Drift Detection Project: Deep Discussion & Brainstorming

1. Project Motivation & Objectives

- Log drift detection is crucial for maintaining system reliability and early anomaly detection.
- Real-world implications include preventing outages, identifying security breaches, and improving system monitoring.
- Objective: Explore and characterize multiple log datasets to inform robust drift detection strategies.

Individual Dataset Deep Dives

Mac Logs

- **Structure:**
  - Typically use a syslog format: `Month Day Time Host Process[PID]: Message`
  - Entries are relatively short, with consistent timestamping.
- **Message Types:**
  - High proportion of `INFO` and `WARNING` messages, with fewer `ERROR` or `CRITICAL` entries.
  - Many logs relate to system daemons, user logins, and background processes.
- **Temporal Patterns:**
  - Activity peaks during user login/logout times and system updates.
  - Some periodicity, but also bursts during system events (e.g., reboots).
- **Components:**
  - Frequent references to system daemons (e.g., `launchd`, `kernel`, `WindowServer`).
  - Some logs tied to user applications, but most are system-level.
- **Notable Findings:**
  - Mac logs are relatively "clean" and structured, but can be verbose.

    - Drift may be indicated by sudden increases in error messages or new/unusual daemons appearing.

Apache Logs

- **Structure:**
    - Standard web server log format: `IP – – [timestamp] "METHOD URL PROTOCOL" status size`
    - Highly structured, easy to parse.
- **Message Types:**
    - Dominated by `INFO` (normal requests), with occasional `ERROR` (failed requests, 404s, etc.).
    - Can include `WARNING` for configuration or access issues.
- **Temporal Patterns:**
    - Clear daily and weekly cycles, with peaks during business hours.
    - Bursts of errors may correspond to attacks, misconfigurations, or outages.
- **Components:**
    - Components are typically URLs, endpoints, or client IPs.
    - Can analyze which endpoints are most/least active or error-prone.
- **Notable Findings:**
    - Apache logs are ideal for drift detection due to their regularity and volume.
    - Drift may be seen as changes in request patterns, error rates, or new endpoints.

HealthApp Logs

- **Structure:**
    - Logs are semi-structured, often in JSON or key-value format.
    - Entries may include user ID, event type, timestamp, and device metadata.
- **Message Types:**
    - Common types include `INFO` (user actions, app events), `WARNING` (sync issues, minor errors), and occasional `ERROR` (crashes, failed operations).
    - Some logs may include custom event types (e.g., `HEALTH_EVENT`, `SYNC_START`).
- **Temporal Patterns:**
    - Usage peaks during morning and evening hours, reflecting user activity patterns.
    - Periodic spikes may correspond to app updates or health campaigns.
- **Components:**

    - Components often refer to app modules (e.g., `SyncManager`, `ActivityTracker`, `NotificationService`).
    - Device types and OS versions may also appear as components.
- **Notable Findings:**
    - HealthApp logs provide rich user and device context, but can be noisy due to frequent background events.
    - Drift may be indicated by changes in event frequency, new event types, or increased error rates after app updates.

BGL Logs

- **Structure:**
    - Logs are typically structured with fields for timestamp, node ID, event type, and message.
    - Entries often originate from high-performance computing (HPC) clusters, with consistent formatting.
- **Message Types:**
    - Common types include `INFO` (system status, job events), `ERROR` (hardware/software failures), and `WARNING` (resource issues, threshold breaches).
    - Some logs may include `DEBUG` or custom HPC event types.
- **Temporal Patterns:**
    - Activity often follows job scheduling cycles, with bursts during job start/end times.
    - Error and warning spikes may align with hardware failures or maintenance windows.
- **Components:**
    - Components typically refer to compute nodes, storage systems, or job schedulers (e.g., `Node42`, `LustreFS`, `PBS`).
    - Can analyze which nodes or systems are most error-prone.
- **Notable Findings:**
    - BGL logs are highly structured and suitable for automated parsing.
    - Drift may be indicated by changes in error rates on specific nodes, or new types of system events.

HPC Logs

- **Structure:**
    - Logs are structured, often with fields for timestamp, node ID, job ID, event type, and

message.
    - Entries are generated by various subsystems (scheduler, compute nodes, storage, etc.).
- **Message Types:**
    - Includes `INFO` (job status, resource allocation), `WARNING` (resource contention, threshold warnings), and `ERROR` (job failures, hardware faults).
    - May include `DEBUG` for troubleshooting and system diagnostics.
- **Temporal Patterns:**
    - Log activity is closely tied to job scheduling and execution cycles.
    - Peaks in log volume during batch job submissions, completions, or system maintenance.
- **Components:**
    - Components include compute nodes, storage systems, job schedulers, and network interfaces.
    - Can identify problematic nodes or subsystems by error frequency.
- **Notable Findings:**
    - HPC logs are rich in operational detail and suitable for fine-grained drift analysis.
    - Drift may manifest as increased error rates on specific nodes, or shifts in job scheduling patterns.

Linux Logs

- **Structure:**
    - Logs typically follow the syslog format: `Month Day Time Host Process[PID]: Message`.
    - Entries are generated by a wide range of system and application processes.
- **Message Types:**
    - Includes `INFO` (routine operations), `WARNING` (potential issues), `ERROR` (failures), and `CRITICAL` (system-level problems).
    - Some logs may include `DEBUG` for troubleshooting.
- **Temporal Patterns:**
    - Log activity reflects system usage patterns, with peaks during boot, shutdown, and scheduled tasks.
    - Error spikes may correspond to failed services or hardware issues.
- **Components:**

    - Components include system daemons (e.g., `systemd`, `sshd`), user applications, and kernel modules.
    - Can analyze which processes or services are most active or error-prone.
- **Notable Findings:**
    - Linux logs are comprehensive and can be verbose, capturing a wide range of system events.
    - Drift may be indicated by new error types, changes in service activity, or increased warnings/errors after updates.

HDFS Logs

- **Structure:**
    - Logs are highly structured, often with fields for timestamp, log level, component, and message.
    - Entries follow a consistent format, making them suitable for automated parsing.
- **Message Types:**
    - Includes `INFO` (routine operations, block reports), `WARNING` (potential issues), `ERROR` (failures, lost blocks), and `DEBUG` (detailed diagnostics).
    - High frequency of block-related events and system status updates.
- **Temporal Patterns:**
    - Log activity reflects HDFS operations such as file reads/writes, block replication, and system maintenance.
    - Error and warning spikes may correspond to hardware failures, network issues, or cluster rebalancing.
- **Components:**
    - Components include `NameNode`, `DataNode`, `FSNamesystem`, `DFSClient`, and block IDs (e.g., `blk_12345`).
    - Can analyze which components are most active or error-prone.
- **Notable Findings:**
    - HDFS logs are rich in operational detail and ideal for drift and anomaly detection.
    - Drift may be indicated by changes in block failure rates, new error types, or shifts in component activity patterns.

Discussion: Insights from Log Data Analysis

Diversity and Complexity

The log datasets in this project span a wide range of systems: personal computers (Mac, Linux), web servers (Apache), distributed storage (HDFS), high-performance computing (BGL, HPC), and mobile applications (HealthApp).

- **Implications:** This diversity means that no single parsing or analysis approach fits all datasets. It also provides a unique opportunity to compare how drift manifests in different environments—user-driven (HealthApp), system-driven (Linux/Mac), and service-driven (Apache/HDFS/BGL/HPC).
- **Example:** HealthApp logs may reflect user behavior changes (e.g., new app features), while HDFS logs may show drift due to hardware upgrades or failures.

Structure and Parsability

- **Highly structured logs** (Apache, HDFS, BGL, HPC):
  - These logs have well-defined fields (timestamp, log level, component, message), making them easy to parse and analyze programmatically.
  - **Advantage:** Enables automated feature extraction and large-scale analysis.
  - **Limitation:** May miss nuanced or contextual information present in free-form logs.
- **Semi-structured/free-form logs** (HealthApp, Linux, Mac):
  - These may use JSON, key-value pairs, or plain text, and can vary in format even within the same dataset.
  - **Advantage:** Can capture richer context (e.g., user actions, device metadata).
  - **Limitation:** Require more preprocessing and custom parsing logic.

Message Types and System Health

- **Error and warning messages** are direct indicators of system health.
  - **Frequent errors** in HDFS or BGL may signal hardware issues, misconfigurations, or impending failures.
  - **Warning spikes** can serve as early warnings before critical failures.
- **Info/debug messages** provide baseline activity.
  - **Sudden drops** in info messages or **surges** in debug logs may indicate system

reboots, outages, or troubleshooting efforts.

- **Implication for drift:** Monitoring the ratio and frequency of message types can help detect both gradual and abrupt drift.

Temporal Patterns

- **Cyclic patterns** (daily/weekly):
  - Seen in Apache and HealthApp logs, reflecting user or business activity cycles.
  - **Drift detection:** Deviations from these cycles (e.g., a sudden drop in traffic) may indicate drift.
- **Bursts of activity/errors:**
  - Often correspond to system events (e.g., updates, attacks, failures).
  - **Example:** A burst of errors in HDFS during a cluster rebalance.
- **Implication:** Temporal features (event counts per hour/day, time between errors) are crucial for drift detection.

Component Analysis

- **Component diversity:**
  - Some logs (HDFS, BGL) are dominated by a few components (e.g., NameNode, specific nodes), while others (Linux, Mac) have a wide range of active processes.
- **Component-level drift:**
  - A sudden increase in errors from a specific node or service can indicate localized drift or failure.
- **Example:** In BGL, if Node42 starts generating more errors than usual, it may be failing or misconfigured.

Comparative Insights

- **Structured logs** are easier for automated drift detection but may lack context.
- **System/application logs** provide richer context but require more sophisticated analysis.
- **Cross-dataset comparison:** Helps identify universal drift indicators (e.g., error spikes) and dataset-specific patterns.

Challenges

- **Data quality:** Missing, corrupted, or inconsistent entries are common, especially in large-scale systems.
  - **Mitigation:** Implement robust preprocessing and validation steps.
- **Format heterogeneity:** Each dataset may require custom parsing.
  - **Mitigation:** Develop modular, extensible parsing functions.
- **Volume and velocity:** High-volume logs (Apache, HDFS) require scalable analysis methods (e.g., batch processing, streaming).

Implications for Drift Detection

- **Feature selection:** Temporal features, error/warning rates, and component activity are promising.
- **Drift types:** Both covariate drift (input distribution changes) and prior drift (event frequency changes) are likely.
- **Early warning:** Monitoring for sudden spikes in errors, new message types, or shifts in component activity can provide early detection.

