



(Practice) Docker

[1. Intro to Docker](#)

[2. Ingesting NY Taxi Data to Postgres](#)

[3. Connecting pgAdmin and Postgres with Docker networking](#)

[4. Dockerizing the Ingestion Script](#)

[5. Running Postgres and pgAdmin with Docker-Compose](#)

▼ 1. Intro to Docker

▼ 1.1. Familiarisasi dengan Docker

Creating first container

Lets create an interactive container using the Python 3.9 image, providing a terminal where you can run Python commands

In a new terminal run the following command:

```
docker run -it python:3.9
```

docker run: This creates and starts a new Docker container from the specified image.

- **i (interactive):** Keeps the standard input (stdin) of the container open, allowing you to interact with it.
- **t:** Allocates a pseudo-TTY (a terminal), making the interaction more user-friendly

python:3.9: Specifies the Docker image to use, in this case, the official Python image with version 3.9

You should get the python prompt:

```
>>>
```

and write any python code inside the container

When you run this command, Docker will:

Pull the python:3.9 image from Docker Hub if it's not already available locally. Start a container using this image. Launch an interactive Python 3.9 shell (REPL) inside the container, allowing you to execute Python commands directly. This is commonly used for testing or experimenting with Python in an isolated environment.

Pertama, misalnya kita ingin menjalankan kontainer yang menggunakan ubuntu. Kita bisa menggunakan perintah berikut di terminal.

```
docker run -it ubuntu bash
```

Selanjutnya misal ingin menjalankan docker yang berisi python, kita bisa menggunakan

```
docker run -it python:3.9
```

Perintah diatas akan langsung membawa kita ke python. Misal kita ingin masuk ke bash, kita bisa update scriptnya menjadi

```
docker run -it --entrypoint=bash python:3.9
```

1.2. Docker Workflow

Example code

you want to deploy in Container

```
import sys
import pandas as pd

print(sys.argv)
day = sys.argv[1]

# some pandas things


print(f'Finished for day {day}')
```

Dockerfile

Dockerfile adalah file skrip yang berisi instruksi untuk membangun **Docker Image**. Dockerfile menentukan lingkungan yang dibutuhkan, paket yang

harus diinstal, dan perintah yang akan dijalankan saat container berjalan.

<https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/> - Dockerfile Instruction options i.e. 'FROM', 'RUN', e

 Create a new 'Dockerfile' in your code editor. I recommend adding the Docker 'plug in' to your editor.

```
# Menggunakan image dasar Python versi 3.12.8 dari Docker Hub
FROM python:3.12.8

# Menjalankan perintah untuk menginstall library pandas menggunakan pip
RUN pip install pandas

# Menentukan direktori kerja di dalam container. Semua perintah selanjutnya
WORKDIR /app

# Menyalin file `pipeline.py` dari host (komputer lokal) ke dalam direktori `/app`
COPY pipeline.py pipeline.py

# Menentukan perintah default yang akan dijalankan ketika container di-start
# Dalam hal ini, perintahnya adalah menjalankan script Python `pipeline.py`
ENTRYPOINT ["python", "pipeline.py"]
```

Docker Image

Docker Image adalah blueprint atau template untuk membuat container. Setelah kita menulis Dockerfile, kita perlu membangun Docker Image menggunakan perintah:

 Terminal

```
docker build -t {image_name}:{tag_name} .
```

-t	{image_name}:{tag_name}	.
used to denote adding a tag	replace these values for your image_name and tag_name	a build command that uses the current directory (<code>.</code>) as a build context

