Data Science and AI (Full time, Batch 1)

Module 2 - Assignment Project
E-commerce Data pipeline and Analysis

What is this project?

Our aim is to be able to show understanding of the techniques and understandings acquired during the module building upon a real world dataset.

The project has been created by:

Ian
Siew Wen
Eve
Selena
Andrew

It's hosted on Github at: https://github.com/achuakh/Module-2-Project

Executive Summary

This project tests our understanding of orchestrating data from the source to the data warehouse, performing transformation, validation and analysis by plotting relevant charts and written summaries.

Using a period sample of a real world dataset, it allows for exploration and understanding of issues in data integrity and various resolutions methods of data cleaning. The data set is not overly large but provides enough entries allowing for data transformation and for analysis and trend discovery.

About the dataset

The provided dataset of information, from Olist, contains 100k orders from 2016 to 2018 made at multiple marketplaces in Brazil. Its features allows for viewing an order from different perspectives; from order status, price, payment and freight performance to customer location, product attributes and finally reviews written by customers. Additional geolocation information in the dataset of Brazilian zip codes to lat/long coordinates derived from point of order.

Data Ingestion

Acquisition of the data into a pipeline which allows for repeatability and scalability going forward. Although the data is static for the moment, we felt that building a robust pipeline will allow for future development.

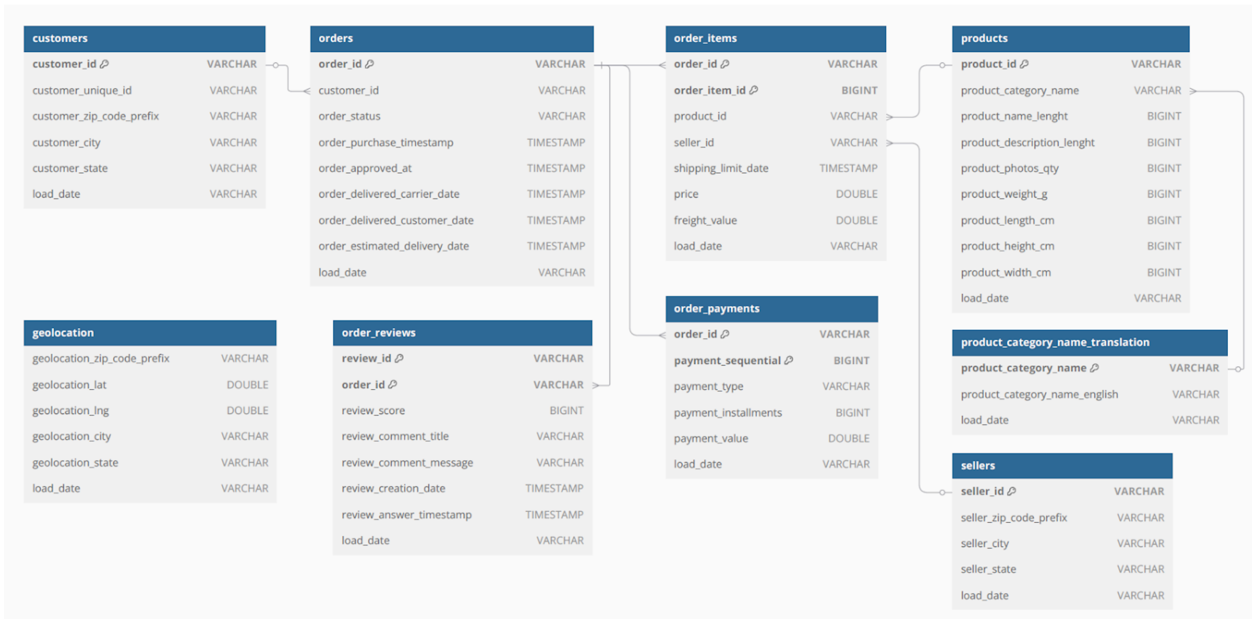
Extracting the data from Kaggle

The original data set is read from Kaggle via the Kaggle API to the local host. This download will provide the 9 original CSV files as follows:

1. olist_customers_dataset.csv
2. olist_geolocation_dataset.csv
3. olist_order_items_dataset.csv
4. olist_order_payments_dataset.csv
5. olist_order_reviews_dataset.csv
6. olist_orders_dataset.csv
7. olist_products_dataset.csv
8. olist_sellers_dataset.csv
9. product_category_name_translation.csv

Making Sense of the Relationship

The data in its original form carries the ERD as shown below.



Data Warehouse Implementation

The core of acquiring the data is ensuring that it meets the business requirement. It must be optimised to be presented logically while crucial information remains unaltered. Data security and scalability needs to be considered.

Schema Design

The star schema was predetermined by the project requirements. It offers advantages of simplicity, faster query performance, and easier understanding for analytical tasks, but it also has disadvantages such as data redundancy, potential data quality issues, and challenges in maintaining and updating.

After evaluation, an expanded version of the star schema, Galaxy Schema Design was chosen for the project for optimisation reasons explained thereafter.

Galaxy Schema Design: when one star isn’t enough

Clear Separation of Subject Areas

By splitting the data into facts and dimension tables, we isolate measurable business events (orders, payments, reviews) from descriptive context (customers, dates, geographies). This modular approach:

- Keeps each table focused and lightweight
- Enables efficient indexing and filtering
- Makes querying more readable and maintainable

Multiple Fact Tables for Granular Control

Each fact table - *FCT_ORDER_ITEMS, FCT_PAYMENTS, FCT_REVIEWS* - tracks a different event type with different grain:

- *FCT_ORDER_ITEMS*: Line-item level
- *FCT_PAYMENTS*: One row per payment attempt per order
- *FCT_REVIEWS*: One row per review per order

This avoids wide, bloated tables and lets analysts choose just the data they need - improving query performance and flexibility.

Reusable and Conformed Dimensions

The *DIM_ORDER* and *DIM_CUSTOMERS* tables are shared across multiple fact tables, ensuring consistency in joins and filters across time-based or customer-centric analyses. Using surrogate keys like pk_date_sid allows:

- Efficient join performance via integers/strings
- Separation of actual timestamps from business logic
- Simplified time intelligence (e.g., calculating year-to-date sales)

Pre-Aggregated & Derived Metrics

The *DIM_ORDERS* table includes summary columns like *balance_amt* and *total_order_amt_wf_freight*. Precomputing these:

- Reduces repeated aggregations in queries
- Accelerates dashboard rendering and ETL

It's especially helpful for business users who want fast insights without complex SQL.

Join Paths Are Clear and Indexed

Every relationship is well-documented:

- One-to-many between *DIM_ORDERS* and each fact
- Many-to-one joins from facts to shared dimensions This design ensures minimal data duplication and logical integrity, and you avoid heavy Cartesian products during joins.

Best Practices Used in the Data Model

Constellation Schema Adoption:

Separated descriptive dimensions (e.g., *DIM_CUSTOMERS, DIM_DATE, DIM_ORDERS*) from fact tables (e.g., *FCT_ORDER_ITEMS, FCT_PAYMENTS, FCT_REVIEWS*), ensuring clarity, scalability, and analytical performance.

Date Dimension Usage:

DIM_DATE supports multiple foreign keys across order lifecycle events, enabling robust time-series and time-lag analyses—*a hallmark of mature data warehousing*.

Load Date Tracking:

Each table includes a load_date column, ensuring data auditability and enabling incremental load strategies in tools like dbt.

Reference Lookup Table:

LKP_STATUS_DESC abstracts status logic away from the fact/dim tables, allowing easy enhancement.

