Data Engineering Take-Home Assignment

🎯 **Multi-Source Profile Data Pipeline**
Expected time: 4-5 hours | **Role context:** Data Engineer at a 0-1 startup | **Primary focus:** Performance, correctness, data quality, communication

Priority Guide

If you're short on time, focus on what matters most:

Priority	Parts	What we care about
Critical	Part 4 (BQ → Firestore Sync), Part 3 (Merge Logic)	Can you solve our real bottleneck and handle messy data?
Important	Part 1 (Profiling), Part 2 (Schema)	Do you understand the data and model it well?
Writeup	Part 5 (Design Questions)	Do you think about production and blast radius?
Required	Part 6 (README)	Can you communicate to non-engineers?

Submission

- Submit as a **GitHub repository**
- Any language or tools or agents allowed
- **Working code required** — this assignment should run against the sandbox
- Diagrams welcome
- README required
- **Clarity > polish**
- Please submit **one hour before your scheduled interview call**

- Please **clean up any sandbox resources** that are not part of your final submission or demo

---

Context

We ingest large **profile datasets from multiple third-party sources** to power internal tools and product features such as search, matching, and discovery.

Key characteristics of our data and system:

- Data arrives as **periodic snapshots**, refreshed at different times
- Each source has **different schemas and data quality**
- The sources contain **mostly distinct profiles**, with **potential partial overlap**
- Overlap, when it exists, should be eliminated
- The pipeline is: **GCS → BigQuery → Firestore**
- Data should be loaded into **BigQuery** for transformation
- The cleaned data should be synced into **Firestore**, the **serving layer** used by the product
- Our **main pain point** today is that the BigQuery → Firestore sync is **slow, expensive, and brittle**

Your task is to design and implement a pipeline that could realistically be shipped by a small team at a 0-1 startup.

Derived Fields Requirement

The final data served from Firestore must include the following derived fields for each profile:

- **Primary portfolio** — the user's main professional domain/function (e.g., "Software Engineering", "Product Management", "Data Science", etc.). You should define which portfolios you deem relevant and how you categorize profiles into these portfolios.
- **Years of experience** — total years of experience in that primary portfolio

How and where you compute these is a design choice — justify your approach.

Provided Data

You are given datasets from two different external sources. Both sources contain professional profile data sourced from LinkedIn, but each vendor structures and names fields differently.

The data is available in Google Cloud Storage in the GCP project `coffeespace-sandbox`:

- **Source A:** `gs://coffeespace-sandbox-source-1/` — a single JSONL file from one vendor
- **Source B:** `gs://coffeespace-sandbox-source-2/` — a collection of JSON files from a different vendor (single logical dataset, split is a vendor export artifact)

Authentication

You will be granted Editor access to the project. To authenticate locally:

```
gcloud auth login
gcloud config set project coffeespace-sandbox
gcloud auth application-default login
```

Part 1: Data Profiling & Quality Assessment

1. Profile each source independently
2. Document the **top 3-5 data quality issues** per source
3. Compare the two sources: which fields are more reliable in which source, where do values conflict, and where does one source add coverage the other lacks?

🔍 **What we're evaluating:** Ability to reason about messy, real-world data, judgment around trust and reliability, avoiding over-cleaning or silent mutation

Part 2: Canonical Data Model

Design a **canonical `people` schema** that merges both sources.

Requirements

- Supports downstream usage (search, matching, serving)
- Preserves source traceability
- Handles conflicting values
- Works with periodic refreshes
- Avoids silent data loss

Deliverables

- Schema definition
- Field-level rationale
- Example merged record

🏗️ **What we're evaluating:** Modeling maturity — strong candidates think in **entities, grain, and semantics**, not columns

---

Part 3: Cleaning, Normalization & Merge Logic

1. Normalize Source A and Source B into the canonical schema
2. Create a **unified dataset (union)** of profiles with provenance
3. Handle dirty or inconsistent fields explicitly

The two sources are **mostly distinct** — do not assume every profile from Source A has a match in Source B.

Cross-Source Deduplication

- Handle **duplicate users across sources** (i.e., records that represent the same real-world person)
- Apply deduplication deterministically
- This logic should be safe to run repeatedly (idempotent) and will be used during periodic refreshes

Deliverables

- Clear deduplication and merge rules
- Implementation of the merge logic

> 🔧 **What we're evaluating:** Real DE work — this is **actual purchased-data engineering**, not textbook ETL

Part 4: BigQuery → Firestore Sync

Efficiently sync the merged dataset from BigQuery into Firestore. There is no pre-existing Firestore database — creating one is part of the exercise.

Requirements

- Incremental
- Idempotent
- Resumable
- Designed for Firestore write limits

Performance Report (Required)

After running your sync, document the following:

- Total number of documents synced
- Total runtime
- Throughput (docs/sec)
- Number of retries or failures
- Any bottlenecks you observed

> 🚀 **What we're evaluating:** Can you solve our real production bottleneck?

Part 5: Design Questions (Writeup)

Answer the following in a short writeup. No implementation needed.

1. Every quarter, new snapshots arrive and existing records may change or disappear. **How would you handle snapshot versioning, delta detection, and rollback** if a bad snapshot is deployed?
2. Before promoting new data to the serving layer, **what validation and data quality checks would you run** to prevent bad data from reaching users?
3. A vendor delivers a new periodic snapshot with 30% fewer records than the previous one. **How do you determine whether this is a bad export or a legitimate change**, and how should the pipeline handle each case?
4. On deduplication: suppose Vendor B has more recent data for a candidate initially sourced from Vendor A, but some fields are null or missing compared to Vendor A's version. **How would you design your merge logic to handle this?**
5. You need to add a third data source with a completely different schema next quarter. **What in your current pipeline design accommodates this, and what would need to change?**
6. Your sync handles the current dataset. **How would you modify it to scale to tens of millions of records?** What would change in your batching strategy, error handling, and infrastructure?

What we're evaluating: Production thinking — do you understand **write amplification**, lifecycle management, and **blast radius**?