Advanced Topics in Data Engineering

***Entity Resolution Assignment***

***Supervisor: Mr. Giorgos Alexiou***

A picture containing dark, high

Description automatically generated

**Thursday August 31st, 2023**

**Dimitris Matsanganis, f2822212**

# Abstract

This assignment focuses on the intricate domain of entity resolution and duplicate detection, utilizing advanced methodologies to augment data precision and integrity. Through a structured progression, each task contributes to a comprehensive comprehension of the methodologies' nuances.

Task A: Token Blocking for Block Creation

Task A employs the Token Blocking approach, schema-agnostic in nature, to construct blocks represented as Key-Value (K-V) pairs. All attributes except the identifier (id) contribute to generating distinct Blocking Keys (BKs) from entity attribute values. The id column remains non-participatory in the block index creation process. Lowercasing string attributes during tokenization ensures precise alignment. A function is employed to visualize the BKs in a reader-friendly manner.

Task B: Calculating Possible Comparisons

In Task B, we compute the full spectrum of necessary comparisons to rectify duplicates within the established blocks from Task A. The quantified number of comparisons offers insights into the computational intricacies of the entity resolution process.

Task C: Meta-Blocking Graph with CBS Weighting Scheme

Task C introduces the Meta-Blocking graph, implementing the CBS (Common Block Scheme) Weighting Scheme. Edges with a weight below 2 are pruned, refining the block collection and eliminating unnecessary comparisons. The pruned collection serves as the foundation for recalculating the ultimate number of comparisons, paralleling Task B's methodology.

Task D: Jaccard Similarity Function

Task D encompasses a custom function for computing Jaccard similarity, focusing on the "title" attribute. While not involving practical comparisons, this function serves as a gauge for attribute similarity.

This analysis is executed in Python, specifically version 3.10, using Jupyter Notebook. The assignment amalgamates theoretical understanding with hands-on application, fostering mastery of advanced entity resolution techniques and bolstering data analysis proficiencies.

# Contents

[Abstract 2](#_Toc142767481)

[Contents 3](#_Toc142767482)

[Table of Figures 4](#_Toc142767483)

[Assignment Description 5](#_Toc142767484)

[Case Study Outline 6](#_Toc142767485)

[I. Introduction 8](#_Toc142767486)

[II. Data Preprocessing 9](#_Toc142767487)

[Loading Data 9](#_Toc142767488)

[Data Type Consistency 9](#_Toc142767489)

[Handling Missing Values 10](#_Toc142767490)

[Token Creation from Attributes 10](#_Toc142767491)

[Cleaning Tokens 10](#_Toc142767492)

[Concatenating Attributes 10](#_Toc142767493)

[Final Data Preparation 10](#_Toc142767494)

[Conclusion 10](#_Toc142767495)

[III. Task A: Token Blocking for Block Creation 11](#_Toc142767496)

[Creating Key-Value Pairs for Tokens 12](#_Toc142767497)

[Cleaning the Key-Value Pairs 12](#_Toc142767498)

[Displaying the Key-Value Pairs 12](#_Toc142767499)

[Addressing Limitations in Jupyter Notebook and Solution 13](#_Toc142767500)

[Conclusion 14](#_Toc142767501)

[IV. Task B: Calculating Possible Comparisons 14](#_Toc142767502)

[Methodology 15](#_Toc142767503)

[Results & Findings 16](#_Toc142767504)

[Conclusion 16](#_Toc142767505)

[V. Task C: Meta-Blocking Graph with CBS Weighting Scheme 16](#_Toc142767506)

[Initialization and Dictionary Construction 17](#_Toc142767507)

[Entity Counter and Loop Control 18](#_Toc142767508)

[Constructing Entity Pairs and Weights 18](#_Toc142767509)

[Weight Aggregation 18](#_Toc142767510)

[Why Aggregate Weights? 19](#_Toc142767511)

[Iterating Through Entity Pairs 19](#_Toc142767512)

[Summing the Weights 19](#_Toc142767513)

[Outcome 19](#_Toc142767514)

[Pruning Entity Pairs 20](#_Toc142767515)

[Why Prune? 20](#_Toc142767516)

[The Pruning Criterion 20](#_Toc142767517)

[Python Dictionary Comprehension 20](#_Toc142767518)

[Outcome 21](#_Toc142767519)

[Displaying Filtered Entity Pairs 21](#_Toc142767520)

[Calculating Total Comparisons After Pruning 22](#_Toc142767521)

[Why Calculate Total Comparisons? 22](#_Toc142767522)

[Summing the Weights 22](#_Toc142767523)

[Displaying the Result 22](#_Toc142767524)

[Interpreting Total Comparisons After Pruning 22](#_Toc142767525)

[Conclusion 23](#_Toc142767526)

[Preliminary Analysis 23](#_Toc142767527)

[VI. Task D: Jaccard Similarity Function 24](#_Toc142767528)

[VII. Conclusion 24](#_Toc142767529)

[VIII. Deliverables 25](#_Toc142767530)

[IX. Appendix 25](#_Toc142767531)

# Table of Figures

[Figure 1: Installation and Setup of Impala via Cloudera QuickStart VM. 5](#_Toc142416818)

[Figure 2: Create the Impala database & the required tables. 9](#_Toc142416819)

[Figure 3: Provided schema. 10](#_Toc142416820)

[Figure 4: Executing the Task B's command variable type error. 10](#_Toc142416821)

[Figure 5: Create the Impala database & the required tables (revisited version). 11](#_Toc142416822)

[Figure 6: Validation procedure of task 3a. 12](#_Toc142416823)

[Figure 7: Executed command successfully adds student record in Impala. 13](#_Toc142416824)

[Figure 8: Execution of Task 3c's query in Impala with Results. 15](#_Toc142416825)

[Figure 9: Execution of Task 3d's query in Impala with Results. 17](#_Toc142416826)

[Figure 10: Results of explain command for demonstrative purposes. 18](#_Toc142416827)

[Figure 11: A sample picture demonstrating the procedure followed for the Task 4. 20](#_Toc142416828)

# Assignment Description

**Task A [30 points]**

Use the Token Blocking (not to be confused with Standard Blocking) method as a schema-agnostic approach to create blocks in the form of K-V (Key-value) pairs. The key for every entry will be each distinct Blocking Key (BK) derived from the entities' attribute values, except for the id column which should ONLY be used as a reference for the blocking index and should NOT be included in the blocking process (block index creation). Please note that Token Blocking, being schema-agnostic, allows the utilization of all attributes from every entity (except id) for creating the blocks. To ensure accurate matching, it is advised to transform every string to lowercase during the tokens' creation before inserting them into the index. At the end of the creation, use a function to pretty-print the index, displaying the Key-Value pairs in a clear and readable format.

