Documentation for Creating a Knowledge Graph with Neo4j: A Simple Machine Learning Approach

Varghese, Justin Ebby([jv8dz@umsystem.edu](mailto:jv8dz@umsystem.edu))

CS PhD student

EECS Department

University of missouri

March 2023

2023

Table of Contents

[1. Overview 2](#_Toc131640273)

[2. Preliminary Setup 3](#_Toc131640274)

[API Key for Google Knowledge Graph: 3](#_Toc131640275)

[Connecting Notebooks to your Sandbox instance: 4](#_Toc131640276)

[3. Workflow 5](#_Toc131640277)

[Method 1: NLP Only Approach: 5](#_Toc131640278)

[1. Notebook 00 6](#_Toc131640279)

[2. Notebook 01 6](#_Toc131640280)

[3. After implementing Notebooks 00/01: 7](#_Toc131640281)

[4. Notebook 02 15](#_Toc131640282)

[5. During Notebook 02 execution 16](#_Toc131640283)

[Method 2: NLP Lite Approach: 23](#_Toc131640284)

[1. Notebook 03 24](#_Toc131640285)

[2. After running Notebook 03 25](#_Toc131640286)

[3. Notebook 04 30](#_Toc131640287)

Documentation for Creating a Knowledge Graph with Neo4j: A Simple Machine Learning Approach

# 1. Overview

A knowledge graph is a powerful way to represent and visualize complex relationships between entities in a structured manner. A knowledge graph consists of a set of nodes and edges that represent entities and their relationships, respectively. Each node and edge can have properties that provide additional information about the entity or relationship. A knowledge graph is generally constructed using Natural Language Processing, data mining and machine learning methods. The data can be sourced from datasets, web pages, etc. The resulting graph can be integrated into different computer systems and used for question answering, analysis, visualization, summarization, etc.  
However, creating a knowledge graph from unstructured text can be a challenging task. The text can be ambiguous, replete with incomplete information, noisy, complex from a linguistic and domain specific standpoint, and contain large volumes of data. In this documentation, we will discuss how to create a knowledge graph in Neo4j from unstructured text.  
For more insight and an explanation of the theory, please watch this training video from Neo4j by Claire Sullivan, PHD.

Neo4j Training video: <https://www.youtube.com/watch?v=nYQLp7itZx8>.

Neo4j GitHub Repo: <https://github.com/cj2001/nodes2021_kg_workshop>

Goals:

* Given some unstructured text, use two different approaches to extract entities and relationships.
* Create a knowledge graph from the extracted information.
* Vectorize the created knowledge graph (create graph embeddings)
* Apply data science and machine learning to knowledge graph

Tools

* Spacy
* Wikipedia Python Package
* Google Knowledge Graph
* Pywikibot
* Neo4j
  + Awesome Procedures on Cypher (APOC)
  + Graph Data Science Library (GDS)
  + Cypher

# 2. Preliminary Setup

Before we jump into the meat of this documentation, let’s finish some preliminary steps so that we can follow the documentation clearly.

## API Key for Google Knowledge Graph:

The Google Knowledge Graph is used for demonstration purposes in this documentation. You can substitute in any other data source in its place.  
To query the Google Knowledge Graph, you need to first create an API key. This key will allow you to have 100,000 read calls per day per project for free. You can find/read up on a more extensive documentation for creating the key at <https://developers.google.com/knowledge-graph/how-tos/authorizing>.

1. We will be creating an API key and not an OAuth 2.0 token. If you don’t already have a key, go to <https://console.cloud.google.com/apis/credentials>. You should see a page that probably looks like the image below. You mostly won’t have any keys under API keys.

Graphical user interface, application

Description automatically generated

1. Click on ‘Create Credentials’ and then on ‘API key’.

Graphical user interface, text, application, email

Description automatically generated

Once the key is created, you’ll see a popup window that displays your API key. Copy it somewhere safe. Your key should now be listed under ‘API keys’. You can also view it later by clicking on the ‘Show key’ button.

You’ll need this key when you are running the notebooks (or your implementation of these notebooks) at <https://github.com/cj2001/nodes2021_kg_workshop/tree/main/notebooks>.