Open Questions

1. Which datasets are most/least challenging for drift detection, and why?

Most Challenging:

- **HealthApp Logs:**
  - Highly variable structure (JSON, key-value, free text).
  - User-driven events introduce non-stationarity unrelated to system drift (e.g., new app features, user behavior changes).
  - Frequent background events can create noise, making it hard to distinguish true drift from normal variability.
- **Linux/Mac Logs:**
  - Enormous diversity in processes and event types.
  - Free-form messages and inconsistent formatting complicate automated parsing and feature extraction.
  - System updates or user actions can cause abrupt, non-drift-related changes.

Least Challenging:

- **Apache, HDFS, BGL, HPC Logs:**
  - Highly structured, with consistent fields for timestamp, component, and message type.
  - System-driven events are more predictable, and drift is more likely to be operational (e.g., hardware failures, configuration changes).
  - Easier to automate feature extraction and apply statistical drift detection methods.

Summary:
Structured, system-driven logs are generally easier for drift detection, while user-driven or free-form logs require more sophisticated preprocessing and context-aware analysis.

2. Are there universal features or patterns that signal drift across all systems?

Universal Features:

- **Error/Warning Rate:**
  - Sudden increases in error or warning messages are a strong indicator of drift or anomalies in all systems.
- **Temporal Patterns:**
  - Deviations from established daily/weekly cycles (e.g., traffic drops, error bursts) often signal drift.
- **Component Activity:**
  - Shifts in which components are most active or error-prone can indicate drift, regardless of system type.

Universal Patterns:

- **Bursts of Errors:**
  - Across all datasets, bursts of errors or warnings often correspond to drift events (e.g., failures, attacks, updates).
- **Emergence of New Message Types:**
  - New or rare message types appearing more frequently can signal changes in system behavior.

Caveat:

While these features are broadly applicable, their interpretation must be context-aware (e.g., a spike in HealthApp errors after an update may be expected, not drift).

3. How can we best visualize and communicate drift to technical and non-technical stakeholders?

For Technical Stakeholders:

- **Time Series Plots:**
  - Show error/warning rates, event counts, or feature values over time to highlight deviations.
- **Heatmaps:**
  - Visualize component activity or error rates across nodes/services.
- **Anomaly Markers:**
  - Overlay detected drift points or anomalies on plots for clarity.

For Non-Technical Stakeholders:

- **Summary Dashboards:**
  - Use simple charts (bar, pie, line) to show trends and highlight periods of concern.
- **Narrative Explanations:**
  - Accompany visuals with plain-language summaries explaining what drift means and its potential impact.
- **Traffic Light Indicators:**
  - Use color-coded signals (green/yellow/red) to indicate system health or drift status.

Best Practices:

- **Contextualize:**
  - Always relate detected drift to business or operational impact.
- **Automate Reporting:**
  - Set up regular, automated reports or dashboards to keep stakeholders informed.

2. Dataset Description

- **Datasets used:** HDFS, Apache, HealthApp, BGL, HPC, Linux, Mac
- Each dataset represents logs from different systems/environments, with varying formats

and semantics.
- Challenges:
  - Heterogeneous formats and log structures
  - Varying log volume and density
  - Potential missing or corrupted entries

3. Exploratory Data Analysis (EDA)

- **Key characteristics explored:**
  - Entry structure and length (min, max, mean, distribution)
  - Message types (Error, Warning, Info, Debug, Critical)
  - Temporal patterns (burstiness, periodicity, downtime)
  - Component diversity (which system parts are most/least active)
- **Notable findings:**
  - Some datasets (e.g., HDFS, Apache) have highly structured logs, others are more free-form.
  - Error and warning rates vary widely between datasets.
  - Certain logs show clear daily/weekly cycles, others are more irregular.
  - Some components dominate activity in certain logs (e.g., NameNode in HDFS).
- **Visualizations:**
  - Histograms, boxplots, and density plots for entry length
  - Bar and pie charts for message types
  - Time series and heatmaps for temporal and component analysis

4. Comparative Analysis

- Datasets differ in noise, error rates, and structure.
- Some logs are dominated by a few message types or components; others are more diverse.
- Groupwise comparison highlights which systems are more prone to drift or instability.

5. Feature Engineering & Preprocessing

- Extracted features: message length, word count, numerical value count, special character

count, component ID, timestamp, log level, etc.
- Preprocessing steps: encoding detection, missing value handling, timestamp parsing, normalization.
- Challenges: inconsistent formats, ambiguous timestamps, non-standard component names.

6. Drift Detection Implications

- Expect to see both covariate and prior drift (e.g., changes in error rates, component activity).
- EDA suggests focusing on temporal features and error/warning bursts for drift detection.
- Hypotheses: Sudden spikes in error messages or component transitions may indicate drift events.

7. Limitations & Future Work

- Data quality: some logs have missing/corrupted entries.
- Not all logs have clear structure or consistent fields.
- Future work: deeper feature extraction, anomaly labeling, integration with real-time monitoring.

8. Summary & Key Takeaways

- EDA reveals significant diversity and complexity in log data.
- Key features and patterns identified will guide drift detection modeling.
- Next steps: formalize feature extraction, implement drift detection algorithms, validate on real data.

> **Brainstorming notes:**
>
> - Which dataset is most/least challenging for drift detection?
> - Are there any clear early-warning signals in the logs?
> - How can we best visualize and communicate drift to stakeholders?

Drift Detection Approaches: Discussion and Selection

1. Statistical Tests

- **What:** Compare distributions of features (e.g., error rates, message lengths) over time or between reference and test windows.
- **Examples:** Kolmogorov-Smirnov (KS) test for continuous features, Chi-squared test for categorical features (e.g., message types).
- **Pros:** Simple, interpretable, and fast. No need for labeled data.
- **Cons:** Sensitive to window size and data volume. May not capture complex, multivariate drift.

2. Time Series Analysis

- **What:** Model feature trends over time and detect anomalies or regime changes.
- **Examples:** Change point detection (e.g., ruptures, Bayesian methods), Control charts (e.g., CUSUM, EWMA).
- **Pros:** Good for detecting abrupt or gradual changes. Can handle temporal dependencies.
- **Cons:** Requires careful parameter tuning. May be less effective for highly irregular or bursty logs.

3. Machine Learning-Based Methods

- **What:** Use models to learn normal behavior and flag deviations.
- **Examples:** Unsupervised (autoencoders, clustering, isolation forests), Supervised (train a classifier to distinguish between reference and test periods).
- **Pros:** Can capture complex, multivariate drift. Flexible and powerful.
- **Cons:** Requires more data and computation. May be harder to interpret.

4. Domain-Specific/Hybrid Approaches

- **What:** Combine statistical, time series, and domain knowledge (e.g., known maintenance windows, expected cycles).

- **Examples:** Custom rules for error bursts, new component appearance, or rare event types.
- **Pros:** Tailored to your data and use case. Can leverage insights from EDA.
- **Cons:** May require more manual effort and tuning.

Considerations for This Project

- Data variety: Both highly structured and semi-structured logs.
- Feature types: Categorical (message types, components) and continuous (message length, error rates, time between events).
- Goal: Detect both abrupt and gradual drift, and provide interpretable results for reporting.

Suggested Approach

1. Start with statistical tests and time series analysis for key features (error rates, message types, component activity, etc.).
2. Visualize detected drift points and compare with known events (if available).
3. Optionally, experiment with ML-based methods if you want to capture more complex drift or have time/resources.

Next Steps: Outlining the Feature Extraction and Drift Detection Pipeline

1. Data Ingestion & Preprocessing

- Load raw log files for each dataset.
- Detect and handle encoding issues.
- Parse logs into structured format (extract fields: timestamp, log level, component, message, etc.).
- Handle missing or corrupted entries (e.g., drop, impute, or flag).

2. Feature Extraction

- **Basic Features:**
    - Message length (characters, words)

    - Log level/message type (INFO, ERROR, etc.)
    - Component/service name
    - Timestamp (parsed to datetime)
- **Derived Features:**
    - Error/warning rate (per time window)
    - Event count per time window (hourly, daily, etc.)
    - Unique component count per window
    - Time between events
    - Frequency of new/unseen message types or components
- **Aggregation:**
    - Aggregate features over sliding or fixed windows (e.g., 1 hour, 1 day)
    - Store as a feature matrix for each dataset

3. Drift Detection

- **Statistical Tests:**
    - Compare feature distributions between reference and test windows (e.g., KS test for continuous, chi-squared for categorical)
    - Flag significant changes as potential drift
- **Time Series Analysis:**
    - Apply change point detection or control charts to key features (e.g., error rate, event count)
    - Visualize and flag detected change points
- **(Optional) Machine Learning-Based Methods:**
    - Train unsupervised models (e.g., autoencoders, clustering) on reference data
    - Score test windows for anomalies/drift
- **Domain-Specific Rules:**
    - Flag bursts of errors, new message types, or sudden component activity as drift

4. Visualization & Reporting

- Plot feature trends and detected drift points over time
- Summarize drift events in tables or dashboards
- Provide both technical and plain-language summaries for stakeholders

5. Evaluation & Iteration

- Compare detected drift points with known events (if available)
- Refine feature extraction, windowing, and detection parameters as needed
- Document findings and update the pipeline for reproducibility

---

Clustering and Advanced Unsupervised Analysis

To further explore and summarize the diversity and structure of our log datasets, we apply several unsupervised clustering and visualization techniques. These methods help us discover natural groupings, identify outliers, and gain deeper insight into the patterns present in the logs.

KMeans Clustering

- **What:** KMeans partitions log entries into a specified number of clusters based on their TF-IDF vector representations.
- **Why:** This method helps us identify the most common types of log messages and group similar entries together. By examining the largest and smallest clusters, we can find both dominant log patterns and rare, potentially anomalous messages.
- **Insights:** Large clusters often correspond to routine or repetitive log entries, while small clusters or outliers may indicate unusual events, errors, or drift.

DBSCAN (Density-Based Clustering)

- **What:** DBSCAN groups log entries based on density, automatically identifying noise points (outliers) that do not belong to any cluster.
- **Why:** Unlike KMeans, DBSCAN does not require specifying the number of clusters and is effective at detecting anomalies and clusters of arbitrary shape.
- **Insights:** Noise points flagged by DBSCAN are strong candidates for anomalies or drift events. The size and number of clusters can reveal the diversity of log content.

