TABLE OF CONTENTS

1. INTRODUCTION **1**

2. DATA ACQUISITION **2**

2.1 Scopus 2

2.2 Data Augmentation 3

3. DATA PREPROCESSING AND RESTRUCTURING **5**

3.1 Trimming Columns 5

3.2 Author Segregation and Dictionary Creation 5

4. QUERIES AND RESULTS **7**

5. NEO4J **15**

5.1 Neo4j Desktop 15

5.2 Neo4j Browser 16

5.3 Neo4j Browser 17

5.4 Queries 18

5.5 Visualization 19

6. LINKS AND REFERENCES **21**

1. INTRODUCTION

This report undertakes a meticulous analysis of author publication data gleaned from a comprehensive dataset sourced from Scopus, a renowned repository of scholarly literature. Our dataset encapsulates a myriad of publication attributes, including author details, publication titles, publication years, citation counts, funding particulars, and other pertinent metadata. Our central aim in this analysis is to unravel intricate publication patterns, unveil collaborative networks among authors, and decipher the profound impact of their scholarly contributions.

In our pursuit of discerning scholarly output trends, we harness cutting-edge tools and methodologies, particularly emphasizing the utilization of advanced data visualization techniques and the implementation of modern data management systems. Leveraging state-of-the-art NoSQL databases, such as Neo4j, we adeptly navigate the complexities of semi- and unstructured data, facilitating swift data processing and enabling seamless handling of interconnected publication data. The adoption of graph database concepts not only empowers us with faster query capabilities but also enhances our capacity for visualizing intricate relationships within the scholarly landscape.

Amidst the vast expanse of academic inquiry, understanding the intricate dynamics of authorship and publication is paramount. Through our rigorous analysis, we endeavor to illuminate emerging research domains, discern patterns in funding allocation strategies, and unearth the symbiotic relationships that underpin collaborative scholarly endeavors. By delving deep into the wealth of data at our disposal and harnessing the power of innovative visualization tools, we aim to transcend conventional boundaries in scholarly analysis, offering insights that resonate across diverse academic disciplines. As we navigate through this report, we embark on a journey to unravel the multifaceted tapestry of academic publishing, poised at the intersection of technology, data analytics, and scholarly inquiry.

2. DATA ACQUISITION

Data acquisition typically refers to the process of importing or collecting data from external sources into a database. This process is essential for populating the database with relevant data so that it can be queried and analyzed effectively.

Data acquisition can be approached through various methodologies, each tailored to suit specific requirements dictated by factors such as data volume, complexity, update frequency, and available resources and expertise. In this context, the selection of an appropriate data acquisition method becomes crucial to ensuring efficient and effective handling of data. For our purposes, we have opted to utilize Scopus, an open-source and crowd-managed database renowned for its comprehensive coverage of scholarly literature and robust data management capabilities.

2.1 Scopus

Scopus is a database of peer-reviewed literature, including books, scientific journals, and conference proceedings. It covers research from the fields of science, technology, medicine, social science, and arts and humanities. Scopus is owned and maintained by Elsevier, a publisher of scientific, technical, and medical content.

Key features of Scopus include:

1. Abstracts and Citations: It provides abstracts and citations for academic articles, allowing researchers to quickly evaluate the relevance and impact of a particular publication.
2. Coverage: It covers a vast array of disciplines, including science, technology, medicine, social sciences, and arts and humanities.
3. Author Profiles: Authors can create profiles in Scopus to showcase their research output, track citations, and collaborate with other researchers.
4. Citation Analysis: It also offers tools for citation analysis, allowing researchers to identify influential papers, track citation trends, and measure the impact of their own research.
5. Search and Discovery: Researchers can use Scopus to search for academic literature using keywords, authors, affiliations, and other criteria. Advanced search options enable precise retrieval of relevant documents.
6. Integration with Other Tools: Scopus integrates with various research tools and platforms, making it easier for researchers to access and analyze scholarly literature.

In our case, we need to use specific filters to get the right data for our analysis. These filters are like tools that help us refine the information we get from the website. We use them to select exactly the kind of data we need.

The filters we use cover different aspects like when the publications were made, where they come from, what subjects they cover, how many times they've been cited, and who the authors are linked to. Each filter helps us narrow down the data to fit our research needs.

By using these filters, we make sure we're getting the most relevant information for our study. This helps us analyze the data more effectively and draw meaningful conclusions. The filters employed are as follows:

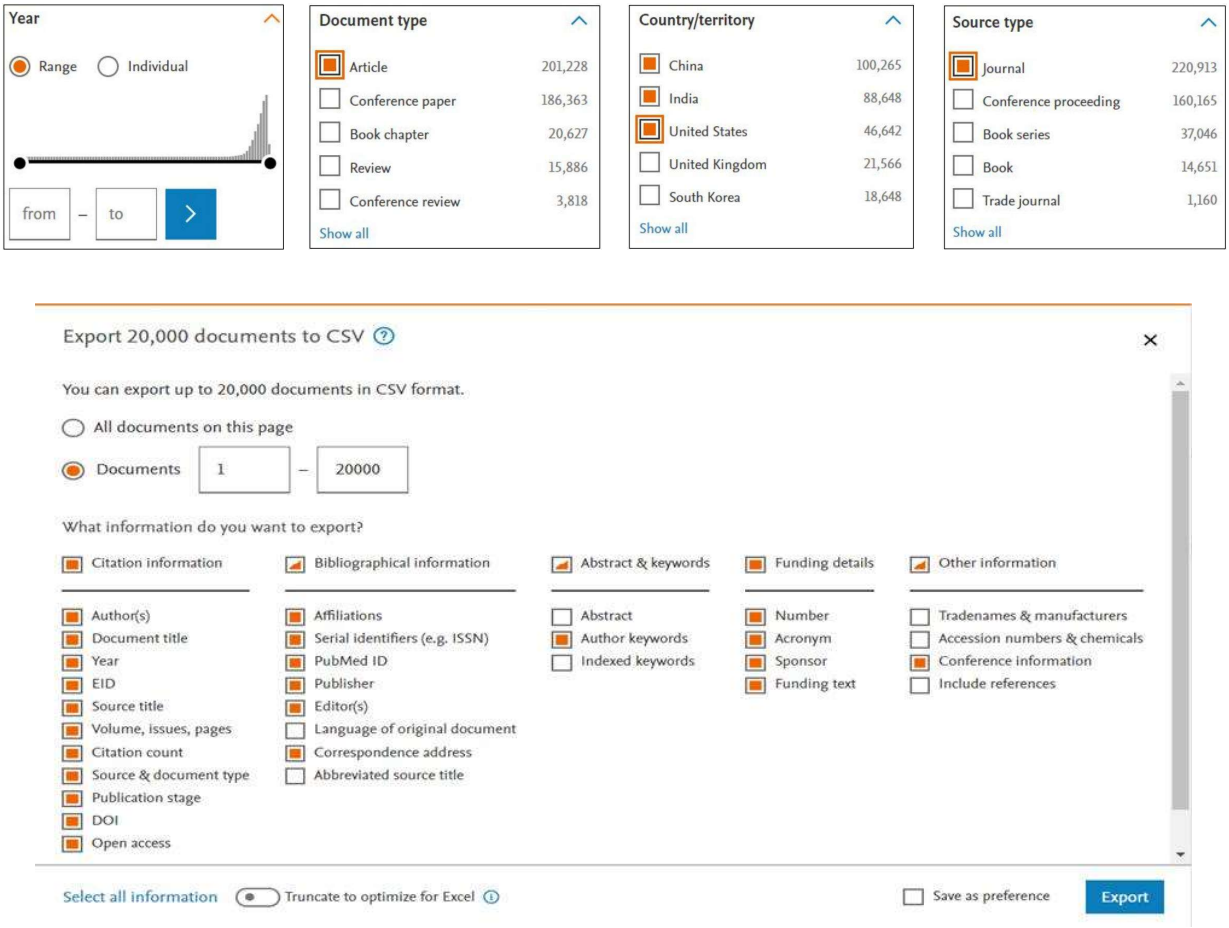


Fig 2.1 List of all the filters applied while acquiring the data

2.2 Necessity of Data Augmentation

While Scopus offers a range of features and filters, it lacks the capability to provide individual country data for each paper, as illustrated in Figure 2.2. This poses a challenge for our analysis since

we require country-specific information. However, since we also lack complete author data, we must resort to a process known as data augmentation.

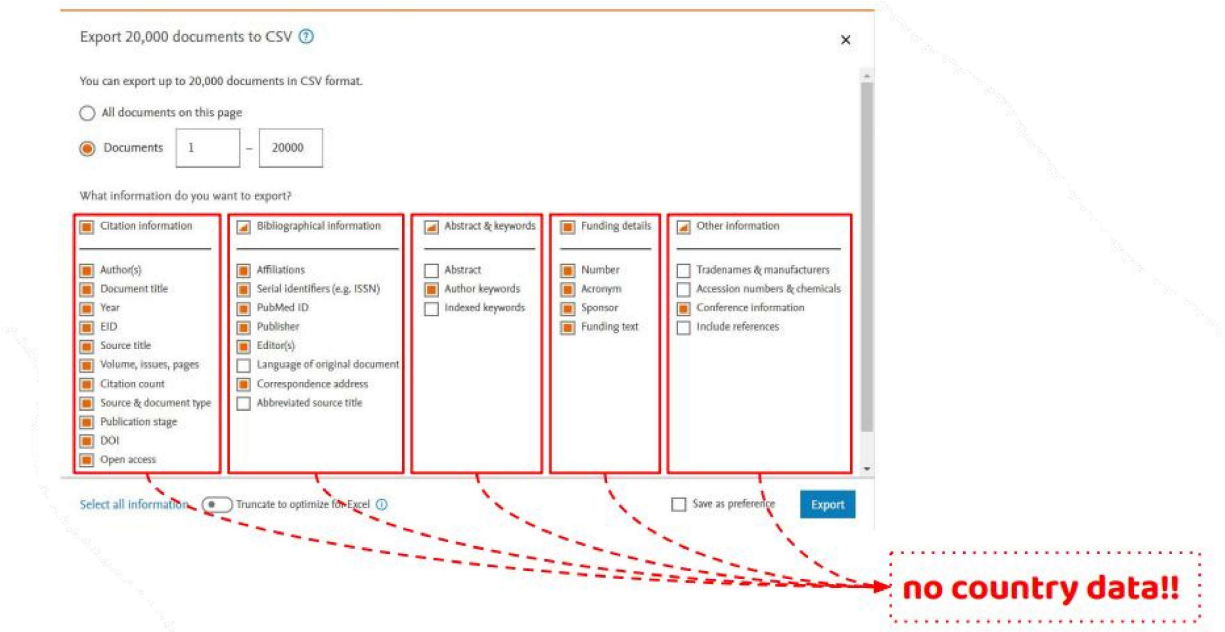


Fig 2.2 Exporting of country data is unavailable in Scopus

Data augmentation, in this context, involves manual intervention wherein we utilize country filters one by one, export the resulting datasets, and append a new column labeled "country" to the .csv file. We then populate this column with the respective country name used during filtering. This process is repeated for all required countries, such as India, USA, and China. Finally, we horizontally merge these datasets, ensuring that the order of merging is inconsequential. This meticulous process enables us to enrich our dataset with essential country-specific information necessary for our analysis.

