Data Engineering Notebook

**Course:** Data Engineering (2026SP-DSCI-6313-A)  
**Author:** Scott Person  
**Date:** February 16, 2026

Table of Contents

[Introduction 4](#_Toc222139839)

[Docker Install 4](#_Toc222139840)

[Why use Docker? 4](#_Toc222139841)

[Docker Installation 4](#_Toc222139842)

[Prerequisites 4](#_Toc222139843)

[Instructions 4](#_Toc222139844)

[PostgreSQL Install 6](#_Toc222139845)

[Why Use Postgres? 6](#_Toc222139846)

[Postgres Installation 6](#_Toc222139847)

[Docker Execution Script 6](#_Toc222139848)

[Script Instructions 7](#_Toc222139849)

[PostgreSQL Data Import 9](#_Toc222139850)

[Anaconda Install 12](#_Toc222139851)

[Deployment Architecture 12](#_Toc222139852)

[Anaconda Docker Configuration 12](#_Toc222139853)

[Docker Execution Script 12](#_Toc222139854)

[Script Instructions 13](#_Toc222139855)

[Docker Network Configuration 13](#_Toc222139856)

[VS Code Install and Configuration 16](#_Toc222139857)

[VS Code Installation 16](#_Toc222139858)

[Python Data Pipeline Setup 17](#_Toc222139859)

[Apache Nifi Install (Ravi, 2023) 19](#_Toc222139860)

[Nifi Docker Configuration 19](#_Toc222139861)

[Docker Execution Script 19](#_Toc222139862)

[Apache Nifi Usage (Ravi, 2023) 22](#_Toc222139863)

[Flow Configuration 22](#_Toc222139864)

[Flow Operations 23](#_Toc222139865)

[Sources 26](#_Toc222139866)

[Appendix A: Code Listings 26](#_Toc222139867)

[Create Tables SQL 26](#_Toc222139868)

[Insert Data SQL 28](#_Toc222139869)

[Extract Script Python 29](#_Toc222139870)

# Introduction

This document provides instructions for installing and configuring PostgreSQL (AKA Postgres) on MacOS using Docker Desktop. Web sites are somewhat fickle. They will change. The instructions below are accurate at the time of writing this document. Regardless of the changes, even if the exact instructions no longer work, it is highly likely that very similar steps will be appropriate for the foreseeable future. Be flexible and follow the spirit of this guide.

# Docker Install

## Why use Docker?

Docker Desktop provides virtual environments in the form of lightweight containers. It allows us to isolate each component to avoid conflicts and ease maintenance and operations. An additional benefit is that we avoid installing software directly on the host computer. This simplifies cleanup. As mentioned above these instructions will cover MacOS but the installation on Windows is quite similar.

## Docker Installation

### Prerequisites

Docker requires 4GB of RAM. Generally any machine with 8GB of total RAM will be able to run Docker successfully. MacOS is supported for the current version and two versions back. This is rolling support and moves as new versions of MacOS are released.

Basic familiarity with MacOS is assumed.

### Instructions

1. Navigate to the [Docker home page](https://www.docker.com/) ([https://www.docker.com](https://www.docker.com/)).
2. Hover over the Download Docker Desktop button and select the version of Docker Desktop that is appropriate for your machine as shown in Figure 1: Docker Homepage Download Link below. This will place a copy of the DMG install archive in your Downloads folder.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Docker Homepage Download Link

1. Open your Downloads folder and launch Docker.dmg.
2. Follow the instructions in the dialog (see Figure 2: Install Docker into Applications) to copy Docker to Applications. Once copied close the dialog.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Install Docker into Applications

1. Launch Docker from Applications and you’ll see a dialog similar to Figure 3: docker.desktop Dashboard. Note that the author uses Docker extensively for work and side projects so the screenshot shows several existing containers. We’ll create our own in the instructions below.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : docker.desktop Dashboard

# PostgreSQL Install

## Why Use Postgres?

Postgres is an open source well supported database engine that runs on most general purpose computing platforms. It works very well as a production database and is widely used in industry. Similarly it is often used in education environments because of its low cost (free) and general applicability to commercial implementations. We will utilizing a pre-built image and running it in Docker.

## Postgres Installation

### Docker Execution Script

I have a script that I like to use that performs the image download (if necessary), container creation (again if necessary), container startup, and finally opening an interactive shell to the container. This script can be used for starting the container in the future. The script is pasted below and explained in the included comments.

