BUSINESS INTELLIGENCE AND DATAWAREHOUSE

BID3000 H2025

Candidate: 7016

Candidate: 7020

03.10.2025

Index

Task 1: Data Warehouse Design and ETL Implementation

1.1 Warehouse design decisions

We chose the Olist Brazilian E-commerce dataset from Kaggle. It is a big and comprehensive dataset consisting of data from an online marketplace in Brazil. It has order information from 2016 to 2018.

OLTP and OLAP

To handle data, we use operational systems OLTP and OLAP. OLTP (online transaction processing) is used for transaction and is responsible for capturing and storing day-to-day data. But OLTP is not design for analysis and complex queries. That's why we use OLAP (online analytical processing) OLAP is designed to analyse organizational data more effectively and efficiently. These two operations rely on each other. OLTP captures the data and OLAP uses this data for analysis and decision support (Sharda et al., 2024, s. 209) In the Olist project, orders, payments, customer, products, order status, reviews and seller come from OLTP sources CSV files. We use ETL to – extract, transform and load them into a star schema OLAP warehouse for analysis over time, product, payment type, order, seller and geography.

ETL and Data Lineage

To make the data usable we run an ETL process, in this project using Pentaho. ETL is an abbreviation for; Extract, Transform, Load. We extract Olist data from the CSV files, and exports it into a staging area. Transform by cleaning, standardizing and integrating the data using rules/lookups and load the cleansed data into the star schema (dimensions first, then the facts). Modern ETL also handles scheduling, error handling, logging and metadata so the pipeline is auditable and reliable. (Sharda et al., 2024, s. 246).

A data warehouse is a pool of data built for analysis and decision support, not for a day-to day transaction. Data are usually structured and prepared to be used in analytical activities like queries, reports, dashboards and data mining. (Sharda et al., 2024, s. 190). Core data warehouse fundamentals following Inmon, a data warehouse is subject oriented, integrated, time-variant and non-volatile and it's built for analysis rather than updates (Sharda et al., 2024, s 194).

In our model this means that data are modelled by business subjects, like orders, products, customers, order status and sellers. They are integrated to a consistent format with surrogate keys; this retain history across 2016-2018 via a date dimension and are loaded append only by ETL (Extract – Transform – Load).

Dataset

The customers dataset holds information about the customer, and each order has a unique customer_id so you can identify returning customers. You can also find customers location. The geolocation Brazilian zip codes can be used to show where the sellers and customers are located.

The order items dataset holds information about the items purchased in the orders and the items price. In the payments dataset you find information about the payment options, how the orders are paid and the transaction value. There is also an order Reviews dataset that includes reviews and comments from the customers with a review-score.

In the orders dataset we find the core data. It holds all the information about the orders with customer, status, delivery and so on. In the products dataset we find information about the different products and description with photos and measurements.

The sellers dataset is where you can find information about the sellers. There is also a translation dataset for translation of the product categories from Portuguese to English.

The payment type dataset provides the value the values used to build dim_payment_type with credit, card and boleto.

The order status dataset provides the values used to build dim_order_status with delivered, shipped and cancelled.

Purpose and Business context

The purpose for Olist's Brazilian marketplace is to build a decision ready star schema warehouse to gain insights and explain customer behaviour, product sales and seller performance over time and geography. The output will show growth and seasonality, uncover where revenue concentrates (by category, state, city and seller), and rank top/bottom performers. It also highlights how freight % affects profitability.

We can track revenue, orders, AOV and freight. In this way we can set targets, allocate inventory, marketing and delivery. The model supports descriptive business intelligence dashboards, drill downs and simple time like YoY and seasonality.

1.2 Data Warehouse script

The DW-script is implemented in SQL and hold the following dimension tables:

Dimensions: Dim_customer, dim_products, dim_seller, dim_payment_type, dim_order_status and dim_date.

Fact tables: fact_order_sales, fact_order-reviews, fact_payments and fact_orders

Modeling - Star Schema

Star schema is the simplest and most widely used dimensional model. It has a central fact table that is surrounded by dimension tables that are connected via foreign key. (Sharda et al., 2024, s. 239). The star schema design provides a clear separation between the facts and the dimensions. Facts are the quantitative measurements and can represent amounts quantities, rates, percentages and duration. The dimensional are to provide the context for the facts and what the measurements mean. Dimensions can be descriptive, hierarchical, stable and detailed. In the dimensions we always use surrogate keys and primary key (business key), not the natural key. This is for faster joins, support of slowly changing dimension strategies, and handle missing values and changes in the natural keys. Date, product, seller and customer act as conformes dimensions sheard across the fact tables, enabling cross subject analysis, etc sales vs reviews by month and seller.

Fact tables (Grain)

Granularity refers to the level of details that is stored in the facts table. It determines the questions the data warehouse can answer. The rule is to store data at the lowest level of detail that is needed for analysis because you can always aggregate up (Sharda et al., 2024, s. 124).

Fact tables	**Grain**
Fact_order_sales	One row per order line item , natural identifier on order_id and order_item_id
Fact_order_reviews	One row per review, natural identifier on review_id
Fact_payments	One row per payment transaction per order, surrogate identifier payment_key, multiple rows may share the same order_id
Fact_orders	One row per order, natural identifier order_id

KPI/Measures

Organizations manage performance with outcome-oriented metrics. For external groups, these are service- level agreements and KPI – key performance indicators, internally. Set as typical enterprise targets to be tracked over time (Sharda et al., 2024, s. 255). In our model, KPIs are implemented as DAX measures in Power BI, they're evaluated at query time over the star schema and inherit filters from dimensions (date, product, seller, customer and order status). In our model we track revenue, orders, AOV, freight%, profit proxy and revenue YoY so we can monitor progress and slice by category, state, city and seller.

Dimensions table

All dimensions use surrogate key (e.g, product_key, seller_key, customer_key etc.) with business keys retained as attributes. Relationship are 1:n from each dimension to the corresponding fact on surrogate keys.

Dimension tables	**Attributes, Surrogate Keys**
Dim_customer	customer_key(*Surrogate key*), customer_unique_id(*Business key)*, customer_zip_code_prefix, customer_city. Customer_state
Dim_products	product_key(*Surrogate key)*, product_id(*Business_key*), product_category_name, product_photos_qty, product_weight, product_length_cm, product_height_cm, product_width_cm
Dim_seller	seller_key(*Surrogate_key)*, seller_id(*business key)*, seller_city, seller_state
Dim_paymeny_type	payment_type_key(*Surrogate key)*, payment_type
Dim_order_status	orderer_status_key(*Surrogate key*), orderer_status
Dim_date	Date_key(*Surrogate key)*, full_date, day_of_week_number,day_of_week_name, day_of_month, day_of_year, month_number, month_name, quarter_number, quarter_name, year, is_weekend, is_holiday

Omitting SCD

Because of the Pentaho learning curve, and lack of knowledge at early stage. We omitted due to time and complexity. The current model still supports period over time analysis via date dimension.

Reflections on ETL process

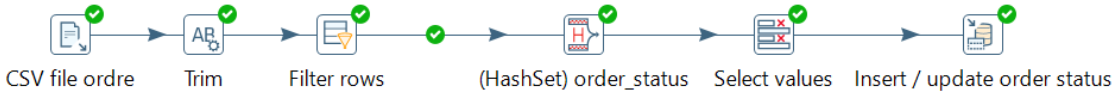
The ETL process step took time because of the Pentaho learning curve, and lack of knowledge at this stage. In hindsight we should have added more transformations, e.g., standardizing city names, trim more whitespace and remove impossible values. With what we know now, we would have added these steps and produce additional text file output, cleaned exports per dimension. If we had more time and now with new knowledge, we would have tried to implement a Type 1 SCD since we now understand these patterns far better than when we started.

1.3 ETL process explanation with screenshots

1.3.1 ETL Dimension tables

ETL_Dim_OrderStatus

olist_order_dataset.csv ==> dim_order_status

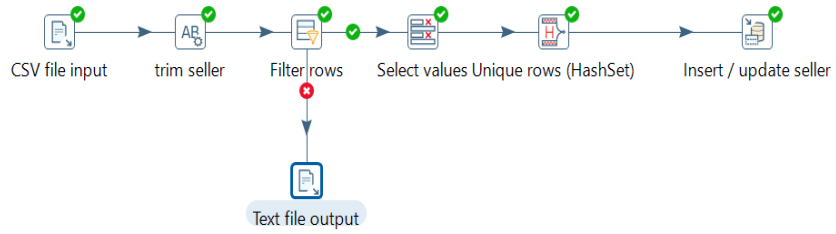


Explanation:

- We extract olist_order_dataset.csv into Pentaho and detect header row. We bring the source fields into the stream and define fields with proper metadata String and length.
- Trim string operations on order_status to remove whitespace noise and a consistent display.
- We filter rows where order_status IS NOT NULL, we don`t accepts NULL in the dimension.
- In select values we keep only the required colums and metadata to prevent mismatches. In the order dataset we keep only order_status.
- We use Hashset so we don't duplicate on order_status, and it ensure one row per order_status.
- We load the transformed data into table insert/update. Insert the cleaned data into postgreSql, using the name public.dim_order_status to connect with OlistDW.

ETL_Dim_Seller

olist_seller_dataset.csv ==> dim_seller

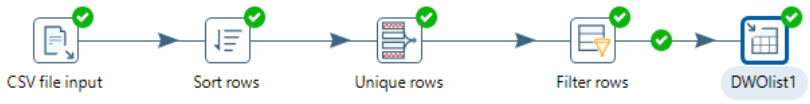


Explanation:

- We extract olist_seller_dataset.csv into Pentaho and detect header row. We bring the source fields into the stream and define fields, seller_id, seller_city and seller_state with proper metadata String and length.
- Trim string operations on seller_id, seller_city and seller_state to remove whitespace noise and a consistent display.
- We filter rows where seller_id IS NOT NULL, we don`t accepts NULL in the dimension. We write rejected rows to a csv text file output.
- In select values we keep only the required colums and metadata to prevent mismatches. In seller we keep seller_id, seller_city and seller_state.
- We use Hashset so we don't duplicate on seller_id, and it ensure one row per seller.
- We load the transformed data into table insert/update. Insert the cleaned data into postgreSql, using the name public.dim_seller to connect with OlistDW.

ETL_dim_customer

olist_customers_dataset.csv ==> dim_customer

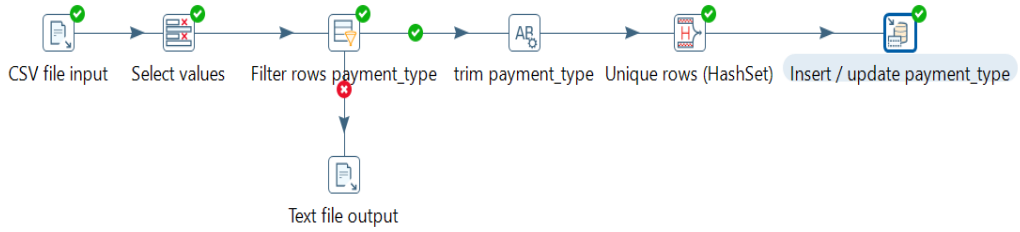


Explanation:

- We extract olist_customer_dataset.csv into Pentaho and detect header row, and define fields, customer_id, customer_unique_id, customer_zip_code_prefix,customer_city, customer_state. We use strings with appropriate lengths.
- In the transform we used sort rows, unique rows and filter rows. We sorted on customer_unique_id and remove duplicates withe unique rows. We filtred out records where customer_ zip_code_prefix is not null fails. This keep only the rows with a zip prefix.
- We load the transformed data into table output. Insert the cleaned data into postgreSql via OlistDW connection into the public.dim_customer table.

ETL_dim_payment_type

Olist_order_payments_dataset.csv ==> dim_payment_type

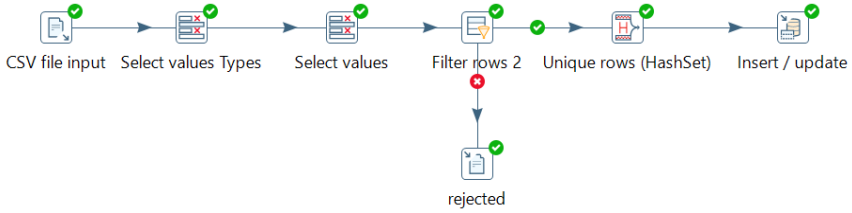


Explanation:

- We extract olist_order_payment.csv dataset into Pentaho and detect header row, and load source fields once with correct types.
- In select values we only keep payment_type, so we can push less data downstream, to make it faster.
- We filter rows where payment_type IS NOT NULL, we don`t accepts NULL in the dimension. We write rejected rows to a csv text file output.
- Trim string operations on payment_type, to remove whitespace noise and a consistent display.
- Use HashSet to dedupe only on payment_type, the dimension should have one row per paymenttype.
- We use insert/update to dim_payment_type with lookup key payment_type

ETL_dim_products

Olist_products_dataset.csv ==> dim_products

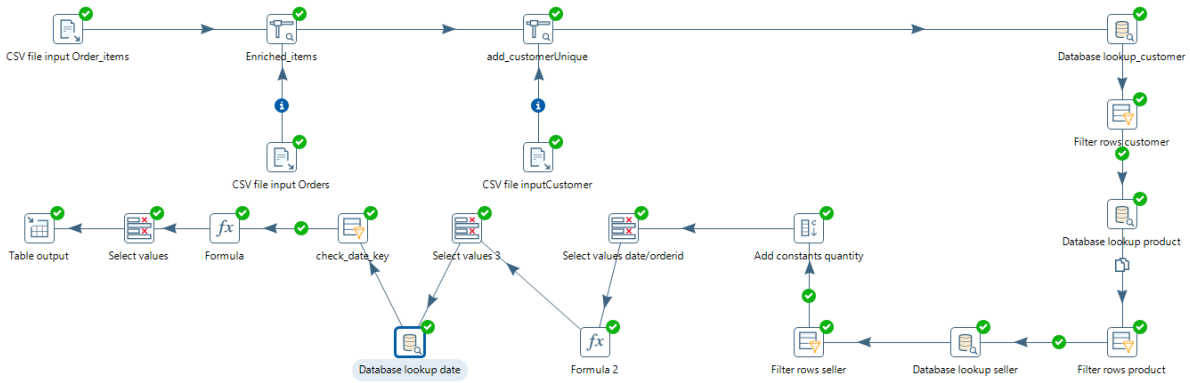


Explanation:

- We extract olist_product_dataset.csv into Pentaho and detect header row, and use select values type for to define metadata for the fields, types and length.
- In the transform we used select values that drop irrelevant fields such's as product_description_length and product_name_length. We used filter rows to keep only the valid rows. Name and quantity to IS NOT NULL, and weight, hight and cm to > 0. We send failures to a file “rejected.txt”. We De-duplicate product_id with HashSet unique rows.
- We inserted/updated the cleaned, unique products into postgresSql OlistDW public.dim_products with the key product_id.

1.3.2 ETL Facts tables

ETL fact_order_sales

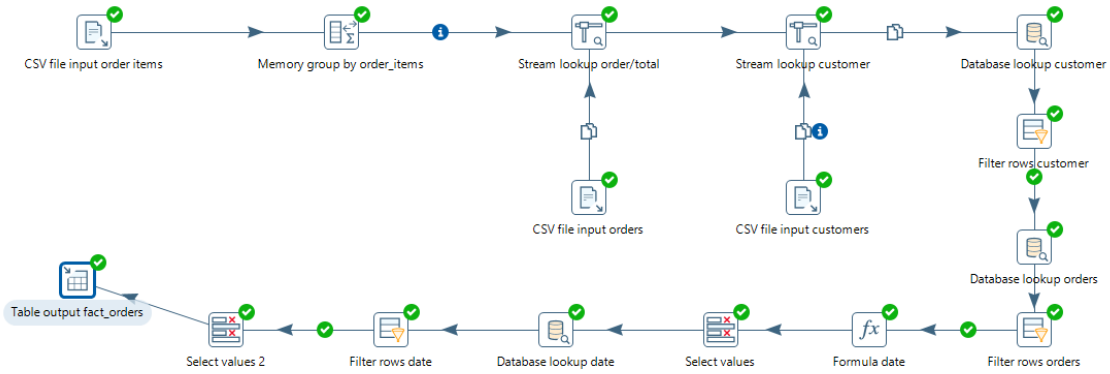


Explanation:

- We extract olist_order_items_dataset.csv into Pentaho and detect header row.
- We joining order_items.csv using a stream lookup on order_id . This extends the dataset with customer_id and order_purchase_timestamp. We added the unique_customer_key by performing a stream lookup against olist_customer_dataset.csv. This is joining the customer_id and ensures that each customer can be tracked across multiple orders.
- Database lookup is used on customer and product to validate and connect customer and products the customer and product dimension. In this step, the keys from the source like customer_id and product_id are replaced with surrogate keys from the dimensions. On these lookups we used filter rows customer and products. Filter rows customer keeps only valid customers. Filter rows products ensures that only valid products references are included.
- We added a database lookup for seller to connect seller information from the seller dimention, and added filter rows to remove invalid or missing seller records.
- Constant quantity is added for the constant field quantity = 1, each row in order_items represents one unit, so default quantity is one.

