# Data Engineering Assessment - Matheus Alves

Python 3.9+ | Pydantic Data Validation | Jupyter Notebook | Alembic Database Migration | MSSQL Server | SQLAlchemy ORM

## The Challenge

1. **By making use of the `invoices.xls` data file as your source, create a Data Warehouse model with best practices in place, using the Kimball methodology.**
2. **Populate the model created in step 1 with the data provided in the Excel sheet.**
3. **Analyze any abnormalities (if any) in the data provided and take any action needed (where possible).**
4. **List any assumptions taken into consideration.**
5. **Provide 3 different aggregations one might use for reporting purposes.**

betsson group

## Table of Contents

---

## 0. Base Constraints

- This task may be done either with MSSQL, Python or C#.
- It is highly recommended that comments are included, and code is structured well.
- If making use of python, you can make use of Jupyter notebooks, thus you can either
- provide a .PY file or .IPYNB file.
- Assumptions & Abnormalities are to be provided in a separate document.

## 0.1 Scope Matrix - Documentation and Code

### Base General Process

| **Feature** | **In Scope (MVP)** | **Out of Scope** |
|---|---|---|
| README.md-based documentation (iterative view) | ✅ | |
| PDF documentation (static) | ✅ | |
| Module and package details | ✅ | |
| Base real case assumptions | ✅ | |
| Packages and modules partially/fully reusable | ✅ | |
| Base data quality check | ✅ | |
| Cloud-related environments | | ✅ |
| Full production-ready | | ✅ |
| Non-Python frameworks for ETL | | ✅ |
| Specific metrics logging (to be used in Prometheus, for instance) | | ✅ |
| Deeply statistic approaches in warehouse/data cleaning | | ✅ |

| Feature | In Scope (MVP) | Out of Scope |
| --- | --- | --- |
| Specific Architecture Consumption/Tunning Scenario | | ☑ |

Base ETL Considerations

| Feature | In Scope (MVP) | Out of Scope |
| --- | --- | --- |
| Process diagrams | ☑ | |
| Isolated environment (.venv, optional) | ☑ | |
| `pip` requirements (`pip install -r requirements_xx.txt`) | ☑ | |
| APIs | | ☑ |
| Scheduler-ready | | ☑ |

Base Warehouse Considerations

| Feature | In Scope (MVP) | Out of Scope |
| --- | --- | --- |
| Data full metadata | ☑ | |
| Client basic tuning example (MSSQL Query Plan) | ☑ | |
| Decision explained | ☑ | |
| Physical computing/database architecture and setup | | ☑ |
| Server basic tuning (on possible queries) | | ☑ |

# 1. Assumptions & Abnormalities

## 1.1 Base Considerations

There are two main logic on this assesment answers.

- **draw.ipynb (Jupyter Notebook):** Containing draws.
- **solution.py (Script):** Pipeline solution. Consideration:
- Assumptions usually are discussed with the areas/teams involved. In a real-world scenario, I would discuss these assumptions with the business team to ensure that they are aligned with the business rules and requirements.
    - **This consideration applies to every main assumption I make.**
- This code is made on python + mssql.
- New data overwrite old data.

---

#### 1.2 Overall Data Assumptions

- I'm assuming that the data is a snapshot and we don't need to update it in the database.
- Data snapshot only; no updates required.
    - Even with that in mind, we have dimensions, for instance, that take into consideration Slowly Changing Dimensions (SCD) Type 2, to keep track of the changes in the data.
- Lowercase will be used for column names, in addition to snake case.
    - This is a common practice to avoid any possible issues with the database.
- Adjust data.
    - Maintaned and flagged, proper differenciated from sales.
        - adjustments can address wastes in Lean Methodologies as well potential losses by possible mistakes, and can come to be explored.
- I am assuming that every valid product adjustment at any category is being properly identificate in the description column.
- I am assuming that every known retail is registered, and related transactions share the same ID across different lines.
- I am assuming that I am dealing with hybrid store sales. That sails are made both online and offline.
- A simple solution to mapped values is to use a dictionary to standardize the values.
    - More complex cases involving large datasets, approaches can range from using a Cartesian mapping stored in a .txt file to developing a classification model to identify and standardize variations in location names automatically.
- No special cases in invoice date column.
- Warehouse inserted data was also approached with null scenarios. They were ignored, as they was threated as special cases. I'm assume I've maitaned referencial integrity.

