**Record Linkage**

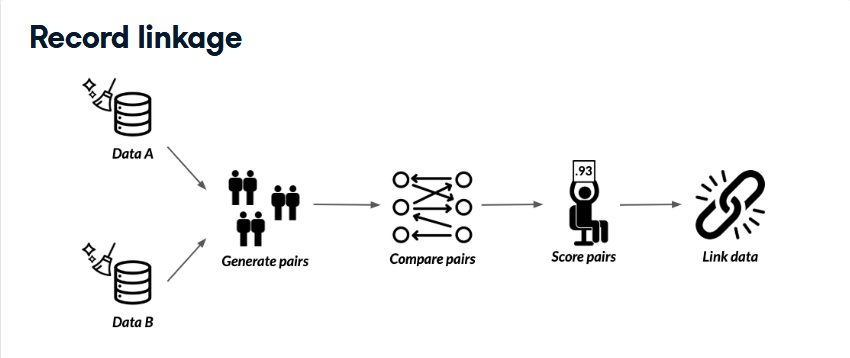
Record linkage is the act of **linking data sets** from different sources regarding the **same entity** with **avoiding the fuzzy duplicated values**.

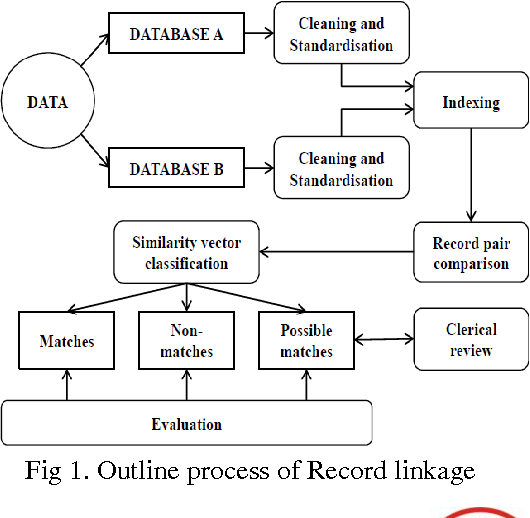
**Why joins don't work ?!**

suppose that here are duplicates values in both Data Frames and the features or columns are **differently named** or the data may be collected manually and are **prone to typos**. and there is **no common unique identifier** or **consistent IDs** between the Data Frames. a regular join() or merge() functions will not work in this case. This is where record linkage comes in.

**Record linkage steps.**

1. **Data cleaning and Standardization**. we clean two or more DataFrames, make sure the datatype is correct and in the same format the involves removing all whitespaces and converting all letters into lower or upper cases. to be appropriate and ready to generate pairs
2. **generate pairs of potentially matching records,**
3. **score these pairs according to string similarity and other similarity metrics**
4. **and link them.**





Let's explain the steps in much more details

**Generating pairs**

Ideally, we want to generate **all possible pairs** between our Data Frames. But what if we had big Data Frames and ended up having to generate millions if not billions of pairs?!

It wouldn't prove scalable and could seriously hamper development time.

**Blocking or indexing.**

This is where we apply what we call blocking, which creates pairs based on a matching column, reducing the number of possible pairs.

Let's cover generating pairs in much more technical details.

**Generating pairs**

we first start off by importing recordlinkage. We then use the recordlinkage.Index() function, to create an indexing object. This essentially is an object we can use to generate pairs from our DataFrames.

To generate pairs blocked on matching column, we use the block() method, inputting the matching column as input. Once the indexer object has been initialized, we generate our pairs using the index() method, which takes in the two dataframes.

The resulting object, is a pandas multi index object containing pairs of row indices from both DataFrames, which is a fancy way to say it is an array containing possible pairs of indices that makes it much easier to subset DataFrames on.

**Comparing the DataFrames**

we've already generated our pairs, it's time to find **potential matches**.

We first start by creating a comparison object using the recordlinkage.compare() function.

This is similar to the **indexing object** we created while generating pairs, but this one is responsible for assigning different comparison procedures for pairs. Let's say there are columns for which we want exact matches between the pairs. To do that, we use the exact(col\_name , label) method. It takes in the **column name** in question for each DataFrame, and a **label argument** which lets us set the column name in the resulting DataFrame.

Now in order to compute **string similarities** between pairs of rows for columns that have fuzzy values, we use the string(col\_name, threshold) method, which also takes in the column names in question, the similarity cutoff point in the **threshold argument**, which takes in a value between 0 and 1

Finally to compute the matches, we use the compute() function, which takes in the possible pairs, and the two DataFrames in question.

Note that you need to always have **the same order of DataFrames** when inserting them as **arguments** when **generating pairs**, **comparing between columns**, and **computing comparisons**.

**Finding matching pairs**

The output is a **multi index DataFrame**, where **the first index** is the **row index** from the first DataFrame, and **the second index** is a **list of all row indices** in the second DataFrame.

The columns are the columns being compared, with values being 1 for a match, and 0 for not a match.

**Finding the only pairs we want**

To find potential matches, we just filter for rows where **the sum of row values** is higher than a **certain threshold**

In the **final evaluation step** is the **quality of the linked records** are evaluated using a variety of measures

The complexity of a linkage can be measured by the number of candidate record pairs generated by an indexing or blocking technique. Measuring completeness and linkage quality requires truth data in the form of known true matching and non-matching record pairs. Linkage quality is generally measured using precision and recall, while completeness is similar to recall but measures how many of all known true matches are included in the set of candidate record pairs.