## Connecting Notebooks to your Sandbox instance:

In Notebooks, there is a particular point where you need to provide a ‘url’ and a ‘password’ value to connect to a sandbox instance of neo4j. This will allow you to connect your Notebook implementation to a Neo4j sandbox instance and add the nodes and edges to the sandbox instance. It will look something like this:

*graph = Graph("bolt://neo4j:7687", name="neo4j", password="kgDemo")*

For more details on how to create a Neo4j sandbox instance and on where to get the ‘url’ and ‘password’ values, check the ["During Notebook 02 execution" section (a).](#Connect_sandbox_instance)

# Workflow

The workflow for going through the major steps in the documentation is shown in the diagram below.  
The text in bold, in the image below shows the names of the sections in this document.

Diagram

Description automatically generated

There are 2 methods of creating a knowledge graph in this documentation.

1. Method 1: NLP only approach
2. Method 2: NLP lite approach

Method 1: NLP Only Approach:  
This method is based on Natural Language Processing, and it uses Spacy to extract triples in the form of (subject, verb, object) from Wikipedia and Google Knowledge Graph. This approach will get you a limitless number of verbs, since they are auto detected from text, but leave you with entity disambiguation.   
Entity disambiguation helps to figure out what entity is being referred to by any subject/verb/object extracted from the text. For example, the word “Paris” can refer to “Paris of Troy” from Greek Mythology, or the city “Paris” which is the capital of France, or the city of “Paris” in Missouri.  
This approach will be a more complex technique as it will rely on the ability to detect verbs accurately and associate them with subjects and objects, which is complicated. It will require more tuning since it is so complicated. It needs to be very specific to the vocabulary and language of the provided text.

You need to run Notebooks 00, 01 to extract data, create a knowledge graph from the extracted information and create embeddings for it.

### Notebook 00

Purpose: Put together a knowledge graph  
In this you will use Spacy to perform entity recognition. Some entity labels like (“Person”, “ORG”, etc) are created to recognize those specific entity types. Some labels are created to determine what type of words will be recognized as a subject, object, or a verb. You will then create a Spacy pipeline with ‘merge\_noun\_chunks’ to merge separate words that are part of the same entity. For example, “Barack”, “Hussein” and “Obama” will be merged into a single token since these words together form the name of a single unique person/entity.  
The Google Knowledge Graph API will get the names, types, descriptions and urls of the entities recognized by Spacy from Google. The helper functions will clean the data and then figure out which of the named entities are subjects and objects and then find the closest verbs to each of them. Some helper functions are used to create node properties for all the recognized objects in the list. ML models in Spacy are used to create word vector embeddings for each node based on the description property associated with that node.

Duplicate nodes are removed and then the word embeddings are added as a node property (if needed later), and then all the nodes and edges are added to the graph.

The graph is created by connecting to a Sandbox instance of Neo4j. You will first need to create a Sandbox instance and then connect to it from wherever you are running the Notebooks (Jupyter or Google Colab).

You also do some entity disambiguation by using cosine vectors of nodes to see if they are the same entity.  
**While running this Notebook:**  
If you face this error while running notebook 00, please consider that the either 1 or both entities that you are using as parameters in the get\_word\_vec\_similarity() function may not be present in the extracted text. In this case, the entity that is absent from the text is ‘mitch mcconnell’. You can substitute this entity with another entity that is present in the text – say ‘Joe Biden’.  
Graphical user interface, text, application, email

Description automatically generated

### Notebook 01

Purpose: Put together a knowledge graph

This is built on top of what was done in Notebook 00 and results in an improved graph over that of Notebook 00. A big problem in Notebook 00 is the absence of relevant data. As a domain expert, you will notice that certain important data that is/will be relevant in the knowledge graph might not have been extracted from the text. For example, “Michelle Obama” was such an entity that was observed to be absent in the results of Notebook 00. Michelle Obama is a relevant entity if the text being used as a source and the graph being created is about Barack Obama.

Notebook 01 uses all the code from 00. The texts used are from Wikipedia about both Barack and Michelle Obama. Then the node and edge list from these texts are assembled and added to the graph.  
The output of Notebook 01 can be extracted as a json file, which then serves as an input to the “**After implementing Notebooks 00/01**”.