3. DATA PREPROCESSING AND RESTRUCTURING

3.1 Trimming Columns

Irrelevant columns were dropped from the dataset, keeping only the essential attributes for analysis.By focusing on the columns directly related to the problem we're trying to solve, we can remove unnecessary data and significantly improve the speed at which we process the information. There were total 30 columns present in the dataset from which we only selected specific columns features are ‘Author full name’, ‘Titles’, ‘Year’, ‘Source Title’, ‘Cited by’, ‘Funding details’, ‘Publishers’, ‘Language of Original Document’, ‘Country’.

Fig 3.1 Dropping of columns that are unnecessary to the task at hand

3.2 Author Segregation and Dictionary Creation

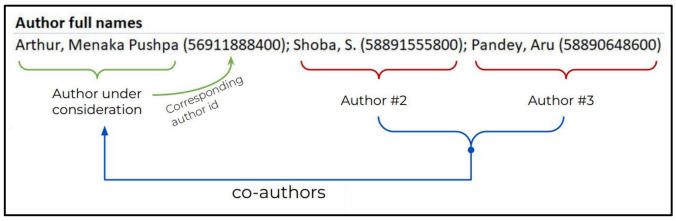
Authors for each publication were segregated, considering multiple authors for a single paper.An author dictionary was created to store detailed information about each author, including their publications, citations, publication years, funding details, co-authors, and other relevant information.We have split the multiple co authors and then extracted their unique ID put it in the dictionary. Then we build Author information adding paper titles (‘Paper_Names’), citation count (‘Citation’), publishing year (‘Year of publishing’), funding details (‘Funding’), the nationality of author (‘Country’) in the dictionary and lastly the co-author (‘Co-authors’) list. (See fig. 3.2, to understand the procedure to obtain the co-author list).

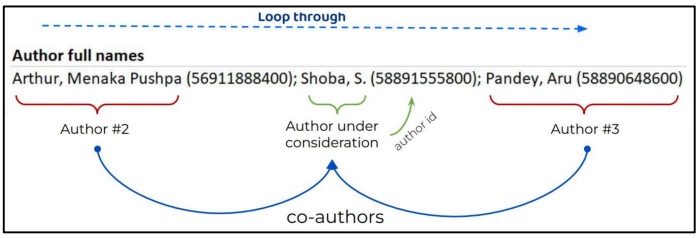
For a schematic diagram to reference see fig. 3.3.

Author full names
Arthur, Menaka Pushpa (56911888400); Shoba, S. (58891555800); Pandey, Aru (58890648600)

(a) Example of how the column 'Author full names' looks like. We have a list of co-authors along with their author id stored in the form of a semicolon separated list.

(b) Th co-author data needs to be traversed separately to create the list of co-authors that have published a paper together.





(c) We do the same for the second author, but the corresponding co-authors change.

Fig 3.2 (a),(b) and (c) showing the procedure to create the co-authors list in the author dictionary

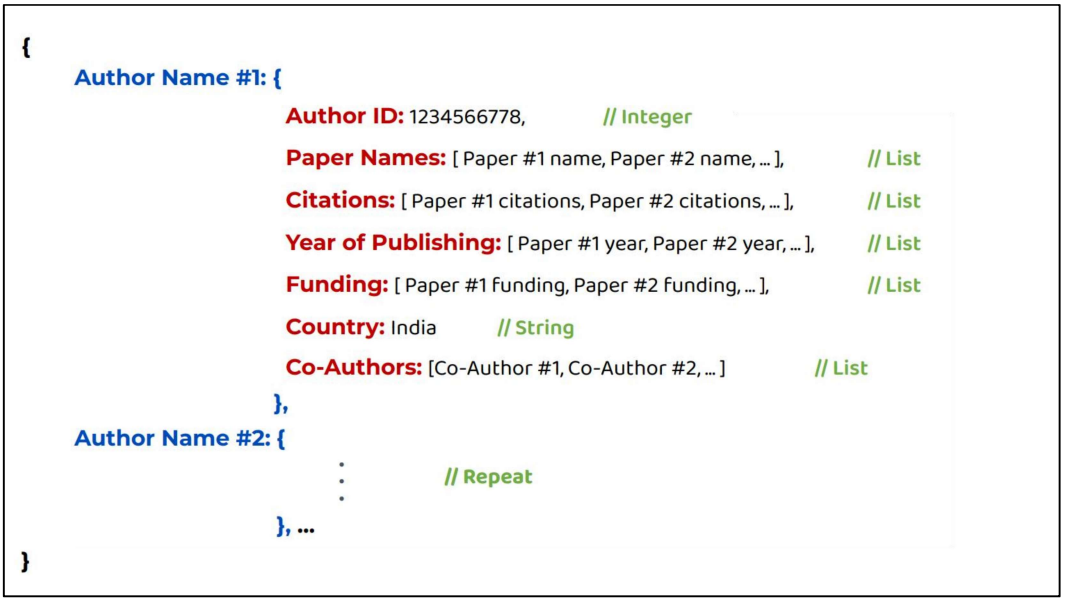


Fig 3.3 Schematic diagram of the author dictionary hence formed.

4. QUERIES AND RESULTS

(a) Highest cited author and his h-index

The analysis focuses on identifying authors with the highest citation counts and computing additional metrics such as the H-index to assess their impact within the academic community.

The code snippet utilizes the 'pprint' module to explore the structure of the author dictionary 'AD', providing insights into the organization and contents of the data.

Authors with the maximum total citations are identified by iterating through the author dictionary 'AD'.For each maximum cited author, details such as name, author ID, total citations, and H-index are stored in a DataFrame 'Qa'. The DataFrame 'Qa' presents the details of the author(s) with the highest total citations, including their names, author IDs, total citations, and H-index values.

The analysis provides valuable insights into the citation impact of authors within the dataset. By identifying authors with the highest citation counts and computing the H-index, the analysis highlights key contributors and their influence within the academic domain. These insights can aid in assessing the significance of authors' research contributions and identifying potential collaboration opportunities.

