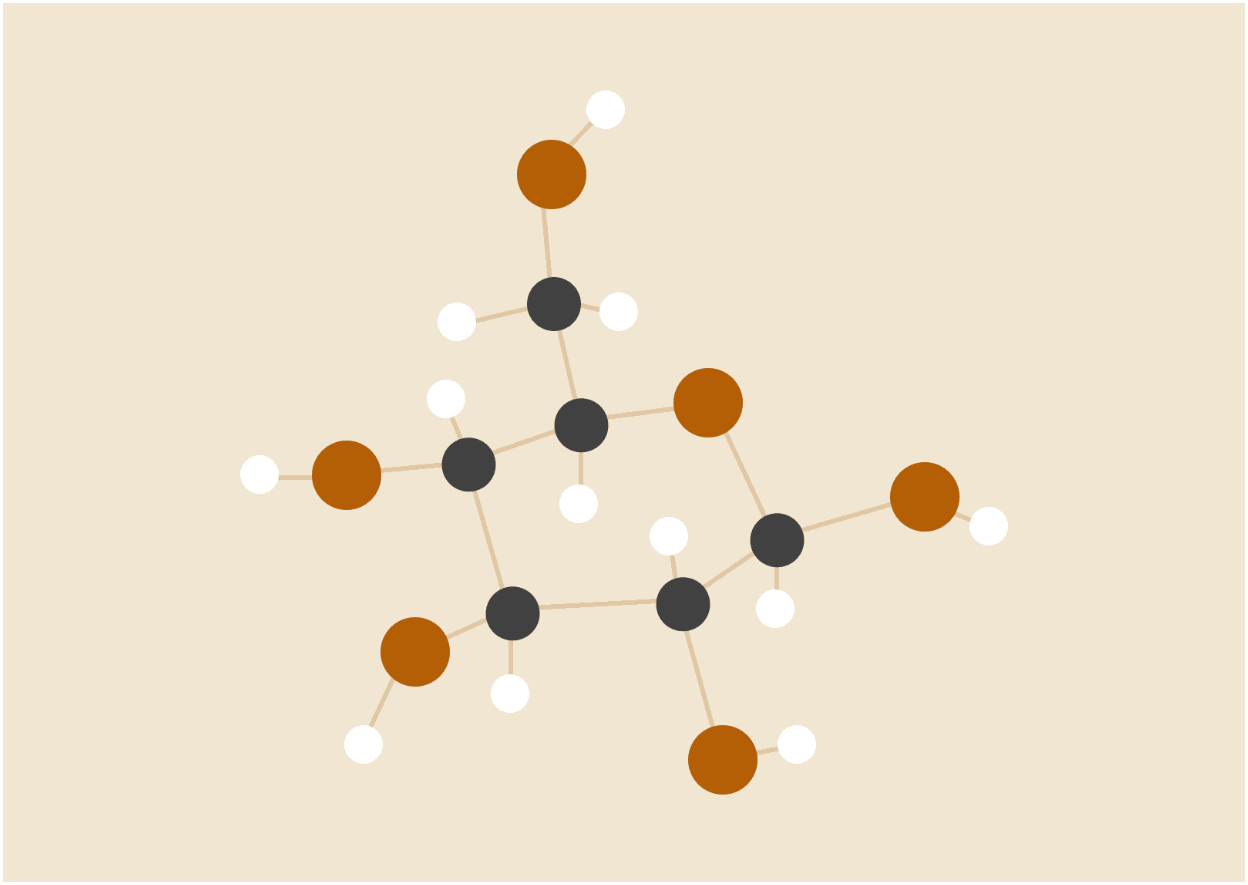
IBD GA3 REPORT

The task to Write a spark code for executing the Hashing example using Google Cloud Dataproc.



Anchit Mandal

21f1000692

INTRODUCTION

In this task, we aim to analyze clickstream data by categorizing user clicks into different hourly blocks and counting the number of clicks in each block. This analysis is implemented using Apache Spark, a powerful distributed data processing framework. The data is stored in Google Cloud Storage (GCS), and the entire process is executed within a Google Cloud Platform (GCP) environment.

