Assignment 3 Report: Spark and Kafka Logs

Part 1: Top 100 most frequently occurring words in a 16GB dataset using Spark, considering only words with more than 6 characters.

Solution: We used Spark's map reduce approach to find the top 100 frequent words in a 16GB dataset. By converting the RDD to a DataFrame, we filtered out short words and performed transformations to calculate word frequencies. We tested performance by varying input block sizes and measuring execution time.

Analysis: Using Spark for MapReduce functions provides different performance profiles.

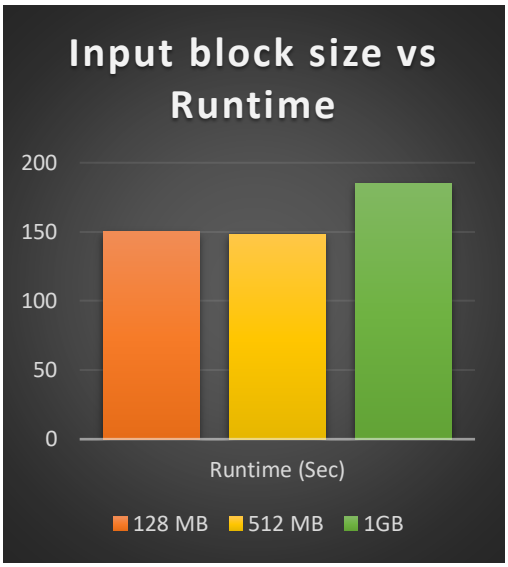

Figure 2 Spark streaming top 100 words, Runtime vs block size

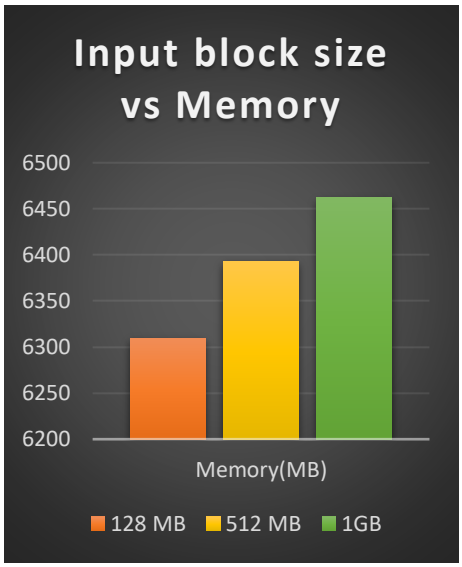

Figure 1 Spark streaming top 100 words, Memory vs block size

The two main distinctions are:

1. Changing the input block size had around 1-2% impact on memory use and 20% on time. But in Hadoop MapReduce, memory usage had a major impact. This resonates with Spark trying to use main memory to process data rather than Hadoop, which uses disk and memory both to process data.
2. The overall processing time is greatly reduced in comparison with Hadoop MapReduce. Compared to Hadoop, the trade-off for such faster performance is achieved by consuming almost 2-3x memory.

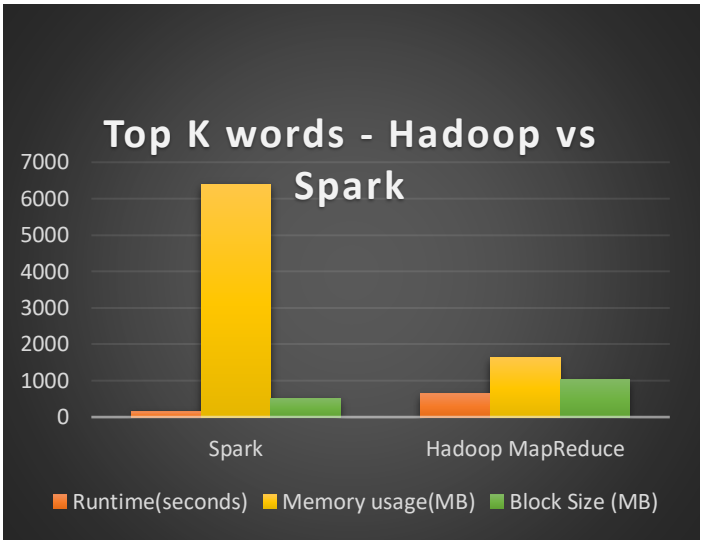

Figure 3 Top K words, Hadoop vs spark

Table 1: Top K words, Hadoop vs spark

	Runtime (seconds)	Memory usage (MB)	Block size (MB)
Hadoop MapReduce	638	1619	1024
Spark	148	6393	512

