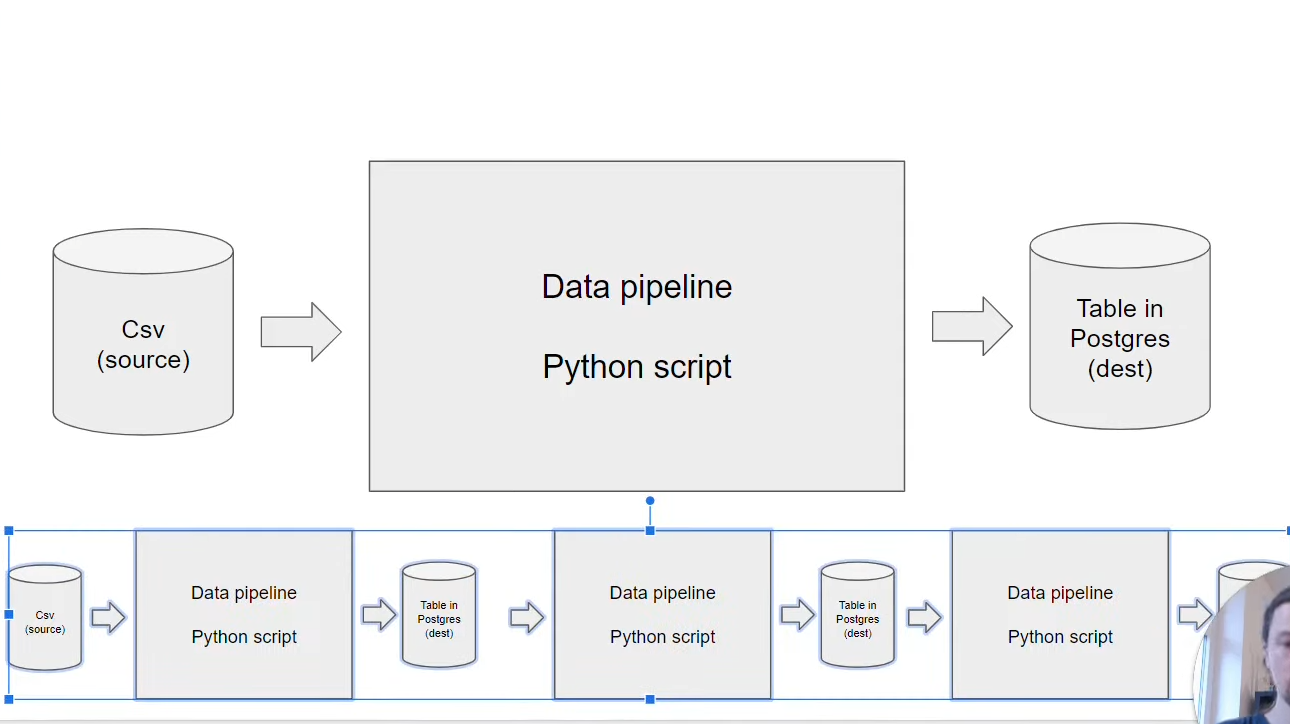
Week1 notes



## Why docker?

We can run things on Docker, so that we don’t have to worry about having conflicts with local machine (if let say you local machine also have postgres or pgadmin running)

-Reproducibility.

-Local experiment

-Integration test (CI/CD), remember the zoomcamp last week, we used docker and did a lot of testing before ultimately pushing it to Amazon and Kubernetes… We can do integration test to make sure data we want is there!

-Running pipelines on the cloud (AWS Batch, Kubernetes jobs)

-Spark (defining data pipelines, specifying dependencies)

-Serverless (AWS Lambda)

# Lesson 1: Docker

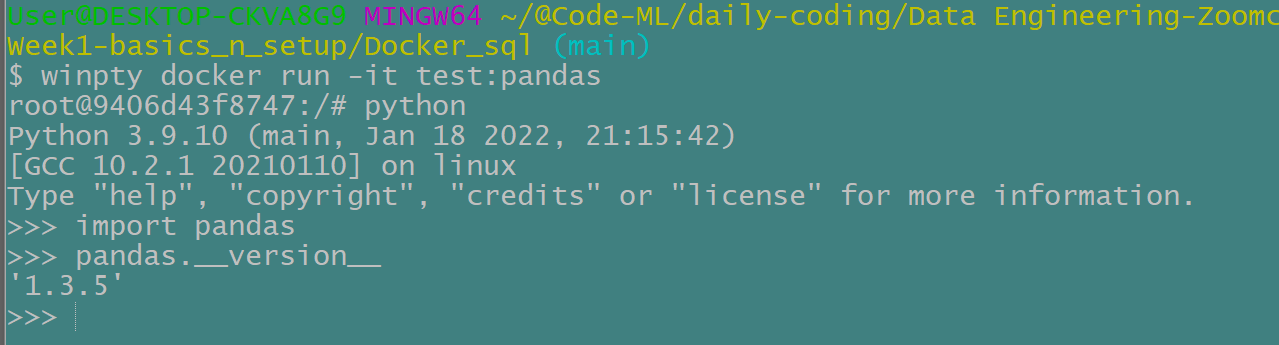
Create docker file

Build docker image:

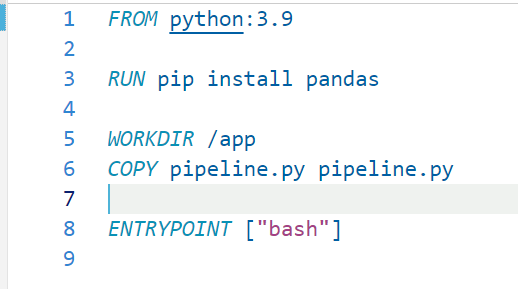
$docker build -t test:pandas .

Run docker:

$docker run -it test:pandas

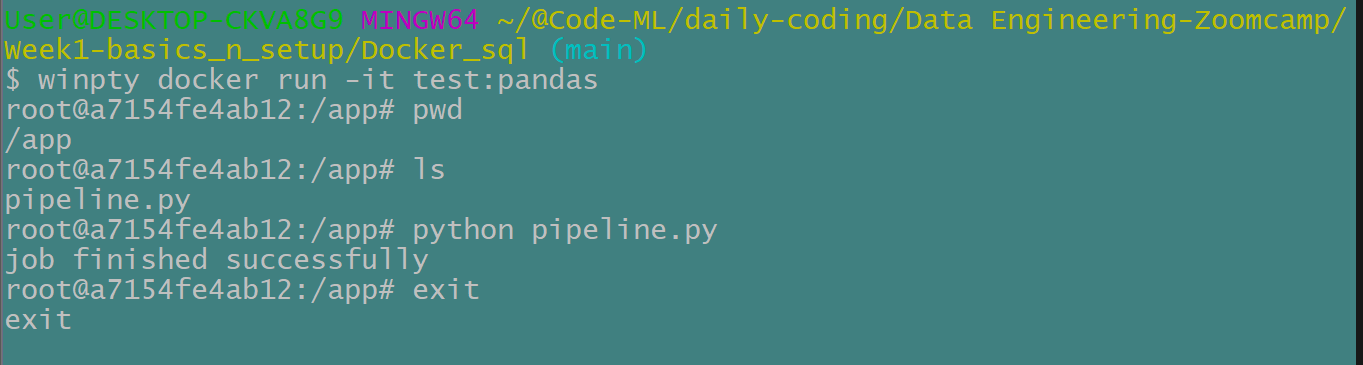


Since we specify the entry point is bash, we need to enter python from the bash. Then since we specify in dockerfile to install pandas, we can import pandas to check!



For copy line, we basically specify the name of the file from our source of our host machine, and then the name on the destination.

WorkDir, the location in the image in the container where we will copy the file

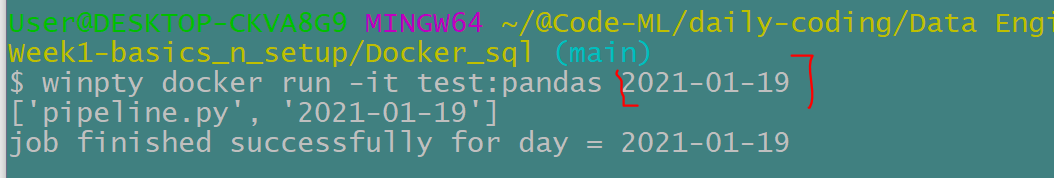


Now we can see our pipeline.py file in the container, and the container directory is /app as specified under dockerfile.



^Passing command line argument to the script.

We also modified the entrypoint in dockerfile to run the script straight away.



Now we can pass the argument this way, \*when we are running the docker image!

It successfully print out the argument (date) passed.

# Lesson 2 Postgres SQL

Volume is use to mapping folder that we have in our file system on the host machine to a folder in the container.

Under volume in the file docker-compose.yaml,

We basically use -v to mount the folder

-v folder path: directory of the container

```

winpty docker run -it \

-e POSTGRES\_USER="root" \

-e POSTGRES\_PASSWORD="root" \

-e POSTGRES\_DB="ny\_taxi" \

-v "c:/Users/User/@Code-ML/daily-coding/Data Engineering-Zoomcamp/Week1-basics\_n\_setup/Docker\_sql/ny\_taxi\_postgres\_data":/var/lib/postgresql/data \

-p 5432:5432 \

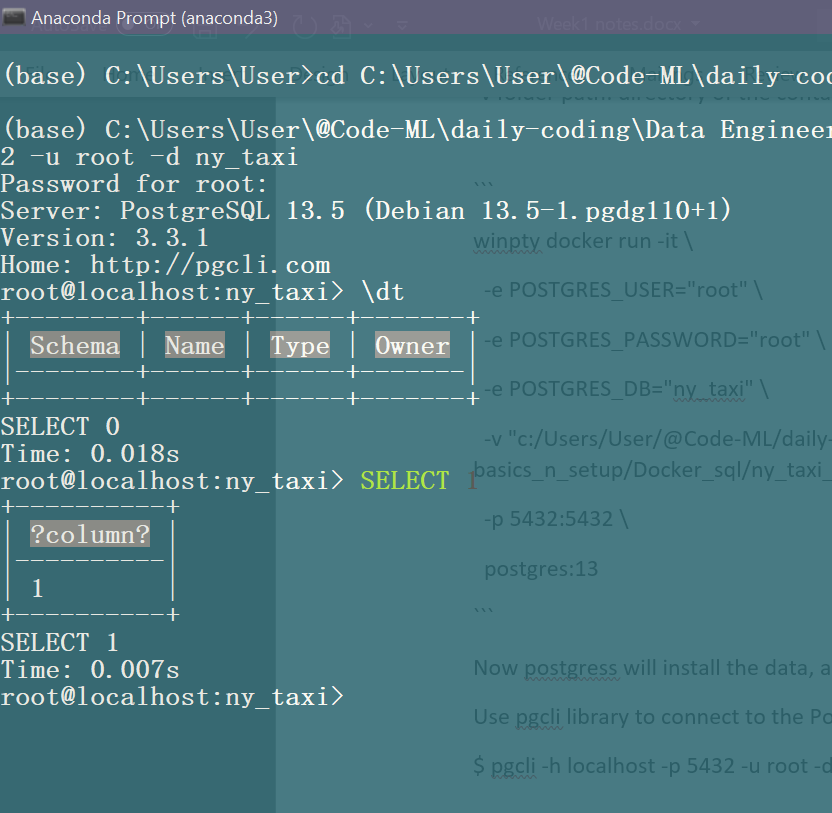
postgres:13

```

Now postgress will install the data, and we can start working on it !

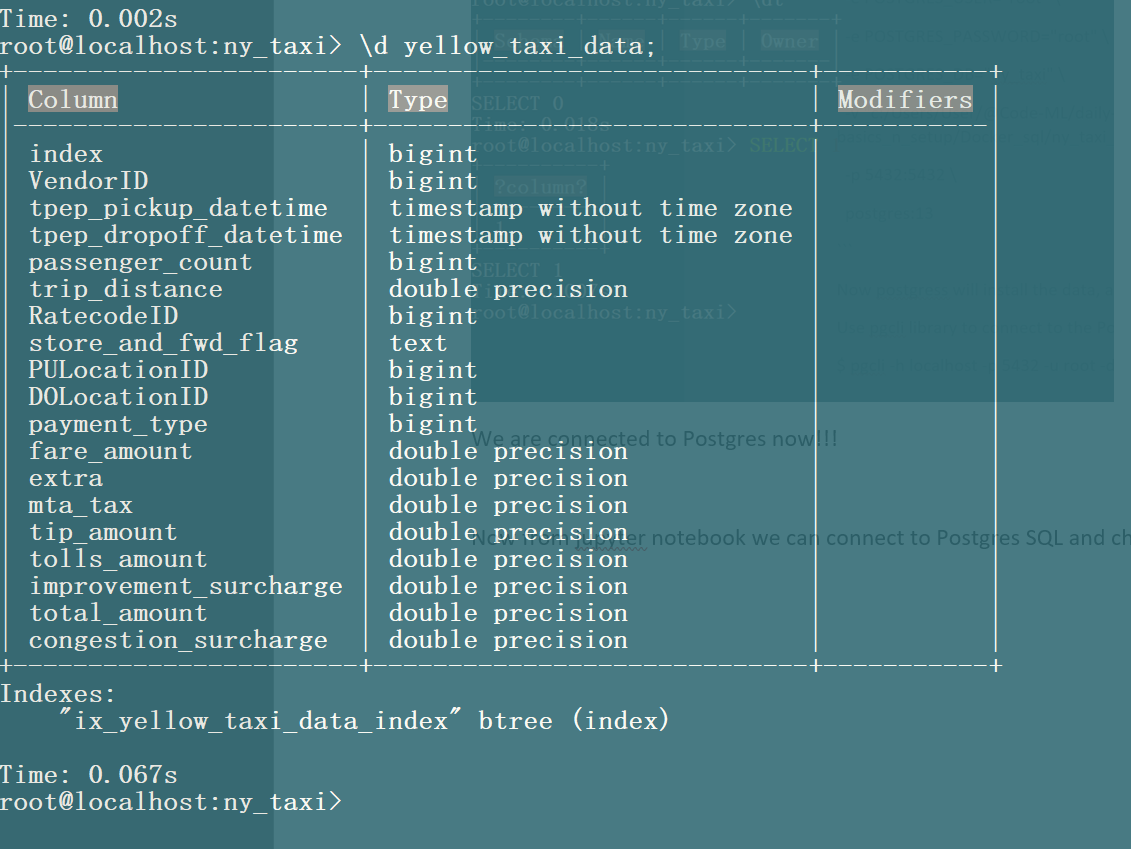
Use pgcli library to connect to the Postgres database

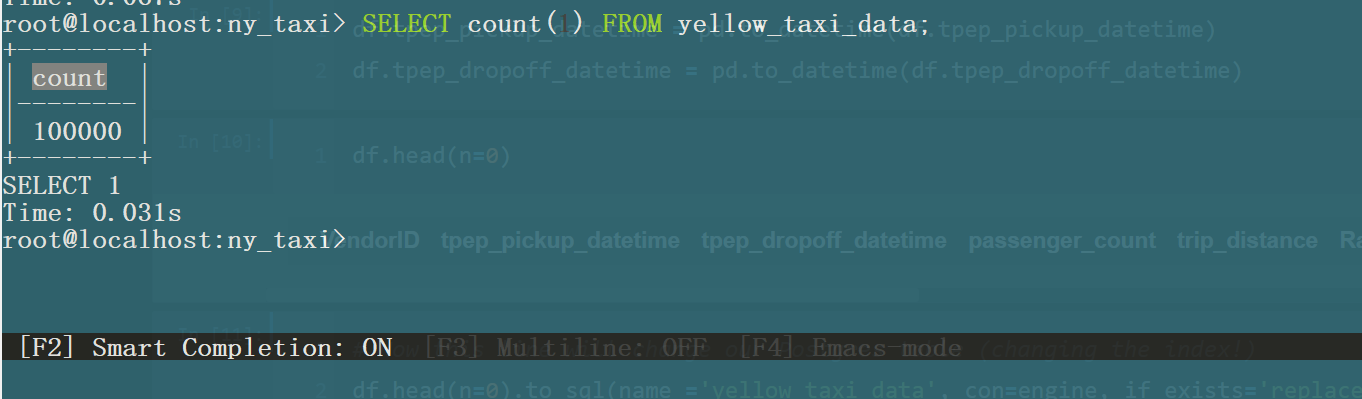
$ pgcli -h localhost -p 5432 -u root -d ny\_taxi



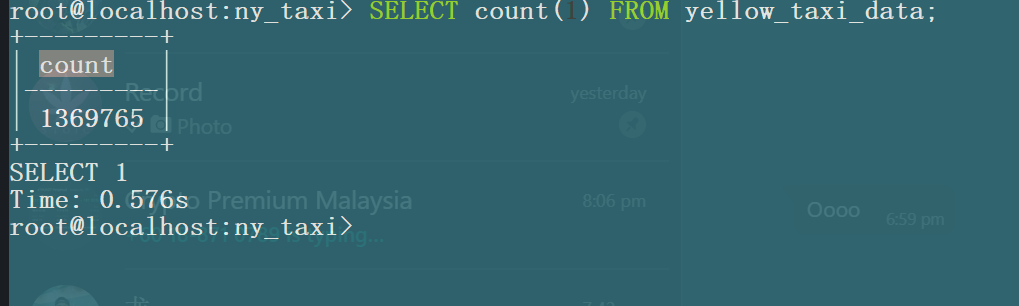
We are connected to Postgres now!!!