By employing the Token Blocking method in a schema-agnostic manner, you will create blocks that capture the essence of the entities' attributes. Each block's key will represent a distinct Blocking Key (BK) derived from these attribute values, excluding the id column. This approach allows for comprehensive analysis by utilizing all available attributes from each entity. Lowercasing the attribute strings during token creation helps avoid mismatches. Finally, ensure the generated index is presented neatly using a function to pretty-print the Key-Value pairs.

**Task B [25 points]**

Compute all the possible comparisons that shall be made to resolve the duplicates within the blocks that were created in Task A. After the computation, please print the final calculated number of comparisons.

**Task C [30 points]**

Create a Meta-Blocking graph of the block collection (created in Task A) and using the CBS Weighting Scheme (i.e., Number of common blocks that entities in a specific comparison have in common) i) prune (delete) the edges that have weight < 2 ii) re-calculate the final number of comparisons (like in step B) of the new block collection that will be created after the edge pruning.

**Task D [15 points]**

Create a function that takes as input two entities and computes their Jaccard similarity based on the attribute title. You are not requested to perform any actual comparisons using this function.

# Case Study Outline

In this case study, we embark on a comprehensive exploration of advanced techniques for entity resolution and duplicate detection. In today's data-driven world, the importance of ensuring data accuracy and quality cannot be overstated. Duplicates lurking within datasets can taint analytical outcomes and introduce erroneous insights. The advanced methodologies uncovered in this case study furnish a robust framework to surmount these challenges and elevate data precision to a new level.

The systematic journey unfolds across four distinct tasks, each contributing to the development of a holistic understanding of entity resolution methodologies. These tasks guide us through the intricate landscape of data cleansing and entity consolidation, armed with techniques that stand at the forefront of data refinement. From the art of creating data blocks through Token Blocking, and the quantification of comparisons to efficiently resolve duplicates, to crafting Meta-Blocking graphs and gauging attribute similarity, this case study equips us with the precise tools essential for navigating complex data scenarios.

At the heart of this case study lies the integration of theoretical knowledge with practical implementation. Python 3.10, a powerful and versatile programming language, collaborates seamlessly with the dynamic environment of Jupyter Notebook. Together, they form the vessel for executing and demonstrating the advanced techniques outlined in this study. The ensuing sections of this case study unravel the intricate threads woven into the fabric of entity resolution and duplicate detection, inviting readers to immerse themselves in the richness of these methodologies.

**Assignment Structure:**

**I. Introduction**

The foundation is laid with an exploration of data quality's paramount importance. Within the expansive realm of data management and analysis, the specter of duplicates threatens accuracy. This assignment's core objective revolves around the investigation of advanced methodologies that elevate data integrity through precise identification and resolution of duplicates.

**II. Data Preprocessing**

In the realm of accurate analysis, the importance of data preprocessing cannot be overstated. Our primary objective is to refine and transform the dataset, making it more amenable to token blocking and subsequent examination.

**III. Task A: Token Blocking for Block Creation**

The initial step introduces us to the Token Blocking method—an avant-garde schema-agnostic technique. By leveraging all attributes except the identifier (id), we construct Key-Value (K-V) pairs that encapsulate distinctive Blocking Keys (BKs). These BKs, derived from entity attribute values, become the building blocks for data consolidation. The id column's passive role in block index creation is emphasized, alongside the importance of lowercasing string attributes for precise tokenization. The result is a structured index, elegantly visualized using a dedicated function.

**IV. Task B: Calculating Possible Comparisons**

In Task B, we tackle the computational intricacies entailed in resolving duplicates. The task entails a comprehensive calculation of all requisite comparisons within the established blocks from Task A. By quantifying the magnitude of computations required, we gain insight into the computational complexity inherent in precise entity resolution.

**V. Task C: Constructing a Meta-Blocking Graph**

Task C introduces us to the concept of Meta-Blocking, a powerful strategy that leverages the CBS (Common Block Scheme) Weighting Scheme. Assembling a graph based on the block collection from Task A, we prune edges with weights below 2—a strategic move to streamline the block collection and curtail superfluous comparisons. The pruned collection forms the bedrock for recalculating the definitive number of comparisons.

**VI. Task D: Jaccard Similarity Function**

Task D unveils a custom-crafted Jaccard similarity function, tailor-made for attribute comparison based on the "title" attribute. Though devoid of practical comparisons, the function serves as an invaluable tool for quantifying attribute similarity—a linchpin in entity resolution.

**VII. Conclusion**

As the case study culminates, the accomplishments of each task and the methodologies' intricacies converge. The assignment concludes with a reflective summary, shedding light on the multifaceted contributions of executed techniques to bolstered data quality, encapsulating the essence of accurate entity resolution and duplicate detection.

**VIII. Deliverables**

The assignments and analysis deliverables will be mentioned in this section.

**IX. Appendix**

Delve further into the world of advanced techniques with these supplementary resources. Explore Token Blocking, Meta-Blocking, Jaccard similarity, Python, Jupyter Notebook, and recommended reading for a well-rounded understanding of entity resolution.

# I. Introduction

In the intricate realm of **data management and analysis**, the **cornerstone of success** lies in unwavering **data quality** and unwavering **accuracy**. Amidst this pursuit, an **omnipresent challenge** emerges in the form of **duplicates** residing within datasets. These duplicates possess the **uncanny ability** to distort insights and introduce misinterpretations, demanding the application of nothing short of **robust and advanced techniques** for effective **entity resolution** and meticulous **duplicate detection**.

Within the **purview** of this assignment, the **profound importance** of **data quality** takes center stage. Duplicates, when left unchecked, can inflict **havoc** upon analytical endeavors, casting a **shadow of doubt** upon the **credibility** of outcomes. To address this **critical concern**, this assignment embarks on a journey to unravel a suite of **advanced methodologies** meticulously crafted to not only detect but to **precisely identify** and rectify duplicates.

Our **voyage of exploration** commences with **Task A**, where the innovative **Token Blocking method** is introduced. This **schema-agnostic approach** ingeniously creates **blocks** through the harmonious pairing of **Key-Value (K-V) pairs**. These pairs, stemming from distinctive **Blocking Keys (BKs)**, encapsulate the very **essence** of entity attributes. However, the **id column** stands distinct, reserved solely for referencing within the **blocking index**, thus serving as a marker but not actively participating in the block creation process. An important note to consider is the transformation of **string attributes** to **lowercase** during token creation, ensuring meticulous accuracy and circumventing **mismatches**. The culmination of this process is a **visually appealing** and **intelligible Key-Value pair representation**, presented through a dedicated **function**.

Advancing forward, **Task B** ventures into the domain of **computational complexity**. Here, we undertake the meticulous **quantification** of all **conceivable comparisons**. This calculation shines a **spotlight** on the intricate web of comparisons required for the **precise resolution of duplicates** within the blocks that were methodically constructed in **Task A**.

As our exploration unfolds, **Task C** unveils the realm of **Meta-Blocking**, introducing a thoughtfully constructed **graph**. This graph is birthed from the **block collection** forged in **Task A**, and it harnesses the potency of the **CBS (Common Block Scheme) Weighting Scheme**. Pruning edges with weights below the threshold of **2** stands as a **strategic decision**, streamlining the block collection and minimizing the **quantum of superfluous comparisons**. This conscientious curation forms the **bedrock** for recalculating the ultimate **number of comparisons**.

**Task D**, the final stride, delves into the creation of a function that mirrors the **Jaccard similarity concept**. This custom function, designed to compute similarity based on the attribute “**title**,” emerges as an **indispensable tool** for gauging the **likeness of attributes**. Although this function does not perform **actual comparisons** within this assignment, it is poised to play a **pivotal role** in evaluating similarity across the **expanse of attributes**.

This assignment unfurls within the canvas of **Python 3.10** and the dynamic tapestry of **Jupyter Notebook**. United in purpose, these tools merge **theoretical understanding** with **hands-on application**. As we journey through these tasks, let’s embrace the **opportunity** to **hone our skills**, cultivate a **profound understanding** of **advanced techniques** for entity resolution, and contribute to the realm of **data integrity** and **accuracy**. Through this **comprehensive voyage**, we shall refine our **expertise**, bridging the gap between **theory and practice**, and ultimately elevate our capability to wield **advanced methodologies** for meticulous **entity resolution** and **duplicate detection**.

# II. Data Preprocessing

In this section we are going to mention all the preprocessing procedures needed prior to initialize our analysis. ***We need to point out that for a better understanding you can look up for our Jupyter notebook, where our analysis has taken place*** (*See* [*Deliverables*](#_VII._Deliverables)).

In other words, to ensure accurate analysis, we must first preprocess our data. This phase forms the bedrock of our subsequent analysis, ensuring that the dataset is in the right shape and format.

## Loading Data

To begin with, every analysis starts with data. For our case study, we'll be loading a dataset named ***ER-Data.csv***. This dataset is a treasure trove of columns filled with entity attributes, which will be the core of our analysis.

## Data Type Consistency

In the initial phase, we focus on *converting* all relevant columns to ***string******type***. When dealing with multiple sources, it’s common to encounter a mix of data types. **Ensuring** that all relevant columns have a consistent data type, specifically strings, is crucial. This uniformity guarantees that operations like concatenation proceed without any type-related errors, forming a *stable foundation* for the steps that follow.

## Handling Missing Values

Next, we address the ever-present challenge of *missing values*, often represented as NaN (Not a Number). These gaps, if overlooked, can disrupt the analysis, and leaving them untreated might lead to inconsistencies. By **replacing** these NaN values and the string ‘nan’ with ‘None’, we ensure uniformity across the dataset. This step underscores the importance of a clean dataset, which is vital for subsequent analysis.

## Token Creation from Attributes

Tokens, the atomic units of our dataset, form the core of our analysis. To create these tokens, we undertake the vital step of ***converting all attribute values to lowercase and then splitting them into individual words*** *(as mentioned in the assignment's instructions)*. These tokens subsequently serve as the building blocks for the **blocking keys**, which will be instrumental in the entity resolution process. The transformation into tokens ensures that we have distinct, manageable units that *encapsulate the essence* of the information contained within the attributes.

## Cleaning Tokens

In our pursuit of perfection, we noticed that some tokens may contain extraneous characters, such as commas, especially after the split operation. These characters, though seemingly insignificant, can hinder our analysis. By **iteratively processing** each token and removing such characters, we ensure that our tokens are clean and truly represent unique words or attribute values. Furthermore, we remove the words **none** and **nan** to not be biased on the current analysis. This cleanliness is pivotal for the effectiveness of the subsequent analytical methods.

## Concatenating Attributes

With our tokens immaculately prepared, the next logical step is to *combine* them into a unified list for each entity. By concatenating the tokens derived from authors, venue, year, and title, we create a comprehensive **fingerprint** (unique identifier **id**) for each entity. This *amalgamation* ensures that we encapsulate all available information about an entity, significantly enhancing the chances of accurate matches in the subsequent stages.