We compared the runtime, memory usage, and block size of Hadoop MapReduce and Spark. Spark demonstrated significantly better performance with a runtime of 148 seconds compared to Hadoop MapReduce's 638 seconds. However, Spark consumed more memory, utilizing 6393 MB compared to Hadoop MapReduce's 1619 MB. Additionally, Spark had a smaller block size of 512 MB, potentially contributing to its faster processing speed.

Part 2: Log Analytics

Section A: Implement log analytics on NASA logs, perform EDA.

Following the analysis steps in the medium article helps us understand how Spark can be used for Data Wrangling and combing EDA tools for further analysis.

Handling Temporal Fields (Timestamp)

In our algorithm, we enhance a DataFrame with a timestamp column by splitting it into separate components in the "Handling Temporal Fields (Timestamp)" section. This allows for analysis and querying based on temporal information, such as the day of the week, year, month, date, hour, minute, and second.

Find the endpoint with the highest invocations on a specific day.

To identify the endpoint that received the highest number of invocations on a specific day of the week, we perform the following steps. We group the original DataFrame by 'day_of_week' and 'endpoint'. Using the `agg` function, we calculate the count of invocations for each group.

Approach: We utilize window functions to rank the rows within each 'day_of_week' group based on descending invocations. By ranking the rows instead of joining with another DataFrame, we reduce overhead. We then filter the resulting DataFrame to retain only the rows with rank 1, indicating the highest invocations. Finally, we select the 'day_of_week', 'endpoint', and 'invocations' columns for display, providing the endpoint with the highest number of invocations on a specific day of the week.

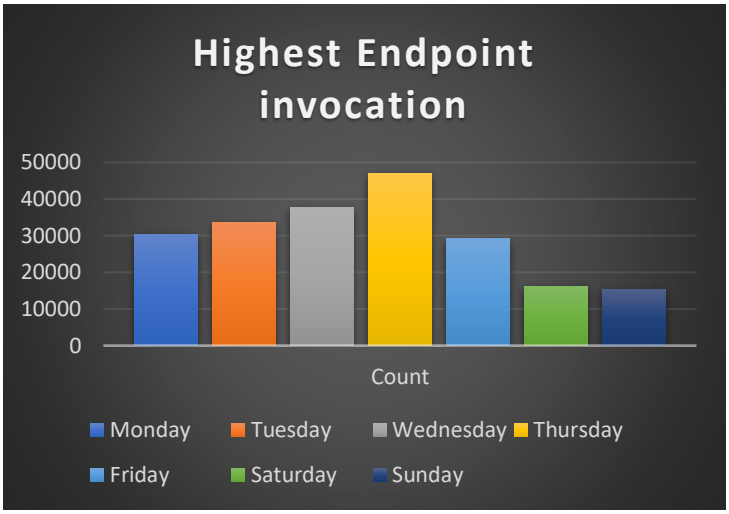

Figure 4 NASA logs Highest endpoint invocation by day of a week

Analysis: Looking at the count of unique hosts per day for a month suggests that there is always a spike in new users for the first five days, which gradually normalizes for the rest of the days until at last 2-3 days when it drops significantly. This behavior indicates that users are actively searching for new updates from NASA at the start of every month, and only those who find their area of interest tend to access the website endpoint frequently.

