## **Assignment: Log File Analyzer**

### **The Scenario**

You’re given a log file (e.g., app.log) containing application logs in this format:

2023-03-01 08:15:27 - ServiceA - INFO - Started processing request #123

2023-03-01 08:15:28 - ServiceB - ERROR - Null pointer exception

2023-03-01 08:15:29 - ServiceA - INFO - Completed request #123 in 2ms

2023-03-01 08:20:05 - ServiceC - WARN - Disk usage is at 85%

...

Each line has the following parts:

1. Date and time (e.g., 2023-03-01 08:15:27)
2. Service name (e.g., ServiceA)
3. Log level (e.g., INFO, ERROR, WARN, etc.)
4. The log message itself

### **Requirements**

1. **Parsing the Log File**
   * Write a script (log\_analyzer.py) that reads a text file, line by line, from the current directory (e.g., app.log).
   * Extract the four pieces of information from each line:
     + timestamp
     + service\_name
     + log\_level
     + message
   * Handle malformed lines gracefully (e.g., lines that don’t fit the expected pattern). Log them or skip them with a meaningful warning.
2. **Data Aggregation & Analysis**
   * Tally the number of log entries by **log level** (e.g., INFO, ERROR, WARN)
   * Tally the number of log entries by **service** (e.g., ServiceA, ServiceB)
   * Identify the **most common ERROR message** (by exact string match)
3. **Outputs**
   * Print or output a short summary that includes:
     + How many lines were **INFO**, **ERROR**, **WARN**, etc.
     + How many lines came from each **service**
     + The **most common ERROR message** (and how many times it appeared)
   * Format the summary in a readable way (console, CSV, or JSON—your choice).
4. **Error Handling**
   * Assume the file might contain unexpected/invalid lines (wrong format, missing fields).
   * Your script shouldn’t crash; it should log or report an error about those lines and move on.
5. **Code Structure**
   * Demonstrate reasonable Python structure (functions and/or classes as needed).
   * Include docstrings or comments to explain what each function does, what inputs it expects, and what outputs it returns.
6. **Bonus (Optional)**
   * Add a function that, given a date/time range (e.g., start and end), filters the log entries and returns only the lines in that range. This can be used to produce a filtered summary.
   * Provide one or two **unit tests** (e.g., using unittest or pytest) that confirm your parsing logic is correct.

### **Deliverables**

1. **The Script** (log\_analyzer.py), runnable from the command line.
2. **A Sample Log File** (a small app.log with at least a few dozen lines) to test against.
3. **Output** (printed summary, CSV, or JSON) showing aggregated log stats.
4. **Short Documentation** or docstrings explaining how to run and what assumptions were made.

Input Data

2023-03-01 08:15:27 - ServiceA - INFO - Started processing request #123

2023-03-01 08:15:28 - ServiceB - ERROR - Null pointer exception

2023-03-01 08:15:29 - ServiceA - INFO - Completed request #123 in 2ms

2023-03-01 08:20:05 - ServiceC - WARN - Disk usage is at 85%

2023-03-01 08:35:10 - ServiceB - ERROR - Null pointer exception

2023-03-01 08:40:05 - ServiceA - INFO - Cleaned up temporary files

2023-03-01 08:44:11 - ServiceD - INFO - Heartbeat check

2023-03-01 09:00:00 - ServiceB - INFO - Started job X

2023-03-01 09:00:01 - ServiceB - ERROR - Job X failed to start

2023-03-01 09:05:30 - ??? - INFO - Malformed line

2023-03-01 09:10:00 - ServiceC - WARN - Low memory

2023-03-01 - ServiceA - ERROR - Missing timestamp detail

2023-03-01 09:11:00 - ServiceE - DEBUG - This is a debug message