Base Labs Project – Functional Description

Brief project description

The main target of this project is building an automated and scalable data architecture that allows the extraction, transformation, and load of information from files published by PwC, in order to convert raw data into business data that can be consumed through dashboards in PowerBI.

Requirements

To fully execute the process, it is necessary to:
- Have Python installed, as well as Visual Studio Code
- Have Docker installed, since a large part of the process runs inside a container
- Install some Python dependencies, which are listed in a `.txt` file in the repository: https://github.com/C021943/LabsProject

Stack of technologies

The project is written entirely in Python, but Docker is also used to run PySpark and Delta without the need to install dependencies locally.
Additionally, a PostgreSQL database hosted on Railway is used as an intermediate step before consuming the data in PowerBI.
https://railway.com/project/810f470f-5122-4983-b294-5641f0fbb82c/service/d0fa5575-6cf6-4062-b795-e2f39e7d7f79/data?environmentId=d81a0282-4067-4e2c-8f6a-256e69115d27&state=table&table=margins_profits

Architecture Overview

Given the limitations of free-tier cloud tools, the decision was made to work locally, simulating a Data Lake. Additionally, since the goal was working with PySpark and Delta, a container was created to run most of the notebooks, except for the ingestion step.

Prelanding

First, running the notebook “01_prelanding/Template_Ingestion.py” creates or updates the prelanding folder, which is intended to store the raw source data. The script crawls the provided website for links and downloads those that end in “.zip”, saving them in the corresponding folder as CSV files.
If the variable “carga” is set to “-”, all ZIP files are downloaded. On the other hand, if a specific value is provided, such as “Sales”, it will download and update only the Sales folder.
The values that this variable can take are: ["-", "Sales", "PurchasesFINAL", "InvoicePurchases", "PurchasePrices", "BegInv", "EndInv"].

To run the notebook, it is not necessary to use the container; it is enough to install the libraries listed in the “requirements.txt” file.

Bronze

In this layer, the data is taken in its original format and saved as Delta files in their corresponding folders within the Bronze directory. No filters are applied; the data is saved in its entirety.

The notebooks used are:
["notebooks/02_bronze/beginv_bronze.py", "notebooks/02_bronze/endinv_bronze.py", "notebooks/02_bronze/invoicepurchases_bronze.py", "notebooks/02_bronze/purchasesfinal_bronze.py", "notebooks/02_bronze/purchasesprices_bronze.py", "notebooks/02_bronze/sales_bronze.py"].

It is important to note that in order to run these scripts, the spark-delta cluster must be started, as configured in the docker-compose.yml file. This can be done through the Docker interface.

Once the container is running, the following command can be used to access it:

```
docker exec -it spark-delta bash
```

and then:

```
spark-submit
--packages io.delta:delta-spark_2.12:3.1.0,org.postgresql:postgresql:42.5.0
--conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
--conf spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
notebooks/04_bronze/file.py
```

to run the notebook. This applies to all scripts except for ingestion and the pipeline.

Silver

In this layer, the data is cleaned and prepared for the Gold layer. The invoice and PurchasePrices tables are discarded since they are not useful for calculating margins and profits. Only the Sales, Purchases, Beginning Inventory, and Ending Inventory tables are processed.

Additionally, rows with null key fields are filtered out, unnecessary columns are removed, and key columns such as ‘description’ are normalized.

Again, to run the scripts in this layer, it is necessary to follow the same steps as in the Bronze layer.