Now from jupyter notebook we can connect to Postgres SQL and change the database and stuff.





Count row, we have 100k rows!



^Total data that we have now!

This class we use pgcli to connect to our database, next we will use pgadmin ! Pfadmin is more friendly!

# Lesson 3: connecting pgadmin and Postgres

<https://hub.docker.com/r/dpage/pgadmin4/>

1. Copy the pgadmin image from docker hub
2. Run the image

```

winpty docker run -it \

-e PGADMIN\_DEFAULT\_EMAIL="admin@admin.com" \

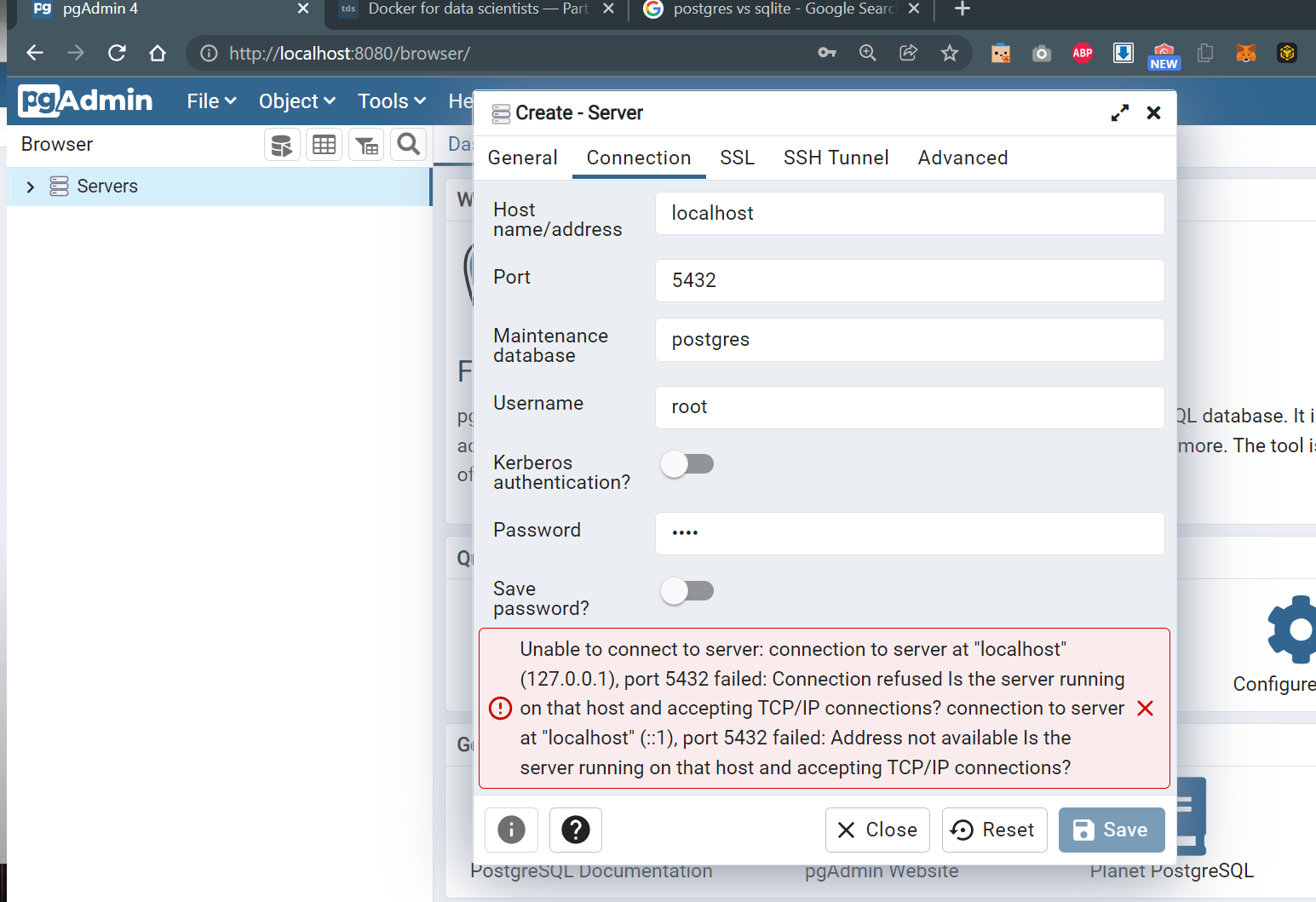
-e PGADMIN\_DEFAULT\_PASSWORD="root" \

-p 8080:80 \

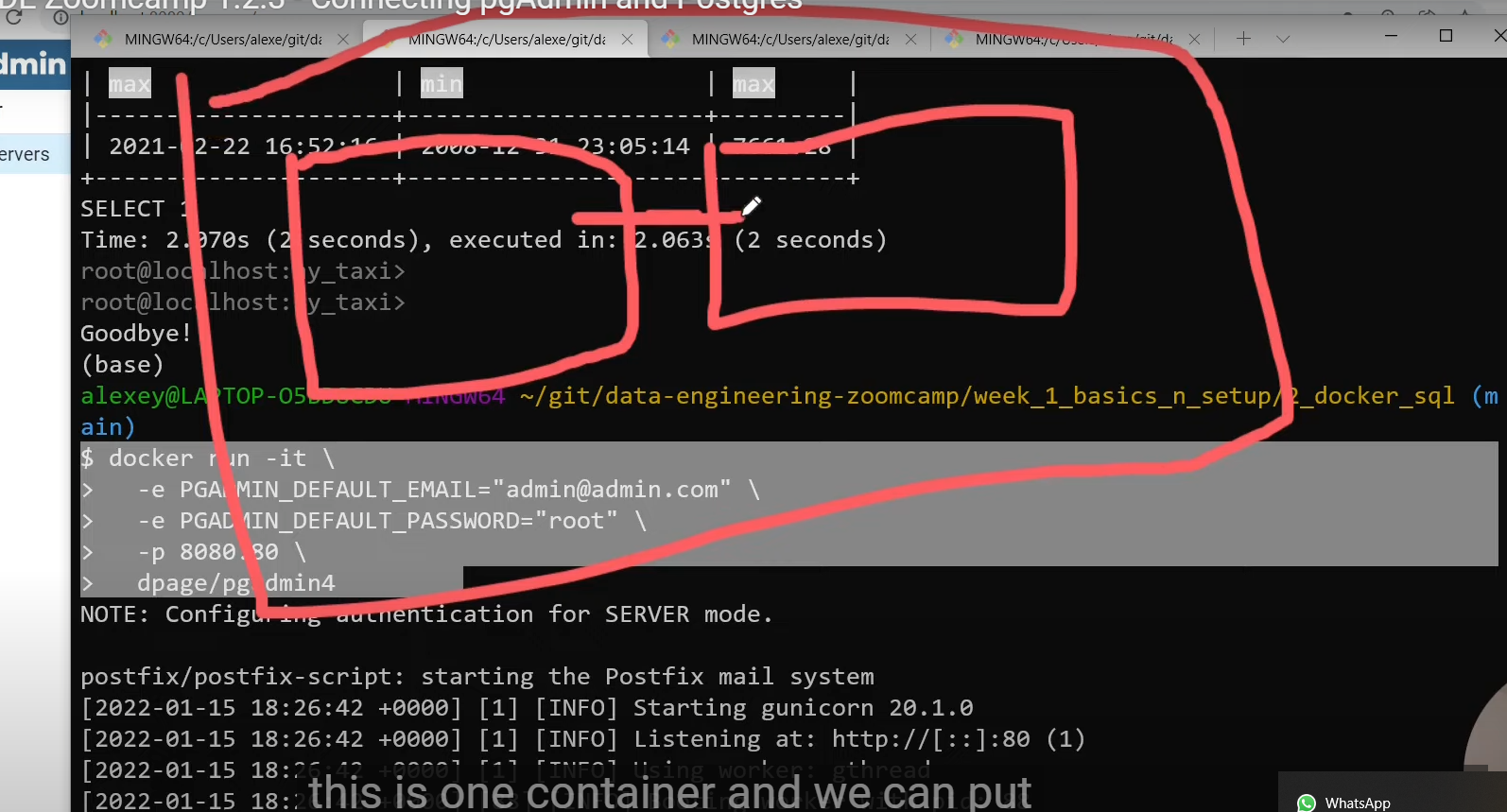
dpage/pgadmin4

```

Now if we visit local host 8080, we can view pgadmin!!



^ There is an error



We need to put two containers in the same network, or else they cant speak to each other, thus we encountered error above.

## Update docker code with network:

$docker network create pg-network

**Postgres container:**

```

winpty docker run -it \

-e POSTGRES\_USER="root" \

-e POSTGRES\_PASSWORD="root" \

-e POSTGRES\_DB="ny\_taxi" \

-v "c:/Users/User/@Code-ML/daily-coding/Data Engineering-Zoomcamp/Week1-basics\_n\_setup/Docker\_sql/ny\_taxi\_postgres\_data":/var/lib/postgresql/data \

-p 5432:5432 \

--network=pg-network \

--name pg-database \

postgres:13

```

**Pgadmin:**

```

winpty docker run -it \

-e PGADMIN\_DEFAULT\_EMAIL="admin@admin.com" \

-e PGADMIN\_DEFAULT\_PASSWORD="root" \

-p 8080:80 \

--network=pg-network \

--name pgadmin \

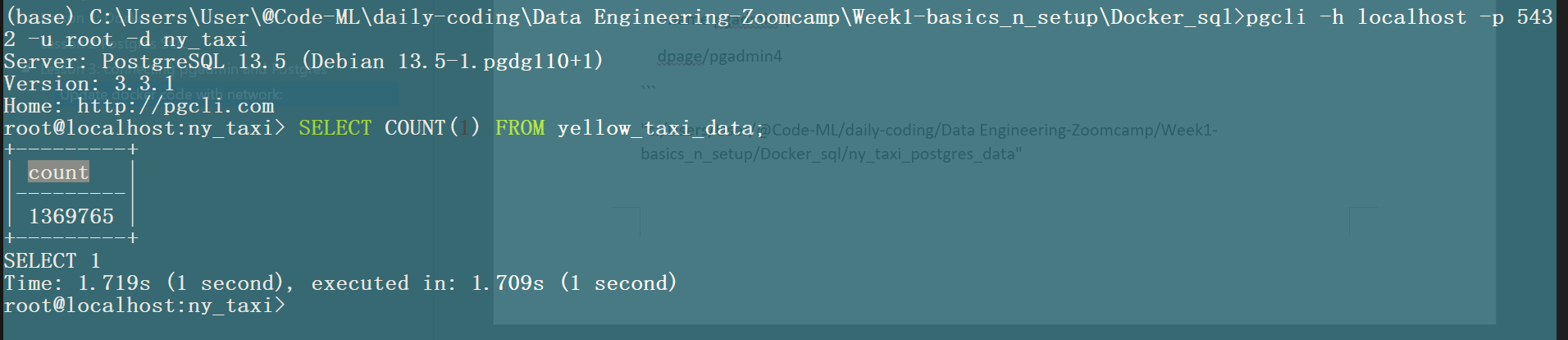
dpage/pgadmin4

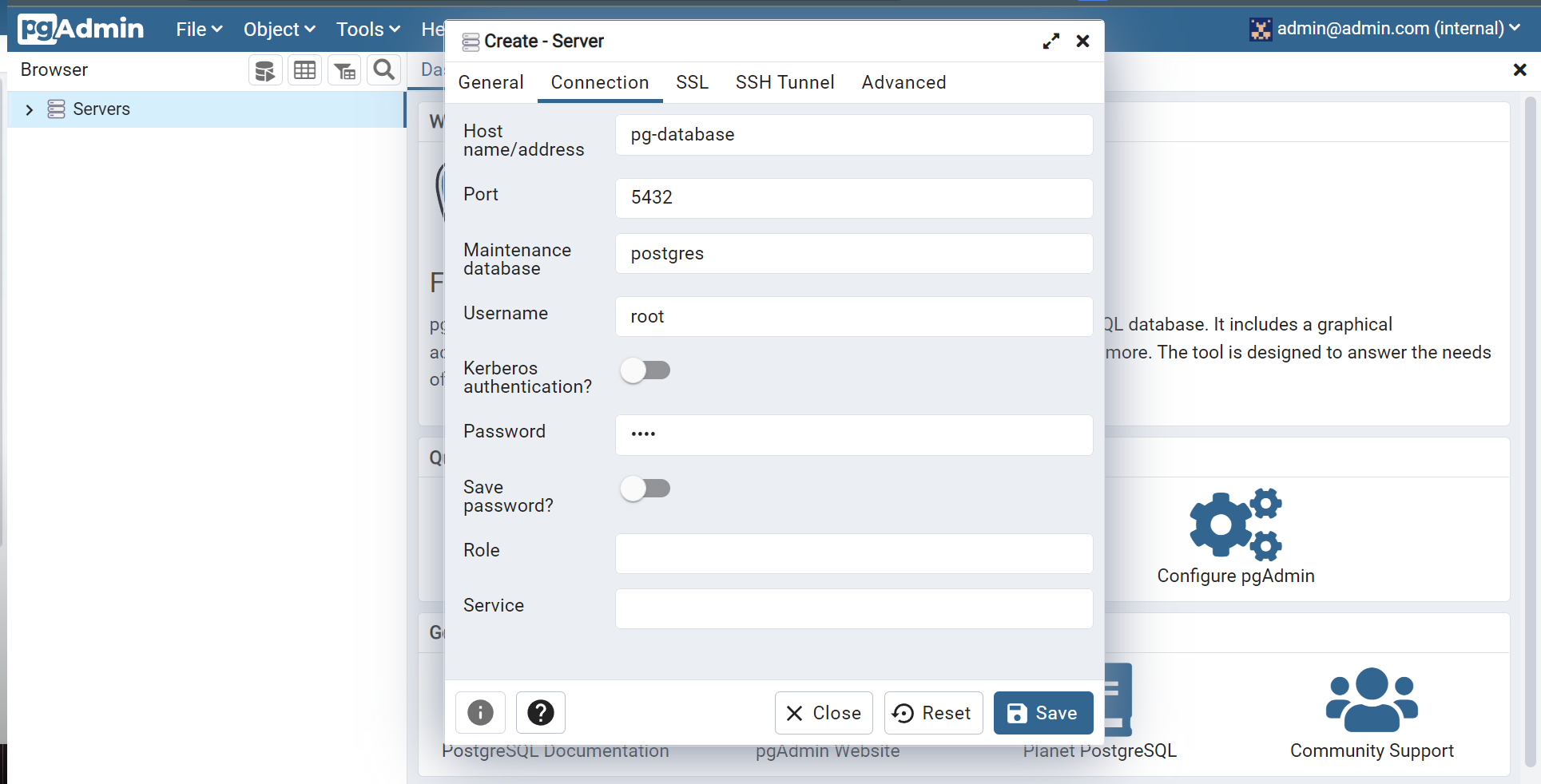
```

"c:/Users/User/@Code-ML/daily-coding/Data Engineering-Zoomcamp/Week1-basics\_n\_setup/Docker\_sql/ny\_taxi\_postgres\_data"

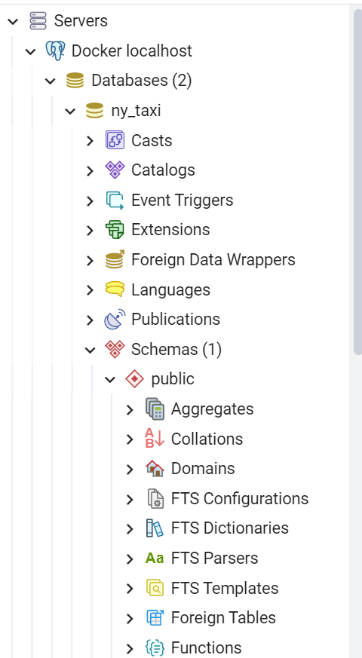
Ok so after we run postgres container, we can use to check if the data is still in our database.

$ pgcli -h localhost -p 5432 -u root -d ny\_taxi





Now we can create, as for name we refer to pg-database.

To view our data, under schemas select tables

# Lesson 4: Dockerizing the ingestion script

After download data, we want ot use the script to add data to our Postgres database basically.

We will explore how to do this on Airflow next week !

We first link to Pgadmin, edit the script, use argparse to pass argument.

Then delete our current data in our Postgres database with

$drop “table\_name”;

Then we want to test the script written!

