# Entity Resolution of Publication Data

Yuxin XUE, Ofir Topchy, Nevo Levi
Github link

## 1. Table of Contents

## 2. Abstract

In this project, we explore the development of data engineering and ML pipelines, with a specific emphasis on constructing an Entity Resolution (ER) pipeline for deduplicating research publication datasets.

The initial phase involves acquiring the datasets and transforming them from TXT to CSV format. Subsequently, we proceed to create a local entity resolution pipeline designed to merge related entities.

In the final stage, we were hands-on PySpark framework to reimplement our local pipeline on top of a data-parallel computation framework. We also evaluate the scalability of the module through comprehensive testing.

## 3. Quick Overview and User Instruction

### 3.1. Installation

The prerequisites in addition to the python packages are

> spark == 3.5.0 scala == 2.12.x graphframes == 0.8.3-spark3.5-s_2.12

graphframes can be installed by running `pyspark --packages graphframes:graphframes:0.8.3-spark3.5-s_2.12` and moving those jars to path-to-spark-home/libexec/jars

```
cd path-to-this-project
pip install -e .
```

### 3.2. Sample to Run Exercise

part1/2/3

```
from erp import part1, part2, part3
part1() # cleaned data stored in "data"
part2() # results of all methods stored in "method_results.csv"
part3() # scability test
```

## 3.3. Quick Project Overview

The project involves implementing an Entity Resolution Pipelining on citation networks from ACM and DBLP datasets.

### 3.3.1. Data Source

The project starts with two large dataset text files you need to download:

- [DBLP-Citation-network V8]
- [ACM-Citation-network V8]

(Can be found here)

Make sure to save these in the local `data` folder.

Below is the structure of the project:

- 📁 **project**
  - 📁 **erp**: Contains Python scripts for the entity resolution pipeline.
  - 📁 **data**: Stores datasets and instruction files.
  - 📁 **results**: Contains results of the entity resolution pipeline process.
  - 📄 `.gitignore`
  - 📄 `requirements.txt`
  - 📄 `setup.py`
  - 📄 `README.md`
  - 📄 `sample.py`

### 3.3.2. erp Folder

The erp folder contains scripts for the entity resolution pipeline with specific configurations:

- **Preparing Data**: Run `erp.preparing.prepare_data("path_to_txt_file")` for both text files. This will clean and extract the relevant data (1995-2004 citations by "SIGMOD" or "VLDB" venues). The resulting csv files will show in `data` folder.
- **Running Pipeline**:
  - Local Version : Run `erp.ER_pipeline(databasefilename1, databasefilename2, ERconfiguration, baseline=False, cluster=True,matched_output="path-to-output-file", cluster_output="path-to-output-file", isdp=False)` (in `erp/main.py` )
  - DP Version: Run `erp.ER_pipeline(databasefilename1, databasefilename2, ERconfiguration, baseline=False, cluster=True, matched_output=F"path-to-output-file", cluster_output="path-to-output-file", isdp=True)` (it calls `ER_pipeline_dp` in `erp/dperp.py` )
- **Configuration Options**:
  - `blocking_method` (String): Methods to reduce execution time `{“Year”, “TwoYear”, “numAuthors”, “FirstLetterTitle”, “LastLetterTitle”, "FirstOrLastLetterTitle", “authorLastName”, “commonAuthors”, “commonAndNumAuthors”}`.
  - `matching_method` (String): Algorithms for entity matching `{"Jaccard", "Combined"}`.
  - `clustering_method` (String): Altogirthm for clustering {"basic"}.
  - `threshold` (float): A value between 0.0-1.0 for the matching similarity threshold.
  - `output_filename` (String): path and file name of clustering results to be saved.

