M1 SA PART 2

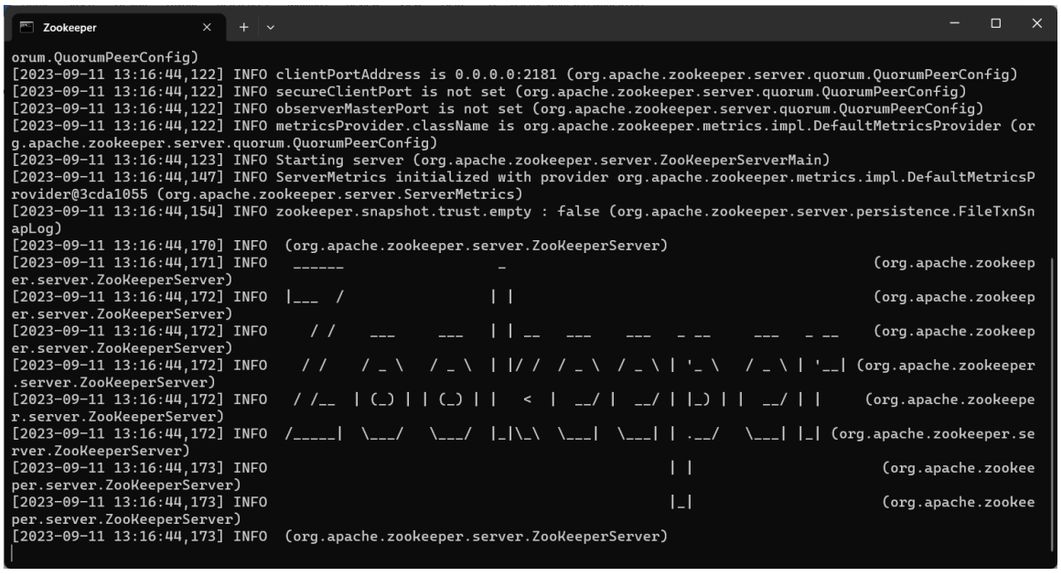
I. Running Kafka Zookeeper

Open command line (CLI). Access the kafka folder using cd command. In this case, kafka is accessed through the command **cd C:\kafka**

To start the zookeeper, access the zookeeper-server-start.bat file along with the zookeeper.properties file. The following command is run on the CLI to start the zookeeper:

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

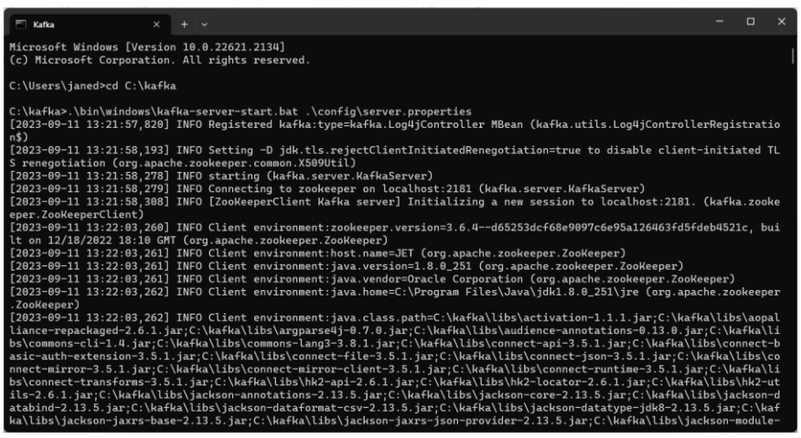
The zookeeper should now start. The output is shown below.



II. Running Kafka Server

Open a new CLI. Access the kafka folder with the cd command. Access the kafka-server-start.bat file along with server.properties file to run the kafka server. The command **.\bin\windows\kafka-server-start.bat .\config\server.properties** is entered in the CLI.

The kafka server now starts after executing the command. The output is shown below.



III. Create Topic

A topic must first be created before data can be produced by the producer and consumed by the consumer. A kafka topic is a virtual group where messages are contained in proper order.

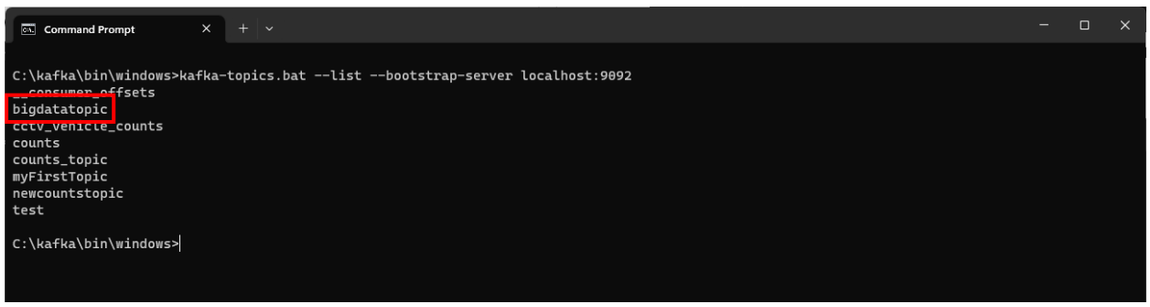
Open a new CLI and access the windows folder within the bin folder within the kafka folder. In this case, the command **cd C:\kafka\bin\windows** is executed.

Execute the command below to create a new topic. --topic bigdatatopic specifies that the topic name is bigdatatopic.

kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic bigdatatopic

Verify that the topic **bigdatatopic** is created by executing the command below. The topic created is shown in the list.

kafka-topics.bat --list --bootstrap-server localhost:9092



IV. Run Producers

A python script is used to run a producer instance for producing messages in the kafka server. The python script of producer1 instance with sensor_01 is shown below. The script encompasses the following

Libraries and Modules:

KafkaProducer: class from kafka used to create the producer instance

datetime: provides date and time utilities

time: used to access sleep method in the script to incorporate delay in sending messages

random: used for generating random numbers

uuid: used for creating unique IDs for each message sent

The producer instance is create using KafkaProducer() and assigned to kafka_producer_obj. The producer instance is assigned to send messages to the bigdatatopic within the loopback 127.0.0.1:9092. A message_list is created and a message variable is initialized to create and send the messages. A loop iterates 3600 times. In each iteration, a message is created containing the following: **timeuuid_id, lgu_code, sensor_id, date_saved, time_saved, total, car, bus, truck, jeepney, bike, tryke, others**. Thus, 3600 messages will be produced by the producer. The message is printed in the console. The message is then sent to the topic **bigdatatopic** using the kafka_producer_obj.send() where it can be consumed by the consumer. The time.sleep(1) code ensures that there is a 1 second delay between creating and sending messages. Below is an instance of a producer, **Kafka Producer1** with **sensor_01**.

```python
from kafka import KafkaProducer
from datetime import datetime
import time
from json import dumps
import random
import uuid

KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer1 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))

    message_list = []
    message = None
    for i in range(3600):
        i = i + 1
        date_today = datetime.now()
        message = {}
        print("Preparing message: " + str(i))

        car = random.randint( a: 0, b: 4)
        bus = random.randint( a: 0, b: 2)
        truck = random.randint( a: 0, b: 2)
        jeepney = random.randint( a: 0,  b: 2)
        bike = random.randint( a: 0,  b: 5)
        tryke = random.randint( a: 0,  b: 3)
        others = random.randint( a: 0,  b: 2)
```

```python
        total = car + bus + truck + jeepney + bike + tryke + others

        message["timeuuid_id"] = str(uuid.uuid1())
        message["lgu_code"] = '1200'
        message["sensor_id"] = 'sensor_01'
        message["date_saved"] = str(date_today.strftime('%m/%d/%Y'))
        message["time_saved"] = str(date_today.strftime("%X"))
        message["total"] = total
        message["car"] = car
        message["bus"] = bus
        message["truck"] = truck
        message["jeepney"] = jeepney
        message["bike"] = bike
        message["tryke"] = tryke
        message["others"] = others


        print("Message: ", message)

        kafka_producer_obj.send(KAFKA_TOPIC_NAME_CONS, message)
        time.sleep(1)

    print("Kafka Producer Application Completed. ")
```

Producer2 Instance with sensor_02

```python
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer2 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```python
        #send data
        message["timeuuid_id"] = str(uuid.uuid1())
        message["lgu_code"] = '1210'
        message["sensor_id"] = 'sensor_02'
```

Producer3 Instance with sensor_03

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer3 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
        #send data
        message["timeuuid_id"] = str(uuid.uuid1())
        message["lgu_code"] = '1220'
        message["sensor_id"] = 'sensor_03'
```

Producer4 Instance with sensor_04

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer4 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
        #send data
        message["timeuuid_id"] = str(uuid.uuid1())
        message["lgu_code"] = '1230'
        message["sensor_id"] = 'sensor_04'
```

Producer5 Instance with sensor_05

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer5 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
        #send data
        message["timeuuid_id"] = str(uuid.uuid1())
        message["lgu_code"] = '1240'
        message["sensor_id"] = 'sensor_05'
```

Producer6 Instance with sensor_06

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer6 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
        #send data
        message["timeuuid_id"] = str(uuid.uuid1())
        message["lgu_code"] = '1250'
        message["sensor_id"] = 'sensor_06'
```

Producer7 Instance with sensor_07

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer7 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
            #send data
            message["timeuuid_id"] = str(uuid.uuid1())
            message["lgu_code"] = '1260'
            message["sensor_id"] = 'sensor_07'
```

Producer8 Instance with sensor_08

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer8 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
            #send data
            message["timeuuid_id"] = str(uuid.uuid1())
            message["lgu_code"] = '1270'
            message["sensor_id"] = 'sensor_08'
```

Producer9 Instance with sensor_09

```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer9 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
            #send data
            message["timeuuid_id"] = str(uuid.uuid1())
            message["lgu_code"] = '1280'
            message["sensor_id"] = 'sensor_09'
```