- Select values is used on date and order_id. We standardize the metadata and date. Order_item_id to string, order_purchase_timestamp to order_date_raw as string and ensure the shipping_limit_date is date.
- We used formula 2 to clean date text from raw timestamp .
- In select values 3 we converted the date text into a numeric key, rename date_key_stt to date_key Integer, length 8.
- To add date we used a database lookup on date. We validate the date_key against dim_date and bring back a marker date_key_found to give integrity to the Date dimension.
- We used filter rows on check_date_key to keep only records where date_key_found IS NOT NULL.
- In formula we made a profit_margin. Profit_margin = Price – freight_value, to have a ready query matric in the fact table.
- In the last select values we choose the exact load set for the fact table.
- Table output maps stream fields to table columns. Validated facts with enforced keys to date, product, seller and customer.

ETL_fact_orders

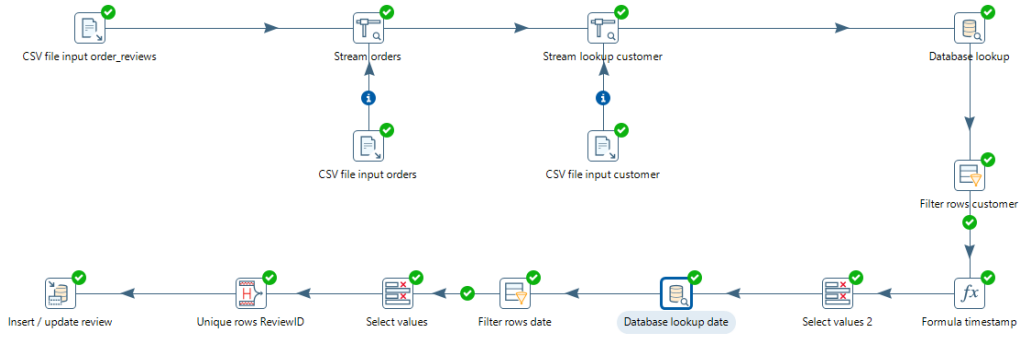


Explanation:

- We extract olist_order_items_dataset.csv into Pentaho and detect header row, and added key fields like order_id, order_item_id, product_id, sller_id, price and freight_value.
- In memory group by order_items we aggregate the order items up to order level. We made a group field order_id and aggregates total_amount and freight_value.
- We made stream lookup on order_id from csv orders, on fields retrievedwe mapped total_amount and freight_value and extends the order stream with aggregated totals.
- We used the olist_customer_dataset csv to extend the orders with the unique customer key and added a stream lookup customer that joins the order stream. Each order gets its unique customer identifier, so we can track customers on multiple orders.
- Database lookup is used on customer and orders to validate and connect customer and orders, to the customer and order dimension. The lookups also replace the natural keys with surrogate keys from the dimensions, which are stable and optimized for joins in the fact table. On these lookups we used filter rows customer and orders. Filter rows customer keeps only valid customers. Filter rows orders ensures that only valid orders references are included.
- We added a database lookup for seller to connect seller information from the seller dimension, and added filter rows to remove invalid or missing seller records.

- We used a formula date to clean date text from order_purchase_timestamp create a friendly date key for joining the date dimension.
- In select values we change the metadata form date_key_str to date_key, Integer, length 8. This ensures date_key matches the format of the surrogate key in dim_date.
- To add date we used a database lookup on date. We validate the date_key against dim_date and bring back a marker date_key_found to give integrity to the Date dimension.
- We used filter rows on check_date_key to keep only records where date_key_found IS NOT NULL.
- In the last select values we choose the exact load set for the fact_orders table.
- able output maps stream fields to table columns. Validated facts with enforced keys to date, orders and customer.

ETL_fact_orders_reviews

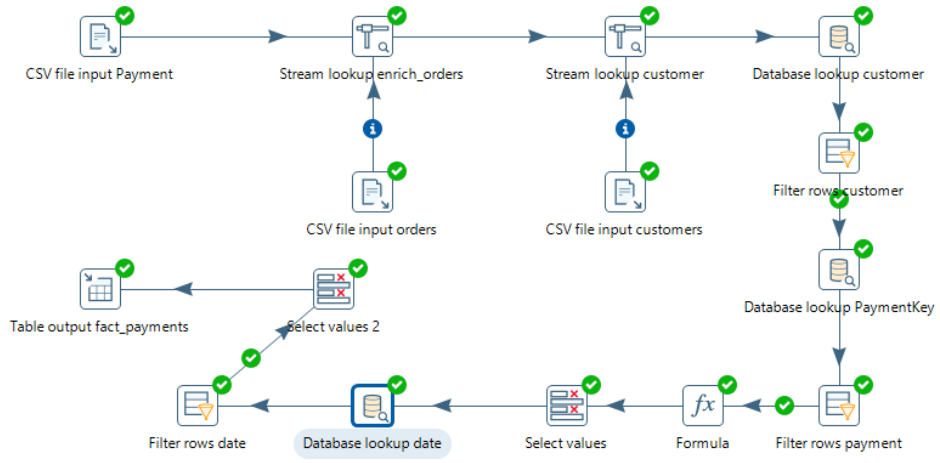


Explanation:

- We extract olist_order_reviews_dataset csv into pentaho and detect the header row. We brought in the key fields from the order_reviews_dataset.
- We then extend the review stream with order information by using stream lookup on order_id form olist_orders_dataset csv. In the lookup we retrieved customer_id, which links each review to the customer who places the order.
- To extend the stream we use olist_customer_dataset csv with a stream lookup customer. This jioins on customer_id, ensuring that each reviwe is linked to the correct customer.
- A database lookup is used to map the customer_unique_id to the surrogate key customer_key in the customer dimension. Surrogate keys are used instead of natural keys because they optimize joins in the star schema. We added filter rows on customer to keep only valid records where customer_ key exists in the dimension.
- We used formula timestamp for calculated fields on has_comment and has title, that uses flag if the review has a written comment or has a title. In review_date_key_str it extracts and formats the creation date to a string.
- When we used select values we converts the date strings into integers since the surrogate keys in dim_date are stored as intergers.
- To add date we used a database lookup on date. We validate the review_date_key against dim_date. Keeps only rows where a valid date_key exists.

- We used filter rows on check date_key to keep only records where date_key_found IS NOT NULL.
- In the last select value we adjust metadata for review_creation_date and review_answer_timestamp so they are proper Date types.
- We used unique rows to remove duplicates based on review_id and order_id. This guarantees one fact record per review/order.
- In the insert /update review we insert the validated records into fact_order_reviews (or updating).

ETL_fact_payments



Explanation:

- We extract olist_order_payments_type_dataset csv into Pentaho and detect the header row. We brought in the key fields from the dataset for all payment related facts.
- Stream lookup extends orders, by joining the payment data with the orders dataset using order_id. Adds information like the customer_id.
- The second stream lookup is joining the customer dataset using customer_id, this adds the customer_unique_id. This step makes it possible to connect the payment to the correct customer.
- We use a database lookup customer, customer_uniqe_id to lokk up the surrogate customer_key. We used filter rows to check that customer_key IS NOT NULL to prevent non existent customer.
- In the database lookup paymentKey we map payment_type to the surrogate key in dim_payment_type. We filter rows payment to validate the payment values, payment_key IS NOT NULL.
- In the formula we calculate date_key_str from order_key_str form orders purchase timestamp, so we later can join the date dimension.
- We used the select values to covert the date string into an integer, date_key.

- To add date we used a database lookup on date. We validate the date_key against dim_date and bring back a marker date_key_found to give integrity to the Date dimension.
- We used filter rows on check_date_key to keep only records where date_key_found IS NOT NULL.
- In the last select values we choose which fields should be passed on to the final table. This cleans and organizes the stream so only the relevant, transformed fields are kept.
- In table output we load data into the OlistDW in fact_payments

1.4 Data quality issues identified and resolution strategies

When we were designing and loading the dimension and fact tables, we considered common data quality issues that may arise during the ETL process. To ensure reliable and consistent data in the warehouse, we choose to use following strategies.

Lookup database and filter rows: The tables may contain customer_id, date_key, product_id values that don't exist in the corresponding dimension table. To enforce referential integrity, we used database lookups against dim_customer, dim_date and dim_products. Records with missing or invalid foreign keys are written to a reject log and excluded from the warehouse load.

Unique rows (HashSet): Multiple entries for the same review, order, or payment can occur in raw data. We would prevent this by using unique row HashSet to keep only one valid record per business event.

Select values and formula: The source files often stored dates and timestamps as text or a mix, some had text and other had date. This made it hard to use directly in the fact tables.

- With formula we could clean and reformat the raw timestamps. We extracted and renamed the date part of ex. Order_purchase_timestamp and removed characters to fit it to numeric key.
- With select values we could change the metadata and convert the clean string into correct data type as integer or date. Then we could match the date with the surrogate date_key in the date dimension.

Constants: To keep data consistent, we added default values. In the order_items dataset, each row represents one product unit, so we used a constant field quantity = 1. This avoids missing or inconsistent quantity values.

Calculations: We used certain calculations in the ETL for analysis purposes, such as profit_margin = price – freight_value, ensuring that these metrics are consistent and available in the fact tables.

Trim: we used trim/ string operations to trim leading and trailing spaces and enforce a consistent case. We applied this to dim_seller, dim_payment and dim_order_status after discovering data-quality issues later in the process. Trim removes the hidden whitespace and prevents duplicate dimension and lookup failures. With more time and in light of this finding, we would also apply the same cleanup to dim_products and dim_customer.

1.5 Performance considerations and optimization techniques applied

To make the ETL process more efficient and fact tables optimizes for querying, we applied some performance techniques.

Surrogate keys: Fact tables is joining to dimension tables through numeric surrogate keys BIGINT or INT. This improves join performance.

Filter rows: Invalid records were filtered immediately after lookups. By doing this we reduce the amount of unnecessary moving through the pipeline.

Stream lookups: We used the stream lookups when reference data was already available in memory, as you can see in the Pentaho CSV input for orders or customers.

Database lookups: Where used for validating existing dimension tables, this minimizes expensive queries.

Metadata: Data types were standardized early using select values, making transformation steps faster and more stable. Dates were intentionally re-typed at multiple points to avoid runtime conversions.

Task 2: Advanced Analytics Implementation

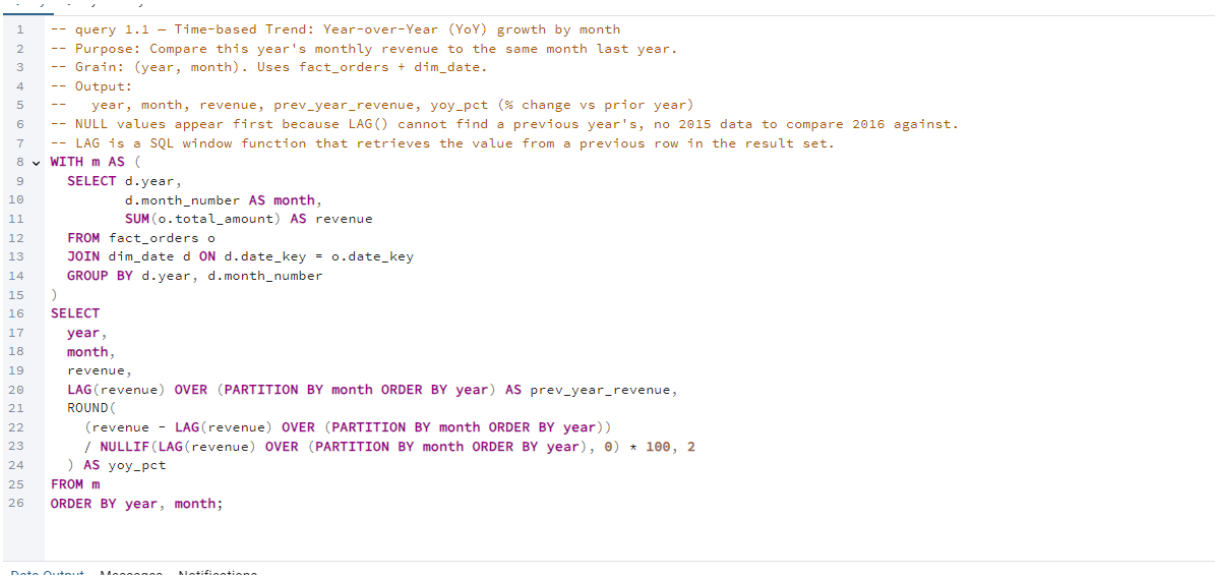
2.1 SQL Analytical Queries

2.1.1 Time-based Trend Analysis

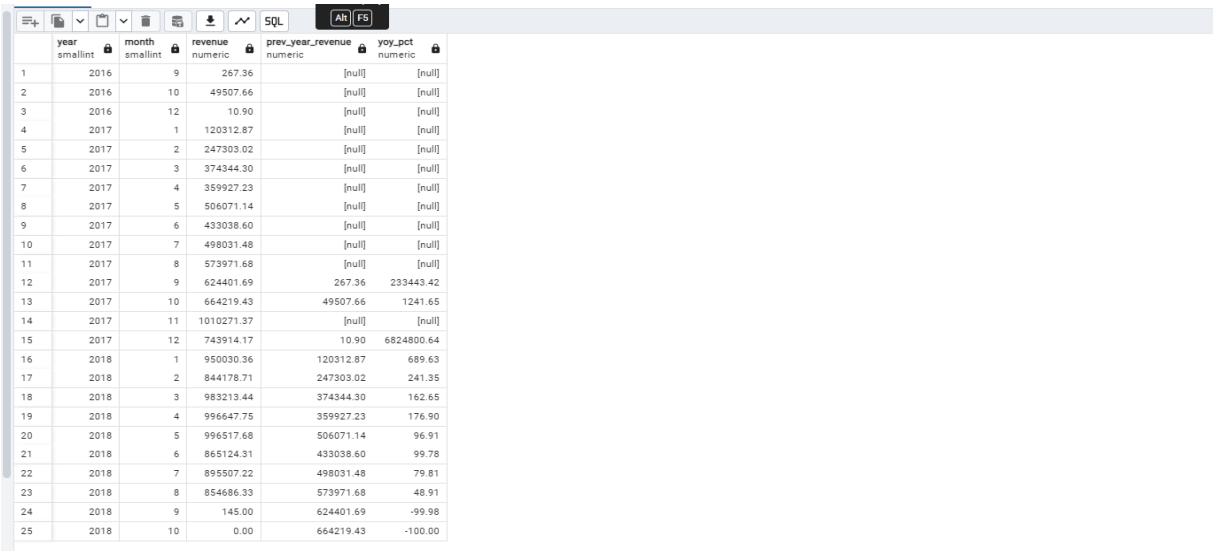
1.1 Year-over-year growth analysis

Use of AI to generate SQL- query:

In this query I built the base aggregation with select, sum, from, join and group by that gives the monthly revenue per year. I used ChatGPT to design the YoY logic with the function Lag(revenue) OVER (PARTITION BY month ORDER BY year) so each month compared to the same month last year. (OpenAI, 2025).



```sql
-- query 1.1 — Time-based Trend: Year-over-Year (YoY) growth by month
-- Purpose: Compare this year's monthly revenue to the same month last year.
-- Grain: (year, month). Uses fact_orders + dim_date.
-- Output:
--   year, month, revenue, prev_year_revenue, yoy_pct (% change vs prior year)
-- NULL values appear first because LAG() cannot find a previous year's, no 2015 data to compare 2016 against.
-- LAG is a SQL window function that retrieves the value from a previous row in the result set.
WITH m AS (
  SELECT d.year,
         d.month_number AS month,
         SUM(o.total_amount) AS revenue
  FROM fact_orders o
  JOIN dim_date d ON d.date_key = o.date_key
  GROUP BY d.year, d.month_number
)
SELECT
  year,
  month,
  revenue,
  LAG(revenue) OVER (PARTITION BY month ORDER BY year) AS prev_year_revenue,
  ROUND(
    (revenue - LAG(revenue) OVER (PARTITION BY month ORDER BY year))
    / NULLIF(LAG(revenue) OVER (PARTITION BY month ORDER BY year), 0) * 100, 2
  ) AS yoy_pct
FROM m
ORDER BY year, month;
```



	year smallint	month smallint	revenue numeric	prev_year_revenue numeric	yoy_pct numeric
1	2016	9	267.36	[null]	[null]
2	2016	10	49507.66	[null]	[null]
3	2016	12	10.90	[null]	[null]
4	2017	1	120312.87	[null]	[null]
5	2017	2	247303.02	[null]	[null]
6	2017	3	374344.30	[null]	[null]
7	2017	4	359927.23	[null]	[null]
8	2017	5	506071.14	[null]	[null]
9	2017	6	433038.60	[null]	[null]
10	2017	7	498031.48	[null]	[null]
11	2017	8	573971.68	[null]	[null]
12	2017	9	624401.69	267.36	233443.42
13	2017	10	664219.43	49507.66	1241.65
14	2017	11	1010271.37	[null]	[null]
15	2017	12	743914.17	10.90	6824800.64
16	2018	1	950030.36	120312.87	689.63
17	2018	2	844178.71	247303.02	241.35
18	2018	3	983213.44	374344.30	162.65
19	2018	4	996647.75	359927.23	176.90
20	2018	5	996517.68	506071.14	96.91
21	2018	6	865124.31	433038.60	99.78
22	2018	7	895507.22	498031.48	79.81
23	2018	8	854686.33	573971.68	48.91
24	2018	9	145.00	624401.69	-99.98
25	2018	10	0.00	664219.43	-100.00

The query's goal is to compare each month's revenue to the same month in the prior year to quantify year over year (YoY) growth.