#### Selected Functions in local pipeline

- Blocking: `erp.blocking(df1,df2,blocking_method)`
  - Parameters:
    - df1,df2 (pandas.DataFrame) : input databases
    - blocking_method(str) : {“Year”, “TwoYear”, “numAuthors”, “FirstLetterTitle”, “LastLetterTitle”, “authorLastName”, “commonAuthors”, “commonAndNumAuthors”}
- Matching: `erp.matching(blocking_df,similarity_threshold, matching_method)`
  - Parameters:

        - blocking_df(pandas.DataFrame)
        - similarity_threshold (float from 0.0 to 1.0)
        - matching_method (String) : {"Jaccard", "Combined"}
- Clustering: `erp.clustering(matched_entities, df1, df2, clustering_method)`
    - Parameters:
        - matched_entities(pandas.DataFrame)
        - df1,df2 (pandas.DataFrame) : input databases
        - clustering_method (String) :{'basic'}

**Selected Configuration**

ERconfiguration:

```
{
  "matching_method": "Combined",
  "blocking_method": "FirstOrLastLetterTitle",
  "clustering_method": "basic",
  "threshold": 0.7,
  "output_filename": "clustering_results_local.csv"
}
```

- This folder also contains `dperp.py` , which serves as a reimplementation of the local entity recognition pipeline within the Apache Spark framework.

### 3.3.3. Results Folder

- The steps above will produce the results. They are saved according to your `output_filename` configuration. In our ERconfiguration shown above, it will be saved as `clustering_results_local.csv` within the `results` folder.
- This folder contains all the results that are calculated and used in part 2 and part 3.

### 3.3.4. Data Folder

The data folder includes the prepared and cleaned datasets and additional samples:

- `citation-acm-v8_1995_2004.csv` : ACM citation network dataset.
- `dblp_1995_2004.csv` : DBLP citation network dataset.
- `DIA_2023_Exercise.pdf` : Project instruction file.

**Note**: Check `requirements.txt` for compatibility before running the code.

# 4. Data Acquisition and Preparation (Part 1)

In this section, we acquire datasets related to research publications. These datasets, available in text format, can be reached by clicking here.

As a prerequisite for Entity Resolution and Model Training, we have generated a dataset containing the following attributes:

- Paper ID, paper title, author names, publication venue, year of publication
- Publications published between 1995 and 2004
- Publications from VLDB and SIGMOD venues

We utilized Pandas DataFrame, to convert the datasets from TXT to CSV. Our code iterates through the text file, extracting entries separated by double newlines and filtering based on the specified criteria. The resulting cleaned dataframes are exported to the local `data` folder.

> The code for this section can be found in the file named `preparing.py` under the function called `prepare_data` . Additionally, the resulting CSV files are available in the local `data` folder with the suffix `__1995_2004.csv` .

## 5. Entity Resolution Pipeline (Part 2)

We aim to apply an entity resolution pipeline to the aforementioned datasets, following the scheme depicted below:

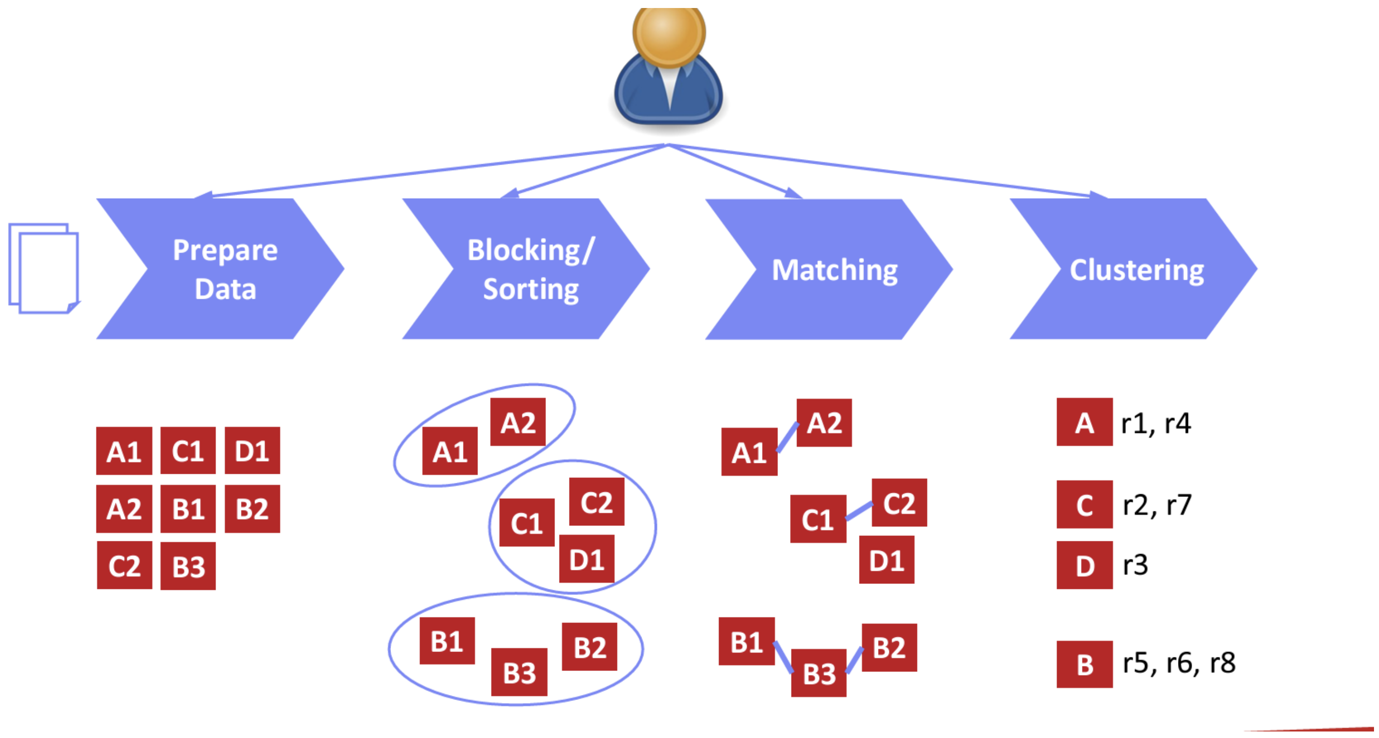


*Image Source: Prof. Matthias Boehm, Data Integration and Large-Scale Analysis Course, TU Berlin.*

### 5.1. Prepare Data

Continuing from the previous section, we employ various data-cleaning techniques. This step converts all characters to lowercase, ensures uniformity, and eliminates special characters, retaining only alphanumeric characters, spaces, and commas. This process standardizes and cleans the textual data for easier comparisonmand analysis.

> The code for this part is available in the file named `preparing.py` under the function called `prepare_data` .

### 5.2. Blocking

Blocking is employed to reduce the number of comparisons by using effective partitioning strategies. In each 'bucket', we run the comparisons (see the section below). Our blocking is achieved through partitioning based on attributes:

1. **Year:** Articles that were published in the same year would be in the same bucket.
2. **Two Year:** Articles that were published in the same year or in the adjacent year would be in the same bucket.
3. **Num Authors:** Articles with a similar number of authors (up to 1 difference) would be in the same bucket.
4. **Common Author:** Articles with at least one common author would be in the same bucket.
5. **Num Authors and Common Author:** Articles with at least one common author and a similar number of authors (up to 2 differences) would be in the same bucket.
6. **First Letter:** Articles with the same first letter of the title would be in the same bucket.
7. **First or Last Letter:** Articles with the same first letter or the last letter of the title would be in the same bucket.
8. **Last Name:** Articles with at least one author with a common last name would be in the same bucket.

> The code for blocking is in the file named `matching.py` , with functions named `blocking` and `create_xBlocking` , where x is the respective blocking method. (see selected functions for more detail to use them.)

### 5.3. Matching

Before discussing comparison methods, some terms related to our pipeline are introduced:

- Baseline - We establish a baseline by comparing every pair between datasets, given a certain similarity function applied. This is our 'ground truth'.
- Matched Entities - Our matched entities are generated by comparing each pair within a bucket, with the same similarity function applied to the respective baseline.

**Jaccard -** The Jaccard similarity function is employed to measure the extent to which two sets share common elements. It does so by calculating the ratio of the shared elements to the total elements in both sets. Thresholds of 0.5 and 0.7 are used in the comparison of the 'paper title' attribute.