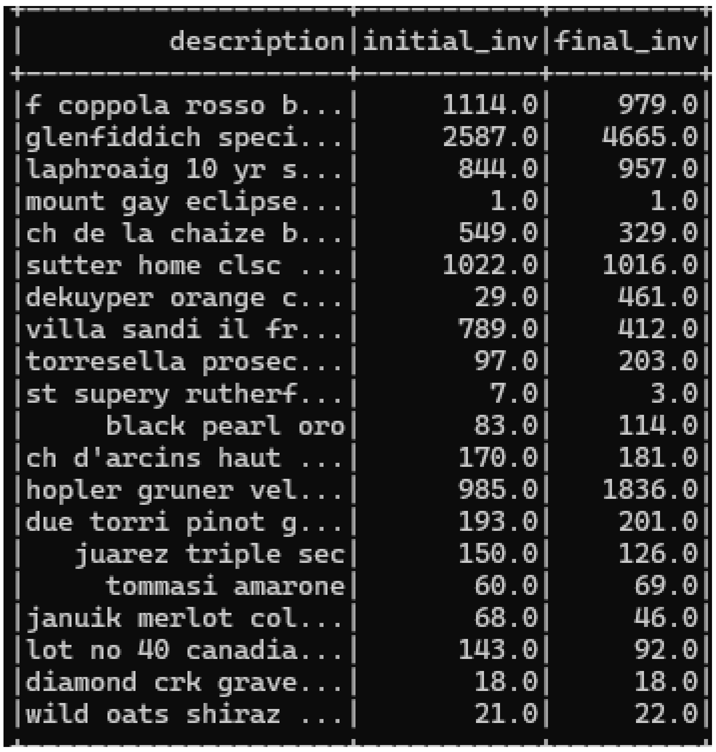
Gold

In this layer, only the tables needed to calculate the stated problems are created. A table called ‘margins_profits’ is generated, which is built from ‘Sales’ and ‘purchasesFINAL’, since sales represent the revenue from sold products (excluding tax), and purchases represent the expenses from buying products.

To achieve this, the total purchase cost is calculated per product and vendor. Additionally, revenue and tax are calculated per product and vendor.

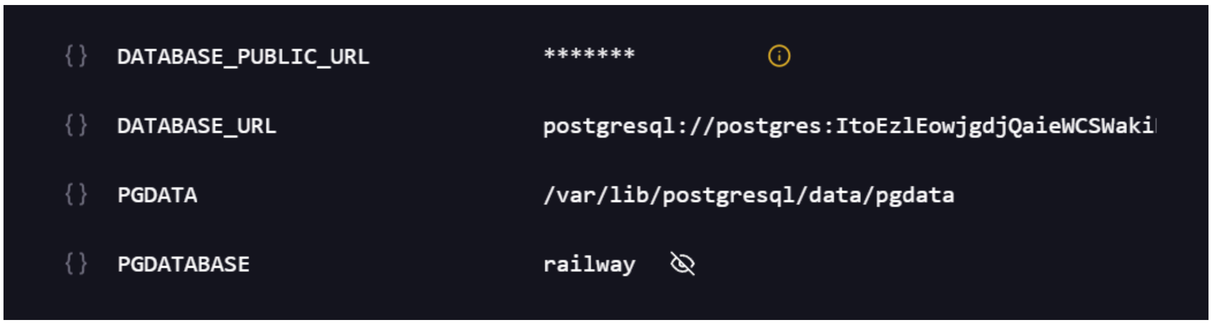
Next, only matching records between both tables are joined, since there are products that were bought but not sold, and vice versa. This happens because only one specific year is analyzed, so it’s possible a product was bought in December 2015 and sold in 2016, or bought in December 2016 and sold in 2017. In both cases, it’s not fair to calculate margin and profit. That’s why we only analyze matching products.

Lastly, although inventory information was considered to complement the margin and profit analysis, it is not accurate, since only two movements are recorded: stock at the beginning and at the end of the year. However, we don’t know what happened in between—some products have more inventory, others less—making it a black box that requires too many assumptions to be useful.