Table 2 Endpoint with the highest invocations on day of week

Days of week	Endpoint Invocations	Count
Monday	/images/NASA-logosmall.gif	30380
Tuesday	/images/NASA-logosmall.gif	33685
Wednesday	/images/NASA-logosmall.gif	37598
Thursday	/images/NASA-logosmall.gif	46920
Friday	/images/NASA-logosmall.gif	29139
Saturday	/images/NASA-logosmall.gif	16168
Sunday	/images/KSC-logosmall.gif	15218

number of invocations on Thursday with a count of 46,920.

Secondly, looking at the behavior of the highest endpoint invocation, which is a gif image NASA logo for most days except for Sunday when Kennedy Space Center (KSC) was accessed, most suggest that users are interested in downloading the NASA logo for personal use.

The table shows the count of invocations for a specific endpoint on different days of the week. The endpoint "/images/NASA-logosmall.gif" received the highest

Days of week, sorted by least 404 status codes.

To determine the days of the week with the least 404 status codes, we filter the DataFrame for 'status' equal to 404 and group it by the 'day_of_week' column. We count the occurrences of each day of the week and sort the result in ascending order.

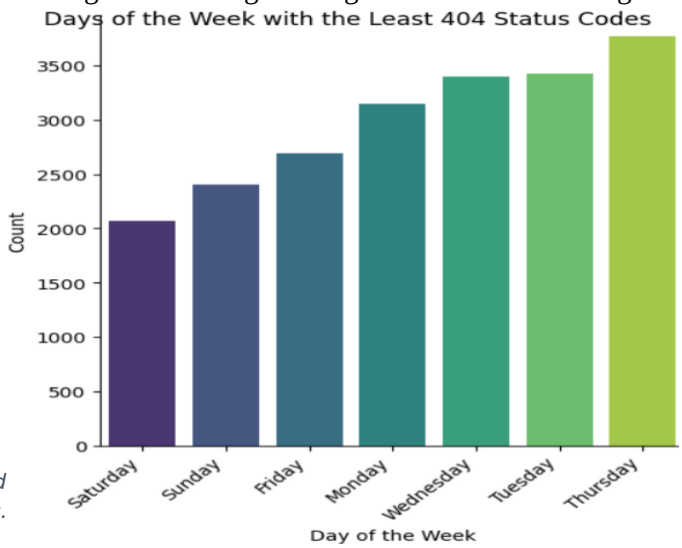

Figure 5 Days of week, sorted by least 404 status codes.

Table 3 Days of week, sorted by least 404 status codes

Days of Week	Count
Saturday	2069
Sunday	2400
Friday	2691
Monday	3145
Wednesday	3397
Tuesday	3425
Thursday	3772

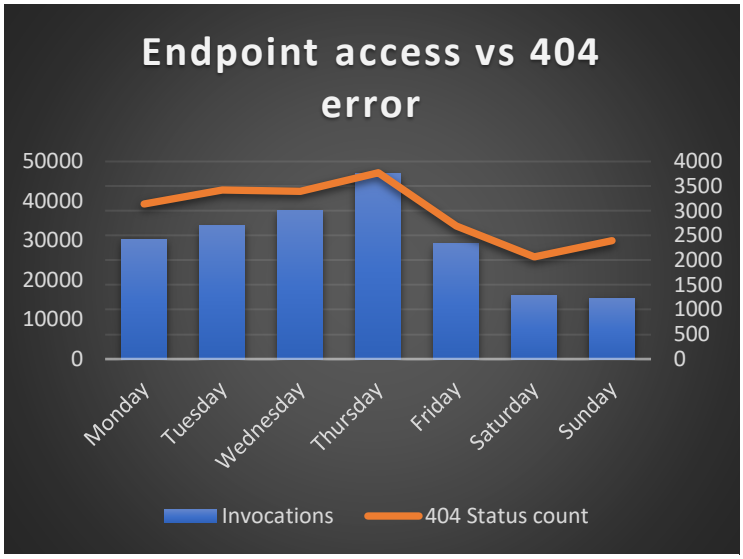

Figure 6 Endpoint access vs 404 Error

Analysis: We conclude that users trying to access the gif image for the NASA logo are probably trying to access another, but NASA mainly removed the file.