In the result prev_year_revenue and yoy_pct are NULL in 2016. This is because 2016 is the first sales year. This is expected and we keep them NULL to avoid implying that there is "no change".

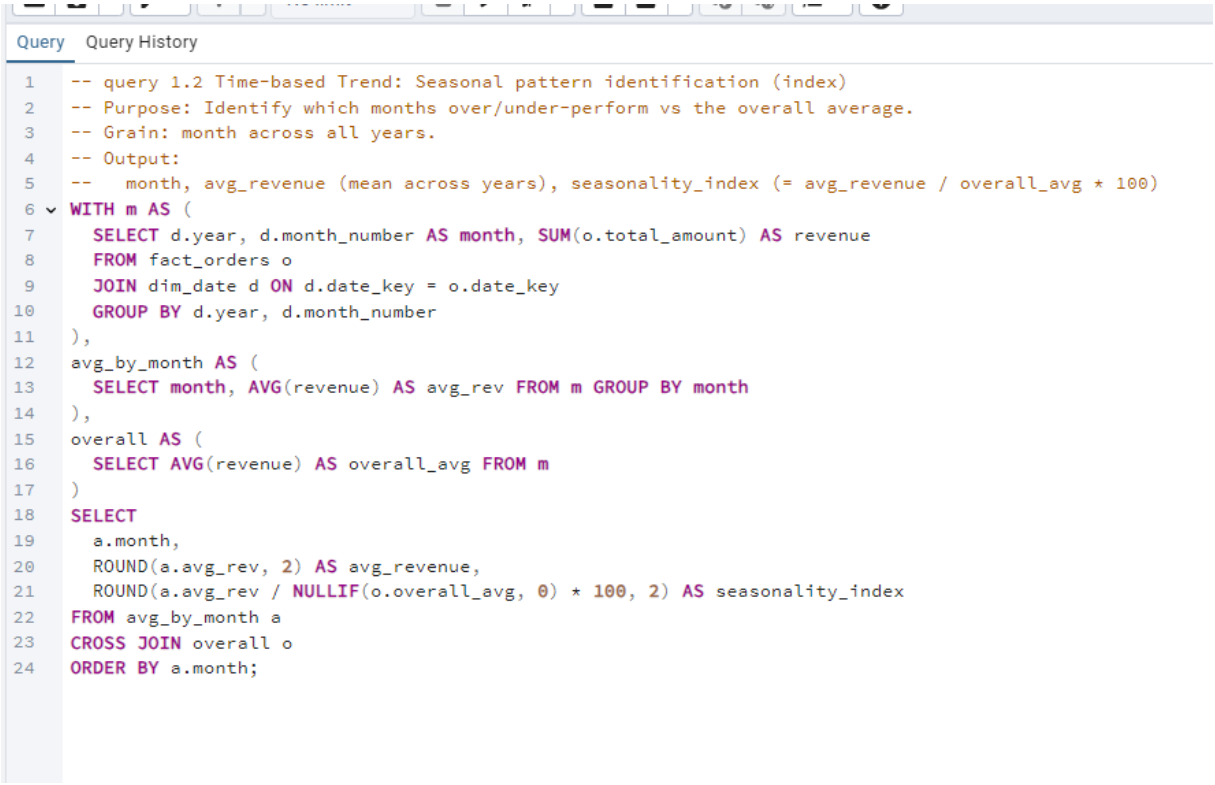
Revenue spikes in November 2017, consistent with Black Friday, that offers sales and promotional activity. There is a pullback in December 2017, but still far above December 2016. This is a typical dip after November campaigns.

YoY compare each month to the same month last year, if no prior year value exists, we show NULL in the column not 0. The YoY is extremely high in late 2017 because the baseline in 2016 is NULL. So, these values are not very meaningful.

1.2 Seasonal pattern identification

Use of AI to generate SQL- query:

I made the monthly part myself. With SELECT d.year, d.month_number AS month, SUM(o.total_amount) AS revenue from fact_orders. I joined to dim_date on date_key, and used GROUP BY year and month. I also wrote with little help the step that makes average per month across years. I found the rest hard and used ChatGTP to support to add the overall average and used CROSS JOIN. Chat made a query to compute the seasonality_index as avg_rev/ overall_avg*100. It also suggested NULLIF to avoid divide by zero. (OpenAI, 2025).



Query Query History

```sql
-- query 1.2 Time-based Trend: Seasonal pattern identification (index)
-- Purpose: Identify which months over/under-perform vs the overall average.
-- Grain: month across all years.
-- Output:
--   month, avg_revenue (mean across years), seasonality_index (= avg_revenue / overall_avg * 100)
WITH m AS (
  SELECT d.year, d.month_number AS month, SUM(o.total_amount) AS revenue
  FROM fact_orders o
  JOIN dim_date d ON d.date_key = o.date_key
  GROUP BY d.year, d.month_number
),
avg_by_month AS (
  SELECT month, AVG(revenue) AS avg_rev FROM m GROUP BY month
),
overall AS (
  SELECT AVG(revenue) AS overall_avg FROM m
)
SELECT
  a.month,
  ROUND(a.avg_rev, 2) AS avg_revenue,
  ROUND(a.avg_rev / NULLIF(o.overall_avg, 0) * 100, 2) AS seasonality_index
FROM avg_by_month a
CROSS JOIN overall o
ORDER BY a.month;
```

	month smallint	avg_revenue numeric	seasonality_index numeric
1	1	535171.62	98.44
2	2	545740.87	100.38
3	3	678778.87	124.85
4	4	678287.49	124.76
5	5	751294.41	138.19
6	6	649081.46	119.39
7	7	696769.35	128.16
8	8	714329.01	131.39
9	9	208271.35	38.31
10	10	237909.03	43.76
11	11	1010271.37	185.83
12	12	371962.54	68.42

The goal is to identify seasonal high and low months so we can plan inventory and marketing.

The seasonality index shows how a month performs relative to the overall monthly average. A seasonality index of 100 is exactly average. If the index is 120 it's 20% above average, if it is 80 it's 20% below average. We can see that in November the index is 185.8, about 86% above average, may be a strong Black Friday effect. In May to August it's between 119 – 131 that is 19 to 31% above average, this is a good season. In January and February, the index is 98-100 around average. In September and October, the index is 38 and 43, this is well below average and in December it's 68 below average.

The next plan can be to plan extra stock an ads in November for Black Friday week. In early December we can use repeat purchase offers so sales don't get a dip after November. In September and October, we can check if the low numbers are missing data or real. If they are real data, we can try to move some promos to October and see if that helps. We can also add an average of the last 3 months to avoid month to month.

2.1.2 Drill-down and Roll-up Operations

2.1 Multi-level aggregation queries

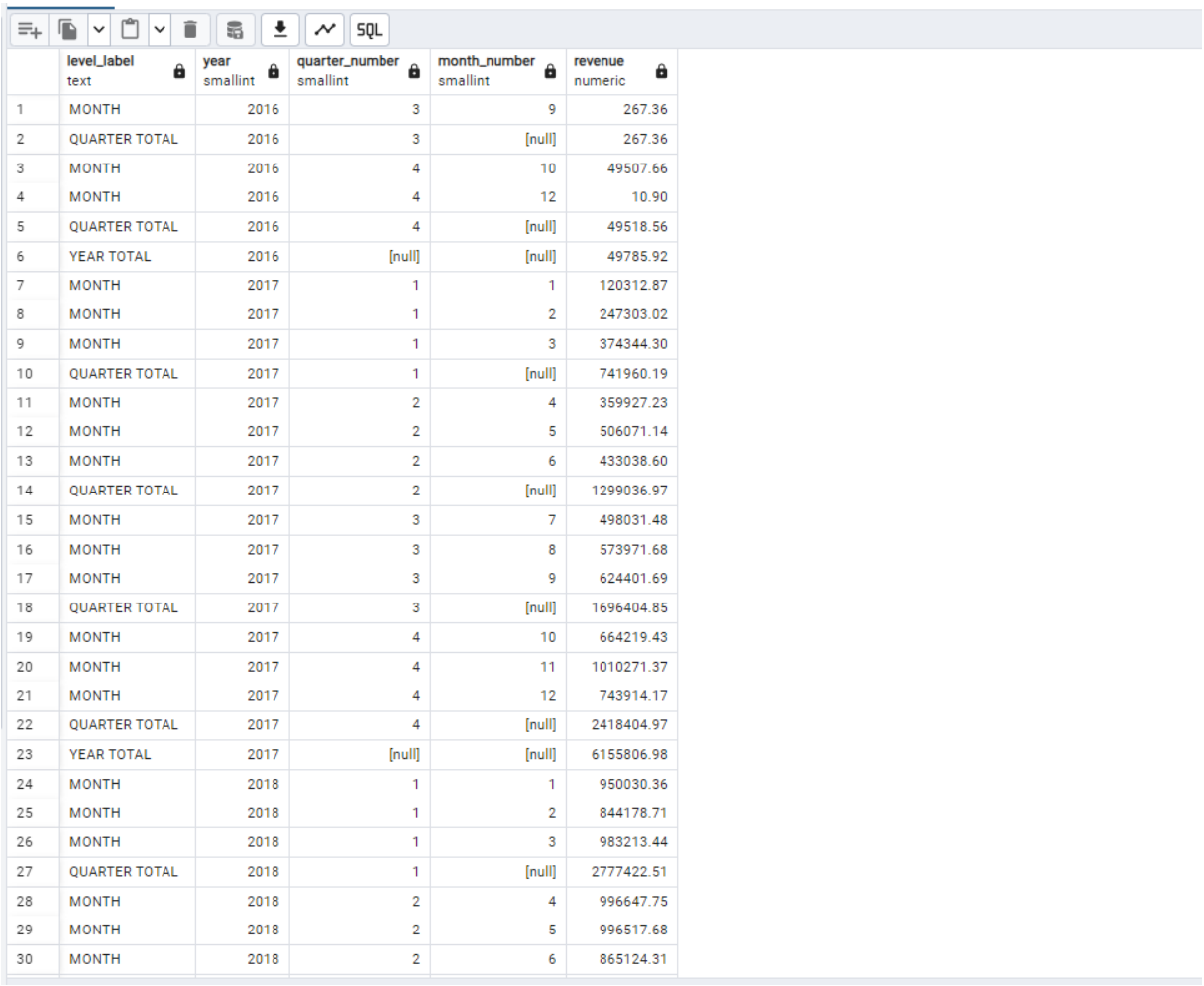
Use of AI to generate SQL- query:

I knew I had to join fact_orders to dim_date on date_key. Select year,quarter_number,month_number and sum. I was stuck with how to get month + quarter, total + year total + grand total in one result. ChatGPT added GROUPING inside a case to label rows, GROUP BY ROLLUP to generate all subtotals and ORDER BY GROUPING so details come first, the quarter totals, then year's total and grand total. (OpenAI, 2025).

iery Query History

```sql
-- query 2.1 — Drill-down/Roll-up: Multi-level aggregation (Year → Quarter → Month)
-- Purpose: show revenue at month level + subtotals per quarter and per year (no NULLs).
-- Grain: ROLLUP over (year, quarter_number, month_number).
-- Output: level_label, year, quarter, month, revenue.

SELECT
  CASE
    WHEN GROUPING(d.year)=1 AND GROUPING(d.quarter_number)=1 AND GROUPING(d.month_number)=1 THEN 'GRAND TOTAL'
    WHEN GROUPING(d.quarter_number)=1 AND GROUPING(d.month_number)=1 THEN 'YEAR TOTAL'
    WHEN GROUPING(d.month_number)=1 THEN 'QUARTER TOTAL'
    ELSE 'MONTH'
  END AS level_label,
  d.year,
  d.quarter_number,
  d.month_number,
  SUM(o.total_amount) AS revenue
FROM fact_orders o
JOIN dim_date d ON d.date_key = o.date_key
GROUP BY ROLLUP (d.year, d.quarter_number, d.month_number)
ORDER BY
  GROUPING(d.year),    d.year,
  GROUPING(d.quarter_number), d.quarter_number,
  GROUPING(d.month_number),   d.month_number;
```



	level_label text	year smallint	quarter_number smallint	month_number smallint	revenue numeric
1	MONTH	2016	3	9	267.36
2	QUARTER TOTAL	2016	3	[null]	267.36
3	MONTH	2016	4	10	49507.66
4	MONTH	2016	4	12	10.90
5	QUARTER TOTAL	2016	4	[null]	49518.56
6	YEAR TOTAL	2016	[null]	[null]	49785.92
7	MONTH	2017	1	1	120312.87
8	MONTH	2017	1	2	247303.02
9	MONTH	2017	1	3	374344.30
10	QUARTER TOTAL	2017	1	[null]	741960.19
11	MONTH	2017	2	4	359927.23
12	MONTH	2017	2	5	506071.14
13	MONTH	2017	2	6	433038.60
14	QUARTER TOTAL	2017	2	[null]	1299036.97
15	MONTH	2017	3	7	498031.48
16	MONTH	2017	3	8	573971.68
17	MONTH	2017	3	9	624401.69
18	QUARTER TOTAL	2017	3	[null]	1696404.85
19	MONTH	2017	4	10	664219.43
20	MONTH	2017	4	11	1010271.37
21	MONTH	2017	4	12	743914.17
22	QUARTER TOTAL	2017	4	[null]	2418404.97
23	YEAR TOTAL	2017	[null]	[null]	6155806.98
24	MONTH	2018	1	1	950030.36
25	MONTH	2018	1	2	844178.71
26	MONTH	2018	1	3	983213.44
27	QUARTER TOTAL	2018	1	[null]	2777422.51
28	MONTH	2018	2	4	996647.75
29	MONTH	2018	2	5	996517.68
30	MONTH	2018	2	6	865124.31

The query show revenue by month, with quarter and year subtotals, so we can spot seasonality, set targets and plan campaigns

In the roll-up results we can see that quarter 4 in 2017 is the strongest quarter, and November 2017 is the single biggest month. Quarter 3 in 2018 looks weak, but that can be because September and October may have missing data, it can be a data issue. If we see the complete months, quarter 2 in 2018 is the strongest quarter in 2018. In Quarter 4 especially November is boosted by campaigns for Black Friday, while 2018 dip can be explained by incomplete data.

The plan can be to use year total to set the annual target and quarter total to split the target across quarters. In November it is a peak and we can prioritize quarter 4 with more stock, ads and campaigns.