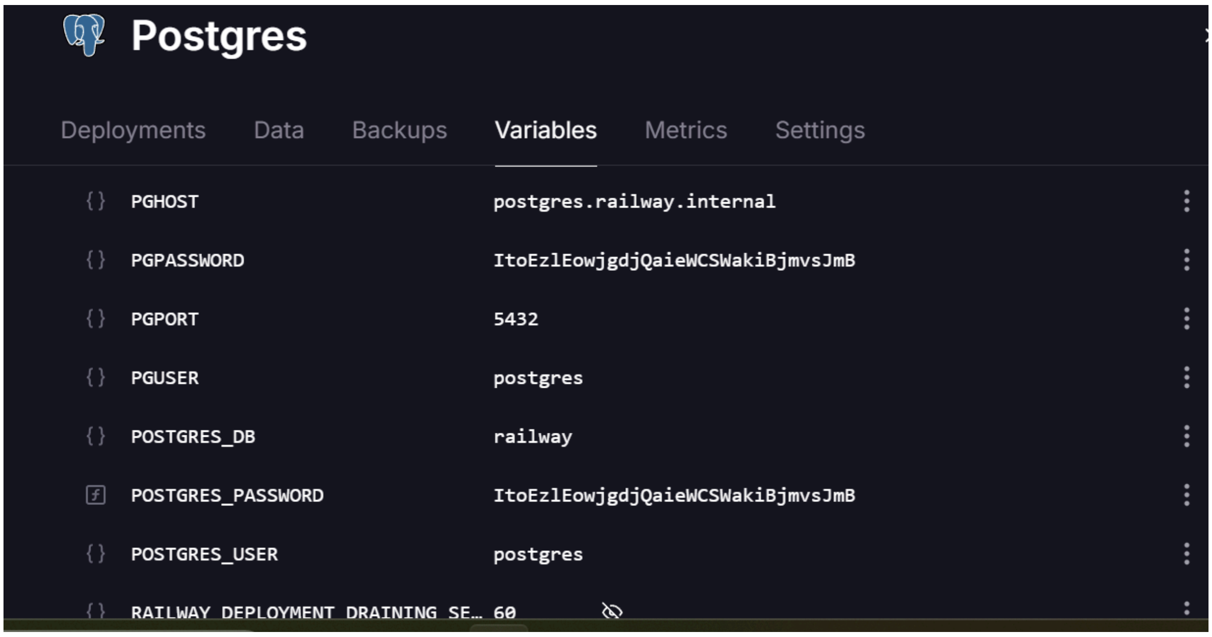


```
+--------------------+-----------+---------+
|         description|initial_inv|final_inv|
+--------------------+-----------+---------+
|f coppola rosso b...|     1114.0|    979.0|
|glenfiddich speci...|     2587.0|   4665.0|
|laphroaig 10 yr s...|      844.0|    957.0|
|mount gay eclipse...|        1.0|      1.0|
|ch de la chaize b...|      549.0|    329.0|
|sutter home clsc ...|     1022.0|   1016.0|
|dekuyper orange c...|       29.0|    461.0|
|villa sandi il fr...|      789.0|    412.0|
|torresella prosec...|       97.0|    203.0|
|st supery rutherf...|        7.0|      3.0|
|     black pearl oro|       83.0|    114.0|
|ch d'arcins haut ...|      170.0|    181.0|
|hopler gruner vel...|      985.0|   1836.0|
|due torri pinot g...|      193.0|    201.0|
|   juarez triple sec|      150.0|    126.0|
|     tommasi amarone|       60.0|     69.0|
|januik merlot col...|       68.0|     46.0|
|lot no 40 canadia...|      143.0|     92.0|
|diamond crk grave...|       18.0|     18.0|
|wild oats shiraz ...|       21.0|     22.0|
+--------------------+-----------+---------+
```

PostgreSQL
This is not a layer of the Data Lake. It is an intermediate step between the Data Lake and the dashboard in Sigma.
A PostgreSQL database is created in Railway, which contains the following variables:



DATABASE_PUBLIC_URL	*******
DATABASE_URL	postgresql://postgres:ItoEzlEowjgdjQaieWCSWaki
PGDATA	/var/lib/postgresql/data/pgdata
PGDATABASE	railway



Postgres

PGHOST	postgres.railway.internal
PGPASSWORD	ItoEzlEowjgdjQaieWCSWakiBjmvsJmB
PGPORT	5432
PGUSER	postgres
POSTGRES_DB	railway
POSTGRES_PASSWORD	ItoEzlEowjgdjQaieWCSWakiBjmvsJmB
POSTGRES_USER	postgres
RAILWAY_DEPLOYMENT_DRAINING_SE…	60

Using the script "05_postgres/marginsprofits_postgres.py", the gold data from margins_profits is sent to a table with the same name in PostgreSQL.