#!/bin/bash

# Variables

CONTAINER\_NAME="postgres\_data\_engineering"

IMAGE\_NAME="postgres:latest"

POSTGRES\_USER="admin"

POSTGRES\_PW="admin"

POSTGRES\_DB="salestracker"

POSTGRES\_PORT="5432"

#MOUNT\_PATH\_ON\_HOST="/Users/jsperson"

#MOUNT\_PATH\_IN\_CONTAINER="/home/jsperson/host"

# Check if the container exists

CONTAINER\_ID=$(docker ps -a -q -f name=$CONTAINER\_NAME)

# If the container does not exist, create and run it

if [ -z "$CONTAINER\_ID" ]; then

echo "Container does not exist. Creating and starting..."

docker run -d \

--name $CONTAINER\_NAME \

-e POSTGRES\_USER=$POSTGRES\_USER \

-e POSTGRES\_PASSWORD=$POSTGRES\_PW \

-e POSTGRES\_DB=$POSTGRES\_DB \

-p $POSTGRES\_PORT:$POSTGRES\_PORT \

$IMAGE\_NAME

else

# Check if the container is running

RUNNING\_CONTAINER\_ID=$(docker ps -q -f name=$CONTAINER\_NAME)

if [ -z "$RUNNING\_CONTAINER\_ID" ]; then

echo "Container exists but is stopped. Starting..."

docker start $CONTAINER\_NAME

else

echo "Container is already running."

fi

fi

# Connect to the container

echo "Connecting to container..."

docker exec -it $CONTAINER\_NAME /bin/bash

### Script Instructions

1. If you have not already, create a working directory where you will be storing files for this module.
2. Using a test editor copy the contents of the script in its entirety into a file. I named my file run\_docker\_postgres.sh.
3. At the command prompt in your working directory change permissions of the script created above to allow execution. Use something like: chmod 700 <script name>
4. Execute the script by typing: ./<script name>
5. During execution the script:
   1. Downloads the Postgres image – see Figure 4: Postgres Image
   2. Creates a container using the image – see Figure 5: Running Postgres Container
   3. Starts the container
   4. Connects to the container in a shell environment
6. In the future your script can be used to restart the container or you can use the Docker Desktop GUI. The GUI can also be used to delete the container (note: complete loss of the container data) and the image.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Postgres Image

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Running Postgres Container

1. From the command line of the container (you drop in there after running the above script in the terminal), execute: psql -U admin -d salestracker

If you have been successful you’ll be in the Postgres command environment with a prompt similar to salestracker=#

# PostgreSQL Data Import

Now we’re going to populate our database with some sample data using a couple of scripts.

1. In the terminal execute the script you created above. This will start the Postgres container if it’s not running and drop you into a shell inside of the container.
2. Enter psql -U admin -d salestracker to get into the Postgres terminal client. You should be sitting at a prompt similar to what is displayed in Figure 6: Database Running.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure : Database Running

1. Execute the Create Tables SQL and Insert Data SQL below. You can copy the entirety of both or either list or execute them one at a time in the terminal. You should see four tables with 15 records each. Code is available in Appendix A: Code Listings.
2. Validate that the tables exist and are populated with the following queries:

select table\_name

from information\_schema.tables

where table\_schema = 'public'

and table\_catalog = 'salestracker';

select count(\*) from <table\_name>;

You should see four tables each with 15 records.

1. A listing of the entire contents of the tables is available in Figure 7: Full Data Listing

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Full Data Listing

# Anaconda Install

## Deployment Architecture

I’m going to use a Docker container to house the Python execution environment. This simplifies the configuration – no polluting the host environment or virtual environment complexities. We’ll connect to the Anaconda container with Visual Studio’s container development extensions (see below).

## Anaconda Docker Configuration

See Postgres Installation above for more information about Docker. We’re going to use a similar procedure here.

### Docker Execution Script

As with Postgres above, I have a script that downloads an image, configures a container, and starts the container.