1. After implementing Notebooks 00/01:  
   **Please run this preferably after running Notebook 01.** The output of Notebook 00/01 will be a json file (which is also present in <https://github.com/cj2001/nodes2021_kg_workshop/tree/main/json_files>). This json file is used to populate the graph in the Neo4j Sandbox instance.
2. **Create blank sandbox on neo4j** (<https://sandbox.neo4j.com/>)

Sign up or login to the neo4j sandbox website. You can use your Google login to log on to the sandbox site.

Once there, you’ll see a ton of prebuilt sandboxes. Select the ‘Blank Sandbox’ option under ‘Your own data’. Click on the ‘Create’ button on the lower left part of the screen.

It will take a minute or two for neo4j to spin up your sandbox instance.

Graphical user interface, application

Description automatically generated

You can click on the ‘Open’ button next to your sandbox instance after a minute.

Graphical user interface, application, Teams

Description automatically generated

You should see the loading bar which shows that the sandbox instance has been successfully instantiated.

If you don’t see the loading bar as seen in the image below (you might just see a blank screen or a 404 timeout or … something else), just close this browser tab and wait for a couple of minutes and open the sandbox instance again.

Timeline

Description automatically generated

1. You can see some json files seen in the link here - <https://github.com/cj2001/nodes2021_kg_workshop/tree/main/json_files>

These json files contain data that was scraped off Wikipedia using Pywikibot, gone through some NLP processing and turned into a graph. (<https://github.com/cj2001/nodes2021_kg_workshop/blob/main/requirements.txt>).

We can import an SVO graph into our sandbox instance from one of these json files using the following cypher query.

CALL apoc.import.json("https://raw.githubusercontent.com/cj2001/nodes2021\_kg\_workshop/main/json\_files/svo.json")

Graphical user interface, text, application, email

Description automatically generated

If this query gives an error first run the following command and then re-run the first query.

CREATE CONSTRAINT FOR (n:Node) REQUIRE n.neo4jImportId IS UNIQUE;

Note: You can find a list of Cypher queries we use for this tutorial here: <https://github.com/cj2001/nodes2021_kg_workshop/tree/main/cypher_queries>

After a successful query implementation, you should be able to see some node labels and relationship types including number of nodes and relations as seen in the image below. You can view this by expanding the left panel in your sandbox instance.

Graphical user interface, application

Description automatically generated

1. We now have a graph loaded into our neo4j instance. To view the graph and the nodes in it, run:

MATCH (n) RETURN n

This returns something that looks the image below.

Diagram

Description automatically generated with medium confidence

1. All the node labels that we see in the image above are of type node. We need to update node labels based on type\_ls list. To do that run this command.

MATCH (n:Node)

CALL apoc.create.addLabels(n, n.node\_labels)

YIELD node

RETURN node

Now the nodes will have different label associated with it. But some of the data is not accurately classified since the information/NLP is noisy. You should see something similar to this image.

Background pattern

Description automatically generated

1. To check for duplicates in the graph run this query:

MATCH (n:Node)

WHERE n.name CONTAINS 'obama'

RETURN DISTINCT n.name

This will return a list of nodes that contain the term ‘Obama’. It returns different distinct names but they all refer to the same individual.

Graphical user interface, text, application, email

Description automatically generated

1. A probable query to figure out Obama’s place of birth:

MATCH (n:Node {name: 'oh bah mə'})-[\*1..5]->(p)

WHERE p:Country OR p:AdministrativeArea OR p:Continent OR p:Place

RETURN n, p

Chart, diagram

Description automatically generated with medium confidence

This query checks nodes with the name property of Obama’s phonetic spelling (‘oh-bah-ma’) at 1-5 relationship hops away and returns the nodes if they have any of the properties listed in the where clause.

A more accurate query to find the exact place of birth would be something that uses the lemma ‘be’ or ‘bear’. Run this query:

MATCH (n:Node {name: 'oh bah mə'})-[:be|:bear]->(p)

WHERE p:Country OR p:AdministrativeArea OR p:Continent OR p:Place

RETURN n, p

Chart, bubble chart

Description automatically generated

1. To check for duplicates in our graph and see the number of duplicates, run:

MATCH (n:Node)

WITH n.name AS name, n.node\_labels AS labels, COLLECT(n) AS nodes

WHERE SIZE(nodes) > 1

RETURN [n in nodes | n.name] AS names, [n in nodes | n.node\_labels] as labels, SIZE(nodes)

ORDER BY SIZE(nodes) DESC

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

We find a lot of duplicates for Barack Obama.