The link is shared:

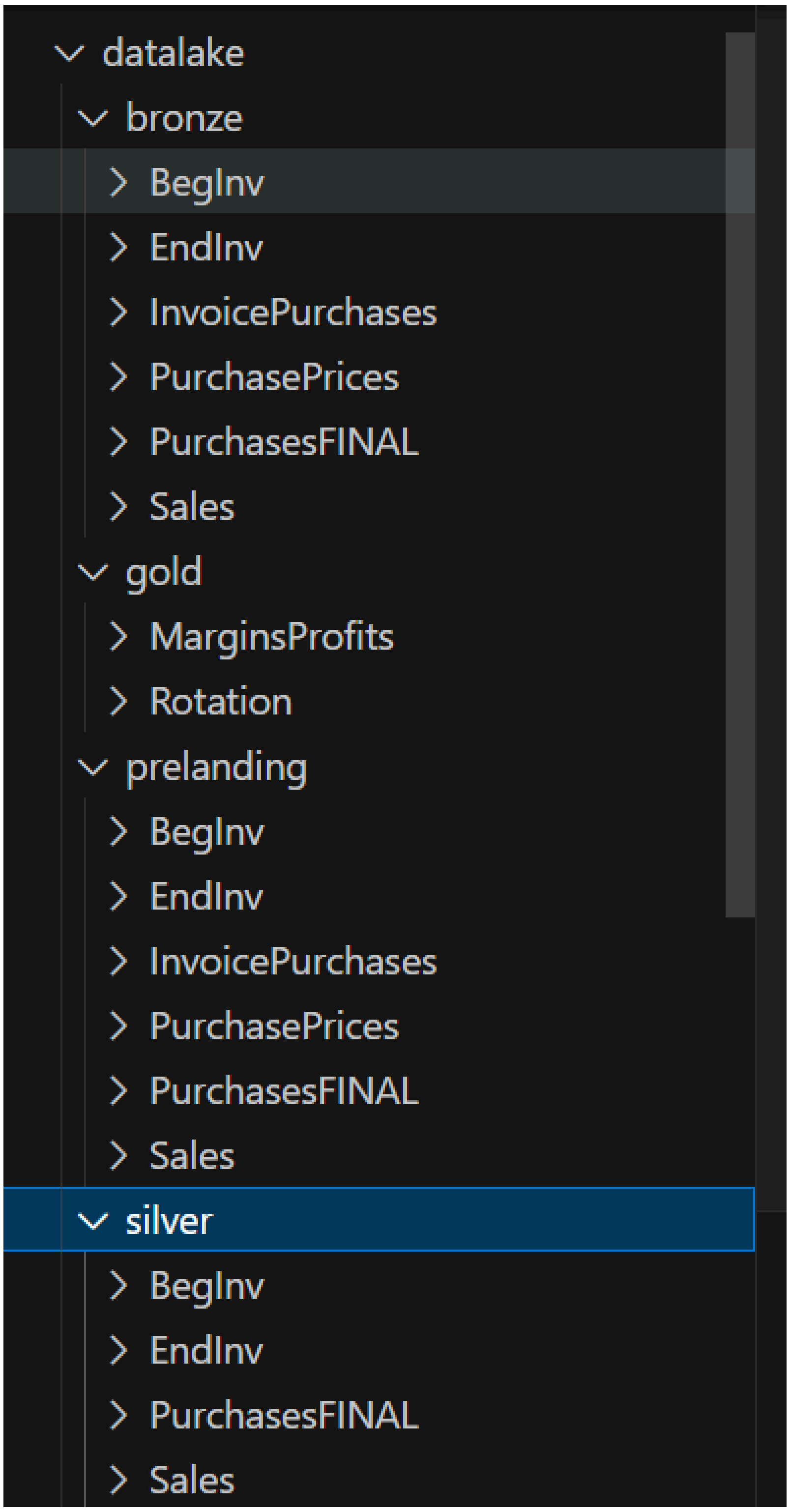
https://railway.com/project/810f470f-5122-4983-b294-5641f0fbb82c/service/d0fa5575-6cf6-4062-b795-e2f39e7d7f79/variables?environmentId=d81a0282-4067-4e2c-8f6a-256e69115d27

