**Lab Exercise 21- ETL (Extract, Transform, Load) data pipeline from CSV to Postgres using Metaflow**

In this lab exercise, we will build an **ETL (Extract, Transform, Load) data pipeline** using **Metaflow**. This pipeline will:

1. **Extract** data from a CSV file.
2. **Transform** the data by performing some basic operations.
3. **Load** the transformed data into a **PostgreSQL** database.

**Objective:**

* Learn how to read data from a CSV file in Metaflow.
* Perform simple transformations on the data.
* Insert the transformed data into a PostgreSQL database.

**1. Prerequisites:**

1. Install the necessary Python packages:

pip install pandas psycopg2-binary metaflow

1. Ensure that you have access to a PostgreSQL database. You can either:
   * Use a cloud-hosted PostgreSQL instance like AWS RDS.
   * Use a local installation of PostgreSQL.
2. Prepare a PostgreSQL table to insert the transformed data:

**PostgreSQL Setup:**

CREATE DATABASE etl\_db;

CREATE TABLE users (

id SERIAL PRIMARY KEY,

name VARCHAR(255),

email VARCHAR(255),

signup\_month VARCHAR(255)

);

1. Prepare a CSV file (e.g., users.csv) with the following structure:

name,email,signup\_date

Alice,alice@example.com,2023-01-15

Bob,bob@example.com,2023-02-20

Charlie,charlie@example.com,2023-03-05

**2. Building the ETL Pipeline**

The ETL pipeline will be designed to:

* **Extract** data from the CSV file.
* **Transform** the data by adding a signup\_month field.
* **Load** the transformed data into the PostgreSQL database.

**File: etl\_csv\_postgres\_pipeline.py**

# etl\_csv\_postgres\_pipeline.py

import pandas as pd

import psycopg2

from datetime import datetime

from metaflow import FlowSpec, step

class ETLFlow(FlowSpec):

@step

def start(self):

"""

Step 1: Extract data from a CSV file.

"""

print("Extracting data from CSV...")

# Load CSV file using pandas

self.df = pd.read\_csv('users.csv')

print(f"Extracted {len(self.df)} rows.")

print(self.df.head()) # Show a sample of the extracted data

self.next(self.transform)

@step

def transform(self):

"""

Step 2: Transform the data by adding a 'signup\_month' column.

"""

print("Transforming data...")

# Convert 'signup\_date' to datetime and extract the month

self.df['signup\_date'] = pd.to\_datetime(self.df['signup\_date'])

self.df['signup\_month'] = self.df['signup\_date'].dt.strftime('%B') # Extract month name

# We no longer need the 'signup\_date' column

self.transformed\_data = self.df.drop(columns=['signup\_date'])

print("Transformed data:")

print(self.transformed\_data.head()) # Show a sample of the transformed data

self.next(self.load)

@step

def load(self):

"""

Step 3: Load the transformed data into the PostgreSQL database.

"""

print("Loading data into PostgreSQL...")

# PostgreSQL connection details

conn = psycopg2.connect(

host="your\_pg\_host",

user="your\_pg\_user",

password="your\_pg\_password",

database="etl\_db"

)

cursor = conn.cursor()

# Prepare insert query

insert\_query = """

INSERT INTO users (name, email, signup\_month)

VALUES (%s, %s, %s)

"""

# Insert each row into the PostgreSQL database

for \_, row in self.transformed\_data.iterrows():

cursor.execute(insert\_query, (row['name'], row['email'], row['signup\_month']))

conn.commit()

cursor.close()

conn.close()

print(f"Loaded {len(self.transformed\_data)} rows into PostgreSQL.")

self.next(self.end)

@step

def end(self):

"""

End the flow.

"""

print("ETL pipeline completed successfully.")

if \_\_name\_\_ == "\_\_main\_\_":

ETLFlow()

**3. Running the Pipeline**

You can run the pipeline using the following command:

python etl\_csv\_postgres\_pipeline.py run

**4. Breakdown of the Pipeline Steps**

* **Step 1: start**
  + We extract data from a CSV file using the pandas library.
  + The data is stored in a DataFrame (self.df).
* **Step 2: transform**
  + We transform the data by adding a signup\_month column, which is extracted from the signup\_date field.
  + The transformed data is stored in self.transformed\_data.
* **Step 3: load**
  + We connect to the PostgreSQL database and insert the transformed data row by row into the users table.
  + We use the psycopg2 library to handle the database connection and insertion.
* **Step 4: end**
  + This marks the end of the flow and prints a success message.

**5. Additional Exercises (Optional)**

1. **Error Handling**:
   * Add try-except blocks around the database connection and data loading steps to gracefully handle errors.
   * Ensure that if the data loading fails, you rollback the transaction.
2. **Data Validation**:
   * Add a validation step before loading the data into the database to ensure there are no missing values in critical columns like name or email.
3. **Parallel Processing**:
   * If the CSV file contains a large number of rows, you can split the data into chunks and process them in parallel using Metaflow’s foreach step.
4. **Logging**:
   * Add detailed logging throughout the pipeline to keep track of each step’s progress and any potential issues.

**6. Summary**

In this exercise, we have built an **ETL data pipeline** using **Metaflow** to:

* Extract data from a CSV file.
* Transform the data by adding a new field based on the signup date.
* Load the transformed data into a PostgreSQL database.

This pipeline can be expanded further by adding more complex transformations, error handling, and validation steps, making it suitable for real-world data workflows.