#!/bin/bash

# Variables

CONTAINER\_NAME="anaconda\_data\_engineering"

IMAGE\_NAME="continuumio/anaconda3:latest"

MOUNT\_PATH\_ON\_HOST="/Users/jsperson"

MOUNT\_PATH\_IN\_CONTAINER="/home/jsperson/host"

# Check if the container exists

CONTAINER\_ID=$(docker ps -a -q -f name=$CONTAINER\_NAME)

# If the container does not exist, create and run it

if [ -z "$CONTAINER\_ID" ]; then

echo "Container does not exist. Creating and starting..."

docker run -d \

--name $CONTAINER\_NAME \

-v $MOUNT\_PATH\_ON\_HOST:$MOUNT\_PATH\_IN\_CONTAINER \

$IMAGE\_NAME \

tail -f /dev/null

else

# Check if the container is running

RUNNING\_CONTAINER\_ID=$(docker ps -q -f name=$CONTAINER\_NAME)

if [ -z "$RUNNING\_CONTAINER\_ID" ]; then

echo "Container exists but is stopped. Starting..."

docker start $CONTAINER\_NAME

else

echo "Container is already running."

fi

fi

# Connect to the container

echo "Connecting to container..."

docker exec -it $CONTAINER\_NAME bash

### Script Instructions

1. Use the same working directory created for the project in the Postgres install to store this script.
2. Using a text editor copy the contents of the script in its entirety into a file. I named my file run\_docker\_anaconda.sh.
3. Modify the mount points as appropriate for your environment (most likely changing the name from jsperson to your login name).
4. At the command prompt in your working directory change permissions of the script created above to allow execution. Use something like: chmod 700 <script name>
5. Execute the script by typing: ./<script name>
6. During execution the script:
   1. Downloads the Anaconda image – see Figure 4: Postgres Image
   2. Creates a container using the image – see Figure 5: Running Postgres Container
   3. Adds a mount path from the container to my host home directory
   4. Starts the container
   5. Connects to the container in a shell environment
7. After running the script you’ll be at the command prompt inside of the container. Validate proper installation by running anaconda --version at the command line. The output should be similar to Figure 8: Anaconda version confirmation.

A screen shot of a computer

AI-generated content may be incorrect.

Figure : Anaconda version confirmation

# Docker Network Configuration

We’re running a couple of Docker containers here. One of the cool things about Docker is that we can join them into their own network. This will allow the two to talk directly without relying on the host’s network.

In a local terminal window Enter the following commands:

docker network create <network name>

docker network connect <network name> <postgres container name>

docker network connect <network name> <anaconda container name>

A computer screen with white text

AI-generated content may be incorrect.

Figure :Docker Network Setup

Use the command docker network inspect <network name> to see the configuration of the network. Note that containers must be running in order to show up in the network. They do not need to be running to be added.

A computer screen shot of white text

AI-generated content may be incorrect.

Figure : Sample Docker Network Inspect Output – Note two containers

# VS Code Install and Configuration

## VS Code Installation

These steps are specifically for MacOS but other operating systems installations should be quite similar.

1. In a web browser navigate to <https://code.visualstudio.com/download>
2. Select the link for the appropriate install. This could be an in-depth discussion depending on what Mac you have. I’m using the Apple Silicon dmg.
3. Assuming downloaded the dmg, find it in the Downloads folder and click on it. You’ll see a window like Figure 11: VS Code Install Screen. Drag Visual Studio Code over to the Applications folder.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : VS Code Install Screen

1. Start VS Code from the Applications folder.
2. In VS Code, on the left-hand side of VS Code, Select the Extensions icon (see Figure 12.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Extension icon indicated by arrow

1. Install the following Extensions:
   1. Python (Python debugger comes with it)
   2. Dev Containers

# Python Data Pipeline Setup

These steps are a bit sparser than the above. My assumption is that anyone doing development in containers knows at least a bit about VS Code. Since I believe the goal here is to prove knowledge and to provide a reference in the future I think this is sufficient.

1. If it’s not already running, start Docker Desktop
2. Start both the Postgres and the Anaconda containers
3. Open VS Code
4. Click on the remote explorer in VS Code (it’s the icon right above the Extensions icon)
5. Click on the “Attach in Current Window” arrow as indicated by the red box and finger pointer in Figure 13.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Attaching to the Anaconda container

1. Open a folder of your choice. I’m using a folder on the host via the mounted directory that is a github repo.
2. Create a new python file in that folder.
3. Make sure you’re running with the conda python3.
4. pip install psycopg2-binary
5. Grab the script from the Canvas page (See Appendix A: Code Listings) and copy it to your file.
6. On line 52 change the host name to the name of your Postgres Docker container. See Figure 14: Extract Script Executed for a screenshot of the modified and executed script. The extracted data files can be seen in the left hand file explorer pane. Data files and scripts are available here: <https://github.com/jsperson/newman_data_engineering> (Extract script and data files are in the “extract” folder at the top of the repo.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Extract Script Executed

# Apache Nifi Install (Ravi, 2023)

These steps are specifically for MacOS but other operating systems installations should be quite similar.