## Final Data Preparation

As we edge towards concluding our preprocessing, we undertake two pivotal steps. First, we **discard** the original attribute columns, as they become redundant post token creation. Subsequently, we *reconfigure* the dataframe’s index using the **id** column. These optimizations not only make our data leaner but also streamline its structure, making it more intuitive for subsequent analysis.

## Conclusion

The data preprocessing stage, often understated, is vital in any data analysis process. By meticulously following the aforementioned steps, we have transformed our dataset into an *optimal format*. This preprocessing lays the groundwork for the token blocking method and further analysis, ensuring that each entity in our dataset is represented by a comprehensive set of tokens derived from all its available attributes. This *structured and clean format* is the bedrock upon which the success of the subsequent stages of our analysis is built.

A sample figure of the first ten rows of our dataframe after the whole data preprocessing procedures follows below:

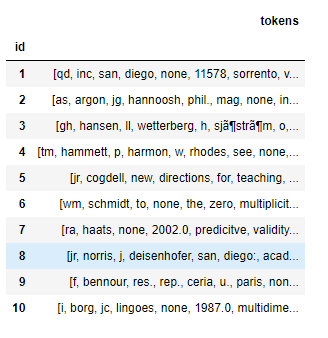


Figure 1: Snapshot of the First 10 Rows of the Final Preprocessed Dataset.

With the completion of the meticulous data preprocessing phase, our dataset is now in an optimal state for in-depth analysis. We shall commence our investigative journey with *Task A*, laying the foundation for subsequent analytical endeavors.

# III. Task A: Token Blocking for Block Creation

In accordance with the task’s directive within this assignment, we embark on the intriguing journey of **Task A** - a pursuit that dives deep into the realm of advanced entity resolution techniques. Employing the dataset presented in the **ER-Data.csv** file, this task harnesses the power of **Token Blocking** as a schema-agnostic methodology to create blocks, subsequently forming **Key-Value (K-V) pairs**.

The foundation of this task resides in the innate need for **data integrity** and the meticulous elimination of duplicates that permeate datasets. While the **Token Blocking** method shares the overarching objective of duplicate detection, it offers a distinct approach. Unlike standard blocking, Token Blocking isn’t confined to specific attributes or schemas, thus encompassing the holistic range of attributes from each entity (excluding the id column). This approach fuels a **comprehensive analysis**, drawing insights from all available attributes.

Within this methodology, the creation of blocks transpires through the ingenious fusion of **Key-Value (K-V) pairs**. Each entry’s key is ingeniously derived from a distinct **Blocking Key (BK)**, which in turn is an embodiment of the entity’s attribute values. Notably, the id column plays a unique role as a mere reference point for the blocking index, abstaining from active participation in the block creation process. A crucial nuance within Token Blocking involves the **transformation of string attributes to lowercase** during tokenization. This tactical maneuver significantly contributes to ensuring accurate matching and precludes potential mismatches that might arise from case discrepancies.

Upon the completion of this process, we culminate with an index that magnificently captures the essence of each entity’s attributes, showcased through the clarity of **Key-Value pairs**. These pairs, the embodiment of the schema-agnostic Token Blocking methodology, illuminate the unique attributes that each entity contributes to the data landscape. Through the thoughtful implementation of a dedicated function, the index is meticulously **pretty-printed**, allowing for clear and discernible observation of the Key-Value pairs.

As we embark on this Task A journey, we stand poised to unravel the intricacies of Token Blocking, where each Key-Value pair serves as a testament to our commitment to data accuracy and duplicate eradication. In parallel, we set the stage for the subsequent tasks, where these meticulously crafted blocks will play a pivotal role in the profound exploration of advanced techniques for entity resolution and duplicate detection.

*Answer of Task A:*

In this segment of our analysis, our primary focus lies in the Token Blocking method, an approach aimed at crafting blocks through Key-Value (K-V) pairs. The objective is clear: discern and record relationships between different entities based on shared tokens, facilitating efficient entity resolution.

## Creating Key-Value Pairs for Tokens

Our analysis begins with the design of a strategic dictionary. Here, each unique token metamorphoses into a key, with corresponding values being the IDs of entities that harbor these tokens: Significance of Multi-entity Keys.

More precisely, ***Significance of Multi-entity Keys*** is one of the foundational pillars of this step was ensuring keys linked to multiple entities. By setting a cut-off at a minimum of 2 entities per key, we underscored the importance of each key in the blocking mechanism.

## Cleaning the Key-Value Pairs

Following the dictionary’s inception, we pivot towards refining it ***Pruning Redundancies***. Certain tokens, while present, don’t contribute meaningfully to our analysis. Specifically, keys resonating with ‘nan’, ‘none’, and blank values were pruned to ensure the cleanliness and relevance of our constructed blocks.

## Displaying the Key-Value Pairs

With the dictionary structured and refined, our next step orbits around its presentation: ***Showcasing Interrelationships.*** We endeavored to present our Key-Value pairs in a manner that was both clear and insightful. Each token, juxtaposed with its associated entities, painted a vivid picture of interlinked relationships, all anchored by shared tokens.

A number on a white background

Description automatically generated

Figure 2: Displaying the Key-Value Pairs.

## Addressing Limitations in Jupyter Notebook and Solution

During our intricate analysis within the Jupyter Notebook environment, we stumbled upon an unexpected challenge. As the volume of data we were processing and visualizing grew, we confronted an ***error related to Jupyter Notebook's IOPub data rate limit***. The precise message was an "IOPub data rate exceeded" warning. This error acts as a safety net, designed to prevent Jupyter's server from overloading the client with vast outputs, which could potentially lead to crashes (*See Figure 3*).

The essence of this error is rooted in the default settings of Jupyter. Our expansive analysis, particularly during the display phases, was generating outputs that surpassed this default threshold. This disruption, while a minor setback, required swift troubleshooting to ensure the continuation of our workflow.

