- Using a shorter time window resulted in relatively quicker but more random results. With a shorter time window, the results were more varied.

```
    }
    producer.send(TOPIC, event)
    print(f"Sent event: {event}")
    time.sleep(random.uniform(0.5, 2.0))  # random interval

```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

```
Sent event: {'song_id': 101, 'timestamp': 1742242742.736314, 'region': 'US', 'action': 'skip'}
Sent event: {'song_id': 303, 'timestamp': 1742242743.6386163, 'region': 'US', 'action': 'like'}
Sent event: {'song_id': 404, 'timestamp': 1742242744.233173, 'region': 'APAC', 'action': 'skip'}
Sent event: {'song_id': 404, 'timestamp': 1742242746.04255, 'region': 'APAC', 'action': 'play'}
Sent event: {'song_id': 101, 'timestamp': 1742242747.185981, 'region': 'EU', 'action': 'skip'}
Sent event: {'song_id': 505, 'timestamp': 1742242748.0372932, 'region': 'EU', 'action': 'like'}
Sent event: {'song_id': 101, 'timestamp': 1742242749.2616951, 'region': 'US', 'action': 'skip'}
Sent event: {'song_id': 404, 'timestamp': 1742242750.4120479, 'region': 'APAC', 'action': 'like'}
Sent event: {'song_id': 303, 'timestamp': 1742242751.9350548, 'region': 'US', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1742242752.8594189, 'region': 'US', 'action': 'play'}
Sent event: {'song_id': 202, 'timestamp': 1742242754.449979, 'region': 'US', 'action': 'play'}
```

- Updated ‘now_trending’.py:

```
# now_trending.py

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, desc, window, from_json, sum as spark_sum
from pyspark.sql.functions import desc
from pyspark.sql.types import StructType, StructField, StringType, DoubleType
from pyspark.sql.types import TimestampType
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number

# 1) Create SparkSession
spark = SparkSession.builder \
    .appName("NowTrendingSongs") \
    .getOrCreate()

```

```
spark.sparkContext.setLogLevel("WARN")

# 2) Read from Kafka
kafka_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "music_events") \
    .option("startingOffsets", "latest") \
    .load()

# 3) Parse the JSON 'value' from Kafka
schema = StructType([
    StructField("song_id", StringType(), True),
    StructField("timestamp", DoubleType(), True),  # or we can store as Double
    StructField("region", StringType(), True),
    StructField("action", StringType(), True)
])

json_df = kafka_df.selectExpr("CAST(value AS STRING) as json_str")

parsed_df = json_df.select(from_json(col("json_str"), schema).alias("data"))
events_df = parsed_df.select("data.*")

# Convert timestamp double -> actual timestamp if we want event time
# But for simplicity, let's do a processing-time approach
# If you want event-time windows, do:
# events_df = events_df.withColumn("event_time", (col("timestamp") * 1000).cast(TimestampType()))

# 4) Filter only "play" events
plays_df = events_df.filter(col("action") == "play")

```

```
# 5) Group by region + 5-minute processing time window
# We'll do a simple processing-time window using current_timestamp
# Alternatively, you can do event-time with a column if you convert 'timestamp' to a Spark timestamp
from pyspark.sql.functions import current_timestamp

windowed_df = plays_df \
    .groupBy(
        window(current_timestamp(), "1 minute"),  # processing-time window
        col("region"),
        col("song_id")
    ) \
    .count()

# 6) Use foreachBatch to do rank-based top N logic each micro-batch
def process_batch(batch_df, batch_id):
    """
    This function is called for each micro-batch. We treat 'batch_df' as a normal batch DataFrame.
    We'll rank by 'count' within each region & window and pick top 3 (or 5, or 100).
    """
    if batch_df.rdd.isEmpty():
        print("No data in this batch.")
        return

    # We'll partition by region + the 'window' column
    w = Window.partitionBy("region", "window").orderBy(desc("count"))

    ranked_df = batch_df.withColumn("rn", row_number().over(w)) \
                        .filter(col("rn") <= 3)
```

```
    # Show the top songs for each region + 5-min window
    print(f"=== Batch: {batch_id} ===")
    ranked_df.show(truncate=False)

# 7) Write Stream with foreachBatch
query = windowed_df \
    .writeStream \
    .outputMode("update") \
    .foreachBatch(process_batch) \
    .trigger(processingTime='5 seconds') \
    .start()

skip_ratio_df = (
    events_df.groupBy("song_id")
    .agg(
        count("*").alias("total_events"),
        count(col("action")).alias("total_skips"),
    )
    .withColumn("skip_ratio", col("total_skips") /
col("total_events"))
)

query = (
    skip_ratio_df.writeStream
    .outputMode("complete")
    .format("console")
    .start()
)

query.awaitTermination()
```

- Updated music_producer.py:

```
# producer.py
```

```python
import time
import random
import json
from kafka import KafkaProducer

# pip install kafka-python

TOPIC = "music_events"
producer = KafkaProducer(
    bootstrap_servers="localhost:9092",
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

songs = [101, 202, 303, 404, 505]   # sample song IDs
regions = ["US", "EU", "APAC"]
actions = ["play", "skip", "like"]

while True:
    event = {
        "song_id": random.choice(songs),
        "timestamp": time.time(),
        "region": random.choice(regions),
        "action": random.choice(actions)  # or skip, etc.
    }
    producer.send(TOPIC, event)
    print(f"Sent event: {event}")
    time.sleep(random.uniform(0.5, 2.0))  # random interval
```