Objective:

Your company runs a collaborative document-editing platform. The product emits raw JSON logs representing user interactions. You've been asked to build a pipeline that processes and models this data for downstream analytics and reporting.

Part 1: Data Ingestion

Input: Provide a folder of JSON logs (`/data/events/`) representing events like `user_login`, `document_edit`, `comment_added`, and `document_shared`.

Tasks:

- Create a Python/SQL-based pipeline that reads raw logs from local files (simulate ingestion from a streaming service).
- Normalize and clean the data into structured tables using your tool of choice (Pandas, Spark, or dbt if preferred).

Part 2: Data Transformation

Goals:

- Parse event data into the following normalized tables:
  - `users`: deduplicated list of users
  - `documents`: list of documents with metadata
  - `events`: flattened schema of all events with timestamps and event-specific fields

Transformations:

- Handle malformed or duplicate events gracefully.
  Add derived columns: e.g., `day_of_week`, `session_duration`, `document_word_count`.

Part 3: Data Storage

- Store the transformed data in a local **PostgreSQL** database (or SQLite if setup time is a concern).
- Ensure relational integrity (e.g., foreign keys from `events` to `users` and `documents`).

Part 4: Analytics Layer

Expose the following metrics using SQL queries or Python notebooks:

1. Daily Active Users (DAU) over the last 30 days
2. Average session duration by user
3. Top 10 most edited documents
4. Number of shared documents per user
5. Any anomaly you identify and explain

Bonus (if time permits):

- Package the solution in Docker.
- Create a simple Airflow or Prefect DAG to orchestrate ingestion + transformation.
- Add unit tests using `pytest`.

Submission Requirements

- GitHub repo or zipped folder with:
    - Code and scripts (ETL, DB setup, etc.)
    - Instructions to run (`README.md`)
    - SQL/notebook with the required queries
- Data model diagram (ERD) — a simple draw.io or markdown table is fine
- Optional: brief write-up (1–2 paragraphs) describing your architecture choices

---

Evaluation Criteria

Category	Weight	What We Look For
Code quality	25%	Clean, modular, readable
Data modeling	20%	Proper normalization, integrity
Problem solving	20%	Handling edge cases, assumptions explained
SQL/analysis	20%	Correctness and relevance of metrics
Bonus execution	15%	Dockerization, orchestration, testing