**Name: Aakash Thatte**

**Roll number: 12041590**

**Wikipedia Graph Analysis**

Total number of nodes collected = 1,26,584

Task : Label Propagation

In this method, I have first used some links collected from wikipedia and hand labeled by me.

Then using web scraping, I visited each of those links, extracted the links present on that page and created nodes for our wikipedia graph. Then after the graph was constructed, I wrote a loop to implement the label propagation algorithm till each of the nodes gets some label.

The algorithm was:

While all nodes get a label:

For each node in graph

Visit node

If node does not have a label:

do nothing

Else:

Give the label of this node to all its neighbors

For each node in graph:

Visit node

If node does not have a label:

give the most frequent of the labels that it received

The code for this implementation can be found in the notebook LabelPropagation.ipynb

Task : NLP features

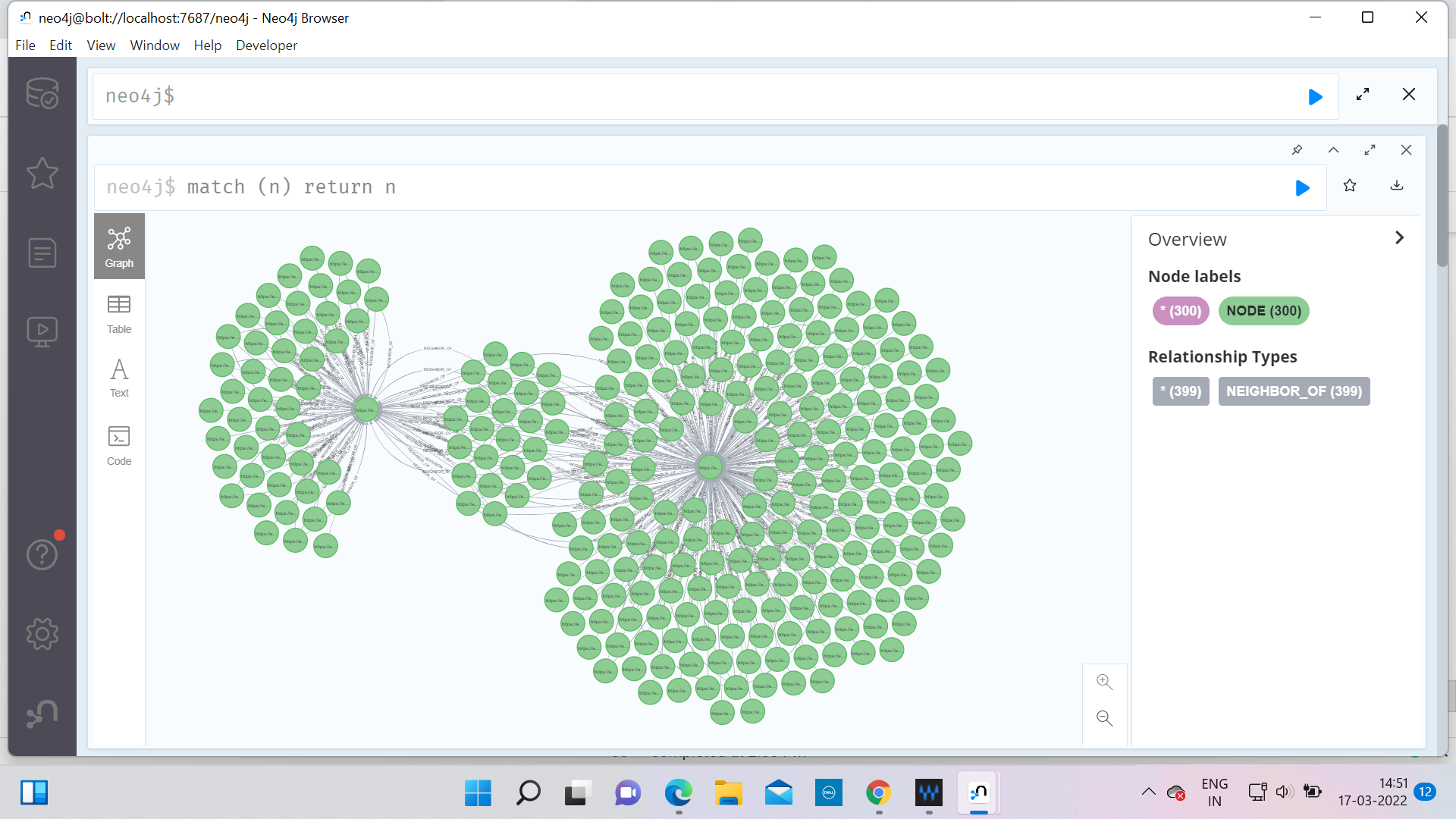
In this, I have used a library known as gensim which has a Doc2Vec functionality which can map a document to a vector. This library essentially uses aggregation functions to combine the embeddings of individual words present in the document and comes up with an embedding that represents the entire document.