The key steps involved in this task are as follows:

Data Preparation: Create a clickstream data file that records user activities at different times of the day.

Setting up the Environment: Configure a Google Cloud environment and initialize a Spark session to process the data.

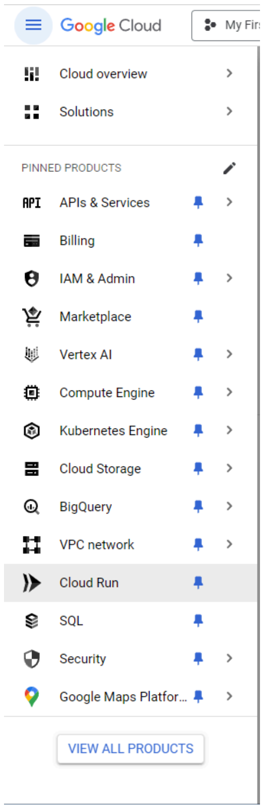
Data Processing: Develop a Python script to read the clickstream data, categorize the clicks into hourly blocks, and count the number of clicks in each block.

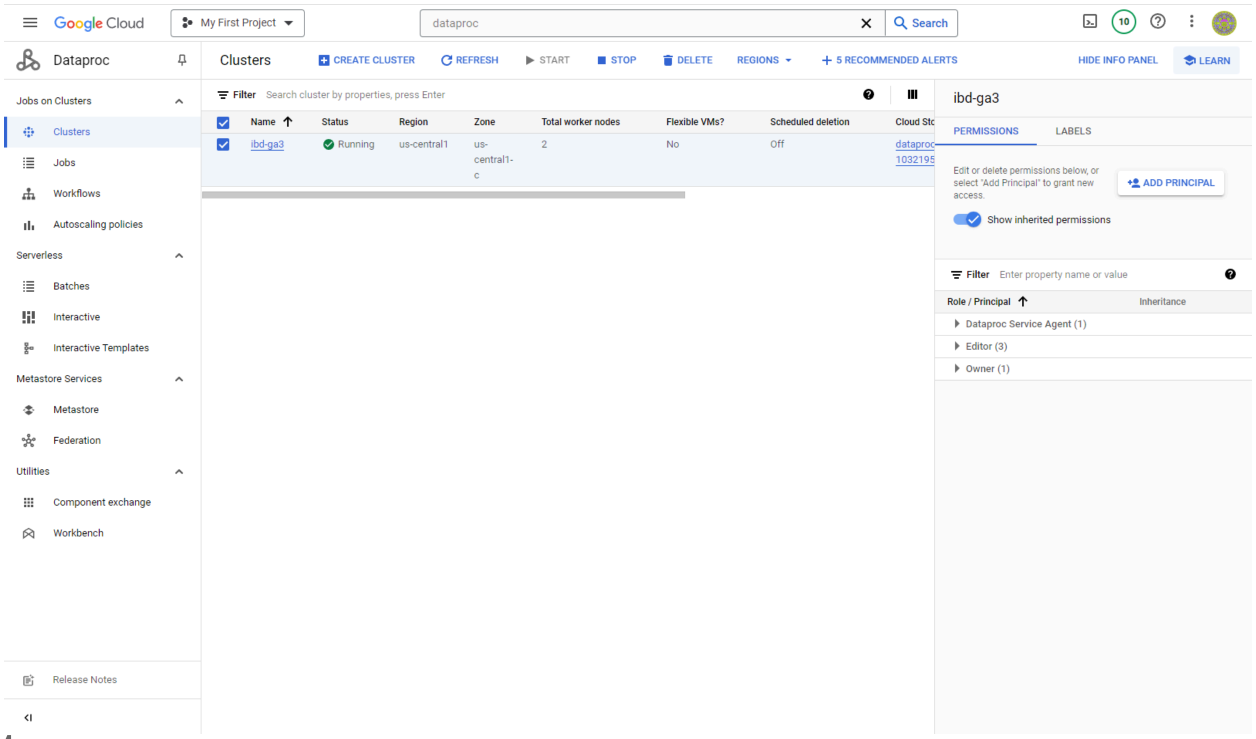
Saving the Results: Write the output, which contains the click counts per hourly block, back to GCS.