<https://docs.docker.com/get-started/docker-concepts/building-images/build-tag-and-publish-an-image/>

```
D:\DE-Zoomcamp-Practice\DE-Zoomcamp-DwiANS-2025\Week_1\DOCKER>docker build -t first_image:test .
[+] Building 69.7s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 676B
=> [internal] load metadata for docker.io/library/python:3.12.8
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.12.8@sha256:e74938514dc67ad3eade8798aa929f5dd569e46
=> => resolve docker.io/library/python:3.12.8@sha256:e74938514dc67ad3eade8798aa929f5dd569e46
=> => sha256:dc93fee027280c5d4750ac4ffad892da8df66dbd396073e95abec4b78ddb5736 248B / 248B
=> => sha256:a113cc029c6057705abf993d8d6c4c3cdd4518e6ee0aabde098241839ff32 25.64MB / 25.64MB
=> => sha256:a57a7d4db73564480295d074a3b484ef4313934c2ad7d5a9fa907ecf79b97496 6.16MB / 6.16MB
=> => sha256:7576b00d9bb10cc967bb5bdeeb3d5fa078ac8800e112aa03ed15ec19966 211.33MB / 211.33MB
=> => sha256:35af2a7690f2b43e7237d1fae8e3f2350dfb25f3249e9cf65121866f9c56c 64.39MB / 64.39MB
=> => sha256:32b550becb62359a0f3a96bc0dc289f8b45d097eaad275887f163c6780b41 24.06MB / 24.06MB
```

Container

Container adalah instansi dari Docker Image yang sedang berjalan. Setelah image dibuat, kita bisa menjalankannya sebagai container:

Terminal

```
docker run -it {image_name}:{tag_name}
```

```
D:\DE-Zoomcamp-Practice\DE-Zoomcamp-DwiANS-2025\Week_1\DOCKER>docker run -it first_image:test 13-2-2025
25 Coba Anjay
['pipeline.py', '13-2-2025', 'Coba', 'Anjay']
Finished for day 13-2-2025
```

▼ 2. Ingesting NY Taxi Data to Postgres

You can run a containerized version of Postgres that doesn't require any installation steps. You only need to provide a few environment variables to it as well as a volume for storing data.

Create a folder anywhere you'd like for Postgres to store data in (locally). We will use the example folder `ny_taxi_postgres_data`.

2.1. Download Containerized Postgres

Run the following command to start up your instance (bash/ubuntu wsl terminal):

```
docker run -it \
-e POSTGRES_USER="root" \
-e POSTGRES_PASSWORD="root" \
-e POSTGRES_DB="ny_taxi" \
```

```
-v (pwd)/ny_taxi_postgres_data:/var/lib/postgresql/data \
-p 5432:5432 \
postgres:13
```

VERSI WINDOWS:

```
docker run -it -e POSTGRES_USER="root" -e POSTGRES_PASSWORD="root" postgres:13
```

Explanation:

Command di atas berfungsi untuk menjalankan sebuah container PostgreSQL (versi 13) berdasarkan image resmi dari Docker Hub. Saat pertama kali perintah ini dijalankan dan image PostgreSQL belum ada di Docker Desktop, Docker akan secara otomatis melakukan proses *pull* untuk mengunduh image tersebut dari Docker Hub. Setelah image berhasil diunduh, setiap kali Anda menjalankan perintah yang sama (misalnya di hari berikutnya), Docker akan langsung membuat dan menjalankan container dari image yang sudah tersimpan secara lokal tanpa perlu mengunduh ulang.

Selain itu, perintah ini juga mengatur beberapa konfigurasi penting:

- **Environment Variables:**

Menetapkan user (`POSTGRES_USER`), password (`POSTGRES_PASSWORD`), dan nama database (`POSTGRES_DB`) yang akan digunakan oleh PostgreSQL.

- **Volume Mapping:**

Opsi `-v "%cd%/ny_taxi_postgres_data:/var/lib/postgresql/data"` memastikan data database disimpan di direktori lokal sehingga data tersebut tetap ada walaupun container dihentikan atau dihapus.

- **Port Mapping:**

Opsi `-p 5432:5432` memetakan port 5432 di container ke port 5432 di host, sehingga Anda dapat mengakses database PostgreSQL melalui `localhost:5432`.

Dengan konfigurasi tersebut, Anda dapat dengan mudah mengelola database PostgreSQL di dalam Docker dan melakukan pengujian atau pengembangan sesuai kebutuhan tanpa harus mengunduh image setiap kali.

2.2. CLI for Postgres

pgcli is used to interact with a PostgreSQL database from the command line, with a focus on improving usability with features like autocompletion, syntax highlighting, and an enhanced user experience.