## Nifi Docker Configuration

See Postgres Installation above for more information about Docker. We’re going to use a similar procedure here.

### Docker Execution Script

As with Postgres above, I have a script that downloads an image, configures a container, and starts the container.

#!/bin/bash

# Variables

CONTAINER\_NAME="nifi\_data\_engineering"

IMAGE\_NAME="apache/nifi:latest"

PORT="8443"

#MOUNT\_PATH\_ON\_HOST="/Users/jsperson"

#MOUNT\_PATH\_IN\_CONTAINER="/home/jsperson/host"

# Check if the container exists

CONTAINER\_ID=$(docker ps -a -q -f name=$CONTAINER\_NAME)

# If the container does not exist, create and run it

if [ -z "$CONTAINER\_ID" ]; then

echo "Container does not exist. Creating and starting..."

docker run -d \

--name $CONTAINER\_NAME \

-p $PORT:$PORT \

$IMAGE\_NAME

else

# Check if the container is running

RUNNING\_CONTAINER\_ID=$(docker ps -q -f name=$CONTAINER\_NAME)

if [ -z "$RUNNING\_CONTAINER\_ID" ]; then

echo "Container exists but is stopped. Starting..."

docker start $CONTAINER\_NAME

else

echo "Container is already running."

fi

fi

# Connect to the container

echo "Connecting to container..."

docker exec -it $CONTAINER\_NAME /bin/bash

1. See the Postgres and Anaconda instructions for background in Docker and the related Docker scripting.
2. Use the script above as an example. As with previous scripts it downloads an image, creates a container, and starts the container. It is intended for normal use of the Nifi container.

A computer screen shot of a computer program

AI-generated content may be incorrect.

Figure : Initial failure to start (Docker not running) then success downloading image

1. Add the Nifi container to the Docker network:

docker network connect <network name> <nfie container name>

1. Point your web browser to <https://localhost:8443/nifi>
2. Accept any warnings (these are due to self-signed https certs)
3. A web page prompting for username and password indicates success.
4. Retrieve the Nifi username and password:

docker logs -f <nifi container name> 2>&1 | grep -A 1 "Generated"

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Successful Nifi Login

# Apache Nifi Usage (Ravi, 2023)

## Flow Configuration

In this section we walk through configuring our first Nifi flow. This flow consists of a static file generator and a file writer process called GenerateFlowFile and PutFile respectively. This flow will generate a single file every minute it is running.

1. Start the Nifi container.
2. Navigate to <https://localhost:8443/nifi>
3. Login using the Nifi credentials. You’ll land on page similar to that shown in Figure 16: Successful Nifi Login.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Nifi canvas with Processor icon indicated

1. Using the Processor icon indicated in Figure 17, add a GenerateFlowFile and a PutFile processor onto the canvas.
2. Configure the GenerateFlowFile (double click on the respective box):
   1. Settings/Name: <Name the processor something reasonable>
   2. Scheduling: No changes but note CRON and Timer driven options.
   3. Properties/File Size: I believe that these are informative to the orchestrator.
   4. Properties/Custom Text: Change to contain some text that will be inserted into each output file.
   5. Relationships: No changes or notes
   6. Comments: Add a comment describing this processor
   7. Click Apply
3. Create a connection between GenerateFlowFile and PutFile by hovering over the former, putting the pointer on the connector icon that appears, and then dragging to the PutFile box. Figure 17 shows the connector between the two processors.
4. Configure PutFile (double click on the respective box):
   1. Settings/Name: <Name the processor something reasonable>
   2. Scheduling/Concurrent Tasks: This allows parallel execution when the setting is greater than one and there is available input. Leave at 1 for now.
   3. Properties/Directory: Give this a path location **on the Nifi container** where it can write files.
   4. Properties/Create Missing Directories: I set this to false and created the folder myself. This way if Nifi can’t write to the folder or it tries to write in an unexpected location an error is likely. This minimizes the chance of silent failure.
   5. Relationships: Set both options to terminate. The PutFile is the last processor in the flow so it is reasonable to terminate on either failure or success.
   6. Comments: Add a comment describing this processor
   7. Click Apply

## Flow Operations

Once the flow is configured, we can walk through some of the operational capabilities of the Nifi interface.