Producer10 Instance with sensor_10

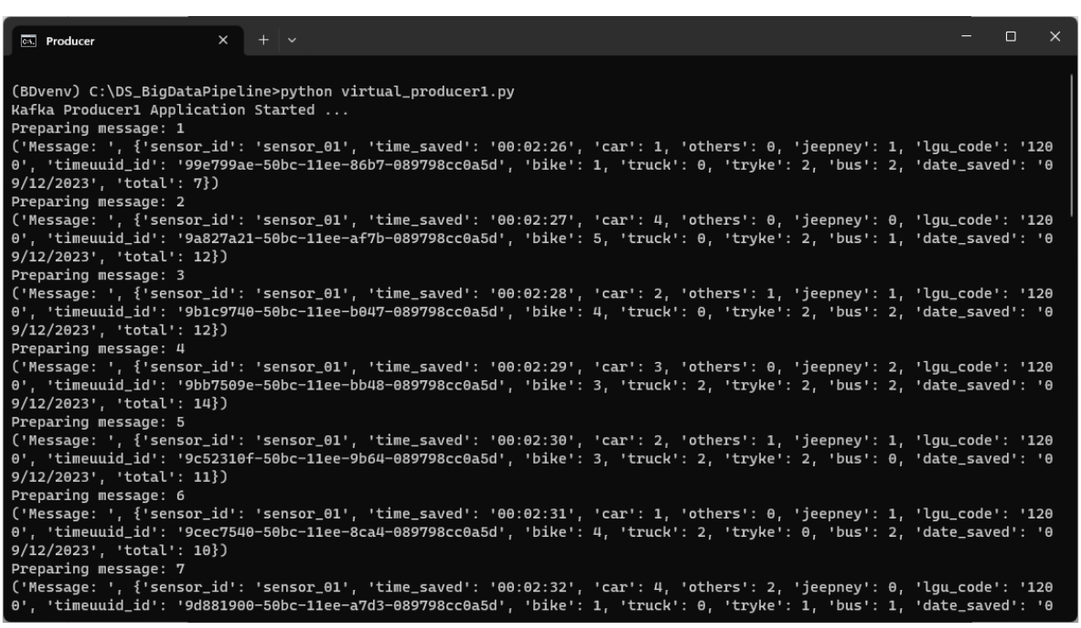
```
KAFKA_TOPIC_NAME_CONS = "bigdatatopic"   #topic
KAFKA_BOOTSTRAP_SERVERS_CONS = '127.0.0.1:9092'

if __name__ == "__main__":
    print("Kafka Producer10 Application Started ... ")
    kafka_producer_obj = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS_CONS,
                                       value_serializer=lambda x: dumps(x).encode('utf-8'))
```

```
            #send data
            message["timeuuid_id"] = str(uuid.uuid1())
            message["lgu_code"] = '1290'
            message["sensor_id"] = 'sensor_10'
```

To run a producer, access the python virtual environment where the python producer script is executed using the cd command. In this case, **C:\DS_BigDataPipeline\BDvenv** is the command executed. Activate the virtual environment. Use the command **Scripts\activate**

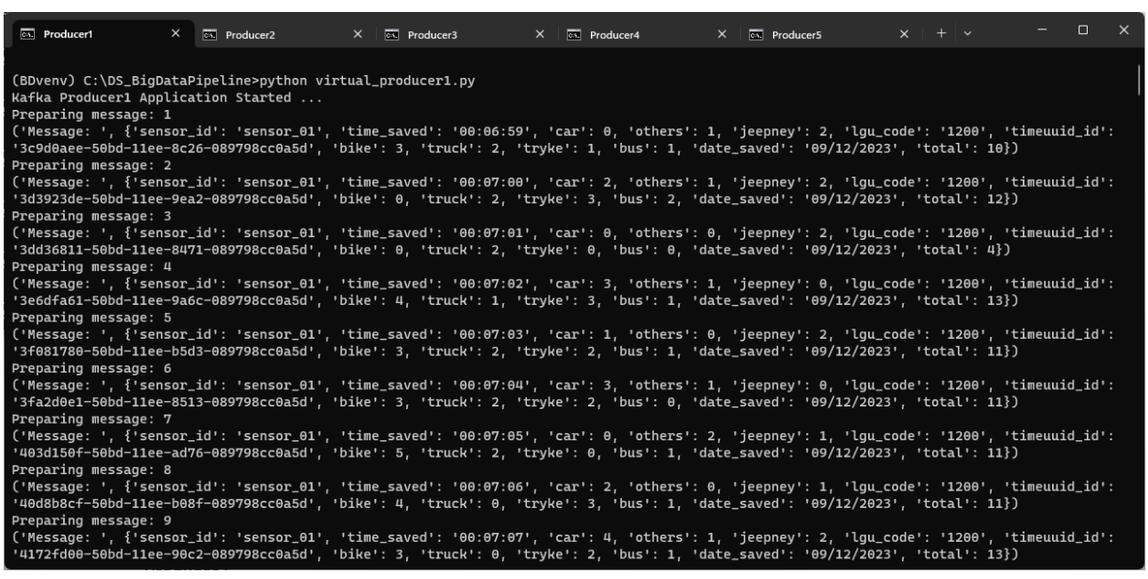
Access the file folder where the producer python script is located. In this case, the python script is in the DS_BigDataPipeline folder. The command **cd C:\DS_BigDataPipeline** is executed.

Run the producer python script to start producing and sending messages to the specified kafka topic. The command executed is **python virtual_producer1.py** where virtual_producer1.py is the python script of the first producer. The producer should start producing messages as shown.