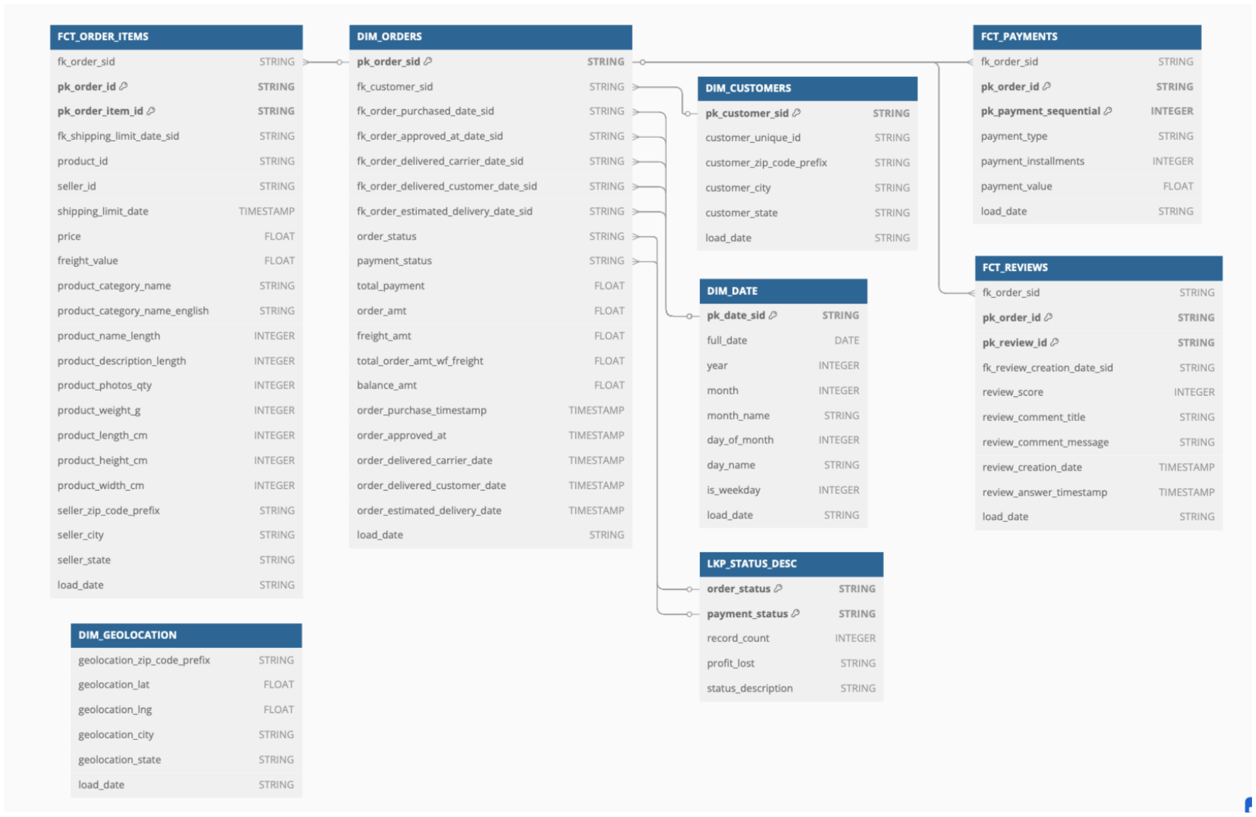
Granularity is Explicit:

Fact tables (*FCT_ORDER_ITEMS, FCT_REVIEWS, FCT_PAYMENTS*) are built around clearly defined grains, allowing precise aggregation logic.

Pre-Aggregated & Derived Metrics

The *DIM_ORDERS* table includes summary columns like balance_amt and payment_status. Precomputing these:

- Reduces repeated aggregations in queries
- Accelerates dashboard rendering and ETL
- Get fast insights without complex SQL.



FCT_ORDER_ITEMS	
fk_order_sid	STRING
pk_order_id	**STRING**
pk_order_item_id	**STRING**
fk_shipping_limit_date_sid	STRING
product_id	STRING
seller_id	STRING
shipping_limit_date	TIMESTAMP
price	FLOAT
freight_value	FLOAT
product_category_name	STRING
product_category_name_english	STRING
product_name_length	INTEGER
product_description_length	INTEGER
product_photos_qty	INTEGER
product_weight_g	INTEGER
product_length_cm	INTEGER
product_height_cm	INTEGER
product_width_cm	INTEGER
seller_zip_code_prefix	STRING
seller_city	STRING
seller_state	STRING
load_date	STRING

DIM_ORDERS	
pk_order_sid	**STRING**
fk_customer_sid	STRING
fk_order_purchased_date_sid	STRING
fk_order_approved_at_date_sid	STRING
fk_order_delivered_carrier_date_sid	STRING
fk_order_delivered_customer_date_sid	STRING
fk_order_estimated_delivery_date_sid	STRING
order_status	STRING
payment_status	STRING
total_payment	FLOAT
order_amt	FLOAT
freight_amt	FLOAT
total_order_amt_wf_freight	FLOAT
balance_amt	FLOAT
order_purchase_timestamp	TIMESTAMP
order_approved_at	TIMESTAMP
order_delivered_carrier_date	TIMESTAMP
order_delivered_customer_date	TIMESTAMP
order_estimated_delivery_date	TIMESTAMP
load_date	STRING

DIM_CUSTOMERS	
pk_customer_sid	**STRING**
customer_unique_id	STRING
customer_zip_code_prefix	STRING
customer_city	STRING
customer_state	STRING
load_date	STRING

FCT_PAYMENTS	
fk_order_sid	STRING
pk_order_id	**STRING**
pk_payment_sequential	**INTEGER**
payment_type	STRING
payment_installments	INTEGER
payment_value	FLOAT
load_date	STRING

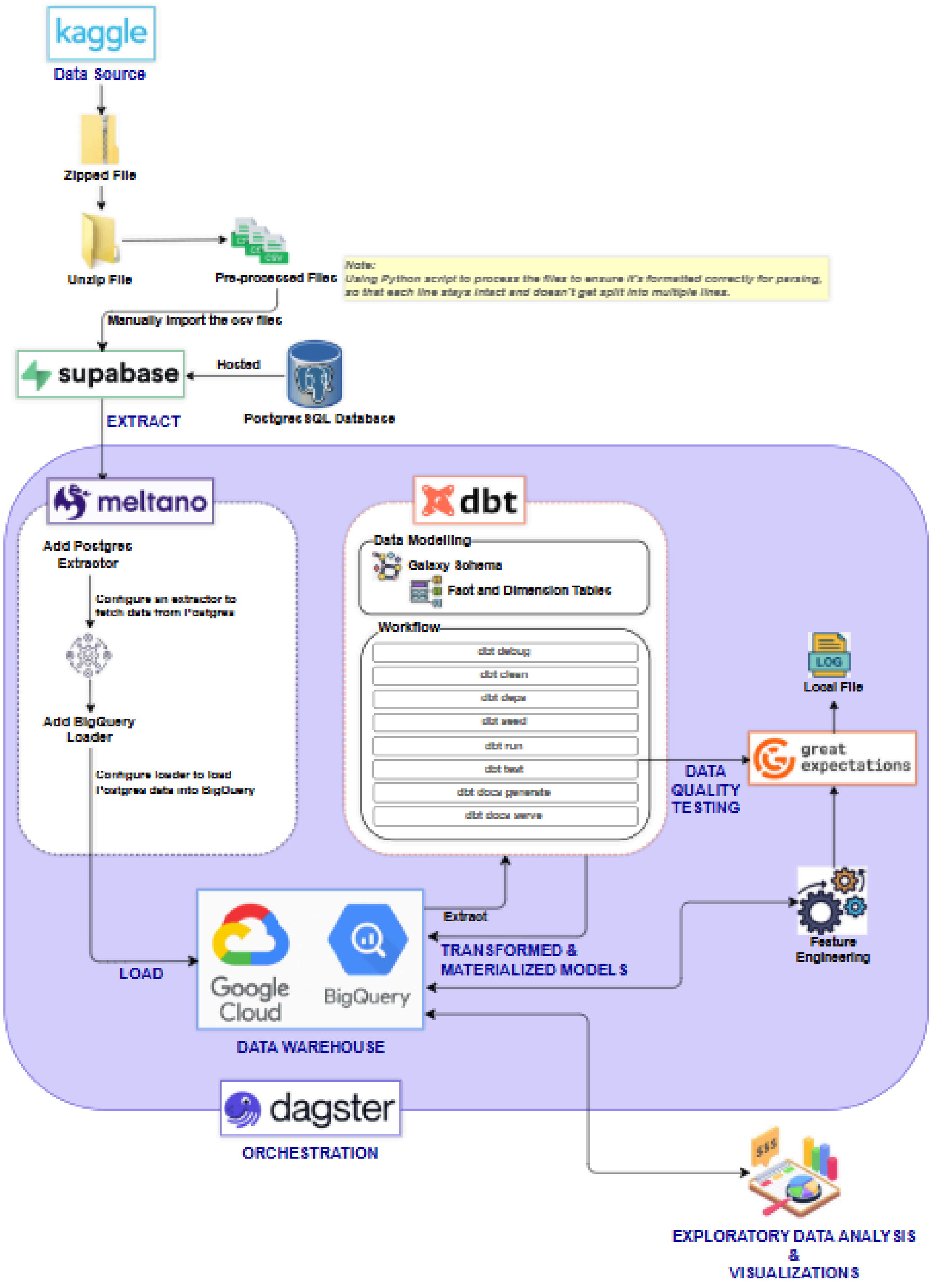
FCT_REVIEWS	
fk_order_sid	STRING
pk_order_id	**STRING**
pk_review_id	**STRING**
fk_review_creation_date_sid	STRING
review_score	INTEGER
review_comment_title	STRING
review_comment_message	STRING
review_creation_date	TIMESTAMP
review_answer_timestamp	TIMESTAMP
load_date	STRING

DIM_DATE	
pk_date_sid	**STRING**
full_date	DATE
year	INTEGER
month	INTEGER
month_name	STRING
day_of_month	INTEGER
day_name	STRING
is_weekday	INTEGER
load_date	STRING

LKP_STATUS_DESC	
order_status	**STRING**
payment_status	**STRING**
record_count	INTEGER
profit_lost	STRING
status_description	STRING

DIM_GEOLOCATION	
geolocation_zip_code_prefix	STRING
geolocation_lat	FLOAT
geolocation_lng	FLOAT
geolocation_city	STRING
geolocation_state	STRING
load_date	STRING

ELT Pipeline



Note:
Using Python script to process the files to ensure it's formatted correctly for parsing, so that each line stays intact and doesn't get split into multiple lines.

Data Migration Process

Local processing (raw data) - Data cleaning

The local CSV files are prepared by checking for anomalies in formatting that could interfere with data parsing. No business data is changed or discarded at this stage. Python script is used to detect and correct these formatting issues. Once this step is complete, the corrected DataFrame is re-exported to CSV and is ready to be loaded into the staging database.

Supabase (Postgres)

Holding tables are created in Supabase for the incoming CSV files. This allows the CSV data to be loaded into these tables, with column data types automatically detected and assigned based on the schema.

This approach also supports future scalability. From a business perspective, it enables different departments to upload data (e.g., from external systems) from various locations over time, without the need for manual file consolidation before loading.

Although this introduces an additional step, it provides a useful layer of isolation from the centralized, long-term storage in Google BigQuery, which serves as the central data warehouse.

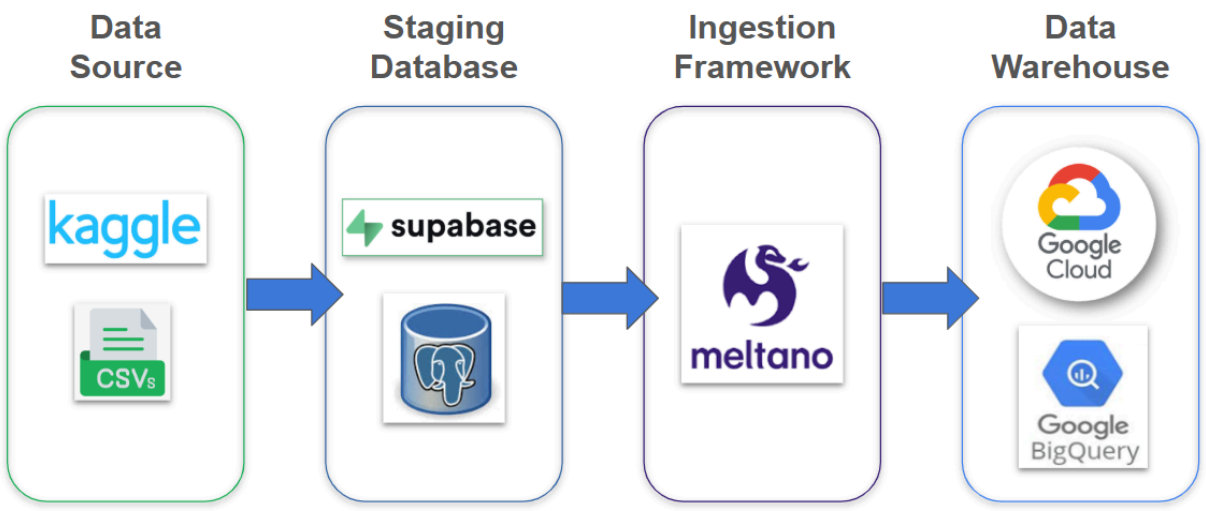
Meltano

As part of the Extract and Load (EL) phase of the data pipeline, Meltano is used to move data from the Postgres Database hosted on Supabase into Google BigQuery. A dataset is created in BigQuery (US region) to house the incoming data. The pipeline is developed and executed within a dedicated Conda environment to ensure consistency across different environments.

A new Meltano project is initialized, a Postgres extractor is added and configured to connect to the source database. A BigQuery loader is also added and set up for writing to the target dataset. The extractor and loader configurations define the tables to be read, connection details and authentication settings.

The full extract-to-load pipeline is executed using *meltano run*, which handles the data transfer end-to-end. The successful completion of the pipeline is verified by checking the ingested data in BigQuery.

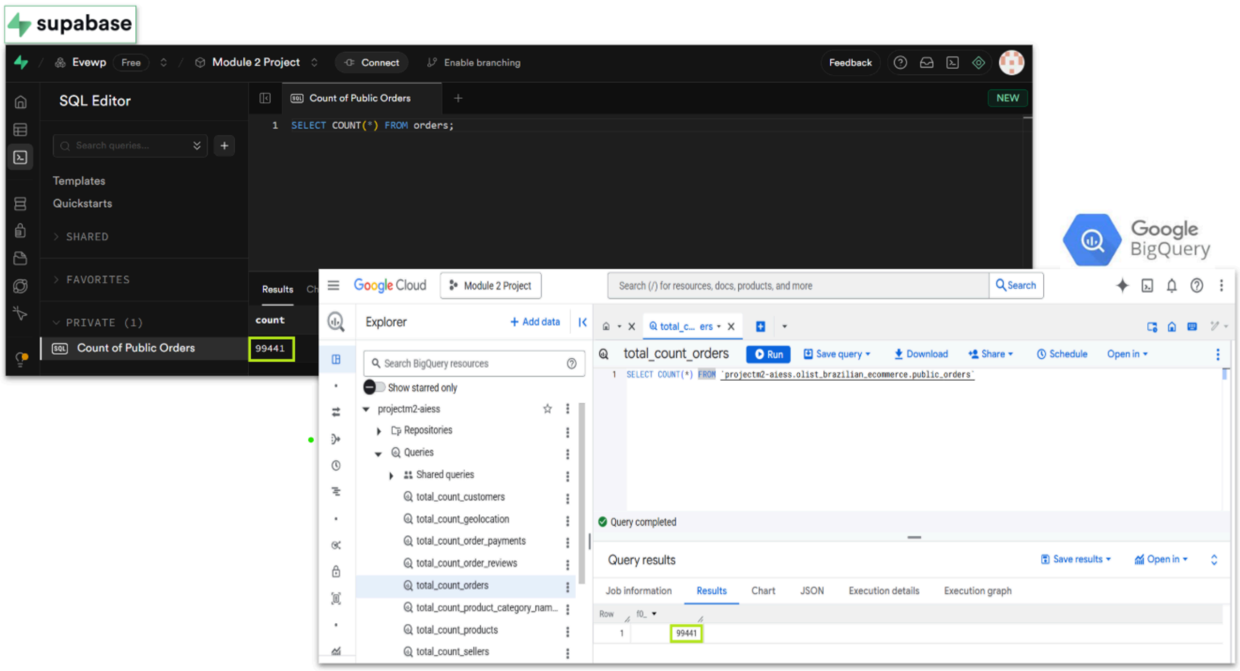
This setup enables repeatable and automated ingestion from departmental systems into centralized storage, forming the foundation for future scalability and transformation layers.