Installing pgcli

```
pip install pgcli
```

Once the container is running, open up another terminal and log into our database with the following command:

```
python -m pgcli -h localhost -p 5432 -U root -d ny_taxi
```

Lets test the db for example with "SELECT 1;", and it should print:

```
ny_taxi=# select 1;
?column?
-----
      1
(1 row)
```

Of course we haven't loaded data into the DB yet.

2.3. Running Postgres Image

Untuk menggunakan PostgreSQL dalam Docker, ada beberapa hal yang perlu dipahami:

1. Perbedaan Image dan Container

- **Image** adalah blueprint atau template yang berisi semua yang dibutuhkan untuk menjalankan aplikasi (dalam hal ini PostgreSQL).
- **Container** adalah instance yang berjalan dari image tersebut.

Jadi, hanya memiliki image di Docker Desktop tidak berarti PostgreSQL sudah berjalan. Kamu harus menjalankan (instantiate) container dari image itu.

2. Menggunakan Docker Desktop

Di Docker Desktop, kamu bisa melihat image yang telah kamu download. Namun, untuk menggunakannya kamu harus membuat container baru.

- Kamu bisa membuat container baru melalui UI Docker Desktop dan mengkonfigurasinya dengan environment variables (seperti POSTGRES_PASSWORD).
- Jika kamu hanya klik "Run" pada image tanpa menambahkan konfigurasi tersebut, maka container tidak akan inisialisasi dengan benar dan menampilkan error seperti yang kamu lihat.

2.4. Running the Same Container (Reuse container)

Setiap kali kamu menjalankan perintah `docker run`, Docker akan membuat **container baru** dari image yang kamu tentukan. Jadi kalau kamu menjalankan perintah yang sama besok, itu akan membuat **container baru lagi**, bukan menggunakan container lama.

Namun, kalau kamu ingin menggunakan **container yang sudah ada**, kamu tidak perlu menjalankan `docker run` lagi. Sebagai gantinya, kamu bisa:

1. Cek container yang sudah berjalan

```
docker ps
```

Kalau container PostgreSQL sudah berjalan, kamu tidak perlu menjalankan ulang.

2. Cek container yang pernah dibuat (termasuk yang sudah berhenti)

```
docker ps -a
```

Kalau ada container PostgreSQL yang pernah dibuat tapi tidak berjalan, kamu bisa menjalankannya lagi dengan:

```
docker start <CONTAINER_ID atau NAMA>
```

3. Gunakan `docker stop` dan `docker start` daripada `docker run`

- Untuk menghentikan container:

```
docker stop <CONTAINER_ID>
```

- Untuk memulai lagi container yang sama:

```
docker start <CONTAINER_ID>
```

4. Gunakan `docker-compose` jika ingin lebih mudah mengelola container

Kalau kamu sering menjalankan container yang sama, lebih baik pakai `docker-compose.yml`.

Jadi kalau kamu ingin **tetap pakai container yang sudah ada**, cukup gunakan `docker start` daripada `docker run` lagi. 🚀

2.5. Generating postgres schema

We will use data from the NYC TLC Trip Record Data website. Specifically, we will use `yellow_tripdata_2021-01.csv`

from: <https://github.com/DataTalksClub/nyc-tlc-data/releases/tag/yellow>

1. Download it, unzip it and move the csv file to the working directory
2. Create an `ingest_data.py` file that reads the csv file and generates the schema for the Postgres database:

Install Pandas and SQLAlchemy with:

```
pip install pandas
pip install sqlalchemy
```

Open Vs code and write the following code in `ingest_data.py`:

```
import pandas as pd
from sqlalchemy import create_engine

df = pd.read_csv('yellow_tripdata_2021-01.csv', nrows=100)
df.tpep_pickup_datetime = pd.to_datetime(df.tpep_pickup_datetime)
df.tpep_dropoff_datetime = pd.to_datetime(df.tpep_dropoff_datetime)

engine = create_engine('postgresql://root:root@localhost:5433/ny_taxi')

print(pd.io.sql.get_schema(df, name='yellow_taxi_data', con=engine))
```


pandas is used for data manipulation and **sqlalchemy** provides tools for database interaction, including creating connections to databases.