**Combined -** This function calculates a combined similarity score between two papers based on their titles and author names. It utilizes Jaccard similarity for title comparison and, if available, trigram similarity for author name comparison. The final combined similarity score is a weighted sum of title and author name similarities, with 70% weight is assigned to the title and 30% to the author names. If author names are missing for either paper, the function defaults to using only the Jaccard similarity of titles.

For the blocking methods mentioned above:

**Jaccard** similarity function with **Year** partitioning identifies matching articles with similar titles published in the same year.

**Jaccard** and **Two-year** partitioning identifies matching articles with similar titles published in the same year or in the adjacent year.

**Jaccard** and **Num Authors** partitioning identifies matching articles with similar titles and a similar number of authors.

**Jaccard** and **First Letter** partitioning identifies matching articles with similar titles and the same first letter of the paper title.

**Jaccard** and **Last Letter** partitioning identifies matching articles with similar titles and the same last letter of the paper title.

**Jaccard** and **First or Last Letter** partitioning identifies matching articles with similar titles and the same first or last letter of the paper title.

**Jaccard** and **Authors last name** partitioning identifies matching articles with similar titles and the author's last name.

**Jaccard** and **Common Authors** partitioning identifies matching articles with similar titles and the same authors.

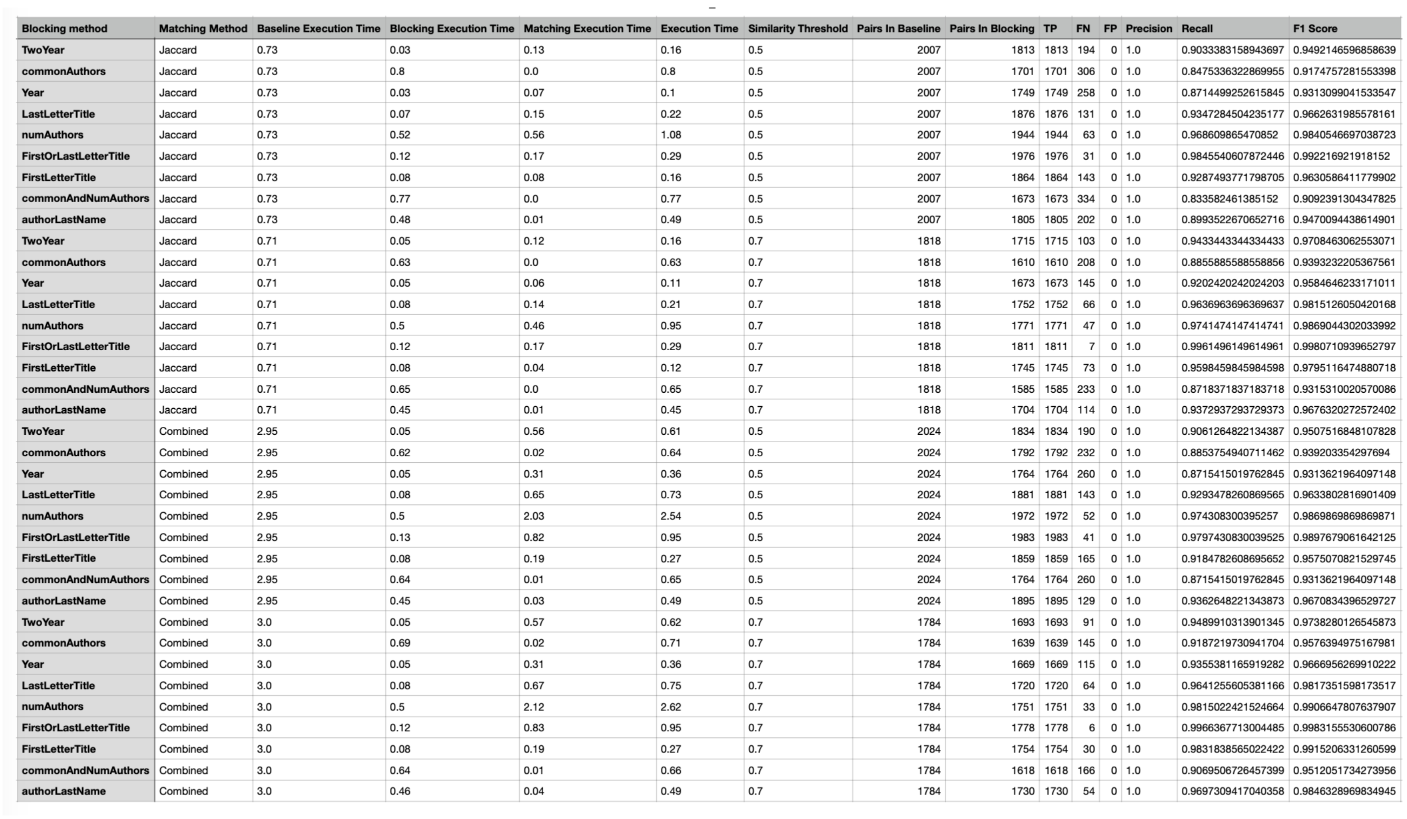
**Jaccard** and **Num of Authors** partitioning identifies matching articles with similar titles and the difference between their numbers of authors are smaller than 2.