Documentation and Presentation: Compile the code, input file, output file, and a detailed report into a zip file for submission. Additionally, create a video presentation explaining the solution.

This report provides a comprehensive overview of each step involved in this task. It begins with the data preparation and environment setup, followed by a detailed explanation of the Spark script used for data processing. The results of the analysis are then presented, and the process of saving these results back to GCS is described. Finally, the report includes relevant screenshots, code explanations, and outputs to ensure a thorough understanding of the entire workflow.

DATAPROC





Name	Status	Region	Zone	Total worker nodes	Flexible VMs?	Scheduled deletion	Cloud Sto
ibd-ga3	Running	us-central1	us-central1-c	2	No	Off	dataproc 1032195

Edit or delete permissions below, or select "Add Principal" to grant new access.

Show inherited permissions: ✓

Role / Principal	Inheritance
Dataproc Service Agent (1)	
Editor (3)	
Owner (1)	

Creating the Cloud Function

From the Dataproc section in the GCP Console, I created a cluster for my task with the following settings:

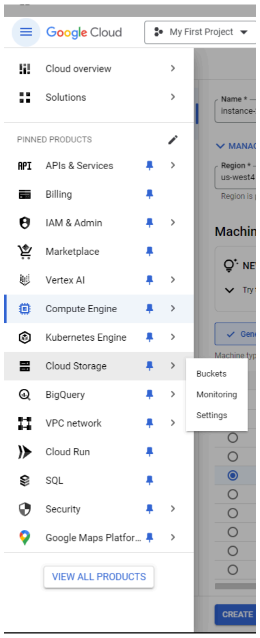
Name: ibd-ga3

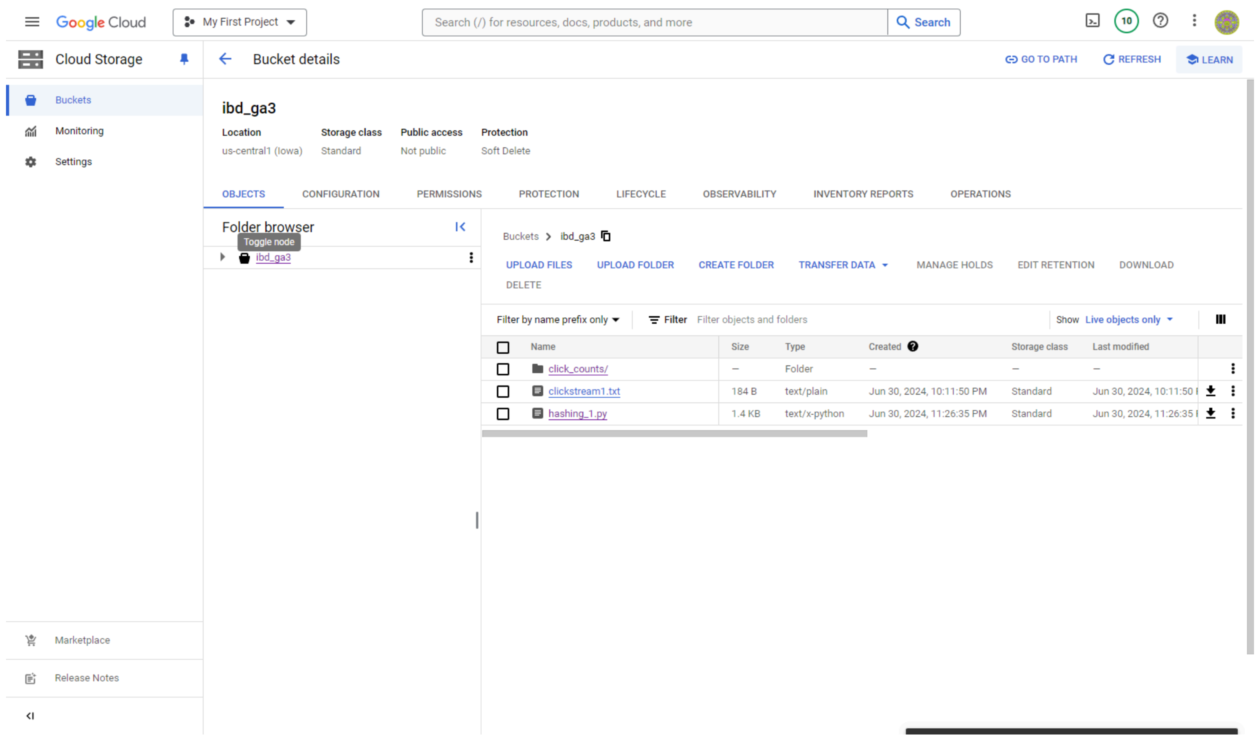
Machine type: e2-standard-2

Primary disk size: 50GB

Rest configurations: default

BUCKET





ibd_ga3

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	Not public	Soft Delete

Name	Size	Type	Created	Storage class	Last modified
click_counts/	—	Folder	—	—	—
clickstream1.txt	184 B	text/plain	Jun 30, 2024, 10:11:50 PM	Standard	Jun 30, 2024, 10:11:50 I