#### 1.3 Overall Data Abnormalities

- Columns contain missing values that can represent various scenarios, ranging from errors/bugs to information that was not shared.
    - I am categorizing these values appropriately.
    - Unspecified: Explicitly chose not to declare their information.
    - Null (e.g., None, NaN): Did not provide the information at all.
- I am assuming this governance rule (when I have not specified a different approach):
    - Placeholder values will be standardized as "Unspecified."
    - Null values will remain as null.
- There were test values that have been addressed and removed from all columns.
    - I am assuming that there are no special cases, such as identifying errors in the client flow that need to be registered.

- To effectively address description and category problems, I would evaluate the possibility of creating a governance policy to map such cases according to the business rules.
    - I'm assuming that this scenario involves areas where I am unable to act.

#### 1.3.1 Country column

- Some entries in the Country column might not represent valid country names (e.g., typos, placeholders, or regions like "Channel Islands").
    - The simplest approach would rename Country to Locations.
- Some entries in the Country column are abbreviated, such as "United Kingdom" and "UK".

#### 1.3.2 Customer ID column

- Test customers identified, when casually understanding data format.
    - There is 'test' data on different columns.
- Null clients often relates (based on description column) to manual inserts, fee, a/b testing. I'll deal with them in the following lines.
- Sales that doesn't have a description, as well doesn't have a customer ID, will be flagged as special cases.
- I'm assuming that customer id and description missing, being < 0, are error entries.

#### 1.3.3 Price column

- Prices equals to zero or negatives.
    - There were unusual descriptions such as "mixed" and "short". They are ignored, as they are relevant sales.
    - I will map returns and consider them valid only when Quantity_return is less than Quantity_sale (based on merging invoice and StockCode data, validating negative values against all related entries).
- Values involving different scenarios.
    - Damages, damaged, bad quality.
    - Damages or negative descriptions appoitnments at any kind will be flagged as lost sales.
        - Descriptions that are not clear or relevant, such as 'lost in space,' will be flagged as lost sales. I will assume these represent some kind of product damage or loss.
        - I am also assuming that there are no valid descriptions that share the same part of their definition. For instance, terms like "wet" in descriptions such as 'WET PLANT PERFECTLY DECORED' are not being used ambiguously.
        - I'm usinng a list of known descriptions patterns to flag lost sales.

#### 1.3.4 Quantity column

- Negative and zero values that isn't some returned data.
    - Some negative data refers to discounts. They are flagged as discount.
- There where match in stockcode agains't null values. For rows without a Description or Price, I am assuming the last recorded Description and Price based on the StockCode. This assumes that StockCode descriptions are static, and the last recorded value is correct and applicable

#### 1.3.5 Description column

- There are ids on description, that seems to be a stockcode, but there aren't any indicator. I'm assuming that they are
- AMAZON FEE, adjustments in bad debt, etc, they will be flagged as financial detail.
- Adjustments are flagged as maintenance data.
  - Only adjustments proper described will be flagged as maintenance data.
- Update are flagged as maintenance data.
- Bargains are discounts, but they are threat directly as financial detail.