Our remedy was both immediate and effective. By invoking the command

***jupyter notebook --NotebookApp.iopub\_data\_rate\_limit=1e10***,

we were able to elevate the IOPub data rate limit considerably. This adjustment paved the way for our data-intensive tasks to operate seamlessly, eliminating the risk of triggering the aforementioned error and ***giving us the chance to output all the K-V pairs***. On top of that, we have included a figure titled "IOPub Data Rate Exceeded Error" immediately following this section, offering a snapshot of the error we grappled with.

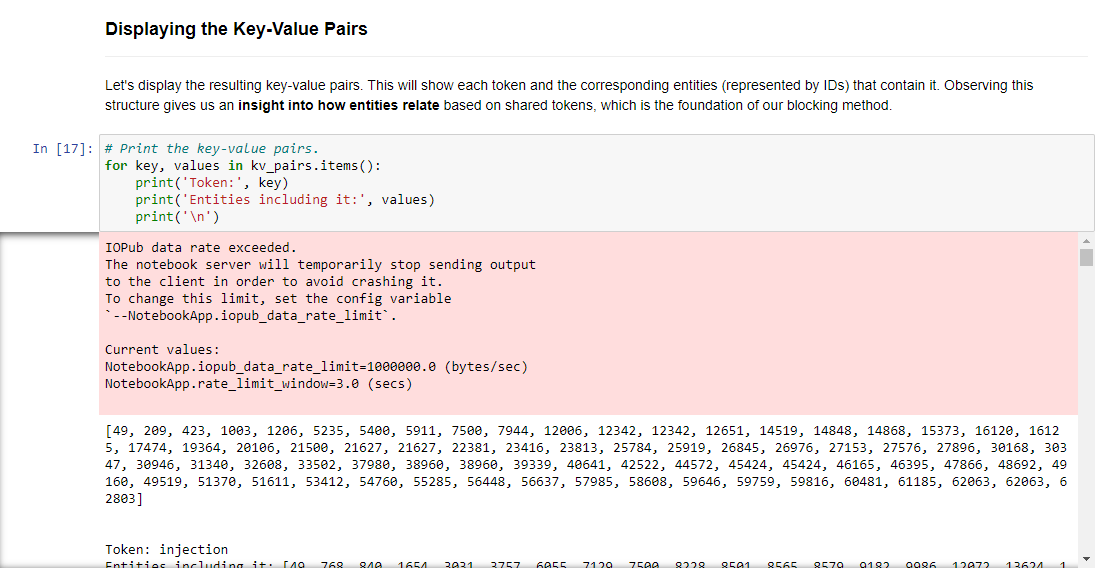


Figure 3: IOPub Data Rate Exceeded Error.

## Conclusion

The culmination of ***Task A*** signifies a blend of methodical planning intertwined with meticulous execution. The Token Blocking method, augmented by our systematic approach, ensures that we’re poised for success in the subsequent segments of our entity resolution process. Each stage, from token identification to visualization, has been engineered to elevate the accuracy and comprehensiveness of our blocks.

# IV. Task B: Calculating Possible Comparisons

In Task B, we embark on a crucial step of the entity resolution process – ***Calculating Possible Comparisons***. This phase is integral to understanding the computational complexity inherent in resolving duplicates within the blocks created in Task A. By systematically quantifying the number of comparisons required, we gain insights into the intricacies of duplicate detection and entity consolidation.

**Duplicates** within datasets can obscure meaningful insights and hinder accurate analysis. The process of **entity** **resolution** entails identifying these duplicates and consolidating them to represent a single real-world entity. This task unveils the mechanics behind evaluating possible comparisons and lays the groundwork for efficient and effective duplicate resolution.

Our methodology involves examining each block crafted in Task A and computing the total number of comparisons needed to identify and merge duplicates within that block. The cumulative count of these comparisons offers a tangible measure of the computational effort required for robust entity resolution. By unveiling the inherent challenges and intricacies, this task underscores the significance of employing advanced techniques for data accuracy.

Let's dive into the code implementation and calculations to shed light on the magnitude of possible comparisons and pave the way for optimal duplicate detection and entity consolidation.

*Answer of Task B:*

In the entity resolution process, understanding the computational complexity inherent in resolving duplicates within the blocks is crucial. **Task B** delves deep into this phase, emphasizing the significance of quantifying the number of comparisons required.

## Methodology

Our approach involves examining each block formed in *Task A* and tallying the total number of comparisons needed to identify and merge duplicates within that block. The combined count of these comparisons provides a tangible measure of the computational effort required for comprehensive entity resolution.

Here’s a step-by-step breakdown of the methodology:

1. **Initialize** an empty list named ***comparisons\_per\_block*** to hold the number of comparisons per block.
2. **Iterate** through each block created in *Task A*.
3. For each block, **calculate** the number of entities, denoted as ( n ).
4. **Compute** the number of comparisons for ( n ) entities using the formula: ( ).
5. **Append** the computed number of comparisons to the ***comparisons\_per\_block*** list.
6. Finally, **sum up** the list to obtain the total number of comparisons.

The code that fulfills the above is provided below:

1. # Initialize an empty list to hold the number of comparisons per block
2. comparisons\_per\_block = []
4. # Iterate through each block
5. **for** block\_entities **in** kv\_pairs.values():
6. # Calculate the number of entities within the block
7. num\_entities = len(block\_entities)
8. # Calculate the number of comparisons for n entities within the block
9. num\_comparisons = num\_entities \* (num\_entities - 1) / 2
10. # Append the number of comparisons to the list
11. comparisons\_per\_block.append(num\_comparisons)
13. # Sum up the list to obtain the total number of comparisons
14. total\_comparisons = sum(comparisons\_per\_block)

## Results & Findings

After executing the prescribed methodology, we derived the total number of comparisons required to robustly resolve duplicates within the blocks of *Task A*. This calculated metric is pivotal in assessing the computational complexity tied to the entity resolution process. Such insights are vital for formulating strategies that augment duplicate detection accuracy and facilitate streamlined entity consolidation.