PowerBi

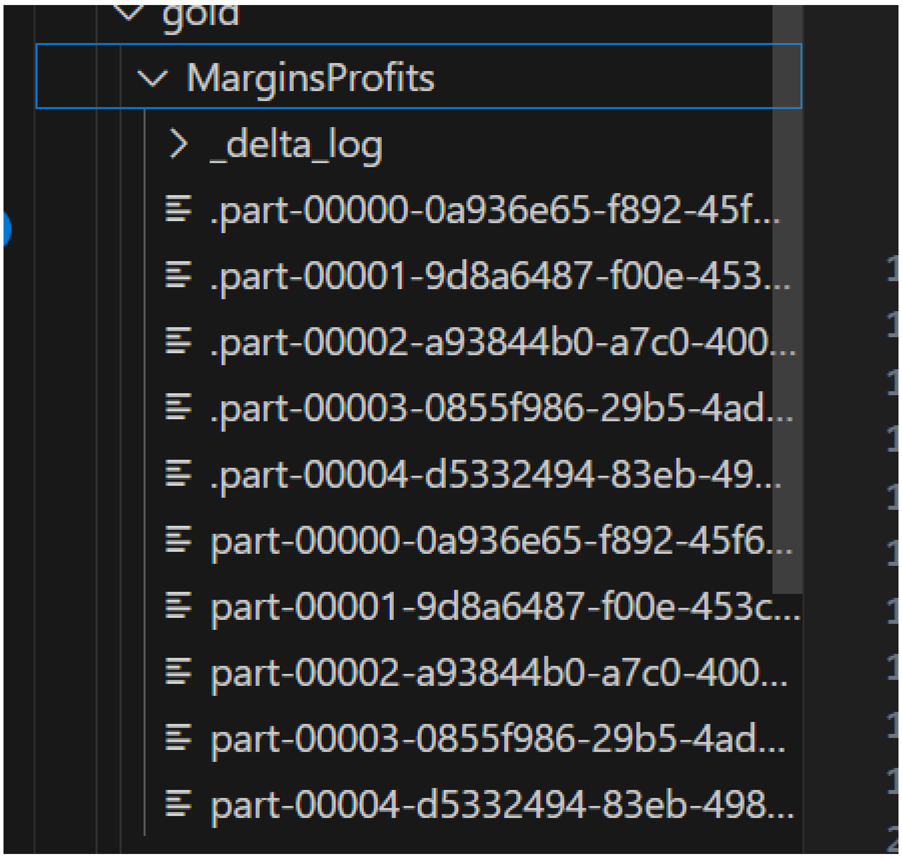
The PostgreSQL database is used as the data source in PowerBi.

Pipeline

Finally, a pipeline written in Python is created, which executes the entire process from data extraction, through each data layer, to the final writing into PostgreSQL.
The ingestion runs locally, and then the remaining steps are executed inside the container. This is possible because the pipeline includes the necessary commands to start the container and run the scripts.
The end result is a simulated Data Lake within the project folder, organized by layers and subfolders within each layer.



```
datalake
  bronze
    BegInv
    EndInv
    InvoicePurchases
    PurchasePrices
    PurchasesFINAL
    Sales
  gold
    MarginsProfits
    Rotation
  prelanding
    BegInv
    EndInv
    InvoicePurchases
    PurchasePrices
    PurchasesFINAL
    Sales
  silver
    BegInv
    EndInv
    PurchasesFINAL
    Sales
```


```
gold
  MarginsProfits
    _delta_log
    .part-00000-0a936e65-f892-45f...
    .part-00001-9d8a6487-f00e-453...
    .part-00002-a93844b0-a7c0-400...
    .part-00003-0855f986-29b5-4ad...
    .part-00004-d5332494-83eb-49...
    part-00000-0a936e65-f892-45f6...
    part-00001-9d8a6487-f00e-453c...
    part-00002-a93844b0-a7c0-400...
    part-00003-0855f986-29b5-4ad...
    part-00004-d5332494-83eb-498...
```
It can be run locally without installing dependencies, from the folder: pipelines/ppl.py