Output:

	Author Name	Author ID	Total Citations	H-index
0	Xu, Li Da	13408889400	13603.0	45

(b) Highest Publication author

The analysis aims to identify authors with the highest number of publications and store their details in a DataFrame.

The code iterates through each author in the author dictionary 'AD'. For each author, the total number of publications is computed by counting the number of unique paper names associated with the author.

Authors with the maximum number of publications are identified, considering the possibility of multiple authors with the same highest publication count. For each maximum publication author,

details such as author name, author ID, and total publications are stored in a temporary list.The temporary lists are appended to a list of lists (dfb) to accumulate details for all maximum publication authors.The list of lists is then converted into a DataFrame (Qb) with appropriate column names.

The analysis provides insights into the publication productivity of authors within the dataset. By identifying authors with the highest number of publications, researchers and analysts can gain an understanding of prolific contributors and their research output. These insights are valuable for assessing authors' productivity, research interests, and potential collaboration opportunities.

Output:

	Author Name	Author ID	Total Publications
0	Choo, Kim-Kwang Raymond	57208540261	243

(c) Highest cited author's avg citations and country name

The objective is to identify authors with the highest citation counts, compute additional metrics such as average citations per publication, and analyze the geographical distribution of these authors.

Authors with the maximum citation counts are identified based on the previously computed 'max_author_list' list. For each maximum citation author, details such as author name, author ID, total citations, average citations per publication, and country are stored in a temporary list (dfc).The temporary lists are appended to accumulate details for all maximum citation authors.The list of lists is then converted into a DataFrame (Qc) with appropriate column names.

The analysis provides insights into the citation impact and publication productivity of authors within the dataset. By computing metrics such as average citations per publication and analyzing the geographical distribution of authors, researchers can gain a comprehensive understanding of the impact and reach of authors' research contributions. These insights are valuable for assessing authors' influence within the academic community and identifying potential collaborations across different regions.

Output:

	Author Name	Author ID	Total Citations	Average Citations	Country
0	Xu, Li Da	13408889400	53	256.660377	India

(d) Total number of publications of the highest cited authors

The objective is to identify authors with the highest citation counts and analyze their publication productivity.

Authors with the maximum citation counts are identified based on the previously computed 'max_cite_author' list. or each maximum citation author, details such as author name, author ID, and total citations are stored in a temporary list ('df').The temporary lists are appended to accumulate details for all maximum citation authors.The list of lists is then converted into a DataFrame (Qd) with appropriate column names.

The analysis provides insights into the citation impact and publication productivity of authors within the dataset. By computing metrics such as average citations per publication and analyzing the geographical distribution of authors, researchers can gain a comprehensive understanding of the impact and reach of authors' research contributions. These insights are valuable for assessing authors' influence within the academic community and identifying potential collaborations across different regions.

Output:

	Author Name	Author ID	Total Publications
0	Xu, Li Da	13408889400	13603.0

(e) Total publication of year

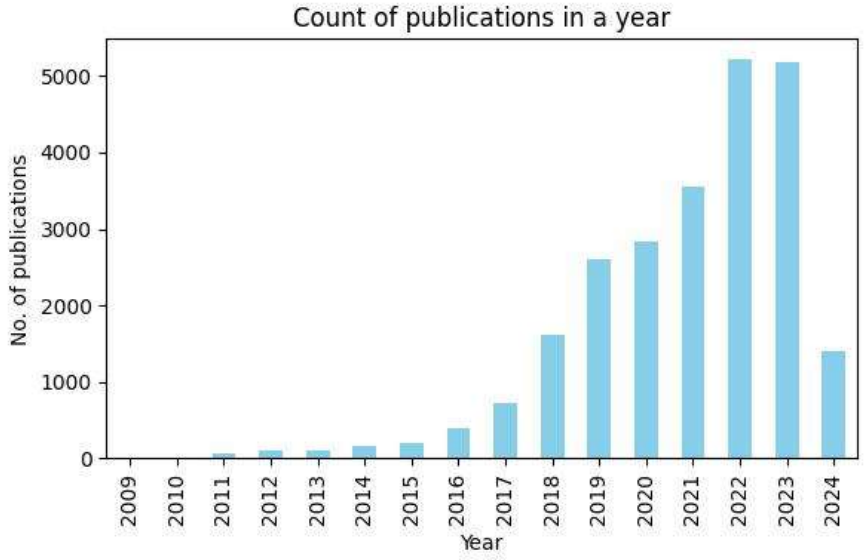
The objective is to visualize the distribution of publications across different years and identify any notable trends or patterns.

The DataFrame 'trimmed_df' is grouped by the 'Year' column, and the count of publications in each year is calculated. A comparison plot is created using matplotlib to visualize the distribution of publications across different years. The x-axis represents the years, while the y-axis represents the

number of publications. The bar plot showcases the count of publications in each year, allowing for easy comparison and identification of trends.

The analysis of publication trends by year offers valuable insights into the evolution of research within the dataset. Understanding the temporal patterns of publication activity can aid researchers in identifying emerging research areas, tracking the impact of significant events or developments, and informing future research directions.