URL = “https://s3.amazonaws.com/nyc-tlc/trip+data/yellow\_tripdata\_2021-01.csv”

(not sure why I cannot pass the parameter to Git bash zzz…)

$python ingest\_data.py \

--user = root \

--password = root\

--host = localhost \

--port = 5432 \

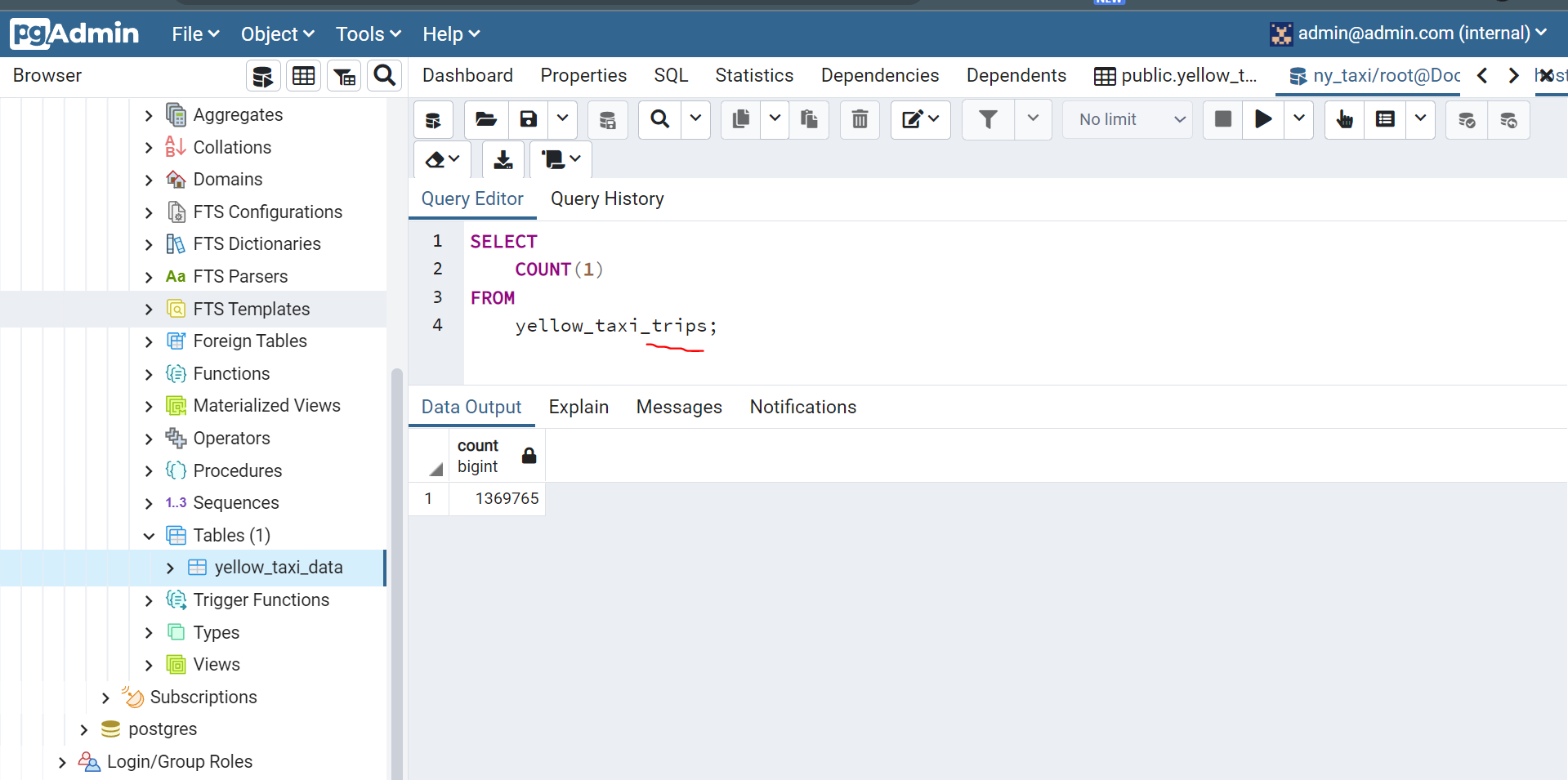
--db = ny\_taxi \

--table\_name = yellow\_taxi\_trips \

--url = “https://s3.amazonaws.com/nyc-tlc/trip+data/yellow\_tripdata\_2021-01.csv”

--url = ${URL} (this doesn’t work zz cuz I cant pass the argument)

Now that we’ve downloaded data and insert to the database, we can test it on Pgadmin right away. Make sure u change the data name to the latest name.



Now we update our docker file to add dependencies and copy the py script file that we need.

Rebuild and run docker container:

$docker build -t taxi\_ingest:v001 .

$ docker run -it taxi\_ingest:v001 \

--user = root \

--password = root\

--host = localhost \

--port = 5432 \

--db = ny\_taxi \

--table\_name = yellow\_taxi\_trips \

--url = “https://s3.amazonaws.com/nyc-tlc/trip+data/yellow\_tripdata\_2021-01.csv”

$ docker run -it taxi\_ingest:v001 --user = root --password = root --host = localhost --port = 5432 --db = ny\_taxi --table\_name = yellow\_taxi\_trips --url = “https://s3.amazonaws.com/nyc-tlc/trip+data/yellow\_tripdata\_2021-01.csv”

\*Remark: Of cuz at this stage we cant run it, we don’t have postgres in the network, in fact we need to run postgres in the network first, then only this current container can connect to the container and insert data to the postgres database (we need both this container and the postgrese database within the same network!)

## Latest docker code (to run within the same network):

$ docker run -it

--network=pg-network \

taxi\_ingest:v001 \

--user = root \

--password = root\

--host = localhost \

--port = 5432 \

--db = ny\_taxi \

--table\_name = yellow\_taxi\_trips \

--url = “https://s3.amazonaws.com/nyc-tlc/trip+data/yellow\_tripdata\_2021-01.csv”

$ docker run -it --network=pg-network taxi\_ingest:v001 --user = root --password = root --host = localhost --port = 5432 --db = ny\_taxi --table\_name = yellow\_taxi\_trips --url = “https://s3.amazonaws.com/nyc-tlc/trip+data/yellow\_tripdata\_2021-01.csv”

## Start a python server on local machine:

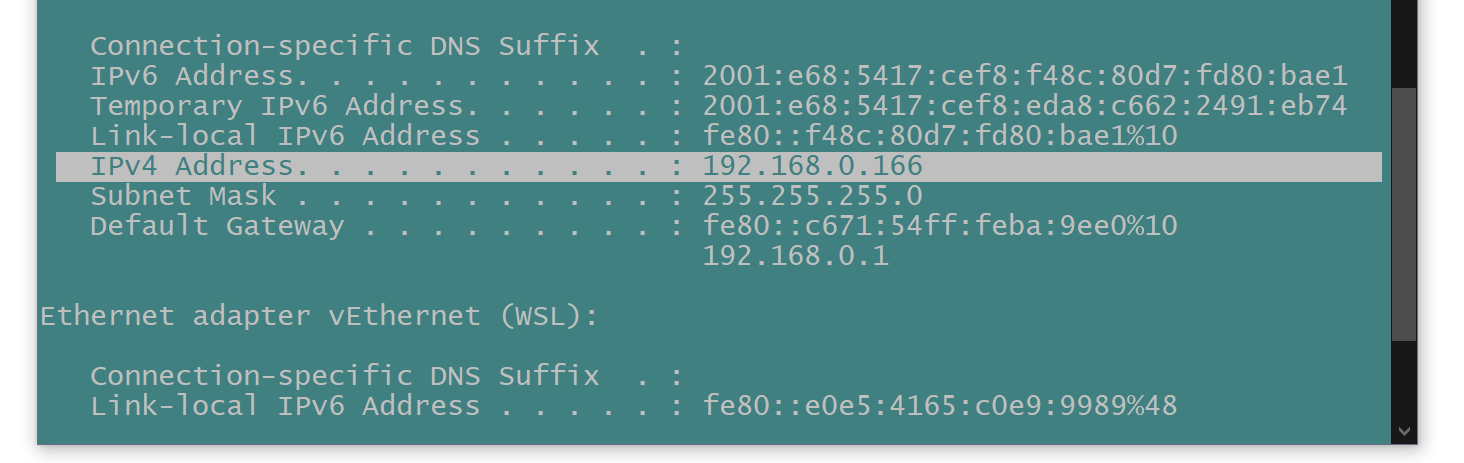
This is so we can pass the data directly, and don’t need to wait for 10 minutes to download the data.

$Python -m http.server

We can actually see our data there.

But since its running on our local computer’s local host, we need to get out IP address

$ipconfig (to get IP address)



Our IP address: <http://192.168.0.166>

Enter on browser: <http://192.168.0.166:8000/> (we can see the same directory page as before).

URL = “<http://192.168.0.166:8000/>yellow\_tripdata\_2021-01.csv”

Update our URL, THIS SHOULD BE FASTER!!!

$ 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=“<http://192.168.0.166:8000/>yellow\_tripdata\_2021-01.csv”

$ 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=“<http://192.168.0.166:8000/>yellow\_tripdata\_2021-01.csv”

THE CSV LINK WORKS! BUT GOT ANOTHER ERROR!! Need to change localhost to pg-database

WORK LIKE A CHARM!!

# Lesson 5: Running Postgres and pgadmin with Docker-Compose

Now we need to run things on a few different terminals.

We need to run the docker postgres database container, and pgadmin container on another console. Instead of that we can specify one yaml file do orchestrate all these (that are under the same network), so we only need to run once in one console, for that we use DOCKER COMPOSE!



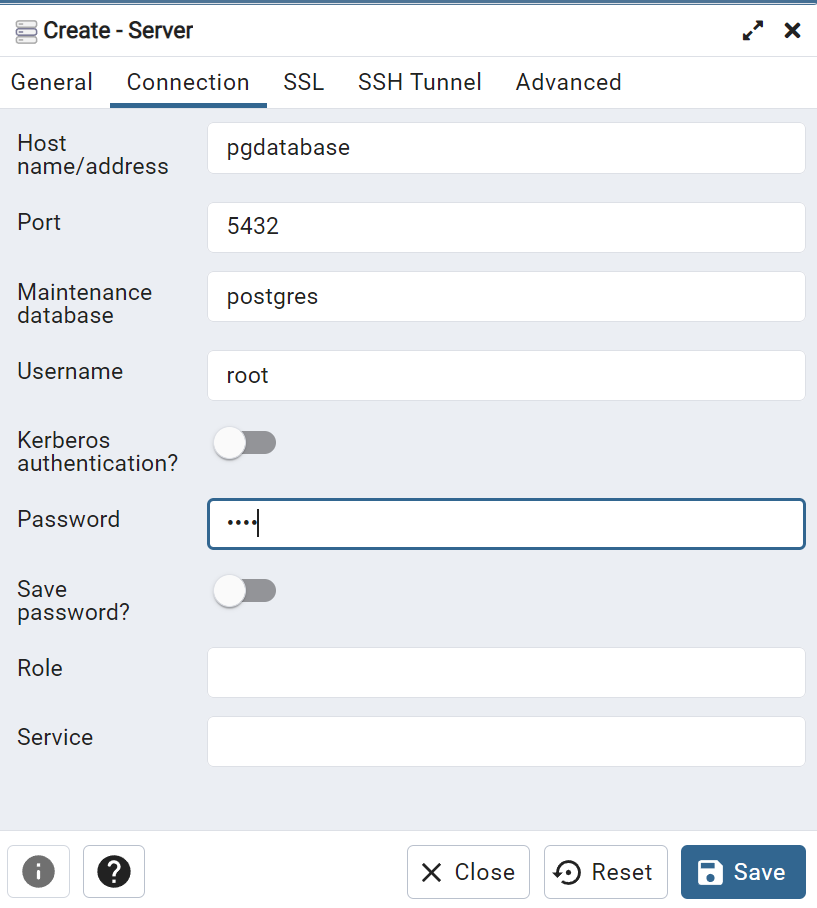
Take note in docker compose we don’t have to write the full path! The rw means read and write.

And the magic command to spin things up:

$docker-compose up

We can then access pgadmin, but we will need to set up the server again because we didn’t do volume mapping for pgadmin under yaml file!!

Here for the configuration, we type in according to the details of pgdatabse service under the yaml file:



# Lesson 6: SQL Refresher

Ok, so now we will do some SQL.

Download new data:

$ wget https://s3.amazonaws.com/nyc-tlc/misc/taxi+\_zone\_lookup.csv

But before that, we need to put the new data to Postgres db and view it using pgadmin.

I think the steps would be:

1. Download data
2. Create folder
3. Run postgres db docker container
4. Inject data
5. Run pgadmin docker container
6. Put them into yaml file and run with docker compose.

#2

Folder name: taxi\_zone\_postgres\_data

#3: Run the postgres db

winpty docker run -it \

-e POSTGRES\_USER="root" \

-e POSTGRES\_PASSWORD="root" \

-e POSTGRES\_DB="taxi\_zone" \

-v "c:/Users/User/@Code-ML/daily-coding/Data Engineering-Zoomcamp/Week1-basics\_n\_setup/Docker\_sql/taxi\_zone\_postgres\_data":/var/lib/postgresql/data \

-p 5432:5432 \

postgres:10

**# Inject data to Postgres database!**

$python ingest\_data\_taxi\_zone.py\

--user=root \

--password=root\

--host=localhost \

--port=5432 \

--db=taxi\_zone \

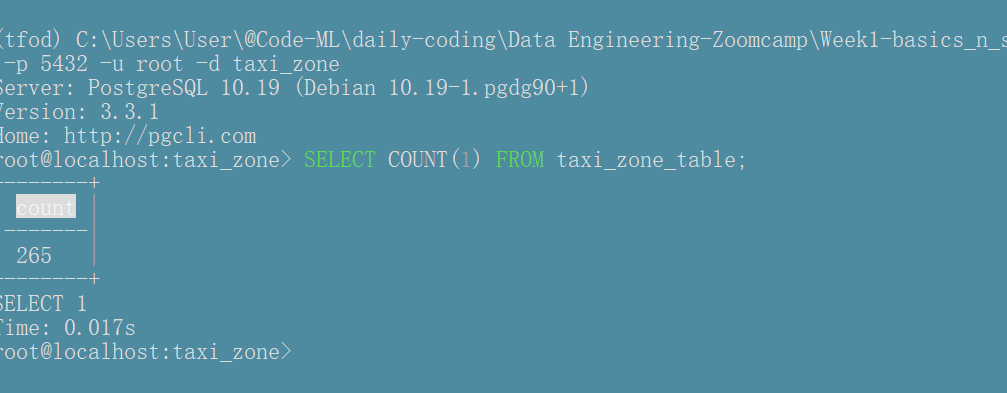
--table\_name=taxi\_zone\_table \

--url=“https://s3.amazonaws.com/nyc-tlc/misc/taxi+\_zone\_lookup.csv”

$ python ingest\_data\_taxi\_zone.py --user=root --password=root --host=localhost --port=5432 --db=taxi\_zone --table\_name=taxi\_zone\_table --url=“https://s3.amazonaws.com/nyc-tlc/misc/taxi+\_zone\_lookup.csv”

**# CHECK if we have injected data correctly!**

$ pgcli -h localhost -p 5432 -u root -d taxi\_zone



Data is injected correctly, yes!!!!

# This step require one more configuration to create a network and pass them to the same network!

winpty docker run -it \

-e PGADMIN\_DEFAULT\_EMAIL="admin@admin.com" \

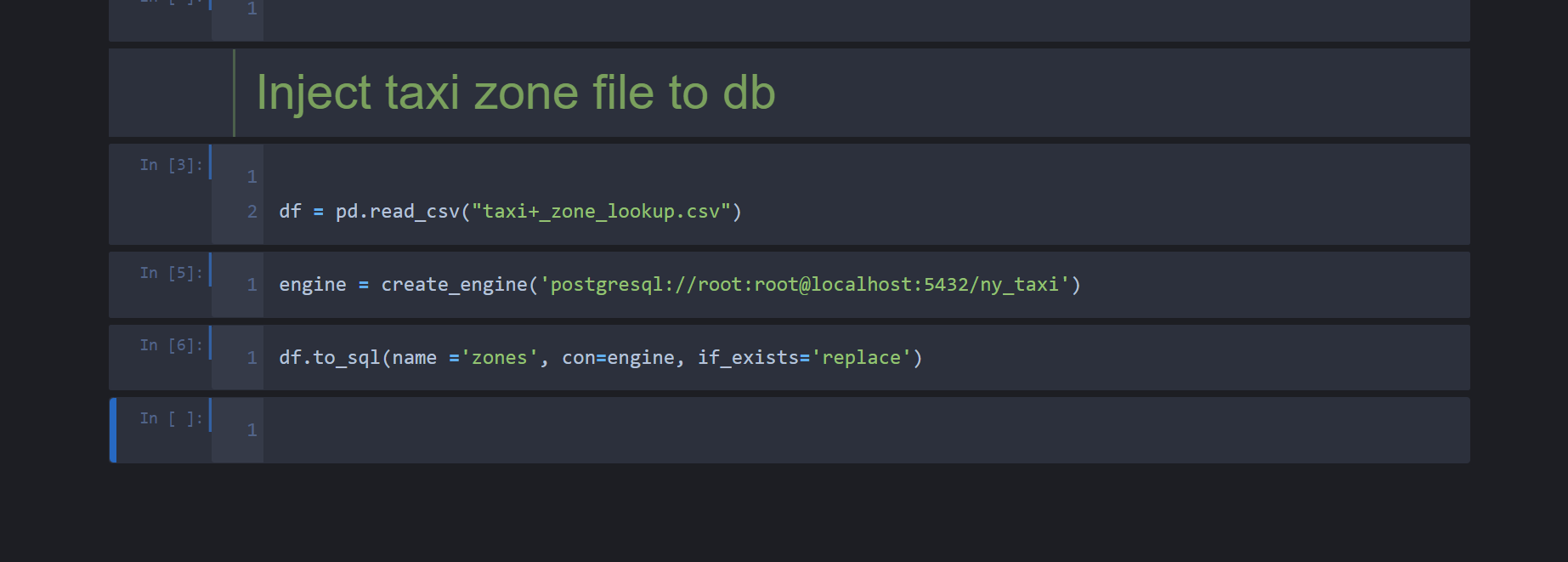
-e PGADMIN\_DEFAULT\_PASSWORD="root" \

-p 8080:80 \

dpage/pgadmin4

## Mistake!

Above steps are unnecessary, I just need to inject directly to the previous docker-compose containers!