1. To delete duplicate nodes, run this query:

MATCH (n:Node)

WITH n.name AS name, COLLECT(n) AS nodes

WHERE SIZE(nodes)>1

FOREACH (el in nodes | DETACH DELETE el)

Duplicates were collected in lists as shown in the previous query. The above query will delete duplicate elements and also their relationships.

Graphical user interface, text, application, email

Description automatically generated

1. To view some node (use Table form to see more details)

MATCH (n) RETURN n LIMIT 1

1. To check for cosine similarity between 2 nodes, run this query:

MATCH (n1:Node {name: 'barack obama'})

MATCH (n2:Node {name: 'mitch mcconnell'})

RETURN gds.alpha.similarity.cosine(n1.word\_vec, n2.word\_vec) AS similarity

But if running the above query gives you an error, run this query instead:

MATCH (n1:Node {name: 'barack obama'})

MATCH (n2:Node {name: 'mitch mcconnell'})

RETURN gds.similarity.cosine(n1.word\_vec, n2.word\_vec) AS similarity

Graphical user interface, text, application, email

Description automatically generated

1. To calculate Levenshtein distance, run this query:

MATCH (n1:Node {name: 'barack obama'})

MATCH (n2:Node) WHERE n2.name CONTAINS 'obama'

RETURN n2.name, apoc.text.distance(n1.name, n2.name) AS distance

A picture containing application

Description automatically generated

### Notebook 02

Purpose: Do data science on KG generated in Notebook 01.

**Notebook 02 is connected to the Sandbox instance created in the previous step. For steps on how and when to run this, look at the next step i.e, “During Notebook 02 execution”.**

You initially do some entity disambiguation with 2 different nodes. You want to check for overlap between those nodes – this lets you know how similar they might be. If a lot or most of the relationships in 1 node overlap with that of the second node, then it is a strong indicator that the 2 nodes might be the same.  
You will then connect to the graph and do some Machine Learning. The initial results will show you that you have a multi-class classification problem at hand, but since the graph is very small the results will not be good. Instead for simplicity, binary classification is done using the Support Vector Classifier from scikit-learn’s SVM. You can then check the accuracy of using word vectors associated with an entity type. This can be then compared to that of graph embeddings. Results for both are really close considering graph embeddings are just 10 dimensional vectors.

### During Notebook 02 execution

1. Open the 02-graph-data-science.ipynb notebook in Google Colab.

*Note: One way to do this is to open Google Colab at* [*https://colab.research.google.com/*](https://colab.research.google.com/)*. Select these menu options File > Open Notebook, and then select GitHub in the pop-up window. Copy and paste the GitHub repo URL address (*[*https://github.com/cj2001/nodes2021\_kg\_workshop*](https://github.com/cj2001/nodes2021_kg_workshop)*) in the space provided. Press enter. You should now see a list of all the ipynb files present in the repo. Since we are running the 02-graph-data-science.ipynb notebook, select that one. When you open it, if you get a warning “This notebook was not authored by Google”, click “Run anyway”.*

In your neo4j sandbox instance listing, you can use the drop-down arrow to expand the listing and see some more details. Under the ‘Connection details’ tab, note the values for ‘Bolt URL’ and ‘Password’. The value for ‘Username’ which is ‘neo4j’ should be the same in your ipynb file. Copy the value for ‘Bolt URL’ for the ‘uri’ variable, and the ‘Password’ value for the ‘pwd’ variable in the ipynb file.

The ‘uri’ and ‘pwd’ variables will be in the 3rd cell in the 02-graph\_data\_science.ipynb file.

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

1. If you see this error while importing in cell 2 of the Notebook  
   Graphical user interface, text, application, email

   Description automatically generated  
   Then replace plot\_confusion\_matrix with “ConfusionMatrixDisplay, confusion\_matrix”. Please refer to the screenshot below. You can find details about the error here: <https://stackoverflow.com/questions/63967530/importerror-cannot-import-name-plot-confusion-matrix-from-sklearn-metrics>  
   If you are using pip, you can run “pip install --upgrade scikit-learn” in your terminal.  
   Then re-run the second cell of the Notebook  
     
   Graphical user interface, text, application

   Description automatically generated
2. Execute all cells until you reach the ‘Observation’ header in the notebook.