hashing_1.py	1.4 KB	text/x-python	Jun 30, 2024, 11:26:35 PM	Standard	Jun 30, 2024, 11:26:35 I

From the Bucket section in the Cloud Storage product in GCP, I created a bucket named ibd_ga3 to store the text file, python code for my task.

CODE EXPLANATION

Python
```python
from pyspark.sql import SparkSession

from pyspark.sql.functions import udf

from pyspark.sql.types import StringType

import datetime



# Initialize Spark session

spark =
SparkSession.builder.appName("ClickstreamAnalysis").getOrCreate()
```

```
# Define the input file path

input_file = "gs://ibd_ga3/clickstream1.txt"


# Read the input file

df = spark.read.csv(input_file, inferSchema=True, header=False)


df = df.toDF("user_id", "time", "date")


def map_to_hour_block(time):

    # Check if 'time' is already a datetime object

    if isinstance(time, datetime.datetime):

        hour = time.hour

    else:

        # Fallback if 'time' is a string

        hour = int(time.split(":")[0])


    if 0 <= hour < 6:

        return "00-06"

    elif 6 <= hour < 12:
        return "06-12"

    elif 12 <= hour < 18:

        return "12-18"

    else:

        return "18-24"


# Register the UDF

map_to_hour_block_udf = udf(map_to_hour_block, StringType())



# Apply the UDF to create a new column 'hour_block'

df_with_blocks = df.withColumn("hour_block",
map_to_hour_block_udf(df["time"]))



# Group by hour_block and count the clicks

click_counts =
df_with_blocks.groupBy("hour_block").count().orderBy("hour_block"
)



# Show the results

click_counts.show()

# Define the output file path
```

