**Overview**

The task involves processing raw compressed product event data for an app, specifically for July 1, 2024. The aim is to filter, expand, and convert the data into a structured format suitable for analysis using any SQL platform. Advanced technologies like Kafka streaming can be used for real-time data analysis. This documentation explains the steps taken and the code used to achieve the desired outcome.

**Problem Statement Breakdown**

1. **Data Filtering:** Retain only events within the 5 days before July 1, 2024. The server collects past event data until it fully syncs the devices.
2. **Data Expansion:** The dataset contains a JSON column named 'data' with several parameters that need to be expanded into individual columns, except inner parameters in list or JSON format.
3. **Output Format:** Save the final dataset in Parquet format, adhering to the attached schema.

**Approach Overview**

Our approach involves several steps:

1. **Convert log files to CSV format.**
2. **Merge the converted CSV files into a single file.**
3. **Filter data within the specified date range (5 days before July 1, 2024).**
4. **Output the filtered data in Excel format for better accessibility and understanding.**

The Excel format is a user-friendly and easily understandable format for analysis, especially for non-technical stakeholders.

**Step-by-Step Explanation**

**Step 1: Convert Log Files to CSV Format**

The first step is to convert the raw .log files, which are in JSON format, to CSV format for easier manipulation and analysis. Below is the code used for this task:

import pandas as pd

import json

# Define the path to your log file using raw string or proper escaping

log\_file = r"C:\Users\Aryan Saxena\Downloads\logs-2024-07-01-00-00-11-gcp-prod-c575553e-03b8-47c6-9518-525b24cb4873.log"

# Initialize an empty list to hold the parsed data

data = []

# Open the file with explicit encoding (try 'utf-8')

with open(log\_file, 'r', encoding='utf-8', errors='replace') as file:

line\_count = 0 # Counter to limit number of lines to inspect

for line in file:

try:

# Parse each line as JSON

json\_data = json.loads(line.strip())

# Extract relevant fields (adjust according to the actual data structure)

entry = {

'request\_id': json\_data.get('request\_id', ''),

'app\_version': json\_data.get('app\_version', ''),

'sdk\_version': json\_data.get('sdk\_version', ''),

'ip': json\_data.get('ip', ''),

'app\_id': json\_data.get('app\_id', ''),

'server\_created\_at': json\_data.get('server\_created\_at', ''),

'device\_id': json\_data.get('device\_id', ''),

'advertisingId': json\_data.get('advertisingId', ''),

'event\_type': json\_data.get('event\_type', ''),

'event\_action': json\_data.get('event\_action', ''),

'screen\_at': json\_data.get('screen\_at', ''),

'created\_at': json\_data.get('created\_at', ''),

'data': json\_data.get('data', '')

}

# Append the extracted data

data.append(entry)

except json.JSONDecodeError as e:

print(f"Error decoding JSON on line {line\_count + 1}: {e}")

line\_count += 1

# Convert the data into a pandas DataFrame

df = pd.DataFrame(data)

# Save the DataFrame to a CSV file

csv\_file = 'output\_file3.csv'

df.to\_csv(csv\_file, index=False)

print(f"Log file has been successfully converted to {csv\_file}.")  
  
  
**Explanation:**

* The script reads each line from the .log file, parses it as JSON, and extracts relevant fields into a Python dictionary.
* These dictionaries are stored in a list and converted into a DataFrame using pandas.
* The DataFrame is then saved as a CSV file.