Graphical user interface, text, application, email

Description automatically generated

1. If you execute the cell under the label “Now let’s connect to the graph and do some ML”, you will notice that in the results, that populate, the values under the “n2v\_all\_nodes” column is all 0.  
   Table

   Description automatically generated  
     
   Table

   Description automatically generated
2. In your sandbox instance, run this query (It’s made up of different commands and are separated by semi colons. You can run them together in one go!):

MATCH (n) WHERE ANY (x in n.node\_labels WHERE x IN ['Person', 'Organization', 'EducationalOrganization', 'Corporation', 'SportsTeam', 'SportsOrganization', 'GovernmentOrganization'])

SET n.pptu\_person = 1;

MATCH (n)

WHERE ANY (x in n.node\_labels WHERE x IN ['Place', 'AdministrativeArea', 'Country', 'Museum', 'TouristAttraction', 'CivicStructure', 'City', 'CollegeOrUniversity',

'MovieTheater', 'Continent', 'MusicVenue', 'LandmarksOrHistoricalBuildings', 'Cemetery', 'BodyOfWater',

'PlaceOfWorship', 'Restaurant', 'LakeBodyOfWater'])

SET n.pptu\_place = 1;

MATCH (n)

WHERE ANY (x in n.node\_labels WHERE x IN ['Thing', 'Periodical', 'Book', 'Movie', 'Event', 'MusicComposition', 'SoftwareApplication', 'ProductMode', 'DefenceEstablishment',

'MusicRecording', 'LocalBusiness', 'CreativeWork', 'Article', 'TVEpisode', 'ItemList', 'TVSeries', 'Airline',

'Product', 'VisualArtwork', 'VideoGame', 'Brand'])

SET n.pptu\_thing = 1;

MATCH (n)

WHERE n.pptu\_person IS NULL

AND n.pptu\_place IS NULL

AND n.pptu\_thing IS NULL

SET n.pptu\_unknown = 1;

Text

Description automatically generated with medium confidence

Then to create a new node label ‘Unknown’ based on .pptu\_unknown, run:

MATCH (n)

WHERE n.pptu\_unknown IS NOT NULL

CALL apoc.create.addLabels(n, ['Unknown'])

YIELD node

RETURN node

Graphical user interface, application

Description automatically generated

1. We will now create an in-memory GDS graph. The GDS library helps to create in-memory graphs or graph projections as efficient ways to store portions of your graph to perform calculations on. Run this query:

CALL gds.graph.create(

'all\_nodes',

{

AllNodes: {label: 'Node',

properties: {word\_vec\_embedding: {property: 'word\_vec'}}}

},

{

AllRels: {type: '\*', orientation: 'UNDIRECTED'}

}

)

YIELD graphName, nodeCount, relationshipCount

If this generates the following error.

Graphical user interface, text, application, chat or text message

Description automatically generated

Then run this query (if you see the above error):

CALL gds.graph.project(

'all\_nodes',

{

AllNodes: {label: 'Node',

properties: {word\_vec\_embedding: {property: 'word\_vec'}}}

},

{

AllRels: {type: '\*', orientation: 'UNDIRECTED'}

}

)

YIELD graphName, nodeCount, relationshipCount

Graphical user interface, text, application, email

Description automatically generated

1. Run node2vec on the graph to create a 10-dimensional graph embedding.

CALL gds.beta.node2vec.stream('all\_nodes', {embeddingDimension: 10})

YIELD nodeId, embedding

RETURN gds.util.asNode(nodeId).name as name, embedding

You should see something like this result!

Graphical user interface, text, application, email

Description automatically generated

1. Output node2vec to the nodes themselves.

CALL gds.beta.node2vec.write('all\_nodes',

{

embeddingDimension: 10,

writeProperty: 'n2v\_all\_nodes'

}

)

The output shows all the hyperparameters associated with node2vec. This is just using the default values, but you can tune them. A blog post (<https://medium.com/towards-data-science/making-fastrp-graph-embeddings-work-for-you-f7344a535dc3>) that details tuning the hyperparameters with a different embedding algorithm might be helpful!

