Challenges of Creating Linkage between Researchers, Institutions, and Data for Qualifying Research Effectiveness.

An Honors Thesis

Presented to the Thesis's Honors Panel, including:

Dr. Clément Aubert of the School of Computer and Cyber Sciences

Dr. Andrew Balas of the Department of Interdisciplinary Health Sciences

Dr. Faith Wiley of the Department of Biology

Augusta University

In fulfillment of the requirements for the

Augusta University Honors Program

By

Micheal Noah Sleeper

Augusta, Georgia

September 20th, 2022

Acknowledgements

We extend our thanks to Dr. Clément Aubert for agreeing to be the advisor for this thesis, as well as to Drs. Andrew Balas and Faith Wiley for agreeing to be members of the Honor Panel for this thesis. In addition, we also thank Dr. Clément Aubert for his dedication, support, and mentorship throughout the last year and a half. We are grateful for his help. Without which, this thesis could not have been completed.

Table of Contents

Introduction

This thesis project began as a question of whether or not it would be possible to measure the quality of a researcher or institution based on data on that entity in a way that is not tedious or overbearing, like doing this by hand would be. These institutions are educational/research institutions, such as universities. The data wanted for entities would be data that would help put together a qualitative metric about the entity. The number of patents put out in a certain time frame, grant amounts given, number of educational/research institutions worked at, and other data like this would be what is wanted. The idea of having a computer automate a process that would achieve this was headed by Drs. Andrew Balas and Clément Aubert, the former of which approached the latter with the project. The general idea of the process involved with this project is to have a program that automatically pulls together data on researchers and institutions from multiple different online sources, link people and institutions within the sources to the data attributed to them, and assign a quality metric to them with as little human direction as possible.

The purpose of this thesis is to create that program. The main "gist" of this program is to link information to people, thereby creating "entities" that roughly translate to a person, or an institution. From there, the program would attach a quality metric to each entity, in order to identify the most effective entities for a particular attribute, such as grant amounts or patents held. In total, the goal is to measure the quality of a researching entity with little-to-no direct human guidance.

Definition of Concepts and Tools

Throughout this project, we used a variety of tools and concepts tightly associated with computer science. This section will explain in simplified detail the general idea of the listed concepts and tools.

Concept: Linkage & Differentiation

Linkage, in Computer Science, is the ability to identify the entities, and attribute data to that entity. This is how we “define a person” within the program. If someone has a grant amount attributed to them, then, in the scope of this project, that person is defined by that data and only that data. In this project, we use linkage to tell who is who, but also who is not who. “How do we tell who to attribute to this data” might seem like a straightforward question, but this becomes difficult when the author’s name matches multiple individuals. How do we tell them apart?

Differentiation works off this concept well. In a similar vein to the question of how we define a person, “how do we tell two similar entities apart?” is a problem we have to deal with. Just as our program must be directed to recognize individuals by linking information that comprises that person, it must also be able to accurately differentiate entities. For example, is John D. from paper A the same John D. from paper B? Further, are either of these two tied to the JohnD@email.com address? Often, differentiation and linkage are two problems that cause confusion. More information on this problem of mismatch can be read below, in the “Problems: Mismatch” section.

Tool: SQL

SQL is a database management language. It works off schemas, tables, relations, and attributes. Data is put into tables, which fall under a certain schema, and is put into attributes that all reference a specific entity — whether that be a name, their paper, or a grant proposal. In SQL, data has to be inserted into already created tables in specific places. Tables that expect numbers cannot have names thrown in them — that generates an error. Therefore, there must be a predefined structure in SQL to properly handle the data.

In this project, SQL is the language used to create and maintain the tables where the downloaded databases will have their information compiled into and linked. With SQL, users pass queries about entities and receive the requested data from the tables.

Tool: Java

Java is the programming language that we use to connect SQL, the downloaded databases, and Excel. It also goes out and retrieves the specified databases, as well as maintaining credentials and various tasks, like file paths.

Java was chosen for multiple reasons. Two large reasons were that Java is a general purpose language, meaning it can be run on all computers. Anyone can run this program, as java is easy to install. The second reason is that Java is good for programs that go through large amounts of data, such as this program. Speed/performance would only be a worry insofar as how efficient our code is, rather than how well can the compiler handle the volume of data passed.