BigQuery (GCP)

After the raw data arrives from the import stage, a simple verification of row counting is performed by running SQL commands to GCP. This is a manual verification check against the arriving data at the development stage. This imported data is written into the dataset of *olist_brazillian_ecommerce.* Nothing has been transformed nor dropped at this point.

Verified that Supabase and BigQuery records matched after the initial data load



```
SELECT COUNT(*) FROM orders;
```

count
99441

```
SELECT COUNT(*) FROM `projectm2-aiess.olist_brazilian_ecommerce.public_orders`
```

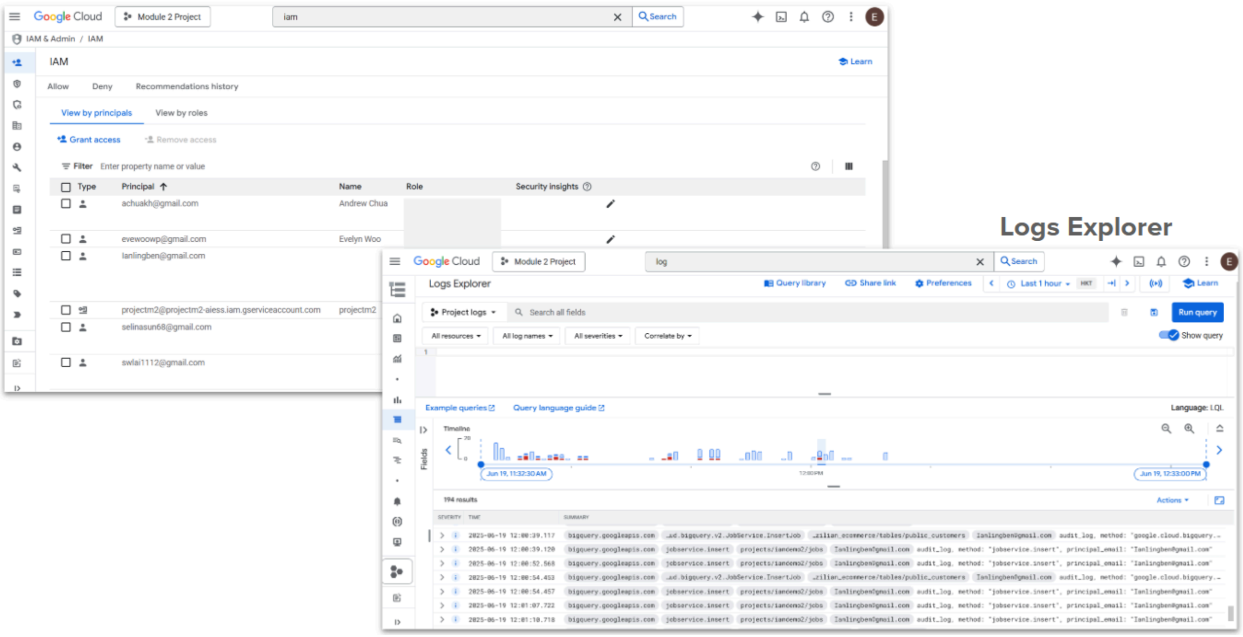
Query completed

Row	f0_
1	99441

Permissions and Logs Monitoring

User Access

Logs Explorer



Type	Principal	Name	Role	Security insights
	achuakh@gmail.com	Andrew Chua		
	evewoowp@gmail.com	Evelyn Woo		
	lanlingben@gmail.com			
	projectm2@projectm2-aiess.iam.gserviceaccount.com	projectm2		
	selinasun68@gmail.com			
	swlai1112@gmail.com			

194 results

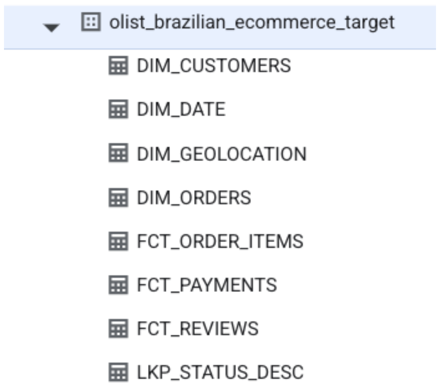
```
2025-06-19 12:00:39.117  bigquery.googleapis.com  …ud.bigquery.v2.JobService.InsertJob  …zilian_ecommerce/tables/public_customers  Ianlingben@gmail.com  audit_log, method: "google.cloud.bigquery.…
2025-06-19 12:00:39.120  bigquery.googleapis.com  jobservice.insert  projects/iandemo2/jobs  Ianlingben@gmail.com  audit_log, method: "jobservice.insert", principal_email: "Ianlingben@gmail.com"
2025-06-19 12:00:52.568  bigquery.googleapis.com  jobservice.insert  projects/iandemo2/jobs  Ianlingben@gmail.com  audit_log, method: "jobservice.insert", principal_email: "Ianlingben@gmail.com"
2025-06-19 12:00:54.453  bigquery.googleapis.com  …ud.bigquery.v2.JobService.InsertJob  …zilian_ecommerce/tables/public_customers  Ianlingben@gmail.com  audit_log, method: "google.cloud.bigquery.…
2025-06-19 12:00:54.457  bigquery.googleapis.com  jobservice.insert  projects/iandemo2/jobs  Ianlingben@gmail.com  audit_log, method: "jobservice.insert", principal_email: "Ianlingben@gmail.com"
2025-06-19 12:01:07.722  bigquery.googleapis.com  jobservice.insert  projects/iandemo2/jobs  Ianlingben@gmail.com  audit_log, method: "jobservice.insert", principal_email: "Ianlingben@gmail.com"
2025-06-19 12:01:10.718  bigquery.googleapis.com  jobservice.insert  projects/iandemo2/jobs  Ianlingben@gmail.com  audit_log, method: "jobservice.insert", principal_email: "Ianlingben@gmail.com"
```

User access control is managed in BigQuery to ensure that team members can securely access and extract data for analysis and reporting. Additionally, GCP logs are monitored to ensure that operations run smoothly.

This setup provides a clean, scalable pipeline from raw data to the data warehouse, enabling seamless downstream processing for the rest of the team.