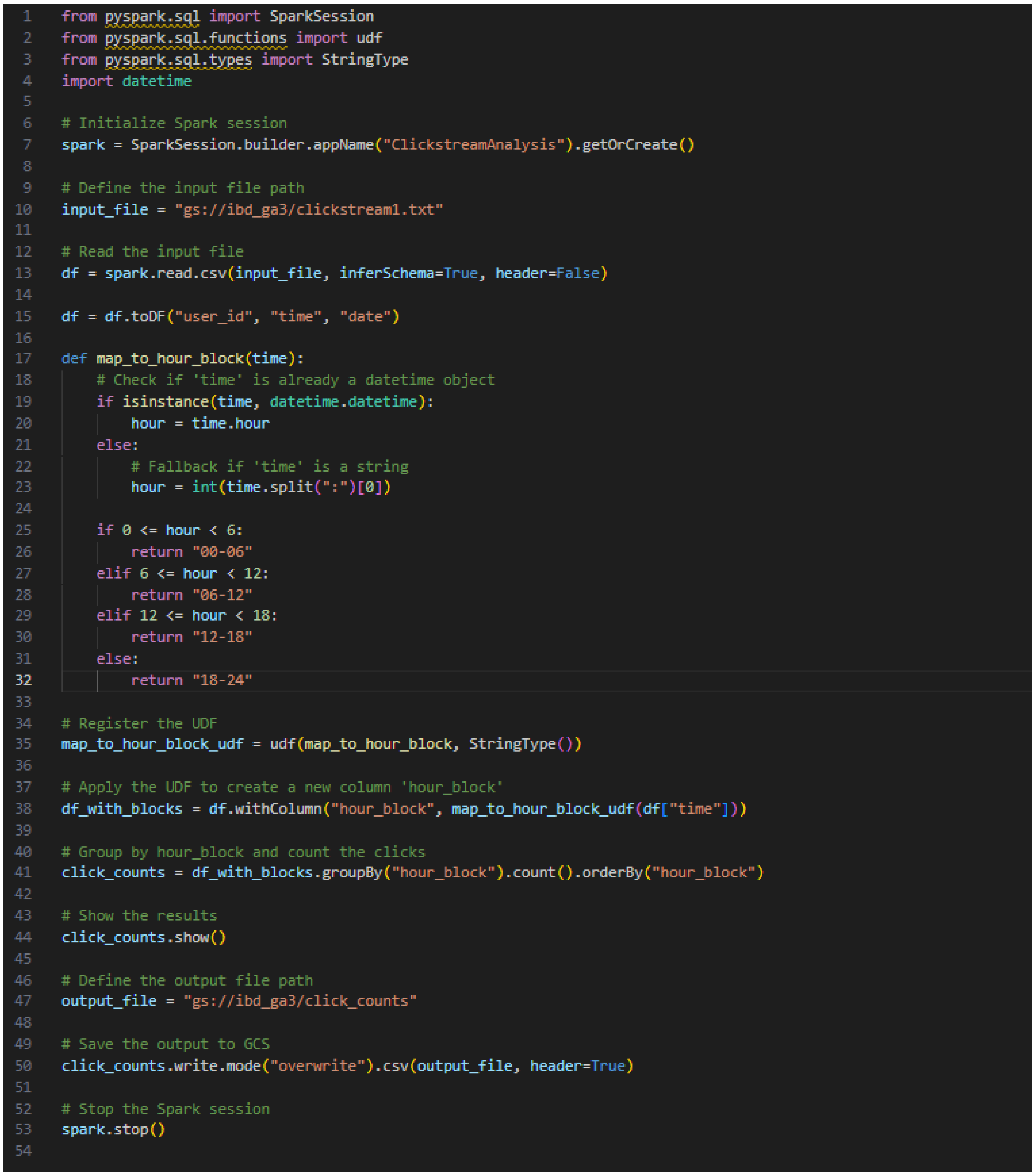
```
output_file = "gs://ibd_ga3/click_counts"


# Save the output to GCS

click_counts.write.mode("overwrite").csv(output_file,
header=True)


# Stop the Spark session

spark.stop()
```

- Import necessary modules from pyspark.sql and datetime.
- Initialize a Spark session with the application name "ClickstreamAnalysis".
- Define the path to the input file stored in Google Cloud Storage.
- Read the CSV file from the specified path into a Spark DataFrame.
- inferSchema=True automatically infers the data types of the columns.
- header=False indicates that the CSV file does not have a header row.
- Rename the columns of the DataFrame to "user_id", "time", and "date" for clarity.
- Define a function map_to_hour_block that takes a time value and returns a corresponding hour block.
- Check if the input time is a datetime.datetime object. If so, extract the hour directly.
- If the input time is a string, split it by ":" and convert the hour part to an integer.