##### 1.3.6 StockCode column

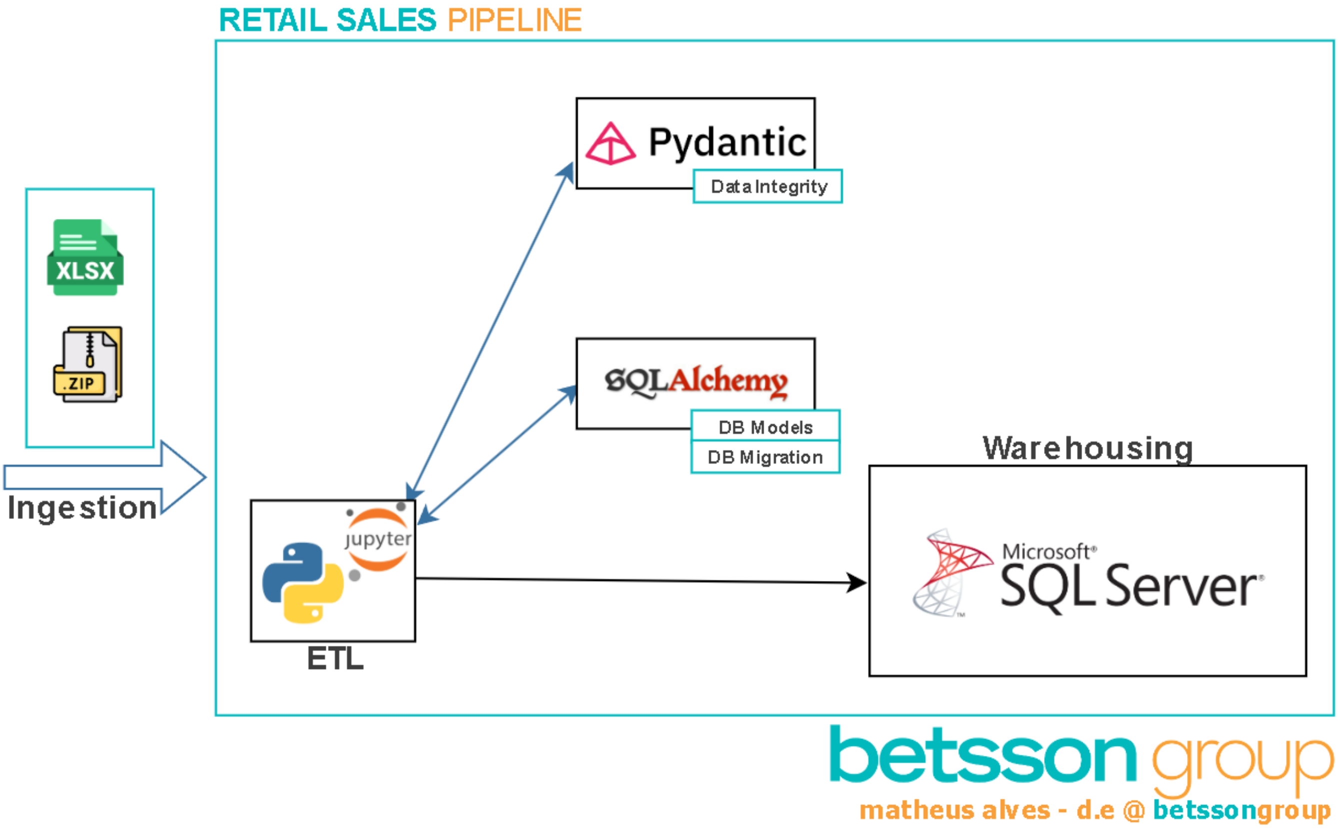
- There is patterns in stockcode like "M", "D". Coming exclusively, they are representating discount. Manual sales are sales as the others, they are ignored.
- Discounts are flagged as discount.
- Stock code S, representing samples, typically has values of -1. However, there are some cases where this is not the case. These will be flagged to be removed.
- Gift are flagged as lost sales
- Test are flagged to be removed.
- Stockode flagged on DW when. without return flag or quantity < 0.
- Bankcharges are flagged as financial detail.