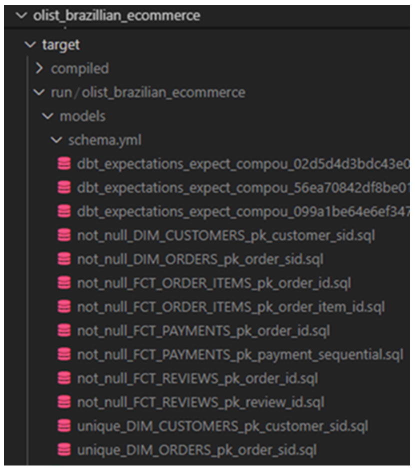
dbt

Using dbt, the data from the *olist_brazilian_ecommerce* dataset in BigQuery is now transformed to the Galaxy Schema Design and is written to the *olist_brazilian_ecommerce_target* dataset.



```
olist_brazilian_ecommerce_target
    DIM_CUSTOMERS
    DIM_DATE
    DIM_GEOLOCATION
    DIM_ORDERS
    FCT_ORDER_ITEMS
    FCT_PAYMENTS
    FCT_REVIEWS
    LKP_STATUS_DESC
```

Using dbt_test and dbt_expectations, 13 tests are implemented to verify that all primary keys are both unique and not null. Failed records are logged into the *run_results.json* file in the *target* folder.



```
olist_brazillian_ecommerce
  target
    compiled
    run / olist_brazilian_ecommerce
      models
        schema.yml
          dbt_expectations_expect_compou_02d5d4d3bdc43e0
          dbt_expectations_expect_compou_56ea70842df8be01
          dbt_expectations_expect_compou_099a1be64e6ef347
          not_null_DIM_CUSTOMERS_pk_customer_sid.sql
          not_null_DIM_ORDERS_pk_order_sid.sql
          not_null_FCT_ORDER_ITEMS_pk_order_id.sql
          not_null_FCT_ORDER_ITEMS_pk_order_item_id.sql
          not_null_FCT_PAYMENTS_pk_order_id.sql
          not_null_FCT_PAYMENTS_pk_payment_sequential.sql
          not_null_FCT_REVIEWS_pk_order_id.sql
          not_null_FCT_REVIEWS_pk_review_id.sql
          unique_DIM_CUSTOMERS_pk_customer_sid.sql
          unique_DIM_ORDERS_pk_order_sid.sql
```

Data Quality Testing

Data quality testing involves evaluating data to ensure it meets defined standards for accuracy, completeness, consistency, and key attributes defined by the business. All tests should pass based on what is expected and how it is supposed to behave.

Great Expectations

Great Expectations testing is performed to ensure data quality and reliability by defining and validating expectations (or assertions) about the data by quickly identifying and addressing data quality problems before they impact downstream processes or analysis.

Tests

Primary Key Schema Check	
DIM_CUSTOMERS	pk_customers_sid
DIM_DATE	pk_date_sid
LKP_STATUS_DESC	order_status payment_status
DIM_GEOLOCATION	geolocation_zip_code_prefix
DIM_ORDERS	pk_order_sid
FCT_ORDER_ITEMS	pk_order_id pk_order_item_id
FCT_PAYMENTS	pk_order_id pk_payment_sequential
FCT_REVIEWS	pk_order_id review_id

Foreign Key Schema Check	
DIM_ORDERS	fk_customer_sid

	fk_order_purchased_date_sid fk_order_approved_at_date_sid fk_order_delivered_carrier_date_sid fk_order_delivered_customer_date_sid fk_order_estimated_delivery_date_sid order_status payment_status
FCT_ORDER_ITEMS	fk_order_sid fk_shipping_limit_date_sid
FCT_PAYMENTS	fk_order_sid
FCT_REVIEWS	fk_order_sid fk_review_creation_date_sid

Error handling

Log files are written every time the validation is executed. Real time acquisition of the data is not required; this allows for further verification of the errors when raised.

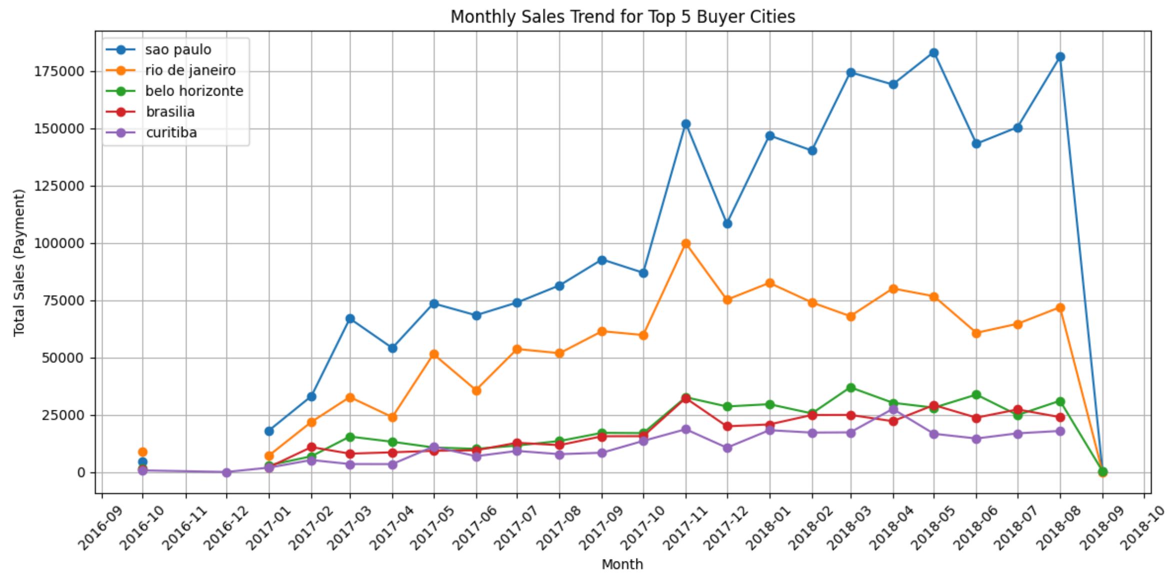
Data Analytics

Findings Summary

Populous cities are a direct reflection of their economic vitality. This translates to higher volume of sales with more people that have higher purchasing power.

Olist's most profitable sales come from the *bed_bath_table* category, with additional contributions from categories of *health_beauty*, *furniture_decor* and *sports_leisure*.

Sales trend by month (Top 5 cities)



Conclusions

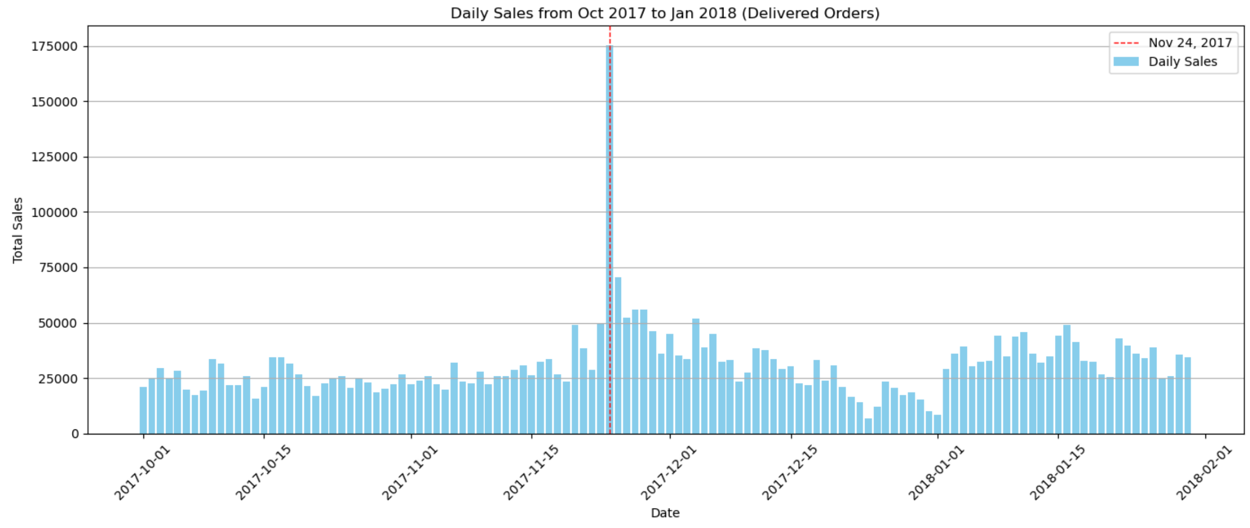
São Paulo consistently has the highest sales by a significant margin throughout the period, followed by Rio de Janeiro. These two cities are the primary drivers of sales among the top 5. Belo Horizonte, Brasilia and Curitiba have considerably lower sales volumes compared to São Paulo and Rio de Janeiro, with their sales often staying below the 50,000 mark.

There appears to be a general upward trend in sales for São Paulo and Rio de Janeiro, especially from late 2016 into 2017. There are also noticeable peaks and troughs, suggesting potential seasonality or specific sales events.