The code reads the first 100 rows of `yellow_tripdata_2021-01.csv` into a Pandas DataFrame called `df`. The columns `tpep_pickup_datetime` and `tpep_dropoff_datetime` in the DataFrame are converted from string format to Pandas datetime objects for easier time-based operations.

engine =

create_engine('postgresql://root:root@localhost:5433/ny_taxi') creates a connection to the database we created in the previous section and since we are accessing from the host machine, we use port 5433.

The code prints the SQL schema that would represent the `df` DataFrame if it were stored in the database table `yellow_taxi_data`. This includes the structure of the table, column names, and data types.

2.6. Ingesting data to Postgres with Python

```
import pandas as pd
from sqlalchemy import create_engine

df_iter = pd.read_csv('yellow_tripdata_2021-01.csv', iterator=True, chunksize=1000)

df = next(df_iter)
df.tpep_pickup_datetime = pd.to_datetime(df.tpep_pickup_datetime)
df.tpep_dropoff_datetime = pd.to_datetime(df.tpep_dropoff_datetime)

engine = create_engine('postgresql://root:root@localhost:5433/ny_taxi')

# Create table
df.head(n=0).to_sql(name='yellow_taxi_data', con=engine, if_exists='replace')

# Insert first chunk
df.to_sql(name='yellow_taxi_data', con=engine, if_exists='append')

# Insert remaining data
while True:
    try:
```

```

df = next(df_iter)
df.tpep_pickup_datetime = pd.to_datetime(df.tpep_pickup_datetime)
df.tpep_dropoff_datetime = pd.to_datetime(df.tpep_dropoff_datetime)
df.to_sql(name='yellow_taxi_data', con=engine, if_exists='append')

print('inserted another chunk')
except StopIteration:
    print('completed')
    break

```

1. create the table:

```

df.head(n=0).to_sql(name='yellow_taxi_data', con=engine, if_exists='replace')

```

This operation is used to create an empty table with the same column names and data types as the DataFrame df. No actual data from the DataFrame is inserted into the table.

The method `df.head(n=0)` creates a DataFrame that contains only the column headers from the df DataFrame without any row data.

- The `to_sql` method writes the DataFrame to a table in the database.
- `name='yellow_taxi_data'`: Specifies the name of the table to be created in the database.
- `con=engine`: Specifies the database connection (created using sqlalchemy earlier).
- `if_exists='replace'`: If a table named `yellow_taxi_data` already exists, it will be replaced with this new table structure.

2. Insert first chunk of data:

Reading and processing the file in chunks prevents memory overload when working with large datasets.

```

df_iter = pd.read_csv('yellow_tripdata_2021-01.csv', iterator=True, chunksize=100000)

df = next(df_iter)
df.tpep_pickup_datetime = pd.to_datetime(df.tpep_pickup_datetime)

```

```
df.tpep_dropoff_datetime = pd.to_datetime(df.tpep_dropoff_datetime)
df.to_sql(name='yellow_taxi_data', con=engine, if_exists='append')
```

- The `pd.read_csv` method is used to create an iterator for the file `yellow_tripdata_2021-01.csv`.
- The `iterator=True` parameter allows the file to be read in chunks rather than loading the entire file into memory.
- The `chunksize=100000` parameter specifies that each chunk contains 100,000 rows.
- The `next(df_iter)` function retrieves the first chunk from the iterator and stores it in the variable `df`.
- The `to_sql` method appends the chunk `df` to a table named `yellow_taxi_data` in the connected database. `if_exists='append'`: Appends the data to the table if it already exists; if not, the table is created.