**Jaccard** and **Num of Authors and Common Author** partitioning identifies matching articles with similar titles and at least one common Author with the difference between their numbers of authors smaller than 3.

Likewise, the **Combined** similarity will yield results for the different blocking methods, with the only difference being that it takes into account the number of authors in the comparison.

> The code for this part is available in the file named `matching.py` , with function `matching(blocking_results, similarity_threshold, matching_method, outputfile)` similarity functions named `calculate_x_similarity` , where x is the respective similarity method. CSV files for each similarity function and blocking method will be exported to a local `results` folder.

By testing various combinations, we obtained results that can be seen at ./source/results/method_results.csv, with a screenshot displayed below:



| Blocking method | Matching Method | Baseline Execution Time | Blocking Execution Time | Matching Execution Time | Execution Time | Similarity Threshold | Pairs In Baseline | Pairs In Blocking | TP | FN | FP | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TwoYear | Jaccard | 0.73 | 0.03 | 0.13 | 0.16 | 0.5 | 2007 | 1813 | 1813 | 194 | 0 | 1.0 | 0.9033383158943697 | 0.9492146596858639 |
| commonAuthors | Jaccard | 0.73 | 0.8 | 0.0 | 0.8 | 0.5 | 2007 | 1701 | 1701 | 306 | 0 | 1.0 | 0.8475336322869955 | 0.9174757281553398 |
| Year | Jaccard | 0.73 | 0.03 | 0.07 | 0.1 | 0.5 | 2007 | 1749 | 1749 | 258 | 0 | 1.0 | 0.8714499252615845 | 0.9313099041533547 |
| LastLetterTitle | Jaccard | 0.73 | 0.07 | 0.15 | 0.22 | 0.5 | 2007 | 1876 | 1876 | 131 | 0 | 1.0 | 0.9347284504235177 | 0.9662631985578161 |
| numAuthors | Jaccard | 0.73 | 0.52 | 0.56 | 1.08 | 0.5 | 2007 | 1944 | 1944 | 63 | 0 | 1.0 | 0.968609865470852 | 0.9840546697038723 |
| FirstOrLastLetterTitle | Jaccard | 0.73 | 0.12 | 0.17 | 0.29 | 0.5 | 2007 | 1976 | 1976 | 31 | 0 | 1.0 | 0.9845540607872446 | 0.992216921918152 |
| FirstLetterTitle | Jaccard | 0.73 | 0.08 | 0.08 | 0.16 | 0.5 | 2007 | 1864 | 1864 | 143 | 0 | 1.0 | 0.9287493771798705 | 0.9630586411779902 |
| commonAndNumAuthors | Jaccard | 0.73 | 0.77 | 0.0 | 0.77 | 0.5 | 2007 | 1673 | 1673 | 334 | 0 | 1.0 | 0.833582461385152 | 0.9092391304347825 |
| authorLastName | Jaccard | 0.73 | 0.48 | 0.01 | 0.49 | 0.5 | 2007 | 1805 | 1805 | 202 | 0 | 1.0 | 0.8993522670652716 | 0.9470094438614901 |
| TwoYear | Jaccard | 0.71 | 0.05 | 0.12 | 0.16 | 0.7 | 1818 | 1715 | 1715 | 103 | 0 | 1.0 | 0.9433443344334433 | 0.9708463062553071 |
| commonAuthors | Jaccard | 0.71 | 0.63 | 0.0 | 0.63 | 0.7 | 1818 | 1610 | 1610 | 208 | 0 | 1.0 | 0.8855885588558856 | 0.9393232205367561 |
| Year | Jaccard | 0.71 | 0.05 | 0.06 | 0.11 | 0.7 | 1818 | 1673 | 1673 | 145 | 0 | 1.0 | 0.9202420242024203 | 0.9584646233171011 |
| LastLetterTitle | Jaccard | 0.71 | 0.08 | 0.14 | 0.21 | 0.7 | 1818 | 1752 | 1752 | 66 | 0 | 1.0 | 0.9636963696369637 | 0.9815126050420168 |
| numAuthors | Jaccard | 0.71 | 0.5 | 0.46 | 0.95 | 0.7 | 1818 | 1771 | 1771 | 47 | 0 | 1.0 | 0.9741474147414741 | 0.9869044302033992 |