The total number of comparisons amounted to an impressive **2,648,668,047**. This staggering figure accentuates the intricate nature of duplicate detection. It underscores the imperative need for sophisticated techniques to deftly traverse the data landscape, ensuring utmost data integrity.

## Conclusion

Entity resolution is a complex task that necessitates meticulous attention to detail. Task B’s emphasis on calculating possible comparisons offers a deep dive into the computational challenges involved in this process. *The revelation of over two billion comparisons (2,648,668,047) reiterates the sheer volume and intricacy of data we often grapple with*. Such insights fortify the rationale behind employing advanced techniques in entity resolution. By understanding the depth of the challenge, one can appreciate the importance of innovation and advanced methodologies in ensuring data accuracy and integrity in real-world applications. This task not only quantified the challenge but also shed light on the path forward, pushing the boundaries of what’s possible in the realm of entity resolution and duplicate detection.

# V. Task C: Meta-Blocking Graph with CBS Weighting Scheme

*In the intricate tapestry of data management,* **entity resolution** stands out as a cornerstone, ensuring the integrity and quality of our datasets. While **Task A** laid the foundation by introducing the Token Blocking technique, **Task C** beckons us into the realm of advanced optimization. Here, we unravel the transformative potential of **Meta-Blocking.**

**Meta-Blocking** is not just another technique; it's an evolution in our approach to entity resolution. By constructing a graph, where each node is a beacon representing an entity, we begin to see the intricate web of relationships within our data. The edges of this graph aren't mere connectors; they are bridges of significance. These edges indicate that the connected entities coexist within the same block, hinting at potential similarities.

But, how do we measure the strength of these relationships? Enter the **CBS (Common Block Schemes) Weighting Scheme.** This ingenious mechanism breathes life into our graph by allocating weights to its edges. The weight of an edge, symbolizing the number of common blocks shared by two entities, is a testament to their potential similarity. It's a quantifiable measure, a score, that whispers tales of how intertwined two entities might be.

Yet, in the vast expanse of data, not all relationships are created equal. Some are fleeting, while others are profound. In our quest for efficiency, we introduce a pivotal step: **Edge Pruning.** By setting a threshold, we decide to prune edges with a weight less than 2. This deliberate act isn't about exclusion; it's about focus. By pruning, we choose to hone in on the most meaningful of comparisons, ensuring that our attention remains undivided.

Finally, the essence of **Task C** culminates in **Computational Refinement.** After our meticulous pruning process, we recompute the number of comparisons for our newly refined block collection. This is not just a mathematical exercise; it's a reflection of our journey towards unparalleled efficiency in entity resolution.

In the subsequent sections, we shall navigate this intricate journey of **Task C**. With Meta-Blocking as our compass, we will explore, refine, and optimize, ever eager to enhance the precision of our entity resolution process.

*Answer of Task C:*

In this section, we delve into an advanced technique known as **Meta-Blocking**. Building upon the blocks generated in Task A, we aim to further optimize our blocking strategy by:

1. Constructing a graph where nodes represent entities and edges signify that these entities appear in the same block.
2. Leveraging the **CBS Weighting Scheme** to assign weights to the edges, with the weight indicating the number of common blocks shared by the two entities.
3. *Pruning* edges that have a weight of less than 2, ensuring a more streamlined comparison process.

Our overarching goal with this approach is to minimize both the number of duplicates and the volume of non-meaningful comparisons, thereby refining our blocking methodology. Let's embark on this task, by firstly initialize a dictionary.

## Initialization and Dictionary Construction

At the very onset of our Meta-Blocking journey, we lay the foundational groundwork by **initializing a dedicated dictionary**. This dictionary, aptly named **entity\_pair\_weights**, is conceptualized to be the linchpin holding together the relationships between entities and the weights they share. The weights, in essence, denote the number of common blocks that entities share, providing a quantifiable measure of their relationship.

## Entity Counter and Loop Control

The sheer volume of data mandates an **efficient** and **controlled** processing approach. To this end, we utilize an entity counter (**processed\_entities\_count**) to track processed entities, and a loop control mechanism (**loop\_control**) to halt processing upon reaching a predefined threshold. These tools work in tandem to ensure the Meta-Blocking process remains optimal and does not get overwhelmed by the data volume.

## Constructing Entity Pairs and Weights

The core of the Meta-Blocking process is to intelligently construct entity pairs and assign them appropriate weights. This step is pivotal as it shines a spotlight on potential duplicates within the dataset. The guiding principle is to traverse each block and pinpoint entities that share the same block. Their shared presence implies potential similarities. The significance of their relationship is then captured as a weight, derived from the *CBS (Common Blocks Scheme)*, which quantifies the blocks in which an entity pair co-exists.

The algorithm follows sequentially the series of following steps: **1. Loop through Blocks:** We initiate by navigating through each block. These blocks, clusters of entities, house potential duplicates and are the cornerstone of our Meta-Blocking process. **2. Process Each Entity in a Block:** Each entity is meticulously processed. To bolster efficiency, not all entities might be subjected to processing, a decision governed by the entity counter. Furthermore, singleton blocks, or blocks with a sole entity, are bypassed as they offer no potential for pairing. **3. Form Entity Pairs and Update Weights:** In this pivotal step, each entity within a block is juxtaposed with subsequent entities, forming pairs. These pairs, representative of potential duplicates, are then weighted based on their shared block occurrences. **4. Avoid Redundant Pairing:** Efficiency demands the avoidance of repetitiveness. Post pairing, entities are pruned from their respective blocks, ensuring they don't undergo redundant pairing in subsequent iterations. **5. Loop Termination:** The algorithm remains vigilant, monitoring the number of processed entities. Upon reaching a set threshold, it gracefully concludes its processing, emphasizing efficiency and precision.