Identify the top 10 years with the least 404 status codes.

Lastly, looking at the 404-error code trend. For calculating the top 10 years with the least 404 status codes. We reused the filter and then group the filtered data by year, count the occurrences of each year, sort the result in ascending order based on the count column, and display the top 10 records.

Section B: Design a data flow using Kafka, Spark Streaming, Parquet files, and HDFS. Feed log data to Kafka producer, transform using Kafka consumer and Spark, convert to Parquet format, and store in HDFS.

Solution: In our implementation, we leveraged Kafka's publish-subscribe model and Spark Streaming's real-time processing capabilities to design an efficient data flow for log data processing.

For the Kafka producer, we incorporated data windowing by configuring a batch size of 1000 lines. This allows the producer to efficiently send log data in batches, minimizing network overhead and ensuring smooth data transfer to the Kafka consumer.

Upon consumption by the Kafka consumer, we utilized Spark's powerful transformation capabilities to process and analyze the log data. These transformations enable tasks such as data cleansing, feature extraction, and aggregations, preparing the data for further analysis and insights.

To optimize subsequent processing and querying, we converted the transformed data into the Parquet file format. This format provides columnar storage, compression, and metadata benefits, enhancing data access and minimizing disk I/O.

Finally, we stored the Parquet files in the Hadoop Distributed File System (HDFS). HDFS provides fault-tolerance, scalability, and distributed storage, making it an ideal choice for storing large volumes of log data.

Analysis:

By combining Kafka, Spark Streaming, Parquet files, and HDFS, our solution offers a robust and scalable data flow for log analytics, enabling efficient processing, analysis, and storage of log data.

1. Parquet file format consumes 87% less data than regular text or CSV files. (Reference: Parquet file format: Databricks). Hence cleaning a 17Gb logs took 500MB of storage space after preprocessing the data.
2. Having preprocessed data available in hdfs (or local storage) provides the flexibility of running Kafka Consumer only once, efficiently reusing the data for different analyses, and saving on network bandwidth and memory overhead from Kafka.

Exploratory Data Analysis (EDA)

While processing the data logs, we found that the records are provided for a single day and time for a superficial year (26 December 2118). Let us dive deep into the remaining columns.

1. **Byte Size Statistics**

Table 4 Byte Size Statistics

	min_byte_size	max_byte_size	mean_byte_size	std_byte_size	count_byte_size
0	1	9998	29975.596575	17334.347685	120000

These statistics provide an overview of the distribution and range of values in the "bytes" column. The count indicates the total number of records,

while the mean represents the average value. These statistics help in understanding the data distribution and identifying any outliers or unusual patterns in the "bytes" column.

2. **HTTP Response Code Analysis**
The table displays the counts of different response codes in the dataset. By analyzing the distribution of response codes, we can gain insights into their frequency and identify any patterns or anomalies. In this dataset, all response codes appear with similar frequencies, suggesting a balanced distribution of responses.

Table 5 Response Codes count

response_code	count
200	17284
404	17173
403	17135
500	17133
304	17131
502	17079
303	17065

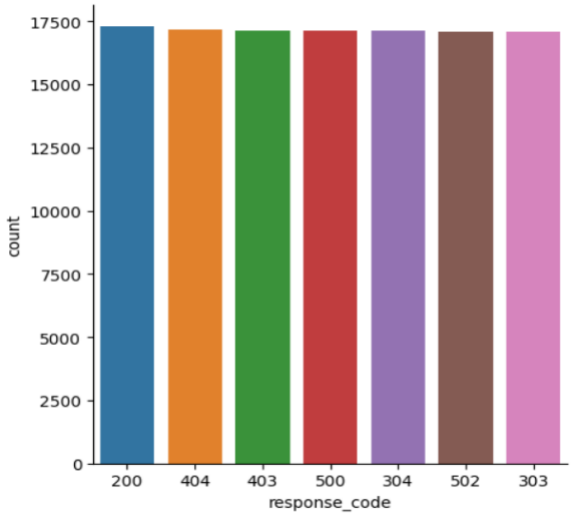

Figure 7 Response Codes count

a. Top ten endpoints with 200 Response Code.