Using the embeddings generated from Doc2Vec, we can use knn or simple similarity search to find documents which are similar to each other. This way, we can suggest to users many documents similar to a particular document that they liked.

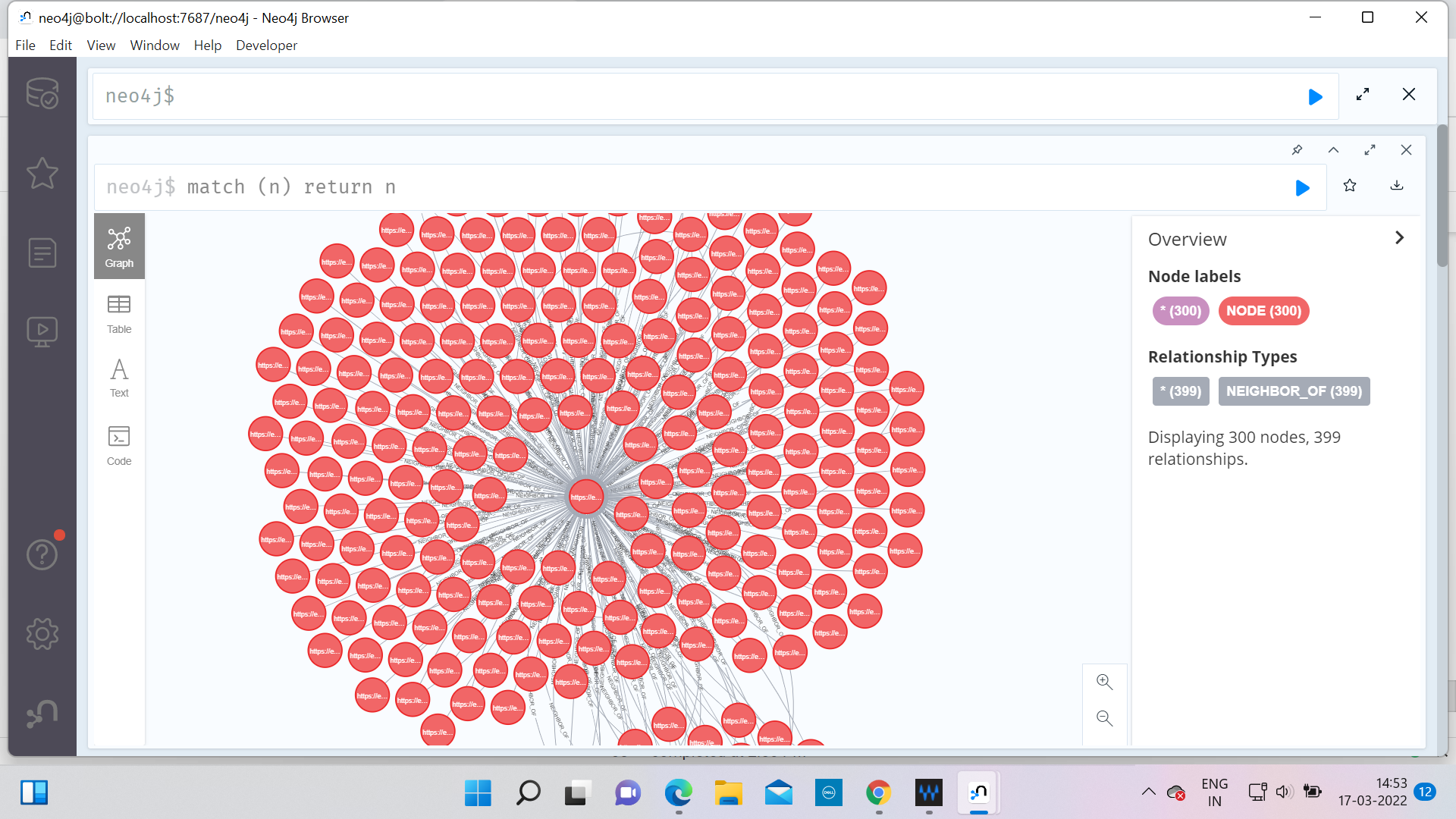
The code can be found in the notebook named NLP\_Features.ipynb