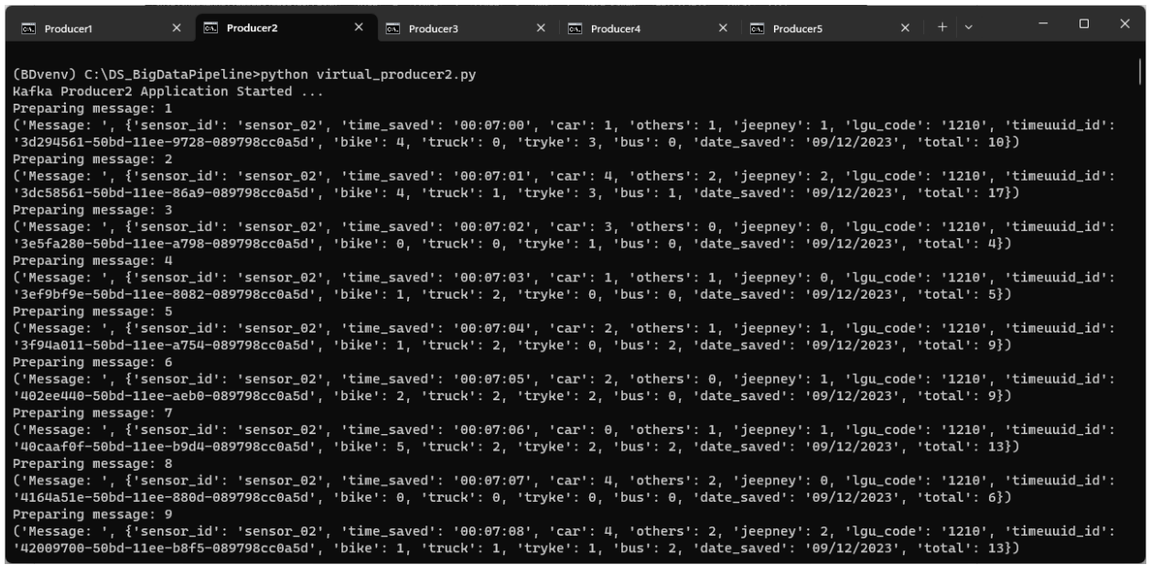


Simultaneously Running Producers

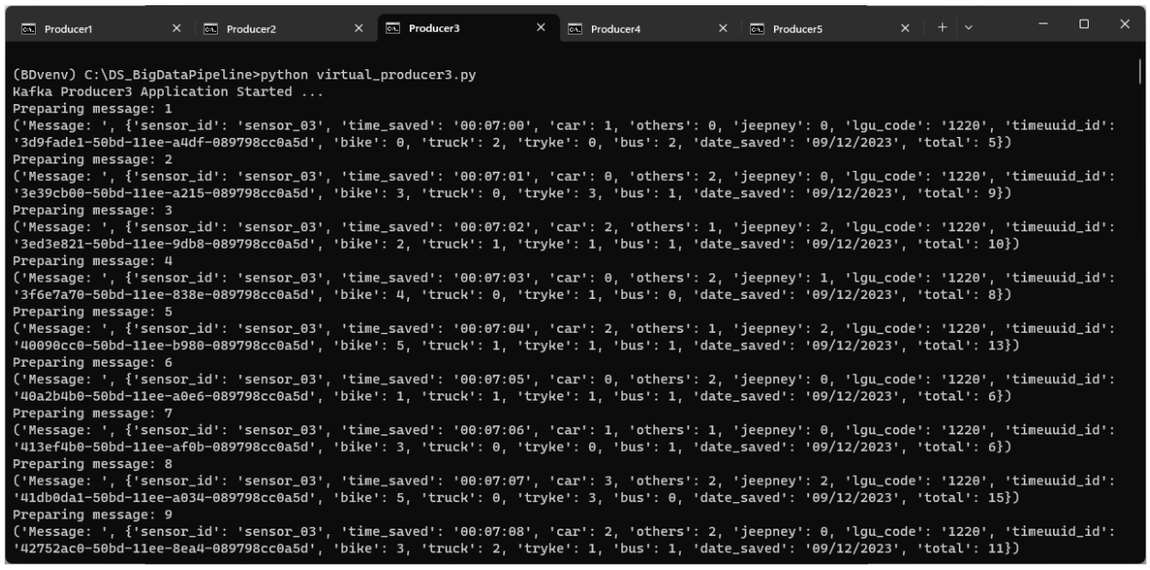
Producer1



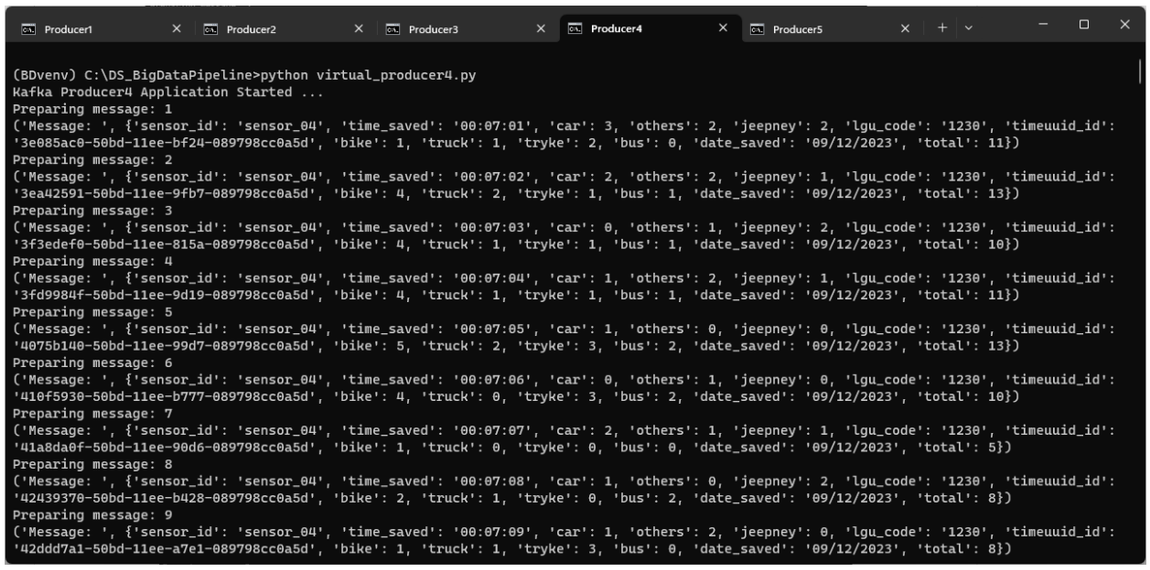
Producer2



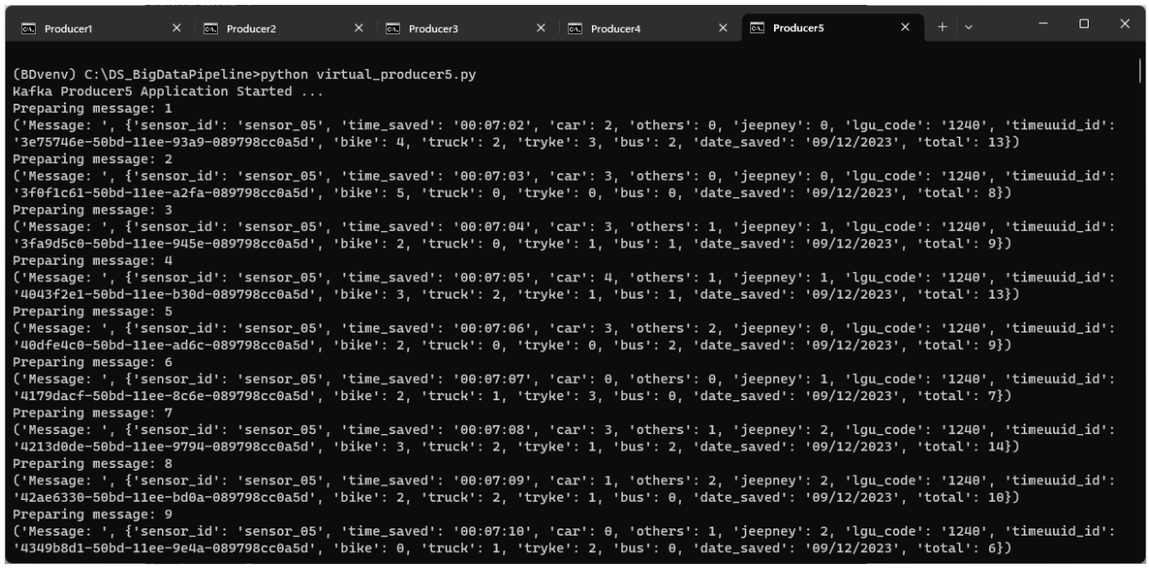
Producer3



Producer4

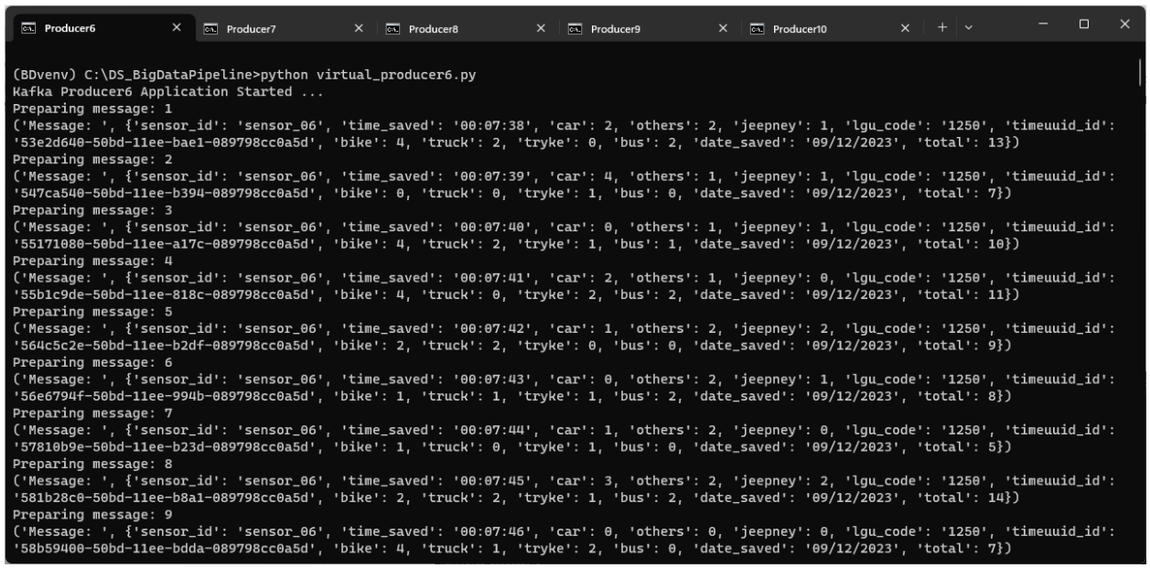


Producer5



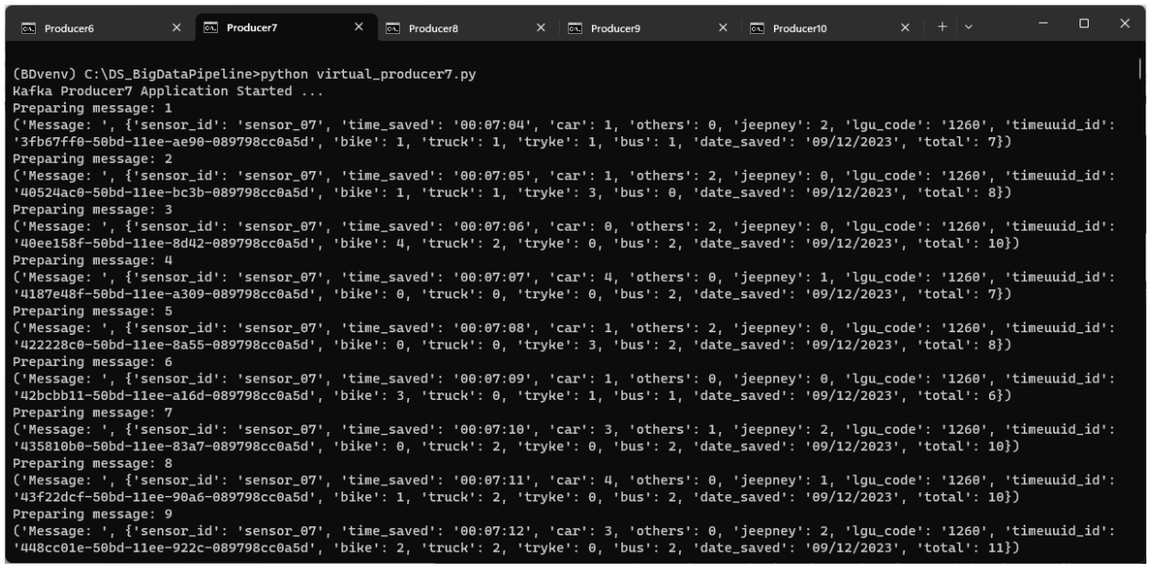

```
(BDvenv) C:\DS_BigDataPipeline>python virtual_producer5.py
Kafka Producer5 Application Started ...
Preparing message: 1
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:02', 'car': 2, 'others': 0, 'jeepney': 0, 'lgu_code': '1240', 'timeuuid_id':
'3e75746e-50bd-11ee-93a9-089798cc0a5d', 'bike': 4, 'truck': 2, 'tryke': 3, 'bus': 2, 'date_saved': '09/12/2023', 'total': 13})
Preparing message: 2
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:03', 'car': 3, 'others': 0, 'jeepney': 0, 'lgu_code': '1240', 'timeuuid_id':
'3f0f1c61-50bd-11ee-a2fa-089798cc0a5d', 'bike': 5, 'truck': 0, 'tryke': 0, 'bus': 0, 'date_saved': '09/12/2023', 'total': 8})
Preparing message: 3
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:04', 'car': 3, 'others': 1, 'jeepney': 1, 'lgu_code': '1240', 'timeuuid_id':
'3fa9d5c0-50bd-11ee-945e-089798cc0a5d', 'bike': 2, 'truck': 0, 'tryke': 1, 'bus': 1, 'date_saved': '09/12/2023', 'total': 9})
Preparing message: 4
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:05', 'car': 4, 'others': 1, 'jeepney': 1, 'lgu_code': '1240', 'timeuuid_id':
'4043f2e1-50bd-11ee-b30d-089798cc0a5d', 'bike': 3, 'truck': 2, 'tryke': 1, 'bus': 1, 'date_saved': '09/12/2023', 'total': 13})
Preparing message: 5
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:06', 'car': 3, 'others': 2, 'jeepney': 0, 'lgu_code': '1240', 'timeuuid_id':
'40dfe4c0-50bd-11ee-ad6c-089798cc0a5d', 'bike': 2, 'truck': 0, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 9})
Preparing message: 6
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:07', 'car': 0, 'others': 0, 'jeepney': 1, 'lgu_code': '1240', 'timeuuid_id':
'4179dacf-50bd-11ee-8c6e-089798cc0a5d', 'bike': 2, 'truck': 1, 'tryke': 3, 'bus': 0, 'date_saved': '09/12/2023', 'total': 7})
Preparing message: 7
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:08', 'car': 3, 'others': 1, 'jeepney': 2, 'lgu_code': '1240', 'timeuuid_id':
'4213d0de-50bd-11ee-9794-089798cc0a5d', 'bike': 3, 'truck': 2, 'tryke': 1, 'bus': 2, 'date_saved': '09/12/2023', 'total': 14})
Preparing message: 8
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:09', 'car': 1, 'others': 2, 'jeepney': 2, 'lgu_code': '1240', 'timeuuid_id':
'42ae6330-50bd-11ee-bd0a-089798cc0a5d', 'bike': 2, 'truck': 2, 'tryke': 1, 'bus': 0, 'date_saved': '09/12/2023', 'total': 10})
Preparing message: 9
('Message: ', {'sensor_id': 'sensor_05', 'time_saved': '00:07:10', 'car': 0, 'others': 1, 'jeepney': 2, 'lgu_code': '1240', 'timeuuid_id':
'4349b8d1-50bd-11ee-9e4a-089798cc0a5d', 'bike': 0, 'truck': 1, 'tryke': 2, 'bus': 0, 'date_saved': '09/12/2023', 'total': 6})
```

