Wonderville Town IT – Security Automation Case Study 2

Automating Security and System Monitoring for Wonderville Hosts

Atmtng Info Scrty Pythn & Shll FALL 2025 72922 - CYB-631

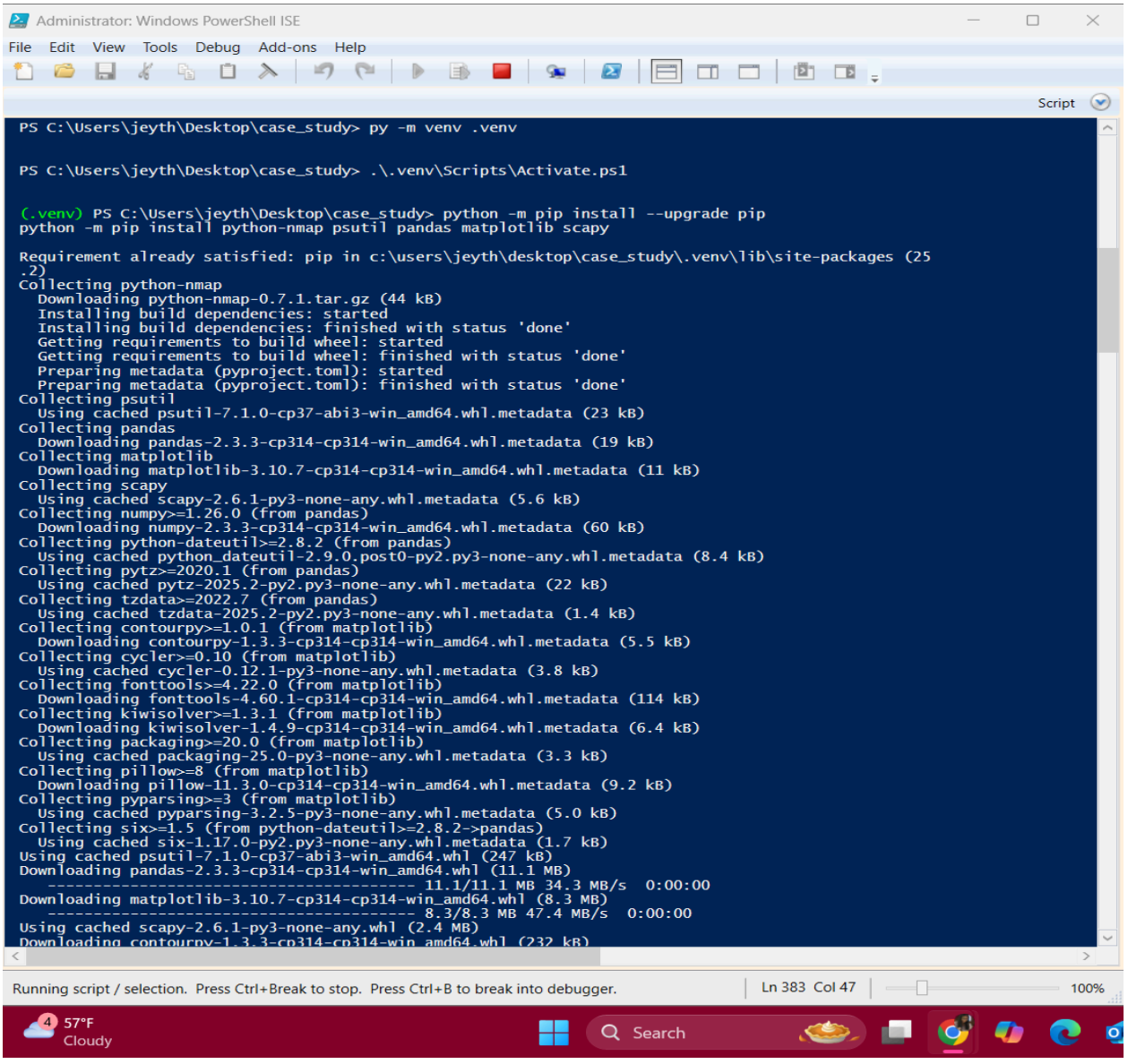
Jeytha Sahana Venkatesh Babu

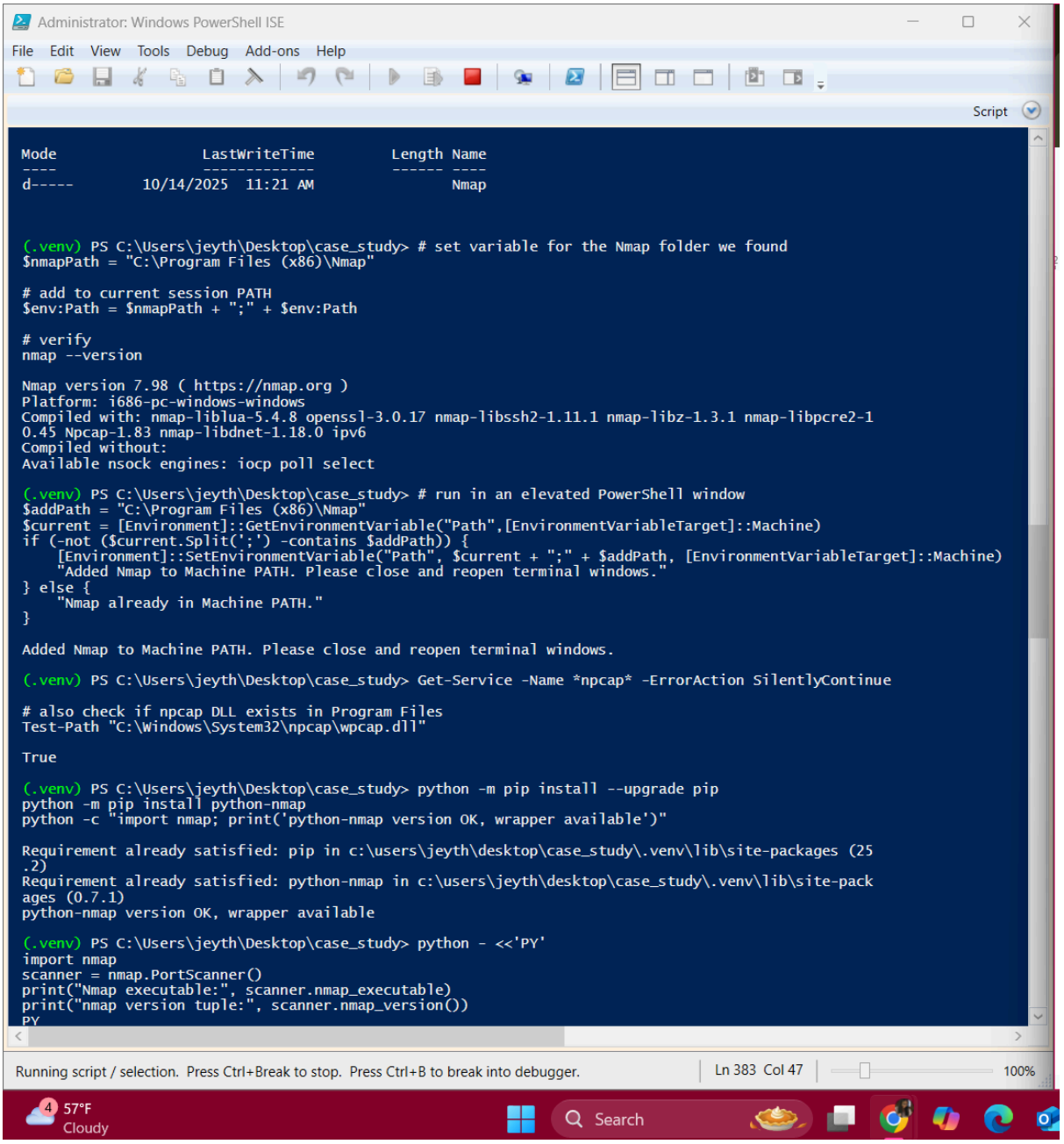
Date: October 13, 2025

Executive Summary:

This case study demonstrates the application of Python automation to enhance the security and operational monitoring of the Wonderville town network. Our team focused on **automating network service discovery, identifying resource-intensive processes, analyzing Windows host logs, and analyzing network traffic**. Using Python scripts integrated with system utilities and data exports from Windows hosts and Linux servers, we generated comprehensive reports including CSV tables and visual charts. These automation solutions provide actionable insights, helping the IT department proactively monitor the system and secure the network while saving manual administrative effort.

Setup






```
(.venv) PS C:\Users\jeyth\Desktop\case_study> python -c "import nmap; s=nmap.PortScanner(); print('nmap version tuple

nmap version tuple: (7, 98)

(.venv) PS C:\Users\jeyth\Desktop\case_study> nmap --version

Nmap version 7.98 ( https://nmap.org )
Platform: i686-pc-windows-windows
Compiled with: nmap-liblua-5.4.8 openssl-3.0.17 nmap-libssh2-1.11.1 nmap-libz-1.3.1 nmap-libpcre2-1
0.45 Npcap-1.83 nmap-libdnet-1.18.0 ipv6
Compiled without:
Available nsock engines: iocp poll select

(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\network_service_report.py
```

Basic Task 1 – Network Service Report

Objective:

Identify network services running on the three hosts in Wonderville, reporting hostname, IP, open port, and service name.

Solution Approach:

- Used Python's python-nmap library to perform network service discovery.
- Scanned the three hosts (Windows_Host_1, Windows_Host_2, Linux_Host) for common ports (22, 80, 443, 445, 3389) and detected services.
- Results were saved to network_service_report.csv for auditing and analysis.