2.2 Hierarchical dimension analysis

Use of AI to generate SQL- query:

For me this was one of the hardest SQL queries. I could do the basic part of join, select and sum as revenue. AI showed me the right SQL query by working on how I wanted it. It showed me GROUP BY ROLLUP to get city rows, state subtotal and grand total . It also showed me with GROUPING inside CASE to label subtotal rows and ORDER BY GROUPING. It also helped me to set the query right to get the correct answer. (OpenAI, 2025).

```
-- query 2.2 – Drill-down/Roll-up: hierarchical dimension (State → City)
-- Purpose: Roll up revenue from city to state and to grand total (keep rollup, no NULLs shown).
-- Grain: ROLLUP(seller_state, seller_city).
-- Output: seller_state, seller_city (labeled as CITY / STATE SUBTOTAL / GRAND TOTAL), revenue.
-- revenue = sum (total_item_value) from fact_orders_sales, dim_seller on seller_key
SELECT
  CASE
    WHEN GROUPING(s.seller_state) = 1 THEN 'ALL STATES'         -- grand total level
    ELSE s.seller_state
  END AS seller_state,
  CASE
    WHEN GROUPING(s.seller_state) = 1 AND GROUPING(s.seller_city) = 1 THEN 'GRAND TOTAL'
    WHEN GROUPING(s.seller_city)  = 1 THEN 'STATE SUBTOTAL'
    ELSE s.seller_city
  END AS seller_city,
  SUM(f.total_item_value) AS revenue
FROM fact_order_sales f
JOIN dim_seller s ON s.seller_key = f.seller_key
GROUP BY ROLLUP (s.seller_state, s.seller_city)
-- order: cities - state subtotal - grand total
ORDER BY
  CASE WHEN GROUPING(s.seller_state)=1 THEN 1 ELSE 0 END,
  s.seller_state,
  CASE WHEN GROUPING(s.seller_city)=1 THEN 1 ELSE 0 END,
  s.seller_city;
```

Data Output | Messages | Notifications

	seller_state character varying	seller_city character varying	revenue numeric
1	AC	rio branco	299.84
2	AC	STATE SUBTOTAL	299.84
3	AM	manaus	1258.80
4	AM	STATE SUBTOTAL	1258.80
5	BA	arraial d'ajuda (porto seguro)	8435.21
6	BA	bahia	1089.04
7	BA	barro alto	66.28
8	BA	eunapolis	1880.00
9	BA	feira de santana	1296.77
10	BA	guanambi	18733.18
11	BA	ilheus	9644.62
12	BA	ipira	224.45
13	BA	irece	363.79
14	BA	lauro de freitas	238675.42
15	BA	porto seguro	1774.94
16	BA	salvador	23078.54
17	BA	STATE SUBTOTAL	305262.24
18	CE	caucaia	3535.16
19	CE	eusebio	90.23
20	CE	fortaleza	19121.06
21	CE	juzeiro do norte	62.11
22	CE	mucambo	809.41
23	CE	pacatuba	224.75
24	CE	varzea alegre	757.75
25	CE	STATE SUBTOTAL	24600.47
26	DF	brasilia	103619.16

The goal is to identify where revenue is concentrated by state and city so we can set priorities.This roll up shows revenue by city with a STATE SUBTOTAL for each state and a grand total. We can see that the total revenue is 15.6 Million, and most sales comes from a few states. We have SP=10.1Million, PR=1,44 Million, MG = 1,20Million, RJ = 0,92Million and SC=0,74Million. These five states have about 92% of all sales. within these states a handful of cities drive up the volume. We have San Paulo, Guarulhos, Santo Andre, Rio De Janero, Porto Alegre etc.

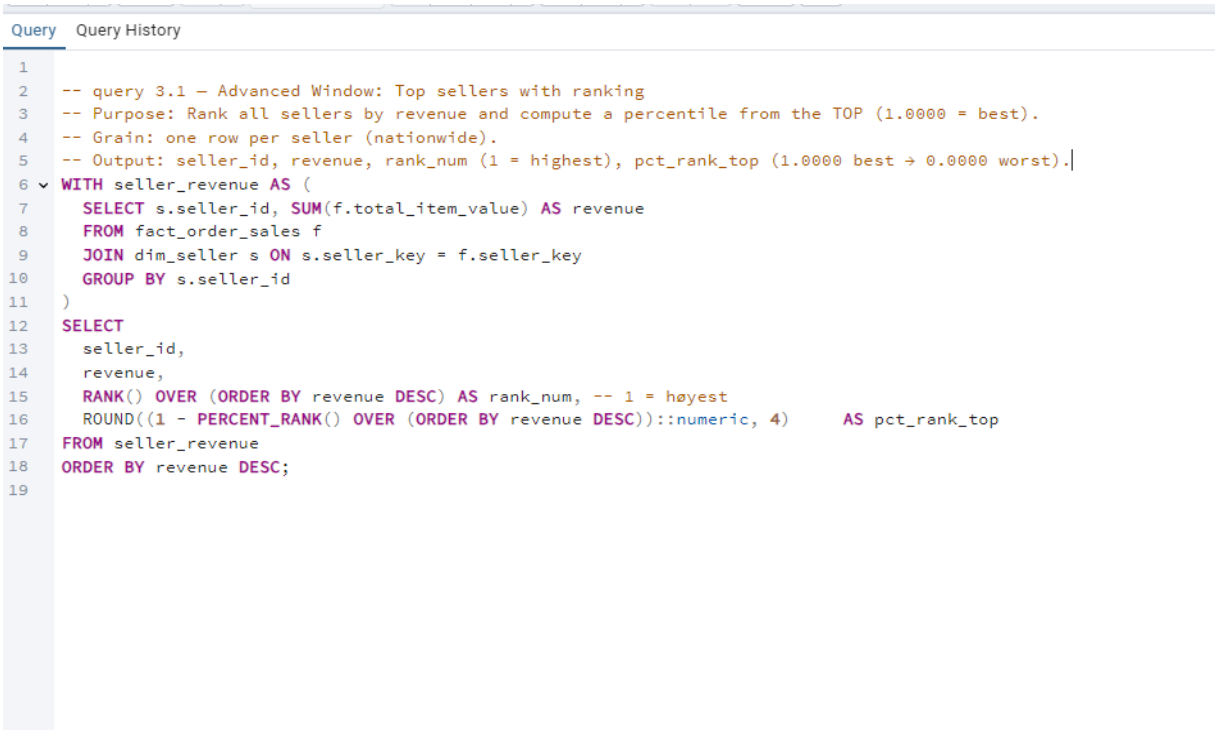
The next step would be to focus inventory, delivery and local marketing in SP and PR the top cities. Focus on growing cities in each top state in these regions, just below the top. We should keep this state – city roll up and review it every quarterly to track where revenue is concentrated.

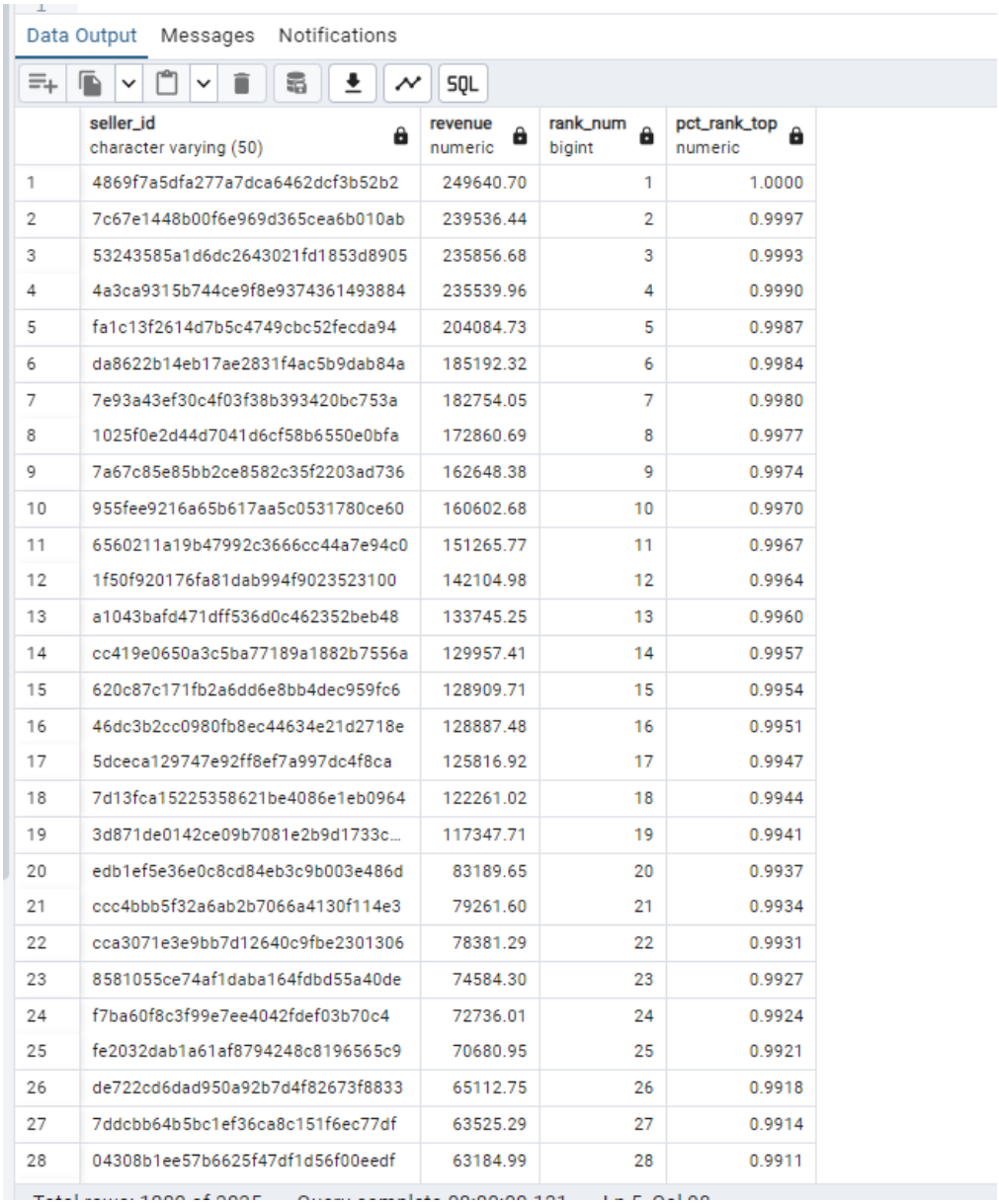
2.1.3 Advanced Window Functions

3.1 Ranking and percentile calculations

Use of AI to generate SQL- query:

I wrote the query that select seller_id and SUM(total_item_value) AS revenue and join to dim_seller, and group by seller_id. AI helped me add the built in window function RANK() OVER and a top oriented percentile using PERCENT_RANK, so we can rank sellers and score them. (OpenAI, 2025).





Data Output | Messages | Notifications

	seller_id character varying (50)	revenue numeric	rank_num bigint	pct_rank_top numeric
1	4869f7a5dfa277a7dca6462dcf3b52b2	249640.70	1	1.0000
2	7c67e1448b00f6e969d365cea6b010ab	239536.44	2	0.9997
3	53243585a1d6dc2643021fd1853d8905	235856.68	3	0.9993
4	4a3ca9315b744ce9f8e9374361493884	235539.96	4	0.9990
5	fa1c13f2614d7b5c4749cbc52fecda94	204084.73	5	0.9987
6	da8622b14eb17ae2831f4ac5b9dab84a	185192.32	6	0.9984
7	7e93a43ef30c4f03f38b393420bc753a	182754.05	7	0.9980
8	1025f0e2d44d7041d6cf58b6550e0bfa	172860.69	8	0.9977
9	7a67c85e85bb2ce8582c35f2203ad736	162648.38	9	0.9974
10	955fee9216a65b617aa5c0531780ce60	160602.68	10	0.9970
11	6560211a19b47992c3666cc44a7e94c0	151265.77	11	0.9967
12	1f50f920176fa81dab994f9023523100	142104.98	12	0.9964
13	a1043bafd471dff536d0c462352beb48	133745.25	13	0.9960
14	cc419e0650a3c5ba77189a1882b7556a	129957.41	14	0.9957
15	620c87c171fb2a6dd6e8bb4dec959fc6	128909.71	15	0.9954
16	46dc3b2cc0980fb8ec44634e21d2718e	128887.48	16	0.9951
17	5dceca129747e92ff8ef7a997dc4f8ca	125816.92	17	0.9947
18	7d13fca15225358621be4086e1eb0964	122261.02	18	0.9944
19	3d871de0142ce09b7081e2b9d1733c...	117347.71	19	0.9941
20	edb1ef5e36e0c8cd84eb3c9b003e486d	83189.65	20	0.9937
21	ccc4bbb5f32a6ab2b7066a4130f114e3	79261.60	21	0.9934
22	cca3071e3e9bb7d12640c9fbe2301306	78381.29	22	0.9931
23	8581055ce74af1daba164fdbd55a40de	74584.30	23	0.9927
24	f7ba60f8c3f99e7ee4042fdef03b70c4	72736.01	24	0.9924
25	fe2032dab1a61af8794248c8196565c9	70680.95	25	0.9921
26	de722cd6dad950a92b7d4f82673f8833	65112.75	26	0.9918
27	7ddcbb64b5bc1ef36ca8c151f6ec77df	63525.29	27	0.9914
28	04308b1ee57b6625f47df1d56f00eedf	63184.99	28	0.9911

The query identified the highest revenue sellers and created a simple percentile score (pct_rank_top) to segment sellers and set sales and operation priorities. Pct_rank_top shows a score from 0 to 1, where 1.0000 is the highest revenue and 0.0000 is the lowest. This is used to segment sellers in the list.

This Query ranks sellers by revenue. We can see that a few sellers make much more than the rest. The top five sellers each generate roughly $204k – 250k. This shows strong concentration that a small group drives most sales.

The next step is to focus on the top sellers, priority inventory, faster delivery and account management. Get grow in the next group of 20 % by coaching, target offers and cross selling. Review quarterly to see changes and new rising sellers.

Query 3.2 Moving Averages and Cumulative Measures

```sql
-- queery 3.2
-- Moving averages and cumulative measures
-- Moving average
-- business question- compute daily revenue, 7-day moving average, and cumulative revenue
-- daily renevue is the total renevue generated per day
-- 7-day moving average is the average daily revenue over the last 7 days
-- cumulative revenue is the running total revenue from start date up to each day
-- is is useful for year to date og periods KPI
-- daily_renevue_rank identifies the top performance days
-- dayly_renevue_percentile shows relative performance of each day
SELECT
    d.full_date,
    SUM(f.total_item_value) AS daily_revenue,

    -- 7-day moving average of daily revenue
    AVG(SUM(f.total_item_value))
        OVER (ORDER BY d.full_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS moving_avg_7d,

    -- Cumulative revenue
    SUM(SUM(f.total_item_value))
        OVER (ORDER BY d.full_date) AS cumulative_revenue,

    -- Rank of daily revenue (highest revenue = rank 1)
    RANK() OVER (ORDER BY SUM(f.total_item_value) DESC) AS daily_revenue_rank,

    -- Percentile of daily revenue
    PERCENT_RANK() OVER (ORDER BY SUM(f.total_item_value)) AS daily_revenue_percentile

FROM fact_order_sales f
JOIN dim_date d
  ON f.date_key = d.date_key
GROUP BY d.full_date
ORDER BY d.full_date;
```

Output | Messages | Notifications

full_date date	daily_revenue numeric	moving_avg_7d numeric	cumulative_revenue numeric	daily_revenue_rank bigint	daily_revenue_percentile double precision
2016-09-04	136.23	136.2300000000000000	136.23	613	0.004878048780487805
2016-09-05	75.06	105.6450000000000000	211.29	615	0.0016260162601626016
2016-09-15	143.46	118.2500000000000000	354.75	612	0.006504065040650406 5
2016-10-02	109.34	116.0225000000000000	464.09	614	0.0032520325203252032
2016-10-03	595.14	211.8460000000000000	1059.23	610	0.00975609756097561
2016-10-04	11229.98	2048.2016666666666667	12289.21	525	0.14796747967479676
2016-10-05	9645.94	3133.5928571428571429	21935.15	549	0.10894308943089431
2016-10-06	9131.23	4418.5928571428571429	31066.38	560	0.0910569105691057
2016-10-07	8135.97	5570.1514285714285714	39202.35	565	0.08292682926829269
2016-10-08	9352.95	6885.7928571428571429	48555.30	553	0.1024390243902439
2016-10-09	4018.98	7444.3128571428571429	52574.28	594	0.03577235772357724
2016-10-10	4493.39	8001.2057142857142857	57067.67	588	0.04552845528455285
2016-12-23	19.62	6399.7257142857142857	57087.29	616	0
2017-01-05	707.27	5122.7728571428571429	57794.56	609	0.011382113821138212

Query 3.2: revenue trend analysis

The objective of this query was to monitor daily revenue, 7-day moving average and track cumulative revenue. Daily revenue was calculated as the total sales value per day, while the moving average smoothed out short term fluctuations so we can see underlying patterns.

The cumulative revenue provides a running total that is useful for monitoring year-to-date quarterly progress. Ranking and percentiles were also applied to identify top-performing days. And, to evaluate each day's performance relative to the entire dataset.