Output:



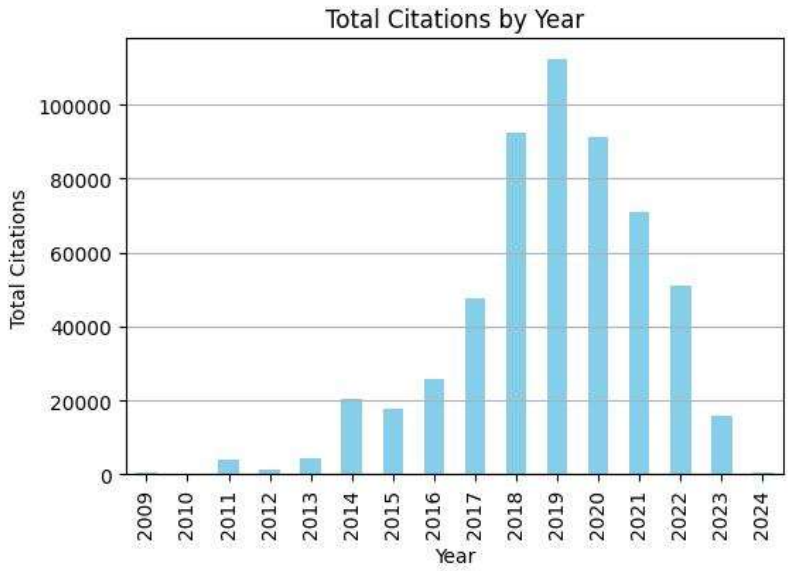
(f) Total citation in a year

The objective is to visualize the distribution of publications across different years and identify any notable trends or patterns.

The DataFrame trimmed_df is grouped by the 'Year' column, and the count of publications in each year is calculated. A comparison plot is created using matplotlib to visualize the distribution of publications across different years.The x-axis represents the years, while the y-axis represents the number of publications.The bar plot showcases the count of publications in each year, allowing for easy comparison and identification of trends.

The analysis of publication trends by year offers valuable insights into the evolution of research within the dataset. Understanding the temporal patterns of publication activity can aid researchers in identifying emerging research areas, tracking the impact of significant events or developments, and informing future research directions.

Output:



(g) Author(country) having highest co-authorship with Indian authors

Addressing this challenge proves formidable as Scopus lacks the functionality to furnish individual author-specific data separately. This crucial data is typically available only through the authors' profiles, necessitating manual extraction by accessing each author's page—a daunting task given the extensive roster of over 10,000 authors.

To overcome this hurdle, employing web scraping emerges as a viable solution. Web scraping involves automating the process of extracting data from web pages, enabling us to systematically retrieve the requisite information from each author's profile. By leveraging web scraping techniques, we can efficiently gather the necessary data without the need for manual intervention, thereby streamlining the data acquisition process and ensuring comprehensive coverage of author-specific information.

This automated approach not only saves time and effort but also ensures the integrity and completeness of the dataset. By harnessing the power of web scraping, we can effectively bridge the gap between Scopus' limitations and our data requirements, facilitating a more thorough and robust analysis of author-specific data.

(h) Highest cited author from India and the university

The objective is to analyze the citation impact of Indian authors and present their details in a DataFrame.

The code iterates through each author in the author dictionary (AD). For authors from India ('Country' == 'India'), the total number of citations is calculated using the sum() function. Authors with the maximum number of citations from India are identified.For each maximum citation author from India, details such as author name, author ID, country, and total citations are stored in a temporary list (dfh).The temporary lists are appended to accumulate details for all maximum citation authors from India.The list of lists is then converted into a DataFrame (Qh) with appropriate column names.

The analysis provides insights into the citation impact of authors from India within the dataset. By identifying authors with the maximum number of citations, researchers can assess the influence of Indian authors' research contributions and their impact on the scholarly community.

Output:

	Author Name	Author ID	Country	Total Citations
0	Xu, Li Da	13408889400	India	13603.0

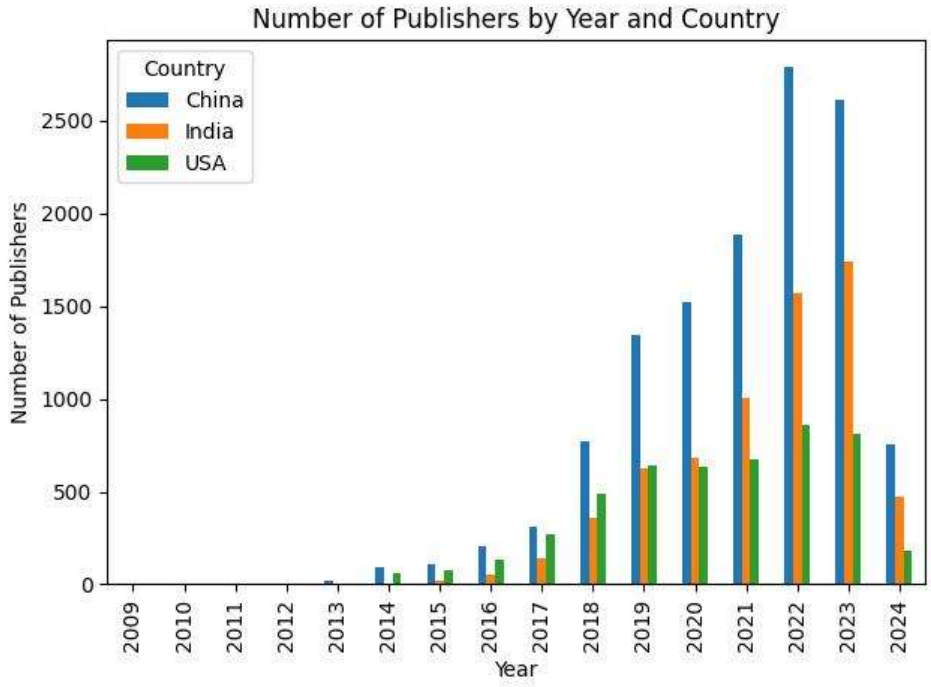
(i) Comparative year wise article publication analysis of India, China and USA

The objective is to visualize the number of publishers over time for each country and identify any notable trends or patterns.