3. Loop to write all the remaining data to the database:

```
while True:
    try:

        df = next(df_iter)
        df.tpep_pickup_datetime = pd.to_datetime(df.tpep_pickup_datetime)
        df.tpep_dropoff_datetime = pd.to_datetime(df.tpep_dropoff_datetime)
        df.to_sql(name='yellow_taxi_data', con=engine, if_exists='append')

        print('inserted another chunk')
    except StopIteration:
        print('completed')
        break
```

```
SELECT count(1) FROM yellow_taxi_data;
```

It should print:

```

root@localhost:ny_taxi> SELECT count(1) FROM yellow_taxi_data;
+-----+
| count |
+-----+
| 1369765 |
+-----+
SELECT 1
Time: 0.880s
root@localhost:ny_taxi> |

```

▼ 3. Connecting pgAdmin and Postgres with Docker networking

pgAdmin is a graphical user interface (GUI) tool for managing PostgreSQL databases. It provides a visual way to interact with databases, visual exploration of database schemas, perform queries, and administer database objects like tables, schemas, users, and permissions.

Use pgAdmin if you prefer a visual interface or need to perform advanced database management tasks with a GUI. Use pgcli if you prefer working in the terminal and executing SQL commands.

It's possible to run pgAdmin as a container along with the Postgres container, but both containers will have to be in the same virtual network so that they can find each other.

3.1. Create Network

Let's create a virtual Docker network called pg-network:

```
docker network create pg-network
```

3.2. Run pg-admin and postgres image

We will now re-run our Postgres container with the added network name and the container network name, so that the pgAdmin container can find it (we'll use pg-database for the container name).

Langkah Operasional:

1. Pertama kali (Inisialisasi):

```
# Buat network jika belum ada
docker network create pg-network
```

```
# Jalankan Postgres container (membuat container baru)
```

```
docker run -it \\\n  -e POSTGRES_USER="root" \\\n  -e POSTGRES_PASSWORD="root" \\\n  -e POSTGRES_DB="ny_taxi" \\\n  -v "D:/ny_taxi_postgres_data:/var/lib/postgresql/data" \\\n  -p 5432:5432 \\\n  --network=pg-network \\\n  --name pg-database \\\n  postgres:13
```

```
# Di CMD:
```

```
docker run -it -e POSTGRES_USER="root" -e POSTGRES_PASSWORD\n="root" -e POSTGRES_DB="ny_taxi" -v "D:/ny_taxi_postgres_data:/var/li\nb/postgresql/data" -p 5432:5432 --network=pg-network --name pg-da\ntabase postgres:13
```

```
# Jalankan pgAdmin container (di terminal lain)
```

```
docker run -it \\\n  -e PGADMIN_DEFAULT_EMAIL="admin@admin.com" \\\n  -e PGADMIN_DEFAULT_PASSWORD="root" \\\n  -p 8080:80 \\\n  --network=pg-network \\\n  --name pgadmin \\\n  dpage/pgadmin4
```

```
# Di CMD:
```

```
docker run -it -e PGADMIN_DEFAULT_EMAIL="admin@admin.com" -e P\nGADMIN_DEFAULT_PASSWORD="root" -p 8080:80 --network=pg-netw\nork --name pgadmin dpage/pgadmin4
```

1. Setelah Container Terbentuk:

```
# Hanya perlu start container yang sudah ada
docker start pg-database # Untuk Postgres
docker start pgadmin     # Untuk pgAdmin
```

Penjelasan Data Synchronization:

1. Volume Mapping Postgres (`v "D:/ny_taxi..."`):

- Data database akan otomatis tersimpan di folder lokal
`D:/ny_taxi_postgres_data`
- **Pastikan direktori ini tidak diubah** saat menjalankan container
- Tidak perlu menjalankan container dari direktori ini, yang penting path mapping-nya konsisten

2. Kapan Harus Jalankan di Direktori Volume?

- Tidak perlu, karena volume mapping menggunakan path absolut
- Volume akan tetap bekerja dimanapun kamu menjalankan perintah

```
docker start
```

3. Yang Perlu Diperhatikan:

- Jika menggunakan `docker run` lagi dengan nama container sama, akan error
- Jika ingin membuat container baru, hapus dulu container lama dengan `docker rm`
- Data akan tetap aman selama volume mapping tidak diubah

Flow Operasional Harian:

1. Start container: `docker start pg-database pgadmin`
2. Stop container: `docker stop pg-database pgadmin`
3. Cek status: `docker ps -a`

Dengan cara ini:

- Tidak ada container duplikat
- Data tetap tersinkronisasi dengan lokal
- Konfigurasi network dan volume tetap konsisten

You should now be able to load pgAdmin on a web browser by browsing to localhost:8080. Use the same email and password you used for running the container to log in.