Task: Applying various centrality algorithms and finding communities.

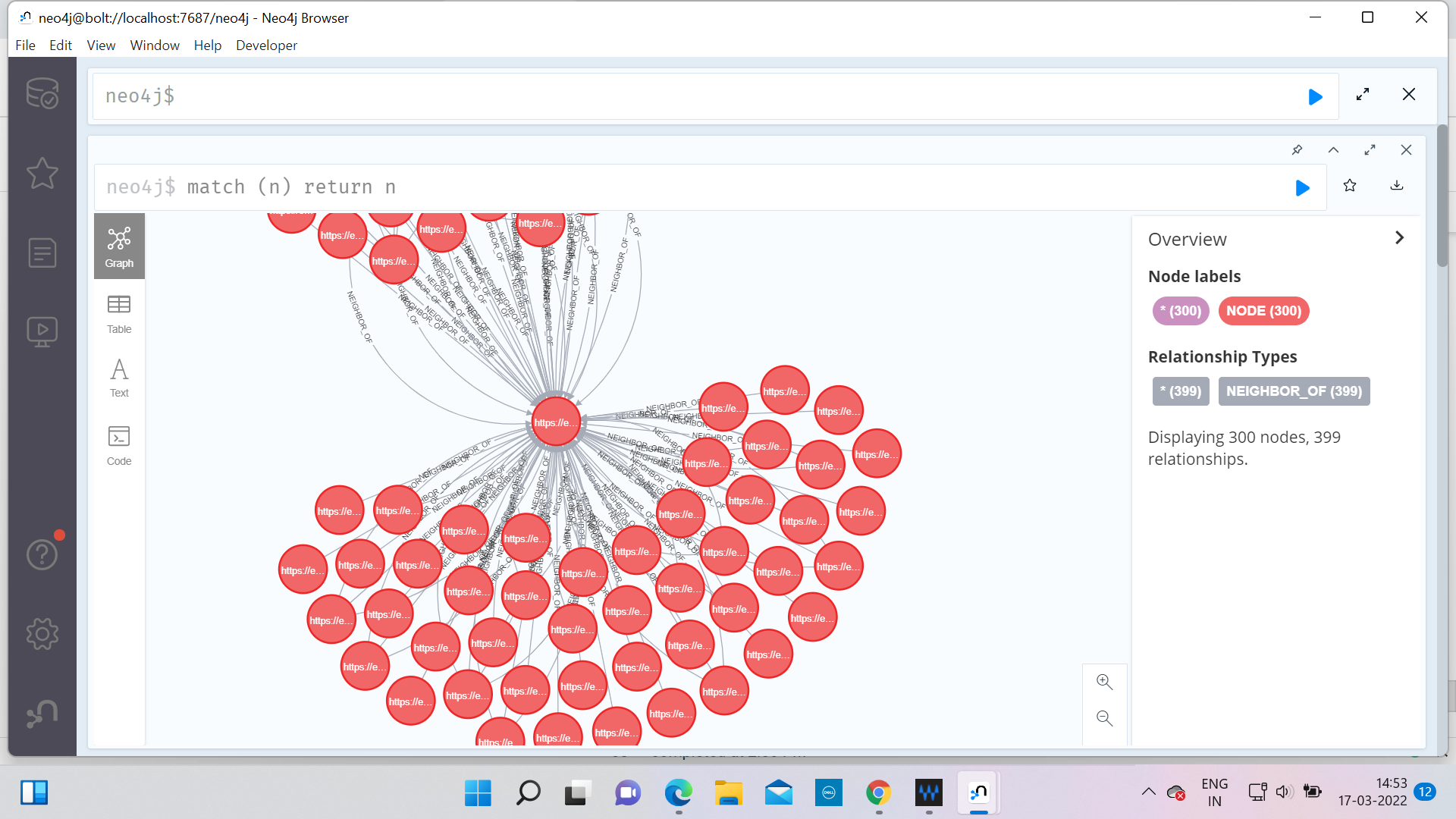
I have used Neo4j to visualize some small sub-graphs of the big wikipedia graph.