We can just do this to inject to the same database that we ald created before this!!!

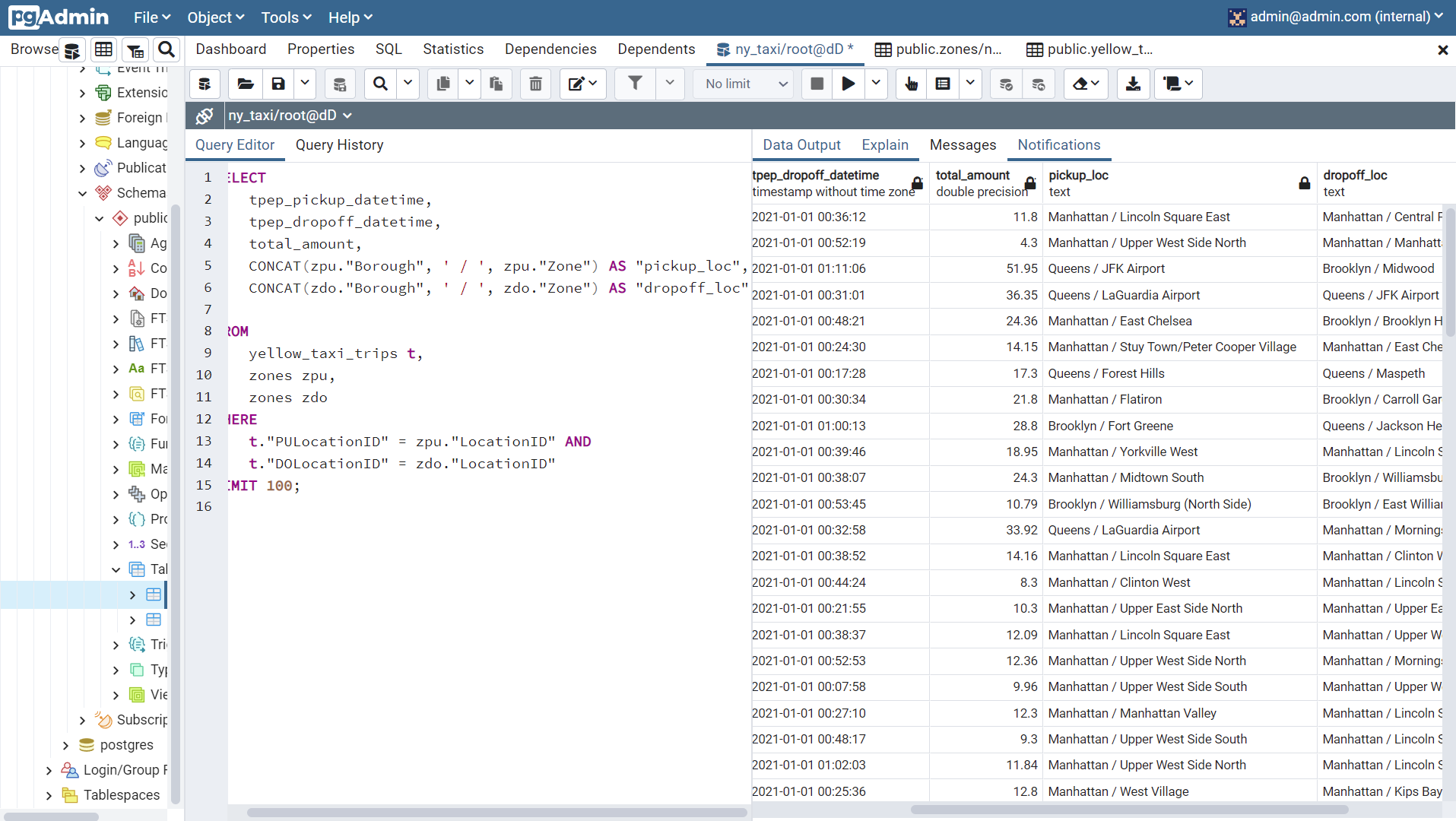
1. Spin up docker compose
2. Login to pgadmin and create server!
3. Start playing with the data!

## Lets start doing SQL!

We want to join the two tables we have

1. SELECT \*
2. FROM
3. yellow\_taxi\_trips t,
4. zones zpu,
5. zones zdo
6. WHERE
7. t."PULocationID" = zpu."LocationID" AND
8. t."DOLocationID" = zdo."LocationID"
9. LIMIT 100;

This will return table, with zone = to the ID from yellow\_taxi\_trips, since we have Pick up ID and Drop off ID, two zones will be appended to the data.



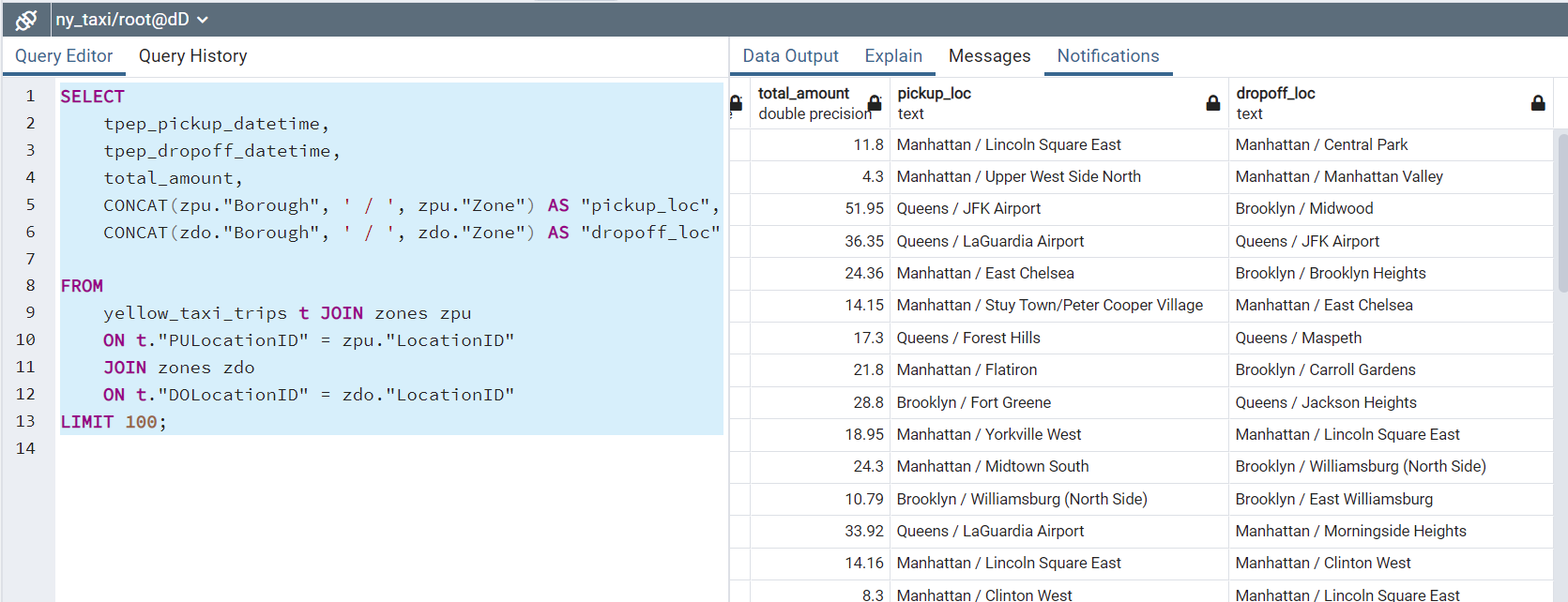
CLEAN!!

1. SELECT
2. tpep\_pickup\_datetime,
3. tpep\_dropoff\_datetime,
4. total\_amount,
5. CONCAT(zpu."Borough", ' / ', zpu."Zone") AS "pickup\_loc",
6. CONCAT(zdo."Borough", ' / ', zdo."Zone") AS "dropoff\_loc"
8. FROM
9. yellow\_taxi\_trips t,
10. zones zpu,
11. zones zdo
12. WHERE
13. t."PULocationID" = zpu."LocationID" AND
14. t."DOLocationID" = zdo."LocationID"
15. LIMIT 100;

See how zpu and zdo are used! And also, the CONCAT function to combine the strings, cool!

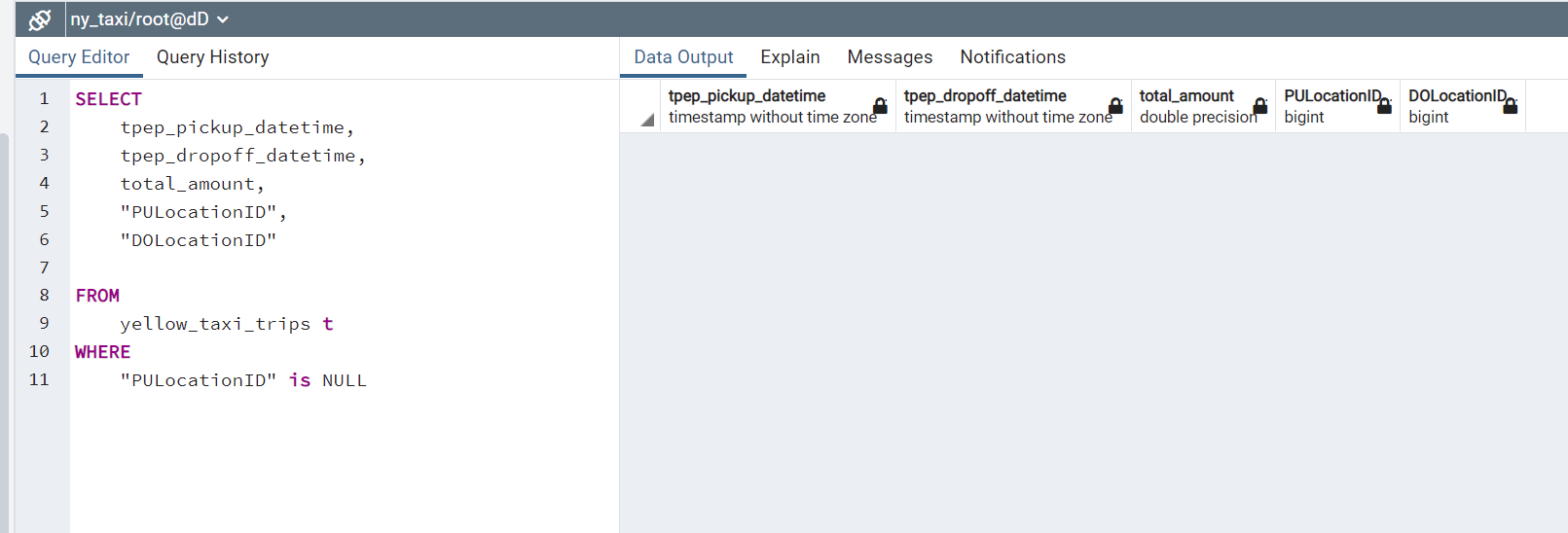
This is actually INNER JOIN, below is alternative way of achieving the same result with JOIN method:

1. SELECT
2. tpep\_pickup\_datetime,
3. tpep\_dropoff\_datetime,
4. total\_amount,
5. CONCAT(zpu."Borough", ' / ', zpu."Zone") AS "pickup\_loc",
6. CONCAT(zdo."Borough", ' / ', zdo."Zone") AS "dropoff\_loc"
8. FROM
9. yellow\_taxi\_trips t JOIN zones zpu
10. ON t."PULocationID" = zpu."LocationID"
11. JOIN zones zdo
12. ON t."DOLocationID" = zdo."LocationID"
13. LIMIT 100;

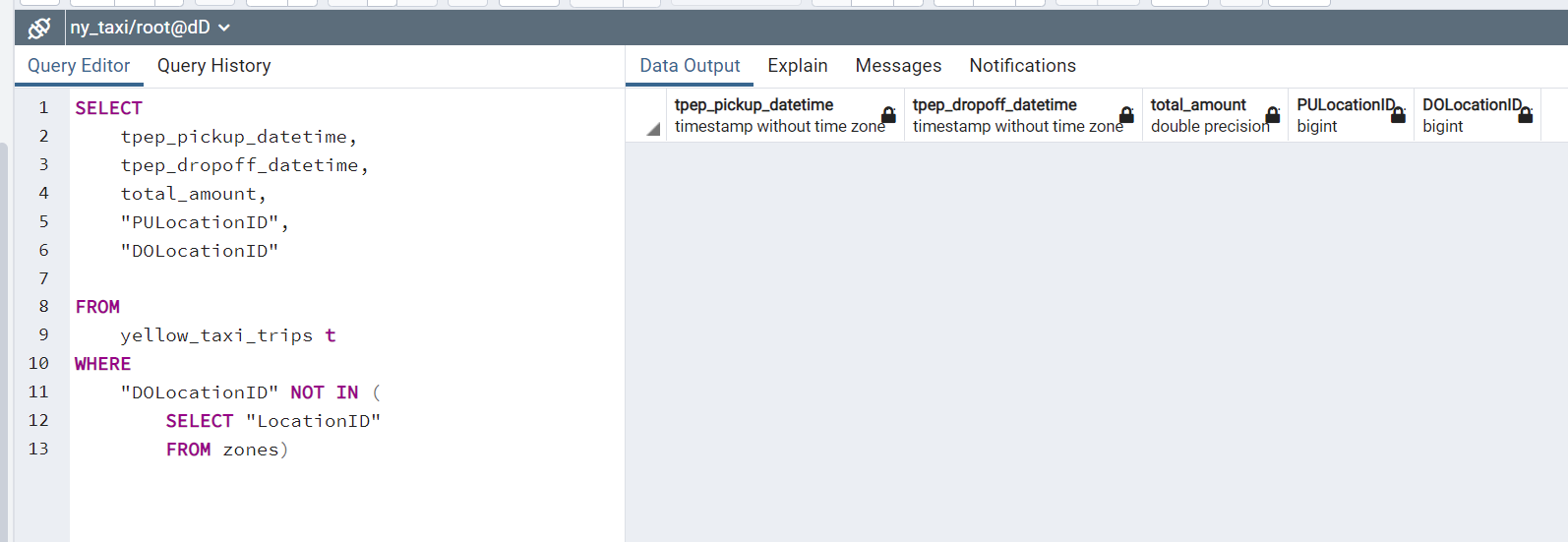


^We can get the same output with this code.

## Null checking:



Check if there is any null data for PULocationID and DOLocationID



We search for the DOLocationID (From zone table) that is not in the locationID (From yellow taxi table)

For practice sake, we will delete some data, then proceed with doing a left join (so that the missing values will appear as null).