Right-click on Servers on the left sidebar → Register → Server

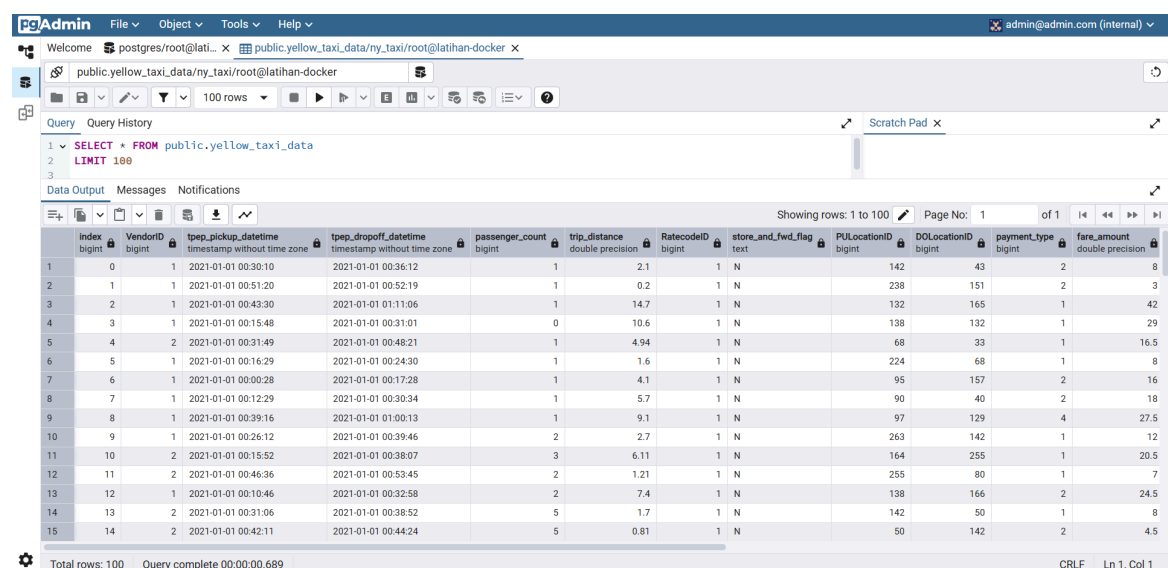
Under General give the Server a name: Docker localhost

Under Connection add the same host name: pg-database, port:5432
user:root and password:root

We use port 5432 because we are accessing from a docker container. If it were the case of accessing from the host machine, it would be port 5433.

In the pgadmin webpp now we can explore the table:

Docker localhost → Databases → ny_taxi → Schemas → Tables → View first 100 rows



The screenshot shows the pgAdmin web interface. The top bar includes the pgAdmin logo and navigation menus. The left sidebar shows the server tree with 'Docker localhost' selected. The main pane displays a query result for the 'public.yellow_taxi_data' table. The query is 'SELECT * FROM public.yellow_taxi_data LIMIT 100'. The result shows 100 rows of taxi trip data. The columns include index, VendorID, tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, RatecodeID, store_and_fwd_flag, PULocationID, DOLocationID, payment_type, and fare_amount. The data is displayed in a table with alternating light and dark rows. The bottom status bar shows 'Total rows: 100' and 'Query complete 00:00:00.689'.