Hierarchical Clustering (Agglomerative)

- **What:** Hierarchical clustering builds a tree (dendrogram) of log entry relationships, showing how clusters merge at different similarity thresholds.
- **Why:** This method provides a visual summary of the relationships between log entries and can help us decide on the number of meaningful clusters.
- **Insights:** The dendrogram can reveal subgroups of similar log messages and highlight outliers that merge late in the hierarchy.

t-SNE Visualization

- **What:** t-SNE projects high-dimensional TF-IDF features into two dimensions for visualization.
- **Why:** This technique allows us to visually inspect the separation and structure of clusters, making it easier to spot well-separated groups and outliers.
- **Insights:** Well-separated clusters in the t-SNE plot suggest distinct log message types, while scattered or isolated points may indicate rare or anomalous entries.

By applying these clustering and visualization methods, we aim to:

- Summarize the main types of log messages present in each dataset
- Detect rare or unusual log entries that may correspond to drift or anomalies
- Provide visual and quantitative evidence for the diversity and structure of the logs

These insights will inform our feature engineering and drift detection strategies, and help us communicate findings to both technical and non-technical stakeholders.

Discussion of Clustering Results

After applying multiple clustering and visualization methods to our log datasets, we interpret the results as follows. (This section will be updated with specific findings after running the analyses.)

KMeans Clustering

- **Number and size of clusters:** [To be filled in]
- **Dominant log patterns:** [Describe the most common types of log messages in the largest clusters.]
- **Outliers or small clusters:** [Note any small clusters or outliers and what they represent.]
- **Insights for drift/anomaly detection:** [Discuss how the clusters relate to system behavior or drift.]

DBSCAN Clustering

- **Number of clusters and noise points:** [To be filled in]
- **Nature of noise points:** [Describe any log entries identified as noise/anomalies.]
- **Cluster diversity:** [Comment on the size and content of DBSCAN clusters.]

Hierarchical Clustering (Dendrogram)

- **Cluster relationships:** [Describe the structure of the dendrogram and how clusters merge.]
- **Outliers:** [Note any entries that merge late or appear as singletons.]

t-SNE Visualization

- **Cluster separation:** [Describe how well-separated the clusters are in 2D.]
- **Outliers or rare patterns:** [Note any isolated points or small groups.]

Checklist for Discussion:

- [ ] How many clusters were found? What are their sizes?
- [ ] What do the largest and smallest clusters represent?
- [ ] Did DBSCAN find any noise points (potential anomalies)?
- [ ] What does the dendrogram show about the relationships between log entries?
- [ ] Are clusters well-separated in t-SNE, or is there overlap?
- [ ] Any surprising or interesting findings?

Summary:

- [Summarize the main insights from clustering.]
- [Discuss how these findings will inform feature engineering, drift detection, or further analysis.]

Sample Log Entries from Each Dataset

Below are representative sample entries from each log dataset. These illustrate the diversity of formats, structures, and content we encounter in our analysis:

HDFS sample entries from hdfs.log:

- 081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_38865049064139660 terminating
- 081109 203807 222 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-6952295868487656571 terminating
- 081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864
- 081109 204015 308 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_8229193803249955061 terminating
- 081109 204106 329 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_-6670958622368987959 terminating

Apache sample entries from Apache.log:

- [Thu Jun 09 06:07:04 2005] [notice] LDAP: Built with OpenLDAP LDAP SDK
- [Thu Jun 09 06:07:04 2005] [notice] LDAP: SSL support unavailable
- [Thu Jun 09 06:07:04 2005] [notice] suEXEC mechanism enabled (wrapper: /usr/sbin/suexec)
- [Thu Jun 09 06:07:05 2005] [notice] Digest: generating secret for digest authentication ...
- [Thu Jun 09 06:07:05 2005] [notice] Digest: done

HealthApp sample entries from HealthApp.log:

- 20171223-22:15:29:606|Step_LSC|30002312|onStandStepChanged 3579
- 20171223-22:15:29:615|Step_LSC|30002312|onExtend:1514038530000 14 0 4
- 20171223-22:15:29:633|Step_StandReportReceiver|30002312|onReceive action: android.intent.action.SCREEN_ON
- 20171223-22:15:29:635|Step_LSC|30002312|processHandleBroadcastAction action:android.intent.action.SCREEN_ON
- 20171223-22:15:29:635|Step_StandStepCounter|30002312|flush sensor data

BGL sample entries from BGL.log:

- ◦ 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.363779 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- ◦ 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.527847 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- ◦ 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.675872 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- ◦ 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.823719 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
- ◦ 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.982731 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected

HPC sample entries from HPC.log:

- 460903 resourcemgmtdaeomon node-25 server subsys 1145552216 1 failed to configure resourcemgmt subsystem err = 10
- 460919 resourcemgmtdaeomon node-25 server subsys 1145552221 1 failed to configure resourcemgmt subsystem err = 10
- 460932 resourcemgmtdaeomon node-25 server subsys 1145552226 1 failed to configure resourcemgmt subsystem err = 10
- 460949 resourcemgmtdaeomon node-25 server subsys 1145552232 1 failed to configure resourcemgmt subsystem err = 10
- 460971 resourcemgmtdaeomon node-25 server subsys 1145552237 1 failed to configure resourcemgmt subsystem err = 10

Linux sample entries from Linux.log:

- Jun 9 06:06:20 combo syslogd 1.4.1: restart.

- Jun 9 06:06:20 combo syslog: syslogd startup succeeded
- Jun 9 06:06:20 combo syslog: klogd startup succeeded
- Jun 9 06:06:20 combo kernel: klogd 1.4.1, log source = /proc/kmsg started.
- Jun 9 06:06:20 combo kernel: Linux version 2.6.5-1.358 (bhcompile@bugs.build.redhat.com) (gcc version 3.3.3 20040412 (Red Hat Linux 3.3.3-7)) #1 Sat May 8 09:04:50 EDT 2004

Mac sample entries from Mac.log:

- Jul 1 09:00:55 calvisitor-10-105-160-95 kernel[0]: AppleThunderboltNHIType2::prePCIWake - power up complete - took 2 us
- Jul 1 09:00:55 calvisitor-10-105-160-95 kernel[0]: AppleThunderboltGenericHAL::earlyWake - complete - took 0 milliseconds
- Jul 1 09:00:55 calvisitor-10-105-160-95 kernel[0]: AirPort: Link Down on awdl0. Reason 1 (Unspecified).
- Jul 1 09:00:55 calvisitor-10-105-160-95 kernel[0]: ARPT: 620651.021206: wl0: wl_update_tcpkeep_seq: Original Seq: 2477329075, Ack: 1662858865, Win size: 4096
- Jul 1 09:00:55 calvisitor-10-105-160-95 kernel[0]: Bluetooth -- LE is supported - Disable LE meta event

HDFS Log Entry Length Overview

The HDFS log dataset contains 2,000 entries.

- **Entry length statistics:**
  - Minimum: 93 characters
  - Maximum: 2,520 characters
  - Mean: 141.92 characters
  - Standard deviation: 76.99 characters

The distribution is right-skewed, with most entries clustered between 100 and 200 characters, but a long tail of much longer entries. This suggests that while most HDFS log messages are moderately sized and structured, there are occasional very verbose entries—likely due to detailed error reports or stack traces.

Insights:

- The presence of a long tail indicates that some log events (possibly errors or system

events) generate much more verbose output.
- The main cluster of entry lengths suggests a standard log format for routine operations.
- The wide range and higher standard deviation reflect the complexity and variability of HDFS system events.

BGL Log Entry Length Overview

The BGL log dataset contains 4,747,963 entries.

- **Entry length statistics:**
  - Minimum: 94 characters
  - Maximum: 928 characters
  - Mean: 155.53 characters
  - Standard deviation: 50.28 characters

The distribution is sharply peaked, with most entries tightly clustered around 150–170 characters, and a few longer outliers. This suggests highly templated logging, likely due to automated system event reporting.

Insights:

- The sharp peak and narrow range indicate strong consistency in log formatting.
- Outliers may correspond to rare or complex events.
- The high volume and regularity make BGL logs ideal for automated analysis and drift detection.

HealthApp Log Entry Length Overview

The HealthApp log dataset contains 253,395 entries.

- **Entry length statistics:**
  - Minimum: 48 characters
  - Maximum: 294 characters
  - Mean: 90.86 characters

    - Standard deviation: 22.49 characters

The distribution is multimodal, with several distinct peaks, suggesting multiple common log templates or event types. Most entries are short to moderate in length.

Insights:

- Multiple peaks likely correspond to different event types (e.g., user actions, sensor data, system events).
- The relatively low mean and standard deviation suggest concise, structured logging.
- The diversity of entry lengths reflects the variety of events captured by the app.

HPC Log Entry Length Overview

The HPC log dataset contains 433,490 entries.

- **Entry length statistics:**
    - Minimum: 43 characters
    - Maximum: 895 characters
    - Mean: 75.40 characters
    - Standard deviation: 34.38 characters

The distribution is right-skewed, with most entries between 50 and 120 characters, but a long tail of longer entries. This suggests a mix of routine and verbose event reporting.

Insights:

- Most log entries are concise, but some events (likely errors or system reports) are much longer.
- The moderate standard deviation reflects some variability in event reporting.
- The structure is suitable for automated parsing, but outliers may require special handling.

Linux Log Entry Length Overview

The Linux log dataset contains 25,567 entries.

- **Entry length statistics:**
  - Minimum: 28 characters
  - Maximum: 1,030 characters
  - Mean: 90.72 characters
  - Standard deviation: 40.90 characters

The distribution is right-skewed, with a main cluster around 80–120 characters and a long tail. This reflects the diversity of system and application events in Linux logs.

Insights:

- The main cluster suggests standard syslog formatting for most events.
- The long tail may be due to verbose error messages or kernel logs.
- The moderate variability is typical for multi-purpose system logs.

Mac Log Entry Length Overview

The Mac log dataset shows a wide range of entry lengths.

- **Entry length statistics:**
  - (Use your actual numbers from the plot)
  - The distribution is broad, with several peaks and a long right tail.

Insights:

- Multiple peaks indicate a variety of log templates and event types.
- The long tail suggests occasional verbose system or kernel messages.
- The diversity in entry lengths reflects the complexity of Mac system logging.

Missing or Malformed Entries Overview

We checked all datasets for empty or obviously malformed log entries using a simple count of empty lines. The results are as follows:

- **HDFS:** 0 empty entries found.

- **APACHE:** 0 empty entries found.
- **BGL:** 0 empty entries found.
- **HEALTHAPP:** 0 empty entries found.
- **HPC:** 0 empty entries found.
- **LINUX:** 0 empty entries found.
- **MAC:** 0 empty entries found.

Interpretation:

- The absence of empty or malformed entries across all datasets indicates high data quality and reliable log collection processes.
- This ensures that subsequent analyses (feature extraction, clustering, drift detection) are not affected by missing data, reducing the need for additional cleaning or imputation.
- The consistency in data quality across diverse log sources further supports the robustness of the datasets for automated and statistical analysis.