##### 1.3.7 General columns data

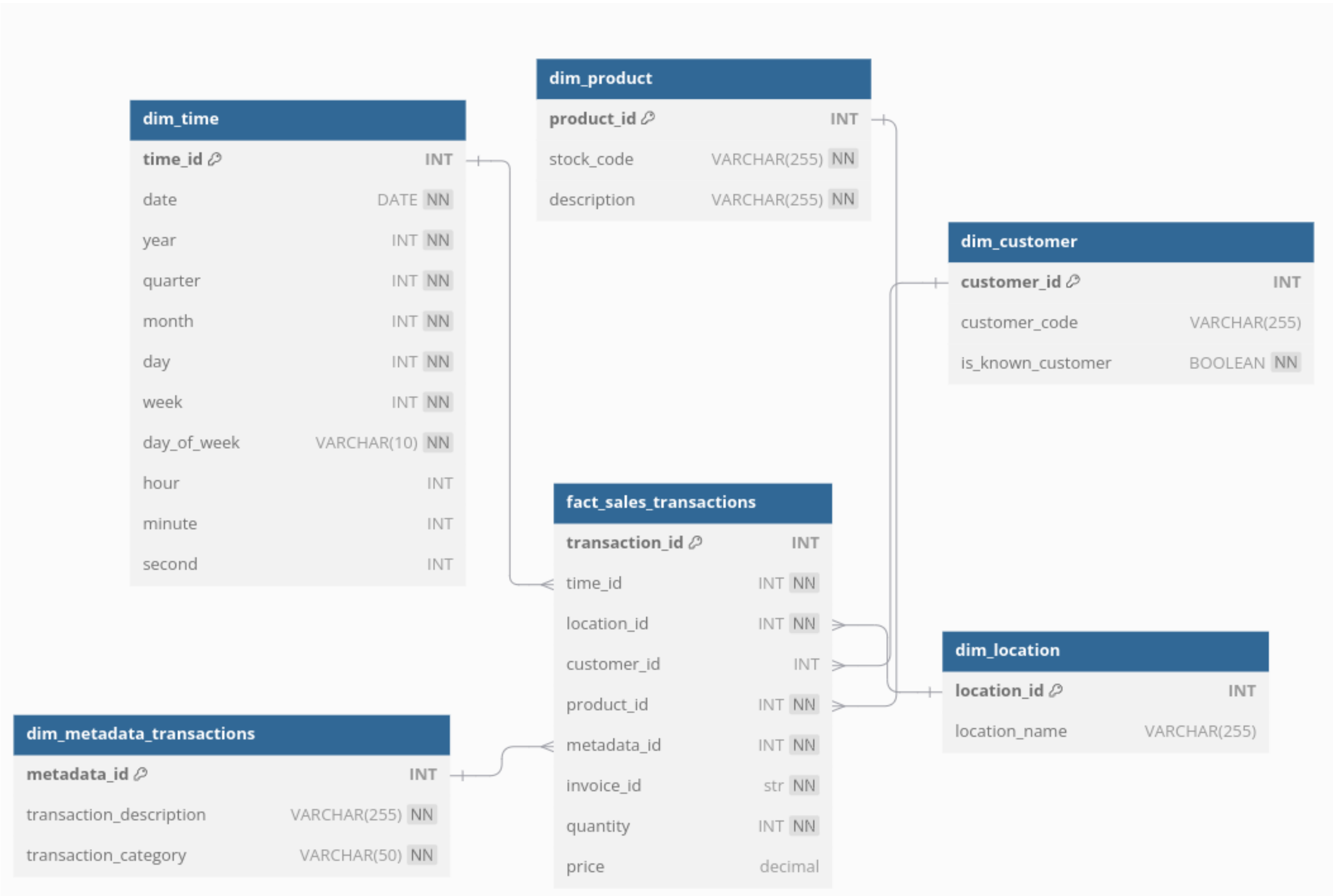
- Guarantee that overall mapped appointment is being properly flagged.
- After filtering all flagged columns, any remaining abnormal data is reviewed.
- I've tried to enrich data, but there were no match:
  - Price: Updated using the most recent value for the same stock_code.
  - Description: Updated using the modal (most frequent) value for the same stock_code.
  - Assuming that if resulting value is null and there is no client id, so it is entry error.

---

# 2. Pipeline Architecture Diagram




## 2.1 Data Warehouse Schema Diagram




## 2.2 Documentation

- Anonimized data.

Tables and Columns

#### `dim_time`

| Column | Description |
| --- | --- |
| `time_id` | Primary key for the time dimension. |
| `date` | The full date in `YYYY-MM-DD` format. |
| `year` | The year of the transaction. |
| `quarter` | The quarter of the year (1-4). |
| `month` | The month of the year (1-12). |
| `day` | The day of the month (1-31). |
| `week` | The week of the year (1-53). |
| `day_of_week` | The name of the day (e.g., "Monday"). |
| `hour` | The hour of the transaction (0-23, optional). |
| `minute` | The minute of the transaction (0-59, optional). |
| `second` | The second of the transaction (0-59, optional). |

---

#### `dim_location`

| Column | Description |
| --- | --- |
| `location_id` | Primary key for the location dimension. |
| `location_name` | The name of the location where the transaction occurred. |

---

#### `dim_customer`

| Column | Description |
| --- | --- |
| `customer_id` | Primary key for the customer dimension. |
| `customer_code` | Unique code identifying the customer, if available. |
| `is_known_customer` | Indicates whether the customer is known (`True`) or anonymous (`False`). |

---

#### `dim_product`

| Column | Description |
| --- | --- |
| `product_id` | Primary key for the product dimension. |
| `stock_code` | Unique code identifying the product stock. |
| `description` | Description of the product. |

---

#### `dim_metadata_transactions`

| Column | Description |
| --- | --- |
| `metadata_id` | Primary key for the metadata transactions table. |
| `transaction_description` | Description of the transaction type or reason. |
| `transaction_category` | Category of the transaction (e.g., "sale", "return", "adjustment", "fee"). |

---

#### `fact_sales_transactions`

| Column | Description |
| --- | --- |
| `transaction_id` | Primary key for the fact table. |
| `time_id` | Foreign key referencing `dim_time`. |
| `location_id` | Foreign key referencing `dim_location`. |
| `customer_id` | Foreign key referencing `dim_customer`. |
| `product_id` | Foreign key referencing `dim_product`. |
| `metadata_id` | Foreign key referencing `dim_metadata_transactions`. |
| `invoice_id` | Unique identifier for the invoice. |
| `quantity` | Number of units involved in the transaction (can be negative for returns). |
| `price` | Price per unit of the product (nullable; may include refunds or adjustments). |