index	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	fare_amount
1	0	2021-01-01 00:30:10	2021-01-01 00:36:12	1	2.1	1	N	142	43	2	8
2	1	2021-01-01 00:51:20	2021-01-01 00:52:19	1	0.2	1	N	238	151	2	3
3	2	2021-01-01 00:43:30	2021-01-01 01:11:06	1	14.7	1	N	132	165	1	42
4	3	2021-01-01 00:15:48	2021-01-01 00:31:01	0	10.6	1	N	138	132	1	29
5	4	2021-01-01 00:31:49	2021-01-01 00:48:21	1	4.94	1	N	68	33	1	16.5
6	5	2021-01-01 00:16:29	2021-01-01 00:24:30	1	1.6	1	N	224	68	1	8
7	6	2021-01-01 00:00:28	2021-01-01 00:17:28	1	4.1	1	N	95	157	2	16
8	7	2021-01-01 00:12:29	2021-01-01 00:30:34	1	5.7	1	N	90	40	2	18
9	8	2021-01-01 00:39:16	2021-01-01 01:00:13	1	9.1	1	N	97	129	4	27.5
10	9	2021-01-01 00:26:12	2021-01-01 00:39:46	2	2.7	1	N	263	142	1	12
11	10	2021-01-01 00:15:52	2021-01-01 00:38:07	3	6.11	1	N	164	255	1	20.5
12	11	2021-01-01 00:46:36	2021-01-01 00:53:45	2	1.21	1	N	255	80	1	7
13	12	2021-01-01 00:10:46	2021-01-01 00:32:58	2	7.4	1	N	138	166	2	24.5
14	13	2021-01-01 00:31:06	2021-01-01 00:38:52	5	1.7	1	N	142	50	1	8
15	14	2021-01-01 00:42:11	2021-01-01 00:44:24	5	0.81	1	N	50	142	2	4.5

▼ 4. Dockerizing the Ingestion Script

https://www.youtube.com/watch?v=B1WwATwf-vY&list=PL3MmuxUbc_hJed7dXYoJw8DoCuVHhGEQb&index=8

👉 In this video we will learn how to convert our jupyter notebook into a python script. Then we will learn a second way of how to ingest our data into postgres using our new python script.

🔄 Recall this information from "2. Ingesting NY Taxi Data to Postgres" where we are loading in our taxi csv data using python notebook. This is another way to achieve the same goal, but likely a more common practice.

4.1. Convert Jupyter Notebook

■ Terminal - Converting the jupyter notebook to a python script

Copy

```
jupyter nbconvert --to=script {notebook_name.ipynb}
```

Run from your project environment in the folder where the ipynb lives

🧹 Clean up your code as needed

4.2. Using argparse

The goal is to allow user inputs for different values such as url or password. You can read more about how to use argparse here

[/docs.python.org/3/library/argparse.html](https://docs.python.org/3/library/argparse.html). The final python script called ingest_data.py can be found [here](#). I **recommend** using this version because at ~11 min in the youtube Alexey mentions needed to add an exception to the code, which this version has - the `'try-except'` statement. There is a link in resources if you are unsure what a `'try-except'` statement is.

4.3. Second way to ingest data

🔄 Recall that our first way was in "2. Ingesting NY Taxi Data to Postgres". To complete the second method you will need to drop your table following the youtube video.

This second way is still a 'manual' method. You need to manually drop your table here <http://localhost:8080/> and then run a command.

🔧 Note that our url link will look different because the taxi ny website no longer has the csv files

■ Terminal

Copy

```
URL="https://github.com/DataTalksClub/nyc-tlc-data/releases/download/yellow/yellow_tripdata_2021-01.csv.gz"
```

```
python ingest_data.py \  
  --user=root \  
  --password=root \  
  --host=localhost \  
  --url=
```



```
--port=5432 \  
--db=ny_taxi \  
--table_name=yellow_taxi_trips \  
--url=${URL}
```

✅ This should print the loops in your terminal window and you should be able to now see the data here <http://localhost:8080/> again (if you dropped your table).

4.4. Third way to ingest data

This is 'Dockerizing' the python script. This method will automatically 'replace' the table according to our python script.

