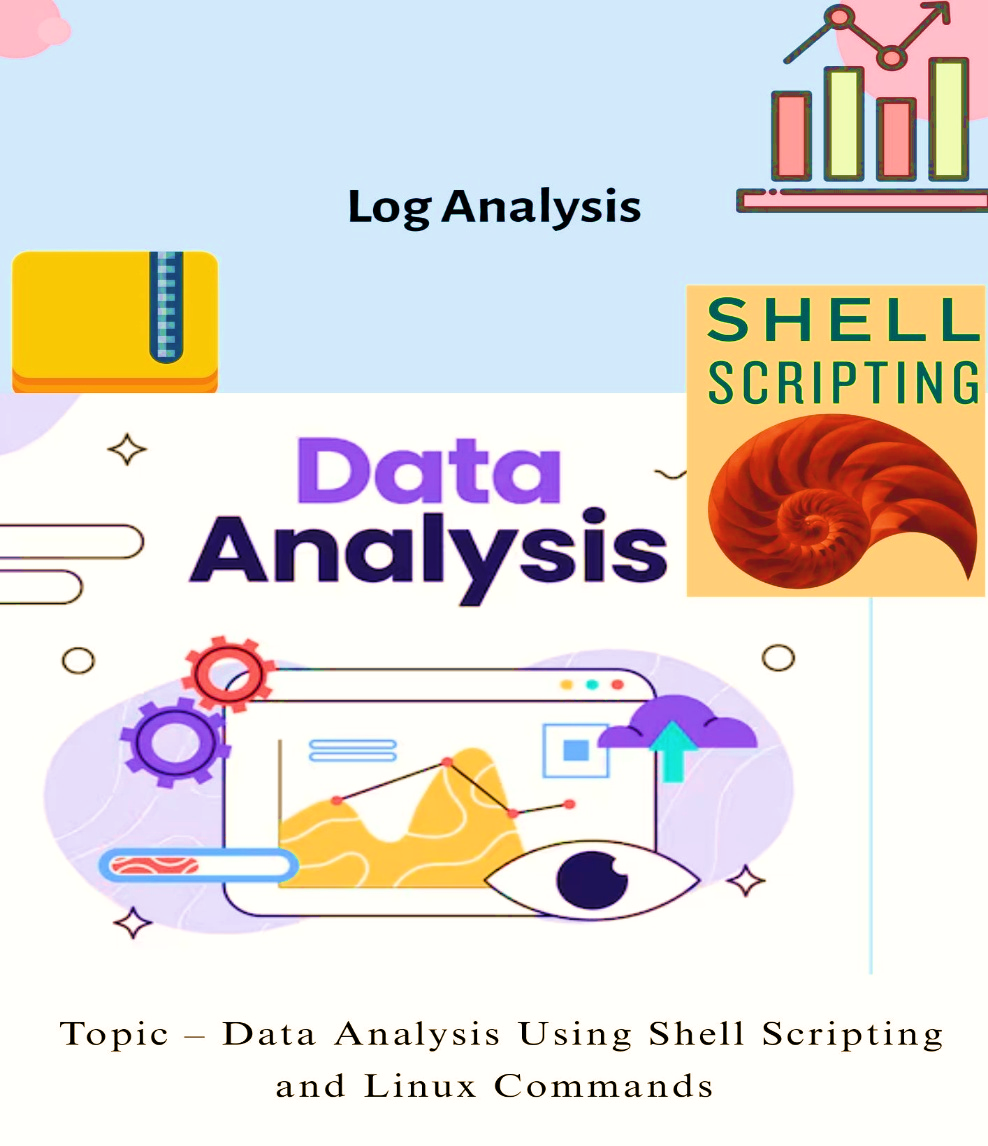
**MOS CIA – 1 ASSIGNMENT**





#### **Introduction -**

Web servers continuously generate log files that record every request made to the server. These logs, while rich in information, are typically unstructured and not directly ready for analysis. They contain various data points such as IP addresses, timestamps, requested URLs, HTTP methods, status codes, and user agent strings.

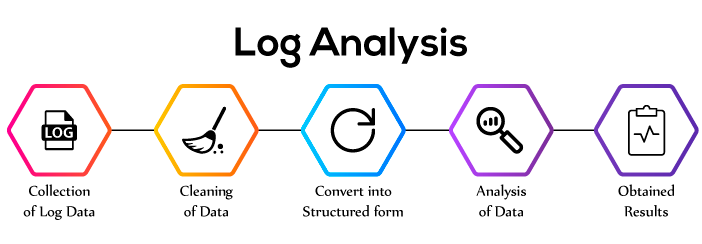
This report presents a systematic approach to extracting, cleaning, and analyzing such unstructured access log data using only standard Linux command-line tools and shell scripting. The analysis emphasizes lightweight, reproducible methods without relying on external programming languages or frameworks.

***Problem Statement –***

Access logs contain valuable information but are often unstructured and difficult to analyse directly. Without proper tools, extracting insights like usage trends, frequent errors, or top users becomes time-consuming and inefficient.

#### **Objective -**

Choose a file that contains unstructured data. Our task is to process, clean, and analyse this data **using only shell scripts and standard Linux command-line utilities** (e.g., awk, sed, grep, cut, sort, uniq, wc, etc.).

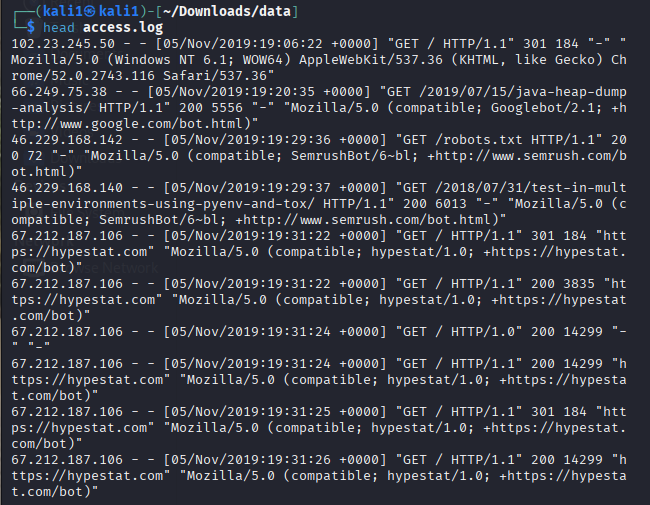


#### **Methodology -**

The analysis was conducted entirely using shell scripting and standard Linux command-line utilities. The process included data inspection, cleaning, transformation, and preparation for visualization. Below is a step-by-step breakdown of the methodology used:

1. **Initial Data Inspection**

* Using command head to see the first few lines from the file or dataset. By default, we can see first ten records stored.



* Initial Observations –

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No.** | **Field** | **Value** | **What it Means** |
| 1 | IP Address | 102.23.245.50 | The visitor's device address |
| 2 | Placeholder | - | No authenticated user |
| 3 | Placeholder | - | No username |
| 4 | Date-Time | [05/Nov/2019:19:06:22 +0000] | Accessed on 5th Nov 2019 at 7:06 PM |
| 5 | Request | "GET / HTTP/1.1" | Visitor asked to load the homepage (/) |
| 6 | Status Code | 301 | Redirect – the page moved somewhere else |
| 7 | Bytes Sent | 184 | Only 184 bytes sent in the response |
| 8 | Referrer | "-" | No referrer – they came directly |
| 9 | User Agent | "Mozilla/5.0 ... Chrome/52.0.2743.116" | They used Chrome on Windows 7 (64-bit) |

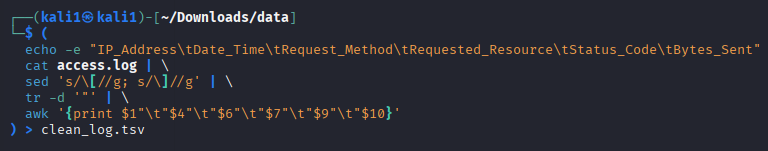
1. **Data Cleaning and Transformation**

**So far, we have seen our unstructured data contains loads of information like visitor IP, access time, bytes sent and so on. But we don’t need so much information, we rather require a clean data that we can analyse and work on.**

**Therefore, information we want to extract –**

|  |  |
| --- | --- |
| **Field** | **Info to Extract** |
| 1 | IP address |
| 2 | Date & time |
| 3 | Request method |
| 4 | Page requested |
| 5 | Status code |
| 6 | Bytes sent |

* In Bash –



* What does it do?

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Command** | **Purpose** | **Key Output/Effect** |
| [1] | echo -e "IP\_Address\tDate\_Time\tRequest\_Method\tRequested\_Resource\tStatus\_Code\tBytes\_Sent" | Adds a header row with tab-separated columns | Column names for the .tsv file |
| [2] | cat access.log | Reads and passes log data | Sends each line of the log to the next tool |
| [3] | sed 's/\[//g; s/\]//g' | Removes square brackets from timestamps | Cleans up [date] → date |
| [4] | tr -d '"' | Removes double quotes from request fields | "GET /..." → GET /... |
| [5] | awk '{print $1"\t"$4"\t"$6"\t"$7"\t"$9"\t"$10}' | Extracts key fields and formats them as tab-separated | IP, Date-Time, Method, Resource, Status, Bytes |
| [6] | > clean\_log.tsv | Saves the result to a new file | Output written to clean\_log.tsv |

1. **Data Aggregation & Analysis Techniques**

The following shell utilities were repeatedly used to generate summary data:

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| cut -fX | Extract specific column (field) from TSV |
| tail -n +2 | Skip header row for analysis |
| sort + uniq -c | Count and rank frequency of IPs, methods, etc. |
| awk '{sum+=$1} END {print sum}' | Sum numeric fields like bytes |
| awk '{sum+=$1} END {print sum/NR}' | Compute averages |

1. **Visualization with Gnuplot**

Each insight was visualized using Gnuplot, with the data first saved to .tsv files:

**Chart types used:**

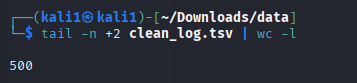
* **Bar Charts** (e.g., top IPs, error-causing IPs)
* **Pie Charts** (e.g., top pages, daily requests)
* **Donut Charts** (e.g., request methods, large responses)
* **Line Charts** (e.g., requests per day)
* **Gauge Charts** (e.g., average bytes per response)

1. **Summary of Tools Used**

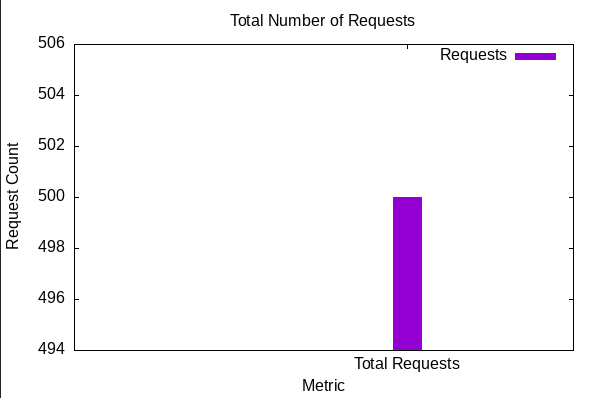
|  |  |
| --- | --- |
| **Category** | **Tools / Commands** |
| File inspection | head, cat, tail |
| Cleaning | sed, tr, awk |
| Analysis | cut, sort, uniq, wc, grep |
| Aggregation | awk (for sums, averages), uniq -c |
| Visualization | gnuplot |
| Scripting | bash, echo, file redirection (>, >>) |

#### **Insights into Analysis and Graphs -**

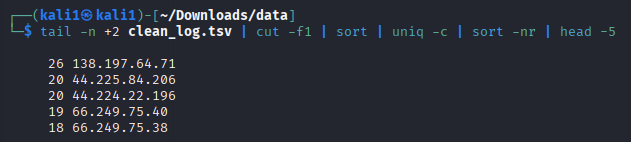
1. ***Total number of requests –***



* tail -n +2: skips the header
* wc -l: counts the lines (i.e., the requests)
* ***Interpretation - There are total of 500 requests made***
* ***Visualization –***



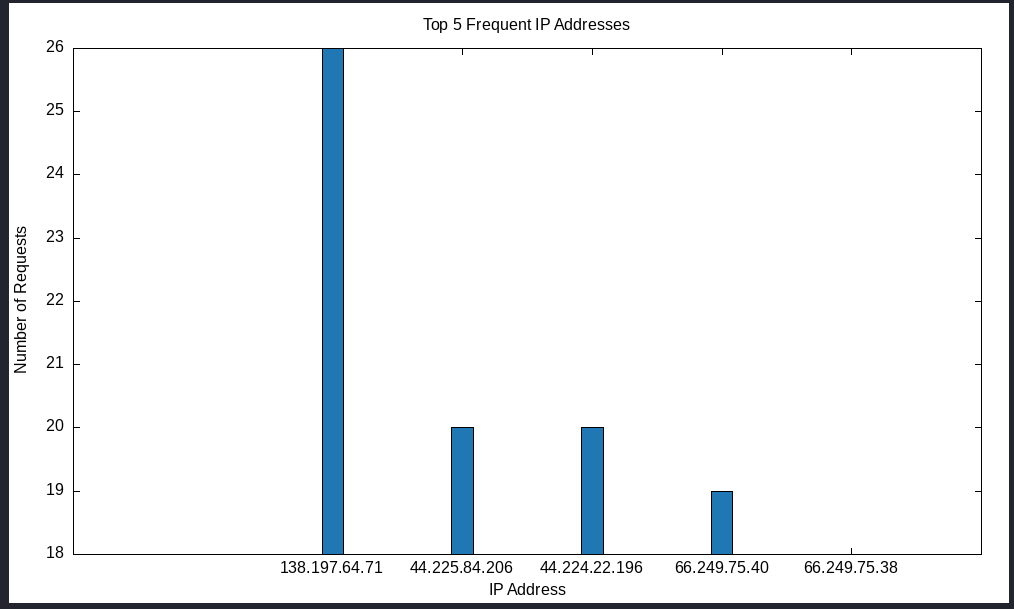
1. ***Top 5 Most Frequent IP Addresses –***



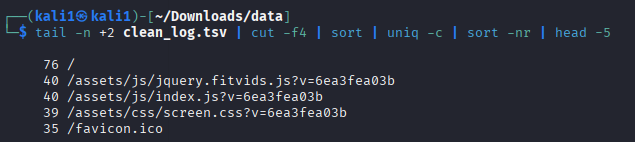
* cut -f1: extracts the IP address
* uniq -c: counts occurrences
* sort -nr: sorts by frequency
* ***Interpretation –***

|  |  |  |
| --- | --- | --- |
| **IP Address** | **Request Count** | **Possible Insight** |
| 138.197.64.71 | 26 requests | Likely a user or bot that accessed the server repeatedly |
| 44.225.84.206 | 20 requests | Possibly an AWS server or scraper |
| 44.224.22.196 | 20 requests | Another client from AWS IP range |
| 66.249.75.40 | 19 requests | Known to be a **Googlebot** IP |
| 66.249.75.38 | 18 requests | Another **Googlebot** address |

* ***Visualization –***



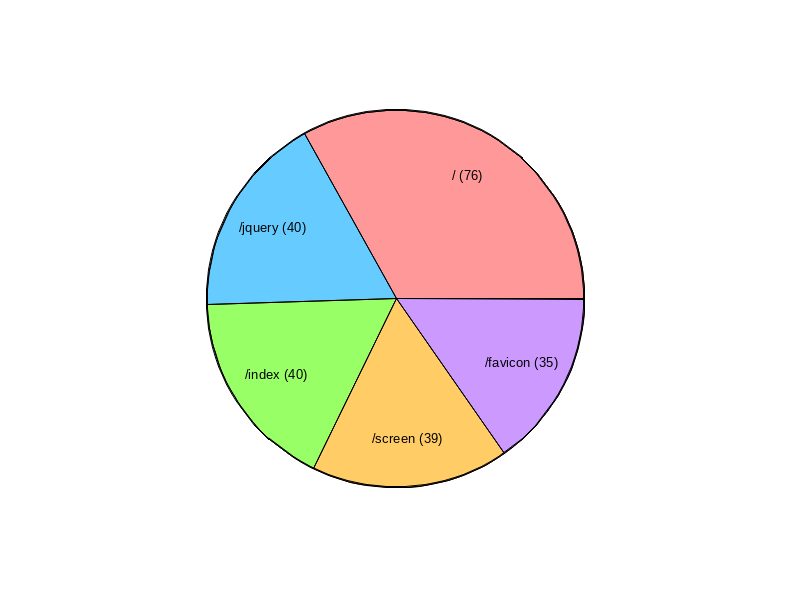
1. ***Top 5 Most Accessed Pages –***



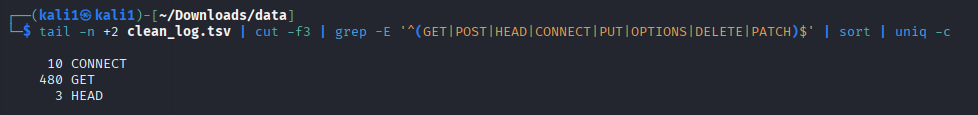
* cut -f4: extracts the resource path
* Counts how often each page is accessed
* ***Interpretation –***

|  |  |  |  |
| --- | --- | --- | --- |
| **Rank** | **Resource Path** | **Description** | **Requests** |
| 1️ | / | Homepage (root of the website) | 76 |
| 2️ | /assets/js/jquery.fitvids.js?... | JavaScript file for video responsiveness | 40 |
| 3️ | /assets/js/index.js?... | Main JavaScript logic file | 40 |
| 4️ | /assets/css/screen.css?... | Website styling (CSS) file | 39 |
| 5️ | /favicon.ico | Browser tab icon | 35 |

* ***Visualization –***



1. ***Requests by HTTP Method***

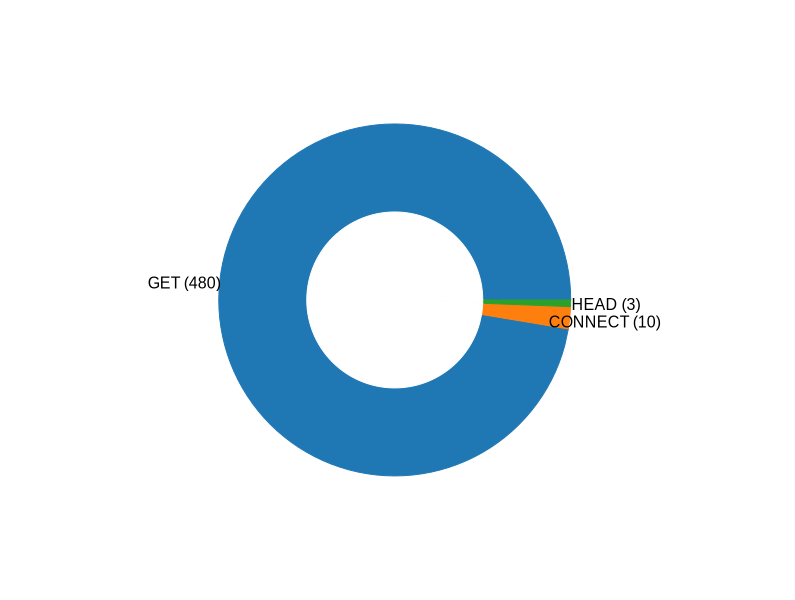


|  |  |
| --- | --- |
| **Step** | **What it does** |
| tail -n +2 clean\_log.tsv | Skips the header line from the .tsv file |
| cut -f3 | Selects the **third column**, which contains the **HTTP request method** |
| `grep -E '^(GET | POST |
| sort | Sorts the methods alphabetically |
| uniq -c | Counts the number of times each unique method occurs |

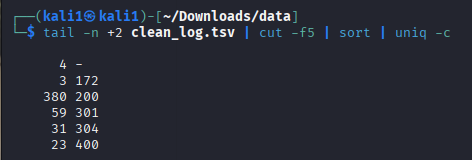
* ***Interpretation –***

|  |  |  |
| --- | --- | --- |
| **Method** | **Count** | **Meaning** |
| **GET** | 480 | Most common — used to **fetch** a webpage or resource |
| **CONNECT** | 10 | Typically used to **initiate HTTPS tunnels** via proxy |
| **HEAD** | 3 | Requests only **headers**, not the full content (used by bots or to check if a resource has changed) |

* ***Visualization –***



1. ***Status Code Distribution (200, 404, etc.)***

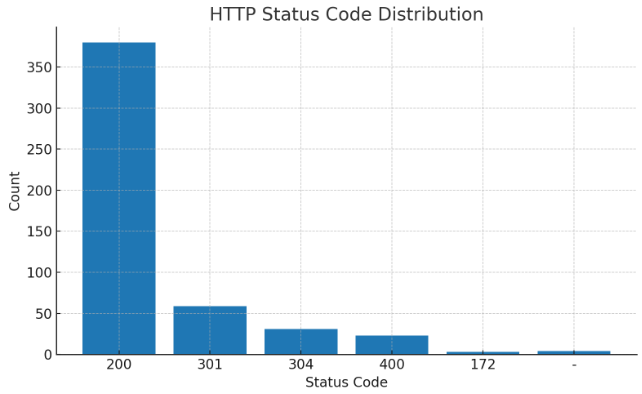


|  |  |
| --- | --- |
| **Step** | **Function** |
| tail -n +2 clean\_log.tsv | Skips the first (header) line of the log file |
| cut -f5 | Extracts the **Status Code** column (e.g., 200, 404, etc.) |
| sort | Sorts the status codes so that uniq can group them |
| uniq -c | Counts how many times each unique status code appears |

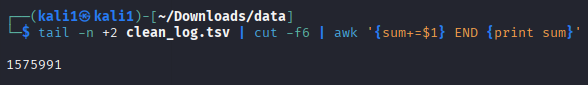
* ***Interpretation –***

|  |  |  |
| --- | --- | --- |
| **Count** | **Status Code** | **Meaning** |
| 380 | 200 | OK — Request was successful |
| 59 | 301 | Moved Permanently (redirect) |
| 31 | 304 | Not Modified (cached) |
| 23 | 400 | Bad Request (client error) |
| 3 | 172 | Unknown/Unusual code (non-standard) |
| 4 | - | Possibly malformed or missing data |

* ***Visualization –***

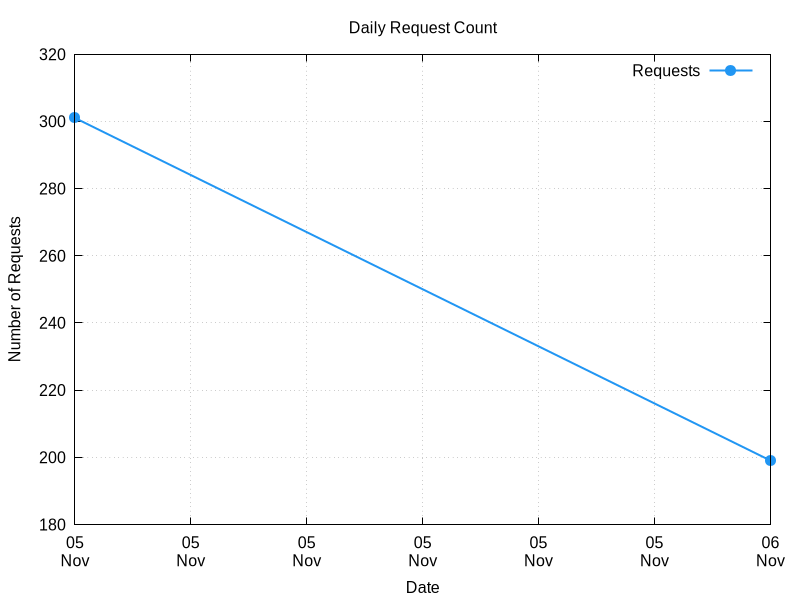


1. ***Total Bandwidth Used***

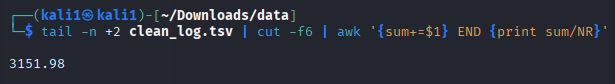


|  |  |
| --- | --- |
| **Part** | **What It Does** |
| tail -n +2 clean\_log.tsv | Skips the header (first line) |
| cut -f6 | Extracts the **6th column**, which is the number of **bytes sent** in the server response |
| awk '{sum+=$1} END {print sum}' | Adds up all the bytes from every line to give the **total bandwidth used** |

* ***Interpretation – A total of 1,575,991 bytes*** *were sent in all the requests combined.*
* ***Visualization –***

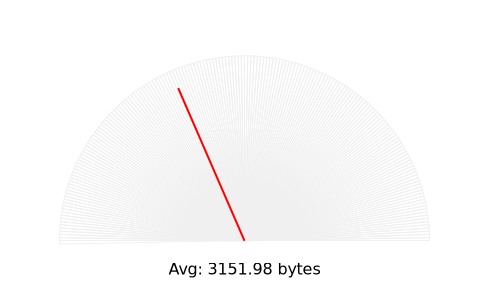


1. ***Average Response Size***

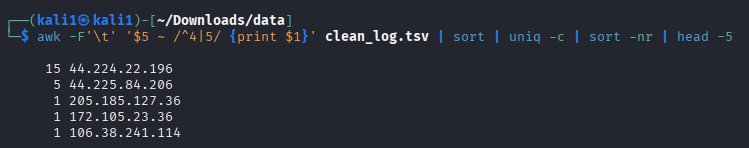


|  |  |
| --- | --- |
| **Part** | **Meaning** |
| tail -n +2 clean\_log.tsv | Skips the header line (so you're only analyzing actual data) |
| cut -f6 | Extracts the **6th column**, which represents the number of **bytes sent** in response to each request |
| awk '{sum+=$1} END {print sum/NR}' | Adds all byte values and divides by NR (number of records) to calculate the **average bytes per request** |

* ***Interpretation – On average, each HTTP response was about 3,151.98 bytes in size.***
* ***Visualization –***



1. ***IPs Causing Most Errors (status ≥ 400)***

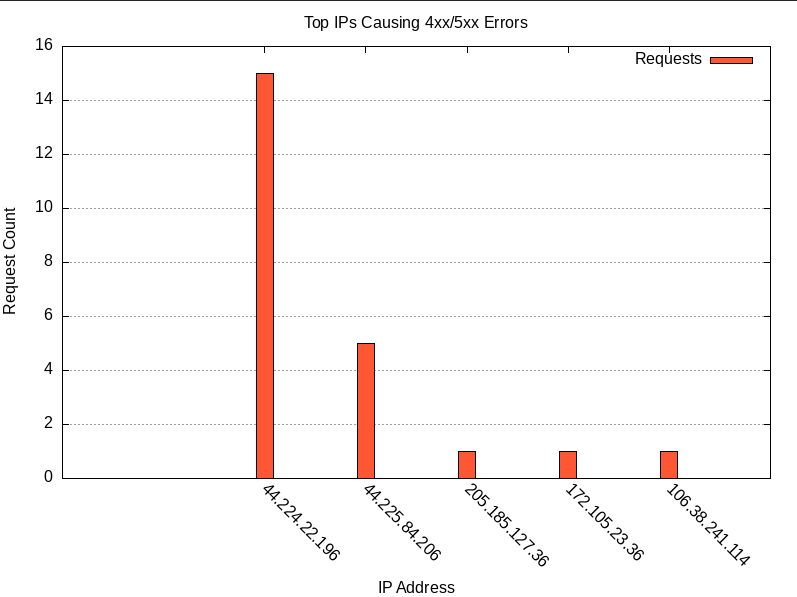


|  |  |
| --- | --- |
| **Part** | **What It Does** |
| `awk -F'\t' '$5 ~ /^4 | 5/ {print $1}' clean\_log.tsv` |
| `sort | uniq -c` |
| `sort -nr | head -5` |

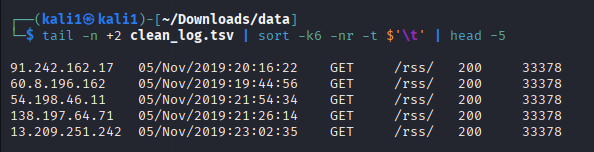
* ***Interpretation –***

|  |  |  |  |
| --- | --- | --- | --- |
| **Rank** | **IP Address** | **Number of Errors** | **Type of Errors** |
| 1 | 44.224.22.196 | 15 | 4xx or 5xx |
| 2 | 44.225.84.206 | 5 | 4xx or 5xx |
| 3 | 205.185.127.36 | 1 | 4xx or 5xx |
| 4 | 172.105.23.36 | 1 | 4xx or 5xx |
| 5 | 106.38.241.114 | 1 | 4xx or 5xx |

* ***Visualization –***



1. ***Top 5 Largest Responses (by Byte Size)***

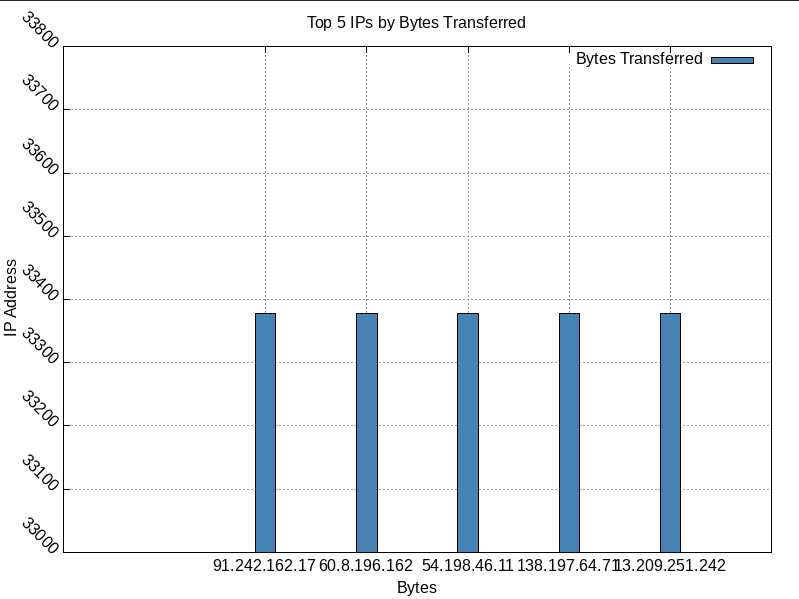


|  |  |
| --- | --- |
| **Part** | **What It Does** |
| tail -n +2 clean\_log.tsv | Skips the header row |
| sort -k6 -nr -t $'\t' | Sorts by **6th column** (i.e., Bytes\_Sent) in **numeric reverse order** using tab \t as the field separator |
| head -5 | Displays only the top 5 results |

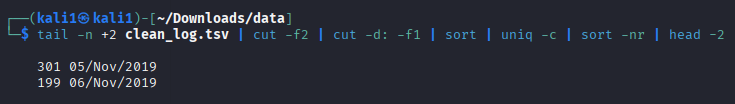
* ***Interpretation –***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **IP Address** | **Date & Time** | **Method** | **Resource** | **Status** | **Bytes Sent** |
| 91.242.162.17 | 05/Nov/2019:20:16:22 | GET | /rss/ | 200 | 33378 |
| 60.8.196.162 | 05/Nov/2019:19:44:56 | GET | /rss/ | 200 | 33378 |
| 54.198.46.11 | 05/Nov/2019:21:54:34 | GET | /rss/ | 200 | 33378 |
| 138.197.64.71 | 05/Nov/2019:21:26:14 | GET | /rss/ | 200 | 33378 |
| 13.209.251.242 | 05/Nov/2019:23:02:35 | GET | /rss/ | 200 | 33378 |

* ***Visualization –***



1. ***Top 2 Dates with Most Requests***

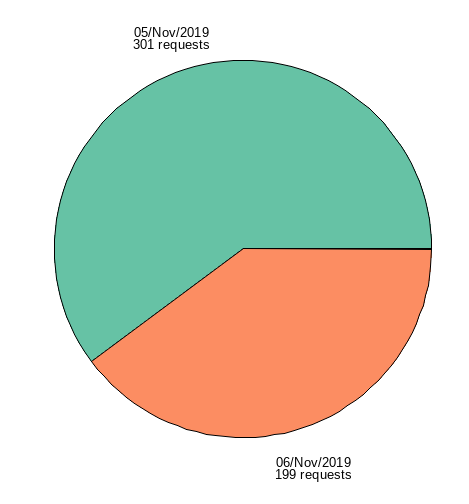


|  |  |
| --- | --- |
| **Part** | **What It Does** |
| tail -n +2 clean\_log.tsv | Skips the first row (header) to only work with real data |
| cut -f2 | Selects the second column (Date\_Time) |
| cut -d: -f1 | Splits the Date\_Time on : and keeps just the **date** (e.g., 05/Nov/2019) |
| sort | Sorts all dates (to group similar ones together) |
| uniq -c | Counts how many times each date appears (i.e., number of requests per day) |
| sort -nr | Sorts the counts in descending order |
| head -2 | Shows the top 2 dates with the highest request counts |

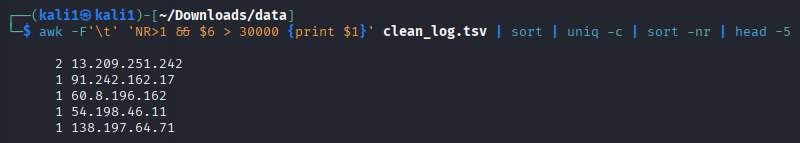
* ***Interpretation –***

|  |  |
| --- | --- |
| **Date** | **Number of Requests** |
| 05/Nov/2019 | 301 |
| 06/Nov/2019 | 199 |

* ***Visualization –***



1. ***Detect IPs Requesting Large Responses***

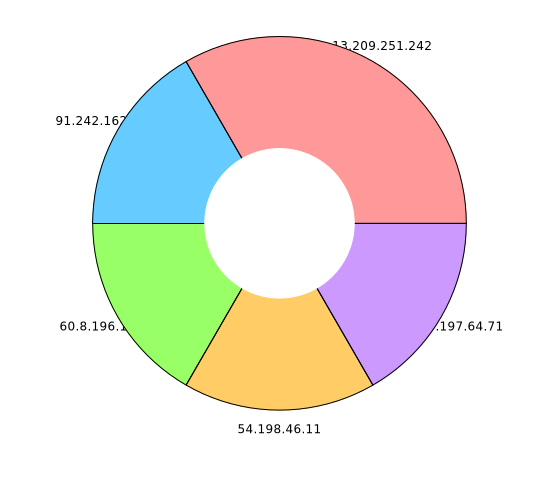


|  |  |
| --- | --- |
| **Part** | **Meaning** |
| awk -F'\t' | Uses tab (\t) as the field separator because your file is tab-separated |
| NR>1 | Skips the header row (starts from row 2) |
| $6 > 30000 | Only considers rows where the **Bytes\_Sent (column 6)** is greater than **30,000** (a large response) |
| {print $1} | Prints the **IP address (column 1)** that requested large responses |
| `sort | uniq -c` |
| `sort -nr | head -5` |

* ***Interpretation –***

|  |  |
| --- | --- |
| **IP Address** | **Large Responses (Bytes > 30,000)** |
| 13.209.251.242 | 2 |
| 91.242.162.17 | 1 |
| 60.8.196.162 | 1 |
| 54.198.46.11 | 1 |
| 138.197.64.71 | 1 |

* ***Visualization –***



***Conclusion –***

This assignment successfully demonstrated the complete lifecycle of working with unstructured log data using only shell scripting and Linux command-line tools. Starting from raw log data, we cleaned and structured it into a tab-separated format that could be easily processed and visualized. Using tools like awk, cut, sort, uniq, and wc, we extracted meaningful insights such as:

* Total requests
* Most frequent IPs
* Top accessed pages
* HTTP method distributions
* Status code breakdown
* Largest responses
* Bandwidth usage

Visualization was achieved using Gnuplot with various chart types like **bar graphs**, **line graphs**, **donut charts**, and **pie charts**. These helped turn raw numbers into intuitive graphical summaries.

Through this exercise, we not only explored the power of shell scripting for data pre-processing but also showcased how meaningful trends can be discovered from raw logs without using heavy analytics frameworks.

***Scope of Improvement –***

1. **Automate the Workflow**  
   Create a single shell script that performs all analysis and visualization steps automatically.
2. **Add Error Handling**  
   Include checks to skip or log malformed rows to avoid processing interruptions.
3. **Hourly Traffic Analysis**  
   Extend the script to visualize traffic trends over each hour to identify peak activity times.
4. **Interactive Dashboards**  
   Export data to CSV and use tools like LibreOffice Calc or Google Sheets for clickable and filterable charts.
5. **Geo-IP Lookup for IPs**  
   Use tools like geoiplookup to map IP addresses to countries for location-based insights.

***Appendix of shell code –*** Uploaded on GitHub classroom.