Specific Anomaly

As seen at the right end of the chart, sales in 3 cities, São Paulo, Rio de Janeiro and Belo Horizonte dropped to the 0 mark, while Brasilia and Curitiba had no data. This is likely due to a data cut off.

Daily Sales Across Time

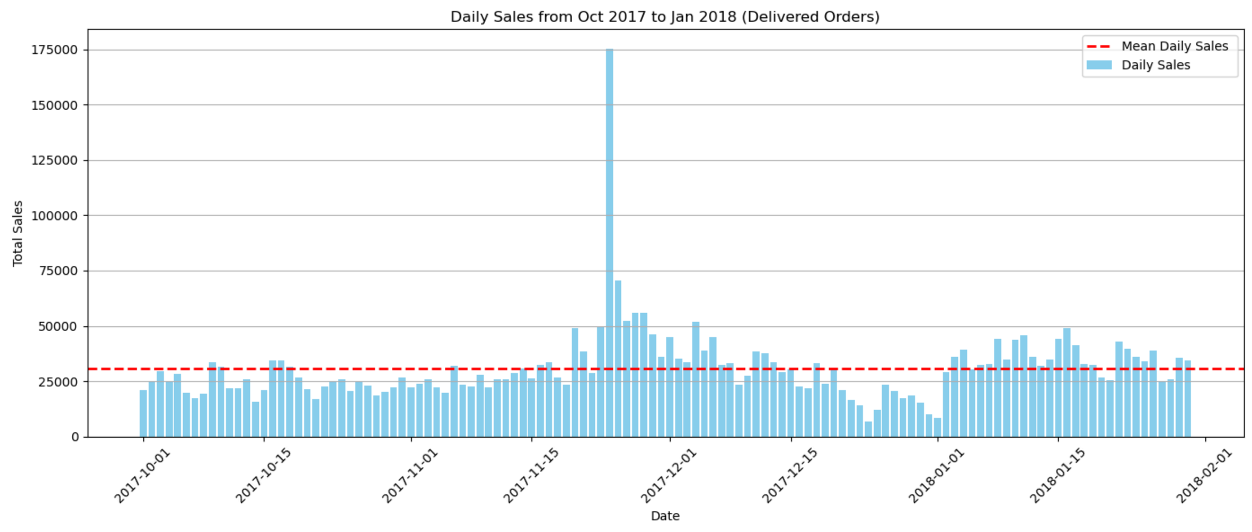


Conclusions

Black Friday Sales was on 24th Nov 2017 in Brazil, resulting in a significant sales increase as shown.

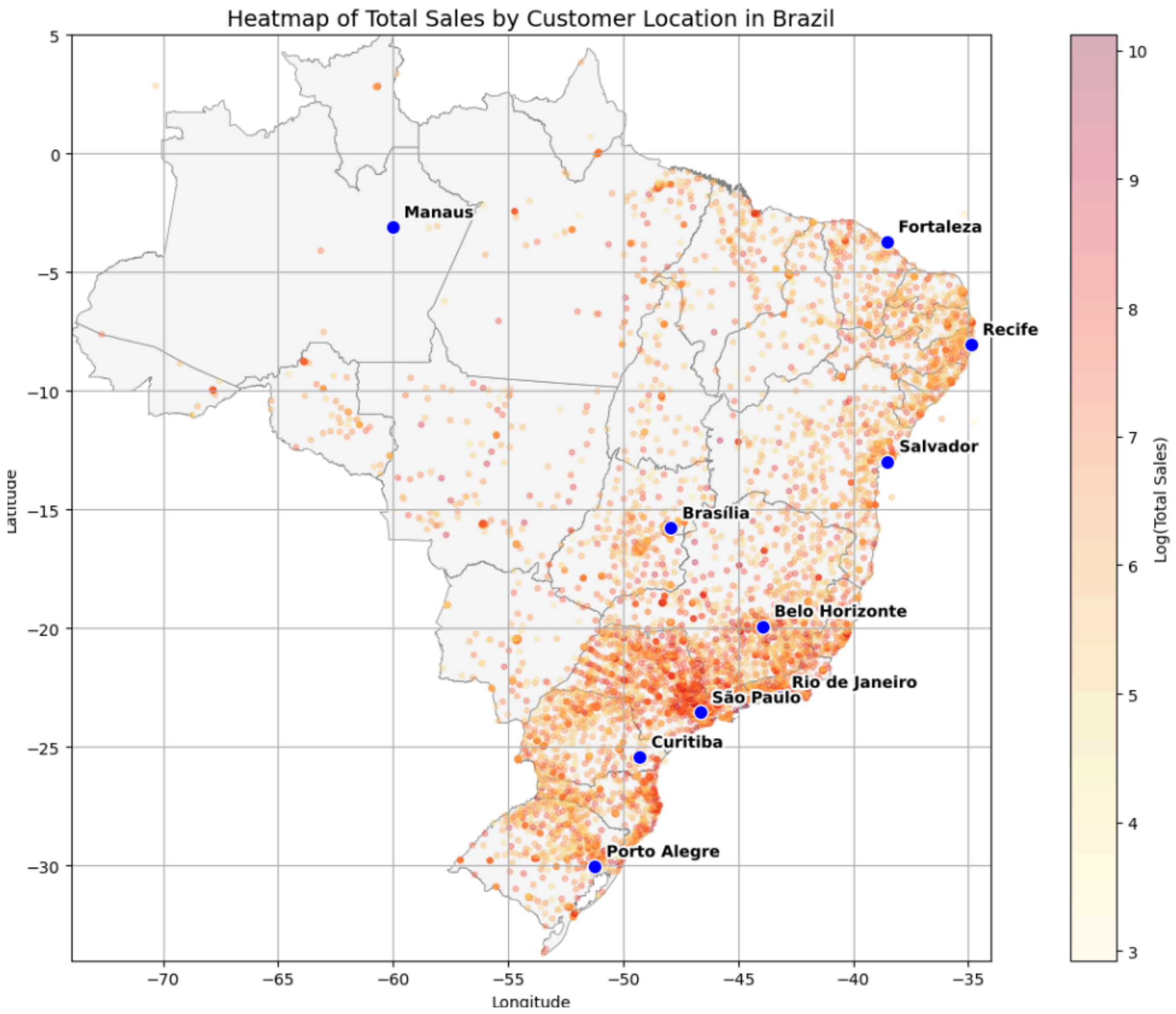
Specific anomaly

An anomaly was noticed during a specific day where sales went beyond the mean daily sales (more than 6x) to reach the 175,000 mark, this has led to the conclusion above



Heatmap plots

In this instance, the total sales is plotted by location data to determine where the bulk of the orders are made. To avoid excessive plotting, the threshold for the plot starts from the range 3 to 10.

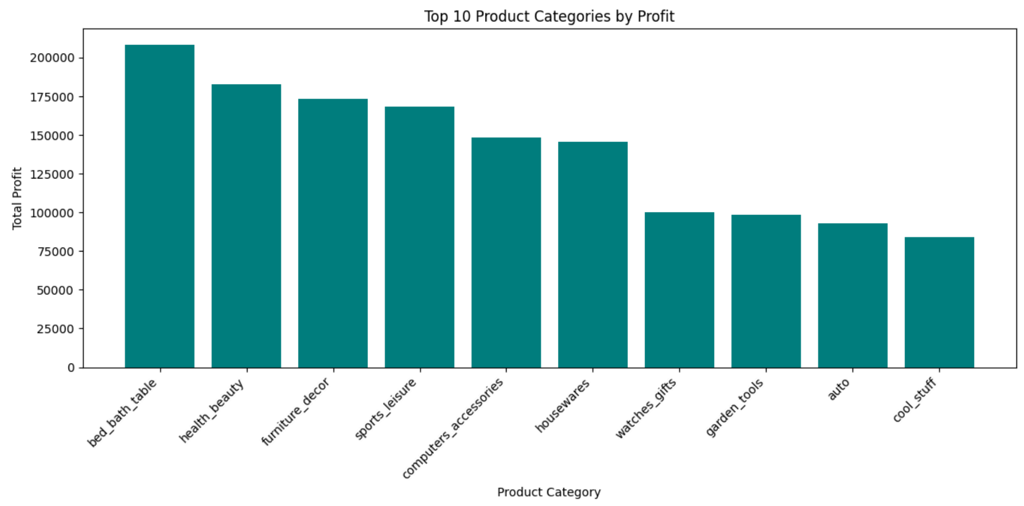


Conclusions

The heatmap visualizes the sales distribution by geo location. It strongly confirms the dominance of São Paulo and Rio de Janeiro being the regions having the most sales.

The high sales observed in São Paulo and Rio de Janeiro are a direct reflection of their status as Brazil's largest and most economically vital cities. More people, combined with higher purchasing power and economic activity, naturally generate a much greater volume of sales compared to other, less populated, or less economically central cities.

Top product categories by profit



Conclusions

The chart data shows the dominance of Home and Personal Care Categories, followed by Leisure and Technology.

Home-Centric Spending: There's a clear strong emphasis on products that enhance or furnish the home.

Personal Well-being and Appearance: *health_beauty* highlights consumer focus on personal care and looking good.

Lifestyle and Hobbies: Categories like *sports_leisure* and *garden_tools* indicate that consumers are actively pursuing hobbies and recreational activities.

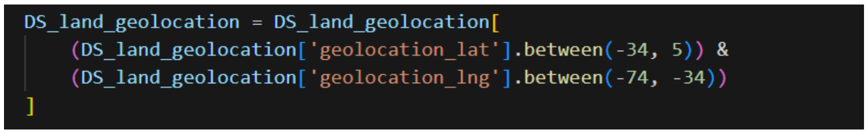
Technological Integration: *computers_accessories* points to the ongoing integration of technology into daily life and the need for related products.

Exploratory Data Analysis (EDA)

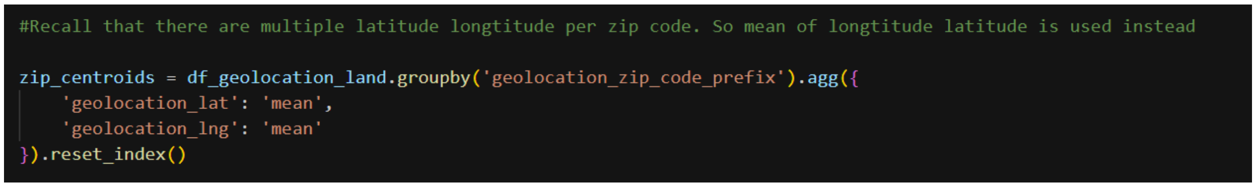
This crucial, initial step involves examining data to try to uncover patterns, anomalies, and relationships without preconceived assumptions; understand the data's structure and potential, guiding further analysis and model building (when needed in the future).

To support some of the findings mentioned these are the documented extracts in the form of coded samples.

Geolocation aggregation for heatmaps



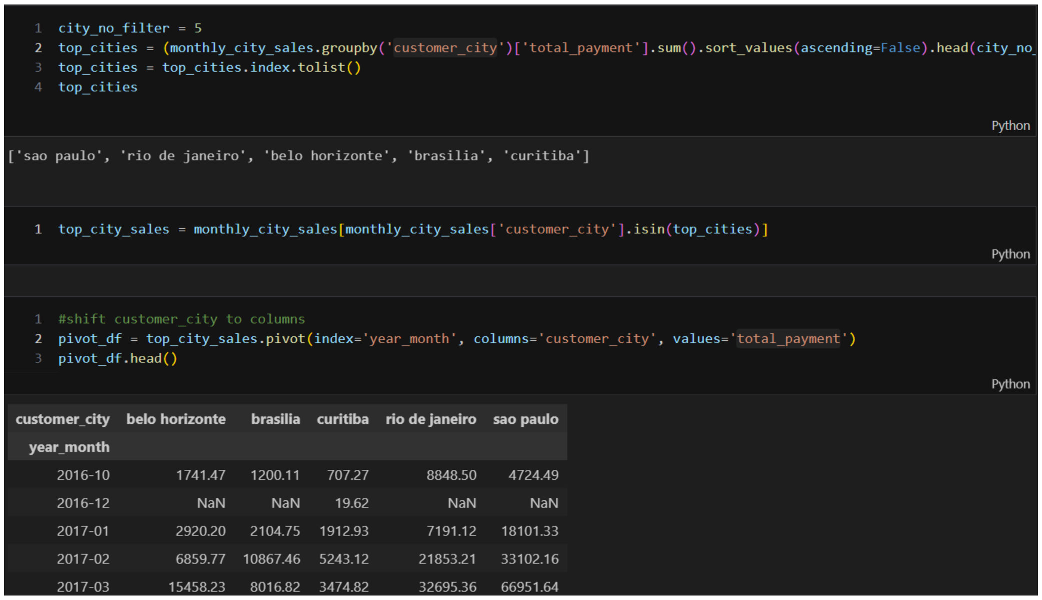
```
DS_land_geolocation = DS_land_geolocation[
    (DS_land_geolocation['geolocation_lat'].between(-34, 5)) &
    (DS_land_geolocation['geolocation_lng'].between(-74, -34))
]
```



```
#Recall that there are multiple latitude longtitude per zip code. So mean of longtitude latitude is used instead

zip_centroids = df_geolocation_land.groupby('geolocation_zip_code_prefix').agg({
    'geolocation_lat': 'mean',
    'geolocation_lng': 'mean'
}).reset_index()
```

Cities with the highest sales



```
city_no_filter = 5
top_cities = (monthly_city_sales.groupby('customer_city')['total_payment'].sum().sort_values(ascending=False).head(city_no_
top_cities = top_cities.index.tolist()
top_cities
```

```
['sao paulo', 'rio de janeiro', 'belo horizonte', 'brasilia', 'curitiba']
```

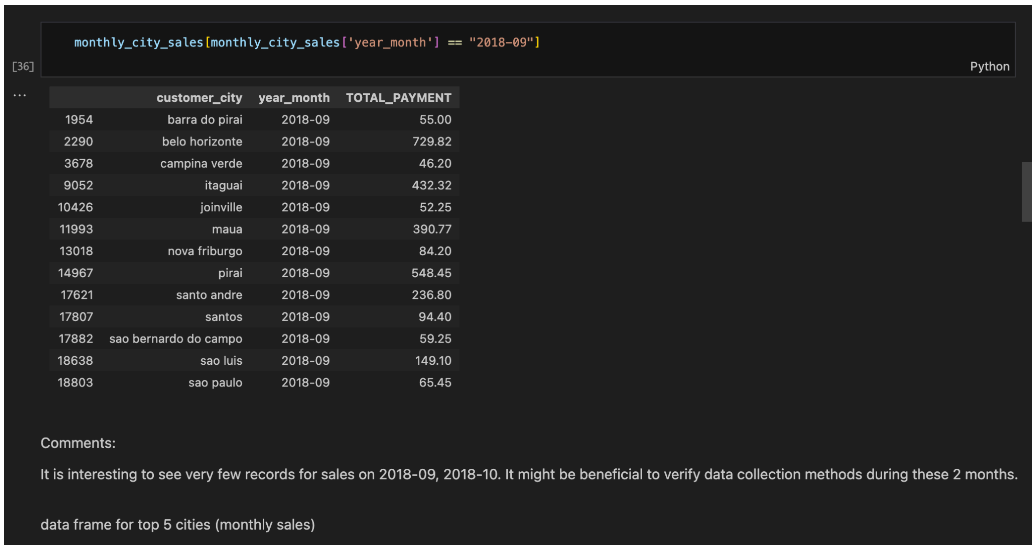
```
top_city_sales = monthly_city_sales[monthly_city_sales['customer_city'].isin(top_cities)]
```

```
#shift customer_city to columns
pivot_df = top_city_sales.pivot(index='year_month', columns='customer_city', values='total_payment')
pivot_df.head()
```

customer_city	belo horizonte	brasilia	curitiba	rio de janeiro	sao paulo
year_month					
2016-10	1741.47	1200.11	707.27	8848.50	4724.49
2016-12	NaN	NaN	19.62	NaN	NaN
2017-01	2920.20	2104.75	1912.93	7191.12	18101.33
2017-02	6859.77	10867.46	5243.12	21853.21	33102.16
2017-03	15458.23	8016.82	3474.82	32695.36	66951.64

Top 5 cities for monthly sales are filtered and data is transformed for ease in plotting graphs to show the monthly sales trend in top 5 cities.