The analysis revealed that the highest revenue occurred on November 24th and 25th, 2027. This is coinciding with the Black Friday weekend in Brazil, confirming the impact of seasonal shopping events. These measures together allow businesses to track progress toward revenue goals and evaluate the success of campaigns.

```sql
-- Sorted by ranking
--highest ranking days are 2017-11-24 and 2017-11-25
-- this is black friday weekend in Brazil so makes sense
SELECT
    d.full_date,
    SUM(f.total_item_value) AS daily_revenue,
    RANK() OVER (ORDER BY SUM(f.total_item_value) DESC) AS daily_revenue_rank,
    PERCENT_RANK() OVER (ORDER BY SUM(f.total_item_value)) AS daily_revenue_percentile
FROM fact_order_sales f
JOIN dim_date d
  ON f.date_key = d.date_key
GROUP BY d.full_date
ORDER BY daily_revenue_rank;
```

Output Messages Notifications

SQL

full_date date	daily_revenue numeric	daily_revenue_rank bigint	daily_revenue_percentile double precision
2017-11-24	177517.99	1	1
2017-11-25	71293.17	2	0.9983739837398374
2018-08-06	65746.87	3	0.9967479674796748
2018-05-16	64739.09	4	0.9951219512195122
2018-05-07	62460.21	5	0.9934959349593496
2018-05-14	61683.68	6	0.991869918699187
2018-05-10	58741.32	7	0.9902439024390244
2018-08-07	58037.89	8	0.9886178861788618
2018-06-11	57157.65	9	0.9869918699186991
2017-11-27	56209.74	10	0.9853658536585366
2017-11-28	55852.66	11	0.983739837398374
2018-04-09	55349.95	12	0.9821138211382113
2018-05-08	54536.21	13	0.9804878048780488
2017-12-04	53038.74	14	0.9788617886178862

Query 3.2.1: sort by ranking

2.1.4 Complex Filtering and Subqueries

Query 4.1 Identifying silent customers

```sql
-- queery 4.1
-- Multi dimensional filtering with EXISTS
-- uses EXISTs and NOT EXISTS to
-- identify silent customers.(Buisness qestion) The ones that shop but dont leve a review so you dont know if they were satisfied.
-- That can be further analysed på seeing if the silent customers have multiple orders.

SELECT c.customer_key,
       c.customer_unique_id,
       c.customer_city,
       c.customer_state
FROM dim_customer c
WHERE EXISTS (
    SELECT *
    FROM fact_orders o
    WHERE o.customer_key = c.customer_key
)
AND NOT EXISTS (
    SELECT 1
    FROM fact_order_reviews r
    WHERE r.customer_key = c.customer_key
);
```

Output Messages Notifications

SQL

customer_key [PK] integer	customer_unique_id character varying (50)	customer_city character varying (50)	customer_state character varying (50)
1	0000366f3b9a7992bf8c76cfdf3221e2	cajamar	SP
2	0000b849f77a49e4a4ce2b2a4ca5be...	osasco	SP
3	0000f46a3911fa3c08054444833370...	sao jose	SC
4	0000f6ccb0745a6a4b88665a16c9f0...	belem	PA
5	0004aac84e0df4da2b147fca70cf82...	sorocaba	SP
6	0004bd2a26a76fe21f786e4fbd80607f	sao paulo	SP
7	00050ab1314c0e55a6ca13cf7181fecf	campinas	SP
8	00053a61a98854899e70ed204dd4b...	curitiba	PR
9	0005e1862207bf6ccc02e4228effd9...	teresopolis	RJ
10	0005ef4cd20d2893f0d9fbd94d3c0d...	sao luis	MA
11	0006fdc98a402fceb4eb0ee528f6a8d4	mimoso do sul	ES
12	00082cbe03e478190aadbea78542e...	itapeva	SP
13	00090324bbad0e9342388303bb71b...	campinas	SP
14	000949456b182f53c18b68d6babc7...	sao bernardo do campo	SP
15	000a5ad9c4601d2bbdd9ed765d521...	porto alegre	RS

Query 4.1: Multi-dimensional filtering with EXISTS

This query focused on identifying customers who made purchases but did not leave a review. By using EXISTS to confirm purchases and NOT EXISTS to exclude those with reviews we can isolate the “silent customers”. These customers represent a challenge for the business, since they can’t know if they were satisfied or not with the purchase.

From a managerial perspective silent customers pose both risks and opportunities. They may be satisfied but don’t want to engage, or they may be dissatisfied an unwilling to share feedback. Identifying these customers is important so the business can design targeted follow-up actions such as surveys, personalized outreach or loyalty incentives. The object is to increase engagement and capture insights that may be lost.

Query 4.2: Correlated subqueries for comparative analysis

```sql
-- queery 4.2
-- Correlated subquery for comparative analysis
-- Busness question- find products that are more expensive than the average price of all products in theis category
-- the category average is calculated once in category_avg and the outer query compares each products average price with the precomputed average
WITH category_avg AS (
    SELECT p.product_category_name,
           AVG(s.price) AS avg_category_price
    FROM fact_order_sales s
    JOIN dim_products p
      ON s.product_key = p.product_key
    GROUP BY p.product_category_name
)
SELECT p.product_id,
       p.product_category_name,
       AVG(s.price) AS product_avg_price,
       ca.avg_category_price
FROM fact_order_sales s
JOIN dim_products p
  ON s.product_key = p.product_key
JOIN category_avg ca
  ON p.product_category_name = ca.product_category_name
GROUP BY p.product_id, p.product_category_name, ca.avg_category_price
HAVING AVG(s.price) > ca.avg_category_price
ORDER BY p.product_category_name, product_avg_price DESC;
```

Output Messages Notifications

SQL

product_id character varying (50)	product_category_name character varying (100)	product_avg_price numeric	avg_category_price numeric
2b69866f22de8dad69c976771daba91c	agro_industria_e_comercio	2990.0000000000000000	342.1248584905660377
b7a60a397d4efd05c1b5d398fb9f9097	agro_industria_e_comercio	2399.0000000000000000	342.1248584905660377
cd2f5c10e4e8dbc701f0bb68a09fdfe8	agro_industria_e_comercio	2199.0000000000000000	342.1248584905660377
eddb814fb553b6951f51b34c5f578ba0	agro_industria_e_comercio	1899.0000000000000000	342.1248584905660377
d5dbb4d9ecbbf2e312169e4c8f1b57f0	agro_industria_e_comercio	1476.3000000000000000	342.1248584905660377
5fb0955cb683eb6f65a1f613e502eef5	agro_industria_e_comercio	1360.0000000000000000	342.1248584905660377
ea1d59339533cb7f68f0319501d8e4d4	agro_industria_e_comercio	1234.0000000000000000	342.1248584905660377
cd5df6a3db7a3d064a55afd08289d762	agro_industria_e_comercio	1180.0000000000000000	342.1248584905660377
c183fd5d2abf05873fa6e1014ed9e06c	agro_industria_e_comercio	989.1000000000000000	342.1248584905660377
c89226b8a795ae3d6bca9d90b20dbf04	agro_industria_e_comercio	940.5000000000000000	342.1248584905660377
f3c179e260e0eeffbe02340259404cb1	agro_industria_e_comercio	934.9400000000000000	342.1248584905660377
3e3f442db862cb6fe99389a41b7acb84	agro_industria_e_comercio	894.9800000000000000	342.1248584905660377
f72144308168311d3ca5926e40e784e5	agro_industria_e_comercio	849.9800000000000000	342.1248584905660377
a95e101cee281e7cd36dfef5f80fabfe	agro_industria_e_comercio	799.9900000000000000	342.1248584905660377
e38487b365f562f0f5f6d8362cd36593	agro_industria_e_comercio	699.9800000000000000	342.1248584905660377
c75a6578b6475cf760d40cb340b3971c	agro_industria_e_comercio	589.9900000000000000	342.1248584905660377
b5015f9057f27b5bdefde4ca99368b54	agro_industria_e_comercio	559.9900000000000000	342.1248584905660377
b8faf829f47ba1f35f169da4acbdb88d	agro_industria_e_comercio	549.0000000000000000	342.1248584905660377
f72b26eeffe2345b486cfeb723305bac	agro_industria_e_comercio	498.0000000000000000	342.1248584905660377
66e94c5c07d64d44c16cd6903a79a4...	agro_industria_e_comercio	495.6000000000000000	342.1248584905660377
49b6d7e8192cd04ef8f69b1bf00b7eb1	agro_industria_e_comercio	469.0000000000000000	342.1248584905660377

Query 4.2: Products Above Category Average Price

The objective of this query was to identify products that are priced above the average within their respective product category. First, we calculated the average price in the categories. Then each products average price was compared to this benchmark.

The findings revealed which products were positioned as premium in their categories. This type of analysis is important for pricing strategy. It highlights opportunities for maximising profit margins while also identifying protentional risks that a product can be

overpriced to the customers expectation. From a business standpoint, these results support more informed decisions regarding competitive positioning and targeted marketing campaigns for premium products.

2.1.5 Business Intelligence Metrics

Query 5.1: Customer Lifetime Value (CLV) Analysis

```
-- query 5.1
--Customer/Product profitability analysis
-- Busness question- calculate clv(customer lifetime value)(total revenue-total freight cost)
-- to identifi the customers who spend most money, are most profitable

SELECT c.customer_key,
       c.customer_unique_id,
       c.customer_city,
       c.customer_state,
       SUM(s.total_item_value) AS total_revenue,
       SUM(s.freight_value) AS total_freight,
       (SUM(s.total_item_value) - SUM(s.freight_value)) AS customer_lifetime_value
FROM fact_order_sales s
JOIN dim_customer c
  ON s.customer_key = c.customer_key
GROUP BY c.customer_key, c.customer_unique_id, c.customer_city, c.customer_state
ORDER BY customer_lifetime_value DESC
LIMIT 10;
```

utput Messages Notifications

SQL

customer_key [PK] integer	customer_unique_id character varying (50)	customer_city character varying (50)	customer_state character varying (50)	total_revenue numeric	total_freight numeric	customer_lifetime_value numeric
3827	0a0a92112bd4c708ca5fde585afaa872	rio de janeiro	RJ	13664.08	224.08	13440.00
81963	da122df9eeddfedc1dc1f5349a1a690c	araruama	RJ	7571.63	183.63	7388.00
44448	763c8b1c9c68a0229c42c9fc6f662b93	vila velha	ES	7274.88	114.88	7160.00
82809	dc4802a71eae9be1dd28f5d788ceb526	campo grande	MS	6929.31	194.31	6735.00
26206	459bef486812aa25204be022145caa62	vitoria	ES	6922.21	193.21	6729.00
95807	ff4159b92c40ebe40454e3e6a7c35ed6	marilia	SP	6726.66	227.66	6499.00
24122	4007669dec559734d6f53e029e360987	divinopolis	MG	6081.54	146.94	5934.60
89689	eebb5dda148d3893cdaf5b5ca3040ccb	maua	SP	4764.34	74.34	4690.00
35071	5d0a2980b292d049061542014e8960bf	goiania	GO	4809.44	209.54	4599.90
27442	48e1ac109decbb87765a3eade6854098	joao pessoa	PB	4681.78	91.78	4590.00

Query 5.1: Customer/Product profitability analysis

The aim of this query is to calculate the Customer Lifetime Value (CLV) for each customer. The CLV is defined as total revenue generated minus freight cost. Customers were then ranked according to their profitability.

The result showed which customers that contributed most to the profit. These insights are essential for customer segmentation because they enable the business to allocate resources more efficiently. High-value customers can be prioritized for loyalty programs, personalized offers and premium services. The less profitable customers can be

targeted differently, by trying to make them more profitable. Overall, the CLV-analysis provides a foundation for customer relationship management strategies aimed at maximizing long-term profitability.

Query 5.2: Performance KPI calculations

```
--queery 5.2
-- Performance KPI calculations specific to your domain
-- what percentage of customers have placed more than one order?
-- KPI - repeat purchase rate
-- indicates customer loyalty
-- takes the customers with >1 order ÷ total customers) × 100
-- the rate 3,12% i very low, only 3 of 100 customers buy again
-- Olist relies on new customer aqusition.
-- the customer loyalty is very limited
-- they schold take action with campaigns, loyalty programs etc. for existing customers.
-- and analyse with the reviews, maye that can give som clues to why
WITH customer_order_counts AS (
    SELECT customer_key,
           COUNT(DISTINCT order_id) AS order_count
    FROM fact_orders
    GROUP BY customer_key
)
SELECT
    COUNT(*) FILTER (WHERE order_count > 1) * 100.0 / COUNT(*) AS repeat_purchase_rate
FROM customer_order_counts;
```

a Output | Messages | Notifications

SQL

repeat_purchase_rate numeric
3.1187562437562438

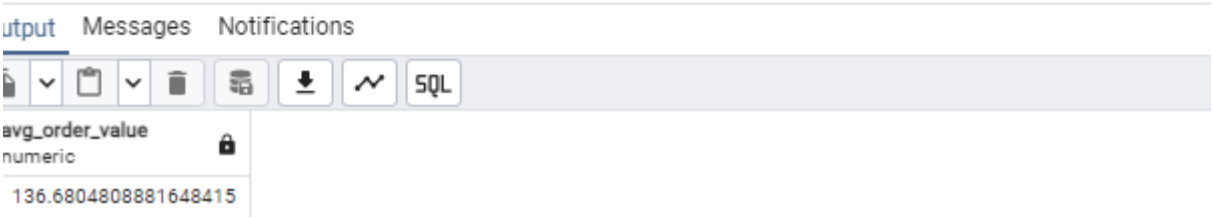
Query 5.2: Performance KPI – repeat purchase rate

The purpose of this query was to measure the degree of customer loyalty by calculating the repeat purchase rate. The repeat purchase rate is defined as the percentage of customers who placed more than one order. The analysis revealed that only 3,12 percent of customers make a repeat purchase.

This result indicates that the business is heavily dependent on constant acquiring new customers and that the loyalty among the existing customers remains minimal.

From a strategic perspective these findings reveal the need for action through targeted campaigns and loyalty programs. Increasing the repeat purchase rate would strengthen the company's long-term sustainability by reducing cost associated with acquiring new customers

```sql
-- query 5.2 one more.
-- KPI-Average Order value
-- buisness question: what is the total value of an order
-- total revenue ÷ total number of orders
-- to measure sale performance
-- results from query:
-- on average each customer brings 136,68 BRL in revenue
-- orders above will be considers high value orders, below low value orders.
-- This information is useful in loyalty programs and campaign targeting
-- this KPI can be uses with repeat purchase rate to calculate clv
SELECT
    AVG(total_amount) AS avg_order_value
FROM fact_orders
WHERE total_amount IS NOT NULL;
```



Query 5.2.1: Performance KPI – Average Order Value

Query 5.2 (Additional): Average Order Value (AOV)

I also calculated the average order value (AOV) by dividing the total revenue by the number of orders. The result showed that each order generates on average 136.68 BRL in revenue. This measure is important for evaluating sales performance. Orders above average can be categorized as high-value and targeted with upselling strategies. Orders below can be categorized for promotional incentives like marketing campaigns. Combined with the repeat purchase rate, AOV becomes a key input for estimating customer lifetime value. From a business perspective, the metric can inform the marketing strategy and the operational planning.

2.1.6 Summary of SQL Findings

We divided the queries between us with candidate 7016 working on the first five and candidate 7020 the last five. This was so we could both work to build our knowledge about complex SQL queries.

Together, these queries provide a comprehensive analytical framework for understanding business performance. The findings reveal both opportunities and weaknesses.

Query 1.1 shows year-over-year growth by month, captures seasonal spikes and treats the first year as a NULL base line. Query 1.2 revealed which months are above/below typical activity relative to the average, for capacity and campaign planning. Query 2.1 shows monthly figures with quarter and year totals to reveal seasons and phase target-setting. Query 2.2 revealed where revenue concentrates geographically to prioritize markets and inventory. Query 3.1 shows seller performance with ranking/percentiles to focus on top sellers and develop the next tier.

Query 3.2 demonstrated that revenue follows strong seasonal trends with exceptional peaks during Black Friday sales. Query 4.1 and 4.2 revealed a blind spot in customer engagement and highlighted products with higher margins. Query 5.1 showed that profitability is concentrated amount a limited group of customers. 5.2 underlined the structural weakness of a very low repeat purchase rate, complemented by an average order value that sets a clear benchmark for performance.

From these finding we have recommendations for the business. First the company should strengthen customer retention strategies. The low repurchase rate indicates most customers are one-time buyers, creating a high reliance on acquiring new customers. Loyalty programs, personalized offers and post-purchase follow-up cold improve retention.

The company should capitalize on premium selling products and high-value customers. Targeted marketing campaigns and differentiated service offerings for these segments may help maximise profitability.