endpoint	count
/Archives/edgar/data/0001168220/000116822009000023/form10k-a_123108.htm	152
/Archives/edgar/data/0001091667/000109166722000024/chtr-20211231.htm	141
/Archives/edgar/data/27673/0001104659-12-037692.txt	141
/Archives/edgar/data/0000004281/000119312517062657/R96.htm	139
/Archives/edgar/data/0000205007/000114554923015369/0001145549-23-015369.txt	137
/Archives/edgar/data/1826671/000121390023023294/0001213900-23-023294.txt	137
/Archives/edgar/data/0001760283/000121390021017155/s131124_10k.htm	136
/Archives/edgar/data/925548/000092153005000094/0000921530-05-000094.txt	135
/Archives/edgar/data/0001260125/000114420419015071/0001144204-19-015071.txt	134
/Archives/edgar/data/0000101382/000156459021009664/umbf-10k_20201231.htm	133

Figure 8 screenshot of top ten endpoints with 200 Response Code

b. Top ten endpoints with 404 Response Codes

endpoint	count
/Archives/edgar/data/0001179929/000117992923000043/moh-20230320_g24.jpg	141
/Archives/edgar/data/1390098/000121390023023295/0001213900-23-023295.txt	139
/Archives/edgar/data/0001002638/000100263815000016/0001002638-15-000016-index.htm	139
/Archives/edgar/data/0001260125/000114420419015093/tv516458_10k.htm	138
/Archives/edgar/data/0001724128/000121390021017151/s131120_10k.htm	138
/Archives/edgar/data/0001798682/000121390021017167/s131143_10k.htm	138
/Archives/edgar/data/27419/0001104659-12-037689.txt	138
/Archives/edgar/data/0000205007/000114554923005581/primary_doc.xml	136
/Archives/edgar/data/0000318833/000035420421000574/SEC13G_Filing.htm	135
/Archives/edgar/data/0000101382/000156459022006546/umbf-10k 20211231.htm	135

Figure 9 screenshot of Top ten endpoints with 404 Response Codes

3. **IP Address Analysis**

ip_address	count
136.107.117.116	2
55.135.36.93	2

Figure 10 screenshot of IP Address count

Among the total of 119998 unique IP addresses in the dataset, there are two IP addresses that appear twice each. These IP addresses stand out as exceptions among the unique addresses, indicating a repetition or duplicate occurrence. It is noteworthy that most IP addresses in the dataset are unique, with only these specific IP addresses showing repetition. Further analysis of these IP addresses and their associated logs may provide insights into any specific patterns or anomalies related to their occurrence.

4. EndPoints Analysis

a. Top 5 Frequent EndPoints

	endpoint	count
0	/Archives/edgar/data/00013275671900032/000132...	863
1	/Archives/edgar/data/0001704304/00009296381900...	861
2	/Archives/edgar/data/0001203957/00011931251022...	853
3	/Archives/edgar/data/1356093/00008843000800001...	853
4	/Archives/edgar/data/0001260125/00011442041901...	852

Figure 11 screenshot of top 5 Frequent EndPoints

b. Top 5 Error Endpoints

endpoint	count
/Archives/edgar/data/0001704304/000092963819000386/form10k.htm	746
/Archives/edgar/data/1356093/000088430008000018/0000884300-08-000018.txt	743
/Archives/edgar/data/00013275671900032/0001327567-19-000032-index.html	739
/Archives/edgar/data/0000318833/000035420421000574/SEC13G_Filing.htm	736
/Archives/edgar/data/0001203957/000119312510223947/0001193125-10-223947-index.htm	734

Figure 12 screenshot of top 5 Error Endpoints

5. Method Analysis

a. Methods frequency

Based on the analysis of the logs, we observe that the methods DELETE, POST, GET, and PUT are called with nearly equal frequency. This indicates a balanced distribution of method invocations in the dataset. The similarity in the number of occurrences suggests that the usage of these methods is fairly consistent and evenly distributed throughout the log entries.

