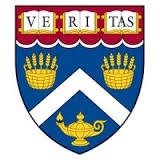
Final Project **Big Data Case Study:   
Institutional Research and Neo4J**

Dalal, Dhairya

**csci e63 Big Data Analytics, 2015**

**Harvard Extension School**

Professor: Zoran B. Djordjević



Abstract 3

Overview 4

Background 4

Goals 4

Data Loading 5

Overview 5

Data Model 6

Loading into Neo4J 7

Data Analysis 9

Data Presentation 10

Conclusion 10

# Abstract

For this final project, this paper will explore if Neo4J is a useful and scalable tool for analysis in Institutional Research. Institutional research is broad field that aims to understand how colleges and universities operate (from admissions and financial aid to faculty productivity and research impact), and leverage university data to aid in policy and decision-making.

I work in Harvard’s Office for Institutional Research, and was intrigued to see if Neo4J offered value to some of the analysis work my office does. In class, we only took a superficial look at Neo4J and created a toy database with a couple of nodes. This paper will explore if Neo4J is scalable to larger datasets and if it offers any novel insights.

Due to sensitive nature of the data we work with, I chose to use a publicly available dataset provided by AMiner, the Open Science Platform, which collects Computer Science publications, their citations, and other useful metadata. It is a large dataset with approximate 2.3 million publications and their associated citation data and metadata.

This paper documents my attempts load the AMiner dataset into Neo4J. Next it documents how I used R to analyze the citation-network using RNeo4J, R, RCharts and the Cypher query language. Finally it covers how I developed an RShiny application to showcase my visualizations and analysis results and the deployment of the application on shinyapps.io.

* Working Demo can be found here:
  + <https://certainentropy.shinyapps.io/CS63_project/>
* Source Code is on github:
  + <https://github.com/ddalal2/cs63_final_project>
* Youtbube recordings can be found here:
  + Long: <https://www.youtube.com/watch?v=zc4-wQmv4Pc>
  + Short: <https://www.youtube.com/watch?v=CznumpkD6_M&feature=youtu.be>

# Overview

## Background

For this final project, we will look at Big Data analysis in the context of Institutional Research (IR) and see if Neo4J is a useful tool for analysis. Institutional research is broad field that aims to understand how colleges and universities operate (from admissions and financial aid to faculty productivity and research impact), and leverage university data to aid in policy and decision-making.

I work in Harvard’s Office for Institutional Research, and was intrigued to see if Neo4J offered value to some of the analysis work my office does. In class, we only took a superficial look at Neo4J and created a toy database with a couple of nodes. This paper will explore if Neo4J is scalable to larger datasets and if it offers any novel insights.

For this project, we will be looking to analyze a citation-network dataset and try to answer interesting institutional questions about research productivity and impact. Due to sensitive nature of the data we work with, I chose to use a publicly available dataset provided by AMiner, the Open Science Platform, which collects Computer Science publications, their citations, and other useful metadata. It is a large dataset with approximate 2.3 million publications and their associated citation data and metadata.

This paper will attempt to see if Neo4J is a useful tool to perform data analysis in the context of a large data and can be used to answer novel IR questions. It will also use R, R Shiny, and RNeo4J. In the goals section below I’ll further explain the goals of this final project.

## Goals

In order to contain the scope of this project, I chose to focus on a specific set of goals for this final project. The primary goals of this project are to:

1. Attempt to load the citation-network dataset into Neo4J
2. Use the cypher query language to answer interesting IR questions about the dataset.
3. Present the findings of the analysis as an interactive website built on R Shiny and R Shiny Server.

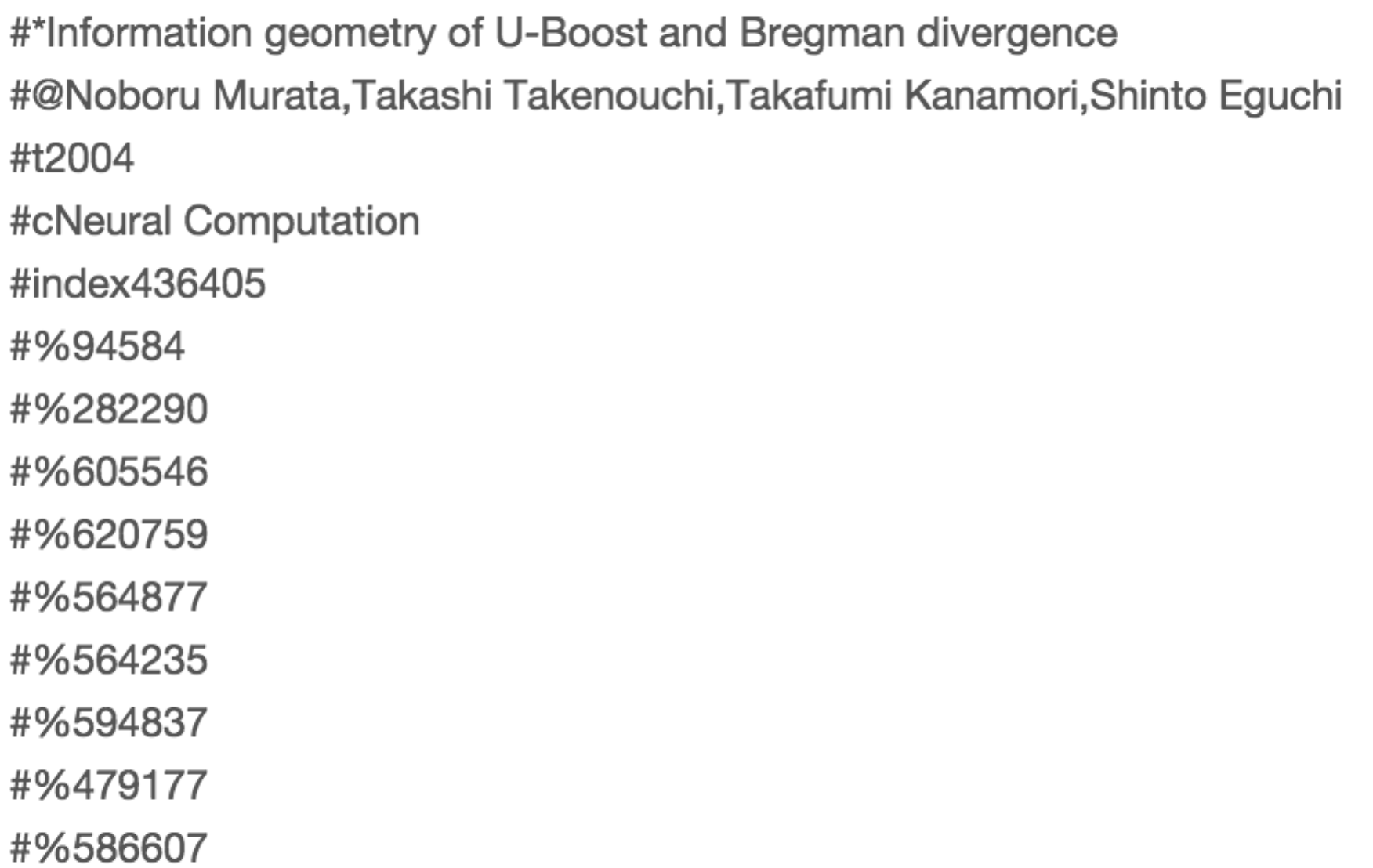
The goals can be broadly categories as Data Loading, Data Analysis and Data Presentation. This paper will split up its findings and methods across the three sections listed above.