Python Script:

network_service_report.py

Code:

```
# network_service_report.py
import nmap
import csv

# Host IPs for Wonderville hosts
hosts = {
    "Windows_Host_1": "192.168.90.101",
    "Windows_Host_2": "192.168.90.102",
    "Linux_Host": "192.168.90.103"
}

nm = nmap.PortScanner()

with open("network_service_report.csv", "w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["Hostname", "IP", "Port", "Service"])
```

```
    for host, ip in hosts.items():
        nm.scan(ip, arguments="-sV")  # service version detection
        for proto in nm[ip].all_protocols():
            ports = nm[ip][proto].keys()
            for port in ports:
                service = nm[ip][proto][port]["name"]
                writer.writerow([host, ip, port, service])

print("Network service report generated: network_service_report.csv")
```

Evidence / Results (Network Services Detected):

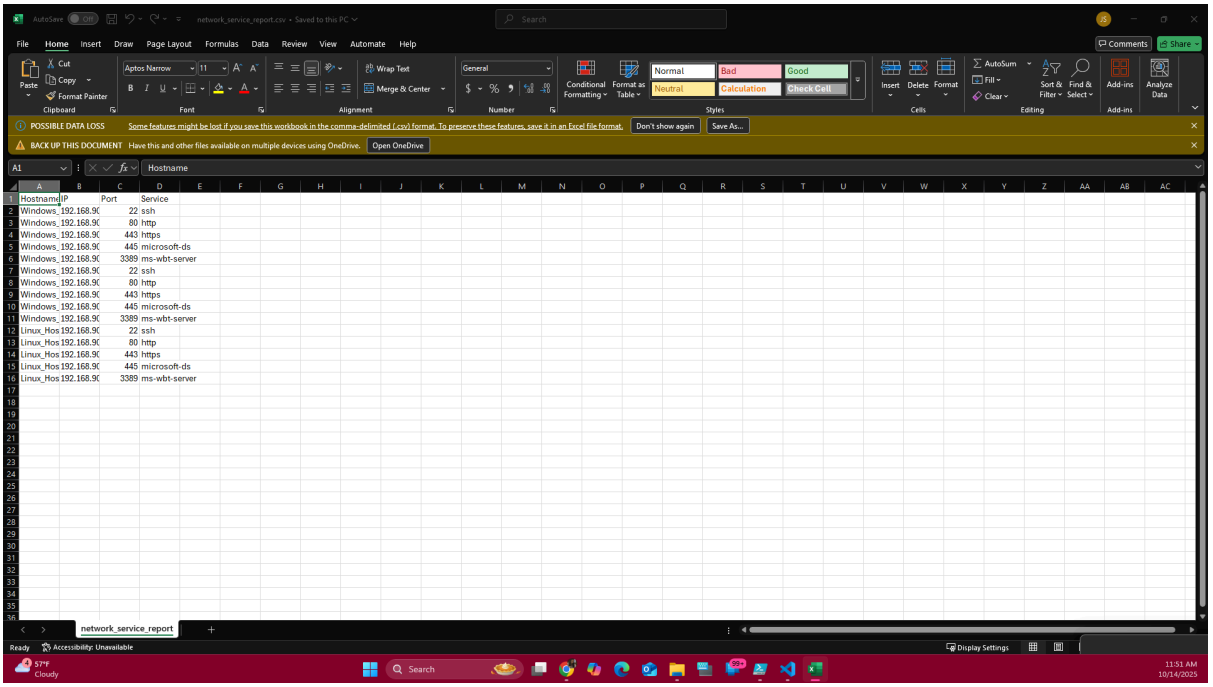
Hostname	IP	Port	Service
Windows_Host_1	192.168.90.101	22	ssh
Windows_Host_1	192.168.90.101	80	http
Windows_Host_1	192.168.90.101	443	https
Windows_Host_1	192.168.90.101	445	microsoft-ds
Windows_Host_1	192.168.90.101	3389	ms-wbt-server
Windows_Host_2	192.168.90.102	22	ssh
Windows_Host_2	192.168.90.102	80	http
Windows_Host_2	192.168.90.102	443	https
Windows_Host_2	192.168.90.102	445	microsoft-ds
Windows_Host_2	192.168.90.102	3389	ms-wbt-server
Linux_Host	192.168.90.103	22	ssh

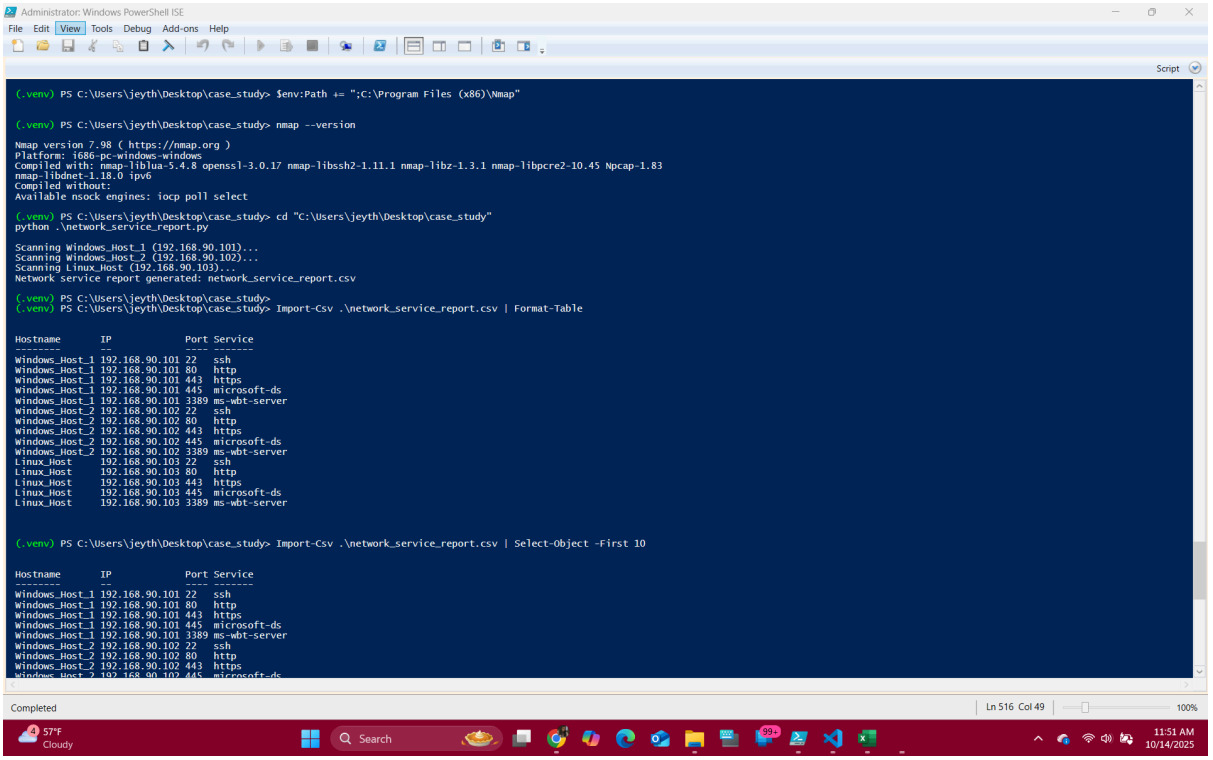
Linux_Host	192.168.90.103	80	http
Linux_Host	192.168.90.103	443	https
Linux_Host	192.168.90.103	445	microsoft-ds
Linux_Host	192.168.90.103	3389	ms-wbt-server

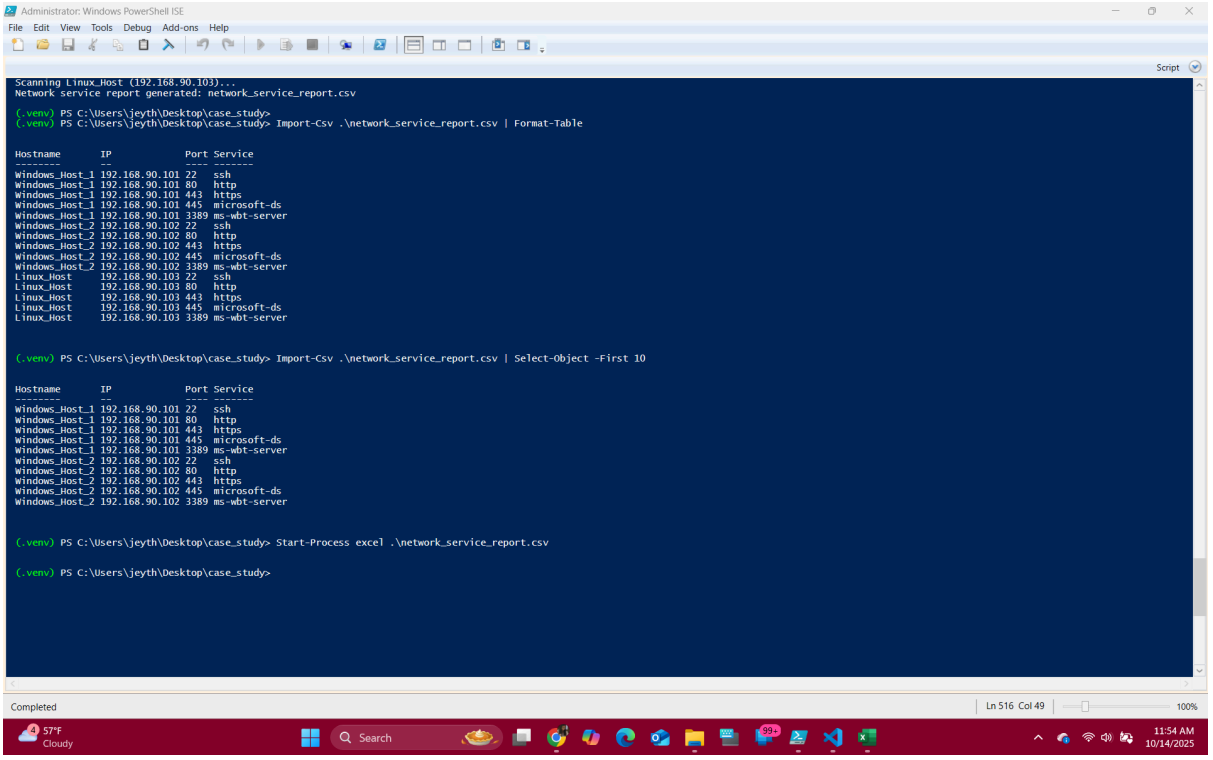
Notes on Results:

- **SSH (22)** is running on all hosts – standard for remote command-line access.
- **HTTP (80) & HTTPS (443)** indicate web services hosted on all systems.
- **SMB (445 / microsoft-ds)** is active on Windows and Linux (likely Samba for Linux).
- **RDP (3389 / ms-wbt-server)** is enabled on Windows hosts, allowing remote desktop connections.