Producer6



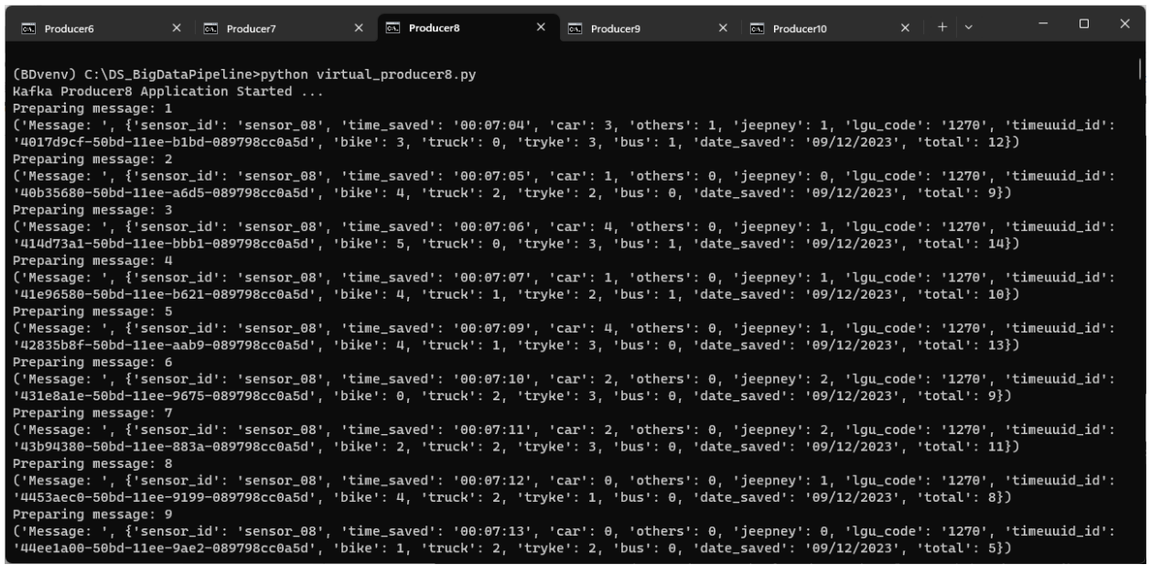

```
(BDvenv) C:\DS_BigDataPipeline>python virtual_producer6.py
Kafka Producer6 Application Started ...
Preparing message: 1
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:38', 'car': 2, 'others': 2, 'jeepney': 1, 'lgu_code': '1250', 'timeuuid_id':
'53e2d640-50bd-11ee-bae1-089798cc0a5d', 'bike': 4, 'truck': 2, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 13})
Preparing message: 2
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:39', 'car': 4, 'others': 1, 'jeepney': 1, 'lgu_code': '1250', 'timeuuid_id':
'547ca540-50bd-11ee-b394-089798cc0a5d', 'bike': 0, 'truck': 0, 'tryke': 1, 'bus': 0, 'date_saved': '09/12/2023', 'total': 7})
Preparing message: 3
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:40', 'car': 0, 'others': 1, 'jeepney': 1, 'lgu_code': '1250', 'timeuuid_id':
'55171080-50bd-11ee-a17c-089798cc0a5d', 'bike': 4, 'truck': 2, 'tryke': 1, 'bus': 1, 'date_saved': '09/12/2023', 'total': 10})
Preparing message: 4
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:41', 'car': 2, 'others': 1, 'jeepney': 0, 'lgu_code': '1250', 'timeuuid_id':
'55b1c9de-50bd-11ee-818c-089798cc0a5d', 'bike': 4, 'truck': 0, 'tryke': 2, 'bus': 2, 'date_saved': '09/12/2023', 'total': 11})
Preparing message: 5
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:42', 'car': 1, 'others': 2, 'jeepney': 2, 'lgu_code': '1250', 'timeuuid_id':
'564c5c2e-50bd-11ee-b2df-089798cc0a5d', 'bike': 2, 'truck': 2, 'tryke': 0, 'bus': 0, 'date_saved': '09/12/2023', 'total': 9})
Preparing message: 6
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:43', 'car': 0, 'others': 2, 'jeepney': 1, 'lgu_code': '1250', 'timeuuid_id':
'56e6794f-50bd-11ee-994b-089798cc0a5d', 'bike': 1, 'truck': 1, 'tryke': 1, 'bus': 2, 'date_saved': '09/12/2023', 'total': 8})
Preparing message: 7
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:44', 'car': 1, 'others': 2, 'jeepney': 0, 'lgu_code': '1250', 'timeuuid_id':
'57810b9e-50bd-11ee-b23d-089798cc0a5d', 'bike': 1, 'truck': 0, 'tryke': 1, 'bus': 0, 'date_saved': '09/12/2023', 'total': 5})
Preparing message: 8
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:45', 'car': 3, 'others': 2, 'jeepney': 2, 'lgu_code': '1250', 'timeuuid_id':
'581b28c0-50bd-11ee-b8a1-089798cc0a5d', 'bike': 2, 'truck': 2, 'tryke': 1, 'bus': 2, 'date_saved': '09/12/2023', 'total': 14})
Preparing message: 9
('Message: ', {'sensor_id': 'sensor_06', 'time_saved': '00:07:46', 'car': 0, 'others': 0, 'jeepney': 0, 'lgu_code': '1250', 'timeuuid_id':
'58b59400-50bd-11ee-bdda-089798cc0a5d', 'bike': 4, 'truck': 1, 'tryke': 2, 'bus': 0, 'date_saved': '09/12/2023', 'total': 7})
```

Producer7

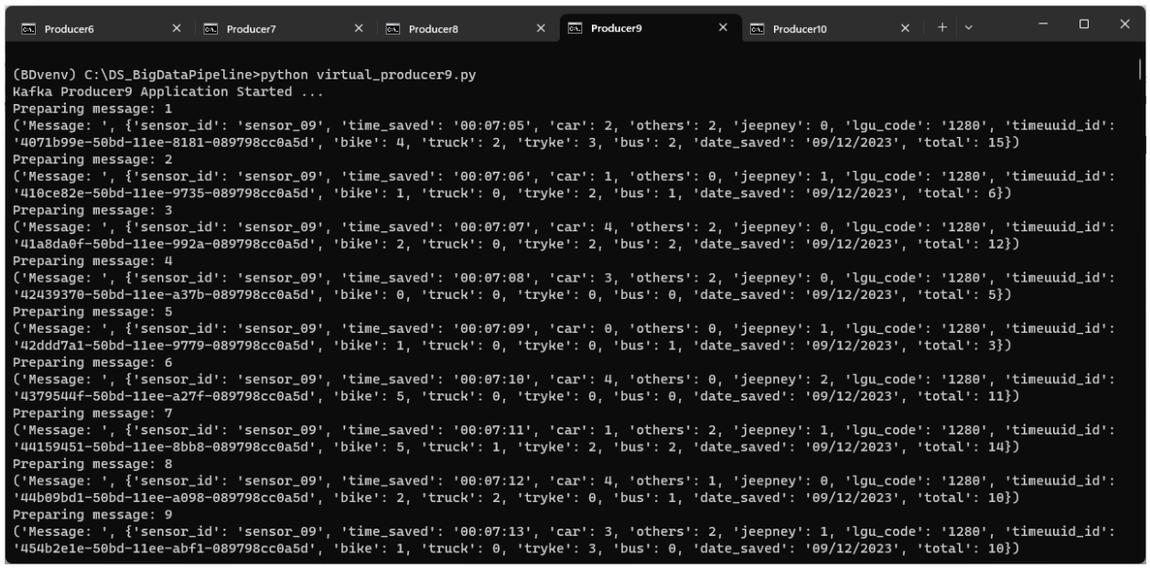



```
(BDvenv) C:\DS_BigDataPipeline>python virtual_producer7.py
Kafka Producer7 Application Started ...
Preparing message: 1
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:04', 'car': 1, 'others': 0, 'jeepney': 2, 'lgu_code': '1260', 'timeuuid_id':
'3fb67ff0-50bd-11ee-ae90-089798cc0a5d', 'bike': 1, 'truck': 1, 'tryke': 1, 'bus': 1, 'date_saved': '09/12/2023', 'total': 7})
Preparing message: 2
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:05', 'car': 1, 'others': 2, 'jeepney': 0, 'lgu_code': '1260', 'timeuuid_id':
'40524ac0-50bd-11ee-bc3b-089798cc0a5d', 'bike': 1, 'truck': 1, 'tryke': 3, 'bus': 0, 'date_saved': '09/12/2023', 'total': 8})
Preparing message: 3
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:06', 'car': 0, 'others': 2, 'jeepney': 0, 'lgu_code': '1260', 'timeuuid_id':
'40ee158f-50bd-11ee-8d42-089798cc0a5d', 'bike': 4, 'truck': 2, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 10})
Preparing message: 4
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:07', 'car': 4, 'others': 0, 'jeepney': 1, 'lgu_code': '1260', 'timeuuid_id':
'4187e48f-50bd-11ee-a309-089798cc0a5d', 'bike': 0, 'truck': 0, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 7})
Preparing message: 5
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:08', 'car': 1, 'others': 2, 'jeepney': 0, 'lgu_code': '1260', 'timeuuid_id':
'422228c0-50bd-11ee-8a55-089798cc0a5d', 'bike': 0, 'truck': 0, 'tryke': 3, 'bus': 2, 'date_saved': '09/12/2023', 'total': 8})
Preparing message: 6
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:09', 'car': 1, 'others': 0, 'jeepney': 0, 'lgu_code': '1260', 'timeuuid_id':
'42bcbb11-50bd-11ee-a16d-089798cc0a5d', 'bike': 3, 'truck': 0, 'tryke': 1, 'bus': 1, 'date_saved': '09/12/2023', 'total': 6})
Preparing message: 7
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:10', 'car': 3, 'others': 1, 'jeepney': 2, 'lgu_code': '1260', 'timeuuid_id':
'435810b0-50bd-11ee-83a7-089798cc0a5d', 'bike': 0, 'truck': 2, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 10})
Preparing message: 8
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:11', 'car': 4, 'others': 0, 'jeepney': 1, 'lgu_code': '1260', 'timeuuid_id':
'43f22dcf-50bd-11ee-90a6-089798cc0a5d', 'bike': 1, 'truck': 2, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 10})
Preparing message: 9
('Message: ', {'sensor_id': 'sensor_07', 'time_saved': '00:07:12', 'car': 3, 'others': 0, 'jeepney': 2, 'lgu_code': '1260', 'timeuuid_id':
'448cc01e-50bd-11ee-922c-089798cc0a5d', 'bike': 2, 'truck': 2, 'tryke': 0, 'bus': 2, 'date_saved': '09/12/2023', 'total': 11})
```