| FirstOrLastLetterTitle | Jaccard | 0.71 | 0.12 | 0.17 | 0.29 | 0.7 | 1818 | 1811 | 1811 | 7 | 0 | 1.0 | 0.9961496149614961 | 0.9980710939652797 |
| FirstLetterTitle | Jaccard | 0.71 | 0.08 | 0.04 | 0.12 | 0.7 | 1818 | 1745 | 1745 | 73 | 0 | 1.0 | 0.9598459845984598 | 0.9795116474880718 |
| commonAndNumAuthors | Jaccard | 0.71 | 0.65 | 0.0 | 0.65 | 0.7 | 1818 | 1585 | 1585 | 233 | 0 | 1.0 | 0.8718371837183718 | 0.9315310020570086 |
| authorLastName | Jaccard | 0.71 | 0.45 | 0.01 | 0.45 | 0.7 | 1818 | 1704 | 1704 | 114 | 0 | 1.0 | 0.9372937293729373 | 0.9676320272572402 |
| TwoYear | Combined | 2.95 | 0.05 | 0.56 | 0.61 | 0.5 | 2024 | 1834 | 1834 | 190 | 0 | 1.0 | 0.9061264822134387 | 0.9507516848107828 |
| commonAuthors | Combined | 2.95 | 0.62 | 0.02 | 0.64 | 0.5 | 2024 | 1792 | 1792 | 232 | 0 | 1.0 | 0.8853754940711462 | 0.939203354297694 |
| Year | Combined | 2.95 | 0.05 | 0.31 | 0.36 | 0.5 | 2024 | 1764 | 1764 | 260 | 0 | 1.0 | 0.8715415019762845 | 0.9313621964097148 |
| LastLetterTitle | Combined | 2.95 | 0.08 | 0.65 | 0.73 | 0.5 | 2024 | 1881 | 1881 | 143 | 0 | 1.0 | 0.9293478260869565 | 0.9633802816901409 |
| numAuthors | Combined | 2.95 | 0.5 | 2.03 | 2.54 | 0.5 | 2024 | 1972 | 1972 | 52 | 0 | 1.0 | 0.974308300395257 | 0.9869869869869871 |
| FirstOrLastLetterTitle | Combined | 2.95 | 0.13 | 0.82 | 0.95 | 0.5 | 2024 | 1983 | 1983 | 41 | 0 | 1.0 | 0.9797430830039525 | 0.9897679061642125 |
| FirstLetterTitle | Combined | 2.95 | 0.08 | 0.19 | 0.27 | 0.5 | 2024 | 1859 | 1859 | 165 | 0 | 1.0 | 0.9184782608695652 | 0.9575070821529745 |
| commonAndNumAuthors | Combined | 2.95 | 0.64 | 0.01 | 0.65 | 0.5 | 2024 | 1764 | 1764 | 260 | 0 | 1.0 | 0.8715415019762845 | 0.9313621964097148 |
| authorLastName | Combined | 2.95 | 0.45 | 0.03 | 0.49 | 0.5 | 2024 | 1895 | 1895 | 129 | 0 | 1.0 | 0.9362648221343873 | 0.9670834396529727 |
| TwoYear | Combined | 3.0 | 0.05 | 0.57 | 0.62 | 0.7 | 1784 | 1693 | 1693 | 91 | 0 | 1.0 | 0.9489910313901345 | 0.9738280126545873 |
| commonAuthors | Combined | 3.0 | 0.69 | 0.02 | 0.71 | 0.7 | 1784 | 1639 | 1639 | 145 | 0 | 1.0 | 0.9187219730941704 | 0.9576394975167981 |
| Year | Combined | 3.0 | 0.05 | 0.31 | 0.36 | 0.7 | 1784 | 1669 | 1669 | 115 | 0 | 1.0 | 0.9355381165919282 | 0.9666956269910222 |
| LastLetterTitle | Combined | 3.0 | 0.08 | 0.67 | 0.75 | 0.7 | 1784 | 1720 | 1720 | 64 | 0 | 1.0 | 0.9641255605381166 | 0.9817351598173517 |
| numAuthors | Combined | 3.0 | 0.5 | 2.12 | 2.62 | 0.7 | 1784 | 1751 | 1751 | 33 | 0 | 1.0 | 0.9815022421524664 | 0.9906647807637907 |
| FirstOrLastLetterTitle | Combined | 3.0 | 0.12 | 0.83 | 0.95 | 0.7 | 1784 | 1778 | 1778 | 6 | 0 | 1.0 | 0.9966367713004485 | 0.9983155530600786 |
| FirstLetterTitle | Combined | 3.0 | 0.08 | 0.19 | 0.27 | 0.7 | 1784 | 1754 | 1754 | 30 | 0 | 1.0 | 0.9831838565022422 | 0.9915206331260599 |
| commonAndNumAuthors | Combined | 3.0 | 0.64 | 0.01 | 0.66 | 0.7 | 1784 | 1618 | 1618 | 166 | 0 | 1.0 | 0.9069506726457399 | 0.9512051734273956 |
| authorLastName | Combined | 3.0 | 0.46 | 0.04 | 0.49 | 0.7 | 1784 | 1730 | 1730 | 54 | 0 | 1.0 | 0.9697309417040358 | 0.9846328969834945 |