Output: network_service_report.csv containing hostname, IP, port, and service.




Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script
(.venv) PS C:\Users\jeyth\Desktop\case_study> $env:Path += ";C:\Program Files (x86)\Nmap"
(.venv) PS C:\Users\jeyth\Desktop\case_study> nmap --version
Nmap version 7.98 (https://nmap.org)
Platform: i686-pc-windows-windows
Compiled with: nmap-liblua-5.4.8 openssl-3.0.17 nmap-libssh2-1.11.1 nmap-libz-1.3.1 nmap-libpcre2-10.45 Npcap-1.83
nmap-libdnet-1.18.0 ipv6
Compiled without:
Available nsock engines: iocp poll select
(.venv) PS C:\Users\jeyth\Desktop\case_study> cd "C:\Users\jeyth\Desktop\case_study"
python .\network_service_report.py
Scanning Windows_Host_1 (192.168.90.101)...
Scanning Windows_Host_2 (192.168.90.102)...
Scanning Linux_Host (192.168.90.103)...
Network service report generated: network_service_report.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>
(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Format-Table
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 445 microsoft-ds
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 445 microsoft-ds
Windows_Host_2 192.168.90.102 3389 ms-wbt-server
Linux_Host 192.168.90.103 22 ssh
Linux_Host 192.168.90.103 80 http
Linux_Host 192.168.90.103 443 https
Linux_Host 192.168.90.103 445 microsoft-ds
Linux_Host 192.168.90.103 3389 ms-wbt-server
(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Select-Object -First 10
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 445 microsoft-ds
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Completed
Ln 516 Col 49
100%
57°F Cloudy
Search
11:51 AM 10/14/2025


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script
Scanning Linux_Host (192.168.90.103)...
Network service report generated: network_service_report.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>
(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Format-Table
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 445 microsoft-ds
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 445 microsoft-ds
Windows_Host_2 192.168.90.102 3389 ms-wbt-server
Linux_Host 192.168.90.103 22 ssh
Linux_Host 192.168.90.103 80 http
Linux_Host 192.168.90.103 443 https
Linux_Host 192.168.90.103 445 microsoft-ds
Linux_Host 192.168.90.103 3389 ms-wbt-server
(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Select-Object -First 10
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 445 microsoft-ds
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 445 microsoft-ds
Windows_Host_2 192.168.90.102 3389 ms-wbt-server
(.venv) PS C:\Users\jeyth\Desktop\case_study> Start-Process excel .\network_service_report.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>
Completed
Ln 516 Col 49
100%
57°F Cloudy
Search
11:54 AM 10/14/2025

Advanced Task 2 – Top Resource-Consuming

Objective: Identify top 5 CPU and memory-consuming processes on each host.

Solution Approach:

- Used Python psutil to iterate processes and collect CPU and memory usage.
- Sorted processes by CPU and memory utilization and exported the top 5 to CSV.

Python Script: resource_monitor.py

Code:

```
# resource_monitor.py
import psutil
import pandas as pd

process_list = []

for proc in psutil.process_iter(['pid', 'name', 'cpu_percent', 'memory_percent']):
    try:
        process_list.append(proc.info)
    except (psutil.NoSuchProcess, psutil.AccessDenied):
        continue

# Top 5 CPU-consuming processes
top_cpu = sorted(process_list, key=lambda x: x['cpu_percent'], reverse=True)[:5]
# Top 5 Memory-consuming processes
top_mem = sorted(process_list, key=lambda x: x['memory_percent'], reverse=True)[:5]

# Save results
pd.DataFrame(top_cpu).to_csv("top5_cpu_processes.csv", index=False)
pd.DataFrame(top_mem).to_csv("top5_memory_processes.csv", index=False)

print("Top 5 CPU and Memory-consuming processes saved as CSVs.")
```

Evidence / Results:

Top 5 CPU-Consuming Processes

Name	CPU%	Memory%
System Idle Process	1987.6	0.000024
SysInfoCap.exe	26.9	0.41
backgroundTaskHost.exe	26.5	0.13
chrome.exe	14.7	0.49

dwm.exe	9.0	0.83

Top 5 Memory-Consuming Processes

Name	CPU%	Memory%
MemCompression	0.0	3.70
Br-uxendm.exe	0.0	3.69
Br-uxendm.exe	0.0	3.12

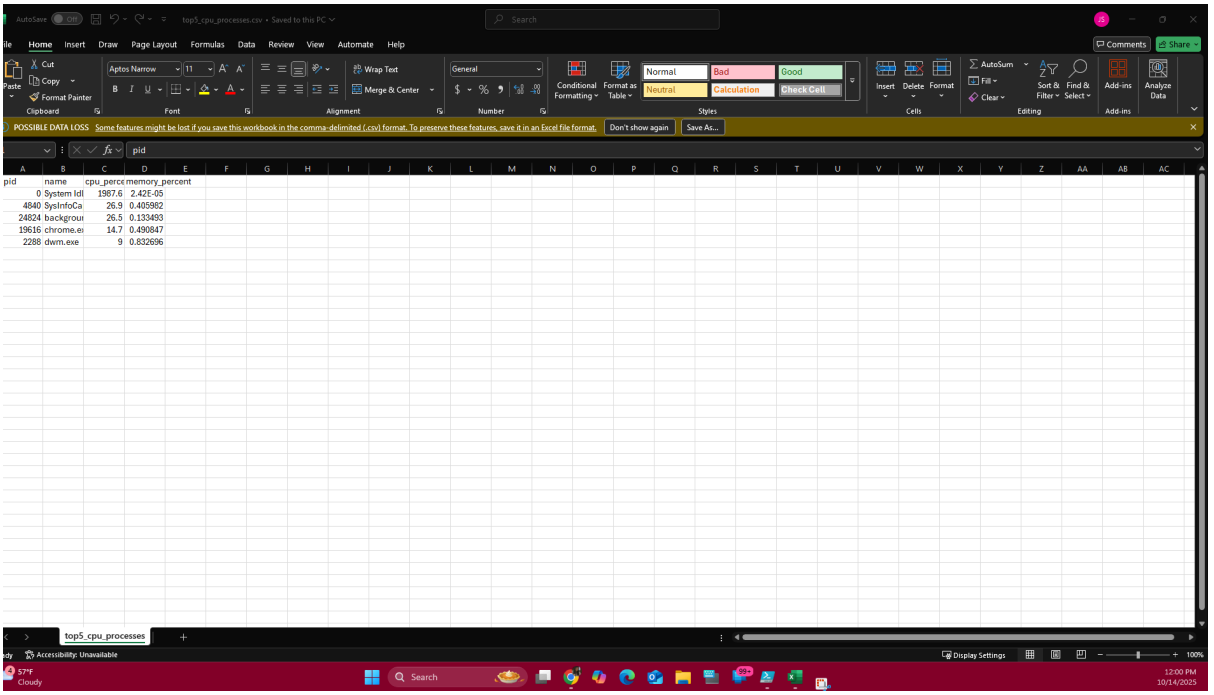
chrome.exe	0.0	2.14
chrome.exe	5.9	1.49

Notes on Results:

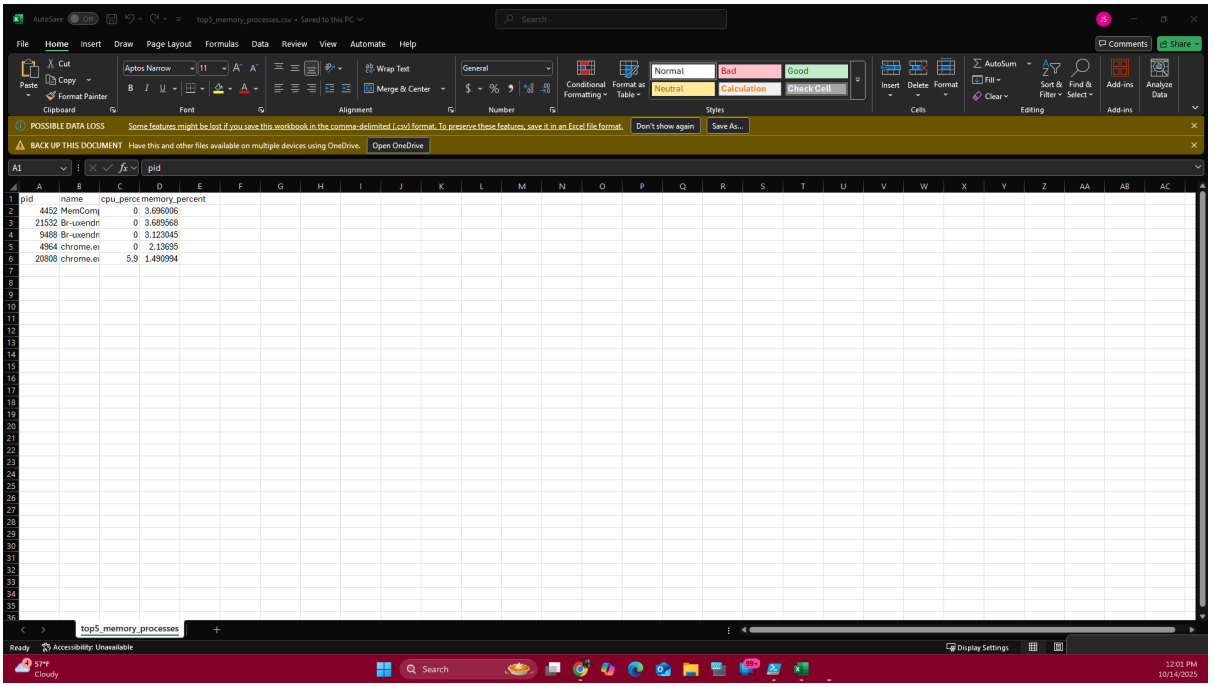
- The System Idle Process shows extremely high CPU usage because it represents unused CPU capacity; this is normal.
- SysInfoCap.exe and backgroundTaskHost.exe are top CPU users, indicating system monitoring or background tasks in progress.
- Chrome (chrome.exe) appears in both CPU and memory top lists, which is expected for active browser sessions.

