Assignment — Build the first stage of a modern data platform (Kafka → Spark → Parquet, orchestrated with Airflow)

Context (short):
You will implement the first streaming ingestion pipeline of a modern data platform. The goal for this assignment is to design and deliver a reproducible data ingestion flow that produces synthetic or real-time-like data into Kafka and consumes it with Spark (streaming or Spark micro-batches), landing the results as partitioned Parquet files on the local filesystem. This is the foundation for later analytics exercises.

Requirements (what you must build):

1. **Producer**

    - Implement a Python-based Kafka **producer** that generates a continuous stream of records from a realistic source of your choice (pick one and state it clearly): e.g., weather sensors, ecommerce clickstream, mobile app events, IoT telemetry, financial ticks, etc.

    - Records should be structured (JSON). Include a short schema and an example record in your repo (timestamp + at least 4 other meaningful fields).

    - The producer must write to **one or more Kafka topics**; if more than one, explain the rationale for split (e.g., different event types). Ensure partitioning strategy is sensible (explain your key choice).

2. **Kafka**

    - Use Kafka as the transport/messaging layer. Provide configuration details to run Kafka locally (topic creation, partitions, retention settings you used). You may use a local Kafka. Provide the docker-compose to start required services.

3. **Spark Consumer**

    - Implement a Spark job in Python (PySpark) that consumes the topic(s) using **Spark Structured Streaming** or, since streaming APIs were not covered in class, implement Spark **micro-batch** reads scheduled by Airflow (see orchestration below).

- The Spark job must transform/normalize the incoming messages into a tabular schema and write the results to the local filesystem in **Parquet** format.

- Parquet files **must be partitioned** by the most significant analytic dimension (choose one and justify it, e.g., `date`, `country`, `event_type`, `category`), and the partition layout must be clearly visible on disk (e.g., `output/date=2025-10-30/...`).

4. **Orchestration (Airflow)**

   - Provide an **Airflow DAG** that runs and coordinates the Spark consumer(s). If using continuous streaming jobs, your DAG should manage job submission, monitoring and graceful restarts; if using micro-batches, schedule ingestion runs (e.g., every minute / 5 minutes) and manage dependencies.

   - DAG must be runnable with a local Airflow setup; provide clear instructions to start Airflow and trigger the DAG.

5. **Deliverables**

   - Source code for producer, Spark consumer, Airflow DAG(s).

   - `docker-compose.yml` or scripts to start Kafka (and any ancillary services) locally. (You may use Docker; if you do not, provide exact shell commands.)

   - README with architecture diagram (simple ASCII or embedded image), schema, run instructions, and justification for design choices (topic layout, partition key, micro-batch vs continuous streaming).

   - Short document (max 1 page) with:
     i. schema, run instructions, and justification for design choices (topic layout, partition key, micro-batch vs continuous streaming).
     ii. describing the testing strategy you used (how you validated correctness and completeness of data landing) to ensure data quality.

Topics to be focused on:

- Correctness: messages flow from producer → Kafka → Spark → Parquet and Parquet files are partitioned as required.

- Reproducibility: you can start your stack locally and reproduce results.

- Schema & design: clear schema, sensible topic/partitioning strategy.

- Code quality: readable, modular Python code, sensible logging and error handling.

- Orchestration: Airflow DAG works and demonstrates scheduling/monitoring.
- Documentation: clear, concise and justification statements.
- Bonus: try to implement spark streaming using online resources.