1. To start a flow, click on the canvas to ensure that the Operation panel indicates “Nifi Flow” then click the start button indicated by the arrow in Figure 18. Note that this screenshot shows the flow partially stopped. We discuss this state below.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 18: Flow with PutFile Stopped

1. Right clicking on a component provides access to “View Status History”. This exposes various selectable metrics and timeframes presented in a graph (See Figure 19).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 19: Metric View

1. A rather unique (in my experience) feature of Nifi is the ability to stop individual flow components. In Figure 18 you can see how the PutFile is stopped (indicated by the stop icon in the processor box. While PutFile is stopped, GenerateFlowFile remains running and its output is queued in the connection TransferGeneratedFile. The figure calls out the connection with a red box where you can observe four files are queued. Restarting the PutFile causes the queue to be rapidly emptied as the PutFile processor writes the files to the filesystem. Figure 20 shows the file output before and after the partial stop. Observe files written every minute up to 1651 where there was a five minute PutFile stop then five files written at 1656 the a subsequent return to the one per minute cadence.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure : File listing of flow output

# Sources

Ravi, J. (2023). *Data management with Apache NiFi* [Online course]. LinkedIn Learning. <https://www.linkedin.com/learning/data-management-with-apache-nifi>

# Appendix A: Code Listings

## Create Tables SQL

-- Create the products table  
CREATE TABLE products (  
    product\_id SERIAL PRIMARY KEY,  
    product\_name VARCHAR(100) NOT NULL,  
    category VARCHAR(50),  
    price DECIMAL(10, 2) NOT NULL,  
    stock INT NOT NULL  
);  
  
-- Create the customers table  
CREATE TABLE customers (  
    customer\_id SERIAL PRIMARY KEY,  
    first\_name VARCHAR(50),  
    last\_name VARCHAR(50),  
    email VARCHAR(100) UNIQUE NOT NULL,  
    join\_date DATE NOT NULL  
);

-- Create the orders table  
CREATE TABLE orders (  
    order\_id SERIAL PRIMARY KEY,  
    customer\_id INT NOT NULL,  
    order\_date DATE NOT NULL,  
    total\_amount DECIMAL(10, 2) NOT NULL,  
    FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)  
);  
  
-- Create the order\_items table  
CREATE TABLE order\_items (  
    order\_item\_id SERIAL PRIMARY KEY,  
    order\_id INT NOT NULL,  
    product\_id INT NOT NULL,  
    quantity INT NOT NULL,  
    total\_price DECIMAL(10, 2) NOT NULL,  
    FOREIGN KEY (order\_id) REFERENCES orders(order\_id),  
    FOREIGN KEY (product\_id) REFERENCES products(product\_id)  
);

## Insert Data SQL

-- Insert sample products  
INSERT INTO products (product\_name, category, price, stock) VALUES  
('Laptop', 'Electronics', 999.99, 50),  
('Smartphone', 'Electronics', 699.99, 200),  
('Headphones', 'Accessories', 49.99, 500),  
('Desk Chair', 'Furniture', 150.00, 100),  
('Coffee Mug', 'Kitchen', 10.99, 1000),  
('Bluetooth Speaker', 'Electronics', 79.99, 150),  
('Keyboard', 'Accessories', 39.99, 300),  
('Gaming Console', 'Electronics', 499.99, 75),  
('Bookshelf', 'Furniture', 120.00, 80),  
('Mouse', 'Accessories', 25.99, 400),  
('Tablet', 'Electronics', 329.99, 100),  
('Monitor', 'Electronics', 199.99, 120),  
('Desk Lamp', 'Furniture', 45.00, 200),  
('Thermos', 'Kitchen', 24.99, 500),  
('Dining Set', 'Furniture', 499.99, 30);  
  
-- Insert sample customers  
INSERT INTO customers (first\_name, last\_name, email, join\_date) VALUES  
('Alice', 'Johnson', 'alice.johnson@example.com', '2023-01-15'),  
('Bob', 'Smith', 'bob.smith@example.com', '2023-02-20'),  
('Charlie', 'Brown', 'charlie.brown@example.com', '2023-03-10'),  
('Dana', 'White', 'dana.white@example.com', '2023-04-05'),  
('Eve', 'Williams', 'eve.williams@example.com', '2023-05-10'),  
('Frank', 'Miller', 'frank.miller@example.com', '2023-06-01'),  
('Grace', 'Davis', 'grace.davis@example.com', '2023-06-15'),  
('Henry', 'Wilson', 'henry.wilson@example.com', '2023-07-01'),  
('Ivy', 'Taylor', 'ivy.taylor@example.com', '2023-07-20'),  
('Jack', 'Moore', 'jack.moore@example.com', '2023-08-05'),  
('Kara', 'Hall', 'kara.hall@example.com', '2023-08-25'),  
('Liam', 'Young', 'liam.young@example.com', '2023-09-10'),  
('Mia', 'King', 'mia.king@example.com', '2023-09-25'),  
('Noah', 'Scott', 'noah.scott@example.com', '2023-10-05'),  
('Olivia', 'Green', 'olivia.green@example.com', '2023-10-15');  
  