The DataFrame trimmed_df is grouped by both the 'Year' and 'Country' columns, and the count of publishers in each group is calculated. The index of the grouped DataFrame is reset to make 'Year' and 'Country' columns again, ensuring they are accessible for further processing. The DataFrame is pivoted to rearrange the data, making each country a separate column and 'Year' as the index. A bar plot is created to visualize the number of publishers by year and country. Each country is represented by a separate colored bar, allowing for easy comparison of publisher counts over time. The legend provides clarity on the country represented by each color.

The analysis provides valuable insights into the distribution of publishers within the dataset, highlighting variations in publishing activity across different countries and over time. Understanding these patterns can help researchers identify global trends in scholarly publishing, track changes in publishing practices, and inform decisions related to research dissemination and collaboration.

Output:



(j) Total number of grants given to the field

This report aims to provide information on the total number of grants related to the field of Internet of Things (IoT) based on the provided datasets.

The 'Funding Details' column from the DataFrame trimmed_df is selected and non-null values are retained. The count of non-null values represents the total number of grants available for analysis.

The analysis reveals the total number of grants available for research or projects related to the field of Internet of Things (IoT). This information can be valuable for researchers, funding agencies, and policymakers interested in understanding the funding landscape and investment opportunities in IoT-related initiatives.

Output:

	Field Name	Total number of grants
1	Internet Of Things (IOT)	12214

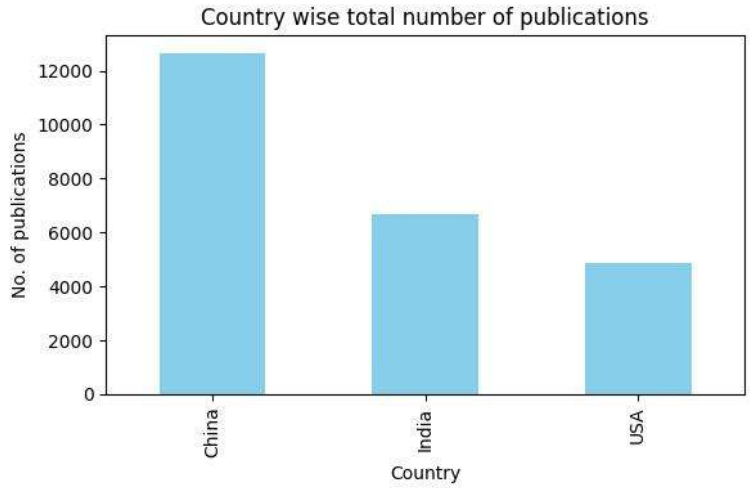
(k) Country wise total number of publication

The objective is to visualize and compare the publication output across different countries.

The DataFrame trimmed_df is grouped by the 'Country' column, and the count of publications in each country is calculated.A bar plot is created using matplotlib to visualize the total number of publications per country.The x-axis represents the countries, while the y-axis represents the number of publications.Each country is represented by a separate colored bar, allowing for easy comparison of publication output across different countries.

The analysis provides valuable insights into the publication output of different countries within the dataset. Understanding the geographical distribution of publications can help researchers identify global research trends, assess the impact of research funding and policies, and foster international collaboration in research endeavors.

Output:



5. NEO4J

Neo4j is a popular graph database management system. Unlike traditional relational databases, which store data in tables, Neo4j is designed to represent and store data in the form of graphs. In a graph database, data is represented as nodes, which are connected by edges. This graph structure is particularly useful for modeling and querying relationships between data points.

Key features of Neo4j include:

1. Native Graph Storage: Neo4j stores data in a graph format natively, allowing for efficient storage and retrieval of interconnected data.
2. Cypher Query Language: Cypher is a declarative query language specifically designed for graph databases like Neo4j. It allows users to express graph patterns and relationships in a simple and intuitive way.
3. Scalability: Neo4j is designed to scale both vertically (by adding more resources to a single machine) and horizontally (by distributing data across multiple machines).
4. ACID Compliance: Neo4j ensures data consistency and reliability through ACID (Atomicity, Consistency, Isolation, Durability) compliance.
5. High Performance: Due to its native graph storage and optimized traversal algorithms, Neo4j can efficiently handle complex queries involving deep relationships between nodes.
6. Graph Analytics: Neo4j provides built-in graph algorithms for analyzing and extracting insights from the data, such as community detection, centrality measures, and path finding.
7. Flexibility: It supports various deployment options, including on-premises, cloud, and hybrid deployments, as well as integration with popular programming languages and frameworks.

Neo4j is widely used in various domains such as social networking, recommendation systems, fraud detection, network and IT operations, and knowledge graph applications. Its flexibility, performance, and ease of use make it a popular choice for developers and organizations dealing with connected data.

5.1 Neo4j Desktop

Neo4j Desktop is a powerful tool designed to streamline the development, management, and visualization of Neo4j graph databases. It serves as a centralized platform for Neo4j users to efficiently work with their graph databases, providing a range of features and functionalities tailored to meet diverse needs.

At its core, Neo4j Desktop offers a user-friendly interface that allows users to seamlessly create, configure, and manage Neo4j database instances. Users can easily set up new projects, import data, and perform administrative tasks with intuitive controls and guided workflows.

One of the key features of Neo4j Desktop is its support for local and remote database connections. Users can work with both local databases running on their machine and remote databases hosted on servers or cloud platforms. This flexibility enables users to develop and test applications locally before deploying them to production environments.

In addition to database management, Neo4j Desktop provides a suite of tools for data visualization and exploration. Users can leverage built-in graph visualization capabilities to interactively explore their graph data, analyze relationships, and gain insights through visually compelling representations. Furthermore, Neo4j Desktop supports the integration of third-party visualization tools, such as Bloom, allowing users to leverage their preferred tools for advanced graph visualization and analysis.

Moreover, Neo4j Desktop fosters collaboration and knowledge sharing within teams by facilitating project sharing and version control. Users can easily share their projects with colleagues, collaborate on database development, and track changes using built-in version control features.