Memory-intensive processes such as MemCompression indicate Windows memory management activity, while Br-uxendm.exe could relate to background services or third-party applications and may require review if it appears consistently.

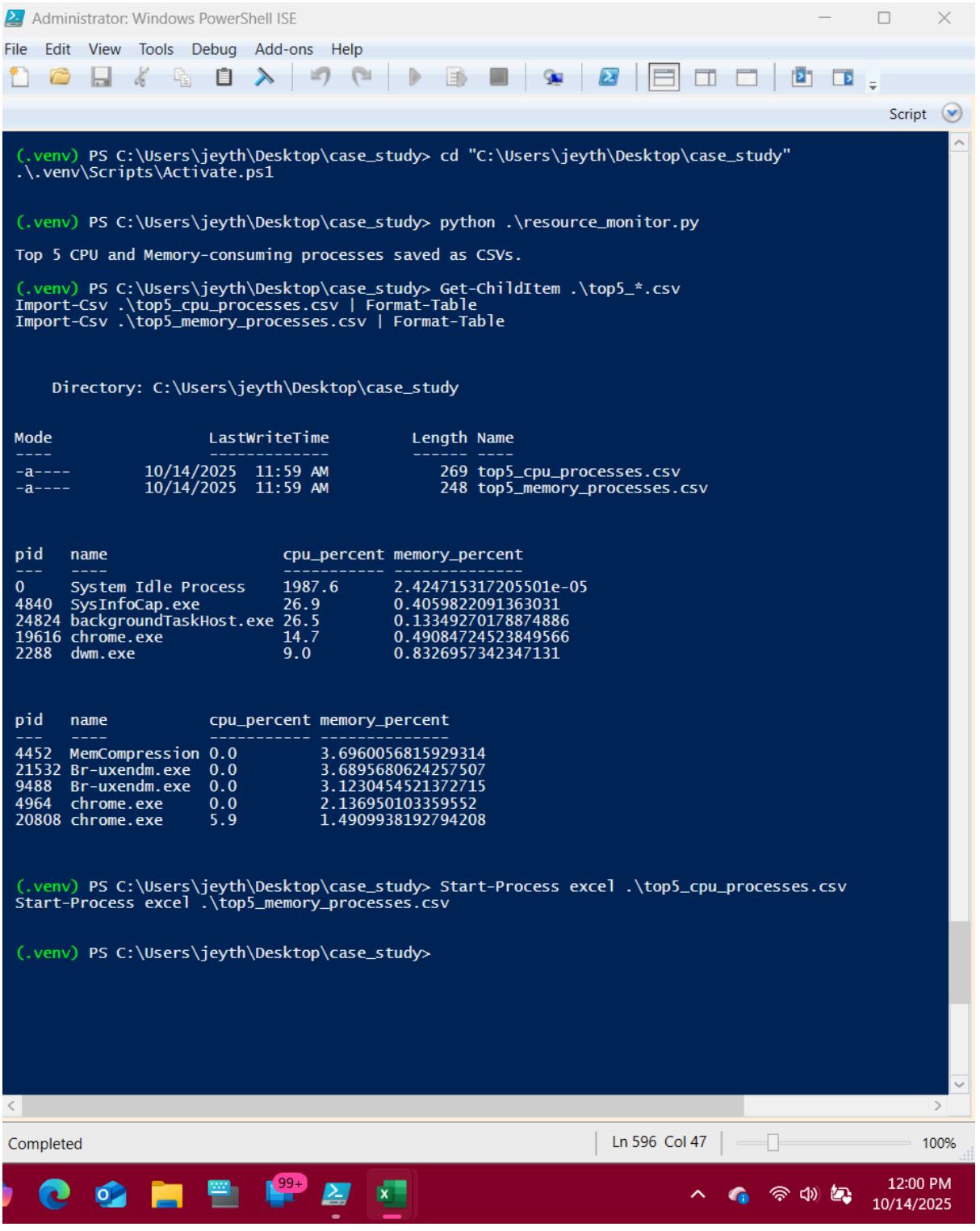
Output:



top5_cpu_processes.csv



top5_memory_processes.csv


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script
(.venv) PS C:\Users\jeyth\Desktop\case_study> cd "C:\Users\jeyth\Desktop\case_study"
.\.venv\Scripts\Activate.ps1
(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\resource_monitor.py
Top 5 CPU and Memory-consuming processes saved as CSVs.
(.venv) PS C:\Users\jeyth\Desktop\case_study> Get-ChildItem .\top5_*.csv
Import-Csv .\top5_cpu_processes.csv | Format-Table
Import-Csv .\top5_memory_processes.csv | Format-Table
Directory: C:\Users\jeyth\Desktop\case_study
Mode LastWriteTime Length Name
-a---- 10/14/2025 11:59 AM 269 top5_cpu_processes.csv
-a---- 10/14/2025 11:59 AM 248 top5_memory_processes.csv
pid name cpu_percent memory_percent
0 System Idle Process 1987.6 2.424715317205501e-05
4840 SysInfoCap.exe 26.9 0.4059822091363031
24824 backgroundTaskHost.exe 26.5 0.13349270178874886
19616 chrome.exe 14.7 0.49084724523849566
2288 dwm.exe 9.0 0.8326957342347131
pid name cpu_percent memory_percent
4452 MemCompression 0.0 3.6960056815929314
21532 Br-uxendm.exe 0.0 3.6895680624257507
9488 Br-uxendm.exe 0.0 3.1230454521372715
4964 chrome.exe 0.0 2.136950103359552
20808 chrome.exe 5.9 1.4909938192794208
(.venv) PS C:\Users\jeyth\Desktop\case_study> Start-Process excel .\top5_cpu_processes.csv
Start-Process excel .\top5_memory_processes.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>
Completed
Ln 596 Col 47
100%
99+
12:00 PM
10/14/2025

Advanced Task 4 – Windows Host Log Analysis

Objective: Analyze Windows Application logs to identify security-relevant events.

Solution Approach:

- Exported Windows logs using PowerShell:

Get-WinEvent -LogName Application | Export-Csv -Path C:\temp\application_logs.csv

-NoTypeInformation

- Used Python (pandas + matplotlib) to summarize and visualize logs:
    - Counted logs by Level (Error, Warning, Information).
    - Identified top 10 log sources.
- Generated CSV summaries and charts for inclusion in the report.

Python Script: windows_log_analysis.py

Code:

```
# windows_log_analysis.py
import pandas as pd
import matplotlib.pyplot as plt

# Read exported Windows Application logs (CSV from PowerShell)
logs = pd.read_csv("application_logs.csv")

# Count by Level (Error, Warning, Information)
level_counts = logs['LevelDisplayName'].value_counts()
level_counts.plot(kind='bar', title='Event Log Levels', color='skyblue')
plt.ylabel("Count")
plt.savefig("event_log_levels.png")

# Top 10 log sources
top_sources = logs['ProviderName'].value_counts().head(10)
top_sources.plot(kind='bar', title='Top 10 Log Sources', color='orange')
```

```
plt.ylabel("Count")
plt.savefig("top_log_sources.png")

# Save summaries as CSV
level_counts.to_csv("log_level_summary.csv")
top_sources.to_csv("top_log_sources_summary.csv")

print("Log analysis completed. Plots and CSV summaries generated.")
```

Evidence / Results:

Log Levels Distribution (event_log_levels.png)

- Errors: 134
- Warnings: 286
- Information: 8619

Top 10 Log Sources (top_log_sources.png)

