databricks Assignment 5 - Richard Lopez

(https://databricks.com)

Assignment 5.1: Hands-on with Databricks

The goal of this assignment is to interactively compute real-time metrics, such as running counts and windowed counts on a stream of time stamped actions (e.g., Open, Close, etc.).

In this exercise, we will build a streaming application to process streaming in real-time. The step by step process is listed one by one.

1. Access and download the Dataset

```
%fs ls /databricks-datasets/structured-streaming/events/
```

Table

	path	name	size	modificationTime
1	dbfs:/databricks-datasets/structured-streaming/events/file-0.json	file-0.json	72530	1469673865000
2	dbfs:/databricks-datasets/structured-streaming/events/file-1.json	file-1.json	72961	1469673866000
3	dbfs:/databricks-datasets/structured-streaming/events/file-10.json	file-10.json	73025	1469673878000
4	dbfs:/databricks-datasets/structured-streaming/events/file-11.json	file-11.json	72999	1469673879000
5	dbfs:/databricks-datasets/structured-streaming/events/file-12.json	file-12.json	72987	1469673880000
6	dbfs:/databricks-datasets/structured-streaming/events/file-13.json	file-13.json	73006	1469673881000
7	dbfs:/databricks-datasets/structured-streaming/events/file-14.json	file-14.json	73003	1469673882000

50 rows

Load the data – this dataset is pre-packaged with Databricks

```
%fs head /databricks-datasets/structured-streaming/events/file-0.json
```

```
[Truncated to first 65536 bytes]
{"time":1469501107,"action":"Open"}
{"time":1469501147,"action":"Open"}
{"time":1469501202,"action":"Open"}
{"time":1469501219,"action":"Open"}
{"time":1469501225,"action":"Open"}
{"time":1469501234,"action":"Open"}
{"time":1469501245,"action":"Open"}
{"time":1469501246,"action":"Open"}
{"time":1469501248,"action":"Open"}
{"time":1469501256,"action":"Open"}
{"time":1469501264,"action":"Open"}
{"time":1469501266,"action":"Open"}
{"time":1469501267,"action":"Open"}
{"time":1469501269,"action":"Open"}
{"time":1469501271,"action":"Open"}
{"time":1469501282,"action":"Open"}
{"time":1469501285,"action":"Open"}
{"time":1469501291,"action":"Open"}
{"time":1469501297,"action":"Open"}
{"time":1469501303,"action":"Open"}
```

Query the data to examine the share/format of the underlying dataset

```
from pyspark.sql.types import *

inputPath = "/databricks-datasets/structured-streaming/events/"

# Since we know the data format already, let's define the schema to speed up processing (no need for Spark to infer schema)
jsonSchema = StructType([ StructField("time", TimestampType(), True), StructField("action", StringType(), True) ])

# Static DataFrame representing data in the JSON files
staticInputDF = (
  spark
    .read
    .schema(jsonSchema)
    .json(inputPath)
)

display(staticInputDF)
```

Table

	time	action
1	2016-07-28T04:19:28.000+0000	Close
2	2016-07-28T04:19:28.000+0000	Close
3	2016-07-28T04:19:29.000+0000	Open
4	2016-07-28T04:19:31.000+0000	Close
5	2016-07-28T04:19:31.000+0000	Open
6	2016-07-28T04:19:31.000+0000	Open
7	2016-07-28T04:19:32.000+0000	Close