5.2 Neo4j Browser

The Neo4j Browser is a web-based graphical user interface (GUI) provided by Neo4j for interacting with Neo4j databases. It allows users to visualize and query the graph database using Cypher, Neo4j's query language, and provides various tools for exploring and analyzing graph data.

Key features of the Neo4j Browser include:

1. Interactive Visualization: The Browser provides an interactive graph visualization interface that allows users to visually explore the graph database. Nodes and relationships are represented as visual elements, and users can navigate through the graph by clicking and dragging.
2. Cypher Query Editor: Users can write and execute Cypher queries directly within the Browser. Cypher is a declarative query language specifically designed for graph databases, and it allows users to express graph patterns and relationships in a concise and intuitive way.

3. Query Results Visualization: The Browser displays query results in a tabular format, making it easy for users to view and analyze the data returned by their queries. Results can also be visualized graphically within the graph visualization interface.
4. Built-in Help and Documentation: The Browser provides built-in documentation and help resources for Cypher syntax and Neo4j features, making it easier for users to learn and use the database.
5. Saved Queries and Favorites: Users can save frequently used queries and mark them as favorites for easy access in the future.
6. Export and Import: The Browser supports exporting query results to various formats (e.g., CSV, JSON) and importing data into the database from external sources.
7. User Management: Administrators can manage user access and permissions within the Browser, controlling who can access the database and perform various actions..

5.3 Neo4j Bloom

Neo4j Bloom is a visualization and exploration tool provided by Neo4j for graph databases. It offers an intuitive and interactive interface for users to visually explore and analyze the data stored in Neo4j databases. Here are some key features and functionalities of Neo4j Bloom:

1. Graph Visualization: Neo4j Bloom provides an immersive and visually appealing grapsh visualization interface. Users can explore the graph by navigating through nodes and relationships using zoom, pan, and drag gestures.

2. Natural Language Search: Users can perform searches using natural language queries, allowing them to easily find relevant information within the graph database without needing to write complex queries.

3. Graph Exploration: Neo4j Bloom enables users to explore the graph by visually expanding or collapsing nodes and relationships, focusing on specific parts of the graph, and filtering data based on various criteria.

4. Contextual Insights: Bloom provides contextual insights and recommendations based on the data in the graph, helping users discover hidden patterns, connections, and insights within the graph.

5. Customizable Views: Users can customize the appearance of the graph visualization by adjusting node and relationship styles, colors, and labels, making it easier to interpret and analyze the data.

6. Collaborative Workspace: Neo4j Bloom supports collaborative workspace, allowing multiple users to collaborate on exploring and analyzing the graph data together in real-time.

7. Integration with Neo4j Desktop: Bloom seamlessly integrates with Neo4j Desktop, Neo4j's desktop application for managing and developing graph databases, providing a unified environment for working with Neo4j databases.

8. Security and Access Control: Bloom provides security features such as access control, authentication, and authorization, ensuring that users can only access the data they are authorized to view and modify.

5.4 Queries

We use the Neo4j browser to execute queries and generate nodes and corresponding edges in the graph database. However, the data that needs to be input must be properly structured. It is unnecessary to include all details of the authors in a specific author node; only the author's name is required. Therefore, inputting the dataframe directly to Cypher or the author dictionary to Cypher would be slow due to redundant data. Consequently, we opt to create a separate .CSV file. This dataframe comprises only two columns: the first column is 'Author 1', and the second is 'Author 2'. Each row contains data for two authors only if they are co-authors. Thus, we are solely storing the edges in the dataframe for better clarity.

Another issue arises: as we repeat this process for all authors, duplicate entries occur. For instance, if (A - B) exists, then (B - A) will also exist. To address this efficiently, we sort these tuples and place them in a set to ensure only unique edges. By applying this method to all authors in the author dictionary, we can generate a complete .CSV file for the Neo4j browser containing only unique edges.

Queries to run in Neo4j browser

We have multiple methods for running the code, each with its own pitfalls. To expedite query execution, we leverage the capabilities of *auto transactions*, *batch processing*, and *index creation*. Consequently, we execute the following queries in the browser sequentially.

```
neo4j$ CREATE INDEX FOR (a:Author) ON (a.name);
```

```
//Create graph with undirected edges
:auto
LOAD CSV WITH HEADERS FROM 'file:///database_for_neo4j.csv' as row
CALL{
    WITH row
    MERGE (a1: Author {name: row.`Author 1`})
    MERGE (a2: Author {name: row.`Author 2`})
    MERGE (a1)-[:WROTE_A_PAPER_WITH]->(a2)
    MERGE (a2)-[:WROTE_A_PAPER_WITH]->(a1)
} IN TRANSACTIONS of 500 ROWS
```

The dataset we're handling is substantial, encompassing over 200,000 authors, making the task of creating nodes for each individual quite challenging. In our process, we employ two query types: CREATE and MERGE. While both functions aim to create a node, there's a notable difference between them. CREATE, although faster, does not check for duplicates in the existing database, potentially leading to duplicates. In contrast, MERGE only creates a new node if it doesn't already exist in the database, making it the preferred choice. However, with around 200,000 nodes, the process of checking for preexistence consumes considerable time, particularly given that we perform this check for each new author.

To address this challenge, we implement an INDEX for each author. This allows the MERGE function to utilize these unique indices to search for preexistence, resulting in a significant reduction in execution time, nearly by 70%. Additionally, while we can establish only one bidirectional relationship between a1 and a2, creating two separate unidirectional edges enables us to visualize all the co-authors of a particular author simultaneously, once selected in Neo4j Bloom.

Another optimization we've implemented is auto transactions. Given the size of our dataset, we've introduced checkpoints in our node creation process. Specifically, we process data in batches of 500 and commit the transaction. This ensures that if we encounter any connectivity issues or power outages, the previously created nodes are already saved in the database. Consequently, we can rerun the same code, and the MERGE function will identify duplicates and skip creating additional replicas.