---

## 3. Packages and modules overview

### 3.0. Running the project

- **Requirements**: Python 3.9+ and Jupyter Notebook.
- **Use a virtual environment**: `python -m venv .venv`
- **Installation**: Run `pip install -r requirements.txt`. The requirements file includes two sets of dependencies: one specifically for the Jupyter Notebook (.ipynb) and another for the Python-based script. This separation ensures that the notebook's additional dependencies are only installed if you plan to use it, keeping the main script lightweight and efficient.
- **Create and fill a file called '.env' in the root directory of this project. Fill it with the following environment variables**:

```
MSSQL_WAREHOUSE_URL="mssql+pyodbc://<username>:<password>@<host>/<database>?driver=ODBC+Driver+17+for+SQL+Server&trusted_connection=yes"
```

- **Run the main script**: `python solution.py` on your terminal, from the root project diretory.

### 3.1. Main Script

- `draw.ipynb`: Development notebook. Contains all the exploratory data analysis (EDA) and data cleaning steps. It also includes the initial data quality check and the first assumptions and abnormalities identified.
- `solution.py`: Example of a dedicated pipeline, including all modules. Possible Lineage tracker, with space to implement a run back from stopped stage.
- `.env`: Requirement to DW experience.

### 3.2. infra

- models
    - Dimensional and fact models.
        - `dim.py` - all dimensional to our DW models.
        - `fact.py` - all fact to our DW models.
        - `facts_integrity.py` - all base validators to our fact in the DW.
        - `dims_integrity.py` - all base validators to our dims in the DW.
- pipeline
    - Pipeline specif codes
        - `pipeline_metadata.py` - References to metadata process. Like Mapping, etc.
            - NORMATIZE_LOCATION_MAP - a dict containing the mapping of normalized location names.
            - CLOUD_LOST_PRODUCTS_WORDS - a list of words that indicate lost products.
            - STAGE_III_COLUMNS - renamed columns to be used in the pipeline.
            - validation_models - mapper containing Pydantic models to validate the data.
            - models_map - mapper containing sqlalchemy models to validate the data.
        - `pipeline_lineage.py` - It stores stages related to the pipeline.
            - get_csv_df - Reads CSV files into pandas DataFrame format.
            - PipelineTransformer - BR Contains every stage and their transformations, as well as a saving method.
        - `pipeline_transformers.py` - Business rules (BR) and general transformations (GR) to be used on the pipeline.
            - sanitize_column_data - BR related to fill null data and format types.
            - sanitize_text - BR related to sanitize text data. It will remove special characters, and replace accented characters with their unaccented counterparts.
            - BaseTableGenerator - BR related to generate base tables.
            - DimTimeGenerator - BR related to generate the time dimension.
            - DimLocationGenerator - BR related to generate the location dimension.
            - DimCustomerGenerator - BR related to generate the customer dimension.
            - DimProductGenerator - BR related to generate the product dimension.
            - DimMetadataTransactionsGenerator - BR related to generate the metadata transactions dimension.
            - FactSalesTransactionsGenerator - BR related to generate the sales transactions fact table.
            - generate_warehouse_sales_tables - GR related to generate the warehouse tables.
            - validate_warehouse_sales_data - BR related to validate the warehouse tables.
            - validate_data_integrity - BR related to validate the data integrity.

- handlers
    - General handlers for database related and data processing.
        - `msql_handler.py` - MSSQL connection handler. It will be used to return the connection engine to be orchestrated by sqlalchemy/alembic/direct-queries.

### 3.3. ingestion

- Ingested data as 7z.
- Extracted data as csv/xls.

### 3.4. utils

- `_references.py` - base code references, like logging, etc.
    - get_current_utc_time - datetime now in UTC time.
    - create_logger - created supposed to use as unique logger for the application.
- `file_handlers.py` - utilities to write/read/process/format files.
    - extract_7z - It will extract the 7z file to a folder. This one is our "imaginary API".

### 3.5. assets

- Images and other assets used on README.md and other documentation. It doesn't make part of the codebase.

---

## 4. General Code Structure

- Load the data.
- Understand data scenario.
- Avoided generic changes to the data on the beginning.
- Created specific changes to the data per column.
    - Revisit different columns while exploring possible business rules.
- Transformate specific columns.
- Quick overview of transformations.
- Check data as a whole, after specific transformation.
- Apply generic changes on data.
- Validate transformations.
- Generates warehouse transformations.
- Validate warehouse transformations.
- Save data.

Code can include intermediate passes like "saving stages".