Table 6 Method frequency

method	count
DELETE	30074
POST	30067
GET	29957
PUT	29902

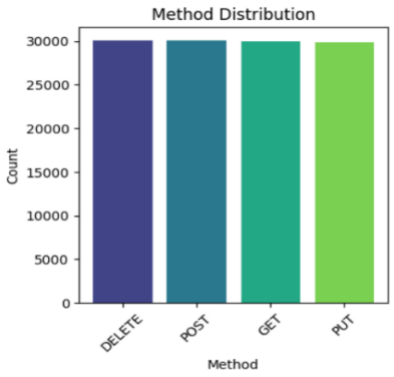

Figure 13 Method frequency

b. Methods to Response Codes

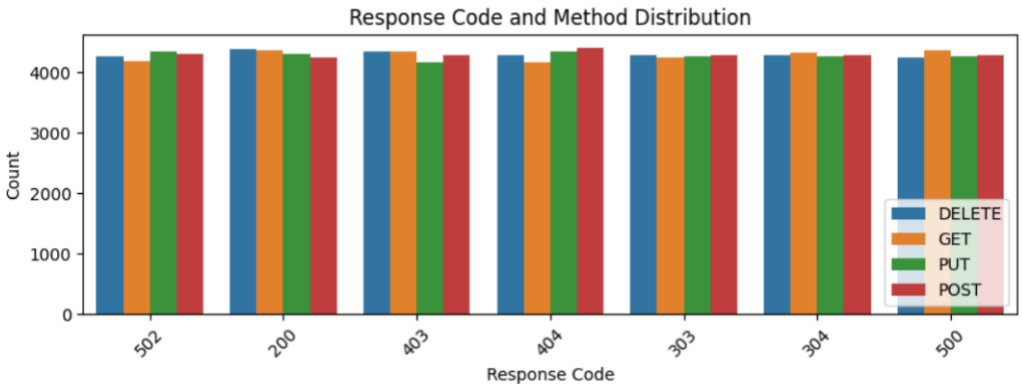

Figure 14: Methods to Response Codes frequency

We can see that All methods are almost equally divided among the Response Codes

c. Methods to Endpoints

Table 7: Methods to Endpoints frequency

endpoint	method	count
/Archives/edgar/data/0001439124/000129281422000788/	GET	198
/Archives/edgar/data/1932485/000110465923037622/tm239175d10_absee.htm	GET	178
/Archives/edgar/data/0001035018/000119312507272779/g47607g69u06.jpg	DELETE	182
/Archives/edgar/data/0001729361/000121390021017176/s131100_10k.htm	GET	211
/Archives/edgar/data/0000205007/000114554923005581/0001145549-23-005581.txt	PUT	190

Section C: Clustering Algorithm for Grouping Requests and Minimizing Clusters While Maintaining Similarity (K-Means Clustering)

Model Training

To train the model by KMeans clustering, it is essential to provide correct factors contributing to meaningful decision-making. We dropped the nonessential columns, such as timestamp and HTTP version, since only one distinct value was provided. The remaining non-integer columns in the parquet file were transformed to index so that the MinMaxScalar function could be applied to complete the data set. Then we calculated the PCA to find the 'elbow' on the five factors from the data set. Then, the Clustering Evaluator library was used to find the optimal number of clusters for the selected two factors. Finally, four optimum clusters were selected, which combined HTTP response codes and the HTTP Request method.