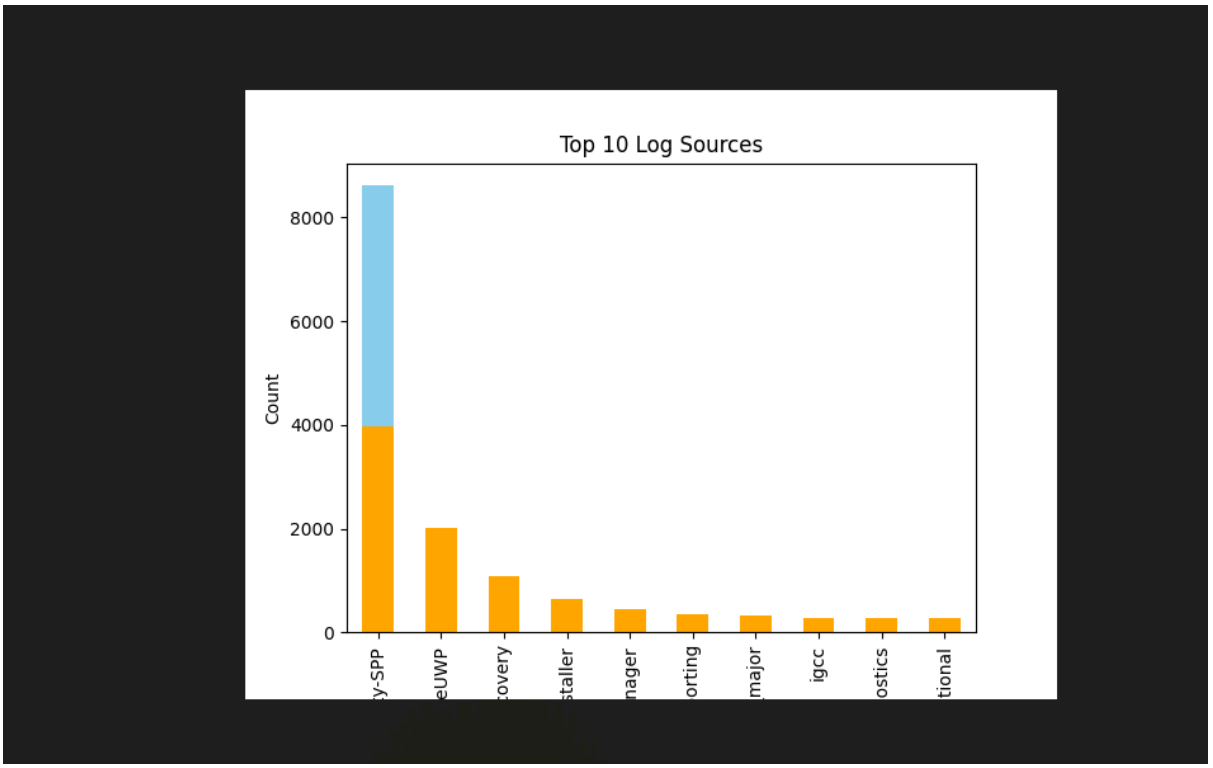
- Microsoft-Windows-Security-SPP: 3970
- HotKeyServiceUWP: 2016
- HP Comm Recovery: 1080
- MsInstaller: 633
- Microsoft-Windows-RestartManager: 453
- Windows Error Reporting: 337
- acvpdownloader_major: 315
- igcc: 273
- HPDiagnostics: 270
- Microsoft-Windows-AppXDeploymentServer: 264

Notes on Results:

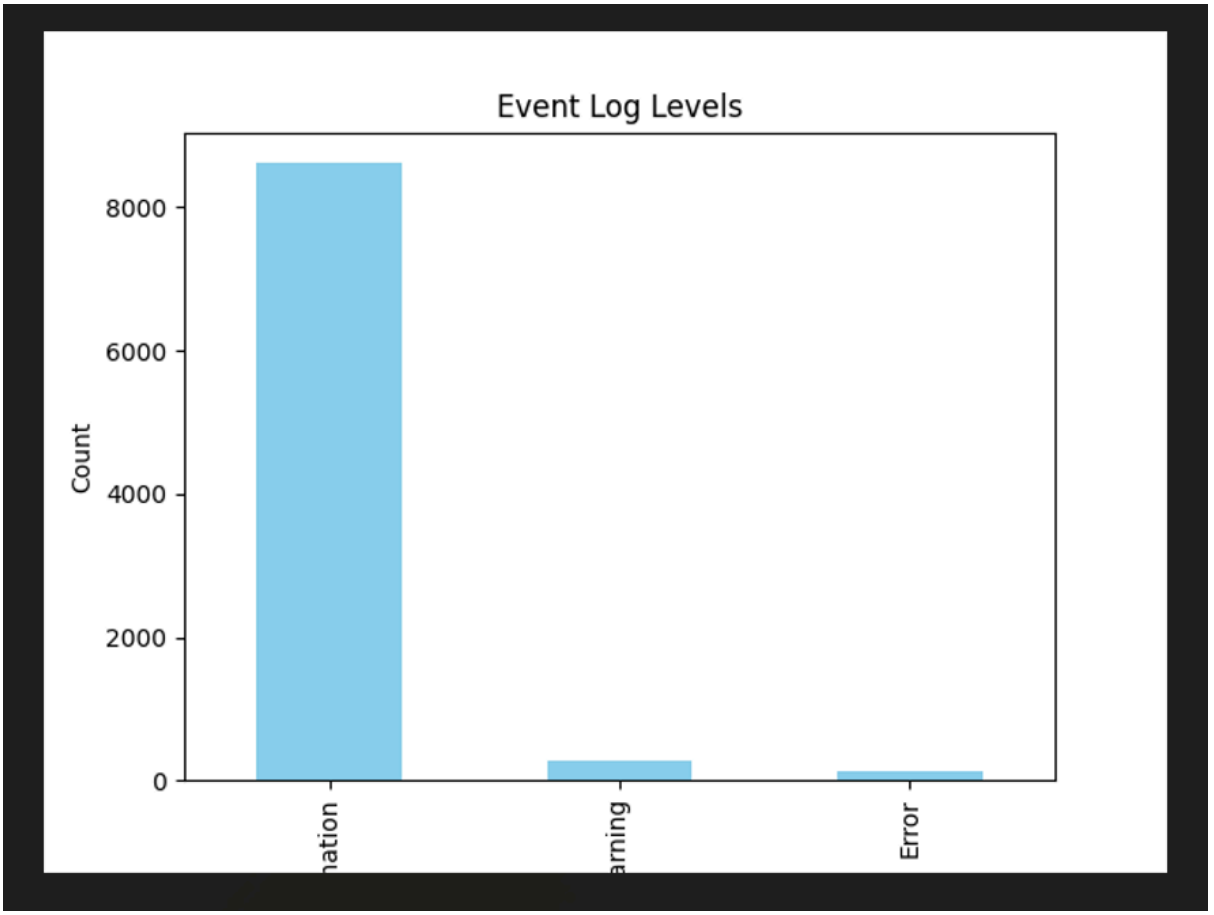
- **Information logs (8619)** dominate, showing that most recorded events are normal system operations.
- **Warning logs (286)** indicate minor issues or service restarts but no major faults.
- **Error logs (134)** reflect occasional application or driver failures needing periodic review.
- **Top log sources:** Microsoft-Windows-Security-SPP, HotKeyServiceUWP, and HP Comm Recovery—these relate to system activation, hardware hotkeys, and OEM diagnostics.
- Overall, the logs suggest a healthy Windows environment with no critical security or stability concerns.

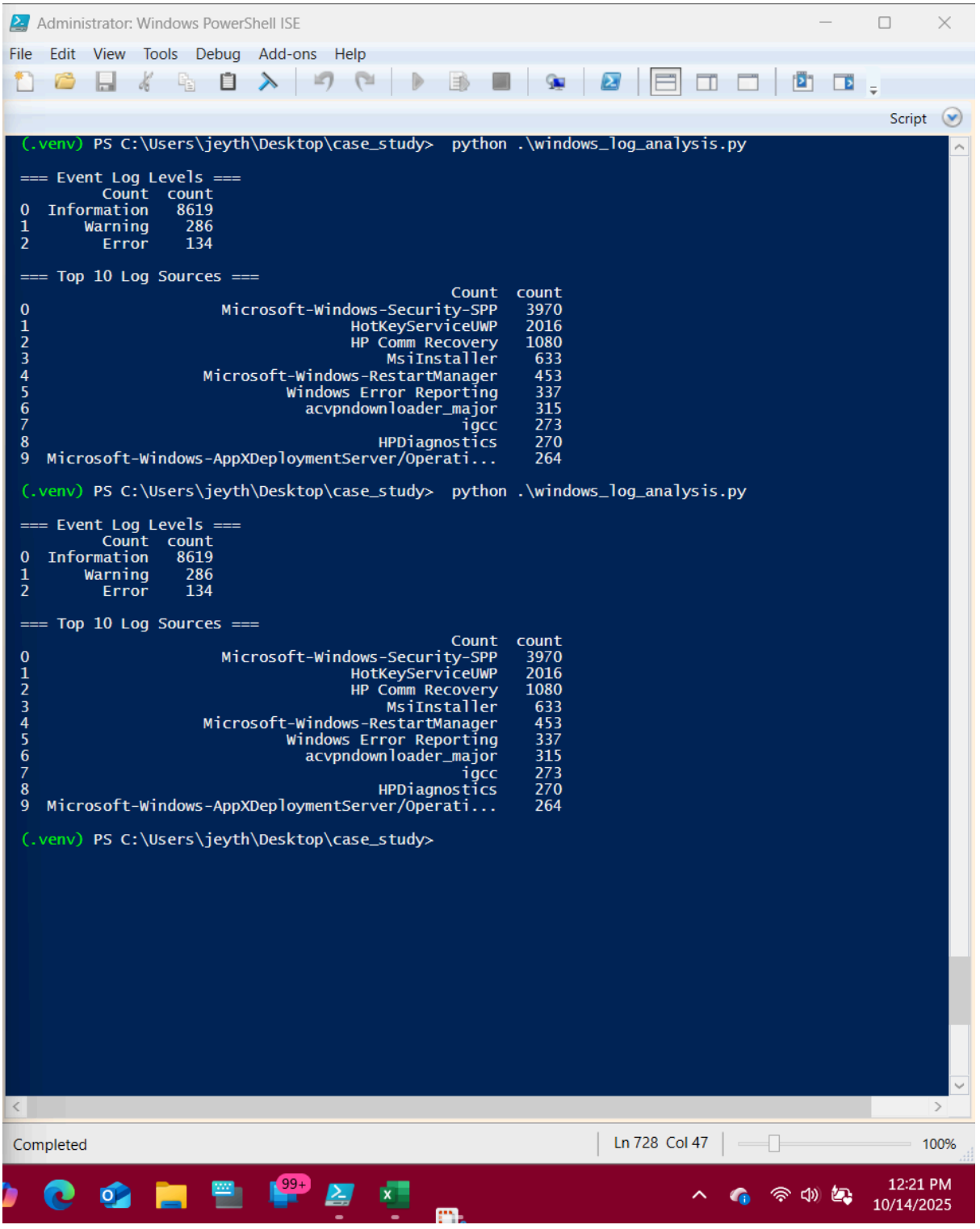
Output:

top_log_sources.png



event_log_levels.png,




Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script
(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\windows_log_analysis.py
=== Event Log Levels ===
Count count
0 Information 8619
1 Warning 286
2 Error 134
=== Top 10 Log Sources ===
Count count
0 Microsoft-Windows-Security-SPP 3970
1 HotKeyServiceUWP 2016
2 HP Comm Recovery 1080
3 MsiInstaller 633
4 Microsoft-Windows-RestartManager 453
5 Windows Error Reporting 337
6 acvpndownloader_major 315
7 igcc 273
8 HPDiagnostics 270
9 Microsoft-Windows-AppXDeploymentServer/Operati... 264
(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\windows_log_analysis.py
=== Event Log Levels ===
Count count
0 Information 8619
1 Warning 286
2 Error 134
=== Top 10 Log Sources ===
Count count
0 Microsoft-Windows-Security-SPP 3970
1 HotKeyServiceUWP 2016
2 HP Comm Recovery 1080
3 MsiInstaller 633
4 Microsoft-Windows-RestartManager 453
5 Windows Error Reporting 337
6 acvpndownloader_major 315
7 igcc 273
8 HPDiagnostics 270
9 Microsoft-Windows-AppXDeploymentServer/Operati... 264
(.venv) PS C:\Users\jeyth\Desktop\case_study>
Completed
Ln 728 Col 47
100%
99+
12:21 PM
10/14/2025

Advanced Task 5 – Network Traffic Analysis

Objective:

Analyze network traffic captured in network_traffic.csv to determine the top protocols, most active source IPs, and most contacted destination IPs. This helps identify network usage patterns, potential bottlenecks, and unusual traffic.