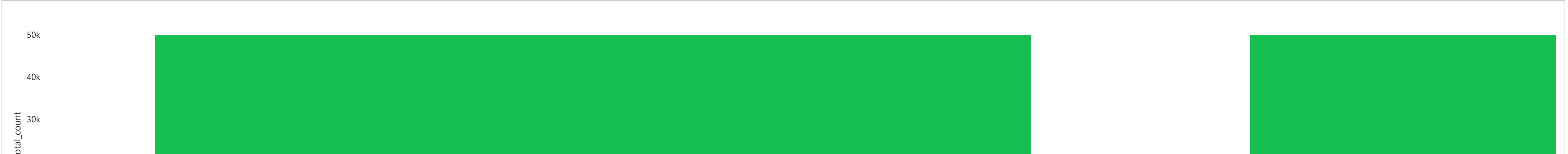
1,000 rows | Truncated data

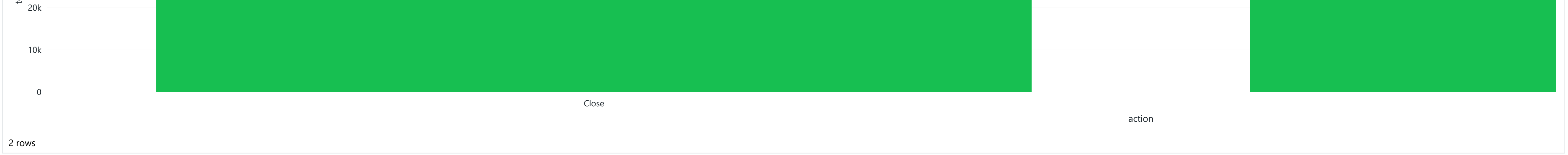
Visualize the number of "open" and "close" actions across all the hours

```
from pyspark.sql.functions import *      # for window() function

staticCountsDF = (
  staticInputDF
    .groupBy(
       staticInputDF.action,
       window(staticInputDF.time, "1 hour"))
    .count()
)
staticCountsDF.cache()

# Register the DataFrame as table 'static_counts'
staticCountsDF.createOrReplaceTempView("static_counts")
```

```
%sql select action, sum(count) as total_count from static_counts group by action
```

Table   Visualization 1



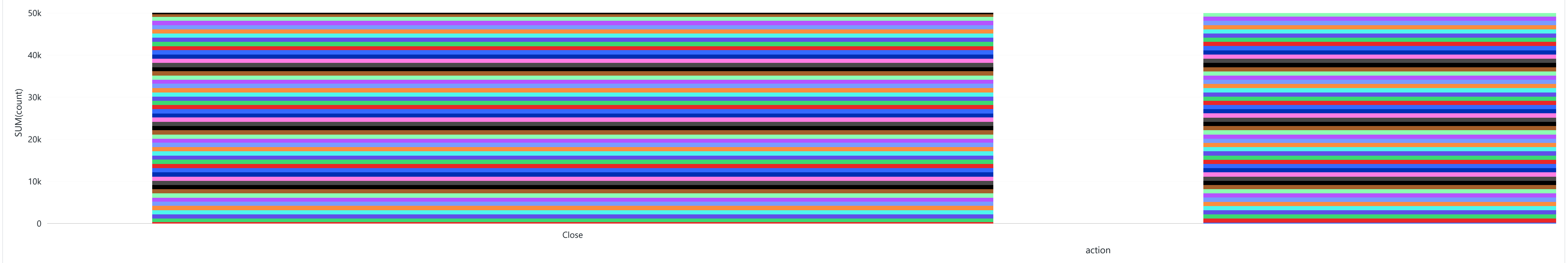


2 rows

Visualize the number of "open" and "close" actions over a timeline of “2 hour” windowed counts

```
%sql select action, date_format(window.end, "MMM-dd HH:mm") as time, count from static_counts order by time, action
```