# Data Loading

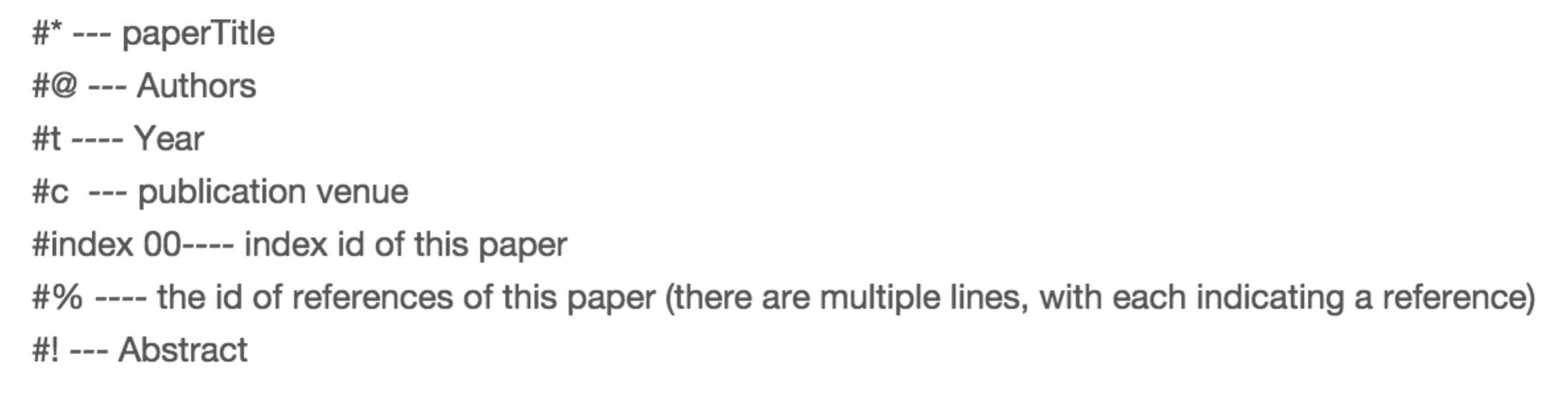
## Overview

The data set for this project was downloaded from AMiner (<http://aminer.org/billboard/citation>). The dataset is called DBLP-Citation-network V7 and is approximately 137 mb compressed and contains 2,244,021 papers and 4,354m534 citation relationships from 2014 to present. It originally was stored in a block format (see fig 1.). Each block represents a single paper, and each line is a property of that paper (e.g. title, author(s), cited papers, etc). Each line is preceded by a special token, which maps to a property (see fig 2).

**Fig 1. Example of block data format**



**Fig 2. Block data format key**



This data transformed into a json file in a separate project by a colleague of mine, Albert Wang using Scala. The JSON file (see fig 3.) is start point for this project. Each row is a JSON object that represent a paper (formerly a block in the original dataset).

**Fig 3. Head of JSON file (console output)**

ds-MacBook-Pro:data dmac$ head -n 2 aminer\_parsed\_lines.json

{"source":"Modern Database Systems","abstract":"","year":1995,"id":2,"authors":["José A. Blakeley"],"title":"OQL[C++]: Extending C++ with an Object Query Capability.","citations":[]}