- Map the hour to one of the four predefined hour blocks ("00-06", "06-12", "12-18", "18-24").
- Register the map_to_hour_block function as a User-Defined Function (UDF) so that it can be used in Spark SQL queries.
- Specify that the UDF returns a StringType.
- Apply the UDF to the "time" column to create a new column "hour_block" in the DataFrame.
- Group the DataFrame by the "hour_block" column and count the number of clicks (rows) in each block.
- Order the results by "hour_block" for better readability.
- Display the resulting DataFrame containing the counts of clicks in each hour block.
- Define the path to the output file where the results will be saved in Google Cloud Storage.
- Save the resulting DataFrame to the specified output path in CSV format.
- Use mode("overwrite") to overwrite any existing files at the output path.
- Include a header row in the output CSV file.
- Stop the Spark session to release resources.

REQUIREMENTS FILE

Unset
```
pyspark
```


```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import datetime

# Initialize Spark session
spark = SparkSession.builder.appName("ClickstreamAnalysis").getOrCreate()

# Define the input file path
input_file = "gs://ibd_ga3/clickstream1.txt"

# Read the input file
df = spark.read.csv(input_file, inferSchema=True, header=False)

df = df.toDF("user_id", "time", "date")

def map_to_hour_block(time):
    # Check if 'time' is already a datetime object
    if isinstance(time, datetime.datetime):
        hour = time.hour
    else:
        # Fallback if 'time' is a string
        hour = int(time.split(":")[0])

    if 0 <= hour < 6:
        return "00-06"
    elif 6 <= hour < 12:
        return "06-12"
    elif 12 <= hour < 18:
        return "12-18"
    else:
        return "18-24"

# Register the UDF
map_to_hour_block_udf = udf(map_to_hour_block, StringType())

# Apply the UDF to create a new column 'hour_block'
df_with_blocks = df.withColumn("hour_block", map_to_hour_block_udf(df["time"]))

# Group by hour_block and count the clicks
click_counts = df_with_blocks.groupBy("hour_block").count().orderBy("hour_block")

# Show the results
click_counts.show()

# Define the output file path
output_file = "gs://ibd_ga3/click_counts"

# Save the output to GCS
click_counts.write.mode("overwrite").csv(output_file, header=True)

# Stop the Spark session
spark.stop()
```

TEST FILE

The file contains 4 line, ie

```
Unset
user1,05:30,2024-03-01
user2,08:45,2024-03-01
user3,14:20,2024-03-01
user1,18:15,2024-03-01
user2,22:00,2024-03-01
user4,03:10,2024-03-02
user1,11:05,2024-03-02
user3,16:40,2024-03-02
```

OUTPUT

The Spark job successfully processed the clickstream data and categorized the user clicks into four hourly blocks. The results of the click counts per hour block are as follows:

```
Unset
+----------+-----+
|hour_block|count|
+----------+-----+
|   00-06|   2|
|   06-12|   2|
|   12-18|   2|

|   18-24|   2|

+----------+-----+
```

The Spark job logs indicated that the output was written to Google Cloud Storage without any issues:

```
Unset

24/06/30 17:52:35 INFO PathOutputCommitterFactory: No output
committer factory defined, defaulting to
FileOutputCommitterFactory

24/06/30 17:52:39 INFO GoogleCloudStorageFileSystemImpl:
Successfully repaired 'gs://ibd_ga3/click_counts/' directory.
```

Upon submitting the job, the Spark job was executed, and the output was saved and logged as expected.

CONCLUSION

This task successfully demonstrates the use of Apache Spark for processing and analyzing clickstream data stored in Google Cloud Storage. By categorizing the clicks into hourly blocks and counting the number of clicks in each block, we were able to efficiently process and summarize the user activity data.

The implementation showcases the following key capabilities:

Data Processing: Efficiently reading and processing large datasets using Spark.

Categorization: Using custom functions to map data into meaningful categories.

Storage and Retrieval: Interacting with Google Cloud Storage for both input and output data.

Automation: Leveraging cloud infrastructure to automate data processing tasks.

Overall, this task highlights the powerful combination of Spark and GCP for big data processing and provides a robust framework for similar data analysis tasks in the future.