Since Java scales well with large data sizes, the large databases downloaded will not have too big an effect on runtime.

Tool: Excel

Microsoft Excel is a spreadsheet software. For the purpose of this program, Excel acts as a visualizer for the wanted data retrieved from SQL. Excel is likely the more well known of the main three tools this project uses. Something of note about Excel, which will be explored later in the “Methods: In-between connectors” and “Problems: Coding” sections is that Microsoft does not like outside “untrustworthy” (eg: non-Microsoft approved) applications/software connecting to any Microsoft application, such as Excel. Java does not have a streamlined way of connecting to these trusted applications, so connecting to Excel gets tricky. We use Apache POI (defined below) to help create a connection between Java and Excel.

Tool: Dependencies and Maven

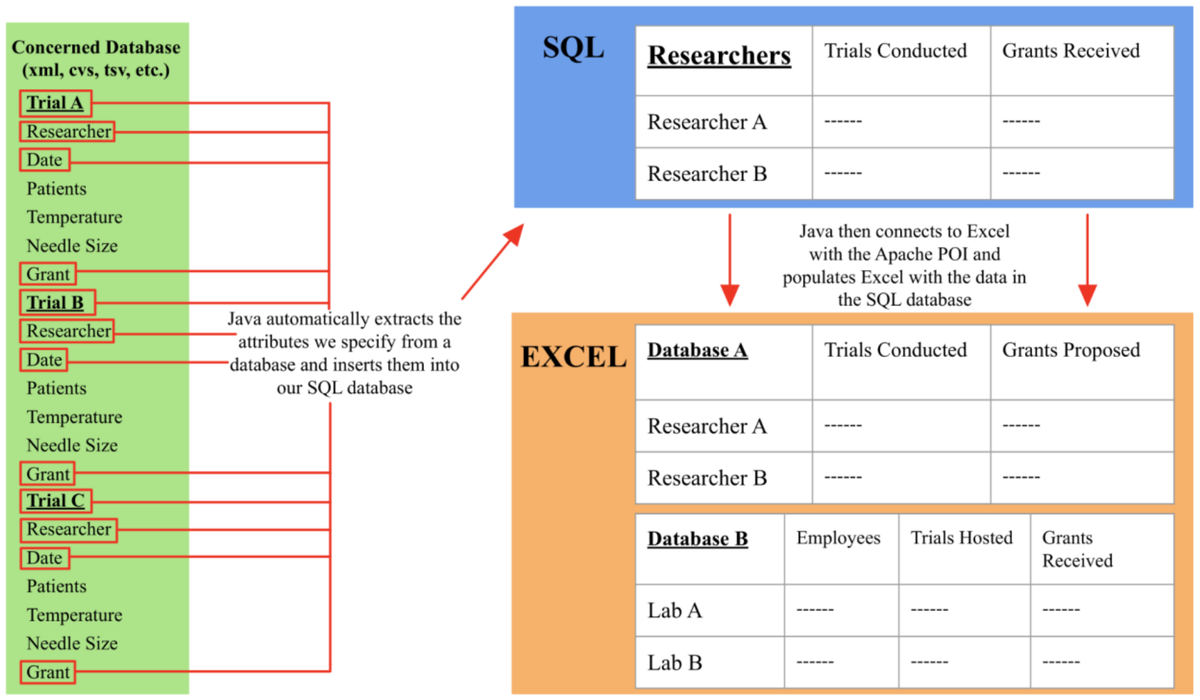
In the broadest of terms, dependencies are something that a dependee relies on to do a task. Dependencies in programming are pieces of software that rely on another to function properly. This could be in the form of libraries or packages of code that are required for a program to function properly. This program has many dependencies, including the Apache POI that connects Excel and Java. Without it, this program would not function.

Apache Maven is a build automation tool used for coding projects, such as those written in Java, which this program is. Maven manages a project's build and dependencies for an easier time integrating dependencies into a program. For example, a dependency that is critical to the function of this project is *mysql*. With this, Java can connect to SQL and make queries or send its own data to it. Maven manages the mysql-connector-java dependency from the POM file, which is a central place a user can specify dependencies the project needs, and Maven will retrieve it.

Tool: Apache POI

Apache POI is a program that provides a way to connect two pieces of software together, like Excel and Java for this program. The “POI” acronym stands for “Poor Obfuscation Implementation” and is a jest at how Microsoft seems to have deliberately, yet poorly, obfuscated any attempts made at interfacing with their applications from an outside piece of software.

