Intelligent Log “Anomalizer”

Problem Statement:

Production logs do not always contain explicit “ERROR” keywords.

Objective

Build a tool that:

Analyzes word frequency in logs

Identifies rare or unusual patterns

Flags potential anomalies

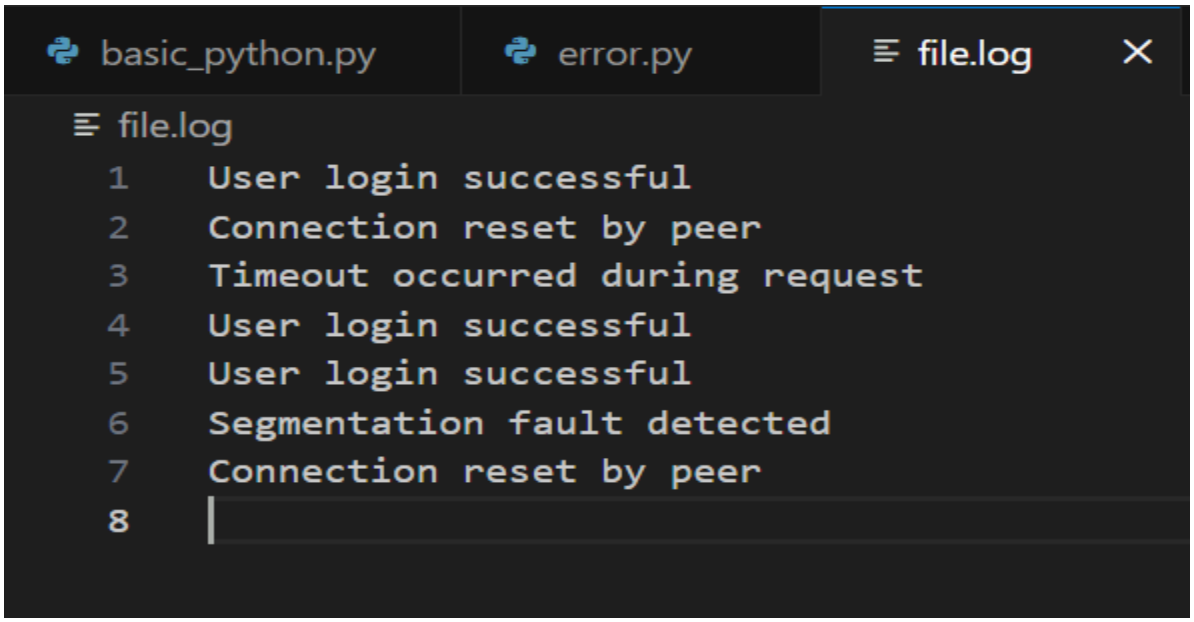
Produces anomaly insights summary

SOLUTION:

What I understand is that instead of just identifying error messages , here we have to identify the word frequency and unusual patterns.

For that :

-First I have created a log file where I have given some example messages.



-Then I have opened the file using with **open(“file_name”)**

-**text = f.read().lower()** is used to read the entire log and **lower()** is used to convert to lower case for consistency.

-**re.findall(r'\b[a-z]+\b', text)** This command is used to extract only words and removes numbers ,symbols etc…