Table Visualization 1

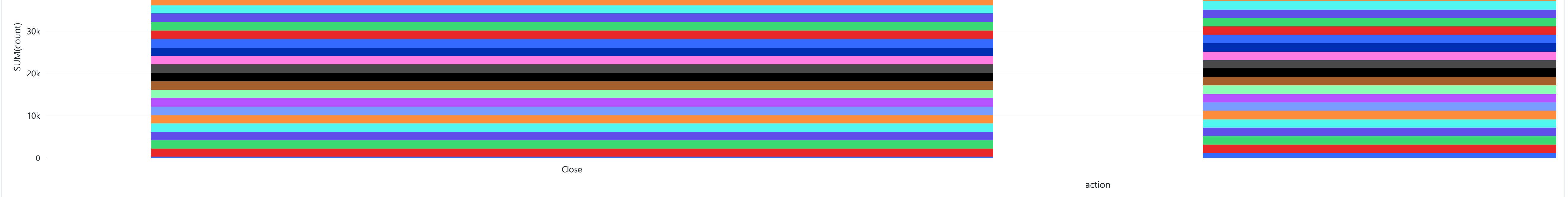


104 rows

```
from pyspark.sql.functions import *      # for window() function

staticCountsDF = (
  staticInputDF
    .groupBy(
       staticInputDF.action,
       window(staticInputDF.time, "2 hours"))
    .count()
)
staticCountsDF.cache()

# Register the DataFrame as table 'static_counts'
staticCountsDF.createOrReplaceTempView("static_counts")
```

```
%sql select action, date_format(window.end, "MMM-dd HH:mm") as time, count from static_counts order by time, action
```

Table Visualization 1



53 rows

Convert the data to a streaming query that continuously updates as data comes i.e., emulate a stream that reads one file at a time

```
from pyspark.sql.functions import *

# Similar to definition of staticInputDF above, just using `readStream` instead of `read`
streamingInputDF = (
  spark
    .readStream
    .schema(jsonSchema)               # Set the schema of the JSON data
    .option("maxFilesPerTrigger", 1)  # Treat a sequence of files as a stream by picking one file at a time
    .json(inputPath)
)

# Same query as staticInputDF
streamingCountsDF = (
  streamingInputDF
    .groupBy(
      streamingInputDF.action,
      window(streamingInputDF.time, "1 hour"))
    .count()
)

# a streaming DF
streamingCountsDF.isStreaming
```

```
Out[22]: True
```

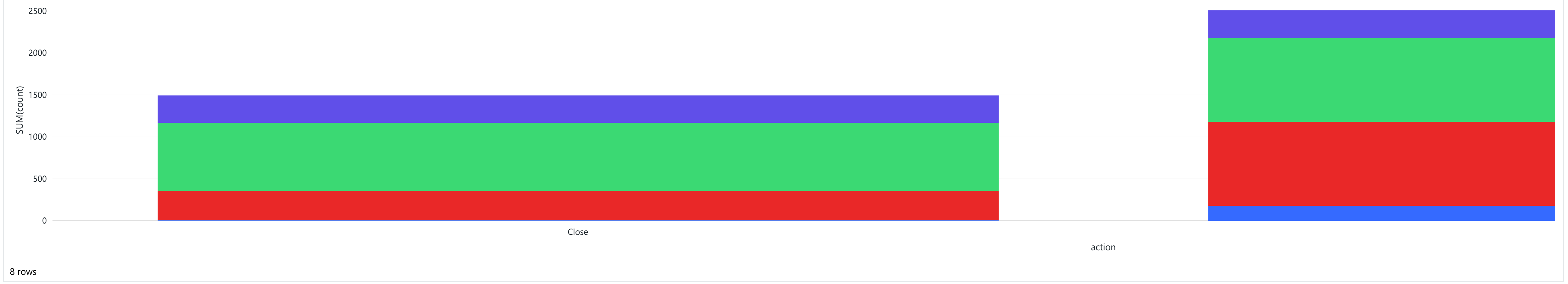
```
spark.conf.set("spark.sql.shuffle.partitions", "2")  # keep the size of shuffles small

query = (
  streamingCountsDF
    .writeStream
    .format("memory")        # memory = store in-memory table
    .queryName("counts")     # counts = name of the in-memory table
    .outputMode("complete")  # complete = all the counts should be in the table
    .start()
)
```

counts (id: 41126be5-3b1f-463c-a222-9eebe967608f) *Last updated: 6 minutes ago*

```
from time import sleep
sleep(5)  # wait a bit for computation to start
```

```
%sql select action, date_format(window.end, "MMM-dd HH:mm") as time, count from counts order by time, action
```