## Weight Aggregation

Having identified pairs of entities that co-exist within common blocks, it becomes paramount to quantify the strength of their relationship. This strength, or weight, signifies the number of blocks in which each entity pair co-exists. The process of deriving this weight is known as **Weight Aggregation**, an integral step in the meta-blocking process. This weight, once calculated, serves as the very foundation of the meta-blocking graph, illuminating the connection between entities.

### Why Aggregate Weights?

During the meticulous construction of entity pairs, each occurrence of an entity pair within a block prompted the addition of a '1' to their associated list. This iterative addition resulted in lists that mirrored their frequency of co-existence. For instance, a list like **[1, 1, 1]** suggests that the corresponding entities co-existed across three distinct blocks. However, to derive meaningful insights and ensure optimal data representation, these lists need to be consolidated. Aggregating these weights paves the way for a clearer and more concise data representation, transforming lists such as **[1, 1, 1]** into a singular integer value, like 3, which stands as a testament to the entities' shared presence across three blocks.

### Iterating Through Entity Pairs

The crux of weight aggregation lies in iterating through each entity pair meticulously cataloged in the **entity\_pair\_weights** dictionary. Within this dictionary, each key symbolizes an entity pair, such as 'EntityA,EntityB', while the corresponding value captures their co-existence across blocks in the form of a list like **[1, 1, 1]**. This list represents the number of blocks in which the entities have been found together.

### Summing the Weights

The weight aggregation process culminates in the summation of these lists, encapsulating the frequency of co-existence of entity pairs into singular integer values. This transformation is executed by the code line: *entity\_pair\_weights[key] = sum(values)*. Here, each list, like **[1, 1, 1]**, undergoes summation, resulting in an integer, such as 3. This integer then replaces the original list within the **entity\_pair\_weights** dictionary, providing a succinct representation of the data.

### Outcome

Upon the successful culmination of the weight aggregation process, the **entity\_pair\_weights** dictionary undergoes a transformation. Each entity pair now boasts an associated integer weight, encapsulating the total number of blocks they share. This weight isn't just a number; it's the bedrock on which the meta-blocking graph is constructed, highlighting the relationship between entities. It is pivotal in determining the strength and significance of the relationship between entities in the ensuing meta-blocking graph.

Last but not least, for a comprehensive understanding, a sample representation of the transformed dictionary is provided below. This illustrative example offers a snapshot of the data post-aggregation, emphasizing the entity pairs and their corresponding weights (*see Figure 4*).

A close up of numbers

Description automatically generated

Figure 4: Sample Representation of Entity Pairs and Weights

## Pruning Entity Pairs

Building on the momentum from the previous steps, we now embark on a mission to refine our dataset by homing in on entity pairs that exhibit a strong likelihood of being genuine matches. This refinement is achieved through a process known as **pruning**, which involves filtering out entity pairs based on the weights they possess. Here, the weight symbolizes the number of blocks where a particular entity pair coexists. Intuitively, a larger weight suggests frequent co-appearances, pointing towards potential similarities or even duplicates among entities.

### Why Prune?

Pruning isn't just a process; it's a strategic optimization step. The rationale behind pruning is twofold. Firstly, by meticulously removing entity pairs that carry lower weights, we drastically slash the number of comparisons awaiting us in the subsequent stages. This not only infuses speed into the process but also fortifies its accuracy. Secondly, pruning minimizes false positives, reducing the risk of mistakenly flagging non-matching entities as potential matches, ensuring precision.

### The Pruning Criterion

Every pruning process requires a guiding light, a criterion that steers its decisions. In our endeavor, this guiding light is a weight threshold. The principle is straightforward: entity pairs burdened with a weight of 2 or less are deemed to have fragile associations and are thus excised. The logic, encapsulated in the code snippet *if value > 2*, suggests a discerning approach: if entities coalesce in just one or two blocks, their association might merely be serendipitous. Conversely, if their camaraderie spans three or more blocks, they are held in higher regard, earning a spot in the subsequent stages of the meta-blocking process.

### Python Dictionary Comprehension

The beauty of the Python language shines through during the pruning process. The task, which might seem daunting, is elegantly executed using Python's dictionary comprehension. This prowess allows us to effortlessly traverse the initial **entity\_pair\_weights** dictionary and chisel out a refined dictionary, aptly named **pruned\_entity\_pair\_weights**. The line of code, *pruned\_entity\_pair\_weights = {key: value for key, value in entity\_pair\_weights.items() if value > 2}*, is a testament to Python's efficiency. It ensures that only those pairs, whose weight surpasses 2, find a place in the pruned dictionary.

### Outcome

Upon the culmination of the pruning process, what emerges is a streamlined dictionary - **pruned\_entity\_pair\_weights**. This dictionary, a subset of the original, proudly showcases entity pairs that exude confidence in their association. Each pair, backed by a weight, indicates the number of blocks they harmoniously coexist in. As we venture ahead, these pairs, symbolizing strong associations, will be at the heart of our comparison or matching endeavors.

## Displaying Filtered Entity Pairs

With a pruned set of entity pairs in our arsenal, it becomes imperative to visualize a subset of this data. The goal? To glean insights into the refined data and ensure transparency in the process. The subsequent code segments are designed to showcase the first few entity pairs, precisely five in our case, that boast a weight denoting a shared presence across two or more blocks.

To be more precise the displaying filtered entity pairs procedure steps follows below:

**1. Initialization:** At the outset, a counter springs into action. Its sole purpose? To keep a vigilant eye on the number of displayed entity pairs, ensuring the output remains within manageable bounds, capped at five in our scenario.

**2. Iterate and Display:** With the counter in place, the next step embarks on a journey through the **pruned\_entity\_pair\_weights** dictionary. This dictionary is a repository of pruned entity pairs, each vouching for their strong associations. As we navigate through the dictionary, for each entity pair:

**a.** The respective entities (Nodes) and the count of their shared blocks (Number of Common Blocks) are displayed.

**b.** The counter gets incremented.

**c.** Post the display of five pairs, the loop graciously concludes its task.

For a holistic understanding, an illustrative snapshot of the displayed entity pairs is provided, showcasing the entities and their shared blocks count.

