# Jal Jeevan Mission Water Quality
# Data Fuzzy Matching

July 27, 2024

# Need for Fuzzy Matching

To use JJM Water Quality data for further analysis, we need to merge it with other datasets like shape files for mapping or data on agriculture, irrigation practices, urbanization, deforestation, slope, elevation, vegetation cover, soil type, and precipitation for deeper insights. The main challenge is that merging requires common columns, but inconsistencies in naming for states, districts, sub-districts, and villages can cause issues. Therefore, we first need to standardize these names with a reliable reference, such as Census names, before proceeding with the merge.

**To fuzzy match effectivly**

- First, we fuzzy match State names of JJM Water Quality Data to the Census Data.
- Next, we we fuzzy match District of a state from JJM Data with districts of the same state from census data rather than fuzzy matching with all the districts of India. THis helps in reducing time complexity as well as accuracy.
- Similarly further for same Districts its Villages are mapped and so on.
- Finally, data are merged on Mapped State, District, Subdistrict and Villages.

## Example

### Step 1: Mapping State Names

Mapping created for State Names:

```
'KARNATAKA': 'Karnataka', 'KERALA': 'Kerala'
```

### Step 2: Mapping District Names

Mapping created for Karnataka's District Names:

```
'SHIMOGA': 'Shimoga', 'TUMKUR': 'Tumkur', 'UDUPI ': 'Udupi',
'YADGIR': 'Yadgir'
```

Mapping created for Kerala's District Names:

```
'ALAPPUZHA': 'Alappuzha'
```

### Step 3: Mapping Block Names

Mapping created for Shimoga's Block Names:

```
'BHADRAVATI': 'Bhadravati', 'HOSANAGARA': 'Hosanagara',
'SAGAR': 'Sagar', 'SHIKARPUR': 'Shikarpur', 'SHIMOGA': 'Shimoga',
'SORAB': 'Sorab', 'TIRTHAHALLI': 'Tirthahalli'
```

Mapping created for Tumkur's Block Names:

```
'GUBBI': 'Gubbi', 'KORATAGERE': 'Koratagere', 'SIRA': 'Sira',
'MADHUGIRI': 'Madhugiri', 'PAVAGADA': 'Pavagada',
'TIPTUR': 'Tiptur', 'TUMKUR': 'Tumkur', 'TURUVEKERE': 'Turuvekere'
```

Mapping created for Udupi's Block Names:

```
'KARKAL': 'Karkal', 'KUNDAPURA': 'Kundapura', 'UDUPI': 'Udupi'
```

Mapping created for Alappuzha's Block Names:

```
'AMBALAPUZHA': 'Ambalappuzha', 'ARYAD': 'Kuttanad'
```

### Step 4: Mapping Villages Names

Mapping created for Shimoga's Villages Names:

```
'AGARADAHALLI': 'Agaradahalli', 'ANAVERI': 'Anaveri',
'ITTIGEHALLY': 'Ittigehalli', 'ANTHARAGANGE': 'Antaragange',
'ARABILACHI': 'Arebilachi', 'ARAHATHOLALU': 'Arahatholalu',
'ARAKERE': 'Arakere', 'DANAVADI': 'Danavadi', 'KALLAPURA':
'Kallapura', 'ATTIGUNDA': 'Athigunda', 'BARANDURU': 'Barandooru',
'BILAKI': 'Bilaki', 'MAJJIGENALLI': 'Majjigenahalli',
'THYANANDURU': 'Tyananduru'
```

Mapping created for Tumkur's Villages Names:

```
'KACHENAHALLI': 'Kachenahalli', 'BOMMENAHALLI': 'Bommenahalli',
'HESARAHALLI': 'Hesarahalli', 'NITTUR': 'Nittur', 'KUNNAGHATTA':
'Kunaghatta', 'MELEKALLAHALLI': 'Melekallahalli', 'PEDDANAHALLI':
'Peddanahalli', 'GALIGEKERE': 'Galigekere'
```

Mapping created for Udupi's Villages Names:

```
'BAILOOR': 'Bailoor', 'KOWDOOR': 'Kowdoor', 'BELMAN': 'Belman',
'NANDALIKE': 'Nandalike', 'SOODA': 'Sooda', 'BOLA': 'Bola',
'EEDU': 'Eedu', 'NOORALBETTU': 'Nooralbettu', 'CHARA': 'Chara',
'HEBRI': 'Hebri', 'HIRGANA': 'Hirgana', 'INNA': 'Inna', 'KADTHALA':
'Kadthala', 'YELLARE': 'Yellare', 'KANTHAVARA': 'Kanthavara',
'BELENJE': 'Belenje', 'KUCHCHUR': 'Kuchchur', 'KUKKUNDOOR':
'Kukkundoor', 'MALA': 'Mala', 'HERMUNDE': 'Hermunde', 'MARNE':
'Marne', 'MIYAR': 'Miyar', 'MUDAR': 'Mudar', 'KABBINALE':
'Kabbinale', 'MUDRADY': 'Mudrady', 'MULLADKA': 'Mulladka',
'MUNDKURU': 'Mundkuru', 'NALLUR': 'Nallur', 'KANAJARU': 'Kanajaru',
'NEERE': 'Neere', 'NITTE': 'Nitte', 'YADTHADI': 'Athradi',
'YELLUR': 'Yellur'
```

Mapping created for Alappuzha's Villages Names:

```
'AMBALAPPUZHA': 'Ambalappuzha', 'KARUMADY': 'Karumady', 'KALAVOOR':
'Kalavoor', 'PATHIRAPPALLY': 'Pathirappally'
```

### Step 5: Merging

# How Fuzzy Matching works

Fuzzy Matching involves 4 steps

1. Tokenization The strings are split into smaller units (tokens), such as words or characters. This helps in comparing the components of the strings more effectively.

2. Normalization The strings are normalized by converting them to a common case (e.g., all lowercase) and removing any non-alphanumeric characters.

3. Scoring The similarity between strings is quantified using a scoring mechanism. fuzzywuzzy uses the Levenshtein distance to calculate the differences between sequences. The Levenshtein distance is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other.

4. Ratio Calculation The similarity score is then converted into a ratio, usually expressed as a percentage. This ratio reflects the degree of similarity between the two strings

**Example**

**Tokenization and Normalization**

- **Original Strings:**
  - string1: "Apple Inc."
  - string2: "Apple Incorporated"
- **After Normalization:**
  - string1: "apple inc"

– string2: "apple incorporated"

**Levenshtein Distance Calculation**

The Levenshtein distance between "apple inc" and "apple incorporated" involves calculating the minimum number of edits needed to transform one string into the other. This can be done via dynamic programming, where we build a matrix to track the minimum edits.

**Ratio Calculation**

The ratio is calculated based on the Levenshtein distance and the lengths of the strings. For example, if the distance is 12 and the length of the longer string is 18, the similarity ratio might be calculated as:

$$\text{ratio} = \left(1 - \frac{\text{distance}}{\max(\text{len}(\text{string1}), \text{len}(\text{string2}))}\right) \times 100 \tag{1}$$

**Finding the Best Match**

The `process.extractOne` function tokenizes and normalizes the input string and the choices. It calculates the similarity ratio for each choice. The choice with the highest similarity ratio is returned as the best match.