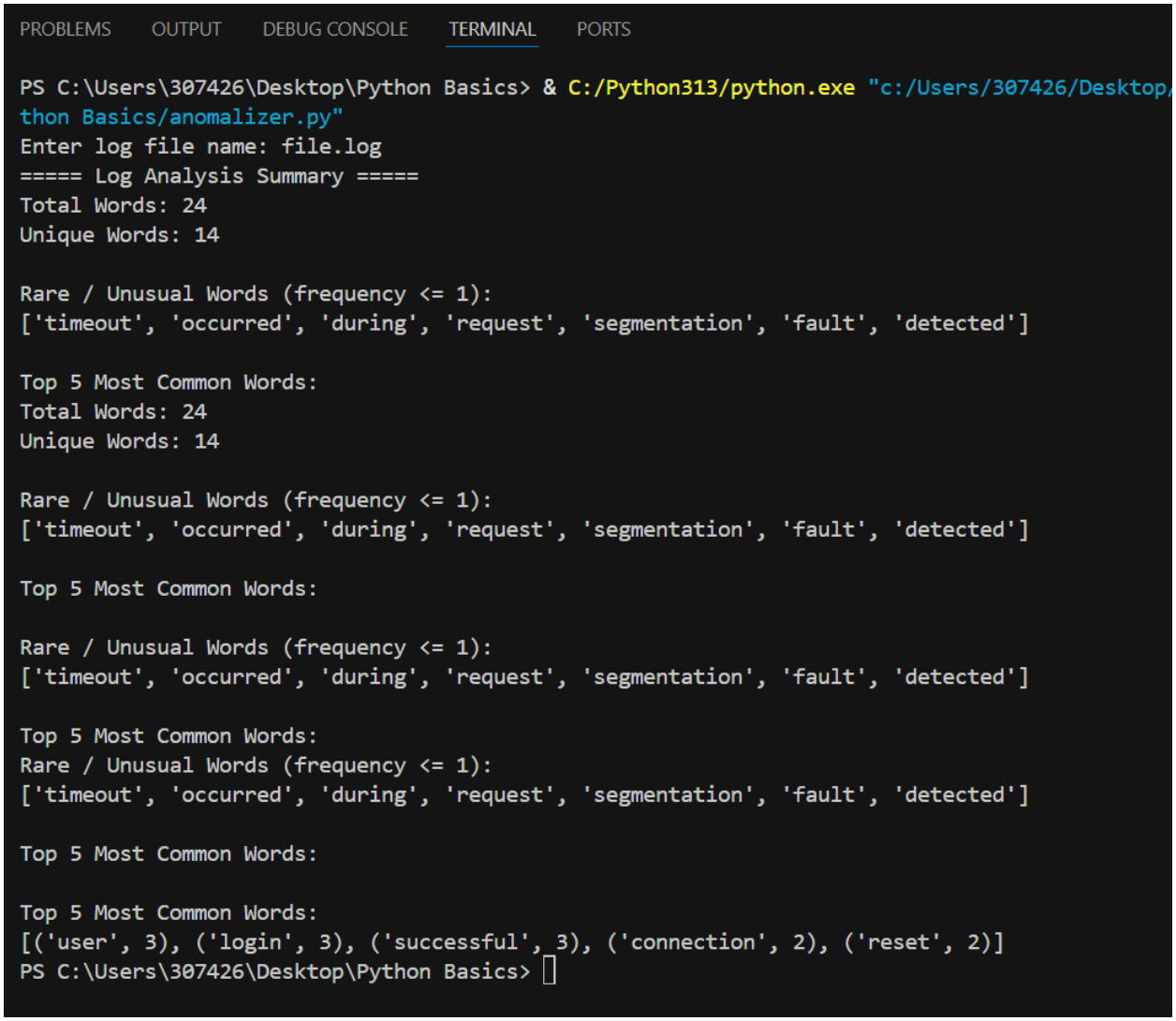
-Later i have calculated the word count by using **if count<=1.**

CODE:

basic_python.py | error.py | file.log | anomalizer.py

anomalizer.py > ...

```python
from collections import Counter
import re

def analyze_log(file_name):
    try:
        with open(file_name, "r") as f:
            text = f.read().lower()

        # Extract only words (remove punctuation)
        words = re.findall(r'\b[a-z]+\b', text)

        # Count word frequency
        word_count = Counter(words)

        print("===== Log Analysis Summary =====")
        print("Total Words:", len(words))
        print("Unique Words:", len(word_count))

        print("\nRare / Unusual Words (frequency <= 1):")
        anomalies = []

        for word, count in word_count.items():
            if count <= 1:
                anomalies.append(word)

        print(anomalies)

        print("\nTop 5 Most Common Words:")
        print(word_count.most_common(5))

    except FileNotFoundError:
        print("Log file not found!")

# Run the analyzer
file_input = input("Enter log file name: ")
analyze_log(file_input)
```



CONCLUSION:

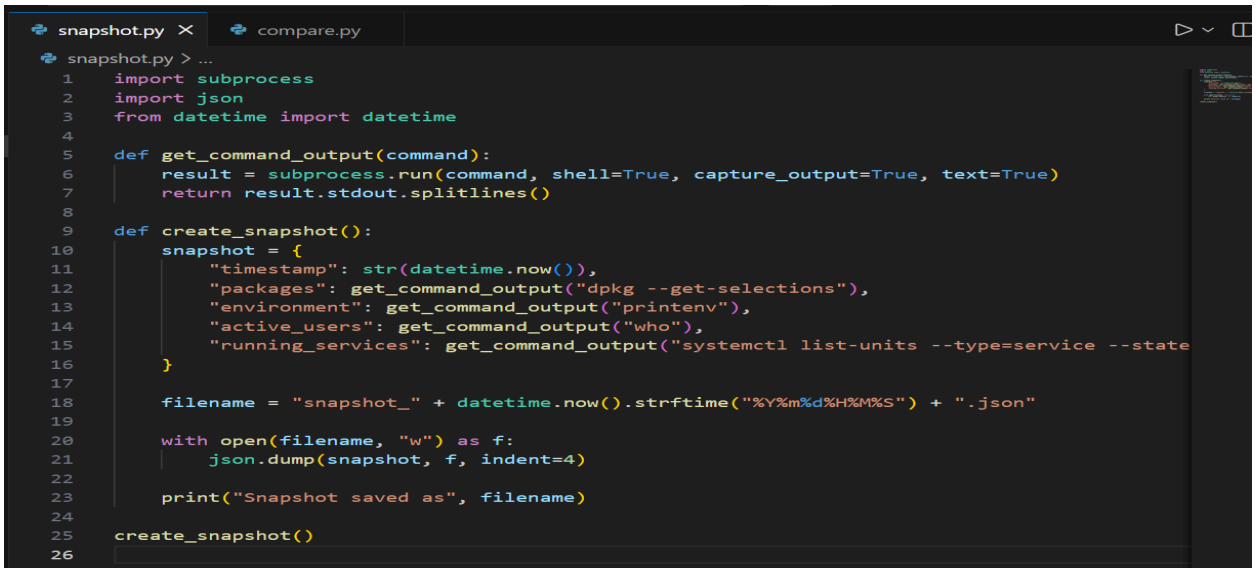
what I understood is that all production files may not explicitly contain error messages. So, by implementing this we can also detect unusual messages which may be an error. Here I have taken all the words by removing numbers and special symbols , and returned the output where the count of the word is 1. So that the least repeated words are returned as output.

