Project Documentation: Building an ETL Pipeline Using Azure for Olist E-commerce Data

Project Overview

In this project, I developed an end-to-end ETL pipeline using Azure to process, clean, and analyze the Olist E-commerce dataset from Brazil. The goal was to extract the raw data, apply necessary transformations, and load it into an Azure SQL Database for further analysis and reporting.

The dataset covers key e-commerce aspects like orders, customers, products, reviews, sellers, and payments, providing a comprehensive view of the business. My approach was to ensure scalability, efficiency, and accuracy by leveraging cloud-based tools and services.

Objectives

The primary goal was to create a fully functional ETL pipeline that handles data cleaning, transformation, and validation. Additionally, I implemented advanced analysis techniques like calculated columns and rolling window functions. Finally, the data was loaded into a cloud database (Azure SQL Database) where it could be queried and used for reporting.

Step 1: Setting Up Azure Data Lake for Storage

The first task was to create an Azure Data Lake Storage Gen2 account to store the raw CSV files. Data Lake Storage is perfect for handling large-scale data and provides a secure environment to store raw files for processing.

- **Reason**: I chose Azure Data Lake Storage for its scalability and ability to handle large datasets, making it an ideal starting point for the ETL pipeline.

Step 2: Configuring Databricks for Data Processing

Next, I created a Databricks workspace to develop notebooks and run Python scripts. I also created a cluster to handle the computational needs of the

project.

- **Databricks Workspace**: I named the workspace **test_databricks_workspace**.
- **Cluster Creation**: The cluster **testcluster** was created to handle all computational operations like reading the files and running transformations.

Once the workspace and cluster were set up, I linked Databricks to Azure Data Lake Storage by creating an application **testapplication**, using OAuth authentication to securely connect and access the data.

- **Reason**: Databricks was chosen for its seamless integration with Azure and its ability to scale compute resources, allowing for efficient processing of large datasets.

Step 3: Data Cleaning and Transformation

Once the raw data was loaded into Databricks, the first task was to inspect and clean the data. I inspected the datasets to identify missing values, duplicate records, and any formatting inconsistencies. For example:

- **Missing Values**: I filled missing numeric values using the median where appropriate and cleaned up missing textual values by standardizing formats (e.g., city names).
- **Duplicates**: I removed duplicate records from transaction-based datasets like `order_items` to ensure data accuracy.

After cleaning, I created new calculated columns such as **Total Price** (sum of product price and freight value) and **Delivery Time** (difference between delivery and purchase dates).

- **Reason**: Cleaning the data ensures data quality, which is crucial for any downstream analysis. Creating calculated columns added value to the datasets by making them more insightful for analysis.

Step 4: Merging Datasets and Applying Window Functions

After cleaning, I merged several datasets to create a unified view. This allowed me to analyze the relationships between different datasets and gain better insights.

Next, I applied window functions:

1. **Total Sales per Customer**: I calculated a running total of sales for each customer.
2. **Average Delivery Time per Product Category**: I calculated a rolling average to track delivery performance.

- **Reason**: Merging datasets creates a comprehensive view of the data, enabling relational analysis across orders, customers, and products. Using window functions allows for cumulative and rolling metrics that are important for time-based and customer-specific insights.

Step 5: Saving Data to Azure SQL Database

Once all the cleaning and transformations were complete, I split the data into fact and dimension tables, which were then saved to an Azure SQL Database. The tables were structured as follows:

- **Fact Table**: Order Items with calculated columns like Total Price, Delivery Time, etc.
- **Dimension Tables**:
  - **Customers**: Details of customers and their geographic information.
  - **Products**: Product-related data.
  - **Sellers**: Seller information and geographic data.
  - **Dates**: Purchase and delivery dates as time-related dimensions.
- **Reason**: Azure SQL Database was chosen as the final destination because it provides a scalable, cloud-based environment for storing and querying data. By splitting the data into fact and dimension tables, I ensured efficient querying and reporting, which is a best practice in data warehousing.

Step 6: Implementing Slowly Changing Dimensions (SCD)

To handle changes in data over time (e.g., customer address updates), I implemented Slowly Changing Dimensions (SCD Type 2). This approach stores historical versions of records, allowing for time-based analysis.

Step 7: Data Validation and Reporting

To ensure data accuracy, I ran validation scripts to check that the data loaded into Azure SQL Database matched the processed data. I also verified that key metrics such as Total Sales and Delivery Time were calculated correctly.

Historization and Change Data Capture (CDC)

As part of the project, I also added columns to track changes in the datasets over time. The `Date` dimension table was used to store the historical dates, and additional columns were added to capture when records were inserted, updated, or changed.

Future Enhancements:

In addition to the ETL process, I plan to implement SQL procedures to manage data changes over time effectively:

- **SCD Type 2**: To track changes in key attributes (like customer address), ensuring that historical data is preserved alongside the current version.
- **Change Data Capture (CDC)**: To capture real-time inserts, updates, and deletes in the source data, ensuring that modifications are reflected accurately in the database.
- **Historization**: To record the history of data changes by adding metadata such as creation date, update date, and record version.

These procedures will be designed to automate the management of data changes and ensure historical accuracy for future auditing and analysis.

Project Conclusion

This project highlights my ability to build a robust ETL pipeline on Azure, process large datasets, and store the final cleaned data in a SQL database. The steps involved, from data storage in Azure Data Lake to transformations in Databricks, followed by loading into Azure SQL Database, demonstrate my competency in cloud-based data engineering.

Next Steps: For future improvements, I could extend the pipeline to include more advanced analytics or integrate real-time data updates using Azure Data Factory and Change Data Capture (CDC).

Deliverables: The project, including Jupyter notebooks, SQL scripts, and documentation, is available on GitHub Repository.