In this program, specifically, Apache POI was used to connect Excel and Java together to have the data Java takes from SQL be passed into Excel in a compatible format. These poor obfuscations had provided some trouble in the beginning phases of this program, as I had yet to learn how to connect Java and Excel. Apache POI is very useful for this, being a simple and efficient way of connecting what was needed.



Above is a conceptual image* of how the data within a dataset is transferred to SQL via Java, then to Excel from SQL via Java.

*Image created by Rebecca C. Mann, with full permission to use in this Thesis.

Methods

Collecting and Passing Data

These types of methods are ones that act as connectors between Java, SQL, and Excel. They are the *Write to Excel* and *Write to SQL* methods, and the various methods that connect to the different file types that the program pulls data from, such as .XLS, TXT, and XML.

General Collection, Parsing, and Pathing Methods

The general idea of the methods that connect to files is that they are given a file's location, a list of attributes that the user wants to look for, the file's source (what organization it was pulled from), and credentials to login to SQL. With this data, the connecting methods can know where to look for the file, what to pull, where it came from, and how to pass it further on in the program. The connector method then iterates through a filetype's attributes of data, noting what column the attributes wanted are found at. Then, with that established, it goes through the entirety of the data and pulls data when the column it is currently on matches a column that is an attribute wanted. It pulls the data into its only internally made and held list, and moves on. Once finished with the file, it adds external attributes to the list, such as a timestamp and source, and passes it on.

WriteToSQL

WriteToSQL is the method that passes the data collected from the files to SQL to be put into the appropriate tables. It takes the data list passed to it from a file parser and creates the commands to pass it to SQL. This requires that it first get a copy of the list of attributes. With that, the method sends commands to SQL to have it:

1. Drop the soon-to-be-created table, in case it has already been made. This is done so that there is not an error for trying to create a table with the same name.
    - This will not drop other tables from the same source, however, because tables are named specifically after the filetype and in what order the file is examined and parsed. Ex: xm1, txt2, xml3…
2. Create a new table with the attributes of the list.
3. Create a prepared statement based on the number of attributes the list has.
4. Take each row from the list of data and package it into a prepared statement
5. Finally, send it to SQL to be entered in.

The prepared statements are not necessary for SQL to receive the data, but this allows for a safe and less error prone way of passing the data along without it being intercepted by outside parties. Though this is not something to really worry about, it is good practice to implement and does not impact the overall efficiency of the program. Once SQL receives this data and enters it into the appropriate table, the method prepares for the next Prepared Statement to be sent to it.

WriteToExcel

WriteToExcel is the method that receives data passed to it and outputs it to Excel. It uses Apache POI dependency to be able to do this conveniently and efficiently. It gets passed the workbook that will contain all the data, login information to SQL, a table in SQL to pull data from, and the data that is wanted from the table.

First, the method creates a sheet that will hold an SQL table's data. It then gets data from a table, and puts that data in a list created and maintained by the method. Once finished, it then takes that data, and puts it, row by row, into Excel.

A small note is that, for the "data that is wanted from a table", it would be impossible to know what data is in a table to be able to request it. SQL solves this though, with the "*" symbol. Using this translates to "give me everything in a table" and is a lifesaver.

Main

The main method of this program is large and ever evolving. Currently, it does the following, in order:

1. Create a workbook to put data into

2. Establish a file path
3. Pull login information
4. Create a list of attributes to be pulled for data examination
5. Create a list of specific entities to search for
6. Create a list of table names for SQL to use in data storage
7. Iterate through a list of files for their data
8. Link the data in SQL together
9. Finally, assuming the program works as expected, it exits with `System.exit(0);`. This cuts the program off immediately, allowing for no hang ups or accidentally unclosed connections to continue after the program's purpose has been served.

Essentially, this method is the trunk of the tree that is this program. The other methods listed are branches, ones that hold and interact with the data, or the leaves on the tree.

FilePath Establishers and Creators

In order to reach into a user's file system and get the data files they want to be examined, the program must first be able to find them. For this program, the files will be found in a root or "main" folder wherever the user chooses to download it. From there, the program's file architecture has a "source" folder that houses the program's executables and other information it uses to run and "target" folder that has what the program will use as it runs its course, like a file for SQL login information, and a downloads folder where all the downloaded datasets will be.