The best model is based on the combination of the **'First or Last Letter'** blocking and the **Combined** similarity function, for two main reasons:

1. The Combined similarity function has proven to yield more reliable results for matched entities upon close inspection of the data.
2. First or Last Letter Matching has seemed to outperform all the other methods in terms of Precision, Recall and F1 Score, and the execution time reduction is also very significant.

## 5.4. Clustering

In the final part of the pipeline, we choose to cluster the matched entities.

We employ a very basic clustering technique in which we group papers based on connected component edges exceeding a threshold value θ. These edges represent matched entities inferred from previous steps.

To be more precise, we utilize the Numpy package to construct a graph, wherein we arrange related items into clusters based on their similarity in the clustering process (referred to as clustering_basic). Each item is depicted as a node within the graph, while connections between items sharing similarities, identified during the matching phase, are depicted as edges. Subsequently, we employ depth-first search (DFS) to traverse these edges, updating values as we explore, thereby contributing to the organization of clusters in the final outcomes.

> The code for clustering is available in the file named `clustering.py` , and The resulting CSV file will be exported to a local directory called "results" with the chosen name (by default: "clustering_results_local.csv").

# 6. Data Parallel Entity Resolution Pipeline (Part 3)

At the beginning of this stage, we create an Entity Resolution pipeline using Apache Spark. We walk through all the phases of the Entity Resolution pipeline with the structured data frame. We employ a deployment model in our Pyspark environment utilizing a maximum number of local threads specified as local[*]. This deployment configuration enables parallel processing, leading to a substantial reduction in overall runtime. It also has a number of convenient built-in functions, for example, `df.filter` and `df.groupBy` help us with our blocking method.

In this Data Parallel framework, we mainly deploy one matching method (Combined Similarity), two blocking methods (FirstLetterTitle and FirstLetter Matching) and one clustering method (basic clustering with graph).

> You can see the code for this part at `dprep.py`

After using Spark's data frame, we wanted to compare it with our local pipeline (the one we constructed in part 2). The Two ER pipeline is configured as:

```
DEFAULT_ER_CONFIGURATION = {
    "threshold": 0.7,
    "matching_method": "Combined",
    "blocking_method": "FirstLetterTitle",
    "clustering_method": "basic",
    "output_filename": "clustering_results_local.csv",
}
```

The results are quite the same for all the methods we implemented.

| | **FirstLetterTitle** | **FirstOrLastLetterTitle** |
|---|---|---|
| Number of differences (dp and local) | 0 | 0 |
| Number of matched pairs | 1750 | 1778 |

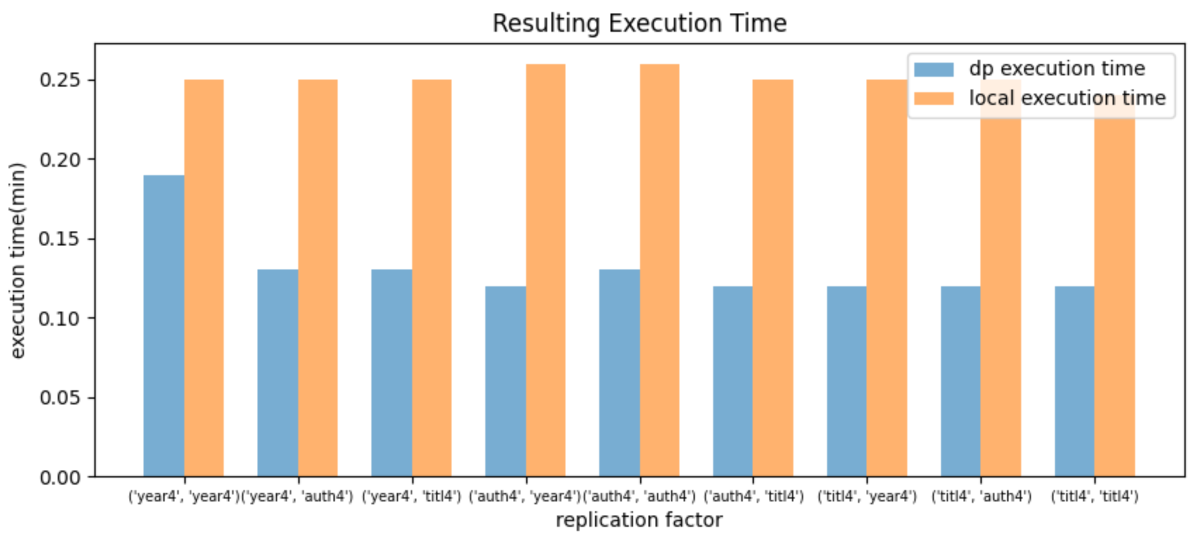
You can see the code for this part under the function `naive_DPvsLocal` in `main.py`

Given that we have established the reliability of Spark's pipeline, our objective is to evaluate the scalability performance of our pipelines. As a result, we have generated larger datasets with several modifications derived from our initial data.

To investigate the impact on our model, we introduced various alterations to the title, year, and author name. Specifically, for string inputs, we randomly selected n positions within the string and replaced a letter at each position with a randomly chosen alphabet. Moreover, for number inputs, we modified them by either incrementing or decrementing the value by n/2.

see the function `create_databaseWithChanges`

attached here are our scalability results:



x-asis: Replication factor (first four letter indicates which factor in the original database we choose to modify, and the last letter indicates the value of n), y-axis: Runtime in minutes