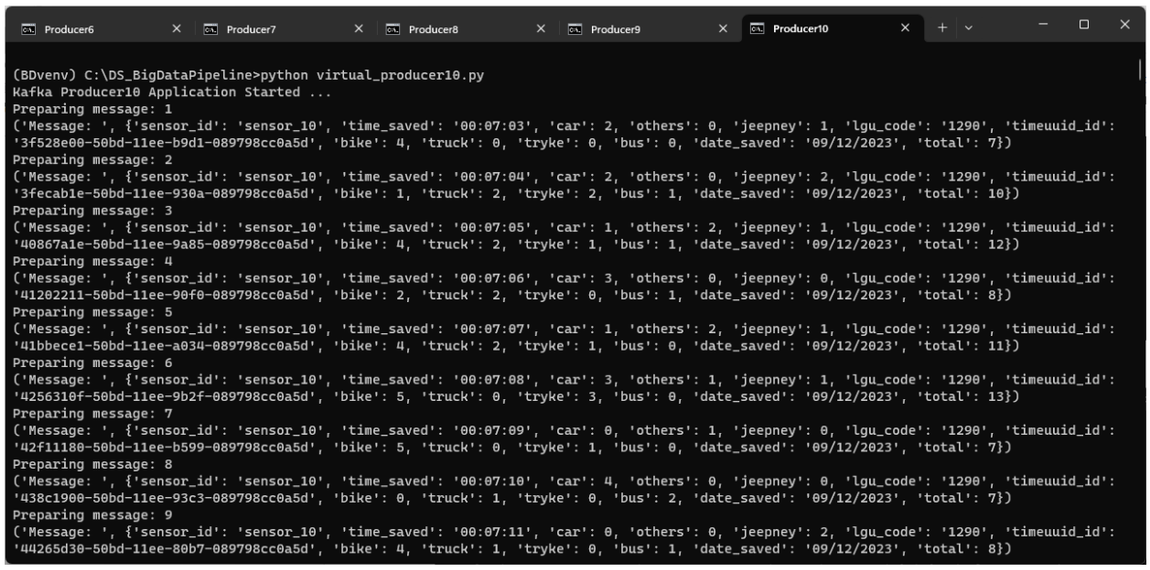
Producer8



Producer9



Producer10



V. Run Consumer

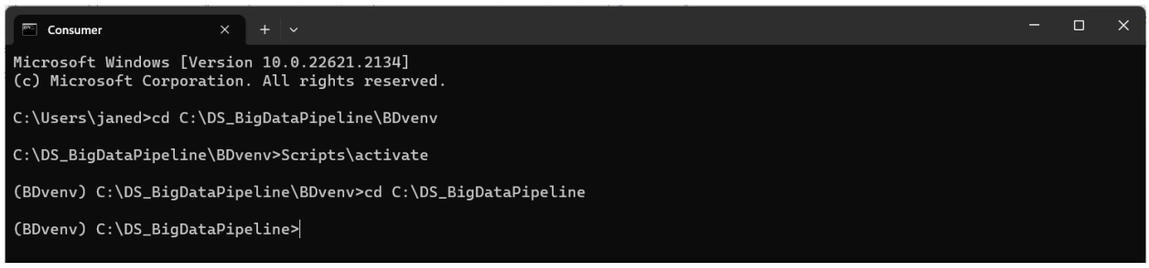
The python script for running the consumer is presented below. KafkaConsumer class is imported from kafka library to create the kafka consumer. KafkaConsumer() creates a new consumer. The consumer subscribes to the topic **bigdatatopic** as specified and will connect to the kafka broker using the loopback address **127.0.0.1:9092**. The consumer will begin reading the latest available

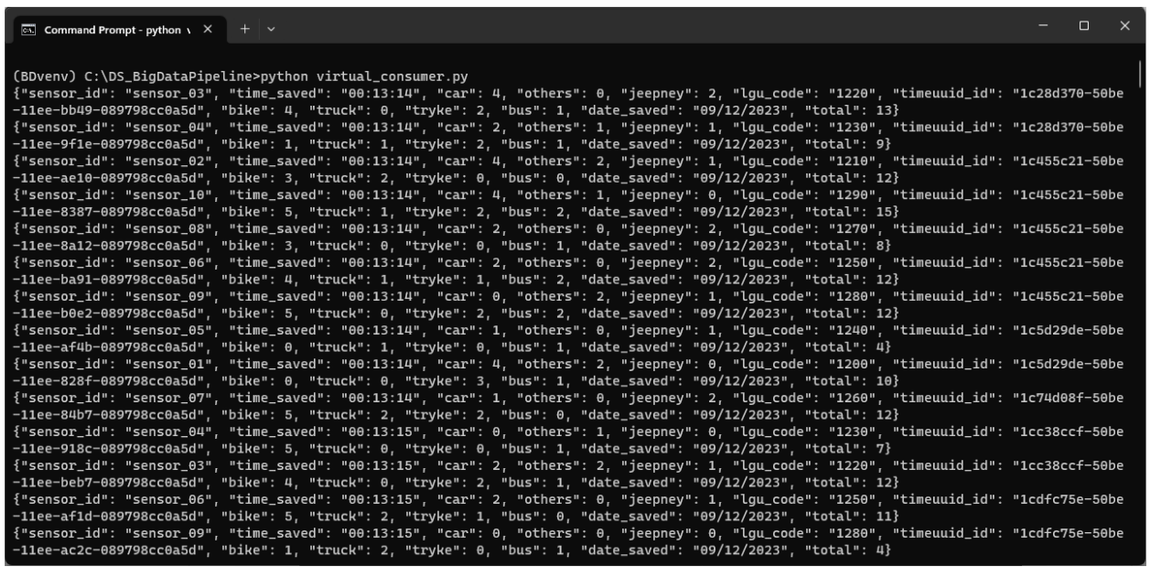
messages and will periodically commit its current position to prevent re-reading messages. A for loop iterates over all messages consumed by the consumer and will print the value of the messages in the prompt.

```python
# -*- coding: utf-8 -*-

from kafka import KafkaConsumer

consumer = KafkaConsumer(
    *topics: 'bigdatatopic',
    bootstrap_servers = ['127.0.0.1:9092'],
    auto_offset_reset = 'latest',
    enable_auto_commit = True
    )

for message in consumer:
    message = message.value
    print(message)
```

Run the consumer python script. The command used is **python virtual_consumer.py** where virtual_consumer.py is the file name of the script.

The output is presented below where all messages produced and sent by producers with from sensor_01 to sensor_10 are consumed by the producer in real-time.





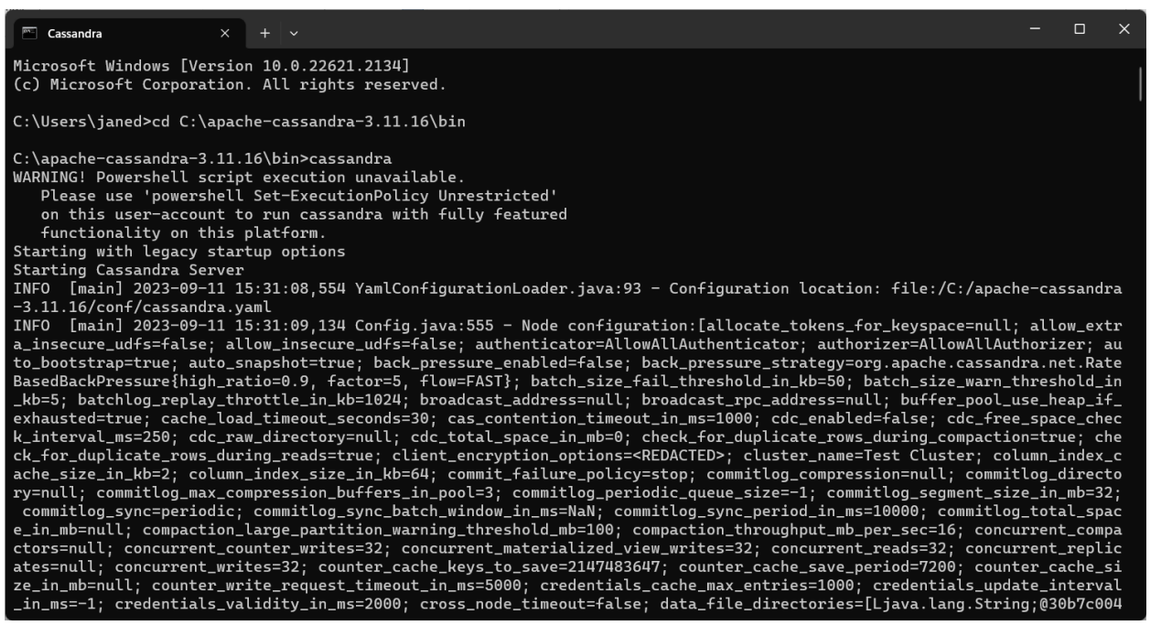
VI. Creating a Cassandra Keyspace and Table