SYSTEM SNAPSHOT AND DRIFT DETECTION TOOL

PROBLEM STATEMENT:
Post-maintenance
system behavior changed unexpectedly.
Objective
Build a snapshot tool that captures:
Installed packages
Environment variables
Active users Running services
Then compare snapshots to detect configuration drift.

SOLUTION:
-First I have to capture the snapshot with timestamps.
-Then I have to save it to a file.
-Repeate the first step again which is to take another snapshot with timestamps.
-Comparing both timestamps.
-Reporting the difference.
-**capture_output=True**, this command is used to get the output of the terminal into the python variable.
-**splitlines()**, this converts the output string into a list of lines.
-**json.dump(),** this command stores data in JSON formate.

CODE:
This is the snapshot.py where I have captured first snapshot which is
`snapshot_20260218152609.json`



Here, This is compare.py which is another python file I have created to capture another snapshot with timestamps to compare it with the first one. The snapshot I captured here is `snapshot_20260218152642.json`

snapshot.py | compare.py

compare.py > ...

```python
import json

def compare_snapshots(file1, file2):
    with open(file1) as f1, open(file2) as f2:
        snap1 = json.load(f1)
        snap2 = json.load(f2)

    print("===== DRIFT DETECTION REPORT =====")

    for key in ["packages", "environment", "active_users", "running_services"]:
        set1 = set(snap1[key])
        set2 = set(snap2[key])

        added = set2 - set1
        removed = set1 - set2

        if added:
            print(f"\nNew {key}:")
            for item in added:
                print(" +", item)

        if removed:
            print(f"\nRemoved {key}:")
            for item in removed:
                print(" -", item)

compare_snapshots("snapshot_20260218152609.json", "snapshot_20260218152642.json")
```

OUTPUT:

```
PS C:\Users\307426\Desktop\DriftTool> & C:/Python313/python.exe c:/Users/307426/Desktop/DriftTool/compare.py
FileNotFoundError: [Errno 2] No such file or directory: 'snapshot_1.json'
PS C:\Users\307426\Desktop\DriftTool> & C:/Python313/python.exe c:/Users/307426/Desktop/DriftTool/compare.py
===== DRIFT DETECTION REPORT =====
PS C:\Users\307426\Desktop\DriftTool>
```

The output is empty because there is no difference between the snapshots because I captured both of them at a time.

CONCLUSION:

What I understood here is that it is just like the laptops our company gives to us which are configured and later based on our requirement we have to install different packages. Then there will be a difference when comparing the laptop before and after.

Disk Usage Alert Script

Problem Statement
Production systems run out of disk space without warning.
Objective
Build a shell script that:
Monitors disk usage
Triggers alert when threshold exceeded
Logs alert details
Can run via cron

SOLUTION:
-In production log grows, backups increase etc… where the disk storage may exceed which leads to crash.
-Here I have to monitor the disk
-Checks the disk Usage whether it is exceeded or not.
-If exceeded then return an ALERT message.
-Log the details and automate this using cron.
-**THRESHOLD=80** This command is used to create a variable name where if the threshold exceeds 80 it will trigger an alert.
-**USAGE=$(df -h / | awk 'NR==2 {print $5}' | sed 's/%//')** this command takes dist memory of the root which is the second line fifth column and it removes the % symbol to make it easier to compare.
-**$()** which is used to capture the output and store it in the variable.
-**TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')** This is a timestamp which records the current date and time.
-**if ["$USAGE" -ge "$THRESHOLD"]** this is the condition used to check the disk usage where -ge is greater than 80 where I have taken the threshold as 80.

CODE:

```
Last login: Wed Feb 18 10:28:02 2026 from 18.206.107.29
[ec2-user@ip-172-31-23-19 ~]$ nano disk_monitor.sh
[ec2-user@ip-172-31-23-19 ~]$ ./disk_monitor.sh
 Disk check completed. Usage: 20% (Within safe limit)
[ec2-user@ip-172-31-23-19 ~]$
```

CONCLUSION:

What I understand here is that to check for disk usage and check whether it is exceeded or not. To avoid the system crash due to disk memory usage I have to check the disk usage and have to implement a condition to check whether it is exceeded above the given limit. If it is exceeded print alert if not print the disk memory usage. And use cron to automate it .