## Aggregations & Reporting

**I'm considering that the data is accurate enough (based on assumed prior reviews), and these represent real insights.**

### Aggregations.sql

**1. Insight - (AVTQ): Absolute Value of Transaction Quantities and Revenues across different categories.**

Magnitude of operational activity within each transaction category, without distinguishing between income and expense.

```
WITH category_totals AS (
    SELECT
        m.transaction_category,
        m.transaction_description,
        SUM(ABS(f.quantity)) AS total_sales_quantity,
        SUM(ABS(f.quantity * f.price)) AS total_sales_revenue
    FROM master.sales_warehousing.fact_sales_transactions AS f
    INNER JOIN master.sales_warehousing.dim_metadata_transactions AS m
        ON f.metadata_id = m.metadata_id
    GROUP BY
        m.transaction_category,
        m.transaction_description
),
grand_total AS (
    SELECT
        SUM(ct.total_sales_revenue) AS grand_total_sales_revenue
    FROM category_totals AS ct
)
SELECT
    ct.transaction_category,
    ct.transaction_description,
    ct.total_sales_quantity,
    ct.total_sales_revenue,
    (ct.total_sales_revenue / gt.grand_total_sales_revenue) * 100 AS
percentage_of_total_sales_revenue
FROM category_totals AS ct
CROSS JOIN grand_total AS gt
ORDER BY
    percentage_of_total_sales_revenue DESC;
```

ılization
onstrates
Absolute
me and
ntity
saction
QT)
ysis
lts. This
ario
lights
nders like
luct
rns,
erating

hts into
we could
oach
cing the
n rate by,
nstance,
, focusing
ne top
reasons.
egies
nt include
ering
omer
back,
erstanding
ons for
ns, and so
Product
ity checks,
omer
ce
ovements,
other
egies
d be
emented.

| | transaction_category | transaction_description | total_sales_quantity | total_sales_revenue | percentage_of_total_sales_reve |
|---|---|---|---|---|---|
| 1 | sale | Sale | 610605 | 1269941.22 | 65.922900 |
| 2 | return | Return | 215989 | 489517.86 | 25.410900 |
| 3 | financial adjustment | Financial Adjustment | 400 | 157169.30 | 8.158600 |
| 4 | maintenance adjustment | Maintenance Adjustment | 401 | 9628.98 | 0.499800 |
| 5 | lost sale | Lost Sale | 1056 | 146.34 | 0.007500 |

**2. Insight - (SLICR): Sales Location Impact Comparison on Revenue**

Monthly contribution of sales transactions by location, expressed as a percentage of the total absolute revenue for all sales transactions. A transaction is a valid sale when transaction_category = 'sale'.

```
WITH location_sales AS (
    SELECT
        l.location_name,
        t.year,
        t.month,
        SUM(f.quantity) AS total_sales_quantity,
        SUM(f.quantity * f.price) AS total_sales_revenue
    FROM master.sales_warehousing.fact_sales_transactions AS f
    INNER JOIN master.sales_warehousing.dim_location AS l
        ON f.location_id = l.location_id
    INNER JOIN master.sales_warehousing.dim_time AS t
        ON f.time_id = t.time_id
    INNER JOIN master.sales_warehousing.dim_metadata_transactions AS m
        ON f.metadata_id = m.metadata_id
    WHERE m.transaction_category = 'sale'
    GROUP BY
        l.location_name, t.year, t.month
```

```
),
total_revenue AS (
    SELECT
        SUM(ABS(total_sales_revenue)) AS absolute_total_revenue
    FROM location_sales
)
SELECT
    ls.location_name,
    ls.year,
    ls.month,
    ls.total_sales_quantity,
    ls.total_sales_revenue,
    ABS(ls.total_sales_revenue) AS absolute_sales_revenue,
    (ABS(ls.total_sales_revenue) / tr.absolute_total_revenue) * 100 AS
absolute_percentage_of_total
FROM location_sales AS ls
CROSS JOIN total_revenue AS tr
ORDER BY
    absolute_percentage_of_total DESC,
    ls.year DESC,
    ls.month DESC;
```

This visualization demonstrates the (SLICR): Sales Location Impact Comparison on Revenue analysis results. This scenario highlights locations with a higher percentage of sales revenue. Based on these insights, we can create targeted promotions and enhance marketing strategies in strong locations while identifying opportunities to explore untapped potential in weaker locations. A solid approach would involve analyzing the factors contributing to the performance of both strong and weak locations, followed by implementing appropriate campaigns and lead generation strategies.