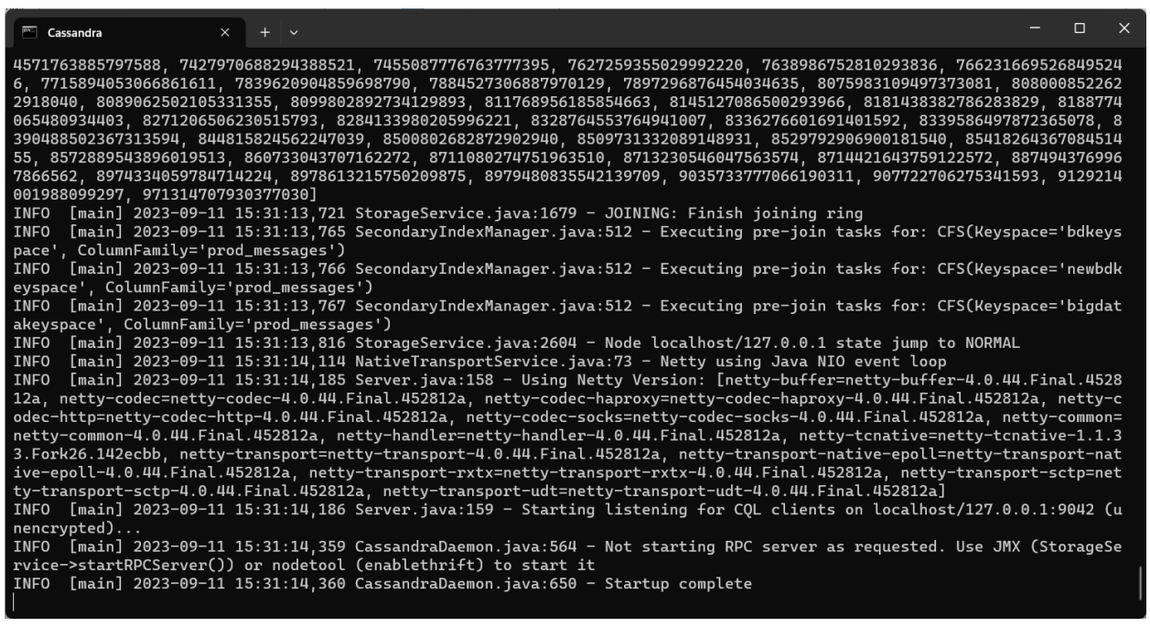
A database is required to store the messages produced by the producer and consumed by the consumer. In this case, a Cassandra keyspace and table will be created.

Activate Cassandra Server

To run the Cassandra server, open a new CLI and access the bin folder within the Cassandra folder. In this case, the command used is **cd C:\apache-cassandra-3.11.16\bin**

Run the Cassandra server using the command **Cassandra**. The output is presented below.





To verify that Cassandra is running, open a new CLI and access the bin folder. Enter the command **cqlsh**. The output is shown below.

Create Cassandra Keyspace

The python script presented below is used to create a Cassandra Keyspace which will be used to hold the table where messages will be saved. Cluster is imported from Cassandra.cluster module and is used to specify the address and port of the Cassandra cluster. The cluster connects

to a session using connect(). A keyspace query is created that creates a Cassandra keyspace called **dskeyspacebd** with configurations of SimpleStrategy class and a replication factor of 1. The query is executed using session.execute() and the session and cluster are both shutdown.

```
from cassandra.cluster import Cluster

cluster = Cluster( contact_points: ['127.0.0.1'], port=9042)
session = cluster.connect()

create_keyspace_query = """
    CREATE KEYSPACE IF NOT EXISTS dskeyspacebd
    WITH replication ={
        'class': 'SimpleStrategy',
        'replication_factor': 1
    };
"""
session.execute(create_keyspace_query)
session.shutdown()
cluster.shutdown()
```

To verify that the keyspace is created, the command **DESCRIBE dskeyspacebd;** is used in cqlsh. The output is shown below.

Create Cassandra Table

The python script shown below creates a table named **prod_messages** in the keyspace **dskeyspacebd**. Cluster is imported from Cassandra.cluster module and is used to specify the address and port of the Cassandra cluster. A session is created that connects to the cluster specified. A query is created that creates a table in **dskeyspacebd** named **prod_messages** and the column names and datatypes or specified. The column names are the labels of the messages produced by the producer. session.execute() executes the query and creates the table. If the table name is already exists in the keyspace, the user is notified that the table already exists, otherwise the user is notified that the table is created. The session and cluster are both shutdown afterwards.

```
from cassandra.cluster import Cluster

cluster = Cluster( contact_points: ['127.0.0.1'], port=9042)
session = cluster.connect()

create_table_query = """
    CREATE TABLE IF NOT EXISTS dskeyspacebd.prod_messages(
        sensor_id TEXT,
        time_saved TEXT,
        car INT,
        others INT,
        jeepney INT,
        lgu_code TEXT,
        timeuuid_id TEXT PRIMARY KEY,
        bike INT,
        truck INT,
        tryke INT,
        bus INT,
        date_saved TEXT,
        total BIGINT
    )
"""

name = 'prod_messages'

existing_tables = session.execute("SELECT table_name FROM system_schema.tables WHERE keyspace_name = 'dskeyspacebd'")
table_exists = False

for row in existing_tables:
    if row.table_name == name:
        table_exists = True
        break

if table_exists:
    print("Table {} already exists".format(name))
else:
    session.execute(create_table_query)
    print("Table {} created".format(name))

session.shutdown()
cluster.shutdown()
```

To verify that the table is created, the following commands are executed in cqlsh:

USE dskeyspacebd;

SELECT * FROM prod_messages;

The output is shown below.

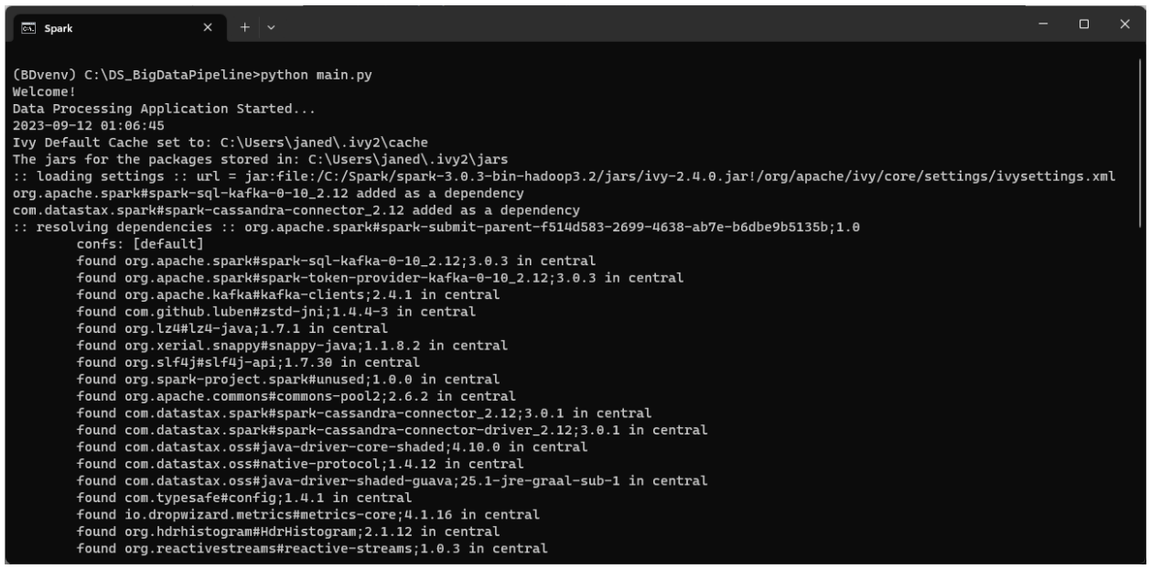
VII. Fetch and Store Messages to Cassandra Keyspace

The python script below utilizes Pyspark, Kafka, and Cassandra connectors to fetch the latest batch of data sent by the producer and store the messages in the dskeyspacebd keyspace and within the prod_messages table created earlier. Utilities within pyspark.sql, pyspark.sql.functions, pyspark.sql.types and time libraries are used. The kafka configurations are initialized which encompass the target topic cluster and the broker address and port. The Cassandra configurations are also initialized which encompass the address, port number, target keyspace and target table within the keyspace. A spark application is created and the configurations of the session are specified. The spark connection to Cassandra is specified in the configuration. The spark.jars used for the application are also included in the configurations. These encompass kafka-clients, spark-streaming-kafka, spark-cassandra-connector, as well as the spark.jars.packages which include org.apache.spark:spark-sql-kafka and com.datastax.spark:spark-cassandra-connector which will all be used to fetch messages from kafka and store them in Cassandra keyspace. The schema is prepared which will be used as the framework for organizing the messages collected which will then be sent to Cassandra. The StructFields are initialized with each message attribute. Spark is used to read data from a kafka source and the target topic is specified for subscription. The messages collected are converted to string and initially stored in kafka_data DataFrame. Afterwards, the messages are parsed into JSON data fitted into the schema. Data is then written to Cassandra using writeStream. Batch, format, target table and keyspace, and mode are specified. The writing process is then carried out using .save(). Upon a keyboard interrupt, i.e. ctrl + c, the query will terminate.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.streaming import DataStreamWriter
import time

kafka_topic_name = "bigdatatopic"
kafka_bootstrap_servers = '127.0.0.1:9092'

cassandra_host_name = '127.0.0.1'
cassandra_port_num = '9042'
cassandra_keyspace_name = 'dskeyspacebd'
cassandra_table_name = 'prod_messages'

if __name__ == "__main__":
    print("Welcome!")
    print("Data Processing Application Started...")
    print(time.strftime('''%Y-%m-%d %H:%M:%S'''))

    spark = SparkSession \
        .builder \
        .appName("Pyspark Streaming with Kafka and Cassandra") \
        .master("local[*]") \
        .config("spark.cassandra.connection.host", cassandra_host_name) \
        .config("spark.cassandra.connection.port", cassandra_port_num) \
        .config("spark.jars", "C:\\DS_BigDataPipeline\\lib\\kafka-clients-3.5.1.jar,"
                              "C:\\DS_BigDataPipeline\\lib\\spark-streaming-kafka-0-10_2.12-3.0.3.jar,"
                              "C:\\DS_BigDataPipeline\\lib\\spark-cassandra-connector_2.12-3.0.1.jar") \
        .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.3,"
                                       "com.datastax.spark:spark-cassandra-connector_2.12:3.0.1") \
        .getOrCreate()
    spark.sparkContext.setLogLevel("ERROR")
```

