Pipeline Design Overview

The pipeline is designed to avoid doing unnecessary work. It first checks if the required data, like **customer journeys** or database tables, already exists. If everything's in place, it moves on to the next step, whether it's generating a CSV or interacting with an API. The use of **simple checks** helps avoid repeating tasks like re-fetching data or sending the same information to the API. This design makes the process smoother and flexible, allowing changes to be added easily without overcomplicating things. The core steps include:

1. **Check for Pre-Existing Data**
   - If `customer_journeys.json` exists, reuse it.
   - If `channel_reporting` exists, proceed to CSV generation.
   - Otherwise, send data to IHC API and populate `channel_reporting`.
2. **Customer Journey Extraction**
   - Extract session data for each conversion.
   - Store extracted journeys in `customer_journeys.json`.
3. **Sending Data to the IHC API**
   - Send journeys in chunks to comply with API limits.
   - Store results in `attribution_customer_journey`.
4. **Channel Reporting & Metrics**
   - Populate `channel_reporting` from relevant tables.
   - Compute **CPO** (Cost per Order) and **ROAS** (Return on Ad Spend).
5. **Final Report Generation**
   - Generate CSV with user confirmation.
   - Check data integrity using IHC sum condition.

Assumptions

- Database tables exist and have the required structure.
- The API accepts and returns valid attribution results.

Potential Improvements

- Replace print() statements with logging to allow different logging levels and better debugging.
- Mock External Dependencies: Use mocking frameworks like unittest.mock to simulate database queries and API calls, ensuring tests.

These enhancements will improve the pipeline’s robustness and efficiency.