| | location_name | year | month | total_sales_quantity | total_sales_revenue | absolute_sales_revenue | absolute_percentage_of_total |
|---|---|---|---|---|---|---|---|
| 1 | United Kingdom | 2010 | 10 | 60190 | 147035.27 | 147035.27 | 11.578100 |
| 2 | United Kingdom | 2010 | 11 | 71143 | 125453.09 | 125453.09 | 9.878600 |
| 3 | United Kingdom | 2010 | 9 | 50009 | 119630.21 | 119630.21 | 9.420100 |
| 4 | United Kingdom | 2010 | 6 | 46927 | 111493.93 | 111493.93 | 8.779400 |
| 5 | United Kingdom | 2010 | 3 | 51339 | 108108.82 | 108108.82 | 8.512800 |
| 6 | United Kingdom | 2009 | 12 | 46347 | 81325.14 | 81325.14 | 6.403800 |
| 7 | United Kingdom | 2010 | 4 | 33233 | 76377.71 | 76377.71 | 6.014200 |
| 8 | United Kingdom | 2010 | 7 | 32004 | 69954.32 | 69954.32 | 5.508400 |
| 9 | United Kingdom | 2010 | 8 | 38028 | 63387.85 | 63387.85 | 4.991400 |
| 10 | United Kingdom | 2010 | 1 | 26618 | 57497.48 | 57497.48 | 4.527500 |
| 11 | United Kingdom | 2010 | 5 | 48219 | 55379.50 | 55379.50 | 4.360700 |
| 12 | United Kingdom | 2010 | 12 | 19916 | 51586.79 | 51586.79 | 4.062100 |
| 13 | United Kingdom | 2010 | 2 | 26571 | 43960.46 | 43960.46 | 3.461600 |
| 14 | Norway | 2010 | 3 | 2 | 13916.34 | 13916.34 | 1.095800 |
| 15 | Ireland | 2010 | 10 | 750 | 9384.05 | 9384.05 | 0.738900 |
| 16 | Ireland | 2010 | 1 | 563 | 7420.23 | 7420.23 | 0.584200 |
| 17 | France | 2010 | 7 | 203 | 4352.45 | 4352.45 | 0.342700 |
| 18 | Japan | 2010 | 12 | 1496 | 3828.40 | 3828.40 | 0.301400 |
| 19 | Ireland | 2010 | 9 | 1107 | 3806.16 | 3806.16 | 0.299700 |

### 3. Insight - (CLV): Customer Lifetime Value

Understand most valuable customers and their revenues over their lifetime. Considered total revenue generated (lifetime_value) and engagement (months_active) for known customers, without filtering by location or transaction type. Valid sales are those with transaction_category = 'sale'. Not distinguishing if they are Legal Entities or Individuals.

```
WITH customer_lifetime_value AS (
    SELECT
        c.customer_id,
        SUM(CASE WHEN m.transaction_category = 'sale' THEN f.quantity * f.price ELSE 0 END) AS lifetime_value,
        COUNT(DISTINCT CONCAT(t.year, '-', t.month)) AS months_active,
        CASE WHEN SUM(CASE WHEN m.transaction_category = 'sale' THEN 1 ELSE 0 END) > 0 THEN 1 ELSE 0 END AS valid_sale_flag
    FROM master.sales_warehousing.fact_sales_transactions AS f
    INNER JOIN master.sales_warehousing.dim_customer AS c
        ON f.customer_id = c.customer_id
    INNER JOIN master.sales_warehousing.dim_time AS t
        ON f.time_id = t.time_id
    INNER JOIN master.sales_warehousing.dim_metadata_transactions AS m
        ON f.metadata_id = m.metadata_id
    WHERE c.is_known_customer = 1
    GROUP BY c.customer_id
)
SELECT
    customer_id,
    lifetime_value,
    months_active,
    lifetime_value / NULLIF(months_active, 0) AS average_monthly_value
FROM customer_lifetime_value
ORDER BY lifetime_value DESC;
```

This visualization demonstrates the Customer Lifetime Value (CLV) analysis results. This scenario highlights the most valuable customers based on their lifetime value and engagement. By identifying these high-value customers, we can develop targeted marketing strategies to enhance customer loyalty and retention. Strategies might include personalized offers, loyalty programs, and exclusive promotions to increase customer engagement and lifetime value. Additionally, for instance, we can associate this with the first insight to understand how customers engage with the company and