```python

    schema = StructType([
        StructField( name: "sensor_id", StringType(), True),
        StructField( name: "time_saved", StringType(), True),
        StructField( name: "car", IntegerType(), True),
        StructField( name: "others", IntegerType(), True),
        StructField( name: "jeepney", IntegerType(), True),
        StructField( name: "lgu_code", StringType(), True),
        StructField( name: "timeuuid_id", StringType(), True),
        StructField( name: "bike", IntegerType(), True),
        StructField( name: "truck", IntegerType(), True),
        StructField( name: "tryke", IntegerType(), True),
        StructField( name: "bus", IntegerType(), True),
        StructField( name: "date_saved", StringType(), True),
        StructField( name: "total", LongType(), True)
    ])

    kafka_stream = spark \
        .readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
        .option("subscribe", kafka_topic_name) \
        .load()

    kafka_data = kafka_stream.selectExpr("CAST(value AS STRING)")
    parsed_data = kafka_data.select(from_json( col: "value", schema).alias("data")).select("data.*")

    cassandra_stream = parsed_data.writeStream \
        .foreachBatch(lambda df, epoch_id: df.write \
                      .format("org.apache.spark.sql.cassandra") \
                      .options(table=cassandra_table_name, keyspace=cassandra_keyspace_name) \
                      .mode("append") \
                      .save())

    query = cassandra_stream.start()

    try:
        query.awaitTermination()
    except KeyboardInterrupt:
        print("Terminating...")
        query.stop()
```

Execute the spark application by running the python script in the CLI. In the virtual environment where the script is stored, the command **python main.py** is executed where main.py is the file name of the spark application python script. The output is shown below.



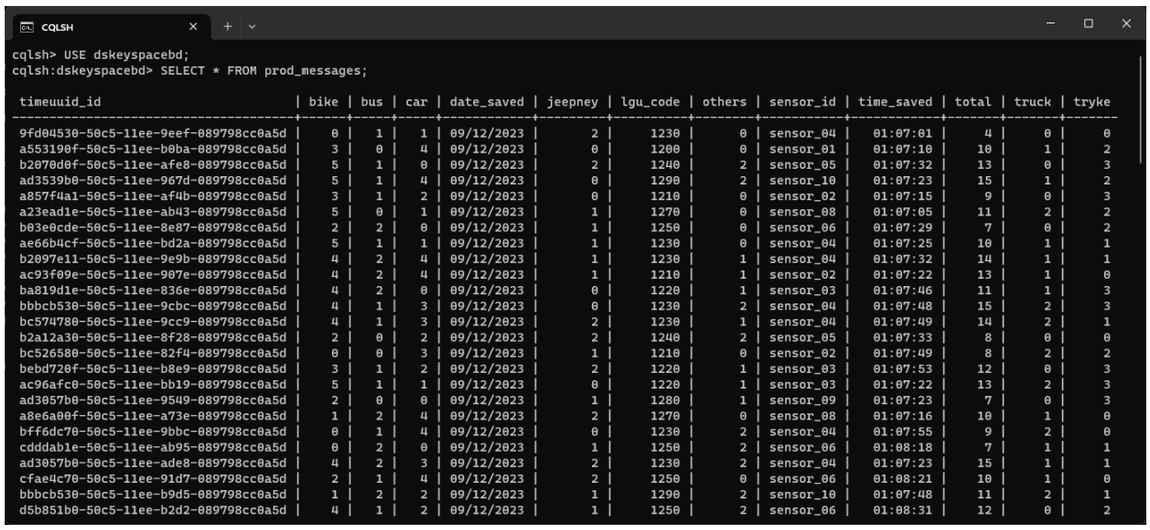
VIII. Verify if Data is Saved in Cassandra

To verify that the spark application is working and the messages sent by the producer are fetched and saved in the Cassandra keyspace table, **cqlsh** is run and the following commands are executed:

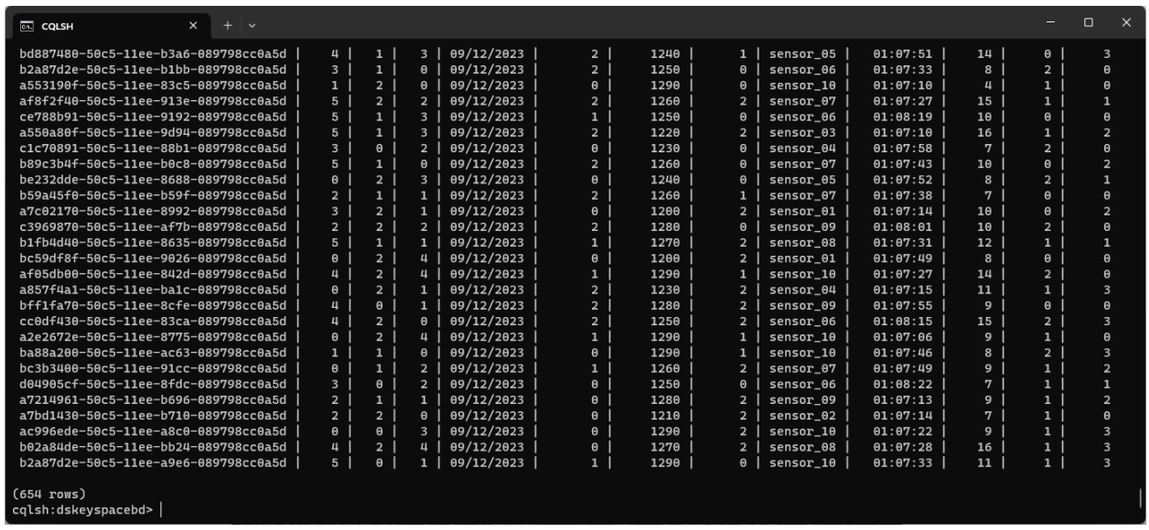
USE dskeyspacebd;

SELECT * FROM prod_messages;

The output is shown below. All message values are successfully stored in the proper columns in the table.



```
CQLSH

cqlsh> USE dskeyspacebd;
cqlsh:dskeyspacebd> SELECT * FROM prod_messages;

 timeuuid_id                          | bike | bus | car | date_saved | jeepney | lgu_code | others | sensor_id | time_saved | total | truck | tryke
--------------------------------------+------+-----+-----+------------+---------+----------+--------+-----------+------------+-------+-------+-------
 9fd04530-50c5-11ee-9eef-089798cc0a5d |    0 |   1 |   1 | 09/12/2023 |       2 |     1230 |      0 | sensor_04 |   01:07:01 |     4 |     0 |     0
 a553190f-50c5-11ee-b0ba-089798cc0a5d |    3 |   0 |   4 | 09/12/2023 |       0 |     1200 |      0 | sensor_01 |   01:07:10 |    10 |     1 |     2
 b2070d0f-50c5-11ee-afe8-089798cc0a5d |    5 |   1 |   0 | 09/12/2023 |       2 |     1240 |      2 | sensor_05 |   01:07:32 |    13 |     0 |     3
 ad3539b0-50c5-11ee-967d-089798cc0a5d |    5 |   1 |   4 | 09/12/2023 |       0 |     1290 |      2 | sensor_10 |   01:07:23 |    15 |     1 |     2
 a857f4a1-50c5-11ee-af4b-089798cc0a5d |    3 |   1 |   2 | 09/12/2023 |       0 |     1210 |      0 | sensor_02 |   01:07:15 |     9 |     0 |     3
 a23ead1e-50c5-11ee-ab43-089798cc0a5d |    5 |   0 |   1 | 09/12/2023 |       1 |     1270 |      0 | sensor_08 |   01:07:05 |    11 |     2 |     2
 b03e0cde-50c5-11ee-8e87-089798cc0a5d |    2 |   2 |   0 | 09/12/2023 |       1 |     1250 |      0 | sensor_06 |   01:07:29 |     7 |     0 |     2
 ae66b4cf-50c5-11ee-bd2a-089798cc0a5d |    5 |   1 |   1 | 09/12/2023 |       1 |     1230 |      0 | sensor_04 |   01:07:25 |    10 |     1 |     1
 b2097e11-50c5-11ee-9e9b-089798cc0a5d |    4 |   2 |   4 | 09/12/2023 |       1 |     1230 |      1 | sensor_04 |   01:07:32 |    14 |     1 |     1
 ac93f09e-50c5-11ee-907e-089798cc0a5d |    4 |   2 |   4 | 09/12/2023 |       1 |     1210 |      1 | sensor_02 |   01:07:22 |    13 |     1 |     0
 ba819d1e-50c5-11ee-836e-089798cc0a5d |    4 |   2 |   0 | 09/12/2023 |       0 |     1220 |      1 | sensor_03 |   01:07:46 |    11 |     1 |     3
 bbbcb530-50c5-11ee-9cbc-089798cc0a5d |    4 |   1 |   3 | 09/12/2023 |       0 |     1230 |      2 | sensor_04 |   01:07:48 |    15 |     2 |     3
 bc574780-50c5-11ee-9cc9-089798cc0a5d |    4 |   1 |   3 | 09/12/2023 |       2 |     1230 |      1 | sensor_04 |   01:07:49 |    14 |     2 |     1
 b2a12a30-50c5-11ee-8f28-089798cc0a5d |    2 |   0 |   2 | 09/12/2023 |       2 |     1240 |      2 | sensor_05 |   01:07:33 |     8 |     0 |     0
 bc526580-50c5-11ee-82f4-089798cc0a5d |    0 |   0 |   3 | 09/12/2023 |       1 |     1210 |      0 | sensor_02 |   01:07:49 |     8 |     2 |     2
 bebd720f-50c5-11ee-b8e9-089798cc0a5d |    3 |   1 |   2 | 09/12/2023 |       2 |     1220 |      1 | sensor_03 |   01:07:53 |    12 |     0 |     3
 ac96afc0-50c5-11ee-bb19-089798cc0a5d |    5 |   1 |   1 | 09/12/2023 |       0 |     1220 |      1 | sensor_03 |   01:07:22 |    13 |     2 |     3
 ad3057b0-50c5-11ee-9549-089798cc0a5d |    2 |   0 |   0 | 09/12/2023 |       1 |     1280 |      1 | sensor_09 |   01:07:23 |     7 |     0 |     3
 a8e6a00f-50c5-11ee-a73e-089798cc0a5d |    1 |   2 |   4 | 09/12/2023 |       2 |     1270 |      0 | sensor_08 |   01:07:16 |    10 |     1 |     0
 bff6dc70-50c5-11ee-9bbc-089798cc0a5d |    0 |   1 |   4 | 09/12/2023 |       0 |     1230 |      2 | sensor_04 |   01:07:55 |     9 |     2 |     0
 cdddab1e-50c5-11ee-ab95-089798cc0a5d |    0 |   2 |   0 | 09/12/2023 |       1 |     1250 |      2 | sensor_06 |   01:08:18 |     7 |     1 |     1
 ad3057b0-50c5-11ee-ade8-089798cc0a5d |    4 |   2 |   3 | 09/12/2023 |       2 |     1230 |      2 | sensor_04 |   01:07:23 |    15 |     1 |     1
 cfae4c70-50c5-11ee-91d7-089798cc0a5d |    2 |   1 |   4 | 09/12/2023 |       2 |     1250 |      0 | sensor_06 |   01:08:21 |    10 |     1 |     0
 bbbcb530-50c5-11ee-b9d5-089798cc0a5d |    1 |   2 |   2 | 09/12/2023 |       1 |     1290 |      2 | sensor_10 |   01:07:48 |    11 |     2 |     1
 d5b851b0-50c5-11ee-b2d2-089798cc0a5d |    4 |   1 |   2 | 09/12/2023 |       1 |     1250 |      2 | sensor_06 |   01:08:31 |    12 |     0 |     2
```