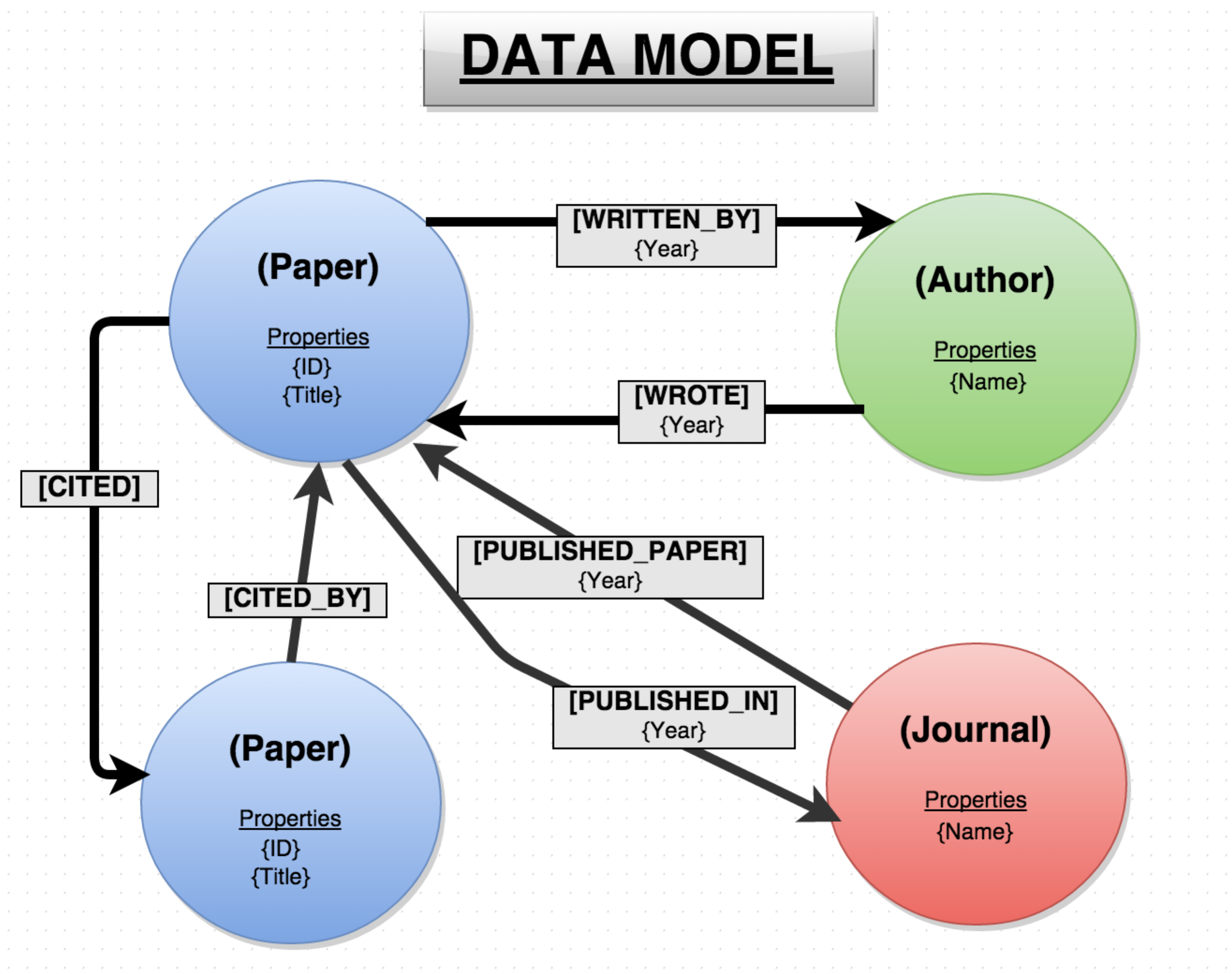
{"source":"Modern Database Systems","abstract":"","year":1995,"id":3,"authors":["Yuri Breitbart","Hector Garcia-Molina","Abraham Silberschatz"],"title":"Transaction Management in Multidatabase Systems.","citations":[]}

ds-MacBook-Pro:data dmac$

Next this paper will describe the intended data model for the Neo4J graph and then it will describe how that data was loaded into Neo4J.

## Data Model

**Fig 4. Data Model**



I chose to create a fairly simple data model with three main nodes: Paper, Author, and Journal. All nodes are unique and indexed. The Paper node is indexed by the id property, while Author and Journal are indexed by name. A paper node can cite or by cited by another paper node, which can be seen in the relationship [CITED] and [CITED\_BY] in fig 4. Also each paper node has relationship with the author and journal node. This simple graph structure also allows for indirect analysis now, which was previously untenable in the original relation data structure. For example, if you are interested in which authors have been published in a particular journal, you can trace a path from Journal – [PUBLISHED\_PAPER] -> Paper – [WRITTEN\_BY] -> Author. This type of path mapping provides powerful mining capabilities and forms the basis of Neo4J’s cypher query language.

Next we’ll talk about loading the data into Neo4J.

## Loading into Neo4J

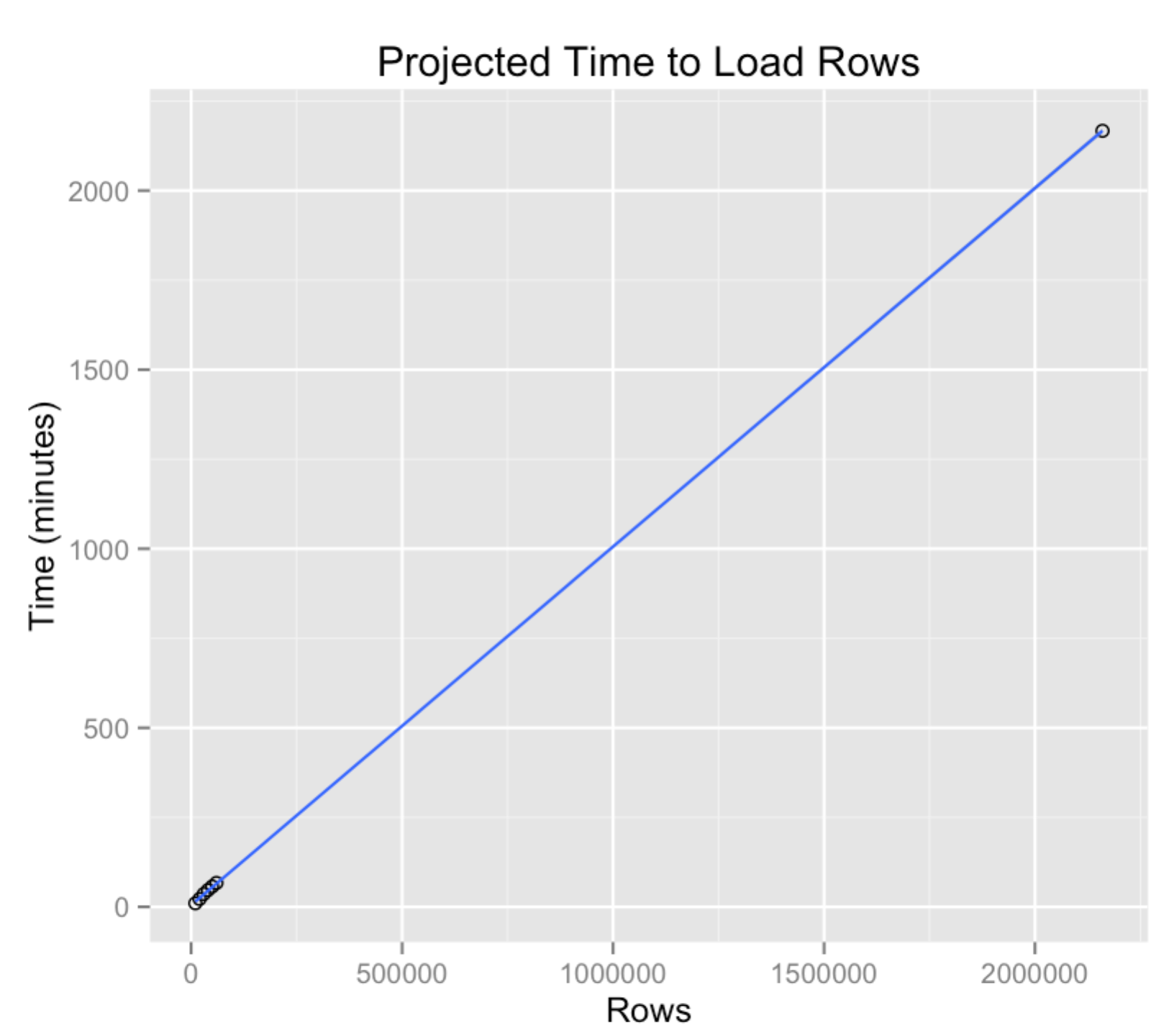
As with most data project, an inordinate amount of time was spent trying to format and load the data. Initially I tried to use RNeo4J which has an API to connect and write to Neo4J databases. Unfortunately that solution proved un-scalable as it took about 1 hour to load 10,000 nodes and was projected to take about 1 year to load the remaining 1.9 million papers ( plus the 4.3 million citation relationships).