A screenshot of a computer

Description automatically generated

Figure 5: Illustrative Display of Filtered Entity Pairs

## Calculating Total Comparisons After Pruning

Emerging from the rigors of pruning, our next endeavor is to discern the impact of our efforts. The beacon guiding this quest? The total number of entity comparisons post-pruning. This metric not only offers a lens into the efficiency gains secured through the meta-blocking process but also charts the course for subsequent endeavors.

### Why Calculate Total Comparisons?

At the heart of the entity resolution process lies the essence of comparisons. The foundational blocking mechanism herded entities into distinct blocks, where each block echoed with potential matches. But without a refined lens, the sheer volume of comparisons can be daunting, a veritable challenge for any system. Herein lies the beauty of the meta-blocking technique, especially the act of pruning. It doesn't just reduce numbers; it curates them, ensuring only the most promising candidates make the cut. Determining the total number of comparisons post this refinement illuminates two crucial aspects:

1. The prowess and effectiveness of the meta-blocking strategy.  
2. The computational trajectory of the upcoming matching processes.

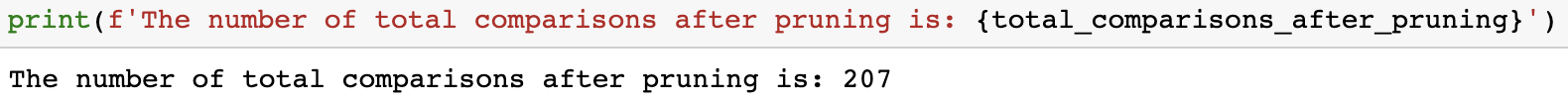
### Summing the Weights

Emerging from the rigors of pruning, our next endeavor is to discern the impact of our efforts. The beacon guiding this quest? The total number of entity comparisons post-pruning. This metric not only offers a lens into the efficiency gains secured through the meta-blocking process but also charts the course for subsequent endeavors.

Aggregating these weights, a task made seamless by Python's *sum()* function, equips us with the total comparisons required in the aftermath of pruning.

### Displaying the Result

Transparency is key. Thus, post the aggregation, the resultant number is presented, offering a crystal-clear perspective on the computational efficiency achieved:***207****.*



*Figure 6: Displaying the result after pruning.*

## Interpreting Total Comparisons After Pruning

Numbers, while quantitative, often tell a qualitative tale. The figure '207', emerging from the shadows of the meta-blocking process, carries with it a narrative of refinement. It signifies that after the meticulous pruning, hinged on the CBS (Common Blocks Scheme) weighting scheme, a total of 207 entity-to-entity comparisons beckon us.

## Conclusion

Task C wasn't just a task; it was a journey, a voyage through the intricate corridors of entity resolution. Commencing with the solid foundation of Task A and propelled by the revelations of Task B, Task C soared into the realms of advanced techniques like meta-blocking and pruning.

The initial phase of Task B, with its revelation of 2.6 billion comparisons, was akin to a spotlight, highlighting the vastness and intricacies of our data landscape. Such a mammoth figure underscored the challenges that lurked in the entity resolution domain. It also amplified the clarion call for further refinement.

Answering this call was Task C, armed with the Meta-Blocking graph. The graph, in its essence, was a tapestry, weaving together entities based on their shared blocks. The CBS weighting scheme added another dimension, infusing the tapestry with depth by attributing weights to each thread, each connection.

But what truly set Task C apart was the art of pruning. Through discerning choices, edges with weights less than two were gracefully excised, reducing computational demands and accentuating potential matches. This pruning phase was a testament to efficiency, a strategy to sharpen focus and enhance accuracy in the subsequent phases of entity resolution.

Re-evaluating the number of comparisons post this pruning, juxtaposed against the initial 2.6 billion, offered a tangible metric of our accomplishments. But beyond numbers, it painted a narrative of effectiveness, of the meta-blocking strategy's transformative potential.

In its entirety, Task C was both a challenge and a revelation. While it highlighted the intricacies of handling vast datasets, it also showcased the potential of innovative techniques in navigating this vastness. By integrating advanced methodologies, Task C not only ensured computational efficiency but also championed the cause of data accuracy and integrity.

## Preliminary Analysis

In our preliminary implementation, we closely examined a subset of **100 entities**. Our objective was to determine how many pairs from the entire set of **66,879 entities**, as highlighted in Task A, were present within our initial dataset. Our findings revealed that the number of the **final comparisons** retained amounted to the limited amount of **207**.   
  
However, it's crucial to highlight that our solution was based on a **significantly limited sample size**. Consequently, the real number of final comparisons might differ when analyzed on the complete dataset. Regardless, what stands out from our initial assessment is the **substantial reduction** in potential comparisons that could lead to actual matches.

# VI. Task D: Jaccard Similarity Function

# VII. Conclusion

# VIII. Deliverables

To ensure a thorough understanding of the process and the methodologies employed in our entity resolution tasks, the following deliverables are provided:

1. **Documentation Report**: This document provides an in-depth description of the assignments tasks, elucidating the methodology, findings, and implications. It serves as a comprehensive guide for readers to understand the significance of the entity resolution process.
2. **Analysis Jupyter Notebook**: Alongside this documentation, we are providing a Jupyter Notebook that contains the actual code implementation and analysis. The notebook, titled "**advanced\_techniques\_for\_entity\_resolution\_and\_duplicate\_detection.ipynb**", is developed using *Python 3.10*. It offers a hands-on view of the techniques described in this documentation, allowing readers to delve into the actual coding and analytical practices employed during the process.

These deliverables are designed to offer a holistic view of the entity resolution tasks, combining both theoretical insights from the documentation and practical application from the Jupyter Notebook. By leveraging both, readers can gain a profound grasp of the intricate subject matter at hand. ***Therefore, please have a look on both deliverables files.***