HDFS Message Type Distribution Overview

- **INFO:** 1,920 (96.00%)
- **ERROR:** 80 (4.00%)

Interpretation:

- The vast majority of HDFS log entries are informational, indicating routine system operations.
- Errors are rare, suggesting stable system behavior during the sampled period.
- The low error rate is a positive indicator for system health, but the presence of any errors should be further investigated for potential drift or anomalies.

Apache Message Type Distribution Overview

- **INFO:** 13,755 (24.35%)
- **ERROR:** 38,081 (67.42%)
- **OTHER:** 4,478 (7.93%)

- **WARNING:** 168 (0.30%)

Interpretation:

- Errors dominate the Apache logs, making up over two-thirds of all entries. This may indicate frequent issues, misconfigurations, or aggressive error logging.
- Informational messages are also common, while warnings and other types are rare.
- The high error rate warrants further investigation to determine if it reflects true system instability or simply verbose error reporting.

BGL Message Type Distribution Overview

- **ERROR:** 1,394,876 (29.38%)
- **INFO:** 2,917,660 (61.45%)
- **CRITICAL:** 384,491 (8.10%)
- **DEBUG:** 15,429 (0.32%)
- **WARNING:** 34,550 (0.73%)
- **OTHER:** 957 (0.02%)

Interpretation:

- BGL logs are dominated by informational and error messages, with a significant proportion of critical events.
- The presence of many critical and error messages suggests the system is closely monitored and logs a wide range of event severities.
- Debug and warning messages are rare, indicating focused logging on major events.

HealthApp Message Type Distribution Overview

- **OTHER:** 251,661 (99.32%)
- **ERROR:** 1,704 (0.67%)
- **INFO:** 25 (0.01%)
- **DEBUG:** 5 (0.00%)

Interpretation:

- Nearly all HealthApp log entries are classified as "OTHER," suggesting non-standard or custom message types dominate.
- Errors are rare, and standard INFO/DEBUG messages are almost absent.
- This may reflect a highly customized logging scheme or a need to refine message type extraction for this dataset.

Linux Message Type Distribution Overview

- **OTHER:** 21,230 (83.04%)
- **ERROR:** 4,181 (16.35%)
- **INFO:** 38 (0.15%)
- **WARNING:** 19 (0.07%)
- **DEBUG:** 99 (0.39%)

Interpretation:

- Most Linux log entries are classified as "OTHER," indicating a wide variety of message formats or types not captured by standard patterns.
- Errors are relatively common, while standard INFO, WARNING, and DEBUG messages are rare.
- The diversity of message types reflects the complexity of Linux system logging.

Mac Message Type Distribution Overview

- **OTHER:** 95,450 (81.77%)
- **ERROR:** 19,089 (16.35%)
- **INFO:** 1,705 (1.46%)
- **WARNING:** 323 (0.28%)
- **DEBUG:** 168 (0.14%)

Interpretation:

- The majority of Mac log entries are "OTHER," with a significant proportion of errors.
- Standard INFO, WARNING, and DEBUG messages are rare, suggesting a mix of custom and standard logging.
- The high "OTHER" count may indicate the need for more tailored message type extraction for Mac logs.

Temporal EDA Interpretation: Log Dataset Insights

This section summarizes the key findings from the temporal exploratory data analysis (EDA) plots for each dataset, focusing on hourly activity, time between logs, cumulative log counts, and summary statistics. These insights inform our understanding of system behavior, data quality, and readiness for drift detection.

HDFS

- **Summary:**
  - 2,000 timestamps (2008-11-09 to 2008-11-11), average interval ~68 seconds.
  - Most common hours: 21, 10, 7; most common days: 10, 11, 9.
- **Hourly Distribution:** Activity is spread across the day, with clear peaks at 10 and 21 hours, suggesting batch or scheduled jobs.
- **Time Between Logs:** Most logs are close together (many at 0 seconds), but some large gaps indicate bursts of activity followed by idle periods.
- **Cumulative Logs:** Steady increase with plateaus, reflecting bursty HDFS operations.
- **Interpretation:** HDFS logs show regular, bursty activity typical of distributed storage systems. Temporal features are reliable for drift analysis.

Apache

- **Summary:**
  - 52,004 timestamps, but time range and intervals are nonsensical (1900-01-01 to 1900-12-25, negative/huge time gaps).
- **Hourly Distribution:** Activity throughout the day, with peaks at 3, 4, and 12 hours.
- **Time Between Logs & Cumulative Logs:** Plots are not meaningful due to timestamp parsing issues.

- **Interpretation:** Despite parsing issues, hourly patterns suggest regular web server activity. Timestamp extraction must be fixed for reliable temporal analysis.

BGL

- **Summary:**
  - 11,925 timestamps, time range from 1900 to 2055 (erroneous), large negative/positive intervals.
- **Hourly Distribution:** Strong peaks at 8, 10, 12 hours, likely reflecting scheduled jobs or system events.
- **Time Between Logs & Cumulative Logs:** Plots dominated by single values or discontinuities due to timestamp errors.
- **Interpretation:** Hourly patterns are clear, but timestamp correction is needed for further analysis.

HealthApp

- **Summary:**
  - Only 14 timestamps, all at the same second (2018-01-02 18:52:59).
- **Hourly Distribution:** All logs at hour 18.
- **Time Between Logs & Cumulative Logs:** All zero or at a single point in time.
- **Interpretation:** No temporal diversity; likely a test or event dump. Too small for meaningful temporal analysis.

Linux

- **Summary:**
  - 25,567 timestamps, time range in 1900, negative average time between logs.
- **Hourly Distribution:** Activity throughout the day, with peaks at 5, 12, 14 hours.
- **Time Between Logs & Cumulative Logs:** Plots not meaningful due to timestamp issues.
- **Interpretation:** Hourly patterns show real usage cycles, but timestamp parsing must be fixed.

Mac

- **Summary:**

    - 107,201 timestamps, time range from 1900 to 2017, large negative/positive intervals.
- **Hourly Distribution:** Activity spread across the day, with peaks at 13, 16, 23 hours.
- **Time Between Logs & Cumulative Logs:** Plots not meaningful due to timestamp issues.
- **Interpretation:** Multiple peaks suggest regular system/user activity, but timestamp parsing needs correction.

General Observations & Recommendations

- **HDFS** is the only dataset with correct and meaningful temporal statistics and plots. It shows bursty, scheduled activity typical of distributed storage systems.
- **Apache, BGL, Linux, Mac** all have timestamp parsing issues, leading to negative or nonsensical time intervals and cumulative plots. However, their hourly distributions still reveal real usage patterns (peaks at certain hours).
- **HealthApp** has too few entries for meaningful temporal analysis.

Action Items:

1. **Fix Timestamp Parsing:** Review and correct timestamp extraction for Apache, BGL, Linux, and Mac logs. Re-run temporal EDA after fixing to get accurate time-based insights.
2. **Interpret Hourly Patterns:** Despite timestamp issues, hourly distributions show real system usage cycles (e.g., business hours, scheduled jobs). These can be used for drift detection once timestamps are fixed.
3. **Cumulative and Time-Between Plots:** Currently only meaningful for HDFS. After fixing timestamps, these will help identify bursts, outages, and drift events.
4. **HealthApp:** Check for more data or correct time extraction; current sample is too small.

These findings guide our next steps in data cleaning, feature engineering, and drift detection.

Component Extraction and Top Component Distributions

This section summarizes the results of component extraction and the distribution of top

components for each dataset, as visualized in the bar plots and extraction summaries.

Timestamp Extraction Issues

- **Observation:** The current timestamp extraction failed for all datasets ("No timestamps found or format mismatch"). This confirms the need for custom or dataset-specific timestamp parsing, as previously recommended.
- **Action:** Update and test timestamp extraction logic for each dataset, using sample log entries as reference for correct formats.

Top Component Distributions

Mac

- **Dominant Components:** The most frequent component is labeled as '0', with other top entries being generic or ambiguous (e.g., 'Unspecified', 'enable', 'enabled', '0x0').
- **Issues:** The extraction appears to capture placeholder or default values, rather than meaningful system components or process names.
- **Implications:** For Mac logs, refining the extraction logic to focus on process names, daemons, or application identifiers will be necessary for effective drift detection and monitoring.

BGL

- **Dominant Components:** System-level components such as 'ciod', 's', and specific error/status messages dominate the distribution.
- **Issues:** Extraction is effective, but some generic or single-character components ('s') may need further review.
- **Implications:** These components are strong candidates for drift and anomaly monitoring, especially in the context of HPC job scheduling and hardware events.

HealthApp

- **Dominant Components:** Top entries are timestamp-like strings, likely reflecting event times rather than true components.
- **Issues:** Extraction logic may be capturing the wrong field; focus should shift to event types or module names.

- **Implications:** Accurate extraction of app modules or event types is needed to monitor drift related to user behavior or app updates.

Apache

- **Dominant Components:** 'error' and 'notice' are the most frequent, followed by client IP addresses and some timestamped entries.
- **Issues:** The extraction is effective for log levels and client activity, but some entries may be misclassified as components.
- **Implications:** Monitoring error and notice rates, as well as tracking activity from specific clients, is valuable for detecting operational drift, attacks, or misconfigurations in web server environments.

HPC

- **Dominant Components:** Includes keywords like 'for', status codes, and numeric identifiers.
- **Issues:** Both true components and incidental values are captured; further refinement is needed to distinguish between them.
- **Implications:** Identifying and monitoring key subsystems or node IDs will improve drift detection and operational monitoring.

Linux

- **Dominant Components:** Process names ('httpd', 'pam_unix', 'python', 'sendmail'), device names, and kernel/module identifiers are prominent.
- **Issues:** Extraction is generally effective, but some device or kernel identifiers may require grouping or normalization.
- **Implications:** Tracking activity and errors from major processes and services is crucial for detecting drift related to service changes, security, or system updates.

HDFS

- **Dominant Components:** Core system components ('DataNode', 'PacketResponder', 'FSNamesystem') and block IDs dominate.
- **Issues:** Extraction is effective, with clear identification of key system parts.
- **Implications:** Monitoring these components for changes in activity or error rates is essential for early detection of drift, failures, or maintenance events in distributed storage

systems.

General Insights and Recommendations

- **Component extraction quality varies** by dataset. Highly structured logs (BGL, HDFS, Apache) yield more meaningful component distributions, while semi-structured or free-form logs (Mac, HealthApp, Linux) may require more tailored extraction logic.
- **Action Items:**
  i. Refine component extraction for datasets with ambiguous or generic top entries.
  ii. Use top component trends to guide feature selection for drift detection and system health monitoring.
  iii. Integrate component activity analysis with temporal and message type features for a comprehensive drift detection approach.