Next, I tried to load the data using an embedded Neo4J application in Java. I don’t have too much experience with Java, so it was bit challenging, but I eventually got a program working that directly wrote nodes to a Neo4J database. If you are interested in the code, under code folder **EmbeddedNeo4J.jar** file. Using the class HW as a starting point, I employed the following strategy to import the nodes.

1. Create Neo4J graph object
2. Create uniqueness constraints for Paper, Author and Journal labels.
3. Read in line from JSON dataset and parse JSON object
4. For each line (i.e. paper):
   1. Check if paper exits in graph. If not, create node and set properties (e.g title)
   2. Loop over author(s) and for each author check if author exists as node in graph.
      1. If not, create node.
      2. Create relationships between paper node and author node.
   3. Next check to see if paper’s journal exits in graph as a node.
      1. If not, create node.
      2. Create relationships for paper and journal node.
   4. Loop over citations and for each citation check if cited paper exists as paper node in graph.
      1. If not, create node
      2. Create relationships for paper and cited papers.
5. Close input stream.
6. Shut down graph object.

So this was not the most optimal workflow creating. My program had some canary calls to provide time benchmarks as to how long it was taking load rows. Basically every 10,000 rows my program made a console call with a time elapsed time stamp. About 1 hour in, I could tell it would not be feasible to load all rows in a manageable time. Based on the data points I had, it had taken on average approximately 10 minutes to load 10,000 rows. Part of the challenge way, my program was adding new author and citation nodes for each row in for each. So on the top initial for loop to ingest 2.1 million rows, it had two for loops running to create cited papers and author nodes. The time it would take to load all the rows is a constant O(N). But still this is a linear growth and projects to 2,167 hours to load 2.1 million rows or about a day and half (see Fig 5.).

**Fig 5. Projected Time to Load entire Dataset**



The other reason, I think it took so long to load all these rows, was that I was employing the create unique nodes strategy, where for each node creation, I was checking against the index if the node existed in the graph or not. This probably also added costly overhead time to the creation of the graph. Finally, it should be noted, that Neo4J creates an index for all node and relationship combinations. The end result is rather large database. At 200 thousand nodes (my final graph size) the database files took up nearly 400 mb of space. Given that I was running this on a laptop with limited resources (8 gb of RAM and a Intel i5), I ended up stopping the application after it loaded about 60,000 rows. Next, this paper will take a look at the data analysis part of the project.

# Data Analysis

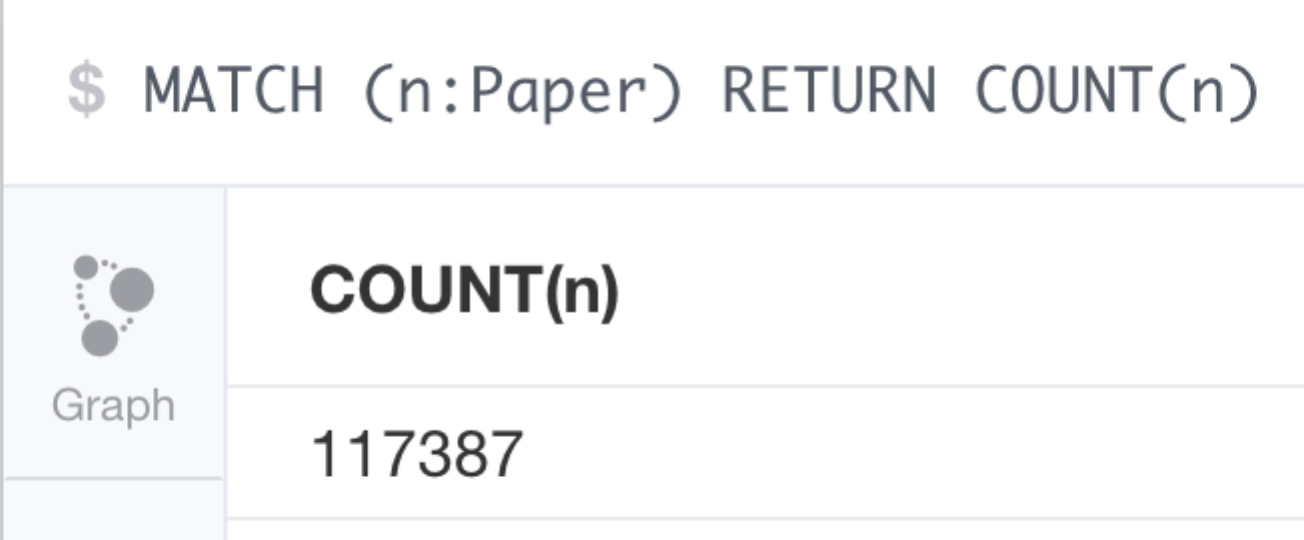
Now we can finally take a look the analytic capabilities of Neo4J. First, this paper will do a quick exploration in the Neo4J browser. Then it will use RNeo4J to run queries and load results into dataframes that can be used to build visualizations for our final demo.

After starting the Neo4J browser (command line bin/neo4j start in the Neo4j directory), we can take a quick cursory look at the size of database. Our graph has a total of 200,125 nodes, of which 82,233 are Authors, 113,387 are Papers, and 505 are Journals.