Graphical user interface, text, application

Description automatically generated

1. Let’s check similarities based on node2vec embeddings. Note that this is graph similarity and not word vector similarity.

MATCH (n1:Node {name: 'oh bah mə'})

MATCH (n2:Node {name: 'michelle lavaughn robinson obama'})

RETURN gds.alpha.similarity.cosine(n1.n2v\_all\_nodes, n2.n2v\_all\_nodes) AS similarity

If this returns an error, run:

MATCH (n1:Node {name: 'oh bah mə'})

MATCH (n2:Node {name: 'michelle lavaughn robinson obama'})

RETURN gds.similarity.cosine(n1.n2v\_all\_nodes, n2.n2v\_all\_nodes) AS similarity

Graphical user interface, text, application, email

Description automatically generated

MATCH (n1:Node {name: 'oh bah mə'})

MATCH (n2:Node {name: 'mitch mcconnell'})

RETURN gds.similarity.cosine(n1.n2v\_all\_nodes, n2.n2v\_all\_nodes) AS similarity

Graphical user interface, text, application, email

Description automatically generated

MATCH (n1:Node {name: 'president barack obama'})

MATCH (n2:Node {name: 'mitch mcconnell'})

RETURN gds.similarity.cosine(n1.n2v\_all\_nodes, n2.n2v\_all\_nodes) AS similarity

Graphical user interface, text, application, email

Description automatically generated

1. Then execute all the cells after ‘Observation’ in the ipynb file. When you execute the first cell under observation, you’ll now see that the “n2v\_all\_nodes” column has non zero values under it.

Table

Description automatically generated  
  
You might run into some more errors here:  
Graphical user interface, text, application, email

Description automatically generated

Please change the ‘modeler’ function in the notebook to the changes you see in the following screenshot:  
   
 Graphical user interface, text, application, email

Description automatically generated

Method 2: NLP Lite Approach:  
Wikidata uses the terms P-value and Q-value. A Q-value stands for ‘query’ and it uniquely identifies an item/entity. A P-value stands for ‘propery’ and it identifies the property/attribute or characteristic of an item/entity. P-values and Q-values are together used for representing items and their properties and to organize and link data in a structured manner.

Given a set of items and their properties, this method queries Wikidata. The Q-values are used for creating entities i.e., subjects and objects and the P-values are used for verb creation. Unlike the Method 1 approach, in this approach Wikidata will handle entity disambiguation but you need to supply the list of verbs that you want to focus on. You will still need to do some NLP in this approach. Most of the named entity resolution (NER, also known as NED (Named Entity Disambiguation)– this is a subtask of Named Entity Recognition) on the text is done by Spacy. This is a cleaner approach.

1. Notebook 03  
   Purpose: Put together a knowledge graph  
   The major difference in this Notebook, is scraping the P-values and Q-values from Wikidata. Relevant P-values that you need must be manually selected and put together. The data is cleaned and processed and then the nodes and edges (the new knowledge graph) are put into new instance of graph data science sandbox.   
   Please run Notebook 03 in the following way.
2. Run 03-wikidata\_kg.ipynb in Jupyter Notebook instead of in Google Colab. Install dependencies like neo4j, Wikipedia using pip. Install pywikibot. Open a terminal and use this to install pywikibot.  
   Navigate to the folder with the 03-wikidata\_kg.ipynb script and run the following.

pip install requests

git clone https://gerrit.wikimedia.org/r/pywikibot/core.git

cd core

git submodule update --init

python pwb.py script\_name

Graphical user interface, text

Description automatically generated

**To run in Google Colab:**

Or else, you can use the user-config.py script generated from the above scripts. Upload this script to the /content folder in Google Colab and you can run the notebook in Google Colab as well.

