# Overview

The solution provided aims to identify in a given set of data entries that are same or similar and flag them accordingly.

The user is in the driving seat when it comes to selecting the columns to use for matching. This is done via the input() function.

The filename and matching score is passed to the script and it is for the purpose of this hard coded ☹

## Dependencies

The solution is written in a windows environment using Python 3.7.

It makes use of the following non-standard libraries:

fuzzywuzzy pip3 install fuzzywuzzy (for matching purposes)

python-Levenshtein pip3 install python-Levenshtein (for matching purposes)

pandas pip3 install pandas (for filtering data etc)

## Assumptions

* The input file contains a header record
* The name and surname within the data is not transposed. However, there is an option in fuzzywuzzy matching that ignores the order but not in SequenceMatcher.
* The user will always select columns for matching and not continue without. There is some error checking within the input() function.

## Running of the script

I have stored the data in

C:\\Fuzzy\_Match\\Input\_Data\\Test\_Data.csv

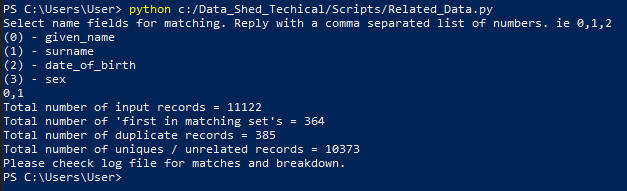
And the script resides in

"C:\\Fuzzy\_Match\\Scripts\\Related\_Data.py"

The application is using the script path location to output the logs and output files. So strictly speaking you do not need to follow this folder structure.

Below is the command to execute the script and a screen grab output.

python C:\Fuzzy\_Match\Scripts\Related\_Data.py



As I mentioned the 2 parameters passed to the script are currently hardcoded

if \_\_name\_\_ == '\_\_main\_\_':

    # # Hard coded params

    file\_name = "C:\\Fuzzy\_Match\\Input\_Data\\Test\_Data.csv"

    match\_ratio = 90

    # # Run main function

    main(file\_name, match\_ratio)

# Logic applied

The idea behind my logic was to create a ‘match\_key’ which would enable me to tie records together.

### Steps

#### 1 – Read input file in to a data frame

#### 2 – User input of field selection

Ask the user to confirm which fields to use for matching (using the header of the file)

#### 3 – Update the data frame with additional columns

‘match\_key’ and ‘match\_level’

#### 4 – Sort the data frame

On **‘match\_key’** column

#### 5 – Check for matches

I used two matching libraries as some of the results were contradicting. The match score is set to 90 for both.

The idea here is that we loop through the sorted records comparing the current index to the next ie **list[x]** with **list[x+1]**.

**If list[x] and list[x+1]** is a match then

we mark it as

**list[x] = 'First in matching set'**

unless it was ‘flagged as a duplicate previously‘

**list[x] = 'Duplicate'**

**If not a match then**

we either mark it as

**list[x] = 'Unique'**

unless it was ‘flagged as a duplicate previously’

**list[x] = 'Duplicate'**

flagged as a duplicate previously – This becomes important as we need to keep track of the state of the **list[x+1]** record as this will become the **list[x]** record in the next iteration of the loop. For this reason **list[x+1]** is stored in a variable to keep track of whether the index was marked as 'Duplicate' before.

Example to explain:

First iteration of the for loop – the consecutive entries are the same

**list[x] = ‘A’** This will be marked as **‘First in matching set’**

**list[x + 1] = ‘A’** The index of this will be stored as a duplicate of the previous

**dupe\_index = x + 1**

Second iteration of the for loop – the consecutive entries are the different

**list[x] = ‘A’** This will be marked as **‘Duplicate’** as opposed to **‘Unique’** because the index will be the same as the stored index. **x = dupe\_index**

**list[x + 1] = ‘B’** The index of this will not be stored as it is not a duplicate of the previous.

During this process the ‘**match\_level**’ column will be populated with the following flags:

**‘First in matching set’**

**‘Duplicate’**

**‘Unique’**

#### 6 – Do some counts

The counts are based on the ‘**match\_level**’ column populated during the matching process.

#### 7 – Output related data to files

A little clarification on terminology that I used:

**'first in matching set'** = is one of the records that has a related entry in the file.

**'Duplicate'** = would be the accompanying entry to the above.

**'matching sets'** = would be records flagged as 'first in matching set' and 'Duplicate'.

**'Unique'** = would be records that are not related to the above 'matching sets'.

So if you wanted to have a clean file then an output of

**'first in matching set' + 'Unique set'** will give you that.

If you wanted to compare the matches then an output of

**'matching sets'** will help you.

3 files are output:

relateddata\_duplicates.csv Records flagged as **‘Duplicate’**

relateddata\_matching\_sets.csv Records flagged as ‘**matching sets’**

unrelateddata.csv Records flagged as **'first in matching set' + ‘Unique’**

# Tests

## Input file related

* Check that file exist (throws an error)
* Check that file is not empty (throws an error)
* Check for inconsistencies in delimiter numbers (throws an error)

## User input related

* Check that user enters a number
* Check that the number entered is within the upper boundary of column count
* Check that it is not a negative number

Entering letters, negative numbers or numbers are greater than the column count will not break the program. These will be ignored and the user will be notified via console.

# Future considerations

* Move away from hard coded params. Use a config file instead or argument parsing.
* Include ‘date of birth’ and ‘sex’ for more precise matching.
* Bit more refactoring of the script.
* Much better unit testing. I have not used automated testing, but reading up on it now.