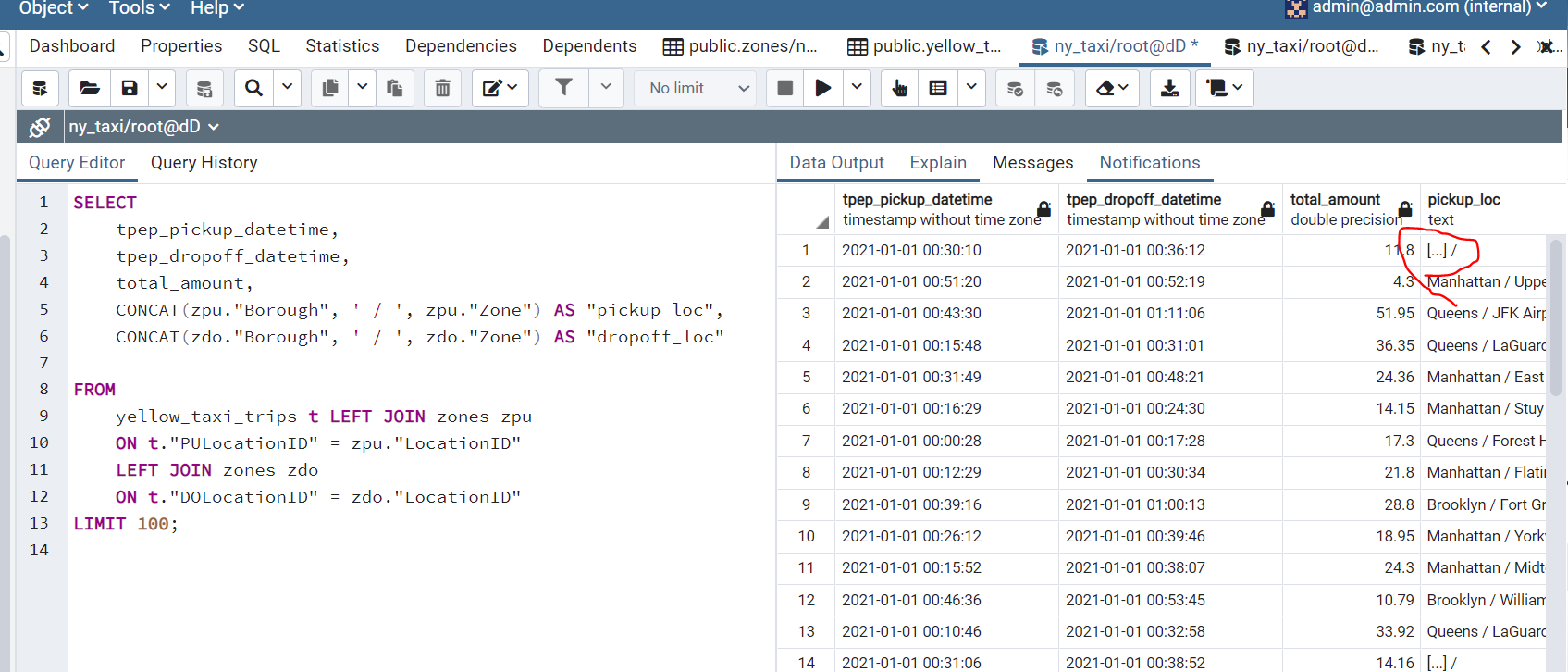
1. DELETE FROM zones
2. WHERE "LocationID" = 142;
3. SELECT
4. tpep\_pickup\_datetime,
5. tpep\_dropoff\_datetime,
6. total\_amount,
7. CONCAT(zpu."Borough", ' / ', zpu."Zone") AS "pickup\_loc",
8. CONCAT(zdo."Borough", ' / ', zdo."Zone") AS "dropoff\_loc"
10. FROM
11. yellow\_taxi\_trips t JOIN zones zpu
12. ON t."PULocationID" = zpu."LocationID"
13. JOIN zones zdo
14. ON t."DOLocationID" = zdo."LocationID"
15. LIMIT 100;

Now when we rerun above scrupt, the data that is missing will disappear (this is what happened when u perform an inner join!).

We don’t want that, we want it to display null values, thus we have to use left join!

Left join: IF there is record present on the left but not on the right, we still want to show them.

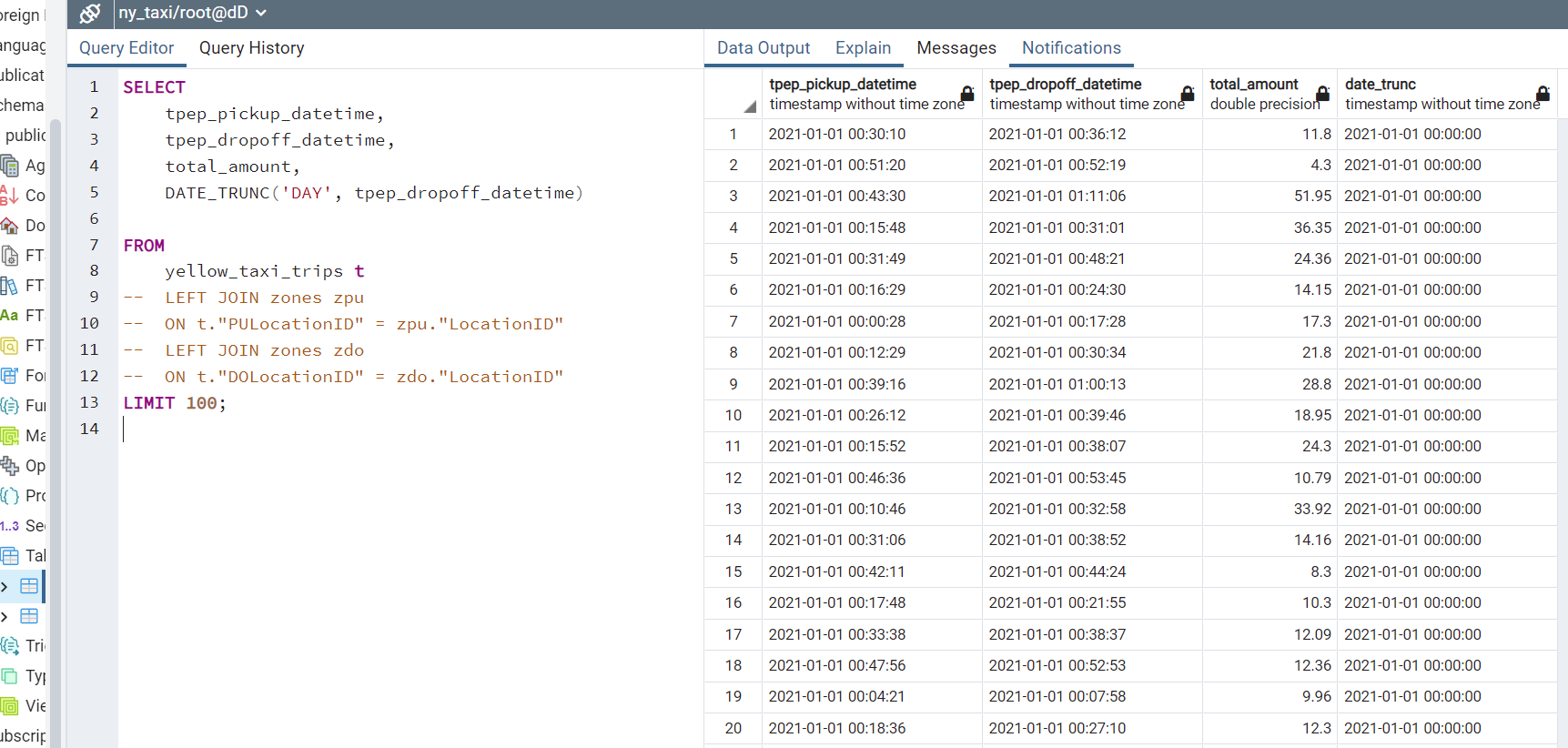
The missing values is now displayed as …



## The group by and aggregate query:

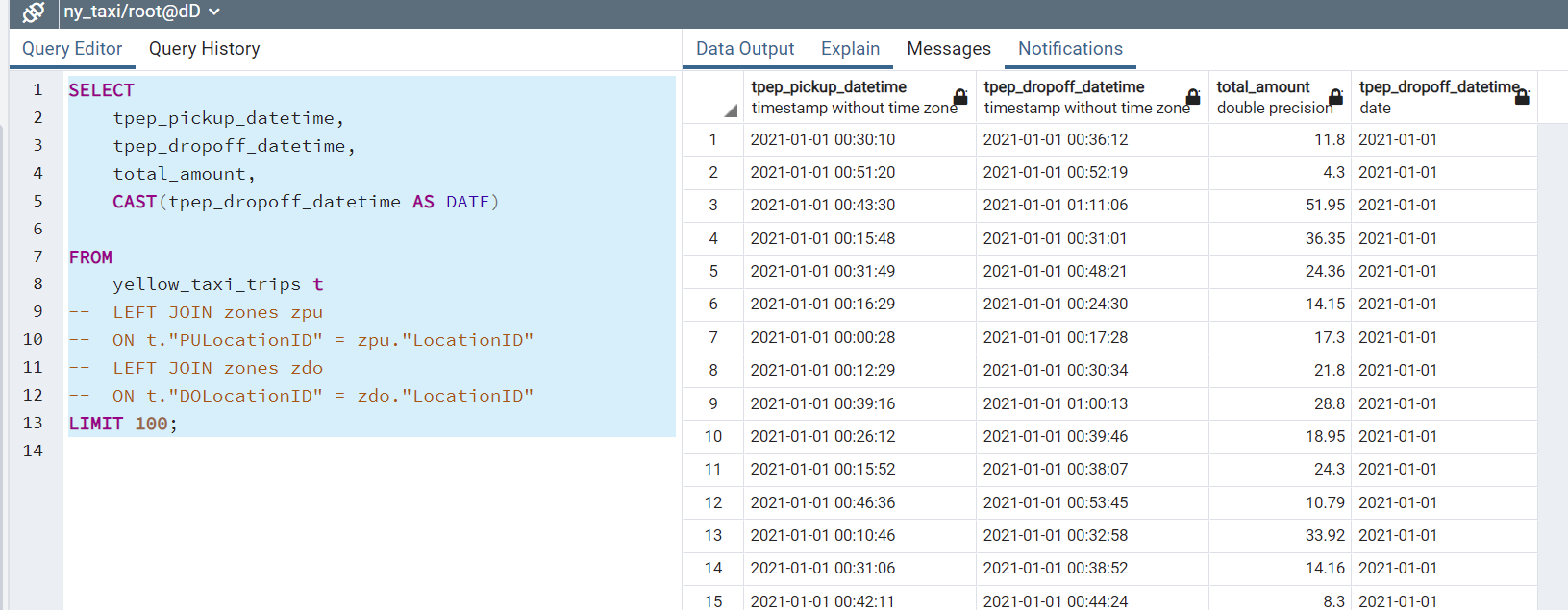
First we want to remove the time and keep the day only:

1. SELECT
2. tpep\_pickup\_datetime,
3. tpep\_dropoff\_datetime,
4. total\_amount,
5. DATE\_TRUNC('DAY', tpep\_dropoff\_datetime)
7. FROM
8. yellow\_taxi\_trips t
9. LIMIT 100;

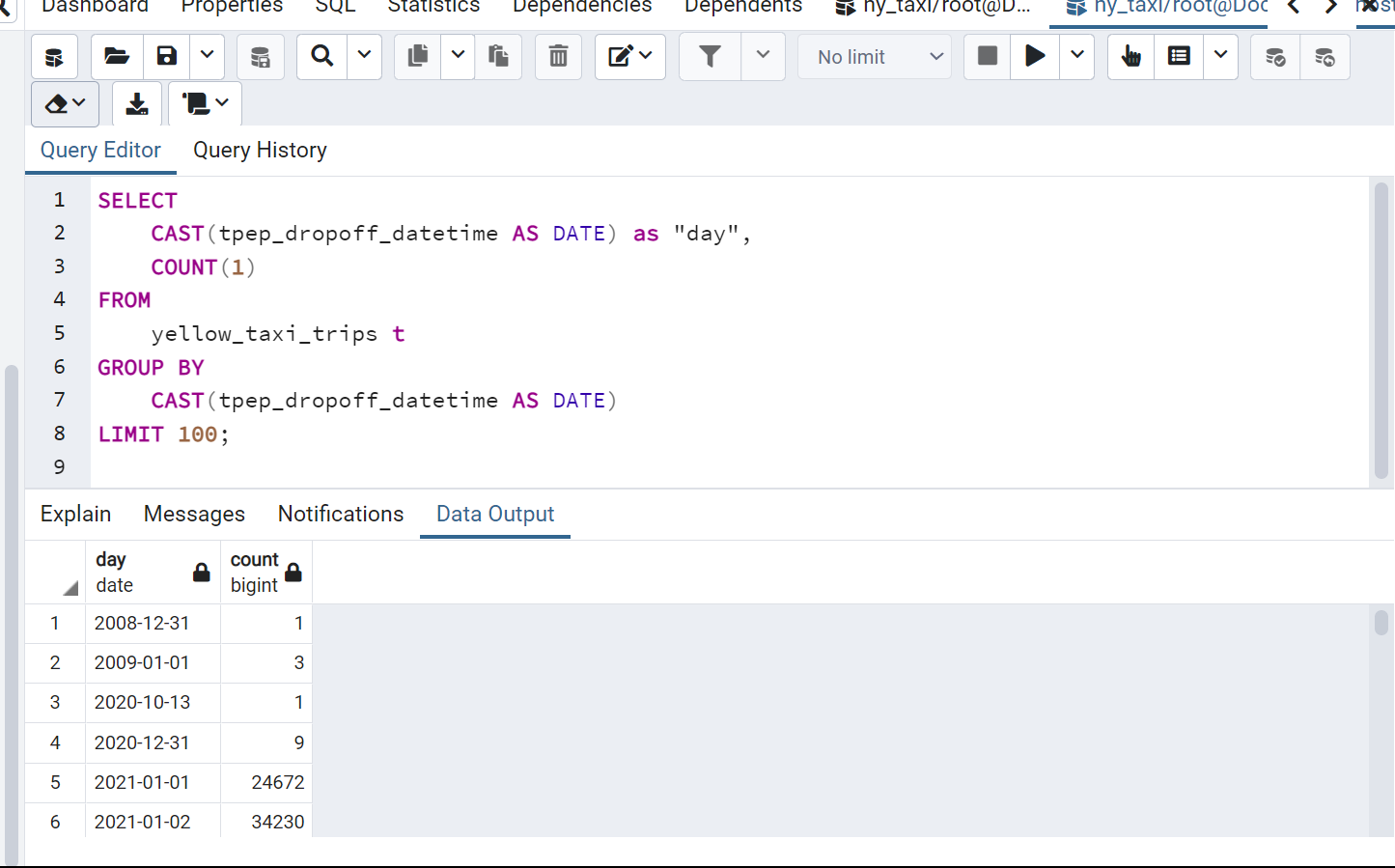


Better alternative is to cast as date:

1. SELECT
2. tpep\_pickup\_datetime,
3. tpep\_dropoff\_datetime,
4. total\_amount,
5. CAST(tpep\_dropoff\_datetime AS DATE)
7. FROM
8. yellow\_taxi\_trips t
9. LIMIT 100;



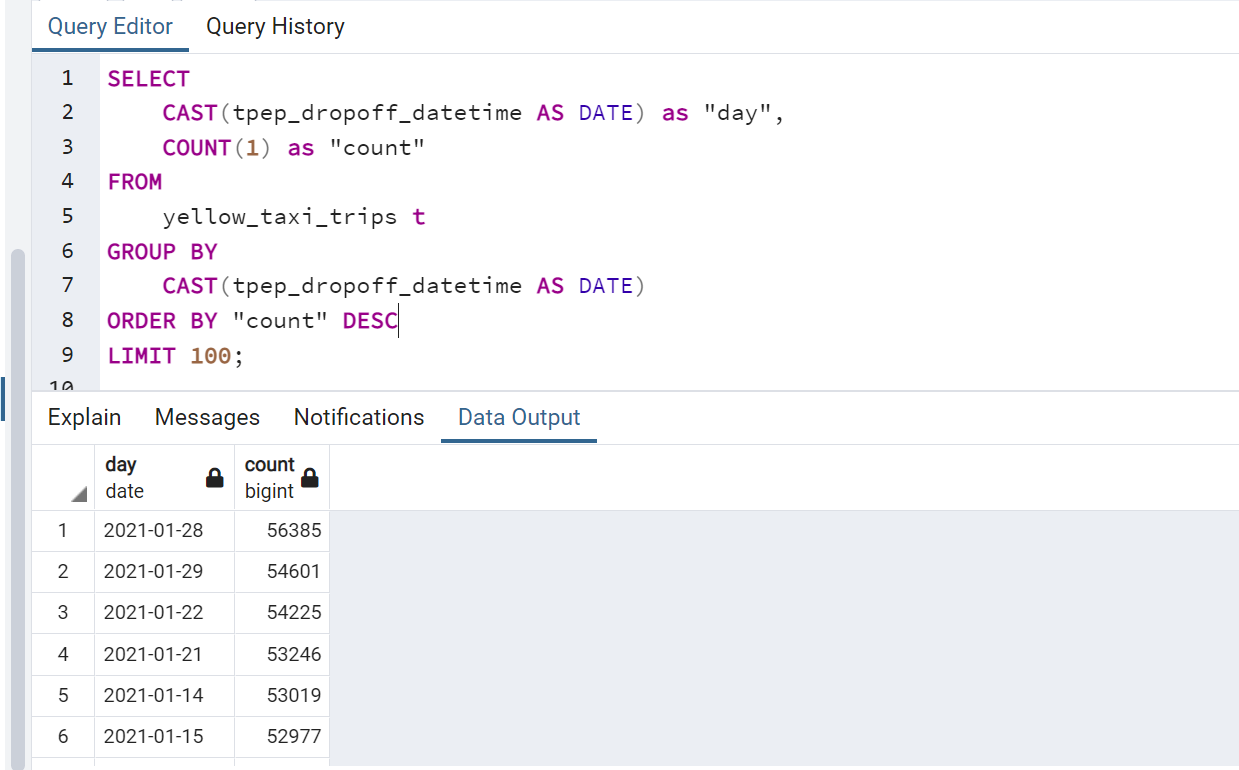
Continue: 12:16



^Group by date basically, but I don’t understand what COUNT(1) does actually..