Seasonal campaigns like Black Friday should be strategically leveraged but balanced with efforts to stabilize revenue throughout the year. This may include promotional events or exploring cross-selling opportunities.

In summary, Olist demonstrates strong sales capacity, but faces challenges in sustaining long-term growth due to weak customer loyalty. Addressing retention while optimizing product and customer profitability will be essential for building a more resilient business.

2.2 Python Analytics Integration

The Python analytics section compliments the SQL-queries by providing deeper statistical, predicative and prescriptive insights. Since the work with the facts tables In Pentaho (Task 1) took so much time we needed to divide the work to use our time more efficiently. Christin ended up doing all the facts transformations in Pentaho, and Mia did the Python part of task 2. ChatGPT and the Anaconda Assistant was used in creating the code for the analysis, as we were shown in class. The interpretation of the results and the business insights are our own.

Data Connection and Preparation

```
#Connect to postgres
import pandas as pd
from Tools.scripts.dutree import display
from sqlalchemy import create_engine

# Database credentials
user = "postgres"
password = [redacted]
host = "localhost"
port = "5432"
database = "OlistDW"

# engine
engine = create_engine(f"postgresql+psycopg2://{user}:{password}@{host}:{port}/{database}")

# Test-tilkobling
test_df = pd.read_sql("SELECT * FROM fact_order_sales LIMIT 5;", engine)
test_df
```

	order_sales_key	order_id	order_item_id	product_key	date_key	seller_key
0	1	00010242fe8c5a6d1ba2dd792cb16214	1	25386	20170913	514
1	2	00018f77f2f0320c557190d7a144bdd3	1	26728	20170426	472
2	3	000229ec398224ef6ca0657da4fc703e	1	22194	20180114	1825
3	4	00024acbcdf0a6daa1e931b038114c75	1	15109	20180808	2024
4	5	00042b26cf59d7ce69dfabb4e55b4fd9	1	8691	20170204	1598

Data was accessed directly from PostgreSQL using SQLAlchemy and pandas. The connection was established with the credentials, and the fact_order_sales table was loaded into a panda DataFrame to verify the connection.

2.2.1 Descriptive Analytics

Descriptive analysis was performed to provide an overview of key sales metrics and to understand the relationship between product, price and customer. Data from fact_order_sales, dim_products and dim_customer tables was extracted and combined. That includes price, freight value, quantity, product category and customer state. A sample of 10 000 records was used for initial exploration.

A statistical summary revealed that the prices vary widely from 3.49 BRL to 6.735 BRL. The median price was 74.98 BRL Most products were inexpensive, but a few high-priced items raised the average. Freight cost averaged 20 BRL, which is high compared to the typical product prices. All 27 states of Brazil were represented in the dataset. Sao Paulo has the most customers with over 40 percent of the orders (4,237), making it the most important region. The dataset included 69 unique categories. The most popular is cama-mesa_banho (beds, tables and bath).

	price	freight_value	quantity	product_category_name	customer_state
count	10000.000000	10000.000000	10000.0	10000	10000
unique	NaN	NaN	NaN	69	27
top	NaN	NaN	NaN	cama_mesa_banho	SP
freq	NaN	NaN	NaN	979	4237
mean	120.568763	20.141958	1.0	NaN	NaN
std	187.546617	16.434631	0.0	NaN	NaN
min	3.490000	0.000000	1.0	NaN	NaN
25%	39.900000	13.080000	1.0	NaN	NaN
50%	74.980000	16.320000	1.0	NaN	NaN
75%	136.917500	21.320000	1.0	NaN	NaN
max	6735.000000	375.280000	1.0	NaN	NaN

```
product_category_name
cama_mesa_banho           979
beleza_saude              847
esporte_lazer             770
moveis_decoracao          757
informatica_acessorios    742
utilidades_domesticas     601
relogios_presentes        533
telefonia                 420
brinquedos                416
ferramentas_jardim        369
Name: count, dtype: int64
customer_state
SP    4237
```

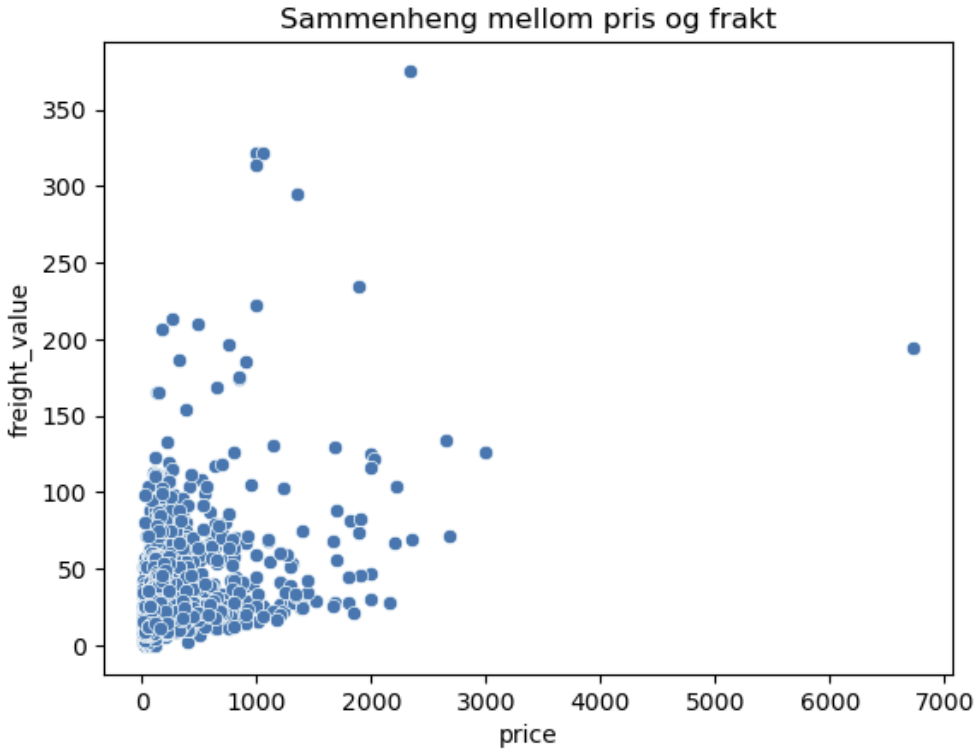
Correlation analysis of numeric variables indicated a weak positive relationship (r=0,44) between product and freight value. Quantity per order line is always one and gives no meaningful correlation. This suggests that price and freight costs should be considered separately in business decisions. Products with high freight cost compared to their price should be investigated and optimized. This can enhance the customer satisfaction and increase the likelihood of repeat purchases that we could see from the SQL analysis part were very low.

The analysis also highlights market characteristics in Brazil. As a very large country, shipping costs can naturally be high, which impacts the overall cost structure and customer experience. This represents an important area for the company to explore further to improve operational efficiency and customer retention.

Visual Interpretation

A scatterplot of price versus freight value confirmed the weak positive correlation and showed that higher priced items generally lead to higher shipping costs. These insights are important in pricing strategies, logistics optimization and promotional planning.

	price	freight_value	quantity
price	1.000000	0.436208	NaN
freight_value	0.436208	1.000000	NaN
quantity	NaN	NaN	NaN



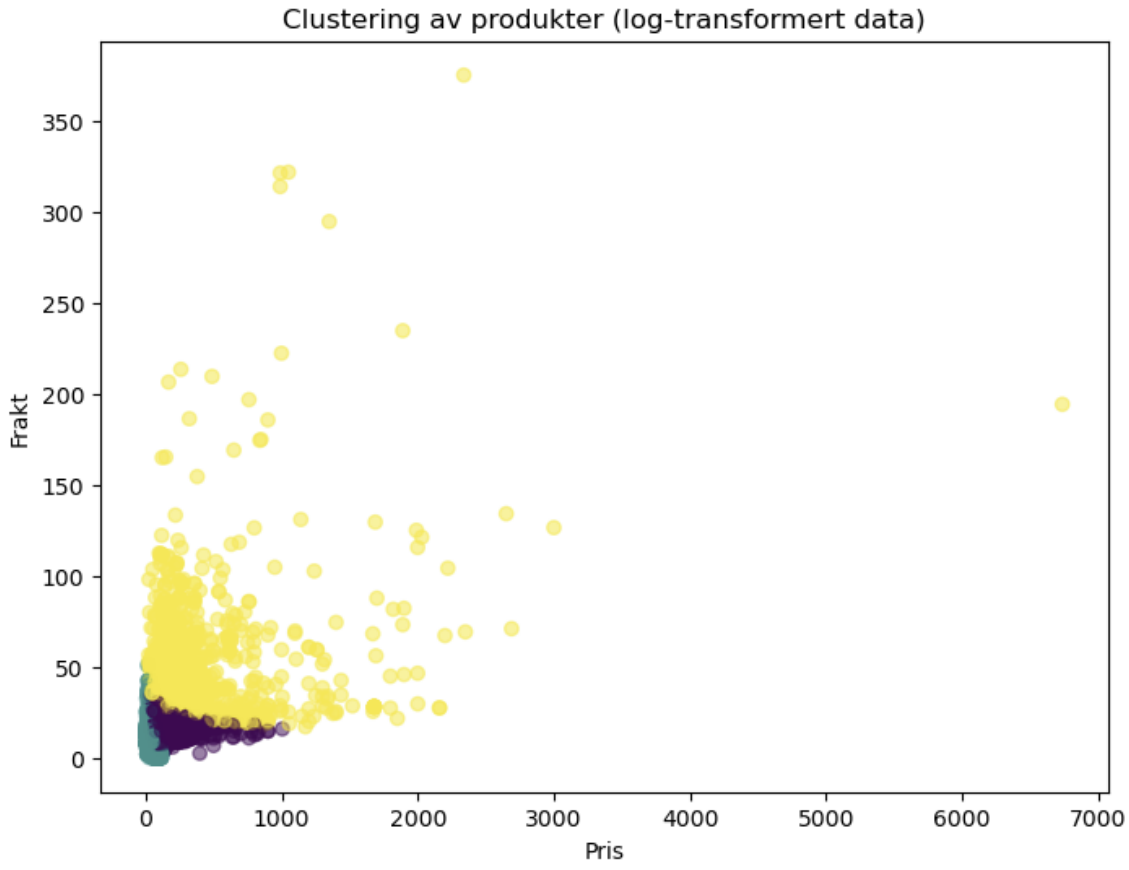
2.2.2 Predictive Analytics

The descriptive analytics provided an initial understanding of sales patterns, product categories, and customer distribution across Brazil. This stage revealed key characteristics of the dataset, including the variation in product prices, the impact of freight costs, and the dominance of regions such as Sao Paulo. These insights formed a baseline for deeper analysis.

To support predictive decision-making, a clustering analysis was conducted on products based on price and freight value. The aim was to segment products into meaningful groups to guide logistics, marketing and promotional strategies.

Numerical features price and freight_value were log-transformed to address skewed distributions and then standardized using StandardScaler to ensure equal weighting. A KMeans clustering algorithm was applied, specifying three clusters with multiple initializations (n_init=20) to ensure stable results.

```
cluster
0    4429
1    4120
2    1451
Name: count, dtype: int64
```



The clustering yielded the following product segments:

Cluster 0 (4,429 products): Low to minimum price and freight. These are small inexpensive products with steady sales volume.

Cluster 1 (4,120 products): Medium price and freight. Products in this cluster have moderate margins and sales volume

Cluster 2 (1,451 products): High price and high freight. These are expensive or large products and often require more logistic management.

A scatterplot visualizing the clusters confirmed the separation of products by price and freight. It highlights distinct groups with protentional operational implications.

Business insights

The clustering provides guidance for Olists operational and marketing strategies. Cluster 2 products should be prioritized for logistics optimization as high freight cost and prices require careful handling. Cluster 0 products represent high-volume, low margin items, where promotions and inventory strategies can focus on maximizing sales. Cluster 1 represents moderate value products and can be targeted with balanced marketing campaigns. Overall, the segmentation helps the company better allocate resources, optimize stock and shipping and design targeted promotions.

2.2.3 Prescriptive Analytics

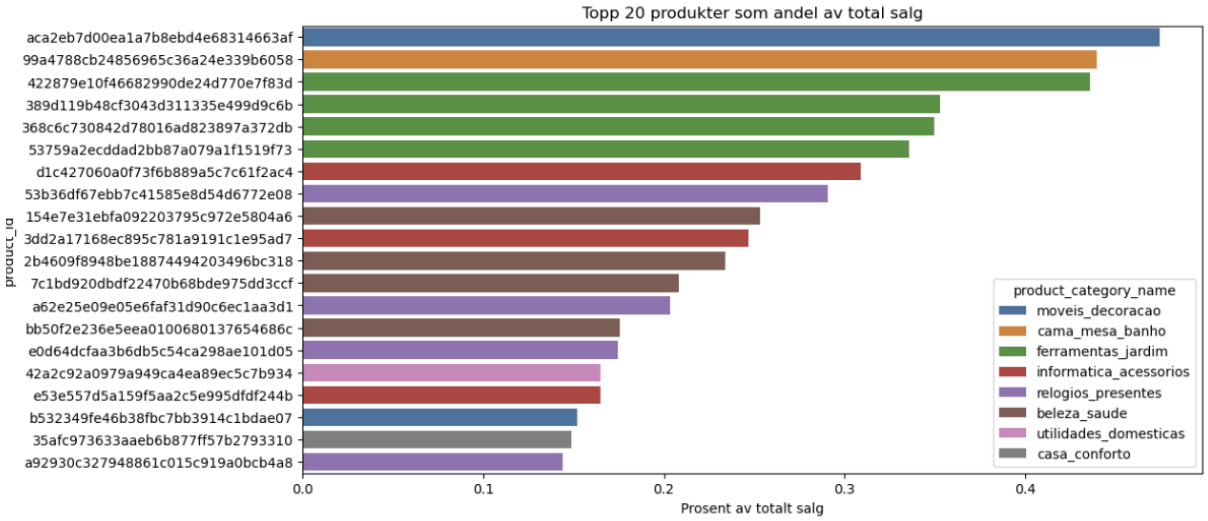
A prescriptive analysis was conducted to identify the top selling products and categories. Product sales were aggregated by category and product ID using SQL, and the top tree products within each category were highlighted. Additionally, the products contribution to total sales was calculated, with the twenty highest performing items visualized as a percentage of overall transactions.

The analysis revealed that certain products clearly dominate sales. For example, products within the *utilidades_domesticas* category achieved total sales 183, 73 and 63 units. That makes them among the most popular items. Also, the *alimentos* products reached 119 and 55 sales. These high performing products should be prioritized in inventory planning, marketing campaigns and promotional activities to maximize

revenue. The products that sold fewer items between 12 and 22, like *telefonia_fixca* and *agro_industria_e_comercio* are lower volume products with limited value. They could either be removed from the assortment or strategically promoted to increase demand.

	product_category_name	product_id	total_sales
0	agro_industria_e_comercio	11250b0d4b709fee92441c5f34122aed	22
1	agro_industria_e_comercio	423a6644f0aa529e8828ff1f91003690	18
2	agro_industria_e_comercio	672e757f331900b9deea127a2a7b79fd	17
74	alimentos	89321f94e35fc6d7903d36f74e351d40	119
75	alimentos	ed2067a9c1f79553088a3c67b99a9f97	55
...	...	...	...
29886	telefonia_fixa	b4f9530c931398e215242293c2c8ba4c	14
29887	telefonia_fixa	4633dfeb3a2588bdb52af32d504b44eb	12
30001	utilidades_domesticas	42a2c92a0979a949ca4ea89ec5c7b934	183
30002	utilidades_domesticas	5a6e53c3b4e8684b13388d6aa4afdf12	73
30003	utilidades_domesticas	f7a17d2c51d9df89a4f1711c4ac17f33	63

[216 rows x 3 columns]



Business insights

The findings suggest that Olist should focus resources on the sellers with the top selling products in high performing categories to ensure stock availability and targeted marketing. Optimization of logistics and inventory for high-demand items will improve efficiency and profitability. Low selling products should be assessed for potential removal or targeted campaign to stimulate interest. It's important to balance the product assortment for maximum business impact.

2.3 Summary of Business Findings

The SQL and Python analyses provide a comprehensive view of Olists sales, customers and products. They reveal key value revenue patterns like Black Friday sales, highlighting the importance of targeted promotions and inventory planning. Customer analytics indicated that the majority of customers are one-time buyers. The repeat purchase rate was only 3 percent. This suggests limited loyalty and opportunities for engagement programs. The analytics of silent customers further emphasizes the need for strategies to increase feedback and satisfaction tracking. CLV calculations identified the most profitable customers, guiding priority targeting for campaigns

Descriptive Python analytics showed that Sao Paulo accounts for over 40 percent of orders. The prices are widely distributed with a few expensive products skewing averages. Correlation analysis indicated a weak positive relationship between price and freight, suggesting logistics should be considered independently in pricing and marketing strategies.