These findings will inform the next steps in feature engineering and drift detection, ensuring that the most relevant and interpretable components are monitored for changes over time.

Improved Temporal Coverage and Log Entry Distributions

After refining timestamp extraction for each dataset, we generated updated temporal coverage plots and statistics. This section summarizes the new findings and their implications for drift detection and further analysis.

Apache

- **Coverage:** 2005-06-09 to 2006-02-28
- **Patterns:** Log activity is highly bursty, with periods of intense logging interspersed with quieter intervals. Notable spikes in late 2005 and early 2006 may correspond to system events, outages, or attacks.
- **Implications:** The improved timestamp extraction enables accurate detection of temporal drift, burst events, and operational cycles in web server activity.

BGL

- **Coverage:** 2005-06-03 to 2006-01-04
- **Patterns:** Extremely high-volume bursts are visible, with some days showing hundreds of thousands of entries. These likely correspond to batch job cycles or major system events in the HPC environment.
- **Implications:** The ability to pinpoint bursts and quiet periods is crucial for identifying drift related to job scheduling, hardware failures, or maintenance windows.

HealthApp

- **Coverage:** 2017-12-23 to 2017-12-31
- **Patterns:** Log entries are distributed across several days, with some days showing much higher activity. The data now covers a meaningful period for user and app event analysis.
- **Implications:** Accurate temporal coverage allows for the study of user behavior cycles, app updates, and potential drift after feature releases.

HPC

- **Coverage:** 2003-07-31 to 2006-04-30
- **Patterns:** Log activity is highly variable, with frequent spikes and long periods of low activity. This reflects the nature of HPC workloads and system events.
- **Implications:** The improved extraction supports the detection of operational drift, job submission cycles, and rare anomaly events.

Linux

- **Coverage:** 2000-01-01 to 2000-12-31
- **Patterns:** Activity is concentrated in certain months, with bursts of logging and long quiet periods. This may reflect system reboots, upgrades, or changes in logging configuration.
- **Implications:** Reliable time extraction enables the study of system usage cycles, error bursts, and the impact of configuration changes on log volume.

Mac

- **Coverage:** 2000-07-01 to 2000-07-08
- **Patterns:** Log entries are distributed over a short period, with daily cycles and some

days showing higher activity. The short coverage may be due to the sample or log rotation.
- **Implications:** Even with limited coverage, accurate timestamps allow for the analysis of daily system cycles and event bursts.

HDFS

- **Coverage:** 2008-11-09 to 2008-11-11
- **Patterns:** Log entries are concentrated over three days, with clear daily cycles and high activity during certain periods. This reflects typical HDFS operational patterns.
- **Implications:** The improved extraction supports the detection of operational drift, maintenance events, and error bursts in distributed storage systems.

General Insights and Recommendations

- **Accurate timestamp extraction** has enabled meaningful temporal analysis across all datasets, revealing true log coverage, burstiness, and operational cycles.
- **Implications for drift detection:** With reliable time features, we can now apply statistical and time series methods to detect drift, anomalies, and regime changes.
- **Next steps:**
    i. Use these temporal features for windowed drift detection and anomaly analysis.
    ii. Integrate temporal insights with message type and component analyses for a holistic view of system behavior.
    iii. Document any remaining gaps or limitations in coverage for future data collection or preprocessing improvements.

Top Component Distributions: Discussion and Implications

This section discusses the results of the top component extraction and visualization for each dataset, based on the bar plots and summary tables.

Mac

- **Dominant Components:** The most frequent component is labeled as '0', with other top entries being generic or ambiguous (e.g., 'Unspecified', 'enable', 'enabled', '0x0').
- **Issues:** The extraction appears to capture placeholder or default values, rather than meaningful system components or process names.
- **Implications:** For Mac logs, refining the extraction logic to focus on process names, daemons, or application identifiers will be necessary for effective drift detection and monitoring.

Apache

- **Dominant Components:** 'error' and 'notice' are the most frequent, followed by client IP addresses and some timestamped entries.
- **Issues:** The extraction is effective for log levels and client activity, but some entries may be misclassified as components.
- **Implications:** Monitoring error and notice rates, as well as tracking activity from specific clients, is valuable for detecting operational drift, attacks, or misconfigurations in web server environments.

BGL

- **Dominant Components:** System-level components such as 'ciod', 's', and specific error/status messages dominate the distribution.
- **Issues:** Extraction is effective, but some generic or single-character components ('s') may need further review.
- **Implications:** These components are strong candidates for drift and anomaly monitoring, especially in the context of HPC job scheduling and hardware events.

HealthApp

- **Dominant Components:** Top entries are timestamp-like strings, likely reflecting event times rather than true components.
- **Issues:** Extraction logic may be capturing the wrong field; focus should shift to event types or module names.
- **Implications:** Accurate extraction of app modules or event types is needed to monitor

drift related to user behavior or app updates.

HPC

- **Dominant Components:** Includes keywords like 'for', status codes, and numeric identifiers.
- **Issues:** Both true components and incidental values are captured; further refinement is needed to distinguish between them.
- **Implications:** Identifying and monitoring key subsystems or node IDs will improve drift detection and operational monitoring.

Linux

- **Dominant Components:** Process names ('httpd', 'pam_unix', 'python', 'sendmail'), device names, and kernel/module identifiers are prominent.
- **Issues:** Extraction is generally effective, but some device or kernel identifiers may require grouping or normalization.
- **Implications:** Tracking activity and errors from major processes and services is crucial for detecting drift related to service changes, security, or system updates.

HDFS

- **Dominant Components:** Core system components ('DataNode', 'PacketResponder', 'FSNamesystem') and block IDs dominate.
- **Issues:** Extraction is effective, with clear identification of key system parts.
- **Implications:** Monitoring these components for changes in activity or error rates is essential for early detection of drift, failures, or maintenance events in distributed storage systems.

General Insights and Recommendations

- **Component extraction quality varies** by dataset. Highly structured logs (BGL, HDFS, Apache) yield more meaningful component distributions, while semi-structured or free-form logs (Mac, HealthApp, Linux) may require more tailored extraction logic.
- **Action Items:**

i. Refine component extraction for datasets with ambiguous or generic top entries.
ii. Use top component trends to guide feature selection for drift detection and system health monitoring.
iii. Integrate component activity analysis with temporal and message type features for a comprehensive drift detection approach.

These findings will inform the next steps in feature engineering and drift detection, ensuring that the most relevant and interpretable components are monitored for changes over time.

Log Entry Word Count Distributions: Discussion and Insights

This section discusses the word count distributions for each dataset, based on the histograms and summary statistics. These distributions provide insight into the structure, verbosity, and diversity of log messages, which are important for feature engineering and drift detection.

HDFS

- **Stats:** Min: 9, Max: 110, Mean: 12.44, Std: 3.73
- **Pattern:** The distribution is tightly clustered, with most entries between 9 and 20 words, and a few outliers with much higher word counts.
- **Implications:** HDFS logs are highly structured and concise, with occasional verbose entries likely corresponding to errors or detailed system events. This regularity supports automated analysis and drift detection.

Apache

- **Stats:** Min: 7, Max: 23, Mean: 12.39, Std: 2.34
- **Pattern:** The distribution is sharply peaked, with several distinct modes, reflecting different log templates or message types.
- **Implications:** Apache logs are highly templated, with little variation in entry length. This makes them ideal for detecting anomalies or drift based on changes in message structure.

BGL

- **Stats:** Min: 9, Max: 102, Mean: 15.17, Std: 7.48
- **Pattern:** The distribution is sharply peaked with a long right tail, indicating a standard log format with occasional verbose entries.
- **Implications:** BGL logs are consistent and suitable for automated parsing, but the presence of outliers suggests rare or complex events that may be important for drift or anomaly detection.

HealthApp

- **Stats:** Min: 1, Max: 21, Mean: 3.28, Std: 1.96
- **Pattern:** The distribution is multimodal, with several distinct peaks, likely corresponding to different event types or log templates.
- **Implications:** HealthApp logs are concise and diverse, reflecting a variety of user and system events. The multimodal nature may help identify changes in app behavior or the introduction of new features.

HPC

- **Stats:** Min: 6, Max: 104, Mean: 9.73, Std: 4.09
- **Pattern:** The distribution is right-skewed, with most entries being concise but some much longer outliers.
- **Implications:** HPC logs are generally concise, but verbose entries may correspond to errors or detailed reports. Monitoring changes in the distribution can help detect operational drift or rare events.

Linux

- **Stats:** Min: 5, Max: 24, Mean: 12.05, Std: 2.37
- **Pattern:** The distribution is tightly clustered with a few distinct peaks, reflecting standard syslog formatting and some diversity in message types.
- **Implications:** Linux logs are regular and structured, supporting automated analysis. Changes in the distribution may indicate new services, configuration changes, or drift.

Mac

- **Stats:** Min: 1, Max: 104, Mean: 13.04, Std: 6.50
- **Pattern:** The distribution is broad with several peaks and a long right tail, indicating a mix of concise and verbose entries.
- **Implications:** Mac logs are diverse in structure, reflecting a wide range of system and application events. The presence of verbose outliers may signal rare or complex events worth monitoring for drift or anomalies.

General Insights and Recommendations

- **Structured logs (HDFS, Apache, BGL, HPC):** Show tight clustering and regularity, making them well-suited for automated drift detection based on entry length or structure.
- **Semi-structured logs (HealthApp, Linux, Mac):** Show greater diversity and multimodality, reflecting a wider range of event types and system behaviors.
- **Action Items:**
    i. Use word count and entry length as features for drift and anomaly detection.
    ii. Monitor for changes in the distribution, emergence of new peaks, or increases in outlier frequency as potential indicators of drift or system changes.
    iii. Integrate word count analysis with message type and component features for a comprehensive view of log evolution.

These findings will inform the next steps in feature engineering and drift detection, ensuring that the most relevant and interpretable components are monitored for changes over time.

Rare Component Analysis: Discussion and Implications

This section discusses the results of rare component analysis across all datasets. Rare components are those that appear very infrequently (e.g., ≤2 times) in the logs. Identifying and monitoring these rare components is important for drift and anomaly detection, as they may correspond to unusual events, new system behaviors, or potential errors.