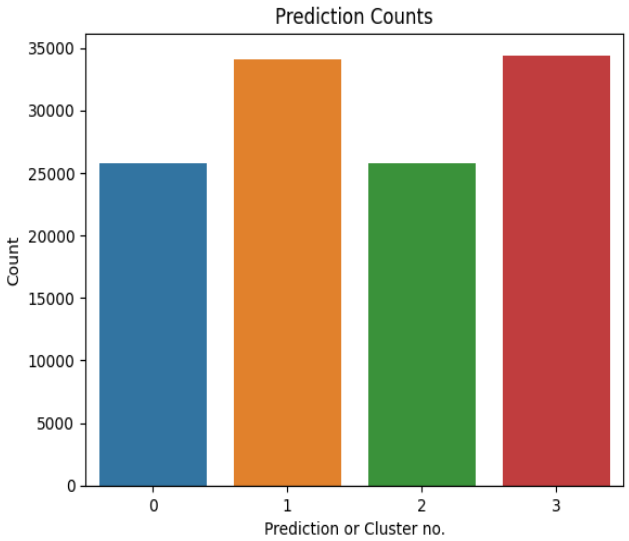

Figure 16 count of clusters.

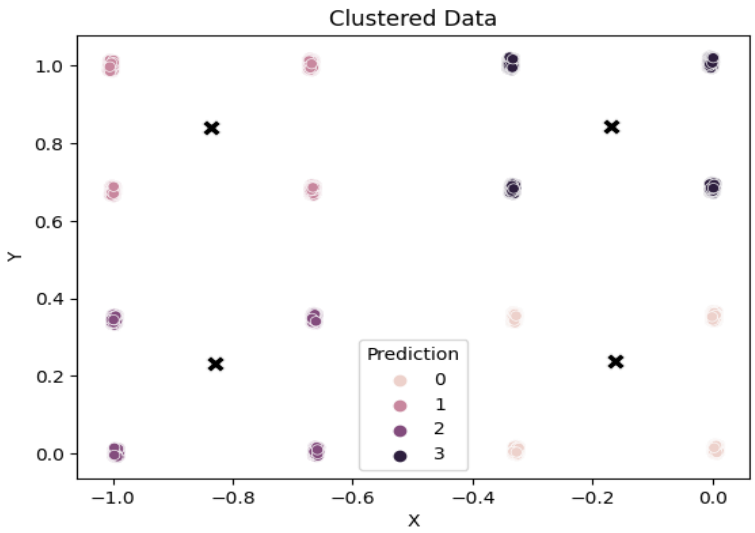

Figure 15 cluster visualization with x = centroids

Analysis:

We successfully predicted the access log into an appropriate cluster by analyzing the Response code and Method distribution graph with the Cluster prediction graph. *Figure 17 and Figure 18 validate the concluded result*. Thus, we are confident any future logs obtained from the Kafka producer will be accurately classified in the designated cluster.

Table 8: Cluster classification using KMeans

Cluster	Response Codes	HTTP Methods
0	successful requests or client-side redirection (304, 303, 200)	POST, DELETE
1	errors or issues with the requested resource or server-side problems (502, 404, 403, 500)	PUT, GET
2	successful requests or client-side redirection (304, 303, 200)	PUT, GET
3	errors or issues with the requested resource or server-side problems (502, 404, 403, 500)	POST, DELETE

From the given information, we can analyze the following:

- Cluster 0 is associated with successful requests or client-side redirection, indicated by response codes 304, 303, and 200. The corresponding HTTP methods for this cluster are POST and DELETE.
- Cluster 1 represents errors or issues with the requested resource or server-side problems. It includes response codes 502, 404, 403, and 500, indicating different types of errors. The HTTP methods associated with this cluster are PUT and GET.
- Cluster 2 is like Cluster 0, as it also represents successful requests or client-side redirection with response codes 304, 303, and 200. The HTTP methods for this cluster are PUT and GET.
- Cluster 3, like Cluster 1, represents errors or issues with the requested resource or server-side problems. It includes response codes 502, 404, 403, and 500. However, the HTTP methods associated with this cluster are POST and DELETE.