**Fig 6. and Fig 7. : Total Nodes and Number of Author Nodes**

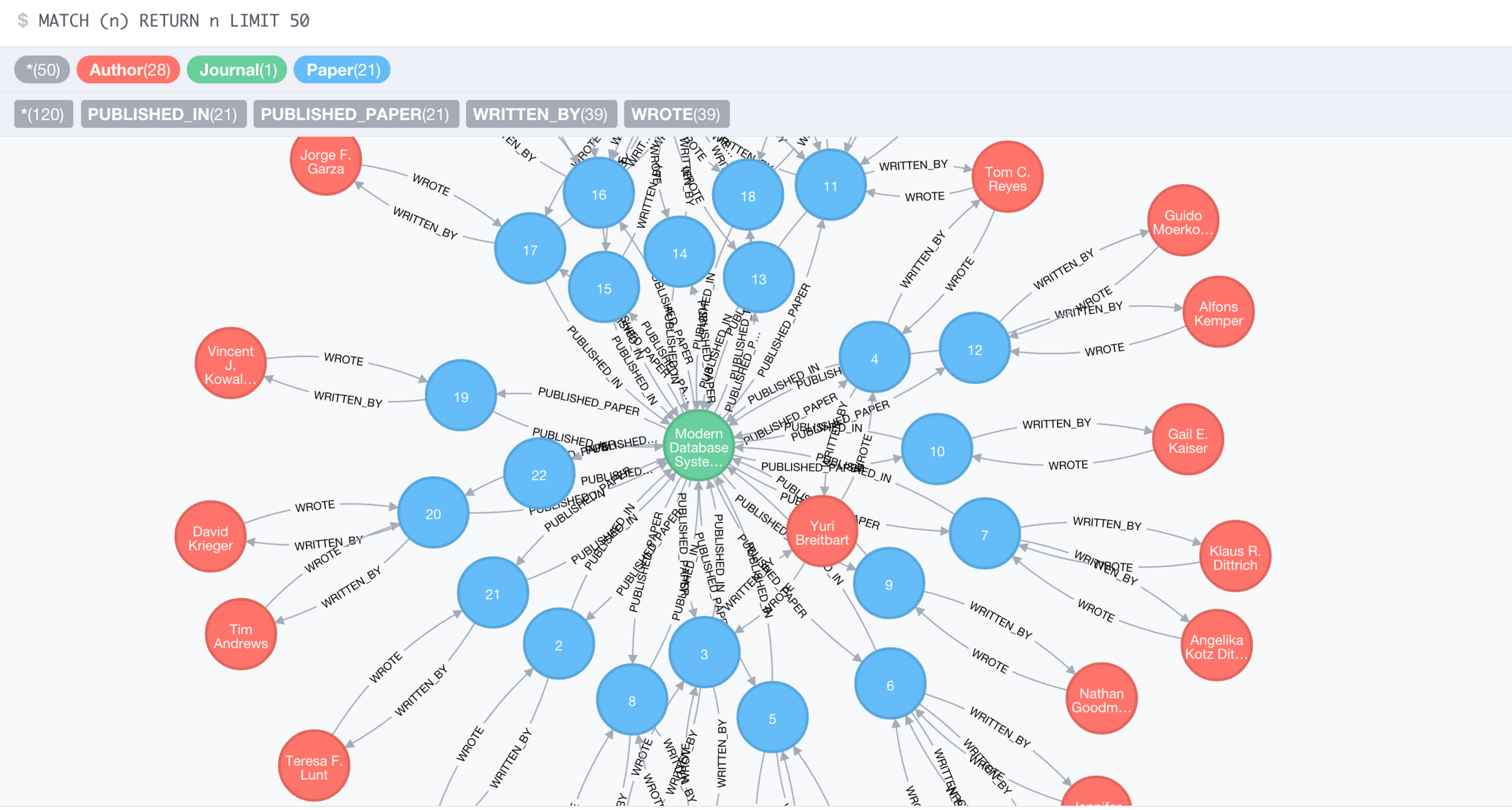
 

**Fig 8. and 9.: Total Paper Nodes and Total Journal Nodes**

We can also take a look at the graph looks like visually.

**Fig 10. A subsection (50 Nodes) of Graph**



Next in R, I created the file Analysis.R. This is where I did analyzed the graph database to answer a few novel questions. I focused on four questions for this project:

1. Which authors have the most publications and what is their longevity score?
2. Who are the rising stars in the field?
3. Which authors are the most sociable, that who has the most co-authors?
4. What are the top journals that have on average the most citations per paper published?

To conduct this analysis, I used the RNeo4J library. RNeo4J is an API that connects to a running server instance of Neo4J and allows you to send cypher commands to the server and store the results in a dataframe. The flexibility of dataframes allowed me to answer certain questions that I wasn’t able entirely capture in a cypher query.

After installing RNeo4j (install.packages(“RNeo4J)), I first launched neo4j locally (bin/neo4j start). Next I created an RNeo4J graph object using the following syntax:

graph = startGraph("http://localhost:7474/db/data/",

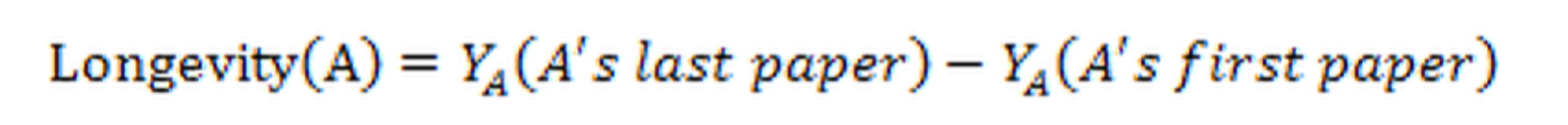
username = 'neo4j',

password = 'password')

All this does is create a socket connection to the neo4j local server and allows you to send cypher commands in R using the cypher(graph, query) method.