-- Insert expanded sample orders  
INSERT INTO orders (customer\_id, order\_date, total\_amount) VALUES  
(1, '2023-05-01', 1149.97),  -- Customer 1  
(2, '2023-05-03', 49.99),    -- Customer 2  
(3, '2023-05-07', 1599.98),  -- Customer 3  
(4, '2023-05-10', 150.00),   -- Customer 4  
(1, '2023-05-15', 2149.94),  -- Customer 1  
(2, '2023-05-20', 349.99),   -- Customer 2  
(3, '2023-05-25', 2799.95),  -- Customer 3  
(4, '2023-06-01', 210.00),   -- Customer 4  
(2, '2023-06-05', 1149.97),  -- Customer 2  
(3, '2023-06-10', 599.97),   -- Customer 3  
(4, '2023-06-15', 450.00),   -- Customer 4  
(1, '2023-06-20', 999.99),   -- Customer 1  
(2, '2023-06-25', 799.98),   -- Customer 2  
(3, '2023-07-01', 499.99),   -- Customer 3  
(4, '2023-07-05', 1000.00);  -- Customer 4  
  
-- Insert expanded sample order items  
INSERT INTO order\_items (order\_id, product\_id, quantity, total\_price) VALUES  
(1, 1, 1, 999.99),    -- Order 1: Laptop  
(1, 3, 3, 149.97),    -- Order 1: Headphones  
(2, 3, 1, 49.99),     -- Order 2: Headphones  
(3, 2, 2, 1399.98),   -- Order 3: Smartphone  
(3, 5, 5, 54.95),     -- Order 3: Coffee Mug  
(4, 4, 1, 150.00),    -- Order 4: Desk Chair  
(5, 1, 2, 1999.98),   -- Order 5: Laptop  
(5, 3, 4, 199.96),    -- Order 5: Headphones  
(6, 5, 10, 109.90),   -- Order 6: Coffee Mug  
(7, 2, 1, 699.99),    -- Order 7: Smartphone  
(7, 4, 1, 150.00),    -- Order 7: Desk Chair  
(8, 3, 5, 249.95),    -- Order 8: Headphones  
(8, 5, 7, 76.93),     -- Order 8: Coffee Mug  
(9, 2, 3, 2099.97),   -- Order 9: Smartphone  
(9, 4, 2, 300.00);    -- Order 9: Desk Chair

## Extract Script Python

import pandas as pd

import psycopg2

# Function to extract data from a PostgreSQL table

def extract\_table\_to\_csv(db\_config, table\_name, csv\_file):

try:

# Establish database connection

conn = psycopg2.connect(

host=db\_config['host'],

port=db\_config['port'],

database=db\_config['database'],

user=db\_config['user'],

password=db\_config['password']

)

print(f"Connected to the database for table: {table\_name}")

# Define the SQL query

query = f"SELECT \* FROM {table\_name};"

# Execute query and load data into a pandas DataFrame

data = pd.read\_sql\_query(query, conn)

# Export DataFrame to CSV

data.to\_csv(csv\_file, index=False)

print(f"Data from {table\_name} exported to {csv\_file}.")

# Close the connection

conn.close()

except Exception as e:

print(f"Error extracting data from {table\_name}: {e}")

raise

# Main function to orchestrate the pipeline

def export\_sales\_tracker\_tables\_to\_csv(db\_config):

print("Starting the data export pipeline...")

# Define table-to-CSV mapping

tables\_to\_export = {

"products": "products\_data.csv",

"customers": "customers\_data.csv",

"orders": "orders\_data.csv",

"order\_items": "order\_items\_data.csv"

}

# Export each table

for table\_name, csv\_file in tables\_to\_export.items():

extract\_table\_to\_csv(db\_config, table\_name, csv\_file)

print("Data export pipeline completed.")

# Example usage

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

# Database configuration

db\_config = {

"host": "postgres\_data\_engineering",

"port": 5432,

"database": "salestracker",

"user": "admin",

"password": "admin"

}

# Execute the pipeline

export\_sales\_tracker\_tables\_to\_csv(db\_config)