```
CQLSH

 bd887480-50c5-11ee-b3a6-089798cc0a5d |    4 |   1 |   3 | 09/12/2023 |       2 |     1240 |      1 | sensor_05 |   01:07:51 |    14 |     0 |     3
 b2a87d2e-50c5-11ee-b1bb-089798cc0a5d |    3 |   1 |   0 | 09/12/2023 |       2 |     1250 |      0 | sensor_06 |   01:07:33 |     8 |     2 |     0
 a553190f-50c5-11ee-83c5-089798cc0a5d |    1 |   2 |   0 | 09/12/2023 |       0 |     1290 |      0 | sensor_10 |   01:07:10 |     4 |     1 |     0
 af8f2f40-50c5-11ee-913e-089798cc0a5d |    5 |   2 |   2 | 09/12/2023 |       2 |     1260 |      2 | sensor_07 |   01:07:27 |    15 |     1 |     1
 ce788b91-50c5-11ee-9192-089798cc0a5d |    5 |   1 |   3 | 09/12/2023 |       1 |     1250 |      0 | sensor_06 |   01:08:19 |    10 |     0 |     0
 a550a80f-50c5-11ee-9d94-089798cc0a5d |    5 |   1 |   3 | 09/12/2023 |       2 |     1220 |      2 | sensor_03 |   01:07:10 |    16 |     1 |     2
 c1c70891-50c5-11ee-88b1-089798cc0a5d |    3 |   0 |   2 | 09/12/2023 |       0 |     1230 |      0 | sensor_04 |   01:07:58 |     7 |     2 |     0
 b89c3b4f-50c5-11ee-b0c8-089798cc0a5d |    5 |   1 |   0 | 09/12/2023 |       2 |     1260 |      0 | sensor_07 |   01:07:43 |    10 |     0 |     2
 be232dde-50c5-11ee-8688-089798cc0a5d |    0 |   2 |   3 | 09/12/2023 |       0 |     1240 |      0 | sensor_05 |   01:07:52 |     8 |     2 |     1
 b59a45f0-50c5-11ee-b59f-089798cc0a5d |    2 |   1 |   1 | 09/12/2023 |       2 |     1260 |      1 | sensor_07 |   01:07:38 |     7 |     0 |     0
 a7c02170-50c5-11ee-8992-089798cc0a5d |    3 |   2 |   1 | 09/12/2023 |       0 |     1200 |      2 | sensor_01 |   01:07:14 |    10 |     0 |     2
 c3969870-50c5-11ee-af7b-089798cc0a5d |    2 |   2 |   2 | 09/12/2023 |       2 |     1280 |      0 | sensor_09 |   01:08:01 |    10 |     2 |     0
 b1fb4d40-50c5-11ee-8635-089798cc0a5d |    5 |   1 |   1 | 09/12/2023 |       1 |     1270 |      2 | sensor_08 |   01:07:31 |    12 |     1 |     1
 bc59df8f-50c5-11ee-9026-089798cc0a5d |    0 |   2 |   4 | 09/12/2023 |       0 |     1200 |      2 | sensor_01 |   01:07:49 |     8 |     0 |     0
 af05db00-50c5-11ee-842d-089798cc0a5d |    4 |   2 |   4 | 09/12/2023 |       1 |     1290 |      1 | sensor_10 |   01:07:27 |    14 |     2 |     0
 a857f4a1-50c5-11ee-ba1c-089798cc0a5d |    0 |   2 |   1 | 09/12/2023 |       2 |     1230 |      2 | sensor_04 |   01:07:15 |    11 |     1 |     3
 bff1fa70-50c5-11ee-8cfe-089798cc0a5d |    4 |   0 |   1 | 09/12/2023 |       2 |     1280 |      2 | sensor_09 |   01:07:55 |     9 |     0 |     0
 cc0df430-50c5-11ee-83ca-089798cc0a5d |    4 |   2 |   0 | 09/12/2023 |       2 |     1250 |      2 | sensor_06 |   01:08:15 |    15 |     2 |     3
 a2e2672e-50c5-11ee-8775-089798cc0a5d |    0 |   2 |   4 | 09/12/2023 |       1 |     1290 |      1 | sensor_10 |   01:07:06 |     9 |     1 |     0
 ba88a200-50c5-11ee-ac63-089798cc0a5d |    1 |   1 |   0 | 09/12/2023 |       0 |     1290 |      1 | sensor_10 |   01:07:46 |     8 |     2 |     3
 bc3b3400-50c5-11ee-91cc-089798cc0a5d |    0 |   1 |   2 | 09/12/2023 |       1 |     1260 |      2 | sensor_07 |   01:07:49 |     9 |     1 |     2
 d04905cf-50c5-11ee-8fdc-089798cc0a5d |    3 |   0 |   2 | 09/12/2023 |       0 |     1250 |      0 | sensor_06 |   01:08:22 |     7 |     1 |     1
 a7214961-50c5-11ee-b696-089798cc0a5d |    2 |   1 |   1 | 09/12/2023 |       0 |     1280 |      2 | sensor_09 |   01:07:13 |     9 |     1 |     2
 a7bd1430-50c5-11ee-b710-089798cc0a5d |    2 |   2 |   0 | 09/12/2023 |       0 |     1210 |      2 | sensor_02 |   01:07:14 |     7 |     1 |     0
 ac996ede-50c5-11ee-a8c0-089798cc0a5d |    0 |   0 |   3 | 09/12/2023 |       0 |     1290 |      2 | sensor_10 |   01:07:22 |     9 |     1 |     3
 b02a84de-50c5-11ee-bb24-089798cc0a5d |    4 |   2 |   4 | 09/12/2023 |       0 |     1270 |      2 | sensor_08 |   01:07:28 |    16 |     1 |     3
 b2a87d2e-50c5-11ee-a9e6-089798cc0a5d |    5 |   0 |   1 | 09/12/2023 |       1 |     1290 |      0 | sensor_10 |   01:07:33 |    11 |     1 |     3

(654 rows)
cqlsh:dskeyspacebd>
```

The Big Data Streaming Pipeline is now complete using Pycharm, Kafka, Pyspark, and Cassandra.

References:

Marquez, P. (2020). *Big Data Streaming*. Medium. https://medium.com/analytics-vidhya/big-data-streaming-c483195751ee

Foley, D. (2019). *Let's Build a Streaming Data Pipeline*. Medium. https://towardsdatascience.com/lets-build-a-streaming-data-pipeline-e873d671fc57

Jeyaraman, B. (2023). *Building Real-time Data Pipelines with Kafka and Machine Learning*. LinkedIn. https://www.linkedin.com/pulse/building-real-time-data-pipelines-kafka-machine-brindha-jeyaraman

Bouras, S. (2017). *Reliable and faster integration tests with Apache Cassandra*. https://medium.com/@saidbouras/running-integration-tests-with-apache-cassandra-42305dc260a6

Kaplarevic, V. (2020). *How to Install Cassandra on Windows 10*. PhoenixNAP. https://phoenixnap.com/kb/install-cassandra-on-windows

DataMaking (2019). *Real-Time Spark Project | Data Analysis | Structured Streaming | Part 6.2 | DM | DataMaking*. YouTube. https://www.youtube.com/watch?v=ZjsObNeUUd4&list=PLe1T0uBrDrfOYE8OwQvooPjmnP1zY3wFe&index=10

Cherif, A. N. (2020). *Beginners guide to learn Cassandra-Part 1: Cassandra overview*. Medium. https://medium.com/@aymannaitcherif/beginners-guide-to-learn-cassandra-part-1-cassandra-overview-bf1634e4ce30

Apache Cassandra (n.d.). *The Cassandra Query Language (CQL)*. The Apache Software Foundation. https://cassandra.apache.org/doc/latest/cassandra/cql/

DataStax (n.d.). *Querying tables*. DataStax Documentation. https://docs.datastax.com/en/cql-oss/3.x/cql/cql_using/useQueryDataTOC.html

jumpstartCS (2020). *Apache Cassandra Tutorial*. YouTube. https://www.youtube.com/playlist?list=PLalrWAGybpB-L1PGA-NfFu2uiWHEsdscD