Storage Format**
  - Persist transformed datasets in **Parquet format** to improve query performance and reduce storage costs.



Step 3 – Analytics & Aggregation (S3 Gold Layer)

Objective

In this step, the objective is to produce analytics-ready, aggregated datasets by joining and summarizing the dimensional and fact tables created in the Silver layer. The Gold layer serves as the single source of truth for business metrics and is optimized for direct consumption by visualization and reporting tools.

Design and Storage

Data in this layer will be structured into different fact and dimension datasets. Transformed data will be stored in a dedicated S3 bucket: **s3://project3-gold-bucket/**

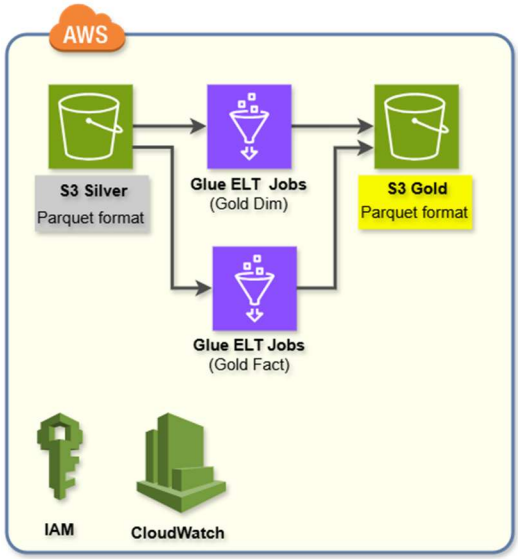
Implementation Approach

Two separate AWS Glue PySpark ETL jobs, named **project3-gold-dim-glue** (for dimension-level aggregations) and **project3-gold-fact-glue** (for fact-level and KPI calculations), will be generate curated datasets and metrics aligned with the business analytics questions, including but not limited to:

- Customer Lifetime Value (CLV) Evolution (Daily)
- RFM (Recency, Frequency, Monetary) Snapshot
- Sales Trends over Time
- Location-Level Performance Metrics
- Customer Churn Indicators
- Other relevant KPI metrics

Notes:

- The Gold layer is explicitly optimized for **Streamlit dashboards**, enabling fast data access, simplified querying, and seamless visualization without additional transformation logic.
- The transformed datasets will be persisted in **Parquet format** to improve query performance and reduce storage costs.



Step 4 – Visualization (Streamlit)

Objective

In the final step, Streamlit dashboards will query data directly from the Gold S3 bucket.

Implementation Approach

- Using pandas for data handling (sufficient for moderate-sized parquet files).
- Build interactive dashboards for stakeholders with filters, charts, and KPI cards.

Note: No data warehouse or query engine is required; this approach is fully serverless and cost-efficient.

Jobs Orchestration and CI/CD

Objective

The objective of this component is to automate the end-to-end batch data pipeline on a daily schedule while ensuring reliable orchestration, clear job dependencies, and maintainable CI/CD practices. All data is ingested from SQL Server into AWS, transformed through Bronze, Silver, and Gold layers, and made available for analytics and visualization without manual intervention. To support continuous development and deployment, all AWS Glue Spark scripts are version-controlled in a GitHub repository and automatically deployed to AWS when changes occur.

1. Scheduling and Orchestration

- A daily **EventBridge** schedule triggers a **Lambda** function.
- The Lambda function starts the **Glue Workflow**, which acts as the central orchestration engine for the pipeline.
- The Glue Workflow executes jobs in a dependency-based sequence:

Glue JDBC Job (Bronze ingestion) → Glue Silver ETL Job → Gold Glue Jobs (parallel execution)

Gold-layer Glue jobs (dimension and fact aggregations) are executed **in parallel** to reduce total processing time.

2. CI/CD Strategy

This approach enables automated deployment, traceability, and reproducibility of data pipelines.

- All PySpark scripts for Glue jobs are stored in a **GitHub** repository**:** **https://github.com/Hadi2468/ELT_Project3**.
- A CI/CD pipeline (e.g., AWS CodePipeline) automatically:
  - Detects changes or additions to Glue job scripts
  - Deploys updated scripts to an S3 code bucket: **s3://project3-code-bucket/**
- All Glue jobs reference their scripts directly from the S3 code bucket, ensuring:
  - Version consistency
  - Easy rollback
  - No manual code uploads

3. Data Ingestion (Bronze Layer)

- The **Glue JDBC job** is triggered by the **Glue Workflow**.
- It connects securely to SQL Server and ingests raw CSV data.
- The ingested data is written to the **S3 Bronze bucket** without transformation.

4. Silver Layer Transformation

- Upon successful completion of the ingestion step, the Glue Workflow triggers the **Silver Glue ETL job**.
- This job:
  - Cleans and standardizes the raw data
  - Resolves missing values and duplicates
  - Produces normalized dimension and fact tables
- Outputs are stored in the **S3 Silver layer** in Parquet format.

5. Gold Layer Aggregation

- After the Silver job completes successfully, the Glue Workflow triggers **multiple Gold Glue jobs** in **parallel**.
- These jobs:
  - Join dimension and fact tables
  - Compute KPIs and analytical aggregates
  - Produce dashboard-ready datasets
- Final outputs are stored in the **S3 Gold layer**.

6. Monitoring and Failure Handling

- Each Glue job writes logs to **Amazon CloudWatch**.
- In case of any job failure:
  - The Glue Workflow stops downstream execution
  - Error details are available in CloudWatch Logs for troubleshooting

7. Streamlit Visualization

- Streamlit dashboards read Gold-layer Parquet files directly from S3.
- Dashboards always reflect the latest successful pipeline execution.
- No additional transformation logic is required at the visualization layer.

Benefits of This Orchestration Strategy

This design ensures:

- Clear and deterministic job dependencies
- Controlled execution order
- Parallel processing where appropriate

- Failure isolation and observability
- Minimal operational complexity
- Scalable and production-ready CI/CD practices

Conclusion

This project delivers a **cost-efficient, fully serverless batch data pipeline** built on AWS managed services and designed using a **Bronze–Silver–Gold lakehouse architecture**. The solution ensures a clear separation of concerns across data lifecycle stages, enabling **reliability**, **scalability**, **security**, and **maintainability**.

Raw data is securely ingested from **SQL Server** and preserved in the **Bronze layer** to guarantee data lineage, auditability, and traceability. The **Silver layer** applies standardized transformations, data quality rules, and schema enforcement to produce clean, reliable dimension and fact tables. The **Gold layer** stores analytics-ready, pre-aggregated Parquet datasets optimized for high-performance querying and downstream consumption.

To ensure **data security at rest**, all Amazon S3 buckets (Bronze, Silver, and Gold) are protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**. This approach enables centralized key management and fine-grained access control through **IAM policies**.

The entire pipeline is **automated, event-driven, and dependency-aware**, **orchestrated** through **AWS Glue Workflow** and integrated with a **CI/CD process** that enables **version-controlled development** and seamless deployment. **Streamlit dashboards** consume the Gold-layer datasets directly from Amazon S3, ensuring fast access to trusted business metrics with minimal operational overhead.

Overall, this architecture provides a **scalable, maintainable, and production-ready foundation** for data analytics and business intelligence while minimizing infrastructure management and operational costs.