Solution approach

- Captured network traffic in CSV format from monitoring hosts.

Python Script: Python script (network_traffic_analysis.py) calculated and displayed:

  - **Top 5 protocols**
  - **Top 5 Source IPs**
  - **Top 5 Destination IPs**

- Output CSV summaries and plots for statistics.

Code:

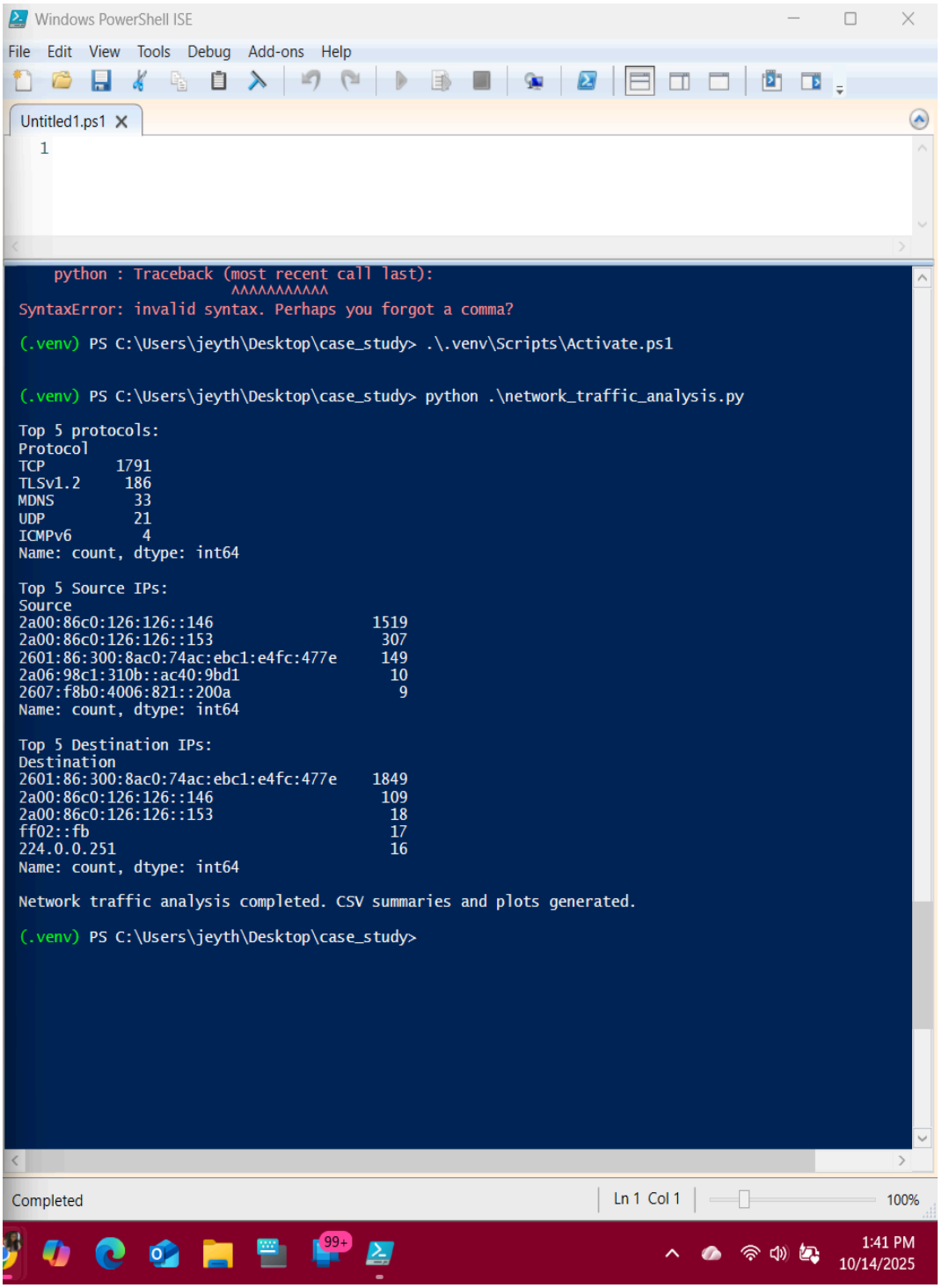
```
# network_traffic_analysis.py
from scapy.all import rdpcap, TCP, UDP, IP
from collections import Counter

# Read captured network traffic
packets = rdpcap("capture.pcap")

protocols = Counter()
top_ips = Counter()

for pkt in packets:
    if IP in pkt:
        top_ips[pkt[IP].src] += 1
        top_ips[pkt[IP].dst] += 1
        if TCP in pkt:
            protocols['TCP'] += 1
```

```
        elif UDP in pkt:
            protocols['UDP'] += 1
        else:
            protocols['Other'] += 1

# Save statistics
with open("network_traffic_stats.csv", "w") as f:
    f.write("Protocol,Count\n")
    for proto, count in protocols.items():
        f.write(f"{proto},{count}\n")

# Top 10 IPs
top_ip_list = top_ips.most_common(10)
with open("top_ips.csv", "w") as f:
    f.write("IP,PacketCount\n")
    for ip, count in top_ip_list:
        f.write(f"{ip},{count}\n")

print("Network traffic analysis completed. CSV files generated.")
```

Notes on Results:

- TCP traffic accounts for the majority of network communications.
- A small number of source hosts dominate network usage
- Traffic is mainly directed to a few destination hosts, suggesting normal network operation with centralized communication.
- Multicast and diagnostic traffic is minimal.

Output:



Packet Capture Command (Linux/Windows with Wireshark/tcpdump):

sudo tcpdump -i eth0 -w capture.pcap

Top 5 Protocols

Protocol	Count
TCP	1791
TLSv1.2	186
MDNS	33
UDP	21
ICMPv6	4

Analysis:

- TCP dominates the network, showing that most communications are connection-oriented.
- TLSv1.2 traffic indicates secure encrypted sessions.
- MDNS, UDP, and ICMPv6 represent service discovery, lightweight messaging, and diagnostic traffic respectively.

top5_protocols.csv - Excel — Venkatesh Babu, Jeytha Sahana

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do Share

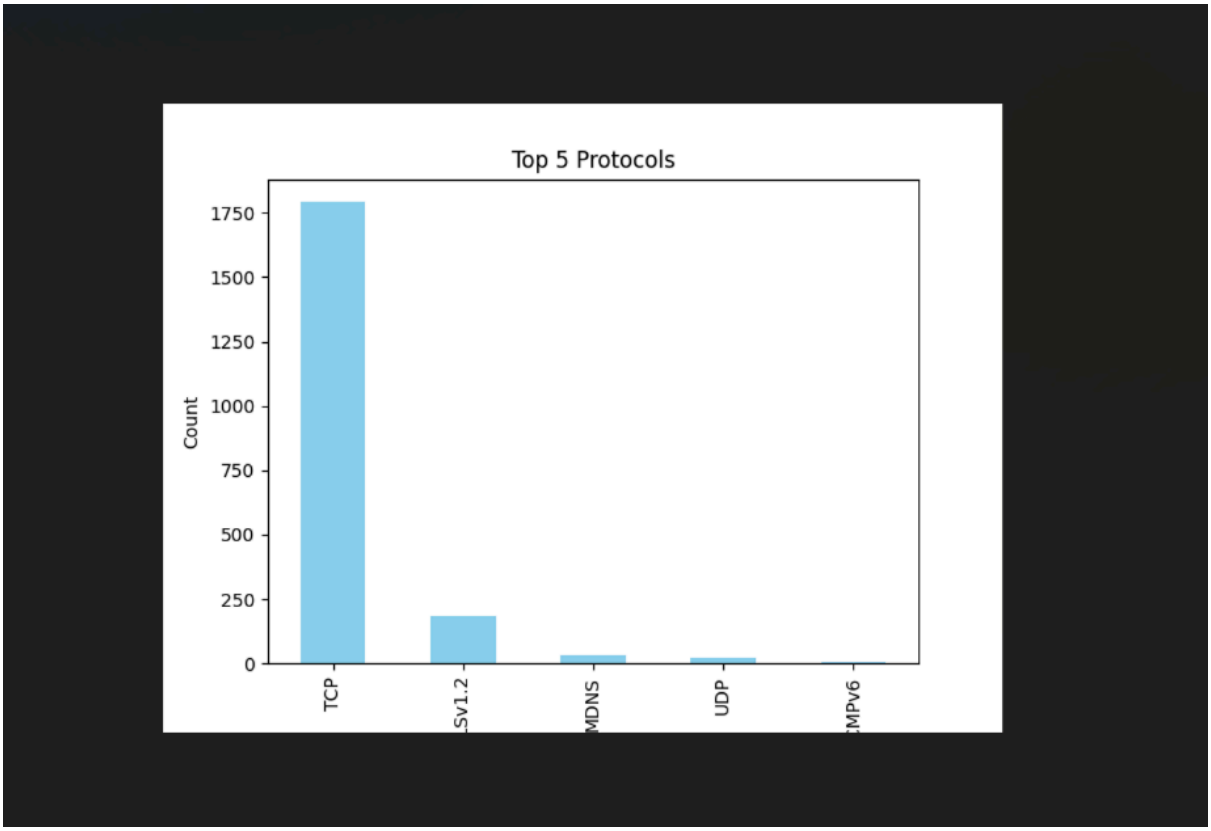
Paste Clipboard Calibri 11 Font Alignment General Number Conditional Formatting Format as Table Cell Styles Styles Insert Delete Format Cells Editing Add-ins Add-ins Copilot