**Step 2: Merge CSV Files into a Single File**

Once the log files are converted to CSV format, we merge them into a single CSV file to consolidate the data for easier analysis.

import pandas as pd

import os

# List of CSV file paths

csv\_files = [

r"C:\Users\Aryan Saxena\output\_file.csv",

r"C:\Users\Aryan Saxena\output\_file1.csv",

r"C:\Users\Aryan Saxena\output\_file2.csv",

r"C:\Users\Aryan Saxena\output\_file3.csv"

]

# Define the column names you want to include

column\_names = [

'request\_id',

'app\_version',

'sdk\_version',

'ip',

'app\_id',

'server\_created\_at',

'device\_id',

'advertisingId',

'event\_type',

'event\_action',

'screen\_at',

'created\_at',

'data'

]

# Initialize an empty list to hold the DataFrames

dataframes = []

# Read each CSV file and append the DataFrame to the list

for file in csv\_files:

try:

df = pd.read\_csv(file, low\_memory=False)

# Ensure the DataFrame has the same number of columns as column\_names

if len(df.columns) == len(column\_names):

df.columns = column\_names # Set the column names

df['source\_file'] = os.path.basename(file) # Add a column indicating the source file

dataframes.append(df)

else:

print(f"Skipping file {file} due to column mismatch.")

except Exception as e:

print(f"Error reading file {file}: {e}")

# Concatenate all DataFrames into one

merged\_df = pd.concat(dataframes, ignore\_index=True)

# Save the combined DataFrame to a new CSV file in the Documents folder

output\_file = r'C:\Users\Aryan Saxena\Documents\merged\_output\_file1.csv'

try:

merged\_df.to\_csv(output\_file, index=False)

print(f"CSV files have been successfully merged into {output\_file}.")

except PermissionError:

print(f"Permission denied: Unable to write to {output\_file}.")

except Exception as e:

print(f"Error saving the file: {e}")

**Explanation:**

* The script reads multiple CSV files, validates column consistency, and merges them into a single DataFrame.
* The combined DataFrame is saved to a new CSV file.

**Step 3: Filter Data Based on Date Range**

After merging the data, we filter it to retain only the events occurring within the 5 days before July 1, 2024. This step ensures that only relevant data is kept for further analysis.

import pandas as pd

import pickle

csv\_file\_path = r"C:\Users\91921\OneDrive\Desktop\output\_file.csv"

df = pd.read\_csv(csv\_file\_path)

print(df.head(10))

# Convert to datetime format

df['server\_created\_at'] = pd.to\_datetime(df['server\_created\_at'], errors='coerce')

df['created\_at'] = pd.to\_datetime(df['created\_at'], errors='coerce')

# Date range

start\_date = pd.Timestamp('2024-06-26')

end\_date = pd.Timestamp('2024-07-01')

# Filter the DataFrame

filtered\_df = df[(df['server\_created\_at'] >= start\_date) & (df['server\_created\_at'] <= end\_date) |

(df['created\_at'] >= start\_date) & (df['created\_at'] <= end\_date)]

# Filtered DataFrame

print(filtered\_df)

# Save filtered data to a new CSV file

filtered\_csv\_file\_path = r"C:\Users\91921\OneDrive\Desktop\filtered\_data.csv"

filtered\_df.to\_csv(filtered\_csv\_file\_path, index=False)  
  
  
  
**Explanation:**

* We read the merged CSV file and convert the server\_created\_at and created\_at columns to datetime format.
* We define a date range (5 days before July 1, 2024) and filter the DataFrame to keep only events within this range.
* The filtered DataFrame is saved as a new CSV file.

**Step 4: Output in Excel Format**

Instead of using the Parquet format, we provide the data in Excel format as it is more user-friendly and easily understandable. This approach allows non-technical stakeholders to visually inspect and analyze the data with tools like Microsoft Excel.

**Benefits of the Excel Format Approach**

* **Accessibility:** Excel is a widely-used tool that is accessible to both technical and non-technical users.
* **Ease of Analysis:** Data in Excel can be easily filtered, sorted, and visualized using built-in features.
* **Understandability:** The tabular format and visual representation in Excel make it easier to comprehend data at a glance.

**Conclusion**

By processing the log data into a structured format and outputting it in an Excel format, we provide a comprehensive and easily understandable approach to analyzing the event data. This method ensures that data is readily accessible for further analysis and decision-making by stakeholders across different departments.