Take-Home Assignment: Patient Roster Matching Web Application

Estimated Completion Time: 6-10 Hours

You are provided with two CSV files: `internal.csv` and `external.csv`. Each file contains a list of patients with the following fields:

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

The **internal file** represents patients from our hospital system. The **external file** represents a patient roster from an outside medical practice.

Objective

Build a **local-only web application** that compares the two patient lists and attempts to identify **likely matches**, recognizing that:

- There is **no unique patient ID** shared across the files
- Fields may contain **typos**, **missing data**, **extra data**, or **changed information** (e.g., phone numbers, last names, addresses)

Your Solution Should Include:

1. Matching Algorithm

Design and implement a flexible algorithm that identifies **probable matches** while minimizing false positives. Consider: Normalization (e.g., stripping punctuation/whitespace, converting to lowercase), exact matching, string similarity matching, a hybrid approach combining strict and soft comparisons.

You may use **any programming language or framework** you prefer. However, please implement your own string similarity logic and **do not use third-party fuzzy matching libraries** such as `fuzzywuzzy`, `rapidfuzz`, or `difflib`.

Example Python starter functions (optional use):

```
# Basic 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 based on edit distance
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 score
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
```

2. Review Interface

Develop a simple **web-based UI (localhost only)** that allows a user to:

- View suggested matches (i.e., an internal patient and an external match)
- Confirm or reject each match
- Optional: display a **match confidence score** or a breakdown of how the match was determined

You may use any framework or library you'd like (e.g., Flask, FastAPI, React, Node.js, Streamlit, etc.) or a simple front-end stack (HTML/JS/CSS).

The application **does not need to be deployed** — it should run locally.

Deliverables

- A link to a publicly available code repository (e.g., GitHub, GitLab, Bitbucket, etc.) containing the project. *Note: We will not be able to access file sharing services (e.g., Dropbox, Google Drive)*. The repo should contain a `README.md` file that includes:
    - Setup instructions
    - A description of how your matching algorithm works
- A file named `matches.csv` containing the **probable matches** – each row should represent the **best match** for that external patient. External patients without a likely match should be excluded from the file.
    - This file should contain two columns:
        - ExternalPatientId
        - InternalPatientId

What to Expect During the Interview

During the interview, you will be asked to share your screen and:

- Demo your UI
- Explain your matching algorithm
- Walk through your code, thought process, and design decisions
- Answer follow-up questions related to enhancements, productionization, troubleshooting, etc.