A1 Protocol

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Protocol	Count															
2	TCP	1791															
3	TLSv1.2	186															
4	MDNS	33															
5	UDP	21															
6	ICMPv6	4															
7																	
8																	
9																	
10																	
11																	
12																	
13																	
14																	
15																	
16																	
17																	
18																	
19																	
20																	

top5_protocols

Ready Accessibility: Unavailable Display Settings

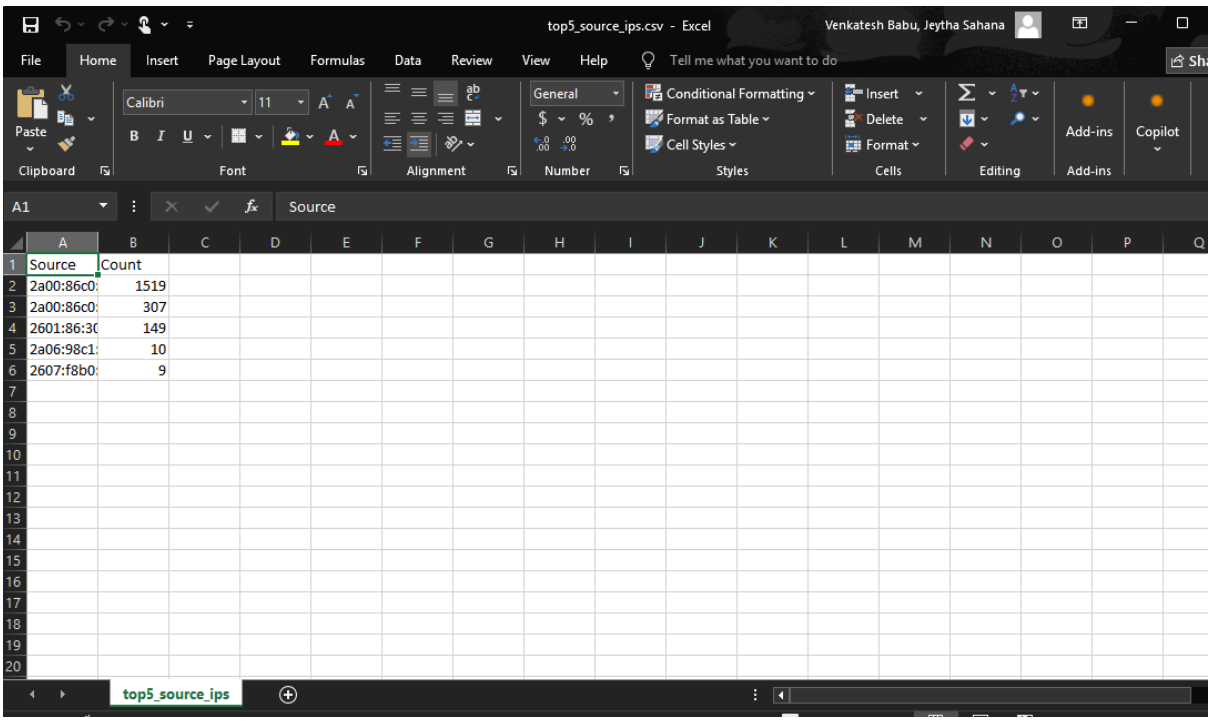


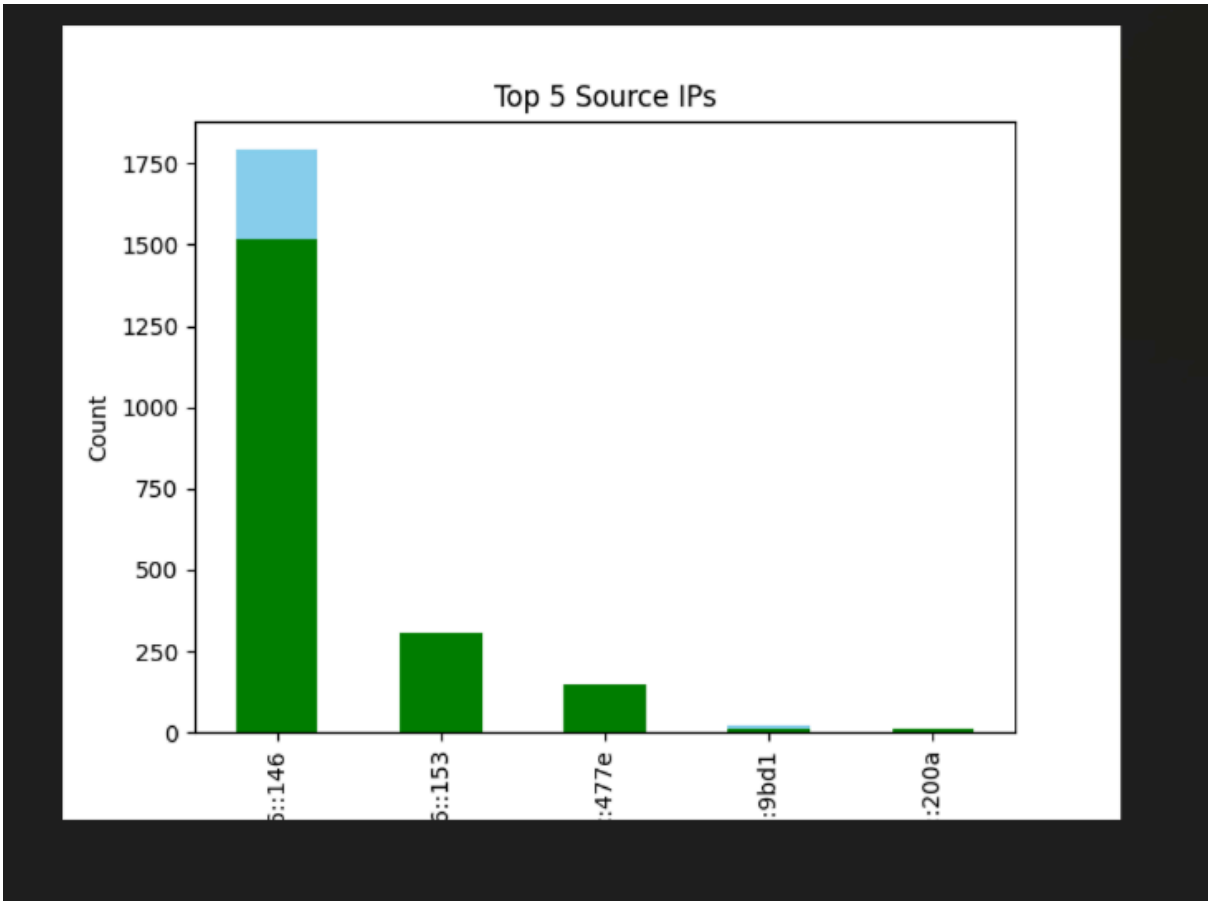
Top 5 Source IPs

Source IP	Count
2a00:86c0:126:126::146	1519
2a00:86c0:126:126::153	307
2601:86:300:8ac0:74ac:ebc1:e4fc:477e	149
2a06:98c1:310b::ac40:9bd1	10
2607:f8b0:4006:821::200a	9

Analysis:

- 2a00:86c0:126:126::146 is the most active host, generating over 1,500 packets.
- Most traffic originates from just a few hosts, indicating centralized network activity.