## IR Question 1 Analysis

To answer this question we need to know two things. One how many total publications each researcher has published and two what is their longevity score. The longevity score is metric from AMiner (<http://aminer.org/ranks/ranks-res-help>), which basically measures how long they have producing research. The longevity score can written as:



where we take the year of the author’s first paper and subtract it from the year of the author’s latest paper published. We can represent this query in cypher as:

MATCH (a:Author) -[w:WROTE]-> (p:Paper)

RETURN a.name AS Name, COUNT(p) AS Count,

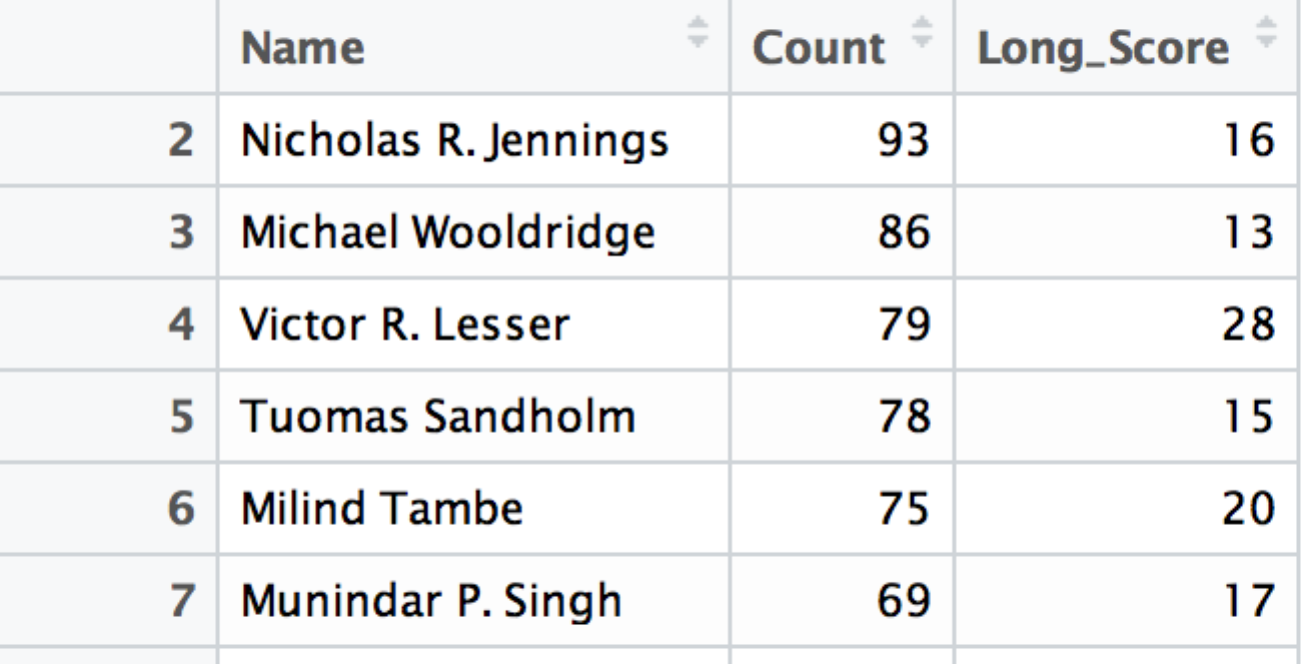
MAX(w.year) - MIN(w.year) AS Long\_Score

ORDER BY COUNT(p) DESC LIMIT 26

We run this query in R with the following code: q1\_results = cypher(graph, q1)[-1,],

which passes our query(*q1*) to the cypher command and returns a dataframe which is stored in the variable q1\_results. The resulting dataframe looks like:

**Fig 11. R Dataframe with query results**

**

Now that we have our results, we can visualize it using rCharts. rCharts is an R library that wraps common javascripts visualization libraries like Highcharts.js and D3.js. The syntax for rCharts is pretty straight forward.

Each chart follows the following workflow:

1. We create a highcharts object
   1. q1 <- rCharts:::Highcharts$new()
2. Next we designate the chart type
   1. q1$chart(zoomType = "xy", type = 'bar')
   2. Zoomtype allows the use to zoom in and out of the chart across the X and Y axis
3. Then we set up the xAxis
   1. q1$xAxis(categories = q1\_results$Name, title = list(text="Authors"))
   2. categories refers to the names in the Name column in my dataframe
   3. title set the label for the X axis
4. Next we add the actual data for the bars in the same order as the names in step 3 for the longevity scores.
   1. q1$series(data = q1\_results$Long\_Score, name = "Longevity Score (years)", legendIndex = 2 )
   2. the data here is the Long\_scre column in our dataframe
   3. legend index set the order of the series in the legend
5. Rinse and repeat for publication counts
   1. q1$series(data = q1\_results$Count, name = "Publication Count", legendIndex = 1)
6. Create a title for the graph
   1. q1$title(text="Most Published Authors")
7. Finally display the graph in the RStudio console
   1. q1$show()

The final result is an interactive highcharts chart that will appear in RStudio’s view panel.

**Fig 12. IR Question 1 chart in rCharts**



Based on our analysis, we can see that Nicholas Jennings has the most publications (93) in our dataset and has been publishing research for about 16 years.

## IR Question 2 Analysis

Our next question will look at who are the risings stars in the field. A rising star is defined as someone whose first publication was in the past 5 years. I wasn’t quite about to figure out how to filter the long\_scores calculated in our first question. So to answer this question I used the cypher query from above and removed the limit, so we got back the full list of authors, their publications, and longevity scores. Next I used dplyr to filter the results, reorder them, and sub-select the top 25 results. The dplyr code to achieve is:

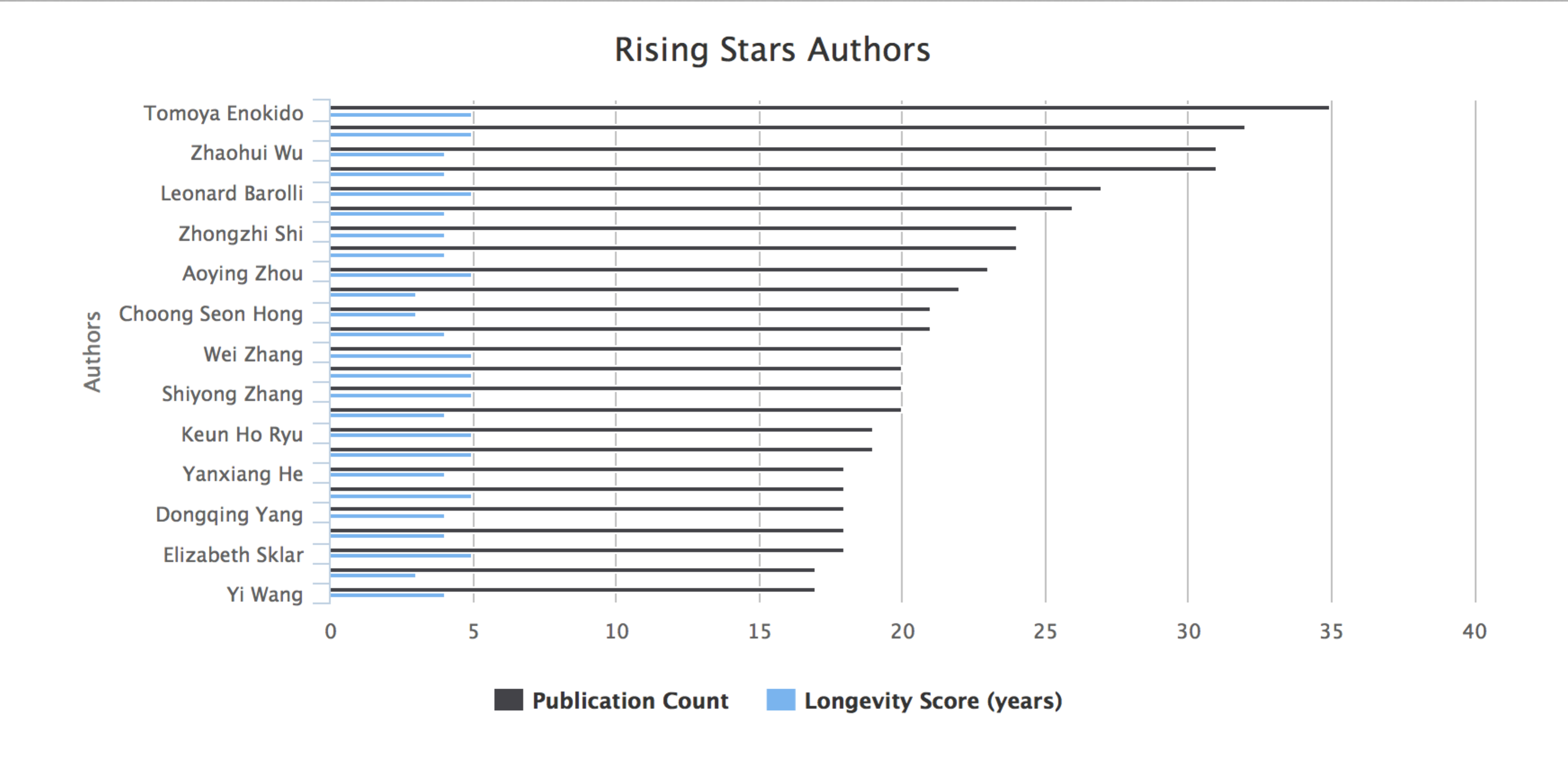
q2\_results <- (q2\_results %>% filter(Long\_Score <= 5) %>%

arrange(desc(Count))

)[1:25,]

Finally, we visualize our dataset in rCharts. The final result is the following chart:

**Fig 14. IR Question 2 Chart**



The brightest rising start in this dataset is Tomoya Enokido who has published 35 articles in 5 years.

## IR Question 3 Analysis

The next question we looked at was which authors were the most sociable. That is who had the most coauthors across all their papers. Again I used AMiner’s sociability score, which can be expressed as:

**Fig 15. Sociability Score Equation**

# 

To answer this question I used the following cypher query:

MATCH (a:Author) -[WROTE]-> (p:Paper)<-[WRITTEN\_BY]- (o:Author)

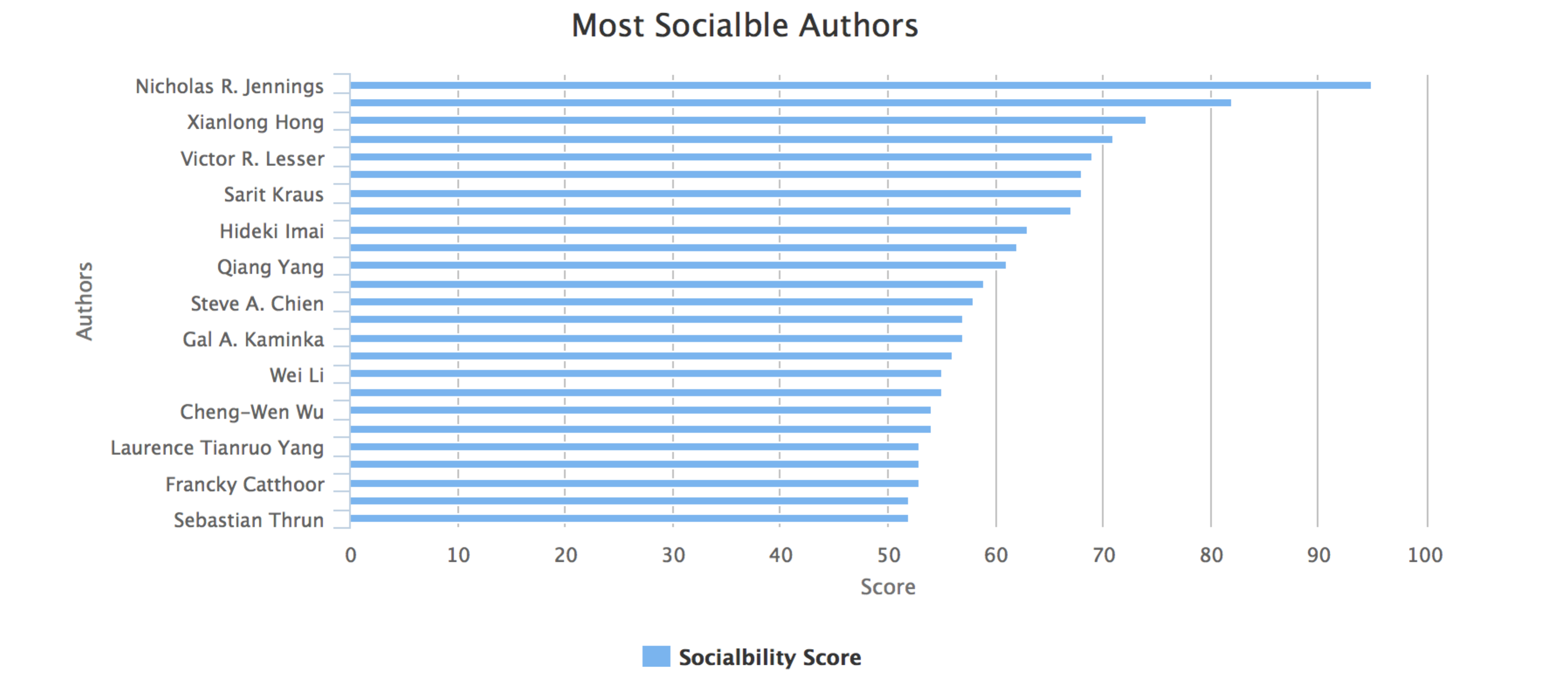
RETURN a.name as Name, 1 + COUNT(DISTINCT(o)) as Social\_Score

ORDER BY Social\_Score DESC

LIMIT 25

After visualizing the results, we get the following chart:

**Fig 16. Most Sociable Authors in our Dataset**

****

The most sociable author being Nicholas R. Jennings who has had 95 co-authors across all his papers.

