1. File Handling Approach

Steps Involved:

- File Upload:
  - The script uses Google Colab's files.upload() method to allow the user to upload a log file interactively.
  - The uploaded file's name is extracted to process its content.

Python

```
uploaded = files.upload()
log_file_path = list(uploaded.keys())[0]
```

- File Reading:
  - The parse_log_file() function reads the entire content of the log file line by line into a list of strings.
  - The use of the with statement ensures the file is properly closed after reading.

Python

```
with open(file_path, 'r') as file:
    return file.readlines()
```

- File Writing:
  - Results are saved to a CSV file using Python's csv module.
  - Structured data is written into separate sections (Requests per IP, Most Accessed Endpoint, Suspicious Activity).

python

```
with open(output_file, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Requests per IP"])
    writer.writerows(requests_per_ip)
```

- File Download:
  - The processed results are made downloadable in Colab using files.download().

```
    files.download(output_file_path)
```

2. String Manipulation Approach

Key Techniques:

The script uses Python's re (regular expressions) library extensively to parse log entries. This approach is efficient for extracting specific patterns from unstructured text logs.

- Extracting IP Addresses:
  - Matches standard IPv4 format using the regex r"(\d+\.\d+\.\d+\.\d+)".
  - Applies re.match() to extract IP addresses at the start of each log line.

python

```
ip_match = re.match(r"(\d+\.\d+\.\d+\.\d+)", log)
```

- Extracting HTTP Methods and Endpoints:
  - Matches HTTP requests (e.g., GET, POST) and their associated endpoints using the regex r"\"(?:GET|POST|PUT|DELETE) (\/\S+)".
  - Uses re.search() to find the pattern anywhere in the log line.

python

```
endpoint_match =
re.search(r"\"(?:GET|POST|PUT|DELETE) (\/\S+)",
log)
```

- Detecting Failed Logins:
  - Searches for specific error indicators like 401 or the phrase "Invalid credentials" to identify failed login attempts.

Python

```
if "401" in log or "Invalid credentials" in log:
```

3. Data Analysis Approach

Techniques Used:

The script employs dictionary-based counting and filtering for data analysis, ensuring clarity and scalability.

- Counting Requests Per IP:
    - A defaultdict is used to count occurrences of IP addresses. This simplifies the code by eliminating the need to check if a key exists before incrementing its value.
    - The results are sorted in descending order of requests.

python

```
ip_counts = defaultdict(int)
ip_counts[ip] += 1
```

- Finding the Most Accessed Endpoint:
    - Similar to IP counting, a defaultdict is used to count accesses to endpoints.
    - The max() function determines the endpoint with the highest count.

Python

```
endpoint_counts = defaultdict(int)
endpoint_counts[endpoint] += 1
```

- Flagging Suspicious Activity:
  - Another defaultdict tracks failed login attempts per IP.
  - Filtering is applied to return only those IPs whose counts exceed the specified threshold.

python

```
failed_attempts = defaultdict(int)
return {ip: count for ip, count in
failed_attempts.items() if count > threshold}
```

- Data Presentation:
  - Analysis results are displayed in a human-readable format in the console and saved in structured sections within a CSV file.

4. Overall Workflow

Integration of File Handling, String Manipulation, and Data Analysis:

1. File Handling:
   - The raw log file is read and passed as input to various analysis functions.
2. String Manipulation:
   - Extracts structured data (IPs, endpoints, errors) from unstructured log entries using regex.
3. Data Analysis:
   - Aggregates the extracted data into meaningful insights like:
     - IP request frequencies.
     - Most accessed endpoint.
     - Suspicious activities.
4. Result Output:
   - Analysis results are displayed on-screen and written to a downloadable CSV file.