Predictive KMeans clustering segmented products into three groups: low-price/low-freight (high-volume, low-margin), medium-price/medium-freight (moderate sales and margins), and high-price/high-freight (requiring careful logistics). This segmentation informs marketing, inventory, and logistics strategies.

Prescriptive recommendations highlighted the best-selling products and categories. These should be prioritized for stock, campaigns and promotions. The low selling products identified could be phased out or promoted strategically.

Overall, these findings allow Olist to optimize inventory and logistics, target marketing campaigns and promotions efficiently, and implement strategies to improve customer loyalty. The SQL and Python analysis provide a foundation for strategic decisions to enhance customer satisfaction, efficiency and profitability.

Task 3: Business Intelligence Dashboard

3.1 Dashboard Design and Structure

The objective of Task 3 was to design an interactive Power BI dashboard to support decision making. The design follows a multipage structure with five main pages: Executive summary, Operational Dashboard, Analytical Deep-Dive, What-if Analysis and Customer Analysis.

The Executive Summary presents high-level KPI such as Revenue, orders count, AOV, Freight and Repeat Purchase Rate. There are also visuals for Revenue/Time and Profit margin. A slicer allows users to select the desired time period, providing managers with a quick snapshot of business performance.

The operational Dashboard provides daily and weekly operational metrics, including Revenue by product category, revenue by seller, KPI cards for revenue, orders and profit margin, and a map to show the metrics on state level. Drill-downs were enabled to navigate from national to state- and product-level detail.

The Analytical Deep Dive Page Contains KPI cards for quick reference, a scatter chart analysing Freight vs. Revenue, Orders by Product Category, and a tree map showing revenue by product category. This page supports exploratory analysis of patterns and relationships.

The what-if Analysis enables scenario planning by adjusting discount levels. The impact is dynamically calculated on revenue and profit margins using DAX measures. It also has a bar chart to illustrate the impact discount have on the different product categories. A parameter slicer lets you set the discount to see the impact on the total sales.

Last in the customer Analytics page, we have included a lot of visual types. It includes bar charts for Customer Lifetime Value (CLV) by customer, a map showing CLV and Repeat Purchase Rate by state, a Q&A visual for natural language queries, a Key Influencers visual, and a Decomposition Tree for drill-down analysis. KPI cards highlight key customer metrics. This page enables segmentation and loyalty assessment.

The overall design of the pages flows and crates a recognisable structure throughout. The dashboard incorporated at least eight visualization types, including KPI cards, bar and column charts, line graphs, heat maps, scatter plots, decomposition trees, key influencer visuals, and tables. Interactive slicers for time, product category, and region allow users to filter dynamically. Cross-filtering was enabled across visuals, and responsive layouts were applied to ensure usability on mobile devices.

3.2 Advanced DAX Calculations

As part of the Power BI dashboard development, several advanced DAX (Data Analysis Expressions) measures and calculated columns were created to provide advanced analytical capabilities. These calculations support both operational monitoring and advanced insights and complement the SQL and Python analyses presented in part 2.

Total Revenue

```
Revenue = SUM(fact_order_sales[total_item_value])
```

This measure aggregates all sales values and forms the baseline for most KPIs. It is used across multiple visuals, including cards, bar charts, and time-series line graphs.

Customer Lifetime Value (CLV)

```
CLV = SUM('fact_orders'[total_amount]) - SUM('fact_orders'[freight_value])
```

This measure replicates the SQL profitability analysis. It highlights high-value customers by accounting for freight costs.

Repeat Purchase rate

```
Repeat Purchase Rate =
VAR CustomersWithMultipleOrders =
  CALCULATE(
    DISTINCTCOUNT('fact_orders'[customer_key]),
    FILTER(
      VALUES('fact_orders'[customer_key]),
      CALCULATE(DISTINCTCOUNT('fact_orders'[order_id])) > 1
    )
  )
VAR TotalCustomers =
  DISTINCTCOUNT('fact_orders'[customer_key])
RETURN
DIVIDE(CustomersWithMultipleOrders, TotalCustomers, 0)
```

This metric calculates the percentage of customers who have made more than one purchase. It provides insights into customer loyalty, complementing the SQL analysis.

Average Order Value

```
AOV = DIVIDE ( [Revenue], [Orders], 0 )
```

Measures the mean value of customer orders, providing insight into sales performance and customer behavior.

Year-over-Year Revenue

```
Revenue YoY =
VAR Prev = CALCULATE ( [Revenue], DATEADD ( dim_date[full_date], -1, YEAR ) )
RETURN DIVIDE ( [Revenue] - Prev, Prev, 0 )
```

This implements time intelligence to track performance across years.

These measures were integrated into cards, trend lines, tables, and decomposition trees. YoY and moving averages were primarily shown on the Executive Summary and Analytical Deep-Dive pages. CLV and Repeat Purchase Rate were highlighted in the Customer Analytics Dashboard to support customer segmentation and loyalty assessment.

AOV was included in the Executive Summary Page to provide a quick snapshot of order performance. What-if parameters were created for discount percentage changes, with dynamic DAX measures recalculating revenue and margin impact. AI Visuals like The Key Influencers visual was used to identify attributes driving high sales, while the Decomposition Tree helped break down revenue by region, product, and customer.

3.3 User Guide for Dashboard Navigation

The Power BI dashboard has been designed to support both strategic and operational decision-making. The dashboard was designed for ease of use with clear navigation and interactive exploration.

Page Navigation

Use the bottom tab navigation to switch between pages: Executive Summary, Operational Dashboard, Analytical Deep-Dive, What-if Analysis, and Customer Analytics.

Slicers and Filters

Each page includes slicers for time, product category, discount and region. Selecting values dynamically updates all visuals.

Cross-Filtering

Clicking on a chart element filters the other visuals automatically. This enables multi-dimensional exploration.

Drill-Down

Many visuals like bar the charts and maps support drill-down. Click the arrow icons to move to the details.

What-if Scenario

On the What-if page, adjust the discount slider to simulate impact. Revenue and margin visuals update in real-time.

Customer Analysis Tools

Use the Q&A visual to type natural-language questions like “show revenue by region”. The Key Influencers visual highlights key drivers of sales, while the Decomposition Tree allows stepwise exploration.

Export Options

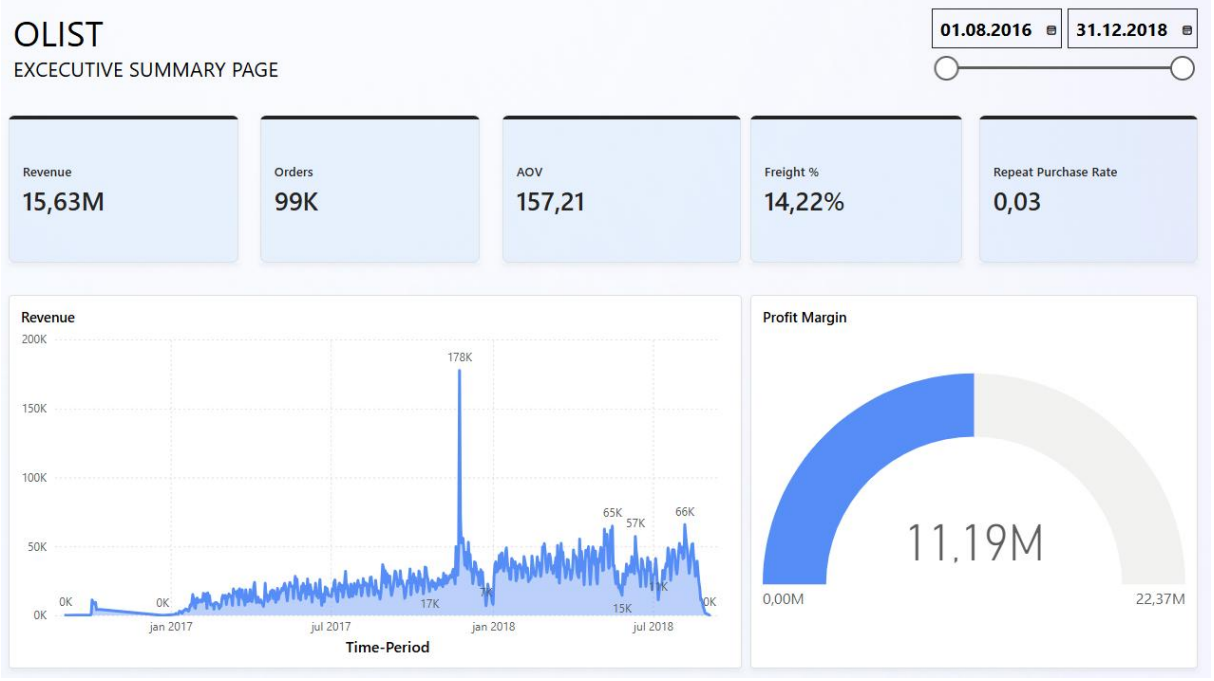
Right-click on any visual to export data to Excel or CSV for further analysis.

3.4 Dashboard Screenshots

To illustrate the design and functionality of the Power BI dashboard, screenshots have been included below.

Executive Summary Page

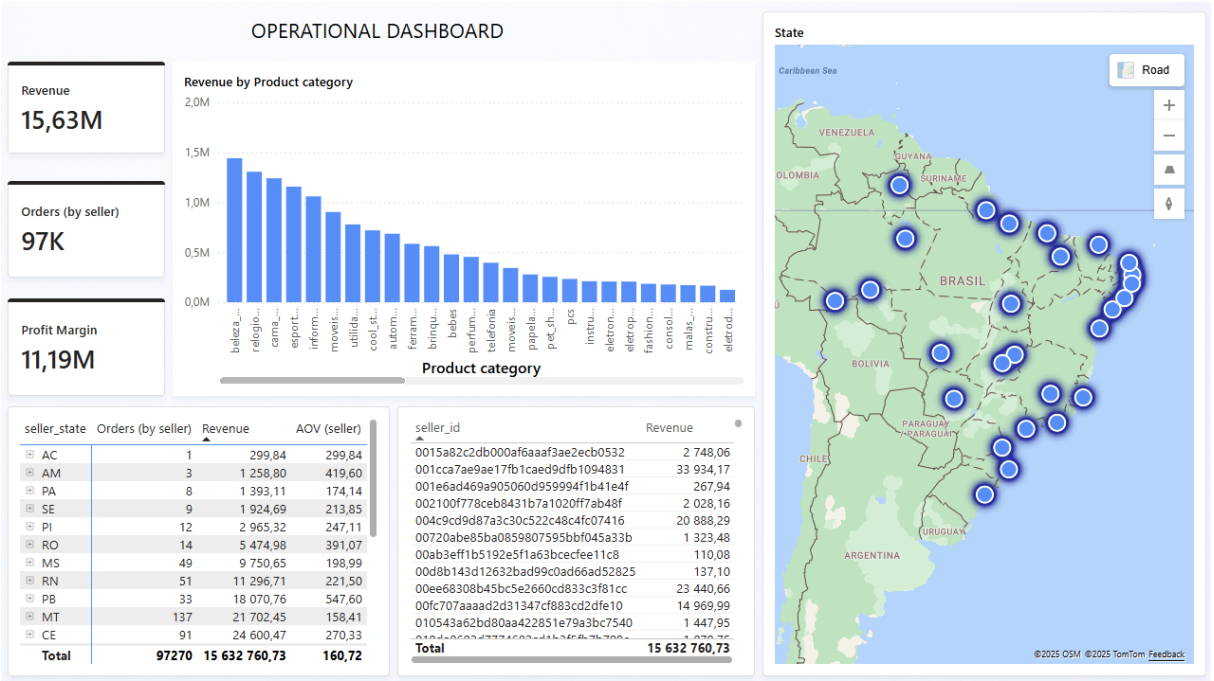
This page presents high-level KPIs including Total Revenue, Order Volume, AOV, Freight and Repeat Purchase Rate. Trend lines show year-over-year growth, while cards and gauge visuals highlight profit margin.



Screenshot 1: Executive Summary Page

Operational Dashboard

This page provides a detailed breakdown of sales and customer distribution by product category, state, and time period. Bar charts, maps, and drill-down line graphs allow managers to identify bottlenecks in sales.



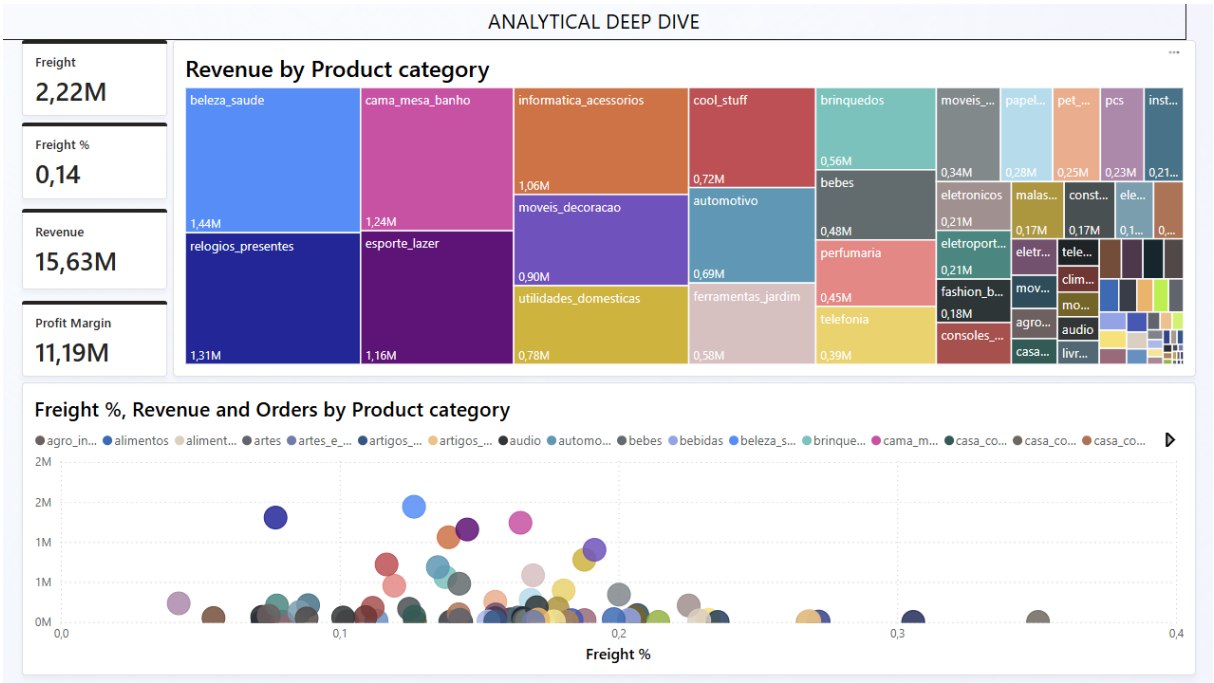
seller_state	Orders (by seller)	Revenue	AOV (seller)
AC	1	299,84	299,84
AM	3	1 258,80	419,60
PA	8	1 393,11	174,14
SE	9	1 924,69	213,85
PI	12	2 965,32	247,11
RO	14	5 474,98	391,07
MS	49	9 750,65	198,99
RN	51	11 296,71	221,50
PB	33	18 070,76	547,60
MT	137	21 702,45	158,41
CE	91	24 600,47	270,33
Total	**97270**	**15 632 760,73**	**160,72**

seller_id	Revenue
0015a82c2db000af6aaaf3ae2ecb0532	2 748,06
001cca7ae9ae17fb1caed9dfb1094831	33 934,17
001e6ad469a905060d959994f1b41e4f	267,94
002100f778ceb8431b7a1020ff7ab48f	2 028,16
004c9cd9d87a3c30c522c48c4fc07416	20 888,29
00720abe85ba0859807595bbf045a33b	1 323,48
00ab3eff1b5192e5f1a63bcecfee11c8	110,08
00d8b143d12632bad99c0ad66ad52825	137,10
00ee68308b45bc5e2660cd833c3f81cc	23 440,66
00fc707aaaad2d31347cf883cd2dfe10	14 969,99
010543a62bd80aa422851e79a3bc7540	1 447,95
Total	**15 632 760,73**