The big problem faced here is that not everyone's file path will be the same, even though every user downloads the same program. This happens because the file separators between operating systems are different. For example, Linux distributions like Ubuntu or Arch use "/" while Windows uses "\". Mixing up the separators will cause an error, so programmers have to be vigilant about what separators get used in what operating system.

To solve this, the program uses the EstablishFilePath method to get the current location of the program's in the filepath. With this information, the program saves this path and modifies it to get a base filepath, which would be something like `C:/Users/examplename/Desktop/DatabaseIO/`. With this in hand, the program then creates the file path to the list of files downloaded to pull files to parse.

A second problem faced is that not every operating system uses the same separator. These separators are forward slashes (/) or back slashes (\). Therefore, hardcoding one into the filepath would make it so that users with operating systems that use the alternative separator would run into problems. To mitigate this, the program uses `File.separator`, which gets the appropriate separator and uses this where called.

LinkTable and Linkage

The linkage methods are tricky to describe. Starting with the CreateLinkTable Method, the program creates a table in SQL that will hold every unique entity in all passed databases. To do this, the program takes the first table in SQL, and compares its entries to all entries in the Linkage Table. If the program finds a match, it links the entry in the compared database to the entry in the Link Table. If the method does not find a match in the Link Table, then that entry is added to the link table. Once one of the two states are reached, the program moves on to the next entry, and repeats until the database has no more entries to compare, at which point the next database is selected, and the process starts all over.

With this done, now a user can search for specific entities and get a list back of the data they want across all databases entered. Notably, entities with the same names, like universities or people, will be rolled under one entity erroneously. Better distinction detection methods are needed to solve this issue.

Miscellaneous methods

There are many miscellaneous methods this program uses that are not big enough to be of important note, but still useful enough to be mentioned.

EstablishFileList creates a list of files to be read and used by the main method when it iterates through a list of files for their data. This method pulls the file names from the downloads folder and stores it for later use.

GetLoginInfo pulls login information from a specific file in the program and stores it to be referenced whenever it is needed. This is used mostly to log into SQL.

Problems

Mismatch

Mismatch is one of the biggest problems faced during this project. As was touched on before, accuracy in assigning data to individuals was always the goal, but complete and perfect accuracy is not a guarantee. As the author of this thesis, my own name presents problems. I share my first name, middle initial, and last name with my father. Micheal N. Sleeper references two people, both of which have a career in/would like to have a career in database applications and management. If the both of us were to publish papers in the field and fail to include our full middle name, a cursory glance at the authors could potentially lead to a mismatch of individuals.

A computer would have an exceptionally hard time differentiating the two without additional data between them — without it, a computer would mistake the two for one. Friends of the two would have little problem telling the difference in a face-to-face setting. However, with only data gathered off of artifacts like papers, such as what a computer would be limited to, it would be hard to tell if the Micheal N. Sleeper they have in mind is the right one. When applied to a much wider scope, such as a nationwide one, we get a larger set of data with a higher likelihood of finding people with similar names, or even names that are exact copies. What guarantee do we have that the data we have on J. J. Smith is for the *right* J. J. Smith?

To add another complication to the matter, Micheal N. Sleeper — the thesis writer — goes by Noah Sleeper or Micheal Sleeper, switching between the two in different settings. How would a computer know that the two names refer to the same person, if it found data for both names? If I were to change my last name to Smith, then Noah Sleeper and Noah Smith, while still being the same individual, would be different individuals to a computer at first glance because of the different last name. Though Micheal Smith, Micheal Sleeper, Noah Sleeper, and Noah Smith are all one person, it would be difficult for a computer to know these four names refer to one individual without a unique identifier.

Take the cases presented and apply them to potentially every researcher and institution in the scope of this project. For example, name changes are a common occurrence in society, due to weddings or personal reasons, and even schools change their name (Augusta University has done so three times as of this thesis). Information such as ID numbers, email handles, and other professional records would be quite useful in differentiating people, but there is no guarantee that such information would be publicly available. Knowing someone is who we say they are is difficult, and not always possible. With enough data and refinement on how a program interprets that data, accuracy can be ensured to an acceptable degree, and further tuning based on feedback is always an option.