Summary of Results

- **HDFS:** 2,201 rare components (≤2 times)
  - Sample: Block IDs such as 'blk_38865049064139660', 'blk_6952295868487656571', etc.
- **APACHE:** 19,953 rare components (≤2 times)
  - Sample: Timestamps, client IPs, and specific log lines (e.g., 'Thu Jun 09 07:11:21 2005', 'client 216.68.171.39').
- **BGL:** 178,301 rare components (≤2 times)
  - Sample: Node IDs, error codes, and long hexadecimal strings (e.g., 'R33-M1-ND', '2032315038334330000000000594c31304b353030343033232').
- **HEALTHAPP:** 9 rare components (≤2 times)
  - Sample: Timestamp-like strings (e.g., '20171227-3', '20171229-7').
- **HPC:** 1,801 rare components (≤2 times)
  - Sample: Numeric codes and ranges (e.g., '239-242', '235-238').
- **LINUX:** 7,345 rare components (≤2 times)
  - Sample: Numeric codes, device names, and process IDs (e.g., '1677', '1681', '-1509093696').
- **MAC:** 6,142 rare components (≤2 times)
  - Sample: Process names, timestamps, and warning messages (e.g., 'Mail', '09:00:56.119', '0701/090056:WARNING:dns_config_service_posix.cc(306)').

Interpretation and Insights

- **What are rare components?**
  - In structured logs (HDFS, BGL, HPC), rare components often correspond to unique identifiers (block IDs, node IDs, error codes) or one-off events.
  - In web/server logs (APACHE), rare components include unique client IPs, timestamps, or specific error messages.
  - In system/application logs (LINUX, MAC, HEALTHAPP), rare components may be process IDs, device names, or timestamped events.
- **Why are they important?**
  - Rare components can signal anomalies, new or unexpected system behaviors, or the introduction of new entities (e.g., new devices, users, or error types).
  - Monitoring the frequency and emergence of rare components is valuable for early drift detection and root cause analysis.

- **Patterns and differences:**
  - Datasets with highly structured, high-volume logs (BGL, APACHE) have the largest number of rare components, reflecting the diversity of system events and identifiers.
  - Application and system logs (HEALTHAPP, MAC, LINUX) have fewer rare components, but these may be more meaningful for detecting user-driven or configuration changes.

Implications for Drift and Anomaly Detection

- **Drift detection:**
  - A sudden increase in the number or frequency of rare components may indicate drift, such as new system behaviors, configuration changes, or the onset of failures.
- **Anomaly detection:**
  - Rare components are strong candidates for anomaly detection, as they may correspond to outliers, errors, or security events.
- **System monitoring:**
  - Regularly tracking rare components can help identify emerging issues, new devices, or changes in usage patterns.

Recommendations

1. Integrate rare component monitoring into the drift and anomaly detection pipeline.
2. Investigate the context and impact of newly emerging rare components, especially those associated with errors or failures.
3. Combine rare component analysis with temporal, message type, and word count features for a comprehensive view of log evolution and system health.

Log Entry Uniqueness: Interpretation and Implications

This section discusses the proportion of unique log entries in each dataset. The uniqueness ratio (unique entries / total entries) provides insight into the diversity and repetitiveness of log messages, which has important implications for system monitoring, drift detection, and

anomaly analysis.

Summary of Results

- **HDFS:** 2000/2000 unique entries (100.00%)
- **APACHE:** 28,326/56,482 unique entries (50.15%)
- **BGL:** 4,747,959/4,747,963 unique entries (100.00%)
- **HEALTHAPP:** 252,979/253,395 unique entries (99.84%)
- **HPC:** 416,673/433,490 unique entries (96.12%)
- **LINUX:** 25,546/25,567 unique entries (99.92%)
- **MAC:** 95,808/116,735 unique entries (82.07%)

Interpretation and Insights

- **High uniqueness (close to 100%):**
  - Indicates a highly variable or noisy system, where most log entries are distinct. This is typical for logs that include unique identifiers (block IDs, timestamps, user/device IDs) or detailed event descriptions.
  - Seen in HDFS, BGL, HealthApp, HPC, and Linux logs.
  - **Implications:** High uniqueness can make automated pattern recognition and anomaly detection more challenging, but also suggests rich information content and the potential for detecting rare or novel events.
- **Moderate uniqueness (50-85%):**
  - Indicates a mix of repetitive and unique messages. Some logs are templated or repetitive, while others are more variable.
  - Seen in Mac logs (82%) and especially Apache logs (50%).
  - **Implications:** Lower uniqueness in Apache logs suggests a high degree of templating or repeated events (e.g., standard access/error messages), which can make it easier to detect anomalies or drift as deviations from the norm.

Implications for Drift and Anomaly Detection

- **High uniqueness:**
  - Systems with high uniqueness may require more sophisticated, context-aware methods for drift and anomaly detection, as simple frequency-based approaches may not be effective.

    - Monitoring the emergence of new types of unique entries or sudden drops in uniqueness can signal drift or system changes.
- **Low/Moderate uniqueness:**
    - Systems with more repetitive logs are well-suited for template-based anomaly detection, as rare or novel messages stand out more clearly.
    - Changes in the ratio of unique to total entries over time can indicate drift, such as the introduction of new log templates, system updates, or changes in user behavior.

Recommendations

1. Track uniqueness ratios over time to monitor for sudden changes or trends that may indicate drift or anomalies.
2. Combine uniqueness analysis with rare component, word count, and temporal features for a comprehensive view of log diversity and system evolution.
3. Use clustering and template extraction methods to group similar log entries and facilitate anomaly detection in high-uniqueness datasets.

Clustering Results: Interpretation and Implications

This section interprets the results of unsupervised clustering (KMeans, k=5) applied to log entries from each dataset. Clustering helps reveal dominant log patterns, repetitive messages, and rare or anomalous entries, providing valuable insights for drift and anomaly detection.

HDFS

- **Clusters:** 5 clusters, sizes range from 263 to 770 entries.
- **Dominant patterns:** Each cluster corresponds to a distinct log template or event type (e.g., block deletions, block receptions, block allocations, system status updates).
- **Largest cluster:** Routine block reception events.
- **Smallest cluster:** Block deletion or allocation events.
- **Implications:** HDFS logs are highly structured, with clear separation between routine operations and less frequent events. Small clusters may indicate rare or significant system events worth monitoring for drift.

APACHE

- **Clusters:** 5 clusters, sizes range from ~4,400 to ~33,000 entries.
- **Dominant patterns:** Clusters correspond to common log templates (e.g., worker initialization, script errors, LDAP/SSL notices, error messages, child process events).
- **Largest cluster:** Standard notice messages (e.g., LDAP, suEXEC, etc.).
- **Smallest clusters:** Error messages and script errors.
- **Implications:** Apache logs are dominated by a few repetitive templates, making it easier to spot anomalies or drift as new or rare clusters emerge.

BGL

- **Clusters:** 5 clusters, sizes range from ~150,000 to over 2 million entries.
- **Dominant patterns:** Clusters reflect different node or event types (e.g., RAS KERNEL events, UNKNOWN_LOCATION, block allocations).
- **Largest clusters:** Routine system or node events.
- **Smallest clusters:** Rare node or error events.
- **Implications:** BGL logs are highly regular, with a few dominant patterns. Small clusters may correspond to rare system events or anomalies.

HealthApp

- **Clusters:** 5 clusters, sizes range from ~11,000 to ~71,000 entries.
- **Dominant patterns:** Clusters correspond to different app modules or event types (e.g., step tracking, broadcast actions, sensor data, report events).
- **Largest clusters:** Routine user or app events.
- **Smallest clusters:** Less common app actions or system events.
- **Implications:** HealthApp logs are diverse, with clusters reflecting user behavior and app features. Changes in cluster composition may indicate drift due to app updates or new features.

HPC

- **Clusters:** 5 clusters, sizes range from ~21,000 to ~203,000 entries.
- **Dominant patterns:** Clusters reflect different system components or error types (e.g., switch module errors, temperature warnings, resource configuration failures, fan speeds,

link errors).
- **Largest clusters:** Routine hardware or system status events.
- **Smallest clusters:** Specific error or warning events.
- **Implications:** HPC logs show clear separation between routine and error events. Monitoring small clusters can help detect emerging issues or drift.

Linux

- **Clusters:** 5 clusters, sizes range from ~2,000 to ~9,000 entries.
- **Dominant patterns:** Clusters correspond to system startup, authentication failures, kernel events, and other routine operations.
- **Largest clusters:** System startup and routine service messages.
- **Smallest clusters:** Authentication failures or rare kernel events.
- **Implications:** Linux logs are structured, with clusters reflecting system operations. Small clusters may indicate security issues or rare system events.

Mac

- **Clusters:** 5 clusters, sizes range from ~4,000 to ~51,000 entries.
- **Dominant patterns:** Clusters reflect different system processes, application errors, kernel events, and user actions.
- **Largest clusters:** Routine system or application events.
- **Smallest clusters:** Application errors or rare system events.
- **Implications:** Mac logs are diverse, with clusters capturing both system and user-level activity. Monitoring changes in cluster sizes or the emergence of new clusters can help detect drift or anomalies.

General Insights and Recommendations

- **Cluster size and composition:** Most datasets show a few large clusters (routine events) and several smaller clusters (rare or anomalous events). This separation is useful for drift and anomaly detection.
- **Dominant patterns:** Clustering reveals the main operational modes of each system, helping to identify what "normal" looks like.

- **Rare clusters:** Small clusters are strong candidates for further investigation, as they may correspond to errors, anomalies, or the onset of drift.
- **Action Items:**
    i. Monitor cluster sizes and composition over time to detect drift or emerging issues.
    ii. Investigate the content of small or new clusters for potential anomalies or system changes.
    iii. Integrate clustering results with temporal, uniqueness, and component analyses for a comprehensive drift detection strategy.

DBSCAN Clustering Results: Interpretation and Implications

Note on Subsampling and Computational Constraints

Due to the large size of several log datasets and the computational intensity of density-based clustering algorithms like DBSCAN, we performed clustering on random subsamples (e.g., 5,000 entries per dataset) rather than the full datasets. While this approach enables practical analysis and rapid iteration, it may limit the detection of rare patterns or subtle anomalies that only appear in the full data. As a result, the clustering results primarily reflect the dominant patterns present in the sampled data, and some rare or outlier events may not be captured. For more comprehensive anomaly or drift detection, future work could explore scalable clustering methods, distributed computation, or targeted sampling strategies.

Summary of Results

- For all datasets (HDFS, APACHE, BGL, HEALTHAPP, HPC, LINUX, MAC), DBSCAN found only **one cluster** and almost **no noise points** (potential anomalies).
- In the case of the Mac dataset, a small number of noise points (e.g., 230 out of 5,000) were detected, but the vast majority of entries were assigned to a single cluster.
- Subsampling (e.g., 5,000 entries per dataset) was used for computational efficiency.