The most crucial column would be *TOTAL_PAYMENT* in the *orders_full* table and aggregated to find out monthly city sales below, an example of 2018 Sep is shown.



```
[36]
monthly_city_sales[monthly_city_sales['year_month'] == "2018-09"]
```

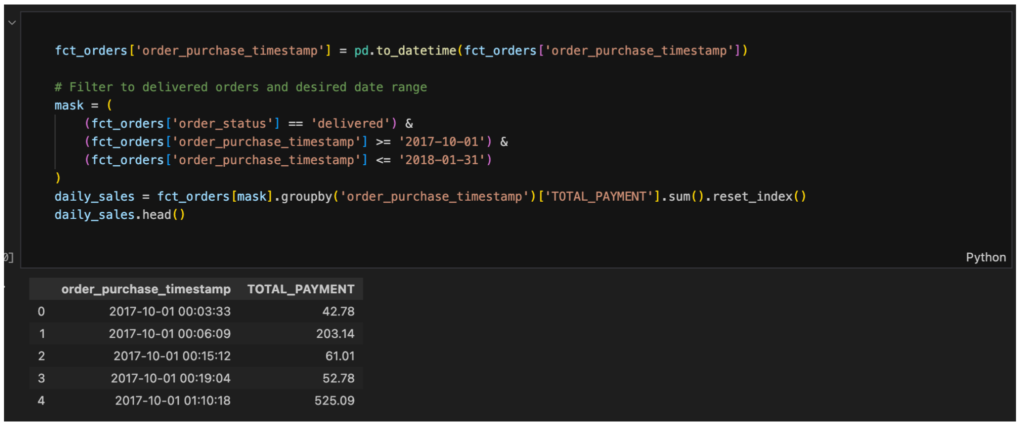
```
                customer_city  year_month  TOTAL_PAYMENT
1954           barra do pirai     2018-09          55.00
2290          belo horizonte      2018-09         729.82
3678            campina verde     2018-09          46.20
9052                  itaguai     2018-09         432.32
10426               joinville     2018-09          52.25
11993                    maua     2018-09         390.77
13018           nova friburgo     2018-09          84.20
14967                   pirai     2018-09         548.45
17621             santo andre     2018-09         236.80
17807                  santos     2018-09          94.40
17882  sao bernardo do campo      2018-09          59.25
18638                sao luis     2018-09         149.10
18803               sao paulo     2018-09          65.45
```

Comments:

It is interesting to see very few records for sales on 2018-09, 2018-10. It might be beneficial to verify data collection methods during these 2 months.

data frame for top 5 cities (monthly sales)

Top 5 largest daily sales value, across the entire time frame of the dataset.



```python
fct_orders['order_purchase_timestamp'] = pd.to_datetime(fct_orders['order_purchase_timestamp'])

# Filter to delivered orders and desired date range
mask = (
    (fct_orders['order_status'] == 'delivered') &
    (fct_orders['order_purchase_timestamp'] >= '2017-10-01') &
    (fct_orders['order_purchase_timestamp'] <= '2018-01-31')
)
daily_sales = fct_orders[mask].groupby('order_purchase_timestamp')['TOTAL_PAYMENT'].sum().reset_index()
daily_sales.head()
```

```
   order_purchase_timestamp  TOTAL_PAYMENT
0       2017-10-01 00:03:33          42.78
1       2017-10-01 00:06:09         203.14
2       2017-10-01 00:15:12          61.01
3       2017-10-01 00:19:04          52.78
4       2017-10-01 01:10:18         525.09
```

Looking beyond - Feature Engineering

To prepare the data for analysis to gather new features, further transformation of the data is performed and these new tables sent back into the data warehouse. They can be used for business case specific tasks or to build machine learning models.

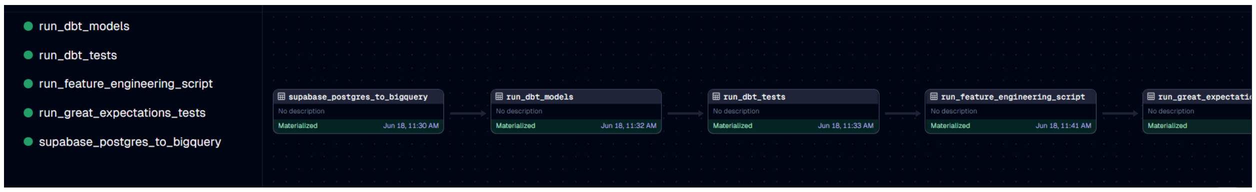
An example would be when calculating generated profits. This requires the payment amount and price through merging tables in the existing schema; one-to-many mapping will occur, and this can be solved by aggregation. With this new dataset, for a specific use case, it can be written back to the warehouse allowing for easy reuse.

Orchestration

With any form of data warehouse, being able to string together the acquisition pipeline would essentially form the key intelligence for the business with updated data.

When performing batch acquisition of data, implementing quality checking into the process is required as it will be run unattended and still needs to yield the same consistency as the initial import.

Dagster

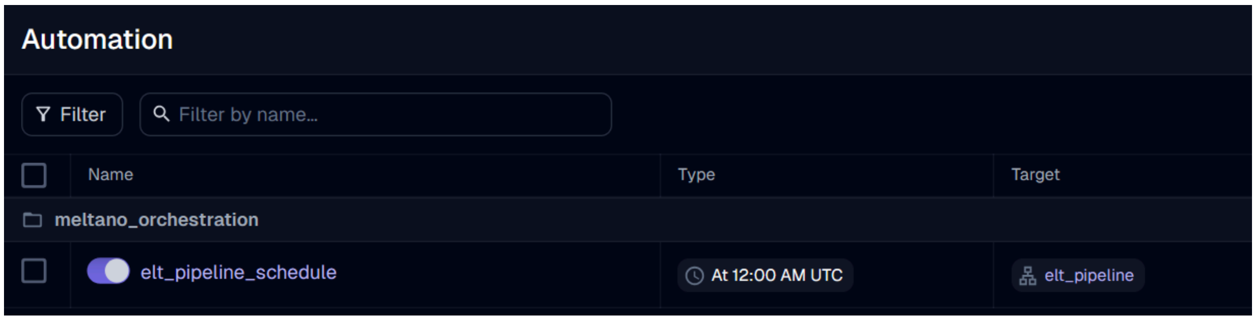


When the data begins its migration process from Supabase, the verification checks ensure the newly acquired data meets the requirements of ingestion.

As there are no requirements yet to how Great Expectations messages are to be delivered, it is currently implemented by logging to file.

Automation by Scheduling

Real time data acquisition, in this scenario, is not required. A simple daily schedule of running once will allow for new data to be ingested into the pipeline with minimal intervention using the in-built scheduler.


Automation

Name	Type	Target
meltano_orchestration		
elt_pipeline_schedule	At 12:00 AM UTC	elt_pipeline

Improvements

Data security

During the current development phase, the codes are run using user access rights to GCP. Depending on the deployment platform and its security features, a service account or API can be used, in place of the user accounts.

Dataset management

As the dataset grows, loading the entire data set daily becomes impractical. Daily updates can be delivered as a delta dataset. A possible solution would be to use dbt to perform delta loading and snapshot captures.

Additional visualisation tools

Beyond the EDA, user centric tools like Power BI or Tableau can be used to allow more convenient access to the data allowing end users to visualise the data without having to write codes in Python.