5.5 Visualization

To visualize, we can either use the Neo4j Browser itself, or we can use Neo4j Bloom.

The problem with Neo4j browser is that it only shows a maximum of 300 nodes at a time (see fig.

5.1), along their edges (relations). This leads to us having a very small subset of our data as we have data ranging in 2 lakhs.

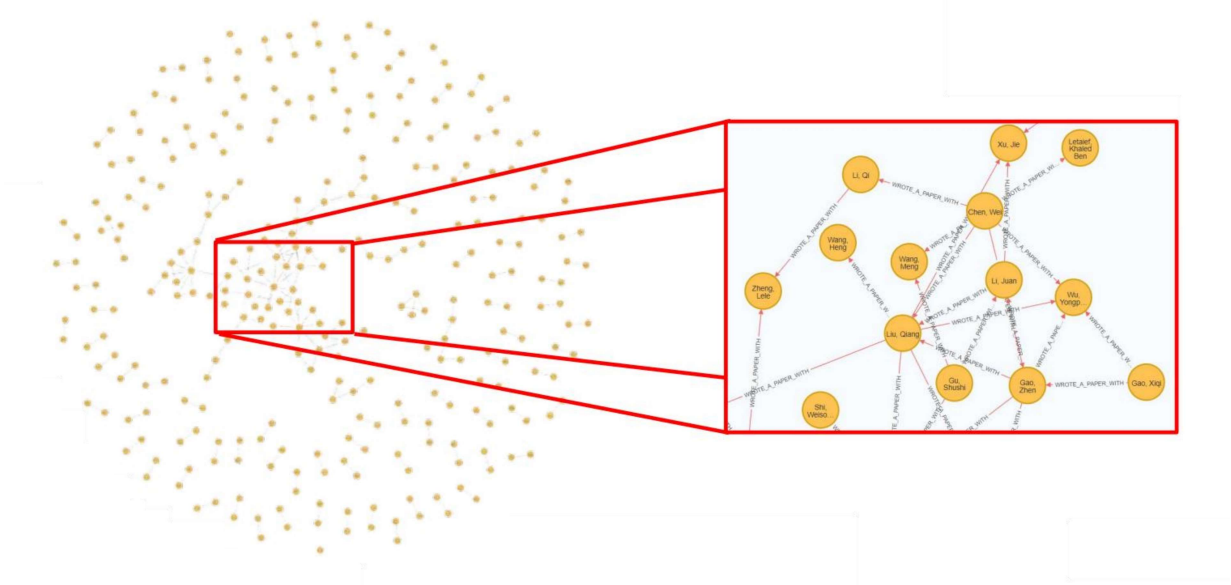


Fig 5.1 Graph visualization is facilitated through the Neo4j Browser, with a maximum display limit set to 300 nodes at a time. To enhance clarity, sections of the graph are zoomed in, enabling a clearer view of the edges.

Although we cannot expect to visualize more than 2 lakh nodes in any visualization tool, Neo4j Bloom help us visualize a considerable 10,000 nodes at a time, with dynamic visual effects that replicate physics based mechanisms. These graphs are also interactive and modifiable, giving us better control of our data. (see fig. 5.2)

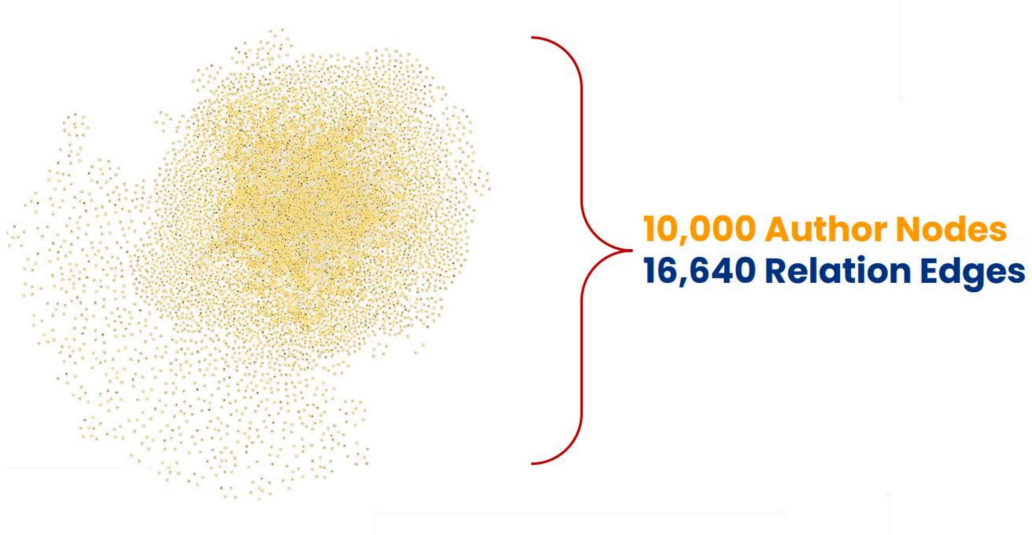


Fig 5.2 Graph visualization in Neo4j Bloom

6. LINKS AND REFERENCES

Links:

Code: https://colab.research.google.com/drive/1jpTdcR-1_bVYdrdxtbRa50_8SvvH8Qb_?usp=sharing

Neo4j Desktop: https://neo4j.com/download/

Scopus: https://www.scopus.com/search/form.uri?display=basic&zone=header&origin=searchbasic#basic

References:

[1] Neo4j. (n.d.). Introduction to Cypher. Retrieved from https://neo4j.com/docs/cypher-manual/current/introduction/

[2] Neo4j. (n.d.). Bloom User Guide. Retrieved from https://neo4j.com/docs/bloom-user-guide/current/