Interpretation and Insights

- **Single cluster:**

    - The fact that DBSCAN found only one cluster in each dataset suggests that, under the current settings and feature representation, the log entries are highly similar in their TF-IDF space.
    - This may reflect the dominance of routine, templated log messages, or the limitations of the chosen parameters (e.g., `eps`, `min_samples`) and feature reduction.
- **Few/no noise points:**
    - The lack of noise points indicates that DBSCAN did not identify any strong outliers or rare patterns in the sampled data. This could mean that the logs are highly regular, or that the clustering is not sensitive enough to detect subtle anomalies.
- **Parameter and method sensitivity:**
    - DBSCAN is sensitive to the choice of `eps` (neighborhood size) and `min_samples`. The current settings may be too permissive, causing all points to be grouped together.
    - Subsampling and aggressive dimensionality reduction (TF-IDF + SVD) may also reduce the ability to distinguish fine-grained patterns or rare events.

Implications for Drift and Anomaly Detection

- **Homogeneity:**
    - The results suggest that, at the chosen scale and feature granularity, the logs are dominated by similar, routine messages. This is consistent with the high regularity seen in previous clustering and uniqueness analyses.
- **Missed anomalies:**
    - If the goal is to detect rare or anomalous events, further tuning of DBSCAN parameters (lower `eps`, higher `min_samples`), less aggressive dimensionality reduction, or targeted sampling (e.g., focusing on error logs) may be needed.
- **Complementary methods:**
    - DBSCAN may be more effective when combined with other approaches (e.g., KMeans, rare component analysis, or domain-specific filtering) to isolate anomalies or drift events.

Recommendations

1. Experiment with a range of `eps` and `min_samples` values to increase DBSCAN sensitivity to rare patterns.
2. Try clustering on filtered subsets (e.g., only ERROR or WARNING logs) or on features that

emphasize rare events.
3. Use DBSCAN results in combination with other clustering and anomaly detection methods for a more comprehensive analysis.
4. Consider running DBSCAN on larger or more diverse samples if computationally feasible, or on embeddings that better capture semantic differences.

Hierarchical Clustering (Dendrogram) Results: Interpretation and Implications

This section interprets the results of agglomerative hierarchical clustering, visualized as dendrograms, for a random sample of 100 log entries from each dataset. Dendrograms provide a visual summary of the relationships between log entries, showing how clusters merge at different similarity thresholds and revealing both dominant patterns and outliers.

Summary of Results

- For each dataset (HDFS, APACHE, BGL, HEALTHAPP, HPC, LINUX, MAC), the dendrograms show a range of branching patterns, with some entries merging early (high similarity) and others joining at higher distances (greater dissimilarity).
- Subsampling (100 entries per dataset) was used for computational feasibility and clear visualization.

Interpretation and Insights

- **Cluster structure:**
  - Datasets with highly regular, templated logs (e.g., HDFS, BGL, APACHE) show many entries merging at low distances, indicating strong similarity and the presence of dominant log templates.
  - Datasets with more diverse or free-form logs (e.g., MAC, LINUX, HEALTHAPP) show more gradual merging and a wider spread of branch heights, reflecting greater diversity in log content and structure.
- **Outliers and rare patterns:**
  - Entries that merge late (at high linkage distances) are potential outliers or rare log types. These may correspond to unusual events, errors, or the introduction of new

log templates.

- **Cluster granularity:**
  - The dendrograms allow for flexible selection of cluster granularity: cutting the tree at different heights yields different numbers of clusters, which can be tuned for specific analysis goals (e.g., anomaly detection vs. template extraction).

Implications for Drift and Anomaly Detection

- **Log diversity:**
  - Datasets with more gradual or uneven dendrograms (e.g., MAC, LINUX) are more likely to contain a mix of routine and rare events, making them rich sources for anomaly and drift detection.
- **Template extraction:**
  - Datasets with tight, low-branching dendrograms (e.g., HDFS, BGL) are well-suited for automated template extraction and monitoring for deviations from established patterns.
- **Outlier detection:**
  - Late-joining entries in the dendrograms are strong candidates for further investigation as potential anomalies or drift events.

Recommendations

1. Use dendrograms to guide the selection of cluster thresholds for template extraction or anomaly detection.
2. Investigate late-joining or singleton entries for evidence of rare events, errors, or drift.
3. Combine hierarchical clustering insights with those from KMeans, DBSCAN, and rare component analysis for a comprehensive understanding of log structure and system behavior.
4. For large datasets, consider automated methods for dendrogram cut-point selection and outlier flagging.

---

t-SNE Visualization Results: Interpretation and Implications

t-SNE (t-distributed Stochastic Neighbor Embedding) was used to project high-dimensional log entry representations into two dimensions for visual inspection of cluster structure. The following summarizes the key findings from the t-SNE plots for each dataset:

Main Findings

- **Well-Separated Clusters:** Most datasets (e.g., HDFS, Apache, HealthApp, HPC) show several well-separated clusters in the t-SNE space, indicating the presence of distinct log entry patterns or templates. This suggests that the logs are composed of a mix of repetitive and unique message types, which is consistent with the results from KMeans clustering.
- **Cluster Compactness and Diversity:** Some datasets (e.g., HDFS, HPC) exhibit tight, compact clusters, while others (e.g., Mac, Linux) show more diffuse or overlapping clusters. This reflects differences in log structure, message diversity, and the degree of templating or regularity in the logs.
- **Dataset Differences:**
  - **HDFS:** Distinct, compact clusters, likely corresponding to well-defined log templates or event types.
  - **Apache:** Multiple clusters with some overlap, reflecting a mix of structured and less-structured log entries.
  - **BGL:** Several clusters, but with more spread, indicating higher diversity or noise in log messages.
  - **HealthApp:** Clear clusters, but with some outliers, possibly due to rare or anomalous events.
  - **HPC:** Well-separated, compact clusters, suggesting strong regularity in log formats.
  - **Linux/Mac:** More diffuse clusters, indicating greater heterogeneity and less templating in log entries.
- **Subsampling and Dimensionality Reduction:** All t-SNE plots were generated on random subsamples (e.g., 2,000 entries per dataset) and after reducing TF-IDF features with SVD. This enables visualization of large datasets but may obscure rare patterns or subtle anomalies present in the full data.

Implications for Drift and Anomaly Detection

- **Cluster Structure:** The presence of well-defined clusters supports the use of unsupervised methods for drift and anomaly detection, as changes in cluster composition or the emergence of new clusters may signal drift.
- **Outliers and Overlap:** Datasets with more overlap or diffuse clusters may require more sophisticated feature engineering or alternative clustering methods to improve separation and interpretability.
- **Monitoring:** Regular t-SNE visualizations can help monitor changes in log structure over time, providing an intuitive way to detect drift or emerging issues.

Overall, t-SNE visualizations confirm the diversity and structure of log entries across datasets, and provide a valuable tool for exploratory analysis and drift detection.

Windowed Drift Analysis: Statistical Test Interpretation

This section interprets the results of windowed drift analysis using the Kolmogorov-Smirnov (KS) test for word count and the Chi-squared test for message type distributions. These tests were applied to consecutive time windows (e.g., daily) to detect both structural and semantic changes in log data.

1. Windowed Feature Summary

- **Mean word count** and **error rate** are calculated for each window, along with message type distributions.
- Sudden drops in error rate or shifts in message type distribution may indicate changes in system behavior, logging configuration, or periods of stability.
- Consistent word count means and low standard deviation suggest stable log structure, while abrupt changes may signal drift.

2. KS Test for Word Count Drift

- The KS test compares the distribution of word counts between consecutive windows.

- **Significant drift** is indicated by a low p-value (typically $p < 0.05$).
- Many window pairs show significant changes (KS stat > 0.3, $p < 0.05$), indicating structural drift in log entries.
- Large jumps (e.g., KS stat > 0.7) suggest abrupt changes, possibly due to system events or anomalies.

3. Chi-squared Test for Message Type Drift

- The Chi-squared test compares message type distributions between windows.
- **Significant drift** is indicated by a low p-value ($p < 0.05$).
- Many window pairs show significant changes, indicating shifts in the frequency of message types (e.g., error bursts).
- Skipped windows are due to insufficient data or zero counts for all types.

4. General Insights

- Drift is not constant: both word count and message type distributions can remain stable for long periods, then change abruptly.
- Combining tests provides a more complete picture of drift, capturing both structural and semantic changes.
- These tests can be automated to flag periods of drift for further investigation, supporting proactive system monitoring.

5. Summary Table Template

Window Pair	KS Stat	KS p-value	Chi2 Stat	Chi2 p-value	Drift Detected?
0 vs 1	0.375	0.428	1.78	0.182	No
...	...	...	...	...	...
4 vs 5	0.907	0.000	7.79	0.005	**Yes**
...	...	...	...	...	...

Actionable Recommendations

- Investigate windows with significant drift for root causes (e.g., system events, updates, incidents).
- Use these tests as part of an automated drift detection pipeline.
- Visualize drift points on time series plots for clear communication to stakeholders.

These statistical tests provide a robust foundation for automated drift monitoring and can be integrated with other analyses for comprehensive system health tracking.

Change Point Detection on Error Rate: Interpretation and Implications

Change point detection was applied to the error rate time series using the `ruptures` library (PELT algorithm, 'l2' model, weekly windows). This method identifies points in time where the statistical properties of the error rate change significantly, which may correspond to system events, incidents, or operational changes.

Main Findings

- **Detected Change Points:** The algorithm identified change points at windows [50, 8125] (example; actual indices may vary by dataset and parameters).
- **Interpretation:**
  - The first change point (e.g., window 50) likely marks a transition from a period of high error rate to a period of low or zero errors, or vice versa. This could correspond to a resolved incident, a system update, or a change in logging configuration.
  - The final change point (e.g., window 8125) typically marks the end of the time series.
- **Operational Implications:**
  - Detected change points highlight periods where system behavior shifted, warranting further investigation for root causes (e.g., outages, deployments, configuration changes).
  - These points can be cross-referenced with logs, incident reports, or system updates to identify underlying causes.

Complementarity with Statistical Tests

- Change point detection provides a global view of major shifts in error rate, while windowed statistical tests (KS, Chi2) capture more granular, local drift events.
- Using both methods together enables robust, multi-scale drift and anomaly detection.

Recommendations

- Integrate change point detection into the automated drift monitoring pipeline.
- Investigate log entries and system events around detected change points for root cause analysis.
- Visualize change points on time series plots to communicate findings to stakeholders and support operational decision-making.

This approach enhances the ability to detect, explain, and respond to significant changes in system behavior, supporting proactive monitoring and reliability.