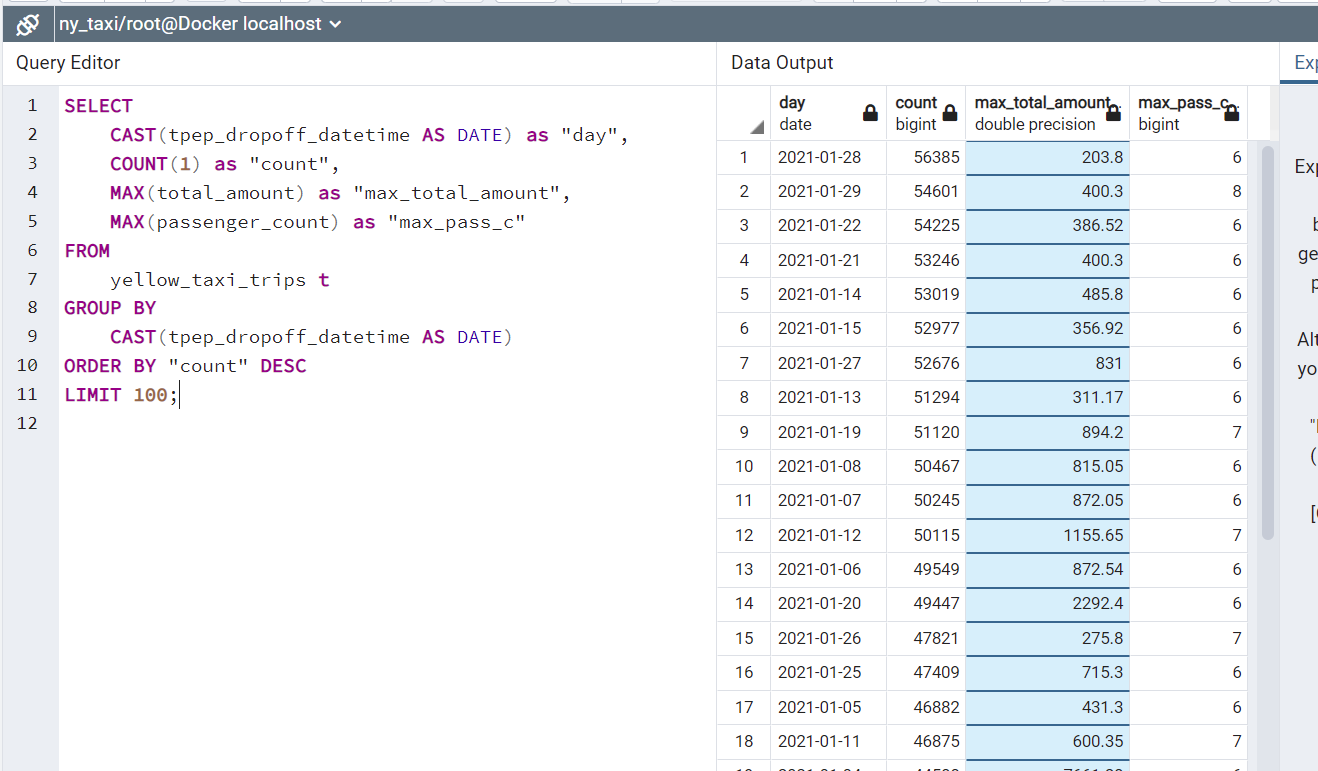
Now I do, since u group by date, the count is just to count how many records fall under that particular date!!

1. SELECT
2. CAST(tpep\_dropoff\_datetime AS DATE) as "day",
3. COUNT(1)
4. FROM
5. yellow\_taxi\_trips t
6. GROUP BY
7. CAST(tpep\_dropoff\_datetime AS DATE)
8. LIMIT 100;

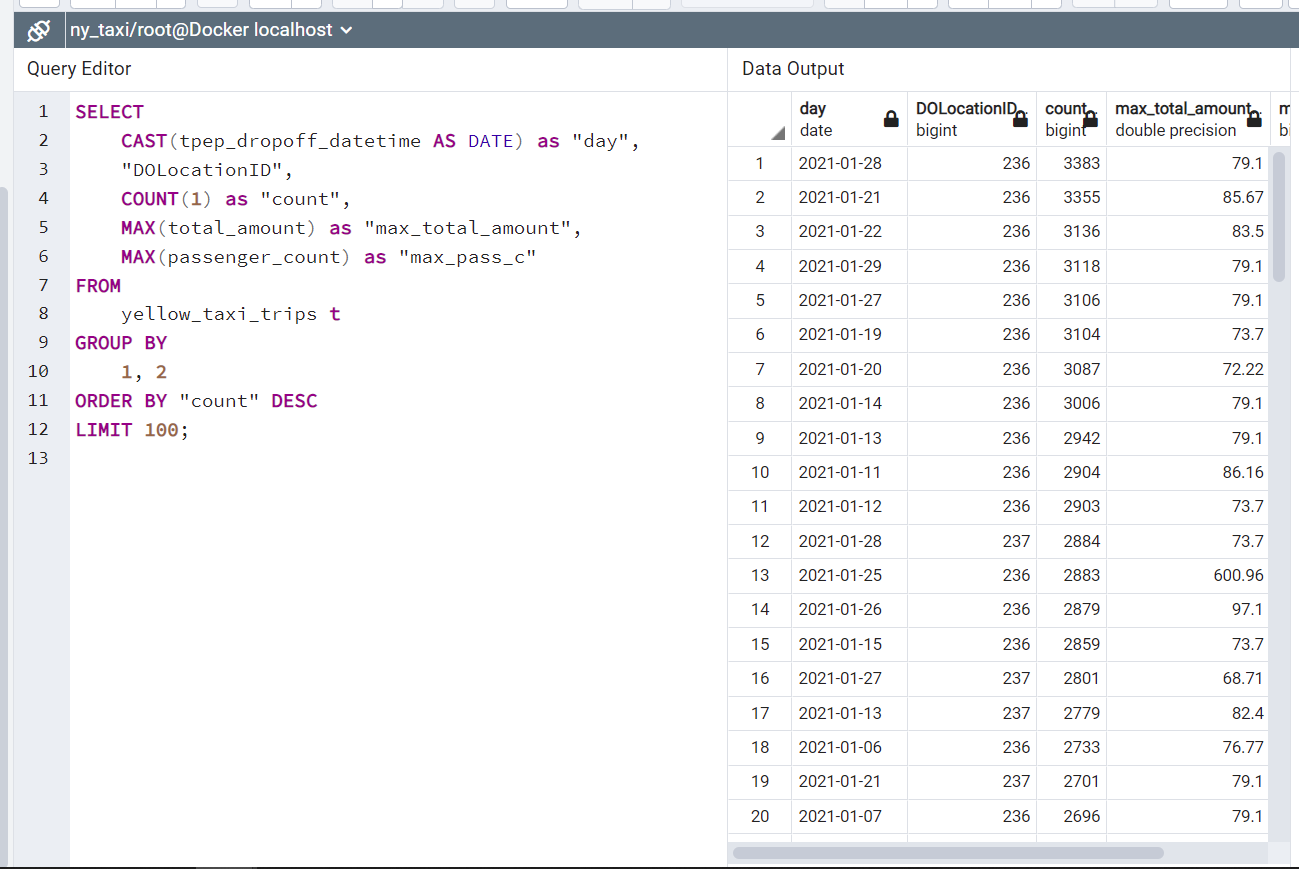


^Here we get count sorted in descending order.

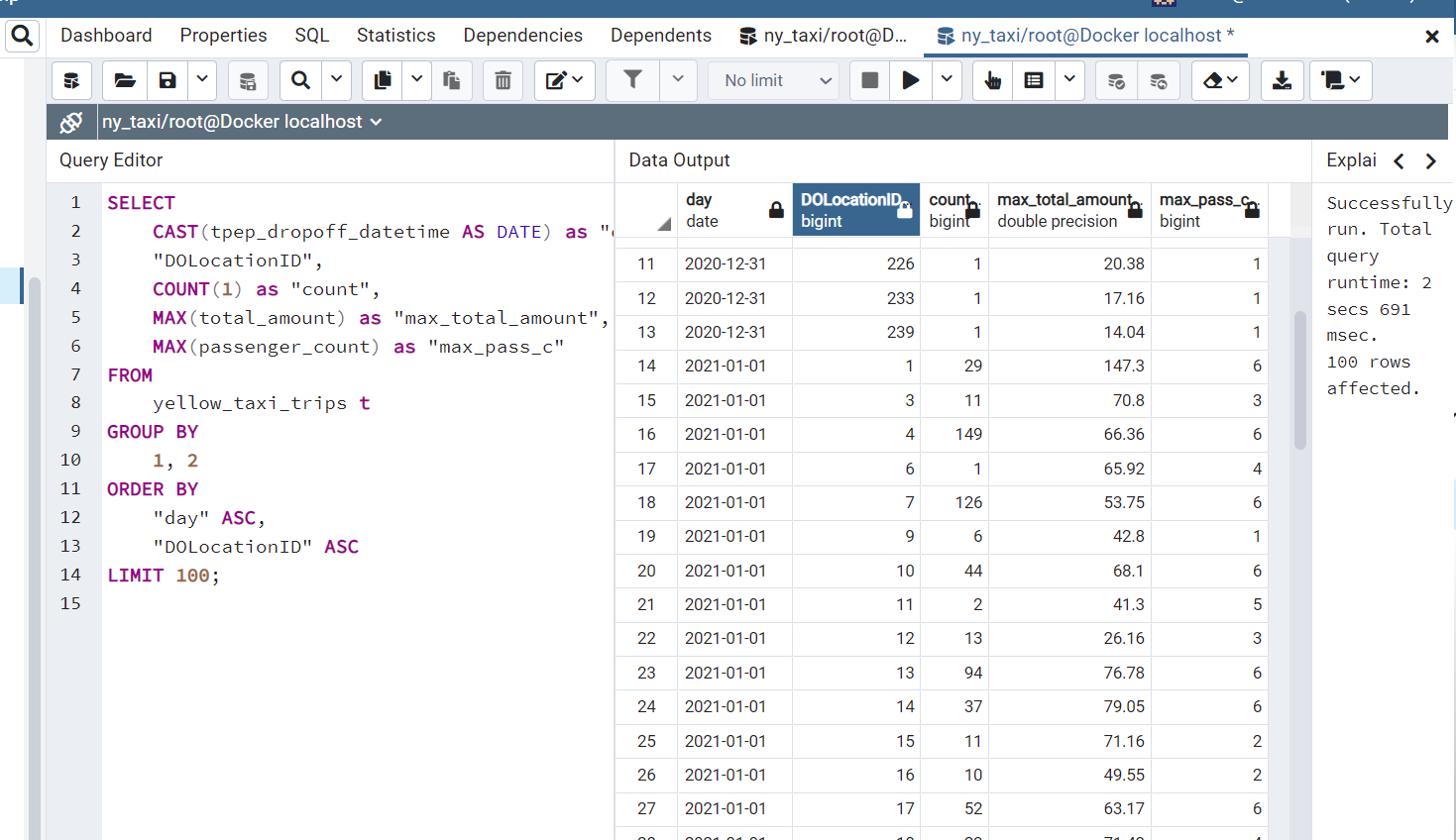
1. SELECT
2. CAST(tpep\_dropoff\_datetime AS DATE) as "day",
3. COUNT(1) as "count"
4. FROM
5. yellow\_taxi\_trips t
6. GROUP BY
7. CAST(tpep\_dropoff\_datetime AS DATE)
8. ORDER BY "count" DESC
9. LIMIT 100;



^get max number, pretty cool



^ GROUP BY 1, 2 (not sure what this means)



^ Sort by day and then DOLocationID

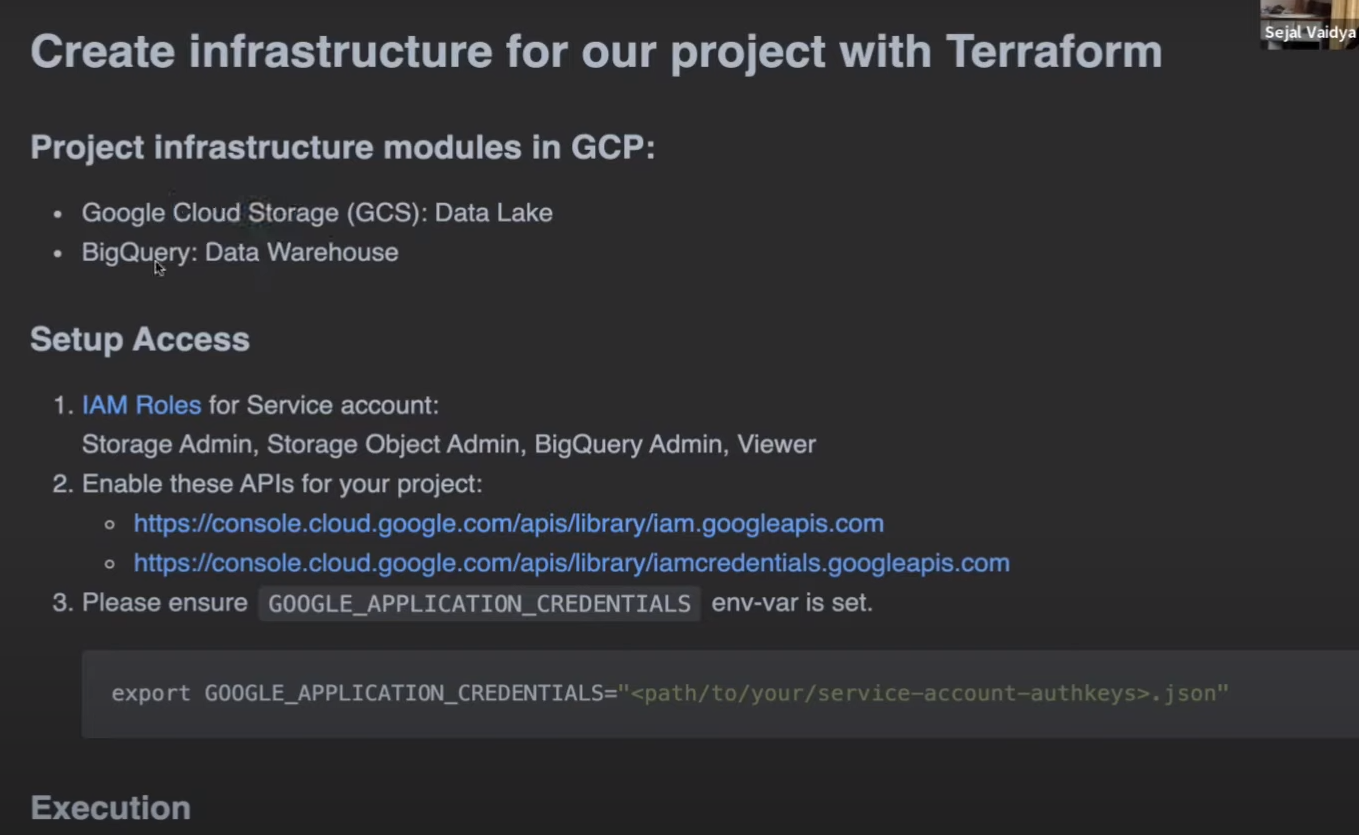
1. SELECT
2. CAST(tpep\_dropoff\_datetime AS DATE) as "day",
3. "DOLocationID",
4. COUNT(1) as "count",
5. MAX(total\_amount) as "max\_total\_amount",
6. MAX(passenger\_count) as "max\_pass\_c"
7. FROM
8. yellow\_taxi\_trips t
9. GROUP BY
10. 1, 2
11. ORDER BY
12. "day" ASC,
13. "DOLocationID" ASC
14. LIMIT 100;

# Lesson 1.3.1 Terraform concepts and GCP pre requisites.

<https://github.com/DataTalksClub/data-engineering-zoomcamp/blob/main/week_1_basics_n_setup/1_terraform_gcp/2_gcp_overview.md>

The first part is only about Google CLOUD SDK authentication.

The core part, what we are gonna do is here:



# Lesson 1.3.2 Creating GCP Infrastructure with Terraform

Explain the terraform files,

Then basically terraform only has got few commands that you can run.

Continue on 13:49 to see how to use terraform…

Terraform is just for configuration, its like a configuration template.

After terraform, we should have a big query schema, dataset, google cloud storage bucket.

On the left panel, you can select cloud storage and big query to view what you just created.

Do not commit variables.tf It contains ur google account credentials!!

# Homework

Postgress execution order.. So better do the AS again. .

Q3:   
How many taxi trips were there on January 15? 53024

1. SELECT
2. CAST(tpep\_dropoff\_datetime AS DATE) AS drop\_off\_date,
3. COUNT(1) AS dropoff\_count
4. FROM
5. yellow\_taxi\_trips
6. GROUP BY
7. CAST(tpep\_dropoff\_datetime AS DATE)
8. HAVING
9. CAST(tpep\_dropoff\_datetime AS DATE) = '2021-01-15'
11. SELECT
12. CAST(tpep\_pickup\_datetime AS DATE) AS drop\_off\_date,
13. COUNT(1) AS dropoff\_count
14. FROM
15. yellow\_taxi\_trips
16. WHERE
17. CAST(tpep\_pickup\_datetime AS DATE) = '2021-01-15'
18. GROUP BY
19. CAST(tpep\_pickup\_datetime AS DATE)

## Q4:

On which day it was the largest tip in January? (note: it's not a typo, it's "tip", not "trip") [2021-01-20]

1. SELECT
2. CAST(tpep\_pickup\_datetime AS DATE) as "Day",
3. MAX(tip\_amount) as Tips
4. FROM
5. yellow\_taxi\_trips
6. WHERE
7. EXTRACT(MONTH FROM (CAST(tpep\_pickup\_datetime AS DATE))) = 1
8. GROUP BY
9. CAST(tpep\_pickup\_datetime AS DATE)
10. ORDER BY
11. Tips DESC

## Q5:

What was the most popular destination for passengers picked up in central park on January 14? Enter the zone name (not id). If the zone name is unknown (missing), write "Unknown".