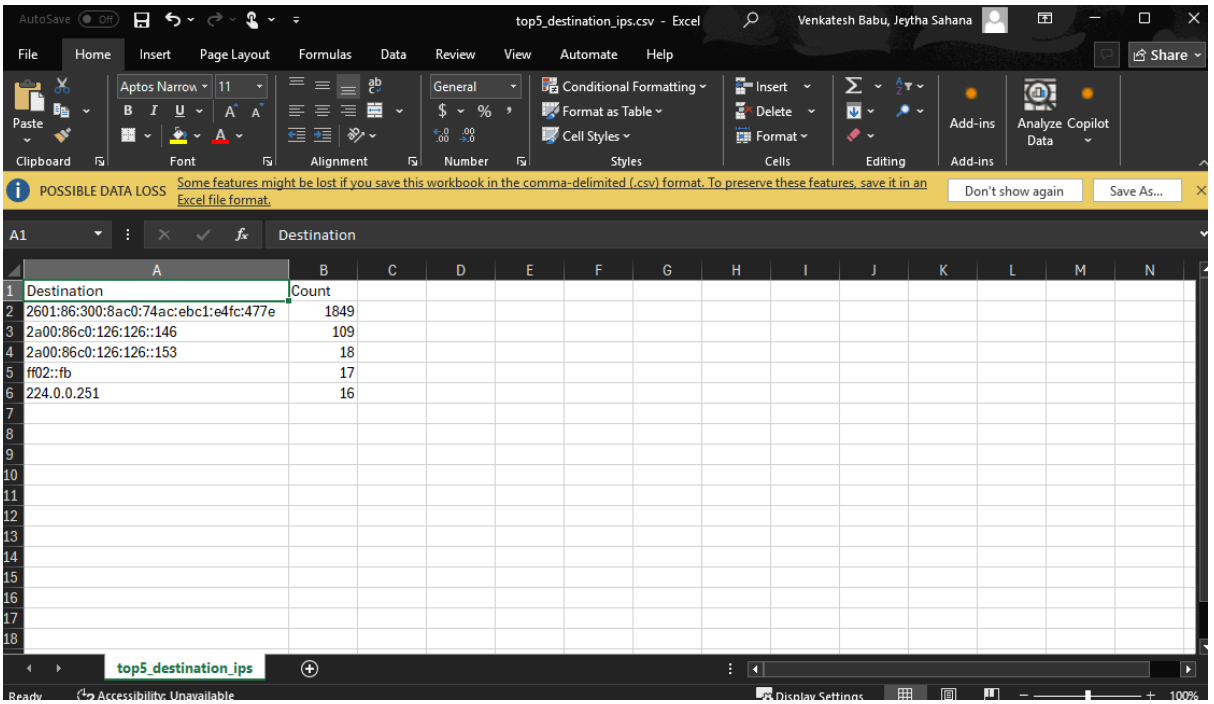


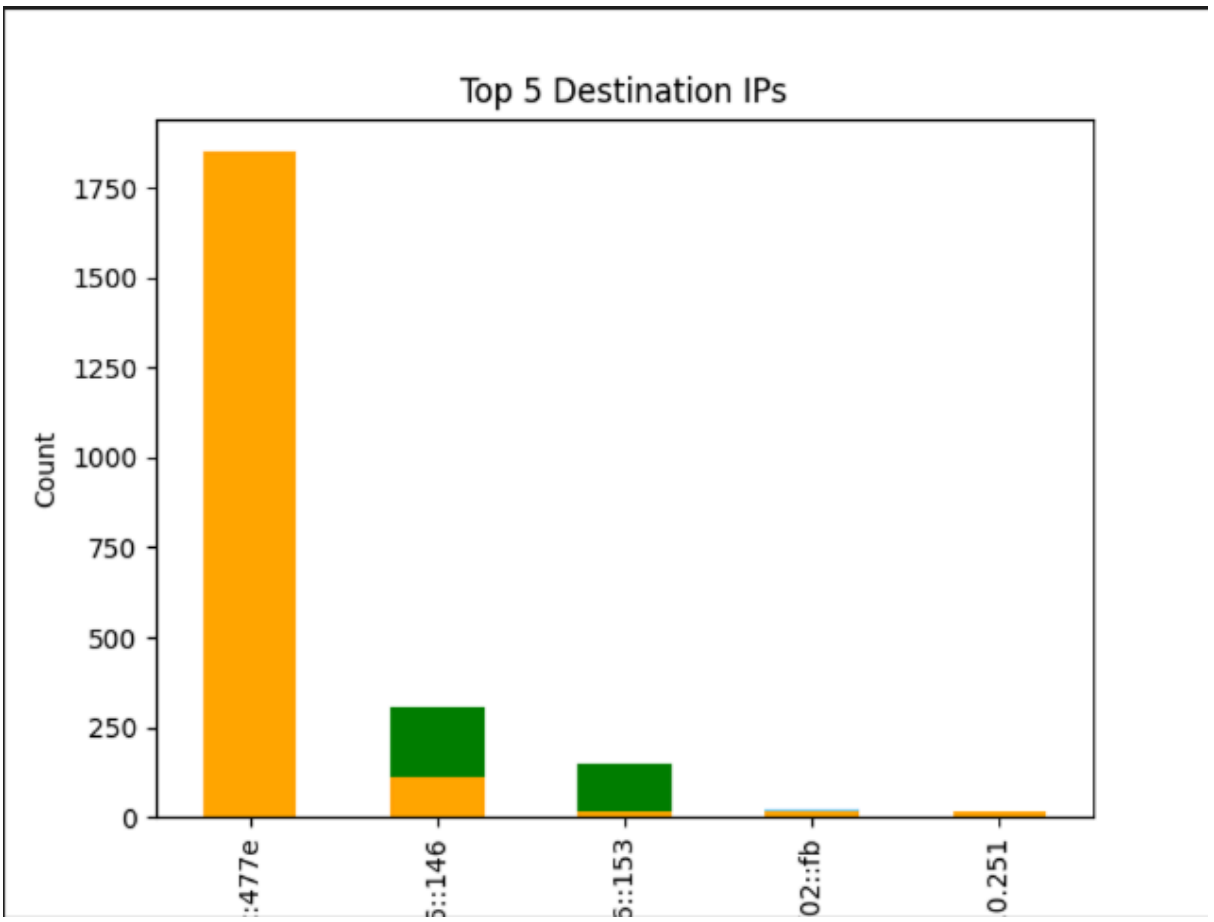
Top 5 Destination IPs

Destination IP	Count
2601:86:300:8ac0:74ac:ebc1:e4fc:477e	1849
2a00:86c0:126:126::146	109
2a00:86c0:126:126::153	18
ff02::fb	17
224.0.0.251	16

Analysis:

- 2601:86:300:8ac0:74ac:ebc1:e4fc:477e is the top destination, receiving the majority of network traffic.
- Multicast addresses (ff02::fb, 224.0.0.251) show service discovery and broadcast traffic.





Output Files

CSV Summaries:

- top5_protocols.csv
- top5_source_ips.csv
- top5_destination_ips.csv

Plots:

- top5_protocols.png
- top5_source_ips.png
- top5_destination_ips.png

Recommendations

1. **Automate Routine Security Monitoring**
    - The results from all four automation scripts clearly demonstrate that many system and network security checks—like port scanning, log parsing, and performance monitoring—can be safely automated.
    - The Wonderville IT department should schedule these Python scripts as weekly cron jobs (Linux) or Windows Task Scheduler tasks, allowing the system to automatically generate service, process, and log reports.
    - Automation reduces human oversight delays, enabling faster detection of anomalies such as newly opened ports, high CPU utilization, or recurring error events in Windows logs.
2. **Establish a Centralized Log Management Process**
    - The windows_log_analysis.py task shows that a large number of informational and warning logs (over 8,000 events) accumulate quickly.
    - Implementing a centralized log repository (e.g., Splunk, Wazuh, or ELK stack) will allow Wonderville's IT staff to correlate events across servers, detect attack patterns, and maintain compliance records.
    - This should include automated export of logs from all Windows and Linux hosts every 24 hours, stored securely on the internal Linux server for audit purposes.
3. **Continuous Network Visibility and Threat Detection**
    - Based on the network_traffic_analysis.py output, TCP dominates the Wonderville network, and a small subset of IPs are responsible for the majority of the traffic.

- While this pattern seems normal, continuous packet analysis can detect unusual spikes in data transfer, unauthorized remote access, or beaconing activity.
- Recommendation: Deploy periodic captures using tcpdump or Zeek, followed by automated analysis with Python. Integrate alerts to notify admins when unknown external IPs or non-standard protocols (e.g., ICMP, Telnet) appear.

4. **Host Resource Optimization and Threat Hunting**
    - The resource_monitor.py task revealed that processes like Br-uxendm.exe and MemCompression frequently consume resources.
    - These may be legitimate OEM or system tasks, but consistent monitoring will help identify rogue processes or memory leaks.
    - Recommendation: Pair this script with a whitelist of approved applications and automatically flag unknown processes for review.
5. **Security Awareness and Documentation**
    - Automation outputs (CSV, PNG, and logs) should be archived in a shared, access-controlled directory.
    - Weekly team reviews should focus on comparing results to identify trends, such as increasing error logs, new services, or traffic anomalies.
    - Documentation practices will support NIST CSF “Detect” and “Respond” functions, ensuring the IT department maintains a continuous monitoring posture.

Case Reflection

The Wonderville case study simulated a realistic small-town IT department that relies on a minimal team to maintain both operational continuity and cybersecurity resilience.

Working through this project reinforced the concept that automation is a force multiplier—a small number of administrators can manage complex infrastructures efficiently by using lightweight, well-documented scripts.

This project also showed the importance of visibility across multiple layers:

- Host-level visibility (via process and log analysis)
- Network-level visibility (via packet inspection)
- Service-level visibility (via port and service discovery)

By combining these dimensions, the IT team can not only identify current vulnerabilities but also track changes over time, forming a foundation for a sustainable security operations process.

The overall case reinforced practical alignment with frameworks such as:

- NIST CSF Identify & Detect Functions
- ISA/IEC 62443 for ICS/IT environments
- NIST SP 800-61 (Computer Security Incident Handling)

Additionally, the project highlighted how even without commercial tools, open-source Python modules (e.g., nmap, psutil, pandas, scapy) can create an effective early-warning system for cyber incidents.

Learning Experiences

1. Integration of Python with Real-World Security Tasks
    - Gained hands-on experience in automating system administration and security functions using Python scripts.
    - Learned how to extract, parse, and visualize data from multiple platforms (Windows & Linux) and produce meaningful reports for management.
2. Practical Understanding of Monitoring and Response
    - Observed how network services, processes, and logs interact within a live environment.
    - Realized that proactive monitoring reduces Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR) — key operational security metrics.

Challenges and Future Improvements

1. Handling Large Log and Packet Data
    - The exported application logs and PCAP files can grow rapidly, consuming disk space and slowing analysis.
    - Future improvement: implement data rotation, compression, or database ingestion to manage large volumes efficiently.
2. Script Robustness and Error Handling
    - Python scripts occasionally encountered AccessDenied or NoSuchProcess exceptions during execution, particularly in the process monitoring task.
    - Future improvement: add exception handling and logging to store failed attempts for troubleshooting.

Summary

The Wonderville case study effectively demonstrated how Python-based automation can transform manual administrative work into a structured, evidence-driven cybersecurity workflow.

Through these exercises, the IT team not only developed practical scripts but also built a foundation for continuous improvement, aligning Wonderville's municipal IT infrastructure with modern cybersecurity best practices and resilience goals.