Coding

The main problem with coding is a question of how best to get from what you intend to a functional product. There are hundreds of methods that are at a programmer's disposal, and exponentially more ways to use them all. To achieve a functioning method, the right method(s) has to be used. Simple goals, like simple arithmetic, are small in scope. Therefore, they are simple to code, and create little hassle in achieving them.

For this project, the multitude of methods needed vary in scope and use. Small methods, like retrieving the current time, are quick and simple. Larger methods, like creating a method to pull data from a text file, require many lines of code to complete. After such a method is made and deemed functional, testing has to begin. What happens to the first line of a text file? Does it get skipped over? What about the last line? Is the last attribute of each line skipped over? Are special characters properly copied? Cases like these take time to properly code a handle for, and vary on a case-by-case basis. The only limit to these larger methods is the number of cases a programmer can think of, because there will always be more edge cases. Always.

One less technical, more psychological problem was the scope of this project and the difficulty it presented felt discouraging at times. To look at what was left to do at any state was like looking up a mountain that was yet to be climbed. It seemed like, regardless of the progress made, there was always a great deal more to be done. Breaking things down into small, achievable goals was a way Dr. Aubert went about things, which made things easy to progress towards. Getting databases from the Downloads folder to SQL with Java became much more manageable when it was broken into steps such as getting Java to read files, getting Java to connect to SQL, and getting Java to input information into SQL.

Dumb errors and mistakes also popped up. When I was working on a method to create a list of data from an XML file, I had a method that would print a list of the first three rows of data to the command line, modified from printing every row of a list. When I went to work on a different method a while later, I had completely forgotten with my edits to the number of lines the method printed and was engaged with the current method I was constructing.

When only a list of three entries appeared, I came up with many reasons as to why this could be: the new list created only had space for three items, I was overwriting anything below the third item with empty space, only three items were getting passed to the method, etc… After a while, I happened to look at the method and think “the method that returns the list only returns the top three rows, per what I dictated earlier.” Once I went back and took this handicap off, I saw that the method was functioning as intended.

This is what I consider to be a “dumb error”. These errors are simple to solve, but difficult to recognize until you take time to step away and cool off. Most times an error is a dumb error, it can be recognized almost immediately after recognizing that there is an error. It is a gut feeling, a situation where you recognize that something is wrong, but cannot pinpoint where or why. These types of errors are reminders to not dive too deep into work or work for longer than a few hours, and to take a step back at times or just a break every once and a while. It is more or less a sign of fatigue, which I ran into often enough to realize that I should take time in between sessions of work or unwind and mentally reinvigorate myself.

Scope

As touched upon earlier, the scope of this program was large for a two man brainstorming and conceptualization team/one man programming team. This program uses many different tools that fit together to make this program work. From an XML file reading technology, to an Excel interface (Apache POI), to a dependency manager (Maven), I had not worked with most of these technologies before. I only had light experience in Java and SQL, as well as a few years of general coding behind me. To create such a large program was to begin reaching out without any idea the best angle to reach at. Most of the technologies used were recommended by Dr. Aubert or posts on coding forums like StackOverflow.

The problem with such a large program was not just the number of technologies needed, but also the understanding required to best/most efficiently code what was needed. The most difficult part of coding was understanding how to link and get data into/in SQL, but thankfully, Dr. Aubert was able to advise me well with his experience in the subject, and his website [2], which I used to a large degree for understanding specifics of code for SQL.

Some of these specific tricks were ones I did not think existed, and also did not think to think about. This program is a product of the best, most efficient practices I know of. If I have methods X, Y, and Z, and there is some niche command that can do all those methods with one, then I would use it if I knew about it, and I would look for that method if I thought to look for it or thought that such a method existed.

Results

With all major functionalities working in one large program, it would be easy to say that this program did exactly what it was intended to do – scrape websites for databases, parse them for important information, and link the entries in multiple SQL databases. However, while it does do that, it is not as dynamic as I would like.

Scraping Websites

In order to pull a dataset from a website, the specific web address of the dataset has to be defined in the program before it tries to pull it. If a user wants to pull from multiple sources, the program has to be supplied with each dataset's link. For example, the link to a collection of nsf .xml files in a given year is

"*https://www.nsf.gov/awardsearch/download?DownloadFileName=*".