Detailed Interpretation of Detected Drift Windows (KS and Chi-squared Tests)

The following window pairs were flagged as exhibiting statistically significant drift by either the Kolmogorov-Smirnov (KS) test (for word count) or the Chi-squared test (for message type distribution):

```
Window 4: 1900-01-05 00:00:00 to 1900-01-06 00:00:00 | KS p=4.28e-05, Chi2 p=0.00526
Window 5: 1900-01-06 00:00:00 to 1900-01-07 00:00:00 | KS p=5.19e-06, Chi2 p=0.00597
Window 6: 1900-01-07 00:00:00 to 1900-01-08 00:00:00 | KS p=1.61e-09, Chi2 p=6.21e-23
Window 7: 1900-01-08 00:00:00 to 1900-01-09 00:00:00 | KS p=0.000814, Chi2 p=0.219
Window 8: 1900-01-09 00:00:00 to 1900-01-10 00:00:00 | KS p=0.0048, Chi2 p=0.00222
... (truncated for brevity)
```

Interpretation and Insights

- **Frequency and Timing:**

    - Drift is detected in many consecutive or closely spaced windows, indicating periods of rapid change in log structure or message type distribution.
    - Some drift windows are isolated, suggesting discrete events or anomalies.
- **Possible Causes:**
    - These drift points may correspond to operational incidents, system updates, configuration changes, or the onset/resolution of anomalies.
    - Clusters of drift windows may indicate extended incidents or transitions, while isolated points may reflect one-off events.
- **Test Complementarity:**
    - Some windows are flagged by both tests (very low p-values for both KS and Chi2), indicating strong evidence of drift affecting both structure and semantics.
    - Others are flagged by only one test, highlighting the value of using both tests to capture different types of drift.
- **Actionable Steps:**
    - Investigate log entries and system events in and around these windows to identify root causes.
    - Monitor for recurring patterns or repeated drift at similar times, which may indicate systemic issues or regular operational cycles.

Operational Value

- This approach enables targeted root cause analysis and supports proactive monitoring by highlighting exactly when and where significant changes occur in the log data.
- Combining both structural (KS) and semantic (Chi2) drift detection provides a comprehensive view of system evolution and potential issues.

These findings should be integrated into the broader drift detection and monitoring pipeline, with flagged windows prioritized for further investigation and reporting.

Example: Log Entry Comparison Across a Detected Drift Window

To better understand the nature of detected drift, we examined sample log entries from before and after a significant drift window (window 4 to 5):

Sample log entries from window 4 (before drift):

```
Jan  5 04:02:05 combo su(pam_unix)[20782]: session opened for user root by (uid=0)
Jan  5 04:02:05 combo su(pam_unix)[20782]: session closed for user root
Jan  5 04:02:06 combo logrotate: ALERT exited abnormally with exit status 1
Jan  5 04:08:04 combo su(pam_unix)[22008]: session opened for user root by (uid=0)
Jan  5 04:08:04 combo su(pam_unix)[22008]: session closed for user root
```

Sample log entries from window 5 (after drift):

```
Jan  6 04:02:05 combo su(pam_unix)[24286]: session opened for user root by (uid=0)
Jan  6 04:02:06 combo su(pam_unix)[24286]: session closed for user root
Jan  6 04:02:07 combo logrotate: ALERT exited abnormally with exit status 1
Jan  6 04:07:40 combo su(pam_unix)[24657]: session opened for user root by (uid=0)
Jan  6 04:07:41 combo su(pam_unix)[24657]: session closed for user root
[Fri Jan 06 09:19:49 2006] [error] [client 68.XXX.XXX.XXX] File does not exist: /var/www/h
[Fri Jan 06 11:07:11 2006] [error] [client 61.XXX.XXX.XXX] File does not exist: /var/www/h
[Fri Jan 06 11:22:00 2006] [error] [client 216.XXX.XXX.XXX] File does not exist: /var/www/
[Fri Jan 06 11:35:06 2006] [error] [client 210.XXX.XXX.XXX] File does not exist: /var/www/
[Fri Jan 06 12:04:21 2006] [error] [client 143.XXX.XXX.XXX] File does not exist: /var/www/
```

Interpretation:

- **Before drift (window 4):** Log entries are primarily system authentication and maintenance messages (e.g., `su(pam_unix)` session events, `logrotate` alerts). These are routine system operations with no web server errors present.
- **After drift (window 5):** The first few entries remain similar (system authentication and maintenance), but starting later in the window, there is a sudden appearance of Apache web server error messages (e.g., `[error] [client ...] File does not exist: /var/www/html/phpmyadmin`). This indicates a shift in system activity, possibly due to a new service being started, a configuration change, or the onset of web-based probing or attacks.

Implications:

- The detected drift corresponds to a clear change in log content and system behavior, from routine system operations to the emergence of repeated web server errors.
- This kind of analysis helps pinpoint the timing and nature of operational changes or

incidents, and can guide further root cause investigation or alerting.

Visualization: Log Feature Trends with Drift Points

The time series plot above shows the trends of key log features (mean word count and error rate) over time, with detected drift points marked as red dashed lines.

Interpretation:

- **Feature Trends:** The blue line represents the mean word count of log entries in each time window, while the orange line shows the error rate. These features are useful for monitoring structural and semantic changes in log data.
- **Drift Points:** The vertical red dashed lines indicate windows where statistical tests detected significant drift. These points highlight periods of abrupt change in log structure or error frequency, which may correspond to operational incidents, configuration changes, or attacks.
- **Date Axis Issues:** Most data points are concentrated around the year 1900, with a few outliers extending far into the future (e.g., 2020s, 2050s). This suggests that many log entries have default or malformed timestamps, likely due to extraction or parsing issues. Such artifacts can distort time-based analyses and visualizations.

Insights and Recommendations:

- The visualization effectively highlights when and where drift occurs, but also exposes data quality issues in the timestamp field.
- For reliable drift detection and monitoring, it is crucial to normalize and validate timestamps, filtering out or correcting entries with implausible dates.
- Stakeholders should be aware that some detected drift may be driven by timestamp artifacts rather than true operational changes.

This plot is a valuable tool for both technical analysis and stakeholder communication, but its accuracy depends on robust preprocessing and timestamp handling.

Summary of Analysis, Methodology, and Key Results

This project undertook a comprehensive, unsupervised analysis of multiple log datasets (HDFS, Apache, BGL, HealthApp, HPC, Linux, Mac) to explore data quality, structure, and temporal/semantic drift. The workflow and main findings are summarized below:

1. Data Quality and Preparation

- All datasets were checked for empty or malformed entries; none were found, indicating high baseline data quality.
- Log files were read with robust encoding detection and whitespace handling.
- Timestamp extraction required dataset-specific logic due to diverse formats; some datasets had malformed or default timestamps, impacting temporal analyses.

2. Exploratory Data Analysis (EDA)

- **Message Type Distributions:** Each dataset showed distinct patterns (e.g., HDFS dominated by INFO, Apache by ERROR, HealthApp by OTHER).
- **Entry Length and Word Count:** Distributions revealed structural differences, verbosity, and outliers across datasets.
- **Component Extraction:** Top and rare components were identified, providing insight into system behavior and potential anomalies.
- **Temporal Patterns:** Activity was visualized by hour, day, and month, revealing operational cycles and gaps.

3. Feature Engineering

- Extracted features included message type, component, word count, entry length, special character count, and numerical value stats.
- Uniqueness ratios and rare component counts were computed to assess log diversity and repetitiveness.

4. Clustering and Visualization

- **KMeans, DBSCAN, Hierarchical Clustering, t-SNE:** Applied to log entries to reveal

dominant patterns, cluster structure, and outliers.
- Subsampling and dimensionality reduction were used for computational feasibility.
- Most datasets showed clear, well-separated clusters, but DBSCAN often found only one cluster due to high similarity or parameter choices.

5. Drift Detection and Statistical Analysis

- **Windowed Drift Tests:** Kolmogorov-Smirnov (KS) and Chi-squared tests were used to detect structural and semantic drift between consecutive time windows.
- **Change Point Detection:** The `ruptures` library identified abrupt changes in error rate time series, often aligning with drift windows.
- **Drift Windows:** Detected drift was investigated by comparing log entries before and after flagged windows, revealing operational changes (e.g., onset of web server errors).
- **Visualization:** Time series plots with drift points provided clear communication of when and where drift occurred, but also exposed timestamp normalization issues.

6. Key Results and Insights

- **High Data Quality:** No empty or malformed entries; robust log reading pipeline.
- **Distinct Log Structures:** Each dataset exhibited unique message type, length, and component patterns.
- **Drift Events:** Multiple statistically significant drift windows were detected, often corresponding to clear changes in log content or error rates.
- **Operational Insights:** Drift analysis pinpointed periods of system change, potential incidents, or external probing.
- **Challenges:** Timestamp inconsistencies and default values (e.g., 1900) affected some temporal analyses and visualizations.

7. Recommendations and Future Work

- **Improve Timestamp Handling:** Normalize and validate timestamps to ensure reliable temporal analysis.
- **Automate Drift Monitoring:** Integrate statistical tests and change point detection into an automated pipeline for real-time alerting.
- **Root Cause Analysis:** Investigate drift windows in detail to link log changes to system events or incidents.

- **Extend Feature Set:** Explore additional features (e.g., NLP embeddings, anomaly scores) for deeper insights.

In summary, this analysis provides a robust, reproducible framework for unsupervised log mining and drift detection, yielding actionable insights into system behavior, data quality, and operational change.

---

t-SNE Cluster Visualization: Interpretation and Insights

t-SNE visualizations were used to project high-dimensional log entry features into two dimensions, allowing us to visually inspect the results of clustering (e.g., KMeans with k=5) for each dataset.

Main Findings:

- **Clear Cluster Separation:** Most datasets (HDFS, Apache, BGL, HealthApp, HPC, Linux, Mac) show well-separated clusters in t-SNE space, indicating that log entries naturally group into a few dominant patterns or templates.
- **Cluster Meaning:** Each cluster typically corresponds to a major log template, event type, or system behavior (e.g., routine operations, error events, user actions).
- **Outliers:** A small number of points are scattered away from the main clusters, representing rare or anomalous log entries. These are potential indicators of drift or system incidents.
- **Subsampling Impact:** Subsampling (e.g., 2,000 entries per dataset) was necessary for computational feasibility. While it captures dominant patterns, some rare clusters or subtle anomalies may be missed.

Implications:

- The strong cluster structure confirms that unsupervised clustering is effective for summarizing log diversity and identifying key operational patterns.
- Outliers and small clusters are valuable for drift and anomaly detection, as they often

correspond to unusual or new system behaviors.

- t-SNE plots provide an intuitive way to communicate log structure and clustering results to both technical and non-technical audiences.