## IR Question 4

The final question we looked at was on average, which journal had the most citations per published paper. Again I wasn’t quite sure how to express this in a cypher query. So the closest I was able to get was to get all papers and their citations for all journals using the following cypher query:

MATCH (j:Journal) -[PUBLISHED\_PAPER]->(p:Paper),

(p:Paper)<-[c:CITED\_BY] - (o:Paper)

RETURN j.name as Journal, p.title as Paper, COUNT(o) as Citations

So I used the following dplyr chain to calculate my results:

q4\_results <- q4\_results %>%

**group\_by**(Journal) %>%

**summarise**(paper\_count = n(), c\_count = sum(Citations),

avg\_cits = round(c\_count / paper\_count,2)) %>%

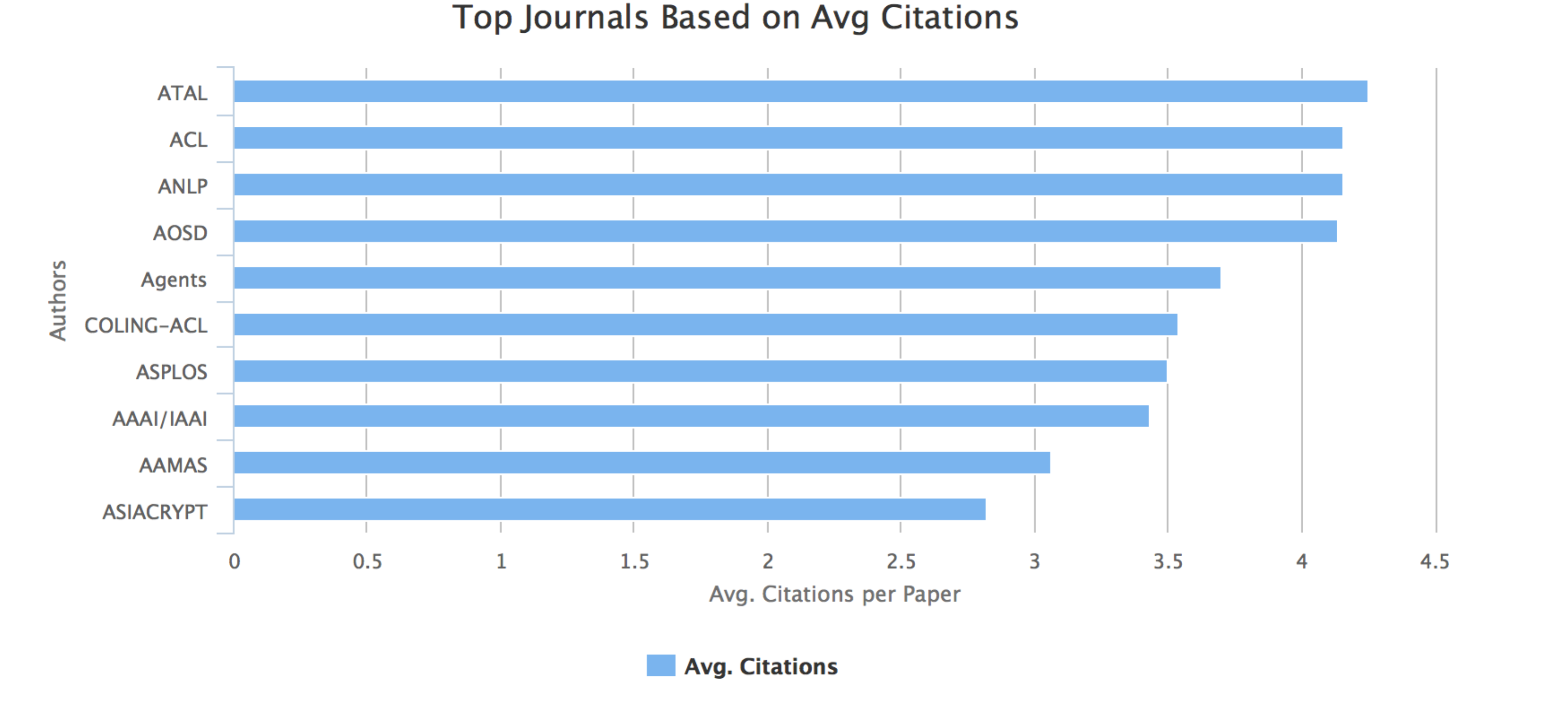
**filter**(paper\_count > 70) %>%

**arrange**(desc(avg\_cits))

The dplyr chain basically grouped all journals together, divided the total count of citations across all the papers by the total number of papers (which is the avg citation count), filtered out journals with less 70 published papers (to account small n skew) and finally arranged the data in descending order with highest average citation count on top.

The final top ten results looked like this visualized:

**Fig 17. Top 10 journals based on avg citations per paper**



# Data Presentation

To present our data, I used RShiny to web a web application that would showcase the visualizations I made in R. The RShiny app was deployed on shinyapp.io. The source code for this app can be found in the code folder. Since the focus of this paper was Neo4J, I won’t go into the details for developing an RShiny app.

# Final Thoughts

So I found Cypher, Neo4J query language really intriguing and powerful in its simplicity and robust in its extensiveness. I believe there is much value graph thinking and analysis would offer to the field of Institutional Research and to the work our office does. Unfortunately, I found it very difficult to load a large dataset in Neo4J. Perhaps if I have time for more research, there will be an opportunity to learn more efficient loading mechanisms. Within the context of our office, I can see this being a useful tool to explore smaller flat files. But more research would need to be done to see how we can use Neo4J to explore our more extensive data warehouse in scalable and efficient way.