Noticeably, the link ends abruptly. The actual files are stored in a naming convention of the year the file was published under followed by the string "&All=true". To pull the files made in 2000, the link would be

“*https://www.nsf.gov/awardsearch/download?DownloadFileName=**2000**&All=true*”.

To pull files for different years, the number “2000” has to be changed. This is accomplished in many ways. One example is with the following *for* loop:

```
for(int year = 2000; year <= 2022; year++) {

      url = “https://www.nsf.gov/awardsearch/download?DownloadFileName=”
      + year + "&All=true";

}
```

The URLs this loop outputs would then be used to pull datasets from the years 2000 to 2022. It is not difficult to code a way to pull the files from this website, but other websites could present a greater challenge. All that must be changed in this example is the year. Other websites may store their files behind links that are not friendly to scraping methods like the one above. An example website might use the link format that incorporates entity initials, like “*https://website.org/FIRST_INITIAL&LAST_INITIAL”*.

For that link, the program must have a predefined list of initials to use. It is not impossible to do, but putting that list together would be time consuming. To add to that, making sure the program pulls the latest datasets would require constant updates to the list of initials, whereas the previous link we used would only have to have the limit year (2022) increased, which is much easier.

All this is to say that how a website stores its databases is important to how its data gets scraped. Since no website will store data in the same way, having a reusable piece of code that can be used for every website a user wants is not possible. This is not too big a deal, but does mean that the program requires a larger amount of time and energy putting together larger collections of websites that can be scraped than if we were able to use one or two universal scraping methods.

One other problem that has to be addressed is if a website changes where it stores its datasets. If the NSF link we used earlier changed the end of the link it uses from `"&All=true"` to `"&AllEntries=true"` or `"&All==true"`, or some other variation, then the links we predefined will not work, and the program can no longer pull databases unless we correct this. This means that any change to the link that we do not replicate will mean that the program will be unable to retrieve a dataset. The only way to get around this is by trying to pull from a specific link, and returning an error if we cannot get any datasets from it. Though it does not solve the problem, it does alert us of its existence, which we can then rectify by updating the link manually.

Linking Entities

In order to link entities in an SQL table, the program has to recognize that two entities are referencing the same entity. At a basic level, this is not hard to do. “Augusta University” and “Virginia Military Institute” are not referring to the same entity, but “Augusta University” and “Augusta University” are. Programming a way to recognize if two strings -- words -- are the same is basic programming. When two strings are not the same but refer to the same entity that it becomes more complicated. “Maryland University” and “Maryland Uni.” reference the same entity, but are not the same. The best way to rectify this is use a dictionary of common words the program will encounter, and how it should treat them.

For example, if it encounters “Uni”, then the program would replace it with “University”, or tell the program to ignore the difference and link them anyway. This can also extend to misspellings. “Marlyand” was one misspelling of “Maryland” found in a file. The program, naturally, did not link the two -- they are different, after all. A dictionary entry that explicitly tells it to view “Marlyand” as “Maryland” would rectify this issue. The larger issue this presents is how can we be sure that our correction is correct -- i.e. if Marlyand University is a separate entity from Maryland University, then our dictionary correction is wrong.

This issue can be corrected with the use of Levenshtein Distance (LD). LD is a measurement of the distance between two strings. LD would be used on two words -- Maryland and Marlyand -- and the output would be how many operations need to be performed on the words to make them the same. For the example words above, the LD is 2. This means two operations have to be done to match the words together. Using this on potential matches would allow easy linkage that would mean we do not have to constantly update a dictionary. If two words have a LD of less than some number, then we say they are close enough words to be considered the same entity, and link them.

Conclusion

In summary, the question of whether or not it would be possible to measure the quality of a researcher or institution based on data on that entity in a way that is not tedious or overbearing is answered with a “Yes”. However, it would take a large amount of foundational work to be able to scrape multiple source and link entities. However, this can be done. It is possible to scrape, parse, link, and output without any human intervention during the process.

References

[1] Aubert, C. (2021) *Webpage for CSCI 3410 - database systems, Augusta University. Database Systems – CSCI 3410*. Available at: https://spots.augusta.edu/caubert/teaching/2021/fall/csci3410/ (Accessed: December 5, 2022).

[2] Sleeper, N. & Aubert, C. (2022). Data Integration for the Study of Outstanding Productivity in Biomedical Research (Version 1.0.0) [Computer software]. https://github.com/popbr/data-integration