1. Update Dockerfile

 Code editor

```
FROM python:3.12  
  
RUN apt-get install wget  
RUN pip install pandas sqlalchemy psycopg2  
  
WORKDIR /app  
COPY ingest_data.py ingest_data.py  
  
ENTRYPOINT [ "python", "ingest_data.py" ]
```

■ Terminal window

```
docker build -t taxi_ingest:v001 .
```


2. Docker run

```
docker run -it \  
--network=pg-network \  
taxi_ingest:v001 \  
--user=root \  
--password=root \  
--host=pg-database \
```

```
--port=5432 \  
--db=ny_taxi \  
--table_name=yellow_taxi_trips \  
--url=${URL}
```

▼ 5. Running Postgres and pgAdmin with Docker-Compose

Docker Compose Explanation:

 [\(Theory\) Container and Infrastructure as Code](#)

5.1. Create docker-compose.yaml file

```
version: '3.8'  
  
services:  
  pg-database:  
    image: postgres:13  
    container_name: pg-database  
    environment:  
      POSTGRES_USER: root  
      POSTGRES_PASSWORD: root  
      POSTGRES_DB: ny_taxi  
    volumes:  
      - "D:/ny_taxi_postgres_data:/var/lib/postgresql/data"  
    ports:  
      - "5432:5432"  
  
  pg-admin:  
    image: dpage/pgadmin4  
    container_name: pg-admin  
    environment:  
      PGADMIN_DEFAULT_EMAIL: admin@admin.com  
      PGADMIN_DEFAULT_PASSWORD: root  
    ports:  
      - "8080:80"  
    depends_on:  
      - pg-database
```

5.2.2. Run docker compose

```
docker-compose up
```

5.2.3. Stop docker compose

```
docker-compose down
```

5.2.4. Reuse Container using Docker-Compose

Jika Docker Compose sudah dimatikan (`docker-compose down`) dan Anda ingin menyalakannya kembali, Anda tetap menggunakan command `docker-compose up` . Berikut penjelasan lengkapnya:

1. Apakah `docker-compose up` Membuat Container Baru?

- **Tidak**, jika container yang sudah dibuat sebelumnya masih ada (tidak dihapus).
- `docker-compose up` akan **menyalakan kembali container yang sudah ada** dan menggunakan konfigurasi yang sama (image, volume, network, dll).
- Jika container sebelumnya sudah dihapus (misalnya, karena menjalankan `docker-compose down` **tanpa opsi `-v`**), maka `docker-compose up` akan membuat container baru.

2. Bagaimana Menghindari Redundansi Container?

- **Pastikan volume tetap ada:**

Saat menjalankan `docker-compose down` , gunakan opsi `--volumes` **dengan hati-hati**. Jika Anda menghapus volume (`docker-compose down -v`), data akan hilang, dan container baru akan dibuat dengan volume baru.

Contoh:

```
docker-compose down # Hanya menghentikan container, tidak menghapus volume
docker-compose up   # Menyalakan kembali container dengan volume yang sama
```

- **Gunakan `docker-compose start` :**

Jika container sudah ada dan hanya perlu dinyalakan kembali (tanpa membuat ulang), gunakan:

```
docker-compose start
```

Perintah ini hanya menyalakan container yang sudah ada tanpa memeriksa perubahan pada `docker-compose.yml`.

3. Perbedaan `docker-compose up` dan `docker-compose start`

- **`docker-compose up` :**
 - Menjalankan container baru jika belum ada.
 - Memeriksa perubahan pada `docker-compose.yml` dan memperbarui container jika diperlukan.
 - Cocok untuk development saat ada perubahan konfigurasi.
- **`docker-compose start` :**
 - Hanya menyalakan container yang sudah ada.
 - Tidak memeriksa perubahan konfigurasi.
 - Cocok untuk production atau saat tidak ada perubahan konfigurasi.

4. Best Practice untuk Menghindari Redundansi

- **Jangan hapus volume:**
Hindari `docker-compose down -v` kecuali Anda ingin menghapus data secara permanen.
- **Gunakan `docker-compose up` :**
Perintah ini akan menggunakan container yang sudah ada jika tidak ada perubahan konfigurasi.
- **Gunakan `docker-compose start` :**
Jika Anda yakin tidak ada perubahan konfigurasi dan hanya ingin menyalakan container yang sudah ada.