By analyzing these clusters, we can gain insights into the patterns and relationships between response codes and HTTP methods, providing a better understanding of the types of requests and their outcomes in the dataset.

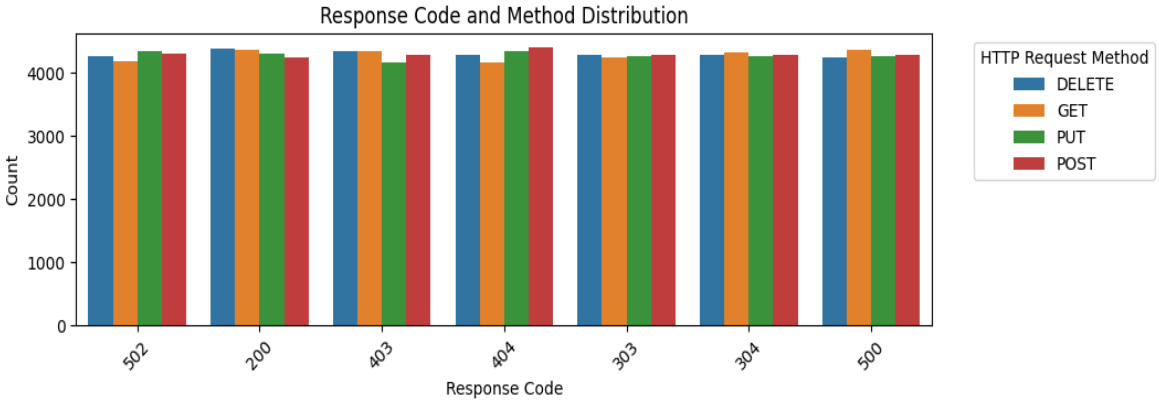

Figure 17: Graph showing actual count of different HTTP request method and HTTP response code for the same.

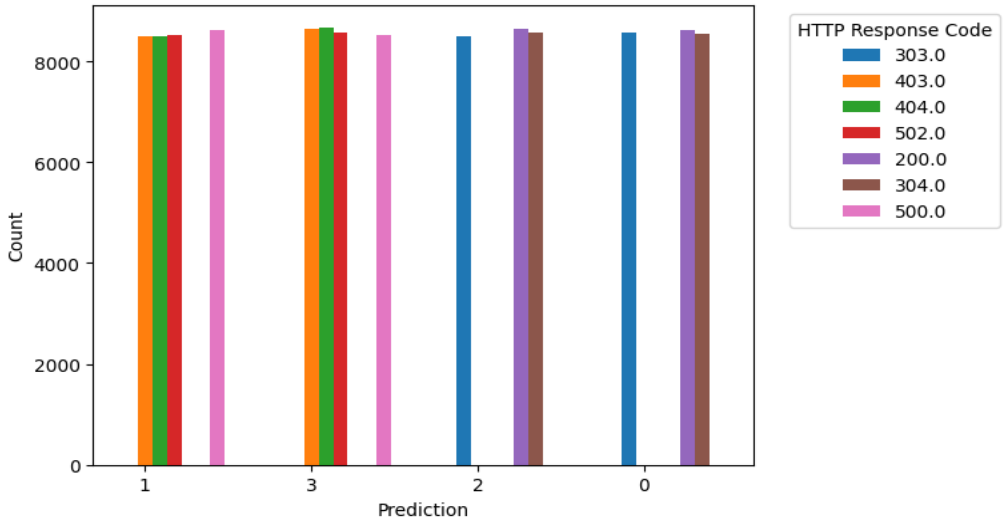

Figure 18: Cluster predicting the response code for the HTTP request methods.

Limitation:

The prediction model achieved a high accuracy level, which closely matched the actual logs. However, it's important to note that this accuracy may be attributed to the fact that the sampling process excluded the log timestamp and HTTP versions. Therefore, the prediction model should be applied cautiously, as it may only be reliable for datasets where the timestamp and HTTP version have a significant impact on the data.