We can see in the above image how three different groups of nodes can be seen, which have three different labels (labels taken from Label Propagation part). We can say that these are three different communities present in this small subgraph (I generated this sub-graph randomly from the big wikipedia graph).



We can see here that the central node (it is for antiderivative) has a very high degree as it is connected to many nodes.

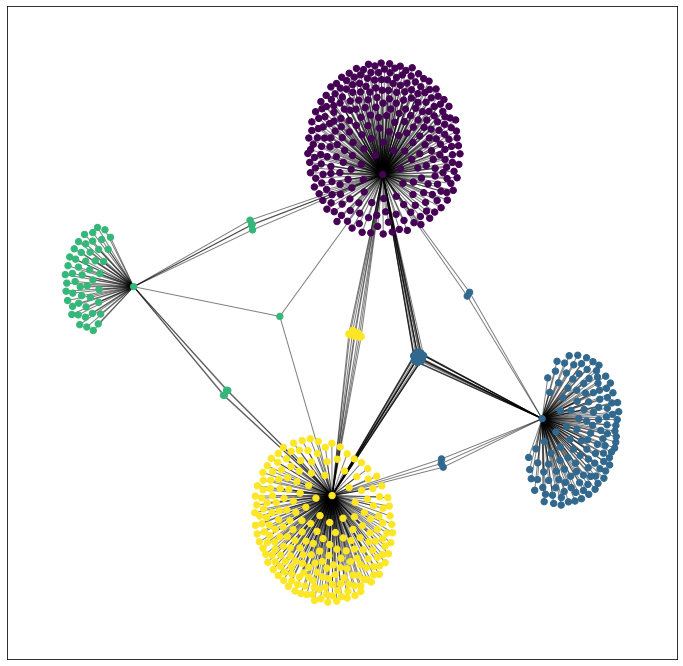


We can see in the above image that this particular node (it is node for Inverse Trigonometric functions) which lies in between the two groups of nodes has a very high betweenness centrality (intuitively we can say this because it kind of lies on all those paths which connect the two groups and thus it can control the flow of information between those two groups).

To find various centrality measures and communities, I have also tried with the networkx library.

The code can be found in notebook names Algorithms.ipynb. Check the image below to find

communities in this subgraph.



Task: Clustering algorithms

I found out the number of triangles that each node is part of using the networkx library.

Also I computed the clustering coefficient of each node using the same library.

Mistakes:

There is one major flaw in the way we have implemented our code. This came to our notice very late (to a point where we are not in a position to run all notebooks). Actually, while collecting the nodes, we have used a for loop and we break out of it when we have collected enough nodes.

After that, we thought that the graph had been explored. But the major issue is that there might be many nodes still in the BFS queue that might not have been explored as we broke out of the loop. This results in less number of edges between nodes than what was supposed to be. This will actually impact all the results and thus needs to be fixed. A solution to this is that we can explore the BFS again after collecting enough nodes. This will ensure that all nodes get explored at least once and thus we won’t miss on any edge.