[Upper East Side South]

1. SELECT trip\_distance
2. FROM yellow\_taxi\_trips y
3. LEFT JOIN zones z
4. ON y."PULocationID" = z."LocationID"

This is just for testing but why do I need the double quote???

I cant even select the column “PULocationID”

I noticed whenever I have a column name that starts with capital letter I have to use double quote!!

Conditions:

Date = January 14

Zone = Central Park

y."PULocationID", y.passenger\_count, z.service\_zone, z."Zone", z."Borough",

1. SELECT
2. z.service\_zone,
3. COUNT(1)
4. FROM yellow\_taxi\_trips y
5. LEFT JOIN zones z
6. ON y."PULocationID" = z."LocationID"
7. WHERE
8. CAST(tpep\_pickup\_datetime AS DATE) = '2021-01-14' AND
9. z."Zone" = 'Central Park'
10. GROUP BY
11. z.service\_zone



1. SELECT
2. zdo."Zone" as "DropOff",
3. COUNT(1)
4. FROM yellow\_taxi\_trips y
5. LEFT JOIN zones zpu
6. ON y."PULocationID" = zpu."LocationID"
7. LEFT JOIN zones zdo
8. ON y."DOLocationID" = zdo."LocationID"
10. WHERE
11. CAST(tpep\_pickup\_datetime AS DATE) = '2021-01-14' AND
12. zpu."Zone" = 'Central Park'
13. GROUP BY
14. zdo."Zone"
15. ORDER BY
16. COUNT(1) DESC

Finally got the answer for Q5, took me some time to figure out what the question is actually asking!!

For the query I should get the destination location, instead of the pickup location! DUMB me LOL.

I did filter for pick up, and then query for pick up and was wondering why it only shows that one zone. LOL.

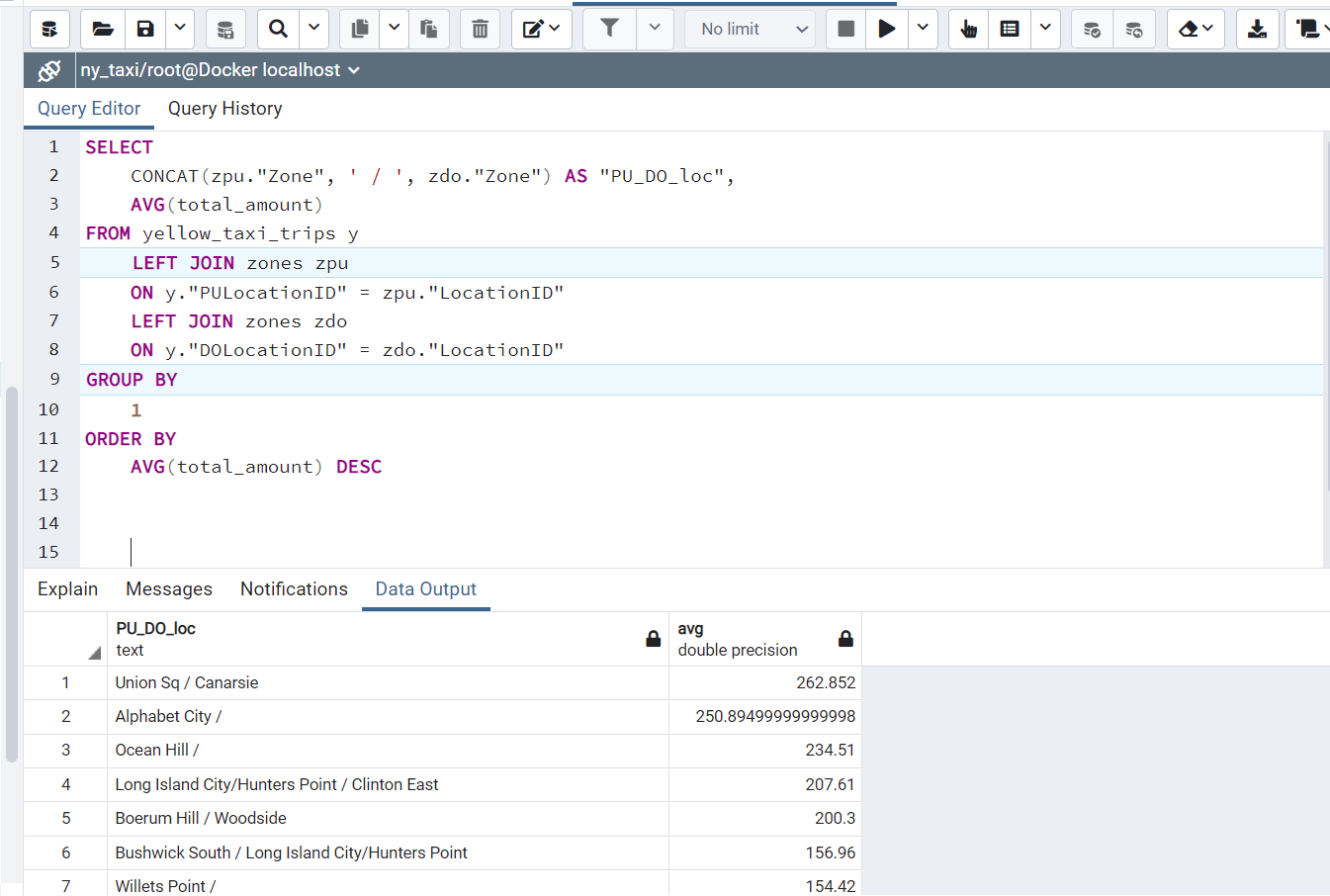
## Question 6: Most expensive route

What's the pickup-dropoff pair with the largest average price for a ride (calculated based on total\_amount)? Enter two zone names separated by a slash For example:"Jamaica Bay / Clinton East"If any of the zone names are unknown (missing), write "Unknown". For example, "Unknown / Clinton East".

[Union Sq/Canarsie]

1. SELECT
2. tpep\_pickup\_datetime,
3. tpep\_dropoff\_datetime,
4. total\_amount,
5. CONCAT(zpu."Zone", ' / ', zdo."Zone") AS "PU\_DO\_loc"
7. FROM yellow\_taxi\_trips y
8. LEFT JOIN zones zpu
9. ON y."PULocationID" = zpu."LocationID"
10. LEFT JOIN zones zdo
11. ON y."DOLocationID" = zdo."LocationID"
13. ORDER BY
14. total\_amount DESC

^First step, I got all the thing that I need, next need to get average price for each pickup-dropoff pair I suppose.



1. SELECT
2. CONCAT(zpu."Zone", ' / ', zdo."Zone") AS "PU\_DO\_loc",
3. AVG(total\_amount)
4. FROM yellow\_taxi\_trips y
5. LEFT JOIN zones zpu
6. ON y."PULocationID" = zpu."LocationID"
7. LEFT JOIN zones zdo
8. ON y."DOLocationID" = zdo."LocationID"
9. GROUP BY
10. 1
11. ORDER BY
12. AVG(total\_amount) DESC

Yes, got the answer right! Notice above and below the different is GROUP BY (results is the same), with 1 its referring to column 1. Below implementation is more explicit but could be messy.

1. SELECT
2. CONCAT(zpu."Zone", ' / ', zdo."Zone") AS "PU\_DO\_loc",
3. AVG(total\_amount)
4. FROM yellow\_taxi\_trips y
5. LEFT JOIN zones zpu
6. ON y."PULocationID" = zpu."LocationID"
7. LEFT JOIN zones zdo
8. ON y."DOLocationID" = zdo."LocationID"
9. GROUP BY
10. CONCAT(zpu."Zone", ' / ', zdo."Zone")
11. ORDER BY
12. AVG(total\_amount) DESC

Ok so, apparently my ans is not the same with Jerome and I need to account for Null values with COALESCE or something…

# Lesson 1.4.1 Setting up env on google cloud (Cloud virtual machine and SSH access)

1. Generate SSH key

$ cd .ssh/

$ ssh-keygen -t rsa -f gpc -C cjj -b 2048 (now we have generated SSH key)

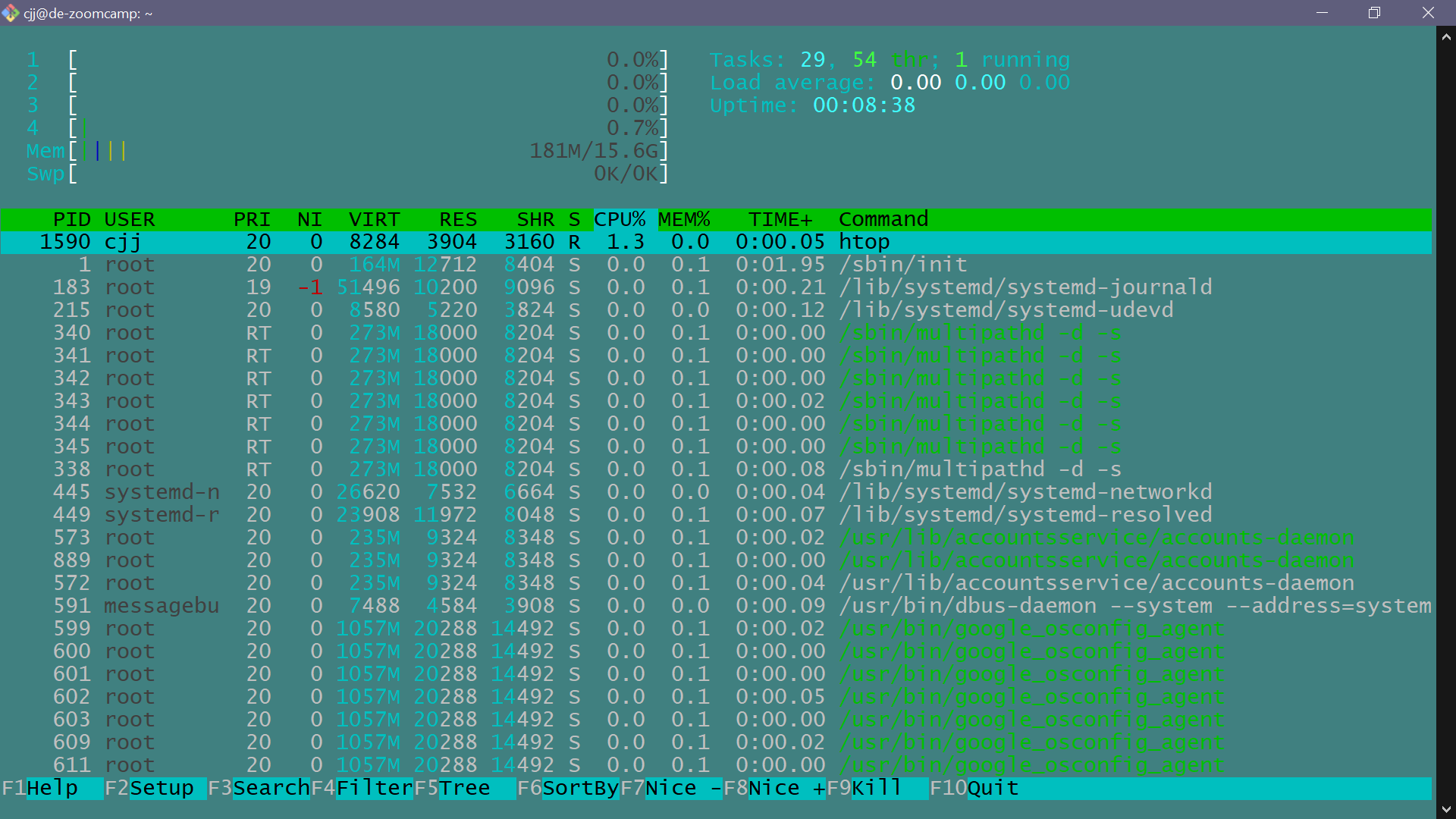
$ cat gpc.pub (this is to show the public key)

1. Add virtual machine on google cloud
2. Connect to VM on local machine.

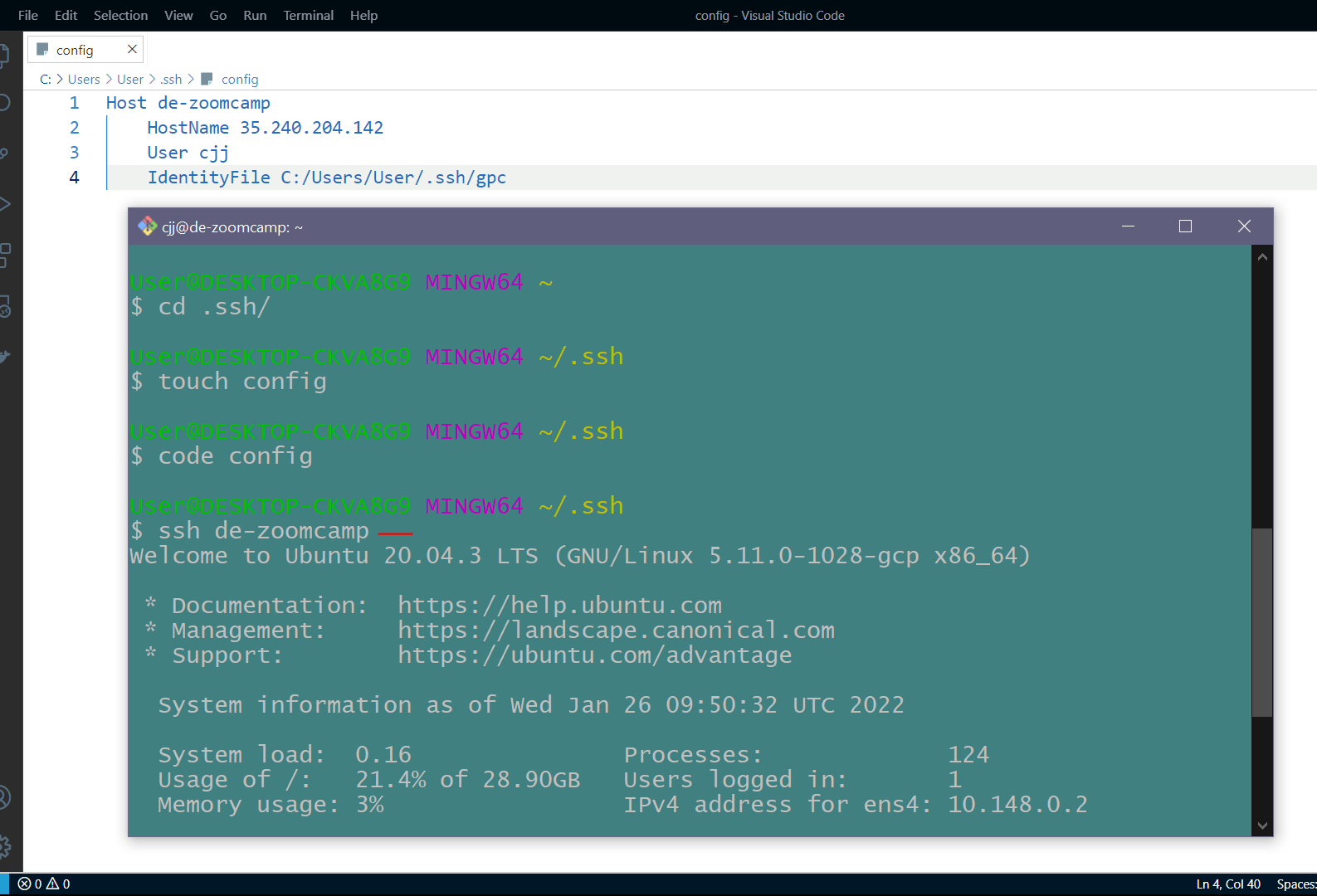
$ ssh -i ~/.ssh/gcp [cjj@35.240.204.142](mailto:cjj@35.240.204.142) (link it to the virtual env created above

$ ssh -i ~/.ssh/gpc [cjj@35.240.204.142](mailto:cjj@35.240.204.142) (now we enter the VM)

$htop (cool!- here we can see we have quadcore and 16gb, just like what we specified during the configuration!!)



## Installation and set up (Conda docker in VM)



Set up a config file so we can login easily to the VM just by running

$ssh de-zoomcamp

I made a mistake when installing conda in the VM:

Download conda

$wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86\_64.sh

$ bash Anaconda3-2021.11-Linux-x86\_64.sh

Conda is now working, just $logout and $ssh-dezoomcamp

Next we install docker

We need to update all packages first!

$ sudo apt-get update

$ sudo apt-get install docker.io

We install a plugin so that our VScode can connect to virtual machine!

Then connect to host and select our VM (de-zoomcamp)!

