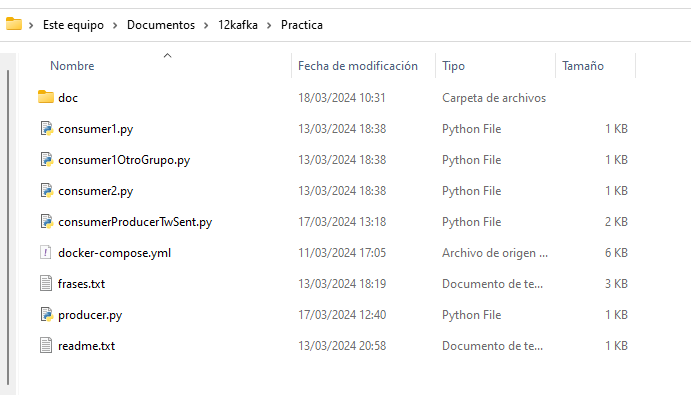
Profesor : Sergio.khayyat@gmail.com

Alumno: Alberto Martín

Descripción del proceso:

En el directorio del proyecto tengo los siguientes ficheros:



1. Readme.txt contiene información importante para desplegar la solución.
2. Fichero docker-compose.yml contiene la información para desplegar varios Docker de los cuales vamos a dar uso a algunos en esta práctica.
3. El fichero frases.txt contiene varias frases y palabras que van a ser evaluadas.
4. Tenemos los productores y los consumidores producer.py consumer\*.py.
5. En la carpeta doc, hay documentación en la que se incluye este documento junto con otra información relevante.

Una vez lanzamos el yml nos muestra que tenemos arrancada la infraestructura:

>Docker compose up -d

[+] Running 8/8

✔ Container zookeeper Started 7.1s

✔ Container mongo Started 5.1s

✔ Container broker Started 8.5s

✔ Container schema-registry Started 12.0s

✔ Container connect Started 14.5s

✔ Container ksqldb-server Started 19.6s

✔ Container control-center Started 23.9s

✔ Container ksqldb-cli Started

En mi entorno tengo Python 3.10.4

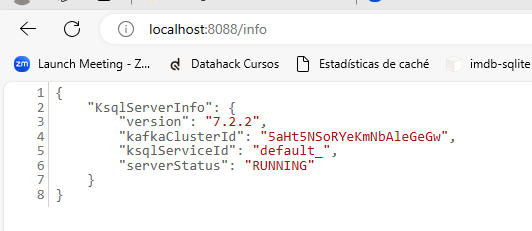
Para que funcione el código Python que tengo desarrollado, tenemos que instalar:

pip install kafka-python

pip isntall transformers

pip install torch

Comprobamos que está corriendo la ksqldb:



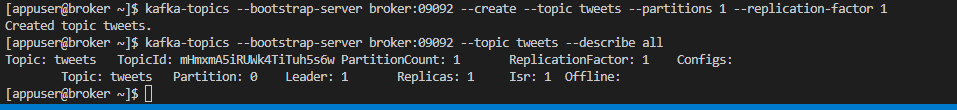
Primero creamos el topic, para hacerlo lanzamos lo siguiente desde un bróker de kafka:

>docker exec -it broker bash # para arrancar el broker

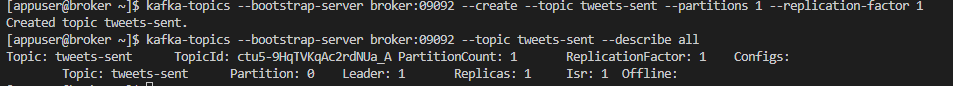
> kafka-topics --bootstrap-server broker:09092 --create --topic tweets --partitions 1 --replication-factor 1 # para crear el topic

Hemos creado sólo una partición, pero en un sistema de producción sería aconsejable crear 3 particiones.

Y comprobamos si está funcionando y tiene la partición que hemos creado:



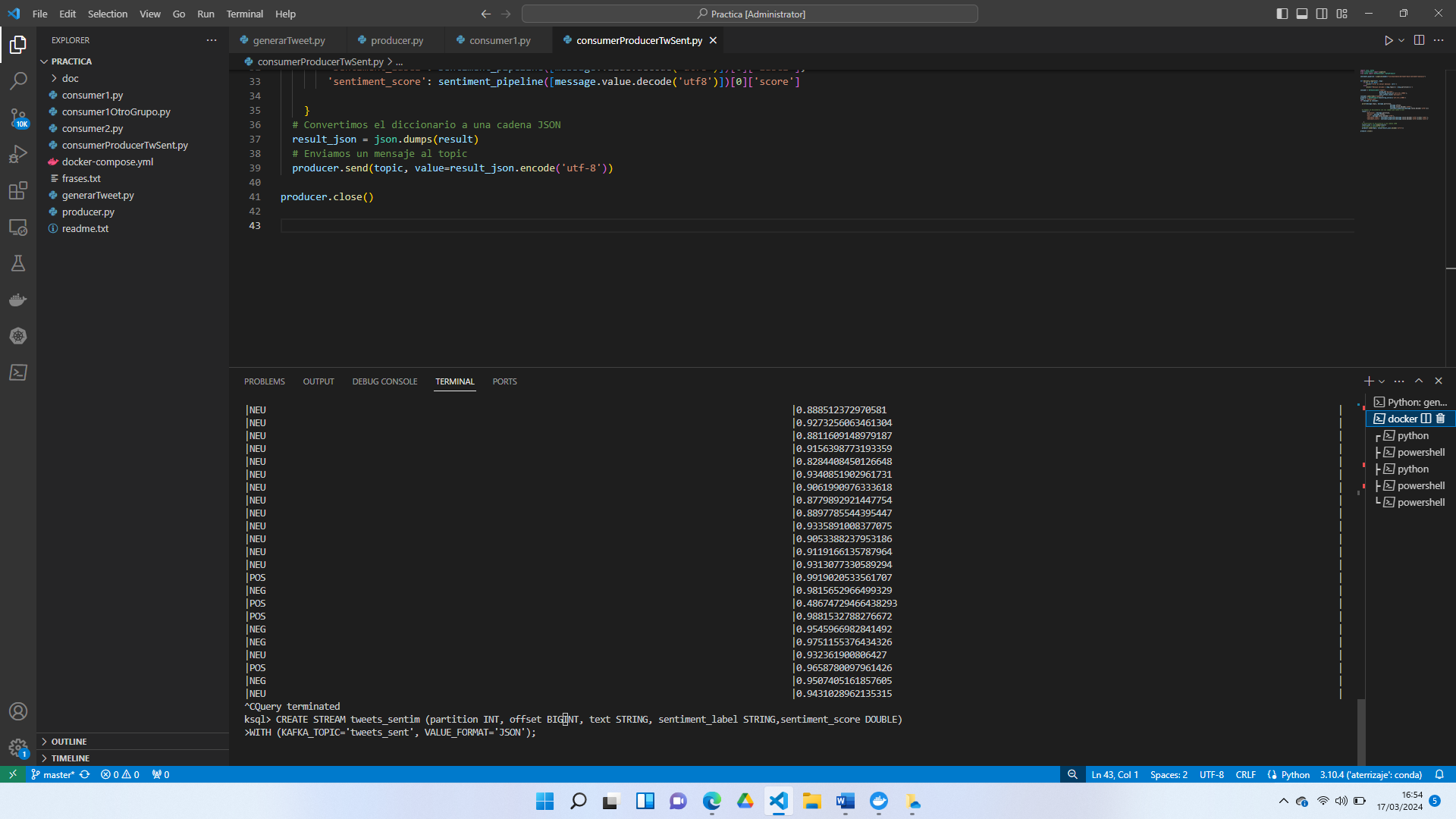
También creamos el otro topic que vamos a usar para tener los sentimientos.

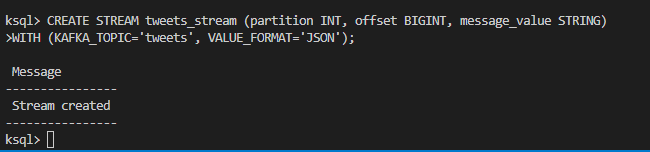


Ahora preparamos el entorno para poder grabar el topic de salida en ksqldb. Para hacerlo lanzamos el Docker correspondiente:

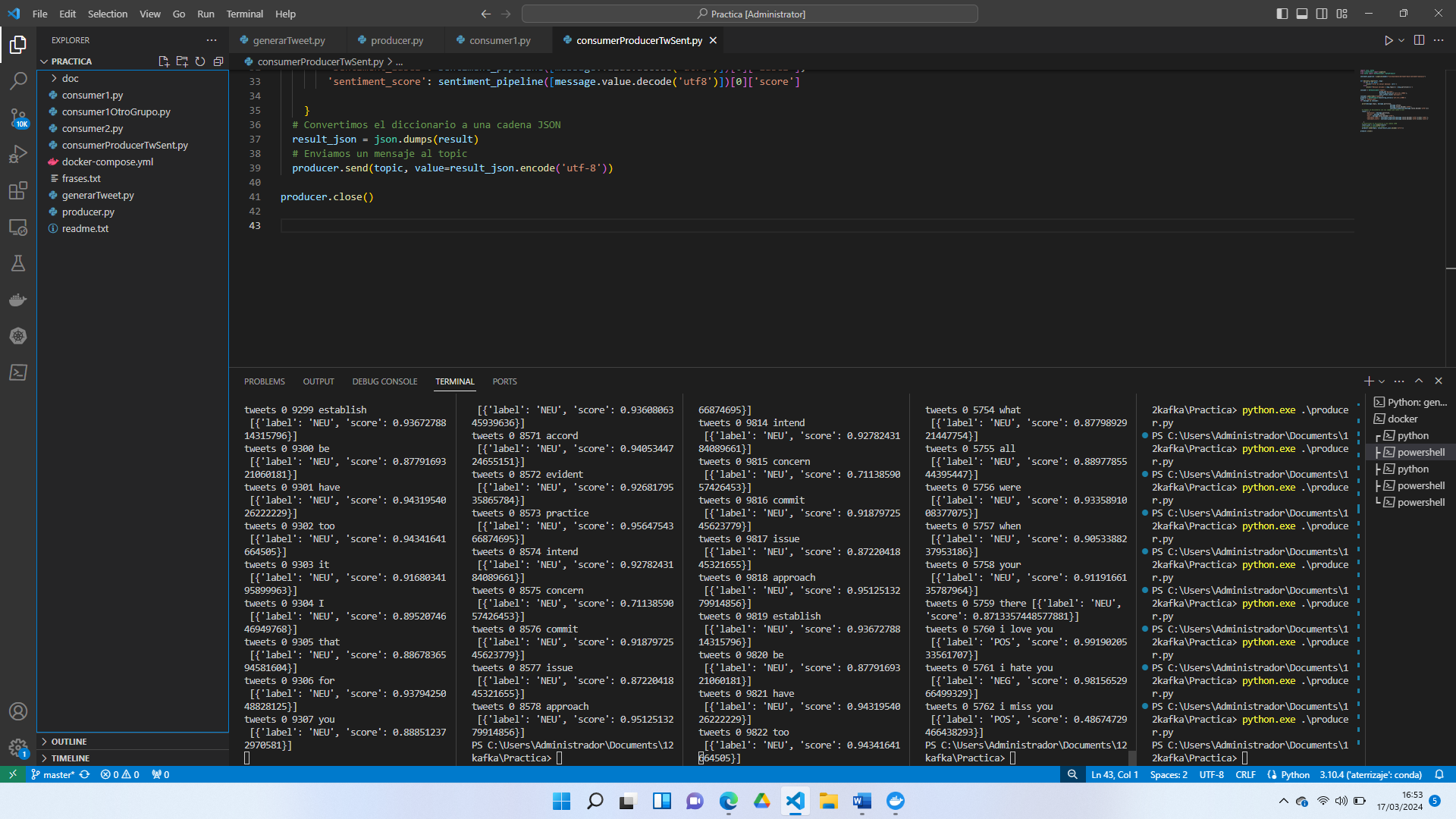
>docker exec -it ksqldb-cli ksql <http://ksqldb-server:8088>

Ahora vamos a crear un stream para guardar los datos del consumer que contiene el cálculo de los sentimientos:





Ahora lanzamos los Consumers y posteriormente el producer inicial y vemos como empiezan a cargar información en los topics:



He comprobado que los dos consumer que tienen el mismo grupo consumen la información de manera compartida, es decir unos mensajes los consume el consumer1 y otros el consumer2.

Este sería el orden de ejecución de los productores y consumidores:

>python .\consumer1.py

>python .\consumer2.py

>python .\consumer1OtroGrupo.py

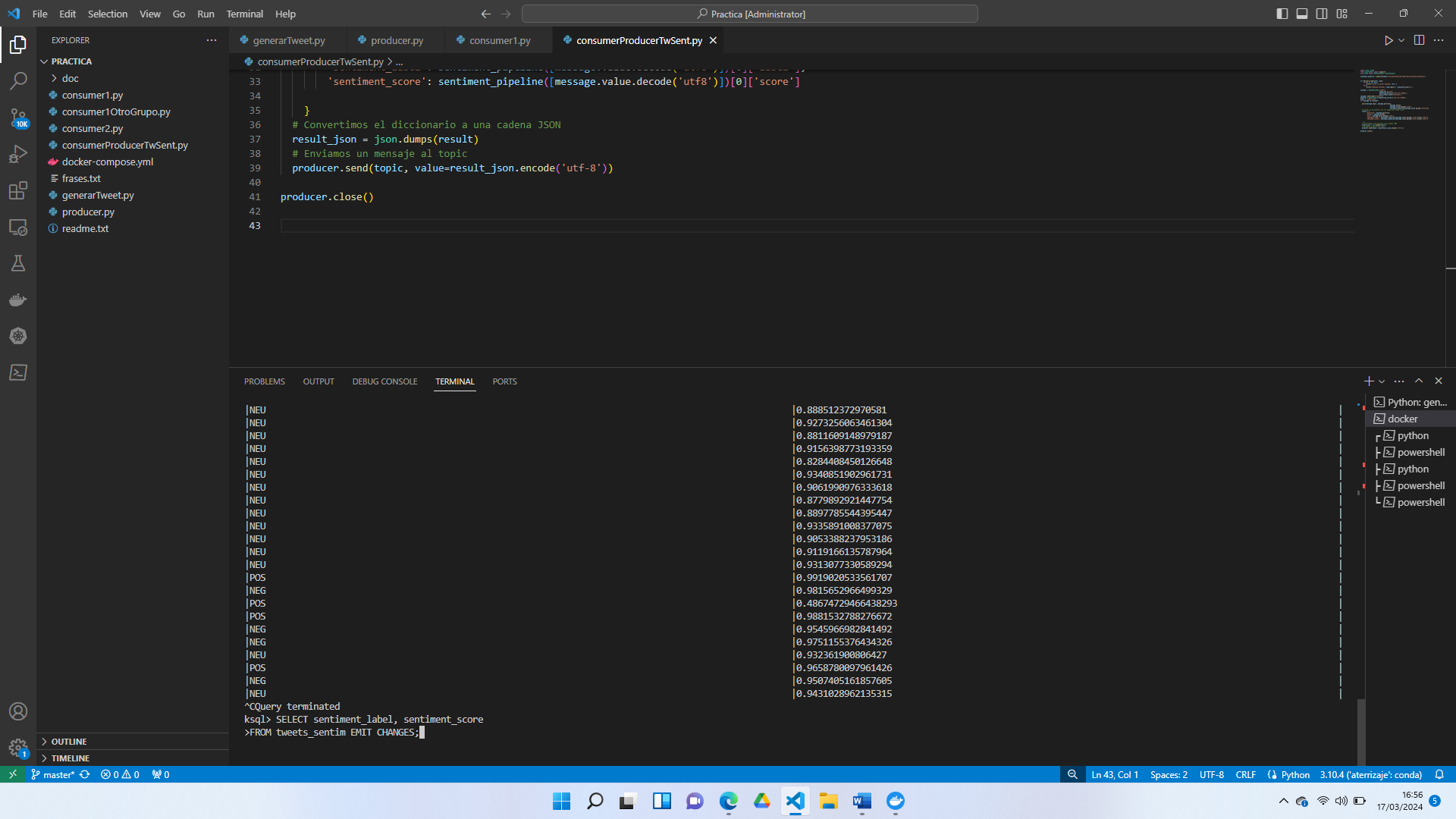
>python .\consumer1OtroGrupo.py

<python .\consumerProducerTwSent.py

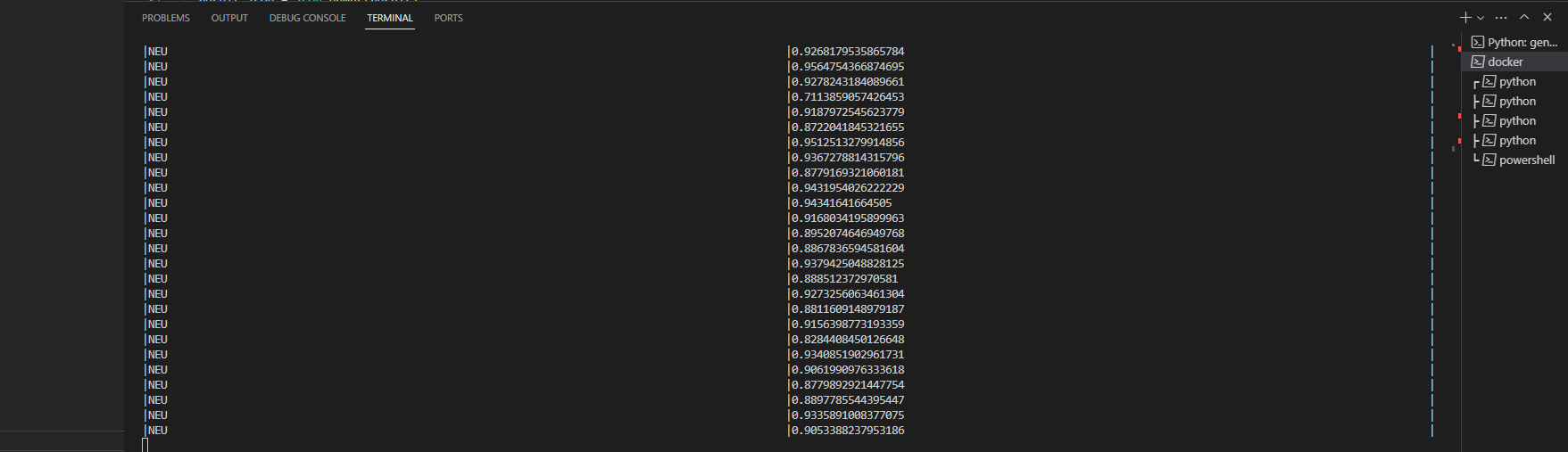
El siguiente paso es lanzar el producer. Para hacerlo:

>python .\producer.py

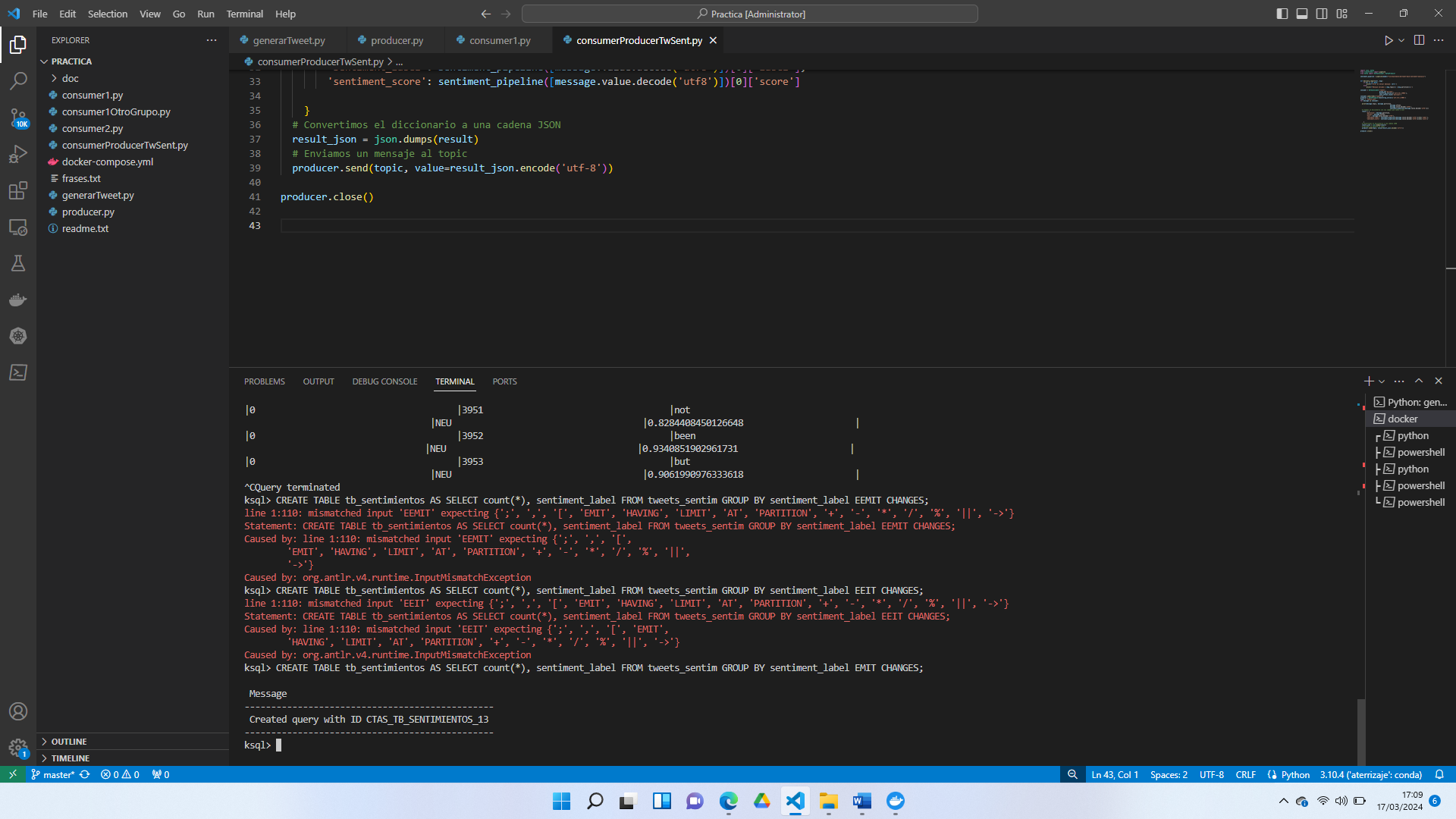
Una vez que tenemos el stream, podemos ver como carga la información según se va introduciendo:

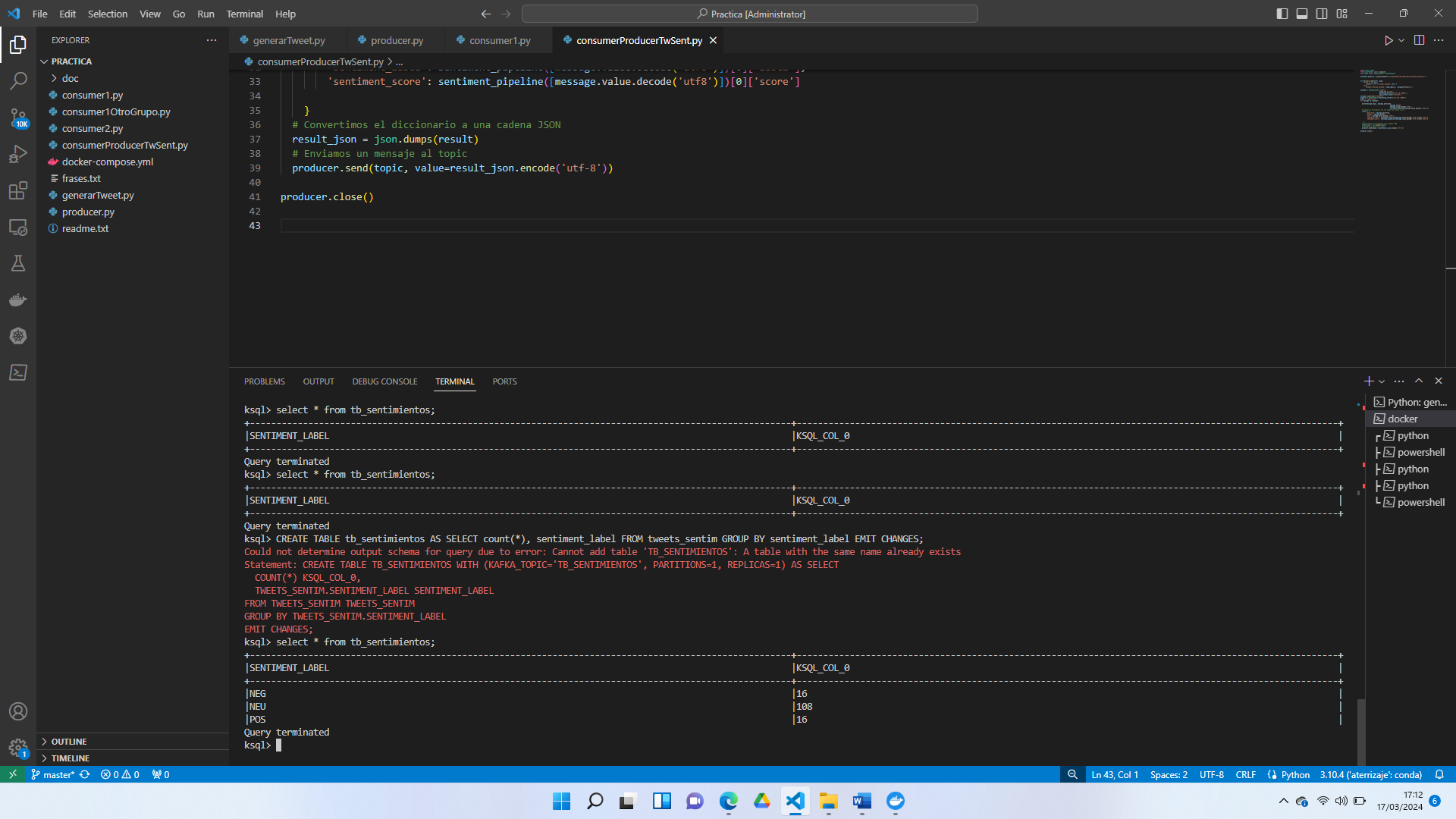


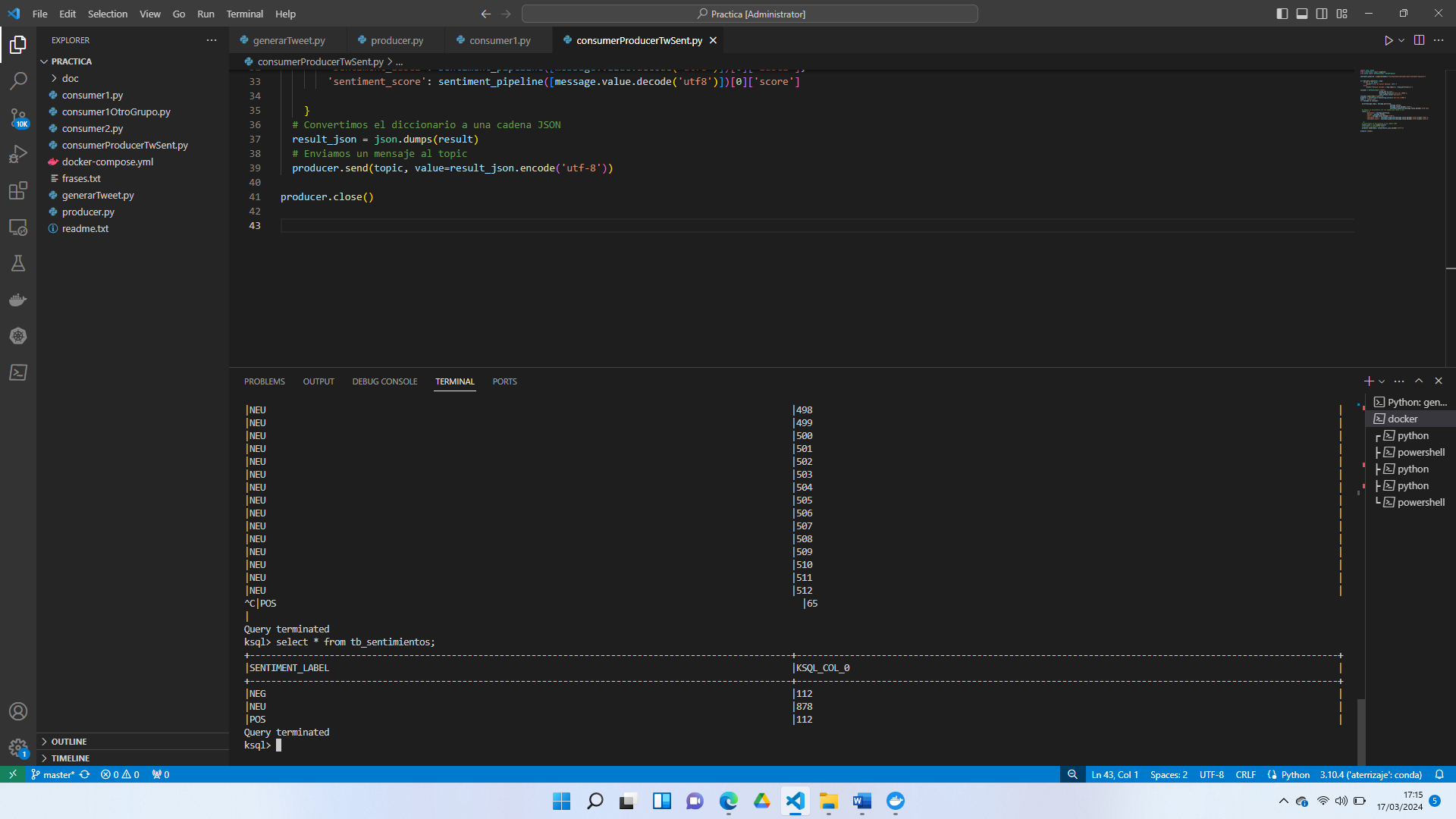
Este es el resultado donde podemos ver como carga online la información en el stream:



Ahora para ver el número de frases por sentimiento creamos una tabla con los datos agrupados:

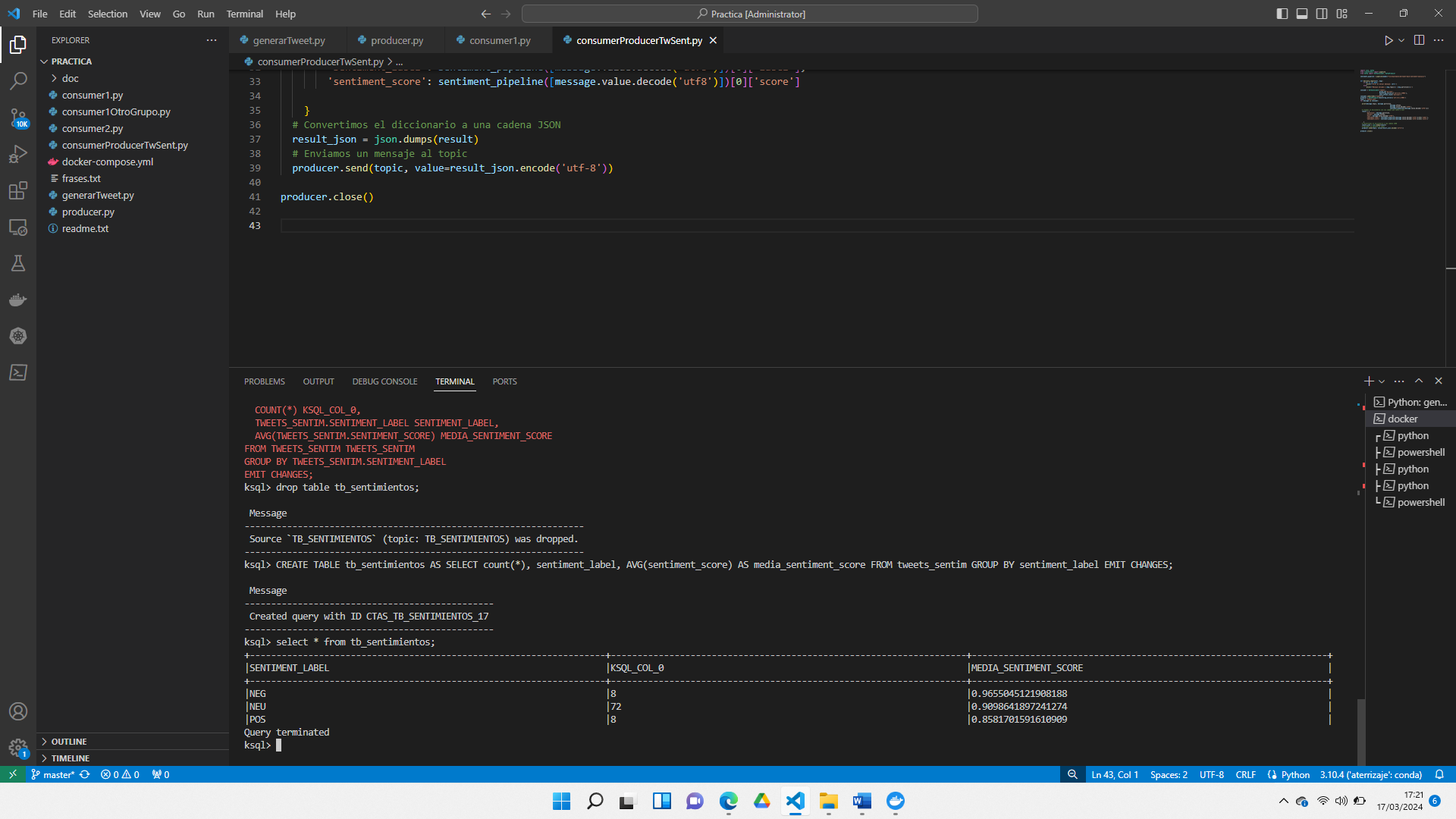


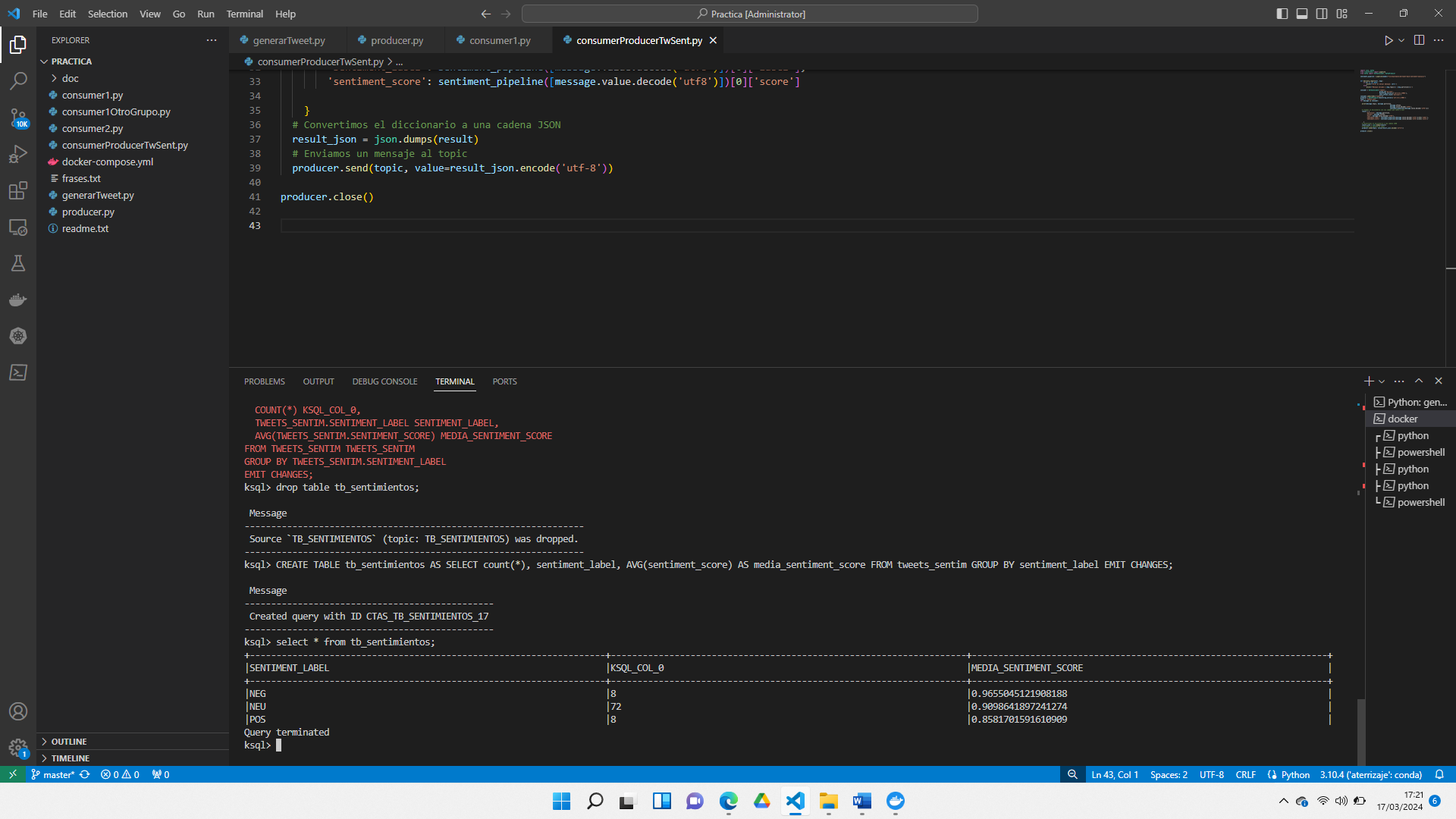




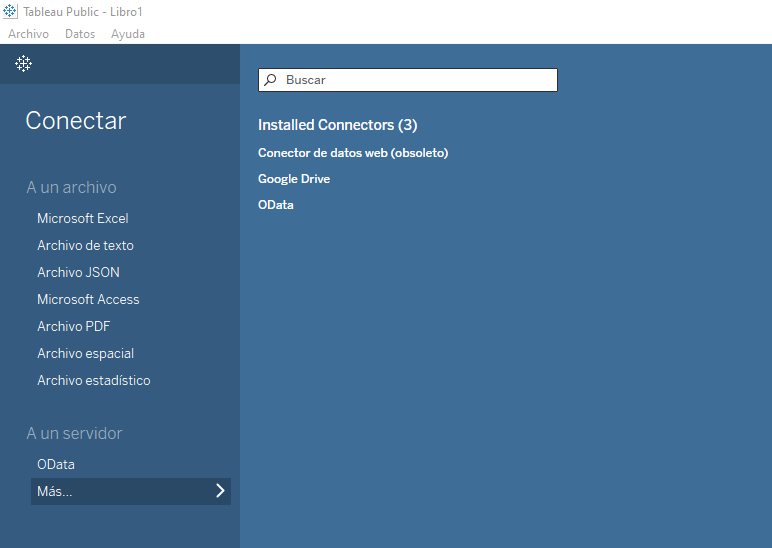
He borrado la tabla y la he creado una nueva tabla pero que incluya el campo score:

Drop table tb\_sentimientos;





Para mejorar la solución, deberíamos por un lado pasar esta información a una aplicación donde se pueda mejorar la salida de información de forma visual, tipo tableau. Lo he intentado, pero la versión que tenemos del master no nos deja importar de bases de datos sql:



También podríamos securizar cada uno de los topic y sus enlaces. Esto necesitaría mas tiempo para ver como hacerlo, ya que no lo hemos visto durante el curso.

El diseño de la arquitectura sería el siguiente:

