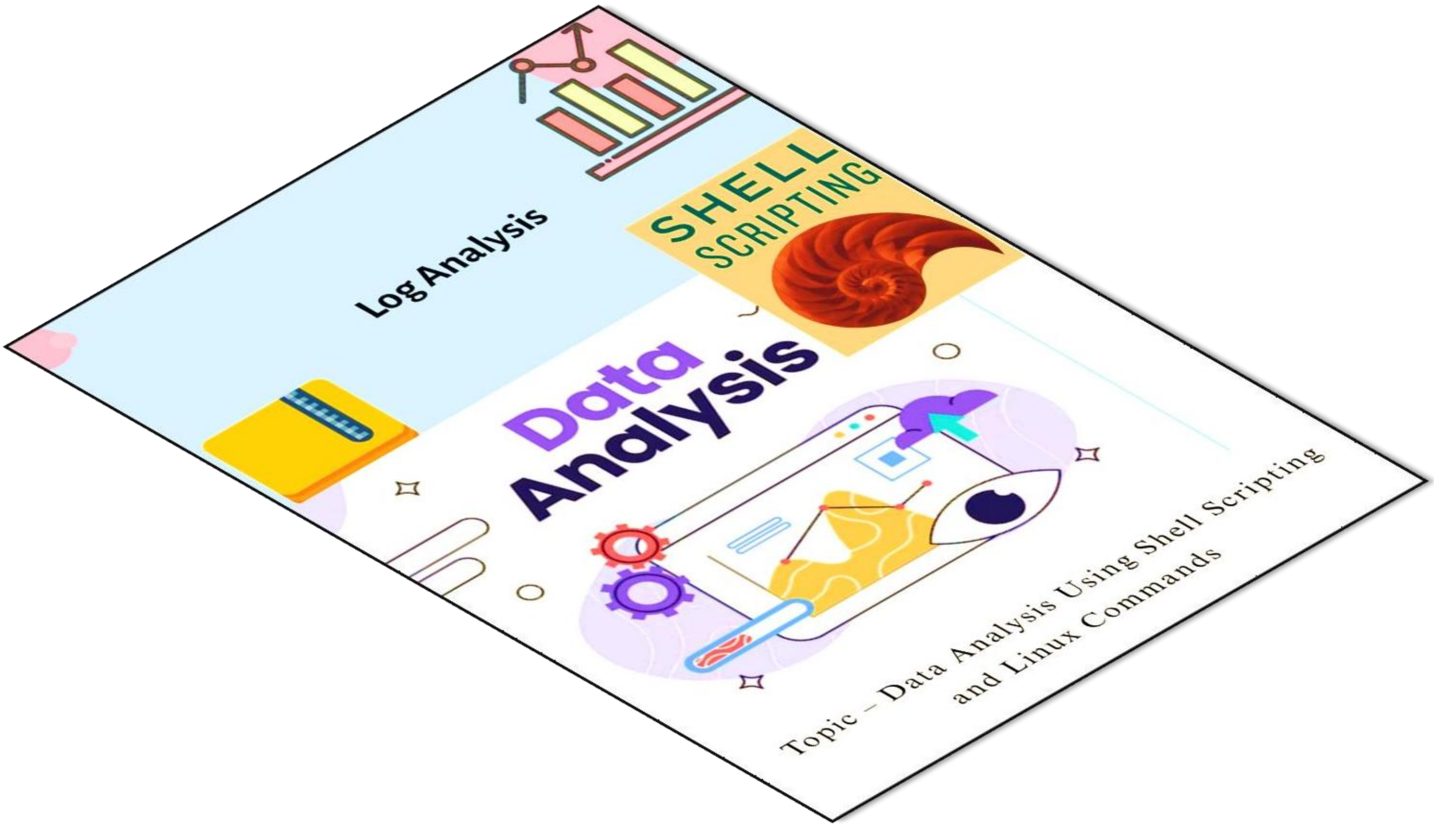
MOS CIA – 1 ASSIGNMENT


Log Analysis
SHELL SCRIPTING
Data Analysis
Topic – Data Analysis Using Shell Scripting and Linux Commands

NAME – VIKAS RANA
CLASS – T.Y.B.Sc.I.T
ROLL NUMBER – 25
UID - 2305033

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.

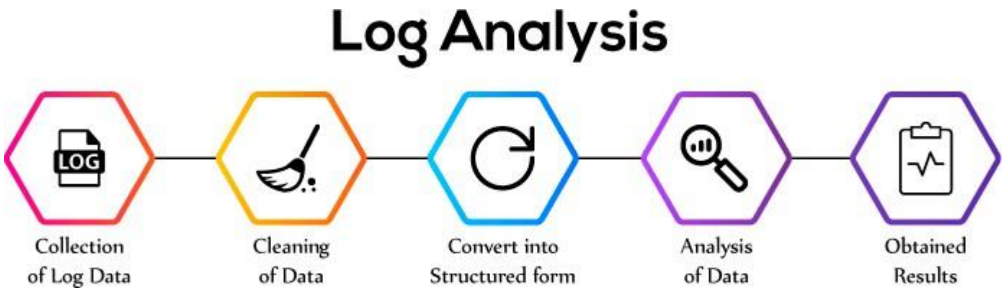
This report presents a systematic approach to extracting, cleaning, and analyzing such unstructured access log data using only standard Linux command-line tools and shell scripting. The analysis emphasizes lightweight, reproducible methods without relying on external programming languages or frameworks.

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.).

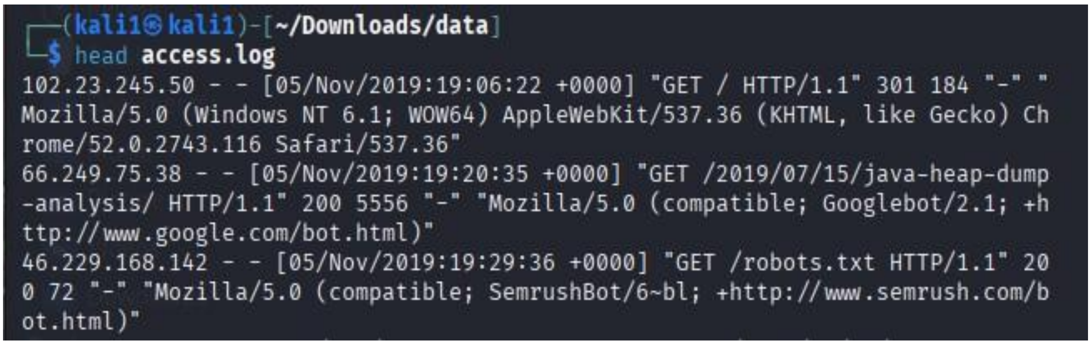


Methodology -

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.



```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ head access.log
102.23.245.50 - - [05/Nov/2019:19:06:22 +0000] "GET / HTTP/1.1" 301 184 "-" "
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
rome/52.0.2743.116 Safari/537.36"
66.249.75.38 - - [05/Nov/2019:19:20:35 +0000] "GET /2019/07/15/java-heap-dump
-analysis/ HTTP/1.1" 200 5556 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +h
ttp://www.google.com/bot.html)"
46.229.168.142 - - [05/Nov/2019:19:29:36 +0000] "GET /robots.txt HTTP/1.1" 20
0 72 "-" "Mozilla/5.0 (compatible; SemrushBot/6~bl; +http://www.semrush.com/b
ot.html)"
```

- 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)

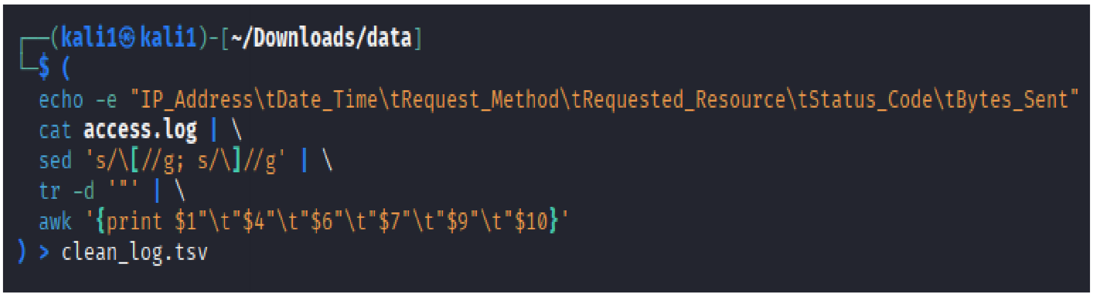
2. Data Cleaning and Transformation

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 –



```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ (
  echo -e "IP_Address\tDate_Time\tRequest_Method\tRequested_Resource\tStatus_Code\tBytes_Sent"
  cat access.log | \
  sed 's/\[//g; s/\]//g' | \
  tr -d '"' | \
  awk '{print $1"\t"$4"\t"$6"\t"$7"\t"$9"\t"$10}'
) > clean_log.tsv
```

- 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`

3. 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

4. Visualization with Gnuplot

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

Chart types used:

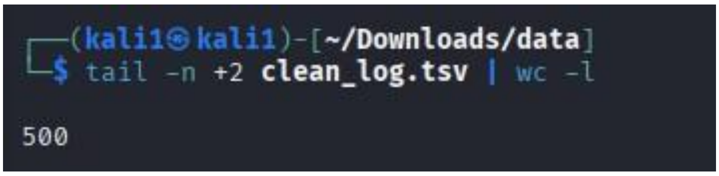
- **Bar Charts** (e.g., top IPs, error-causing IPs)
- **Pie Charts** (e.g., top pages, daily requests)
- **Donut Charts** (e.g., request methods, large responses)
- **Line Charts** (e.g., requests per day)
- **Gauge Charts** (e.g., average bytes per response)

5. 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 (`>`, `>>`)

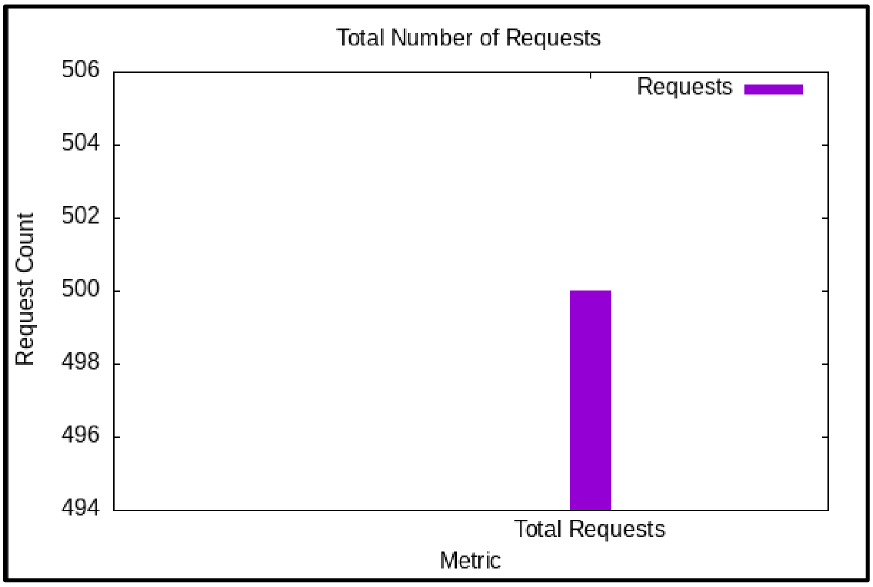
Insights into Analysis and Graphs -

1) Total number of requests –

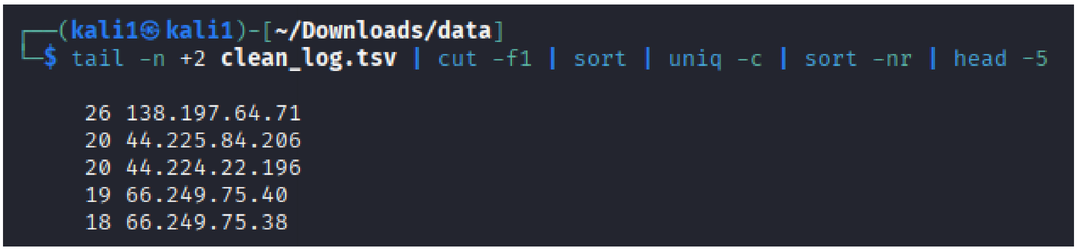


```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | wc -l

500
```

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



2) Top 5 Most Frequent IP Addresses –



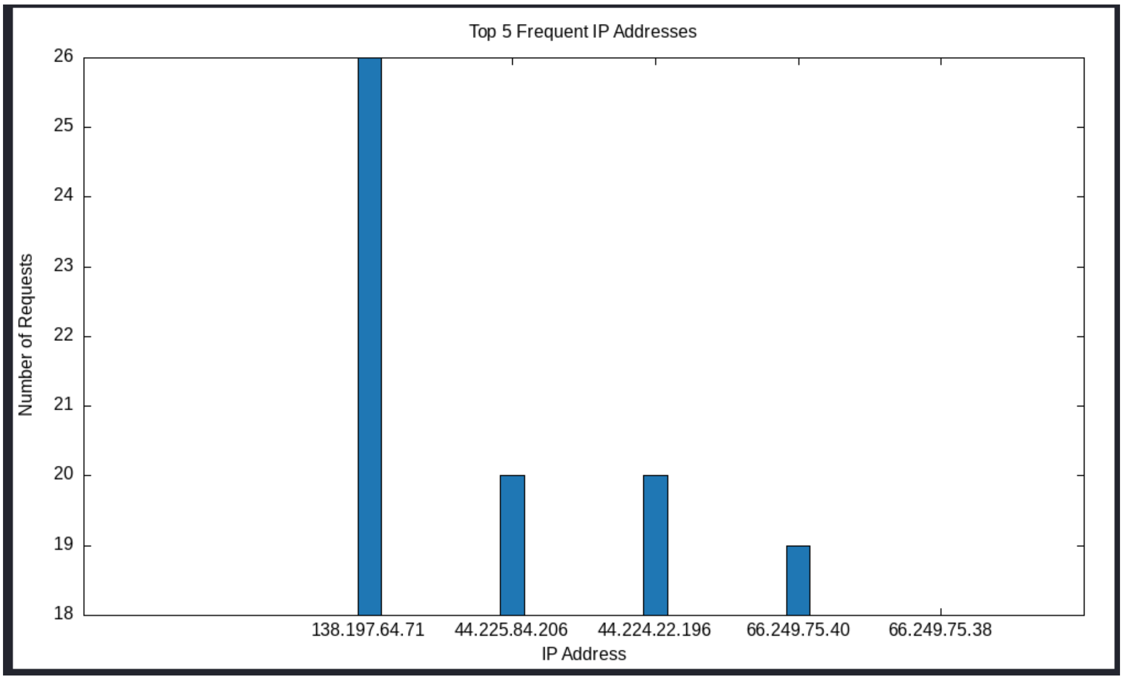
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f1 | sort | uniq -c | sort -nr | head -5

     26 138.197.64.71
     20 44.225.84.206
     20 44.224.22.196
     19 66.249.75.40
     18 66.249.75.38
```

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

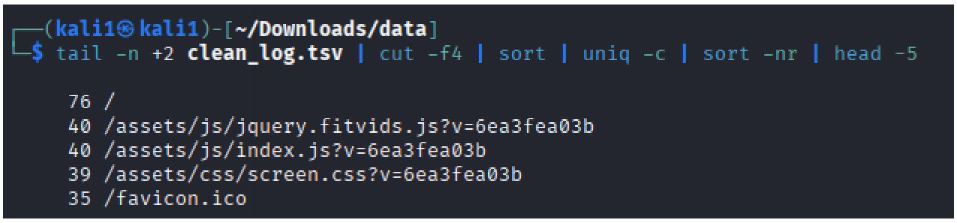
- ***Interpretation –***

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 –***



3) Top 5 Most Accessed Pages –



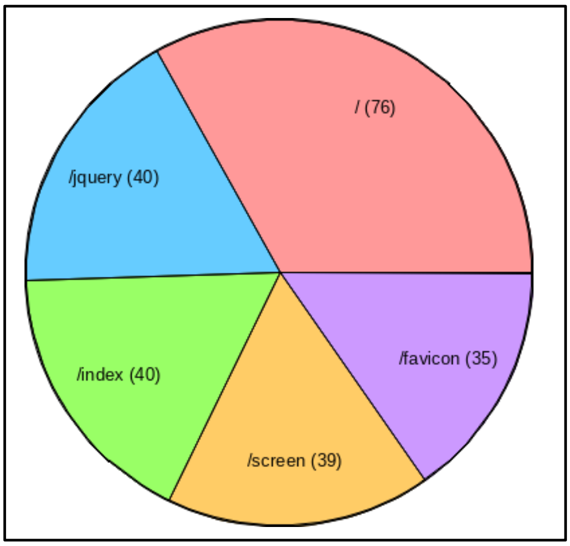
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f4 | sort | uniq -c | sort -nr | head -5

     76 /
     40 /assets/js/jquery.fitvids.js?v=6ea3fea03b
     40 /assets/js/index.js?v=6ea3fea03b
     39 /assets/css/screen.css?v=6ea3fea03b
     35 /favicon.ico
```

- cut `-f4`: extracts the resource path
- Counts how often each page is accessed

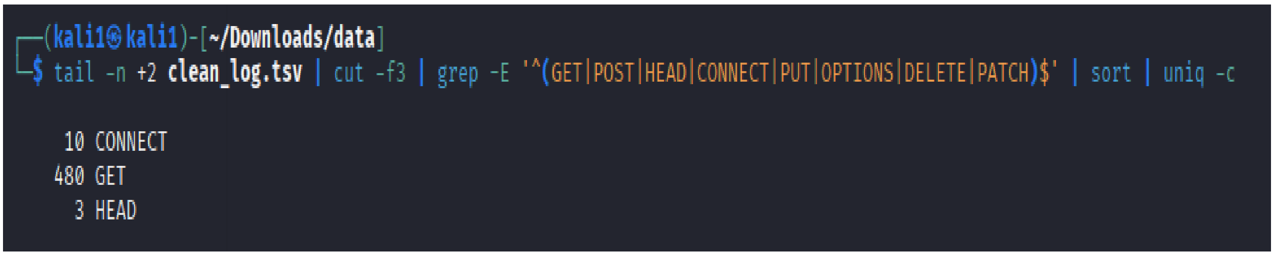
- ***Interpretation –***

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

- ***Visualization –***



4) *Requests by HTTP Method*



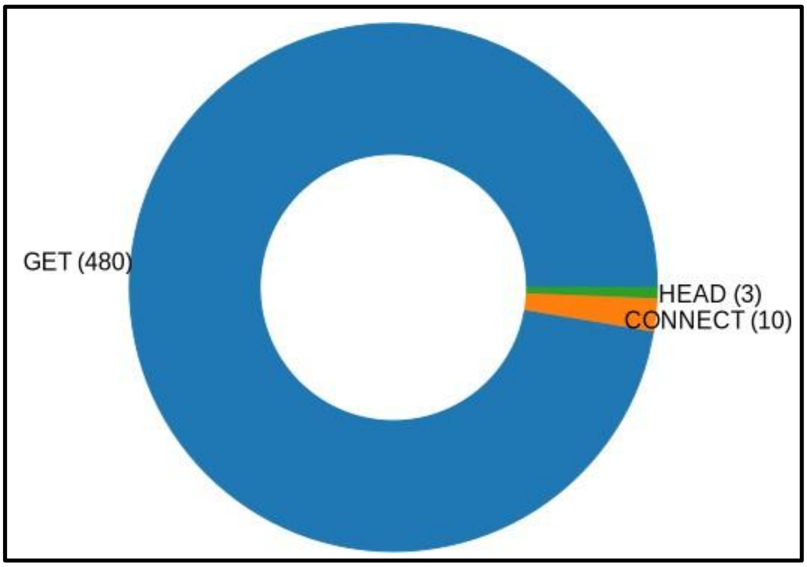
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f3 | grep -E '^(GET|POST|HEAD|CONNECT|PUT|OPTIONS|DELETE|PATCH)$' | sort | uniq -c

     10 CONNECT
    480 GET
      3 HEAD
```

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

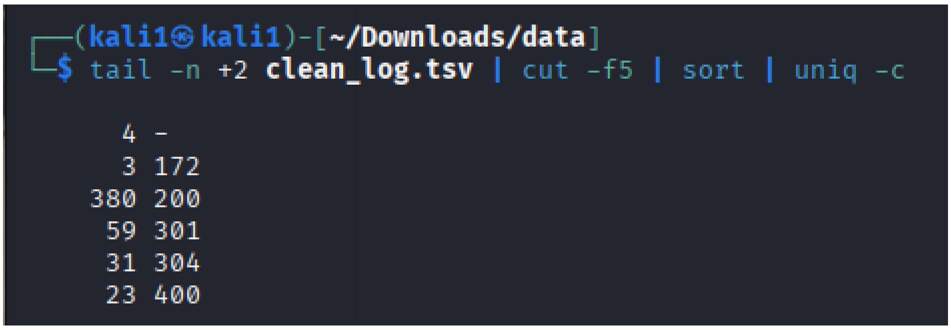
- ***Interpretation –***

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 –***



5) Status Code Distribution (200, 404, etc.)



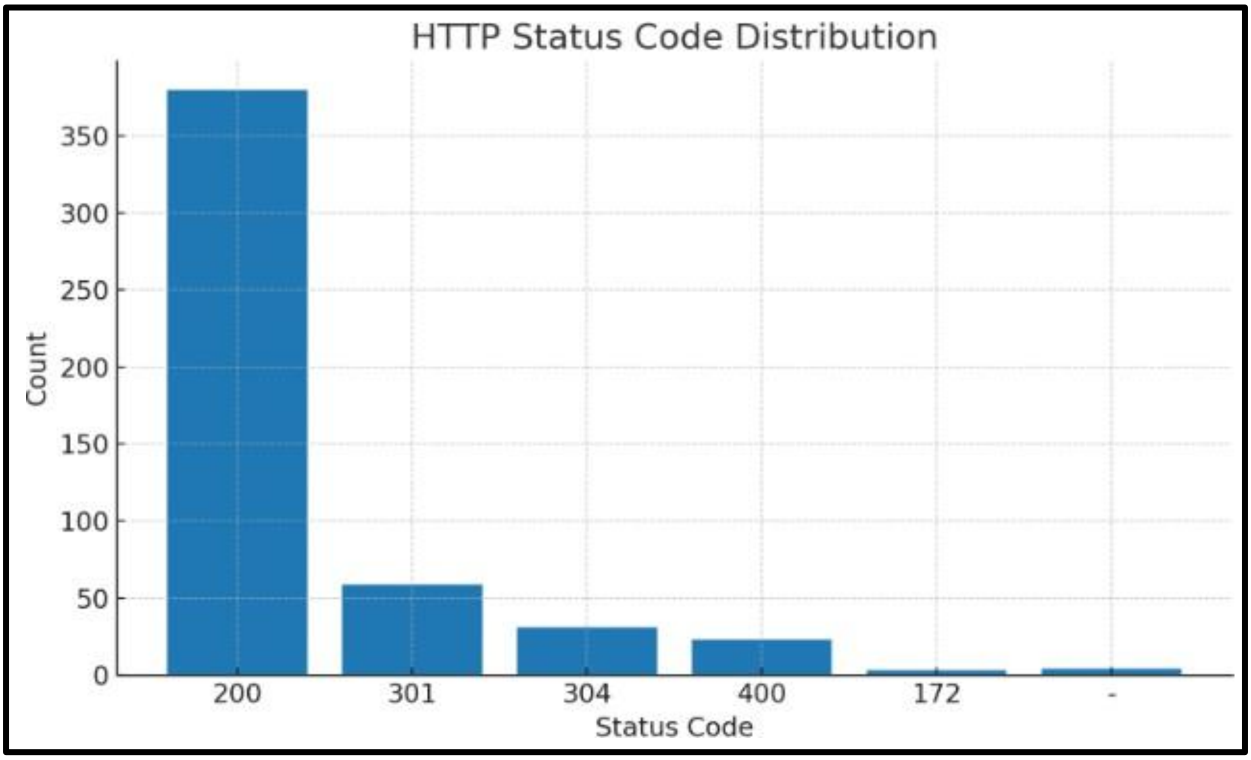
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f5 | sort | uniq -c

      4 -
      3 172
    380 200
     59 301
     31 304
     23 400
```

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

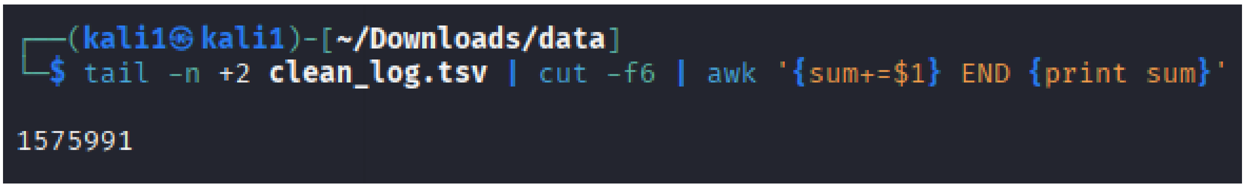
- ***Interpretation –***

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 –***



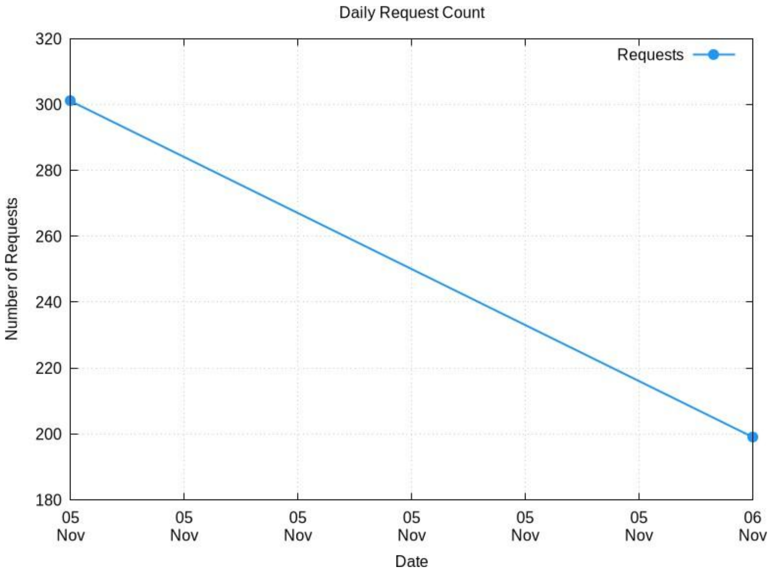
6) Total Bandwidth Used



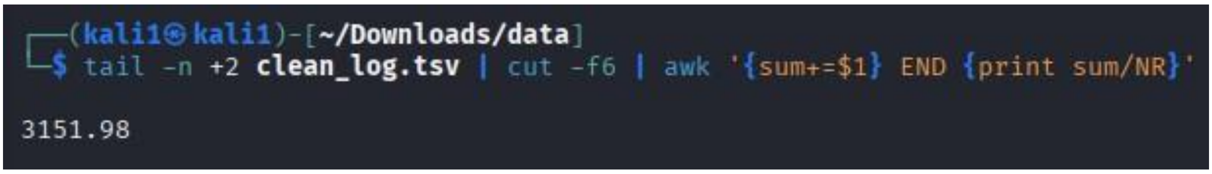
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f6 | awk '{sum+=$1} END {print sum}'

1575991
```

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 –** To analyse this I have used line-graph. So the line-graph tracks the amount of request send between 5 and 6 November. If we add up all the daily request, we get **1575991.***



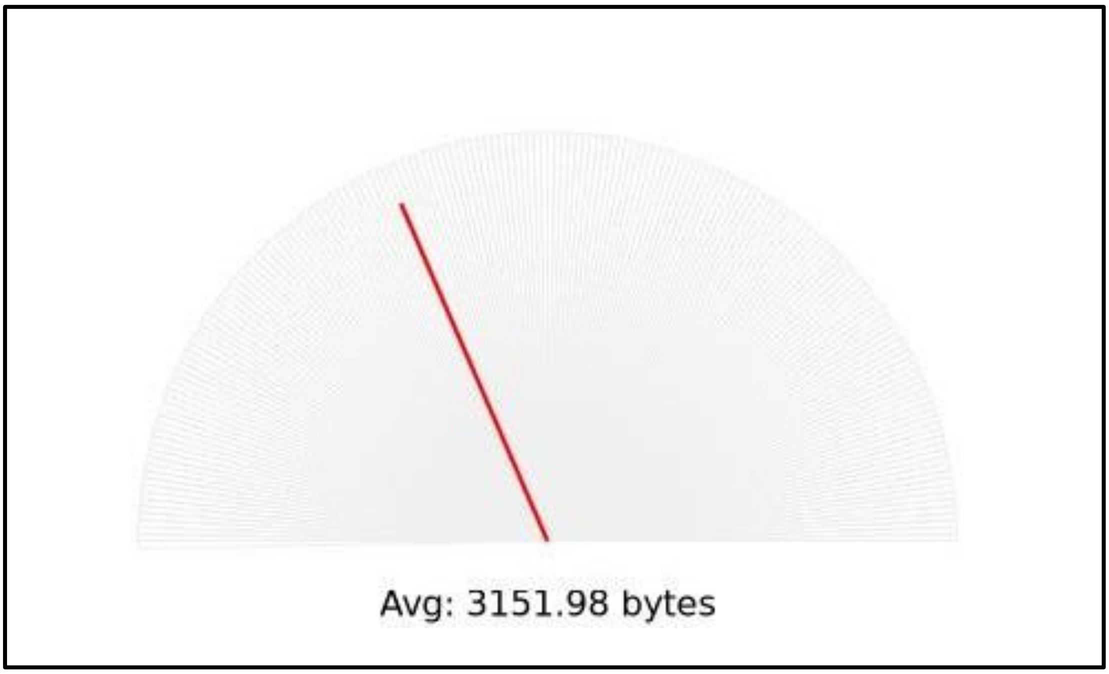
7) Average Response Size



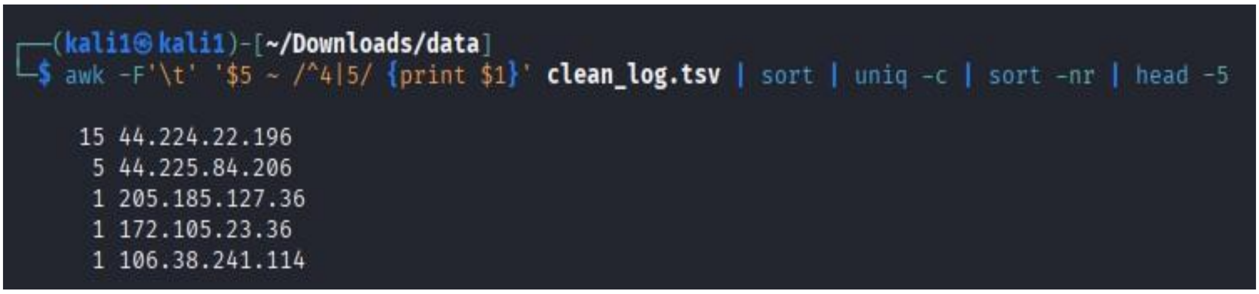
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f6 | awk '{sum+=$1} END {print sum/NR}'

3151.98
```

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 –***



8) IPs Causing Most Errors (status ≥ 400)



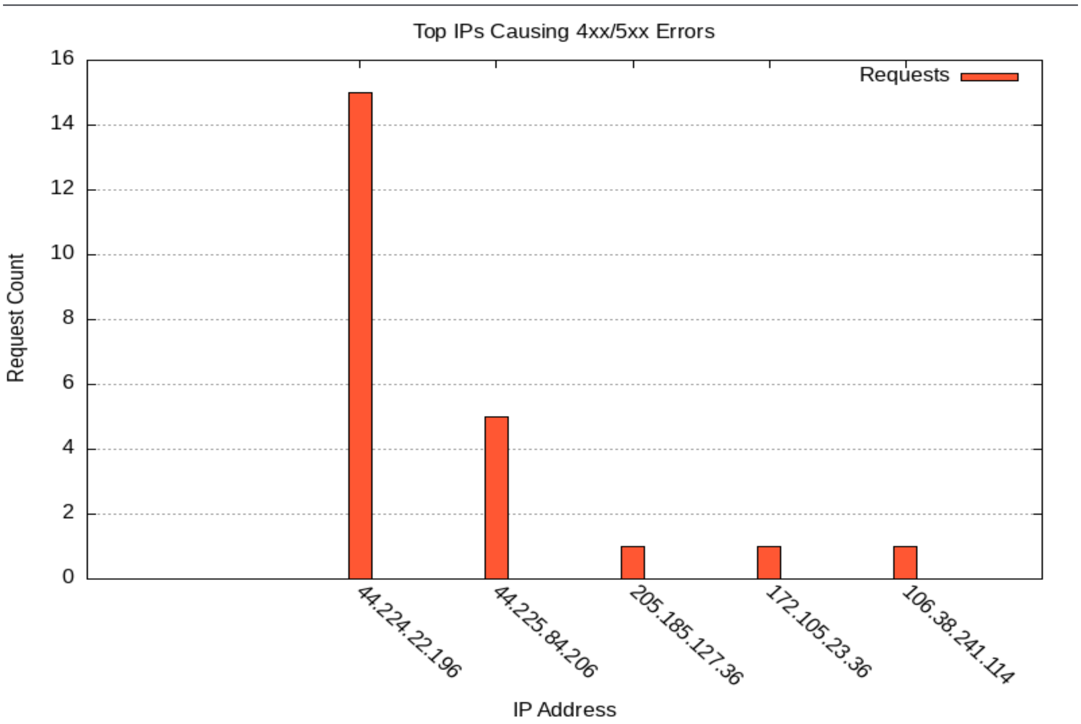
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ awk -F'\t' '$5 ~ /^4|5/ {print $1}' clean_log.tsv | sort | uniq -c | sort -nr | head -5

     15 44.224.22.196
      5 44.225.84.206
      1 205.185.127.36
      1 172.105.23.36
      1 106.38.241.114
```

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

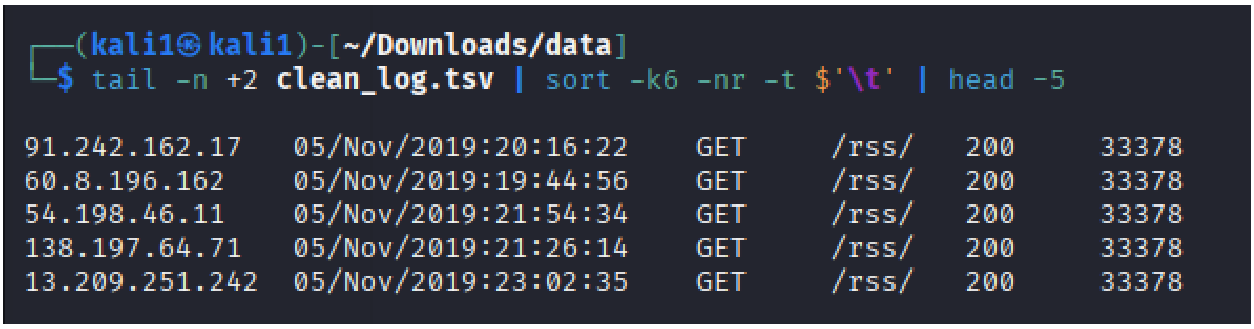
- ***Interpretation –***

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 –***



9) Top 5 Largest Responses (by Byte Size)



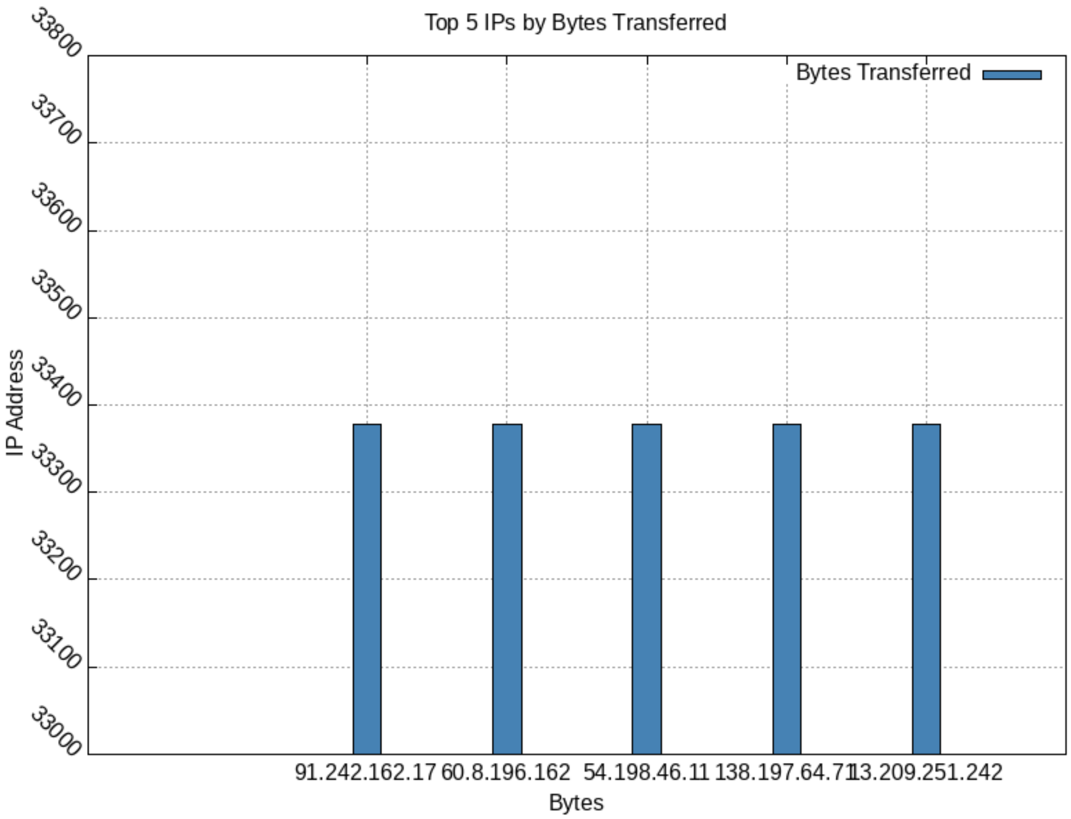
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | sort -k6 -nr -t $'\t' | head -5

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
```

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

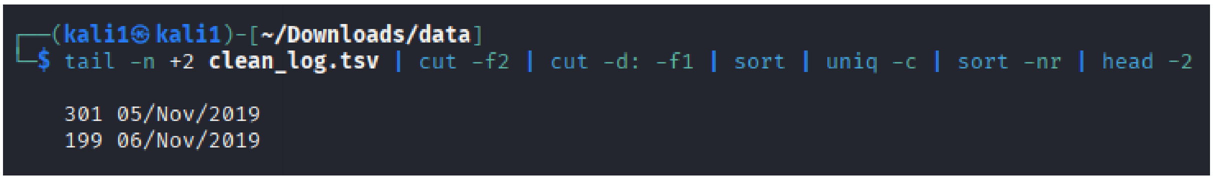
- ***Interpretation –***

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 –***



10) Top 2 Dates with Most Requests



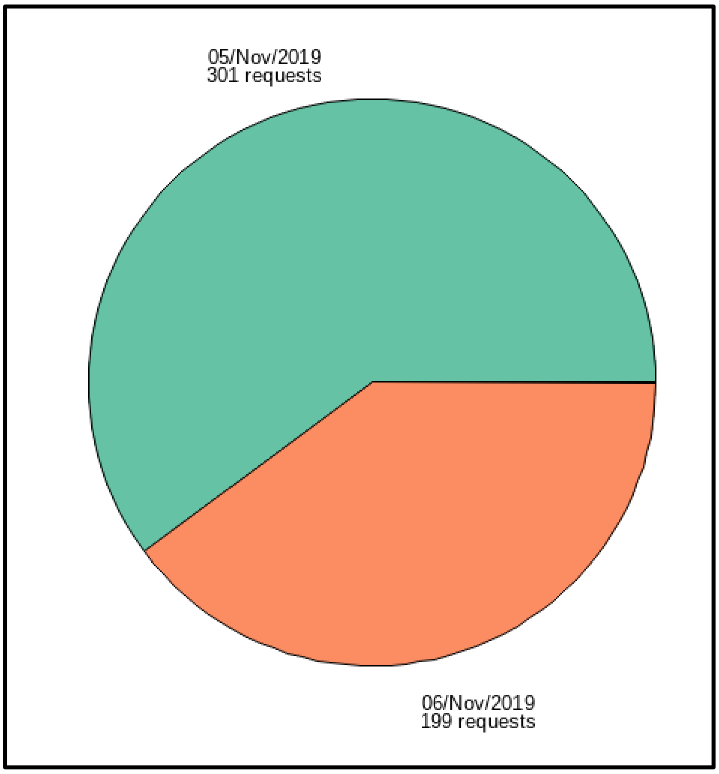
```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ tail -n +2 clean_log.tsv | cut -f2 | cut -d: -f1 | sort | uniq -c | sort -nr | head -2

    301 05/Nov/2019
    199 06/Nov/2019
```

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

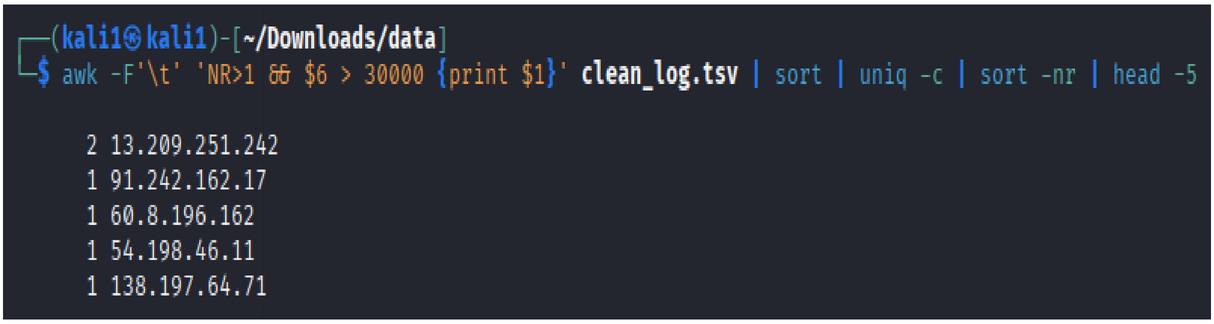
- ***Interpretation –***

Date	Number of Requests
05/Nov/2019	301
06/Nov/2019	199

- ***Visualization –***



11) Detect IPs Requesting Large Responses



```
┌──(kali1㉿kali1)-[~/Downloads/data]
└─$ awk -F'\t' 'NR>1 && $6 > 30000 {print $1}' clean_log.tsv | sort | uniq -c | sort -nr | head -5

      2 13.209.251.242
      1 91.242.162.17
      1 60.8.196.162
      1 54.198.46.11
      1 138.197.64.71
```

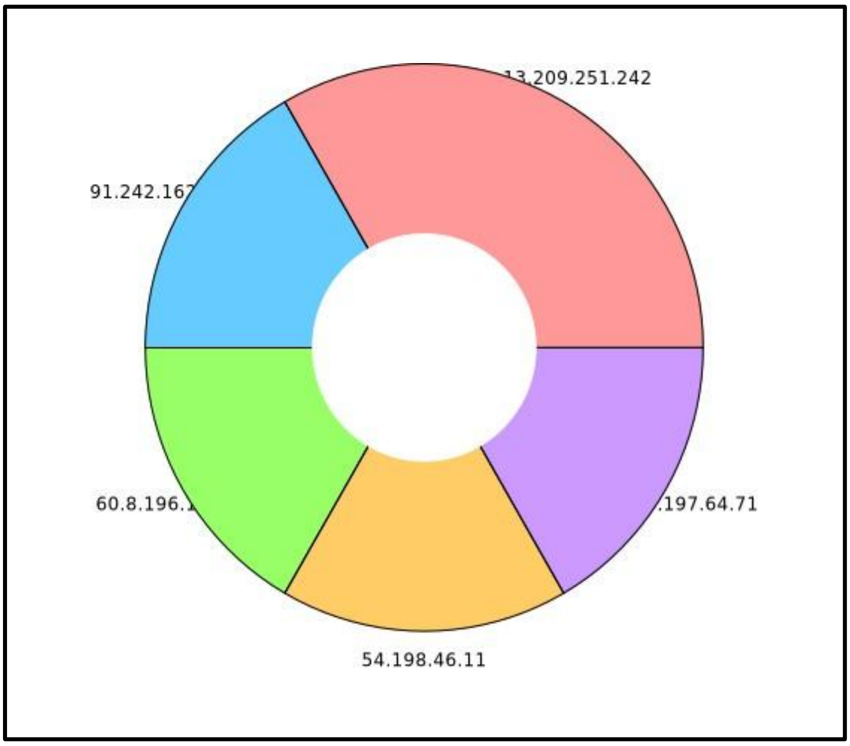
Part	Meaning
`awk -F'\t'`	Uses tab (`\t`) as the field separator because your file is tab-separated
`NR>1`	Skips the header row (starts from row 2)
`$6 > 30000`	Only considers rows where the **Bytes_Sent (column 6)** is greater than **30,000** (a large response)

`{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.

Link - https://github.com/St-Xavier-s-College-Mumbai/mos-data-analysis-using-shell-scripting-and-linux-commands-vikasrana5