Project Documentation

This project has 2 solutions as follows:

1. Whole ELT Pipeline with modern data tools
2. Local solution using jupyter notebook

Solution 1:

Overview

This project demonstrates a modern **data ingestion, transformation, and alerting pipeline** built using **AWS S3, Snowflake, dbt, Docker, Airflow, and Slack.**

The goal is to ingest raw financial data (invoices, organizations), transform it into analytics-friendly models, and trigger alerts when organization balances change by more than 50% day-over-day.

Steps:

1. **Data Ingestion**
    a. **IAM Setup:** Created IAM user/role in AWS for secure authentication and access control.
    b. **S3 Bucket Setup**: Configured an S3 storage bucket to upload .csv files (invoices, organizations).
    c. **File Upload:** Uploaded raw .csv datasets into the bucket.
2. **Data Extraction & Load (Snowflake Setup)**
    a. **Platform Setup:**
        i. Created user: **DEEL**
        ii. Created warehouse: **deel**
        iii. Created database: **deel_db**
        iv. Created schema: **raw** (for staging raw data)
        v. Created role: **deel_ownership** with full privileges (warehouse, schema, tables).
        vi. Assigned role **deel_ownership** to user **DEEL**.
3. **Data Extraction Setup:**
    a. Created a storage integration between **Snowflake** and **AWS S3**.
    b. Created a **Snowflake** stage pointing to **S3** for raw file ingestion.
4. **Data Extraction:**
    a. Used **COPY INTO** commands to load data into:
        i. deel_db.raw.invoices
        ii. deel_db.raw.organizations
5. **Data Transformation (dbt Setup)**
    a. **DBT Setup:**
        i. Installed **dbt-core** locally inside VS Code.
        ii. Initialized dbt project **deel**.
        iii. Configured Snowflake connection in **profiles.yml**.

   b. **Staging Models:**
      i. Created **sources.yml** mapping **deel_db.raw** as dbt source.
      ii. Created **stg_invoices** and **stg_organizations** in schema **deel_db.staging.**
   c. **Warehouse Models (Facts & Dimensions):**
      i. **Facts:**
         1. **fct_org_daily_balances**(grain: date × organization)
         2. Measures: **total_amount, payment_amount, balance_amount = total_amount - payment_amount**
      ii. **Dimensions:**
         1. **dim_organizations** (grain: organization)
         2. Enriched with: total invoices, total payments, total invoice amounts.
   d. **Testing & Documentation:**
      i. Added data tests (**not_null, unique**) on the primary keys of the Staging tables
      ii. Added **uniqueness and not null tests** on **the grain of the fct_org_daily_balances(date/organization) table**
      iii. Added **not null and uniqueness tests** on the **dim_organizations**

6. **Alerts (Airflow + Slack)**
   a. **Docker & Airflow Setup:**
      i. Installed Docker Desktop.
      ii. Built custom Airflow image via **Dockerfile**.
      iii. Created **docker-compose.yaml** to run Airflow with webserver, scheduler, and workers.
   b. **Airflow Connections:**
      i. Configured Snowflake connection (for dbt/fct_invoices).
      ii. Configured Slack connection (Webhook).

   c. **Slack API Setup:**
      i. Created Slack workspace and new channel **#balance-change-alert**.
      ii. Created a new Slack App, generated API token & Webhook URL.
      iii. Added token to Airflow Slack connection.

   d. **Alert Logic (Airflow DAG):**
      i. Query **deel_db.facts.fct_org_daily_balances** for latest date only.
      ii. Compute day-over-day change in **balance_amount**.
      iii. Trigger Slack alert if balance changes > 50%.
      iv. Post formatted alert message in **#balance-change-alert**.

7. **Best Practices / Extras**
   a. Version controlled the entire project with GitHub.
   b. Created requirements.txt to lock dependencies.
   c. Enabled Snowflake RBAC

d. Designed dbt models using a layered architecture: RAW → STAGING → FACTS/DIMS → ALERTS.
e. Added logging in Python alert script in Airflow.

Future Scope

1. All secrets (Snowflake, Slack) stored in Airflow's encrypted backend.
2. Add CI/CD pipeline for SQL + DAG deployment.

Solution 2:

This is a .ipynb file that performs all actions from solution 1. Instead of using modern tools, the solution is created using a jupyter notebook.

Overview:

1. Environment Setup

- Imports Python libraries: pandas, os, json, requests, datetime
- Reads environment variables:
    - INVOICES_CSV, ORGS_CSV → paths to CSVs
    - SLACK_WEBHOOK_URL → Slack webhook URL

2. Data Ingestion

- If CSV paths are set, loads invoices & organizations data from local files.
- Otherwise, throw an error.
- Saves them into a local output folder

3. Staging Layer (Mimicking dbt staging)

- Cleans column names, standardizes date formats.
- Creates **stg_invoices** and **stg_organizations** DataFrames.

4. Warehouse Layer (Facts & Dimensions)

- **Fact Table (fct_invoices)**
    - Aggregated by (organization_id, invoice_date)

    - Metrics:
        - total_amount = sum(amount)
        - total_payment = sum(payment_amount)
        - balance = total_amount – total_payment

- **Dimension Table (dim_organizations)**

    - Aggregated by organization_id
    - Metrics:
        - total_invoices
        - total_invoice_amount
        - total_payment_amount

- Saves both tables as CSVs in the output folder.

5. Data Quality Checks

- Ensures:

    - No duplicate invoice IDs
    - No negative amounts or payments
    - Valid organization references

Prints warnings if issues are found.

6. Alerts Logic (Day-over-Day Balance Change)

- Looks at fct_invoices for the **latest available date**
- Computes % change in balance for each organization:
  pct_change = (latest_balance - prev_balance) / (prev_balance} * 100
- If absolute % change > **50%**, trigger an alert.

7. Alert Delivery

- **If SLACK_WEBHOOK_URL is set**:

    - Sends formatted JSON message to Slack channel #balance-change-alert.

8. Outputs

- Writes following CSVs into the output folder:
    - stg_invoices.csv
    - stg_organizations.csv
    - fct_invoices.csv
    - dim_organizations.csv

- sends alerts on Slack