| | customer_id | lifetime_value | months_active | average_monthly_value |
|---|---|---|---|---|
| 1 | 3cdacdba0a3e4bf931fbe577ee4f94f7 | 56807.36 | 13 | 4369.796923 |
| 2 | 80789d636d68ec8ac889de80365bbd57 | 52383.60 | 13 | 4029.507692 |
| 3 | 21bfef81b08fa7988c78190cc68c241c | 19543.18 | 13 | 1503.321538 |
| 4 | 1671f1468c94ff1c45e249f5fd322ed5 | 18415.36 | 13 | 1416.566153 |
| 5 | d4c074c60c57087c95fd0a17f986210c | 17918.64 | 11 | 1628.967272 |
| 6 | 8a7c03958cbbbb5d374f4be72690ca7e | 16403.77 | 9 | 1822.641111 |
| 7 | 42ae58924107d3be8c6b7b7e3cfd4164 | 16122.29 | 13 | 1240.176153 |
| 8 | 1a62026e6b035b51682672932876a119 | 14689.84 | 4 | 3672.460000 |
| 9 | 7842858ddde53cb0a24dc8c9fea4f92b | 14550.74 | 12 | 1212.561666 |
| 10 | dfc0a2d63b0d7a1ce1cd07ffe3a3aea7 | 14550.13 | 12 | 1212.510833 |
| 11 | 01c8a64a2b3c66c05c2dbf9df27510eb | 14546.26 | 13 | 1118.943076 |
| 12 | 0783683c446cf52f9df7d90d92bf5239 | 14215.71 | 12 | 1184.642500 |
| 13 | ca87e9d08238e6d61791dc55931bb500 | 13916.34 | 1 | 13916.340000 |
| 14 | 4218470a524ef1991202bb63abee5d72 | 13285.25 | 8 | 1660.656250 |
| 15 | 52c031e023c8a03f30f57246f9c3d4f9 | 10953.50 | 1 | 10953.500000 |
| 16 | 1221132d8390ea66832cf2eabd8eb668 | 10109.40 | 5 | 2021.880000 |
| 17 | 74367f78d716836cc099eeb6497f8703 | 8950.84 | 12 | 745.903333 |
| 18 | f916b2ed383e62ec91b915de8ba77e0b | 8897.00 | 13 | 684.384615 |
| 19 | 2779d140faa92aa0ad8df4230aca4590 | 7715.04 | 13 | 593.464615 |

its offered products based on their location, helping to retain new leads and maintain existing ones, enhancing the overall customer experience.

### 4. Data-Driven Deep

*Our approaches are guided by business rules (market, client, company, etc) and the data available to us. These factors shape the reports we generate and follow, ensuring alignment with our strategic goals.*

- The first insight offers a broad overview of events within the specified date range. From this starting point, we could analyze monthly trends to uncover the seasonality of key pain points. By consistently delivering excellent services to our clients, we gain deeper insights into how to remain competitive and innovative.
- The second insight delves into location-specific impacts on revenue, providing a detailed perspective. This approach helps identify the most profitable locations to prioritize for investment, as well as underperforming areas that require strategic intervention. By continuously monitoring and taking timely action, we can prevent strong locations from weakening and improve weaker locations' performance.
- The third insight focuses on customer lifetime value (CLV), a critical metric for understanding customer engagement and loyalty. By identifying high-value customers and their engagement levels, we can leverage other insights to develop targeted strategies. I can think in a broad range of strategies that could include directly engaging with these clients, implementing new marketing approaches, or introducing new products to further engage them, while also strengthening the brand to attract new customers.

**In data we trust**

---

## Tunning Scenario

*This will not be performed, but I'm assuming that I would use these aggregations to create reports for the business team. Here would be my approach for also tuning this scenario.*

*This is a first basic approach. I'm assuming that there aren't logs yet (which could help identify most used columns, for instance).*

- Tuning approach, client side:
  - Query leveraging predicate filters.
  - Understand the Logical plan by the optimizer:
    - Identify bottlenecks in the execution plan by the optimizer.
    - How could I induce the optimizer to better optimize the query plan?
  - Understanding query optimal scenario tuning.
  - Understanding query hints.

Now, after having the execution plan and a proposed scenario, I can proceed to basic server-side tuning (assuming we are on a hybrid ecossystem).

- Indexes: They need special considerations—the trade-off between maintenance and performance. Understand base columns to be indexed

    - Understand base columns to be indexed
- Statistics: They are crucial to the optimizer. There is no logical in having indexes without updated statistics.
    - Governance to set frequent statistics updates.
- Partitioning, if necessary.
- Cost Threshold, parallelism, and other high-level server configurations.

We would need approaches like governance, to ensure optimal database performance. Consumption governance (help with querie base good practices). Among others.

---

*This sales Retail refers to the Attam Store.*
*Matheus Alves @ 2024*