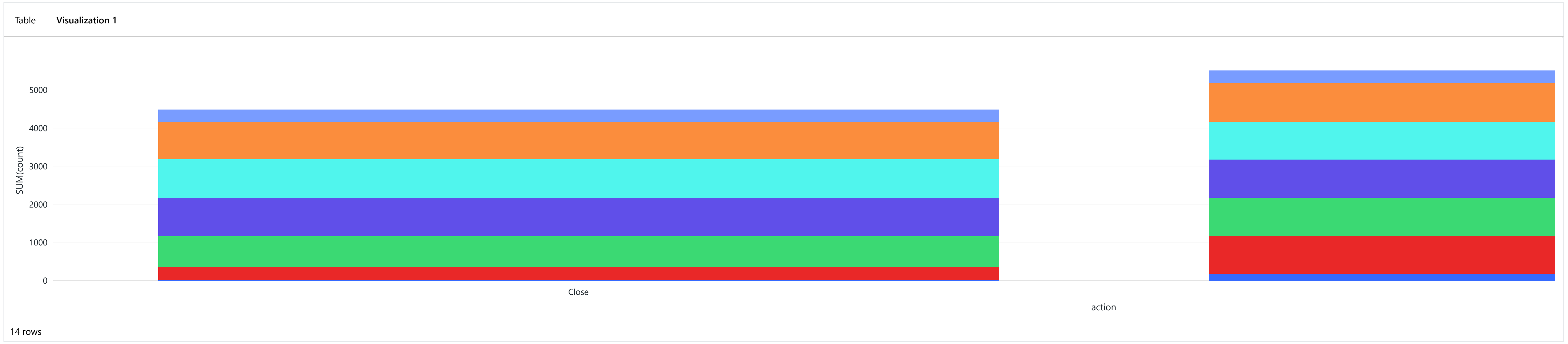
Table **Visualization 1**



8 rows

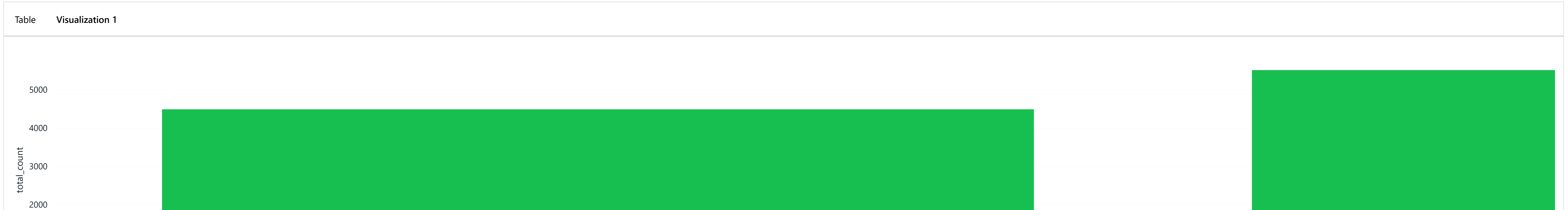
```
sleep(5)  # wait a bit for computation to start
```

```
%sql select action, date_format(window.end, "MMM-dd HH:mm") as time, count from counts order by time, action
```



14 rows

```
%sql select action, sum(count) as total_count from counts group by action order by action
```



2 rows

Summary

The lesson was an excellent introduction to working with Structured Streaming on Azure Databricks. At the end of the project, this set of features made sense to me. I've done a lot of work with streams and tools like RXJS while working with NodeJS. I see similarities to the problems we solve in data analytics with Structured Streaming. In the first part of this notebook, we start with a static loading of data and run queries on the data. In part 2, we use streams to read the data, and we can query in Stream Windows. The last two questions show the results changing while our Spark code was streaming in the events.

References

- Structured Streaming Programming Guide - Spark 3.3.2 Documentation. (2016). Apache.org. https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html (https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html)
- mssaperla. (2023, January 20). Structured Streaming patterns on Azure Databricks - Azure Databricks. Microsoft.com. https://learn.microsoft.com/en-us/azure/databricks/structured-streaming/examples (https://learn.microsoft.com/en-us/azure/databricks/structured-streaming/examples)