Next download docker and docker compose:

Docker compose installation:

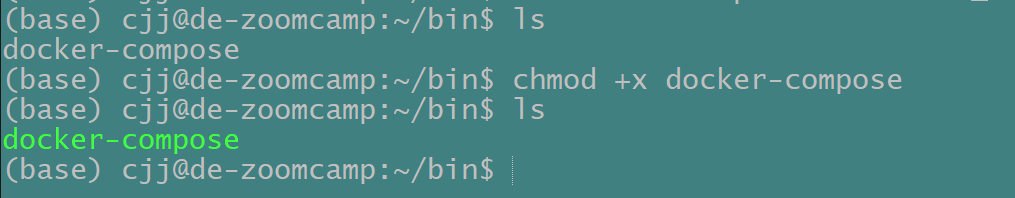
https://github.com/docker/compose/releases

Copy the docker-compose-linux-x86\_64 download URL Link

Then download with

$wget https://github.com/docker/compose/releases/download/v2.2.3/docker-compose-linux-x86\_64 -O docker-compose

Next change it to executable file



Then we use $nano .bashrc

And add this:



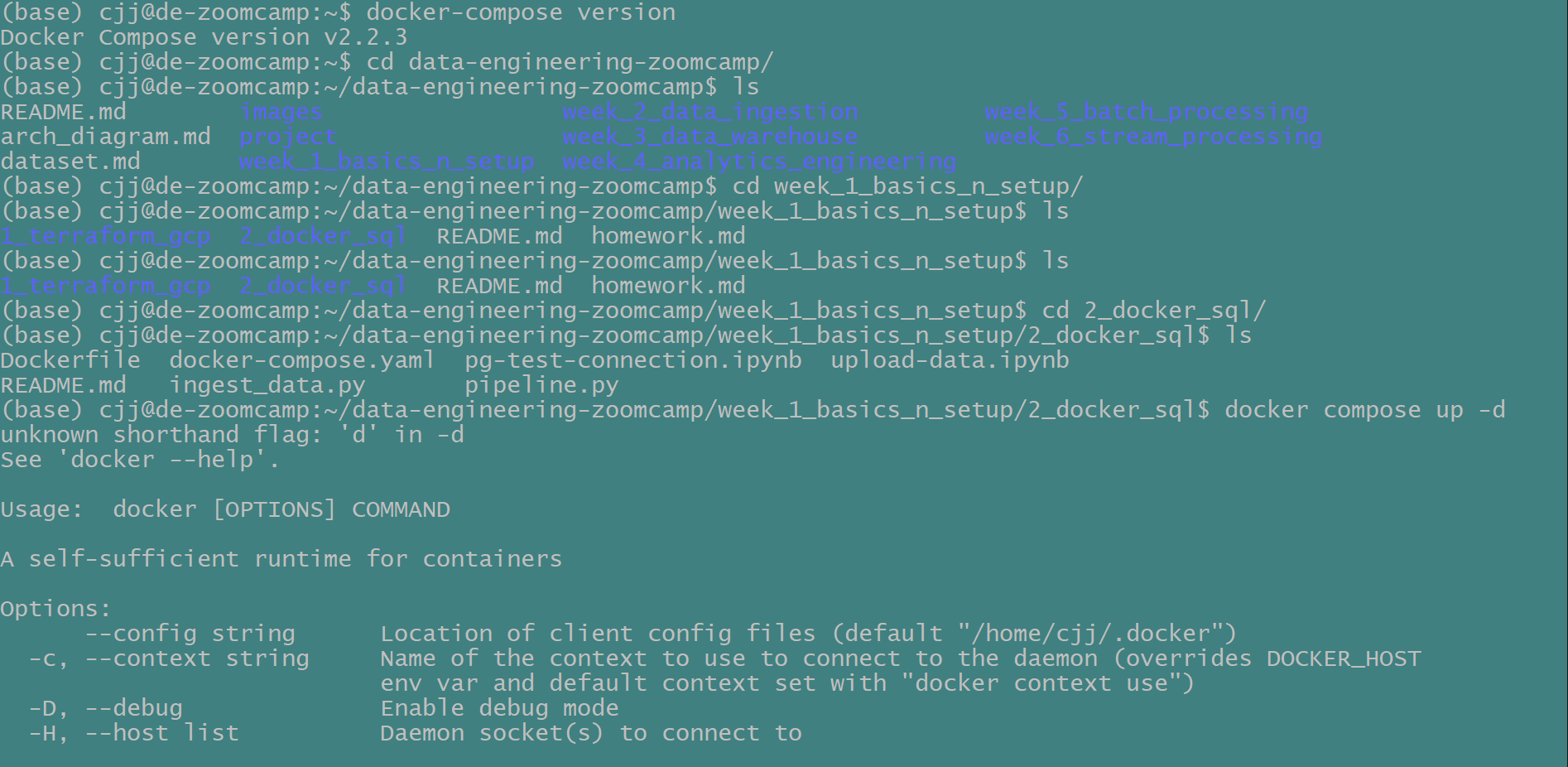
Basically add to path.

In nano we can use control o to save, control x to exit!

After pgcli installation, now we can run pgcli (do remember our docker compose is still running in detached mode).

$pgcli -h localhost -U root -d ny\_taxi

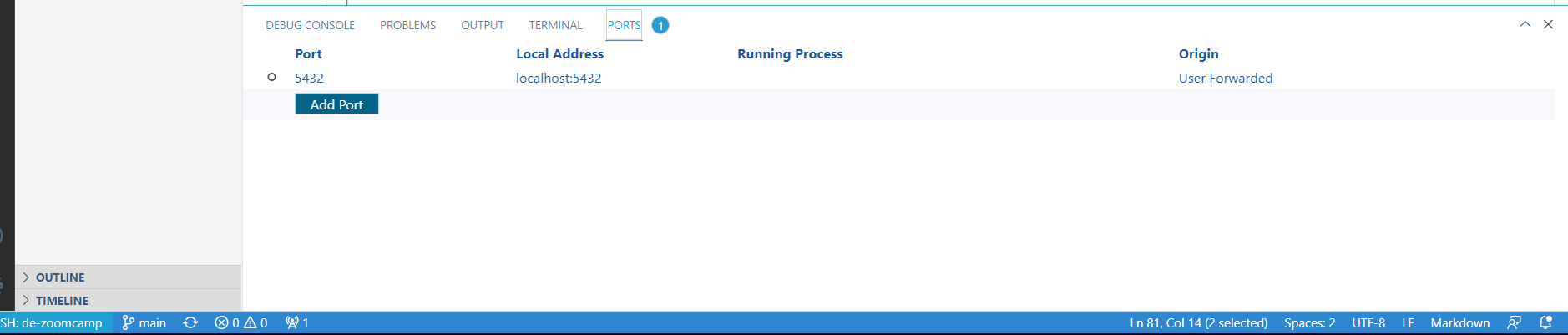
## Docker compose up



## Port forwarding (VM+local)

### pgcli

Now we need to forward the port (VM) to local machine, so that we can interact with that port that is running postgres instance locally.



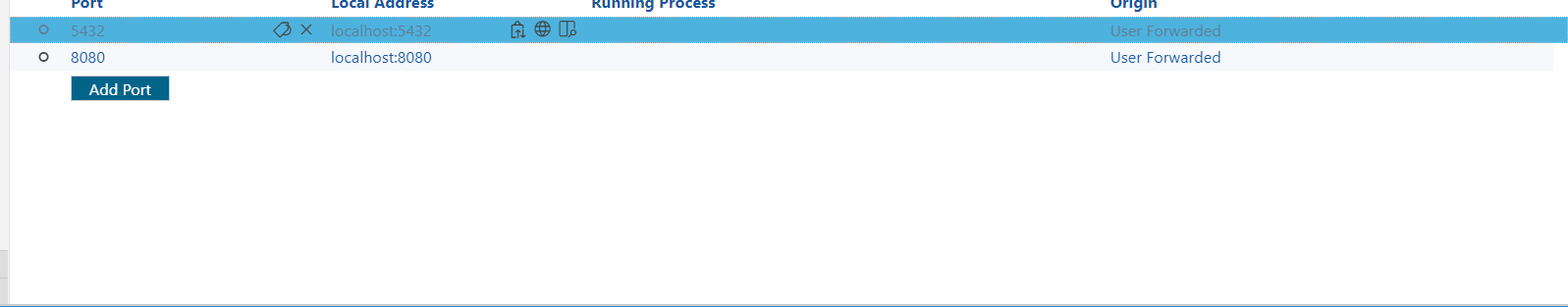
Now we do port forwarding (in vscode)!

We can then run (on local git bash!!)

$ pgcli -h localhost -U root -d ny\_taxi

Super cool!!

### Same thing for Pgadmin!



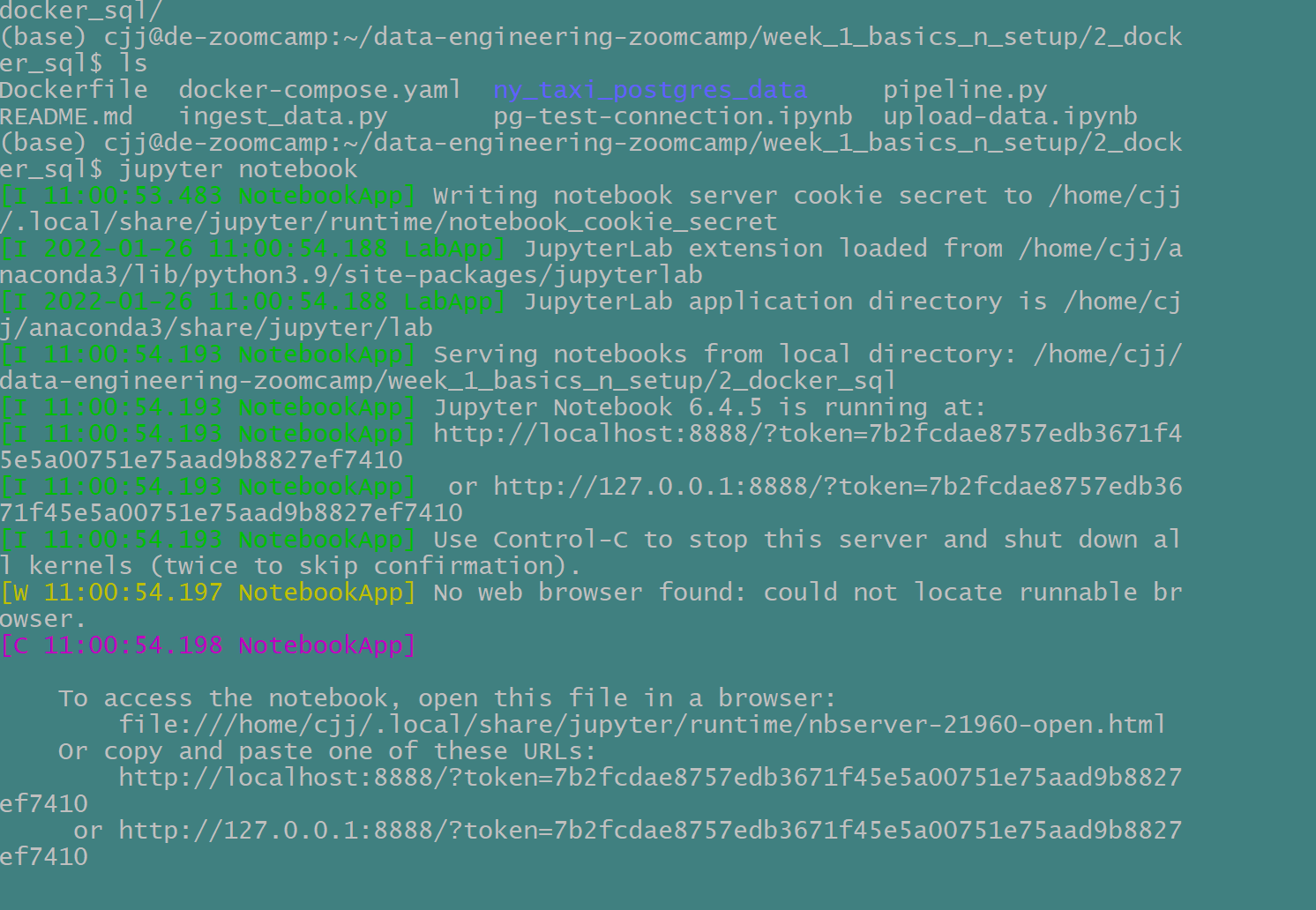
First we do port forwarding!

Then we access <http://localhost:8080/> on local machine!

### Same goes for Jupyter notebook

First we go the folder where we store our notebooks.

Then we do port forwarding in Vscoed (8888)



Then we copy the link here and paste in local machine browser!

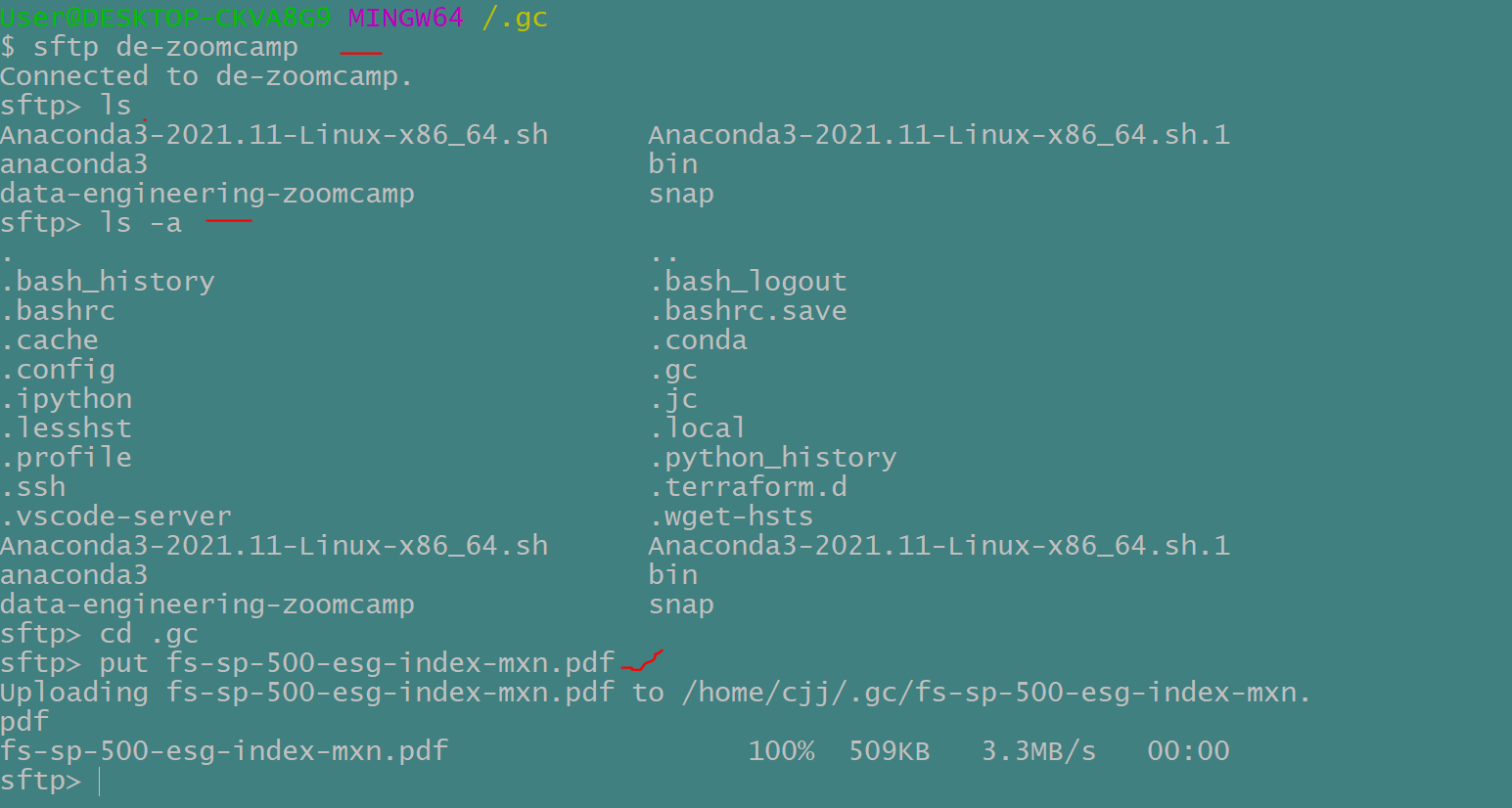
Lastly we also installed terraform

## Using SFTP to transfer file

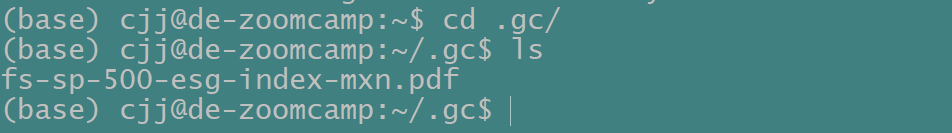
First I have to go into where my file is on local machine!



Next I start sftp and just put the file in! Literally!



Now it is in my virtual machine!!!



## Stop here

I stopped around 43.00, its basically about gcloud authentication..