Screenshot 2: Operational Dashboard

Analytical Deep-Dive

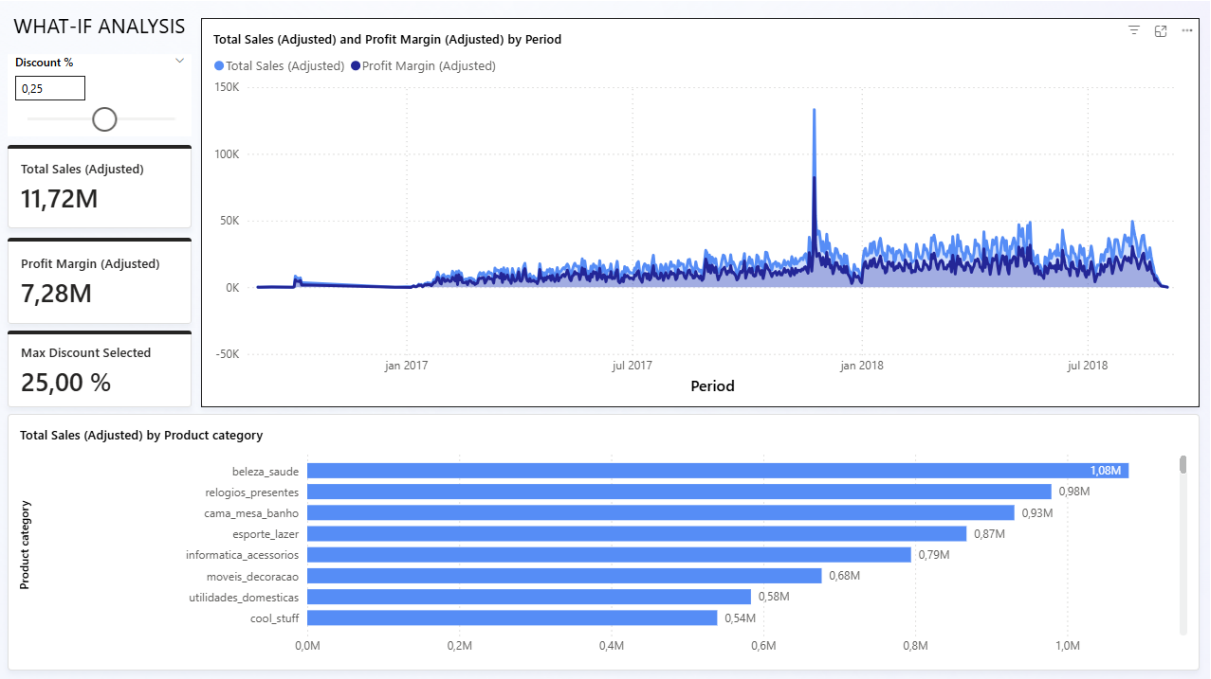
This page supports advanced analysis. A scatter chart shows the relationship between Freight and Revenue, a treemap highlights revenue distribution by category, and KPI cards summarize key performance indicators.



Screenshot 3: Analytical Deep Dive

What-if Analysis Page

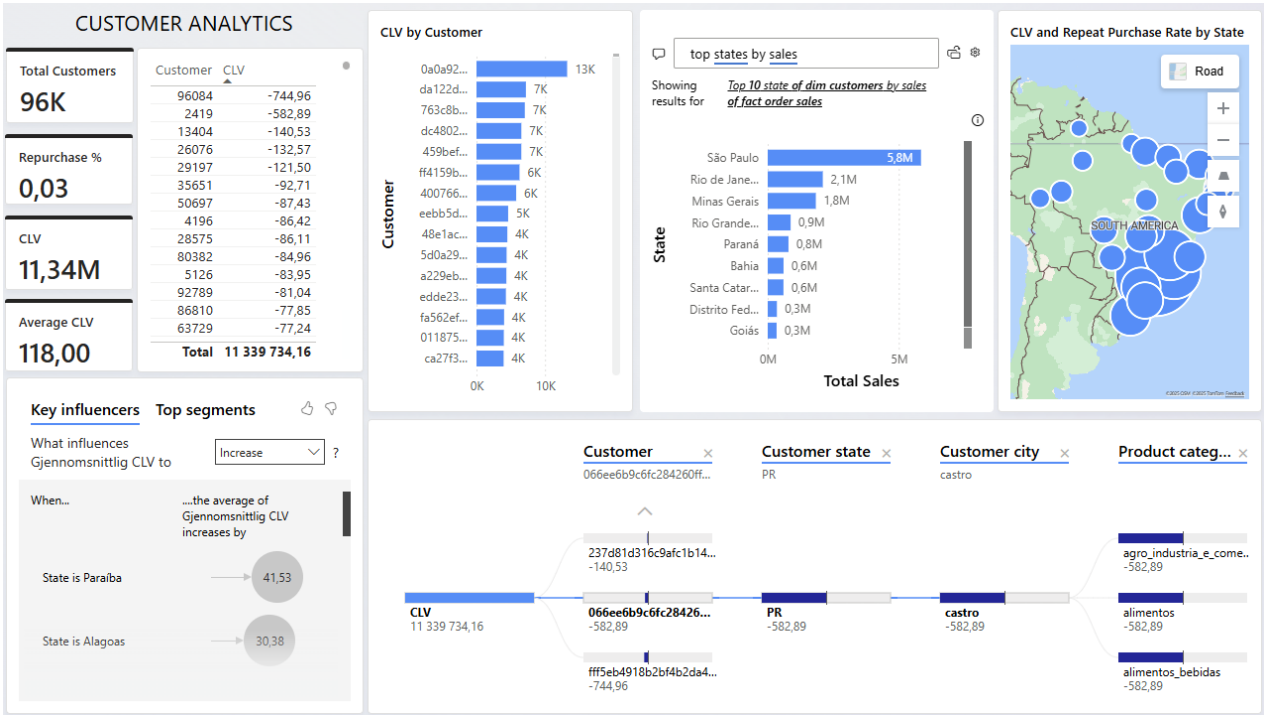
This page enables interactive simulations using parameter slider for applying discount. Changes dynamically update all metrics to support decision-making.



Screenshot 4: What-if Analysis Page

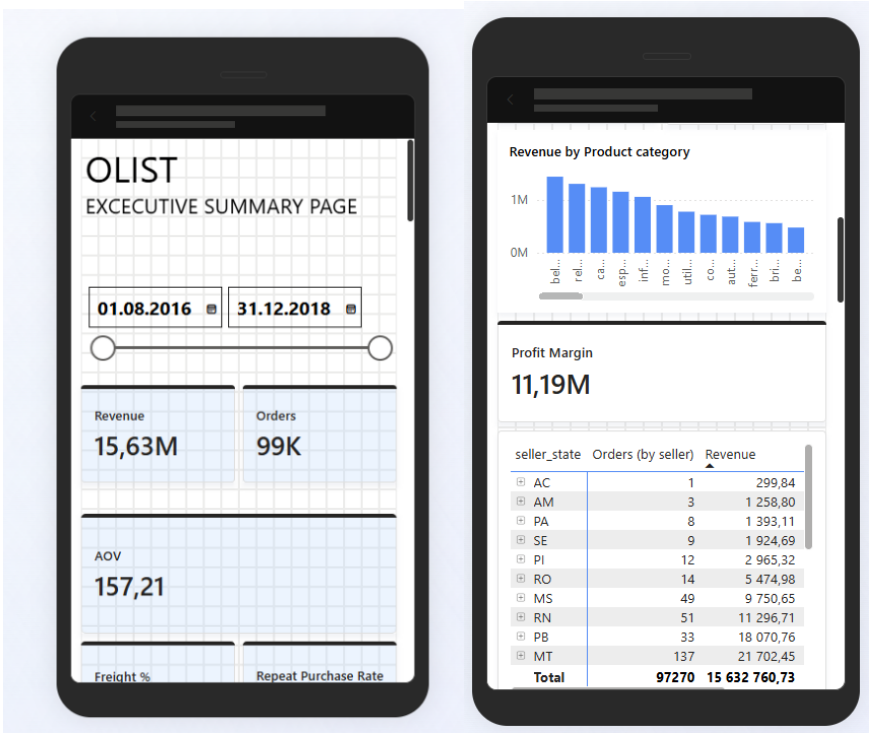
Customer Analytics

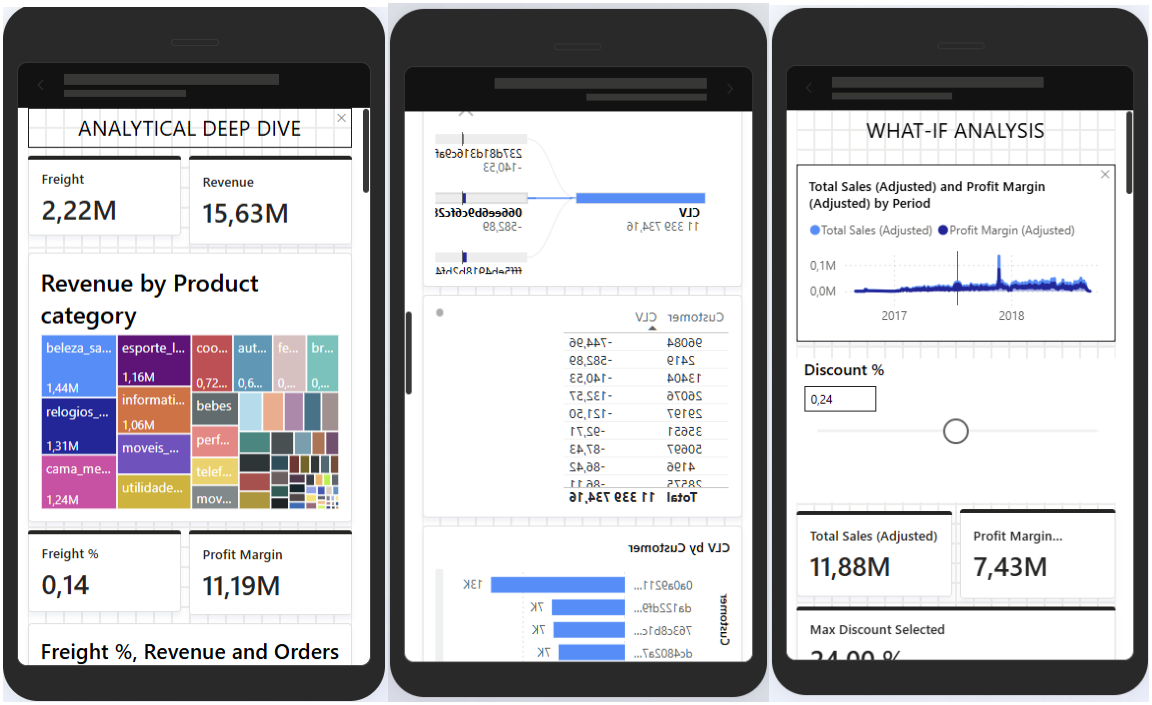
This page focuses on customer value and loyalty. It includes CLV by customer, CLV by state (map), repeat purchase rate, Q&A for data exploration, a Key Influencers visual, and a Decomposition Tree for segmentation. KPI cards show customer metrics immediately.



Screenshot 5: Customer Analytics

Mobile responsive design





3.5 Business Interpretation of Dashboard Findings

The Power BI Olist dashboard provides actionable insights across revenue, customer behaviour, and product performance. The Executive Summary allows managers to quickly monitor overall performance, identifying trends in revenue, AOV, and repeat purchase rate. High-performing periods and regions are immediately visible, supporting resource allocation decisions. The Operational Dashboard highlights state-level sales and product-category performance. Drill-down functionality reveals bottlenecks or underperforming segments, enabling targeted marketing or inventory adjustments.

The Analytical Deep-Dive uncovers relationships between freight costs and revenue and identifies categories contributing most to sales. This allows optimization of logistics and promotional strategies.

The What-if Analysis supports scenario planning, quantifying the impact of discount strategies on revenue and margins, helping managers make data-driven pricing decisions. The Customer Analytics page identifies high-value customers and regions, supporting loyalty programs and retention campaigns. Key Influencers and the Decomposition Tree highlight factors driving revenue, allowing strategic interventions.

Overall, the dashboard integrates SQL and Python analysis results from task 2, transforming data into valuable insights for operational and strategic decision-making. It provides a single view for executives and analysts to monitor, explore, and optimize business performance.

Contribution Statement / Work Distribution

Candidate nr: 7016; Database model dim_product, dim_customer, fact_order_sales, fact_review, dim_date [GoogleAI]. Penthao(ETL), analytic query 1.1 - 3.1, PowerBI Dashboard, Final report.

Candidate nr: 7020, Database model dim_payment_type, dim_seller, dim_order_status, fact_order, fact_payment. analytic query 3.2 - 5.2, python-analysis, PowerBI dashboard, user guide, data_dictonary, final report.

References

Sharda, R., Delen, D., & Turban, E. (2024). *Business intelligence, analytics, data science, and AI: A managerial perspective* (5th ed.). Pearson.

OpenAI. (2025). ChatGPT (August 2025 version) [Large language model]. https://chat.openai.com/

GoogleAI. *Date Dimension in PostgreSQL using* `generate_series` (kodeforslag generert med Google AI).

Appendix

Python Analytics

Anaconda Assistant. (2023). AI-assistanseverktøy for Python-programmering [Programvare]. Anaconda Inc.

pandas: For dataanalyse og -håndtering - **matplotlib** og **seaborn**: For datavisualisering - **numpy**: For numeriske beregninger - **jupyter**: For interaktiv kode og dokumentasjon - **nbconvert**: For å konvertere Jupyter notebooks til andre formater (PDF, HTML) - **python-docx**: For å generere Word-dokumenter fra Python# Descriptive Analytics: Statistical summary and correlation analysis

```
import seaborn as sns

import matplotlib.pyplot as plt


#trekk ut sales + product + customer

query = """

SELECT f.price, f.freight_value, f.quantity,

    d.product_category_name, c.customer_state

FROM fact_order_sales f

JOIN dim_products d ON f.product_key = d.product_key

JOIN dim_customer c ON f.customer_key = c.customer_key

LIMIT 10000;

"""

sales = pd.read_sql(query, engine)


# Statistisk sammendrag

display(sales.describe(include="all"))


print(sales['product_category_name'].value_counts().head(10))

print(sales['customer_state'].value_counts())
```

```
# Korrelasjon (numeriske variabler)
corr_matrix = sales.corr(numeric_only=True)
print(corr_matrix)

sns.scatterplot(x='price', y='freight_value', data=sales)
plt.title("Sammenheng mellom pris og frakt")
plt.show()

#Predictive Analytics med Clustering (KMeans)

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

# Velg numeriske kolonner
data = sales[['price','freight_value']]

# Log-transformasjon for å håndtere skjev fordeling
data_log = np.log1p(data)  # log(1+x) for å unngå log(0)

# Skaler dataene
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_log)

# KMeans clustering med 3 cluster og flere init
kmeans = KMeans(n_clusters=3, random_state=42, n_init=20)
sales['cluster'] = kmeans.fit_predict(data_scaled)
```

```
# Sjekk antall produkter per cluster
print(sales['cluster'].value_counts())

# Visualisering
plt.figure(figsize=(8,6))
plt.scatter(sales['price'], sales['freight_value'], c=sales['cluster'], cmap='viridis', alpha=0.5)
plt.xlabel("Pris")
plt.ylabel("Frakt")
plt.title("Clustering av produkter (log-transformert data)")
plt.show()

# Descriptive + Predictive analytics
sales.to_csv("sales_analysis.csv", index=False)

#Prescriptive Analytics: Optimization or recommendation algorithm

recommendation_query = """
SELECT d.product_category_name, d.product_id, COUNT(*) AS total_sales
FROM fact_order_sales f
JOIN dim_products d ON f.product_key = d.product_key
GROUP BY d.product_category_name, d.product_id
ORDER BY d.product_category_name, total_sales DESC;
"""
recs = pd.read_sql(recommendation_query, engine)

# Gi topp 3 anbefalinger per kategori
```

```
top_recs = recs.groupby("product_category_name").head(3)
print(top_recs)
# Beregn prosentandel av total salg
recs['sales_pct'] = recs['total_sales'] / recs['total_sales'].sum() * 100

# Topp 20 produkter etter prosentandel
top_products_pct = recs.sort_values('sales_pct', ascending=False).head(20)

# Plot
plt.figure(figsize=(12,6))
sns.barplot(
   data=top_products_pct,
   x='sales_pct',
   y='product_id',
   hue='product_category_name',
   dodge=False
)
plt.title("Topp 20 produkter som andel av total salg")
plt.xlabel("Prosent av totalt salg")
plt.show()

top_products =
recs.groupby(['product_category_name','product_id'])['total_sales'].sum().reset_index()
top_products['sales_pct'] = top_products['total_sales'] /
top_products['total_sales'].sum() * 100

# Eksporter til CSV for Power BI
top_products.to_csv("top_products.csv", index=False)
```