Take-Home Assignment: Patient Matching Pipeline

Estimated Completion Time: 6–8 hours

Objective

You are provided with two CSV files — `internal.csv` and `external.csv` — each containing a list of patients with the following fields:

- `FirstName`
- `LastName`
- `DOB` (Date of Birth)
- `Sex`
- `PhoneNumber`
- `Address`
- `City`
- `ZipCode`

The `internal.csv` file represents patients from our hospital system. The `external.csv` file contains patients from an outside medical practice.

Your task is to build a **patient matching pipeline** that attempts to identify likely matches between the two datasets. Keep in mind:

- There is no shared unique identifier between the files
- Fields may contain typos, missing values, inconsistent formatting, or updated information (e.g., new phone numbers or name changes)

What You'll Build

1. Relational Database (SQL-Compatible)

Design and populate a SQL-compatible database (e.g., SQLite, PostgreSQL, MySQL, BigQuery) using the provided CSV files. At a minimum, your database should include the following tables:

- `internal` (based on `internal.csv`)
- `external` (based on `external.csv`)
- `matches` (based on your matching algorithm — see Step 2)

You may include additional tables as needed (e.g., staging tables, logs, scoring details, or archives).

The `matches` table should contain one row per best match between external and internal patients. External patients with no likely match should be excluded. At a minimum, this table should include:

- `ExternalPatientId`
- `InternalPatientId`

---

2. Matching Algorithm

Implement a matching algorithm that can accommodate data imperfections while minimizing false positives. Your solution should consider:

- Data normalization (e.g., converting to lowercase, removing punctuation/whitespace)
- Exact matching
- String similarity scoring (e.g., Levenshtein distance)
- A hybrid approach that combines strict and soft matching logic

You are welcome to use SQL or a programming language of your choice to implement your algorithm. However, if using a programming language, **you must implement your own string similarity logic** — do not use third-party fuzzy matching libraries such as `fuzzywuzzy`, `rapidfuzz`, or `difflib`.

Below are optional Python starter functions you may use or modify:

```
# Levenshtein distance
def levenshtein_distance(s1, s2):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)
    previous_row = list(range(len(s2) + 1))
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]

# Similarity ratio
def similarity_ratio(s1, s2):
    max_len = max(len(s1), len(s2))
    return 1.0 if max_len == 0 else 1.0 - levenshtein_distance(s1, s2) / max_len

# Token-based overlap
def token_overlap_score(s1, s2):
    tokens1 = set(s1.split())
    tokens2 = set(s2.split())
    return len(tokens1 & tokens2) / len(tokens1 | tokens2) if tokens1 and tokens2 else 0.0
```

---

3. End-to-End Data Pipeline

Build a callable script or procedure that:

- Accepts a new `internal.csv` or `external.csv` file
- Updates the corresponding table(s) in the database
- Runs your matching algorithm
- Updates the `matches` table and any supporting/intermediate tables

---

Deliverables

Upload your project to a **public code repository** (e.g., GitHub, GitLab). Be sure to include:

- `README.md` containing:
  - Setup instructions
  - A high-level explanation of your matching algorithm and assumptions
- `matches.csv` (exported from your `matches` table), which should include:
  - `ExternalPatientId`
  - `InternalPatientId`

What to Expect During the Interview

During your virtual interview, you will be asked to:

- Share your screen
- Explain your matching algorithm in plain English (as if speaking to a non-technical audience)
- Walk through your code and database schema
- Connect to your database and write SQL queries to answer questions about the data during the interview