Graphical user interface, text, application, email

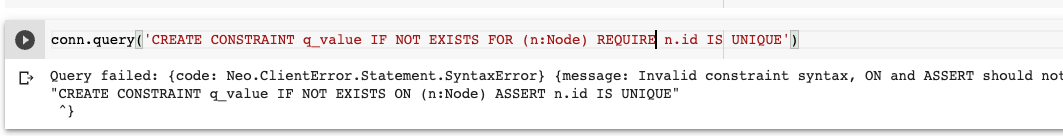
Description automatically generated

Change the code for spacy.displacy to what you see in the screenshot below.

Graphical user interface, text, application, chat or text message

Description automatically generated

Also change the Notebook code based on the given screenshot below.



1. Follow the instructions in <https://heardlibrary.github.io/digital-scholarship/host/wikidata/bot/> to get a token for Pywikibot to scrape data from Wikidata.

You’ll have to save a couple of files – the credentials.txt file and the write-statements.py file from the given link.

Create your Wikidata login and use the credentials.txt file and the write-statements.py file to get your token (it will be printed in the terminal when you run write-statements.py).

1. Execute the cells in the 03-wikidata\_kg.ipynb file.

### After running Notebook 03

For the Notebook take the values for the ‘url’ and ‘pwd’ from ‘Connection details’ under ‘Blank sandbox – graph data science’

1. Use this link <https://sandbox.neo4j.com/?usecase=graph-data-science-blank-sandbox> to create a new graph data science blank sandbox.

Graphical user interface, text, application

Description automatically generated

1. Run the following query into your new browser window for the new graph data science sandbox. This will import a Wikidata graph from a json file on GitHub.

CALL apoc.import.json("https://raw.githubusercontent.com/cj2001/nodes2021\_kg\_workshop/main/json\_files/wiki.json")

If this gives an error, first run:

CREATE CONSTRAINT FOR (n:Node) REQUIRE n.neo4jImportId IS UNIQUE;

And then run:

CALL apoc.import.json("https://raw.githubusercontent.com/cj2001/nodes2021\_kg\_workshop/main/json\_files/wiki.json")

Graphical user interface, text, application, email

Description automatically generated

1. Remove duplicates using this query:

MATCH (n:Node)

WITH n.name AS name, COLLECT(n) AS nodes

WHERE SIZE(nodes)>1

FOREACH (el in nodes | DETACH DELETE el)

Graphical user interface, text

Description automatically generated

1. Convert string labels to list, using this query:

MATCH (n:Node)

SET n.type\_ls = split(n.type, ",")

Graphical user interface, application, Word

Description automatically generated

1. Update node labels, using this query:

MATCH (n:Node)

CALL apoc.create.addLabels(n, n.type\_ls)

YIELD node RETURN node

A picture containing graphical user interface

Description automatically generated

1. Where was Obama born? Find out using this query:

MATCH (n:Node {name: 'Barack Obama'})-[:place\_of\_birth]->(place:Node) RETURN place.name

Graphical user interface, text, application, email

Description automatically generated

1. Let’s create a new label. We are going to do place vs. not place. After creating a list of things that could be places, we set up a property called ‘is\_place’ and set it to 1.

MATCH (n)

WHERE ANY (x in n.type WHERE x IN

['county of Illinois',

'state of the United States',

'oblast of Russian',

'province of Afghanistan',

'province of Iran',

'oblast of Ukraine',

'district of Libya',

'governorate of Iraq',

'province of Cuba',

'governorate of Syria',

'sovereign state',

'autonomous okrug of Russia',

'city',

'krai of Russia',

'city of the United States',

'territory of the United States',

'capital',

'geographic region',

'continent',

'county of Hawaii',

'village',

'historical country',

'autonomous republic',

'organized incorporated territory',

'unincorporated territory',

'census-designated place',

'human settlement',

'borough of New York City',

'Commonwealth realm',

'city of Pennyslvania',

'neighborhood of Washington, D.C.',

'country']

)

SET n.is\_place=1;

Graphical user interface, text, application

Description automatically generated

1. Set the label for non-places as 0 using this query:

MATCH (n) WHERE n.is\_place IS NULL SET n.is\_place=0

Graphical user interface, text, application

Description automatically generated

1. Count the number of places and non-places using this query:

MATCH (n {is\_place: 1}) RETURN COUNT(n)

MATCH (n {is\_place: 0}) RETURN COUNT(n)

1. Create in memory graph, using this Cypher query:

CALL gds.graph.project(

'all\_nodes',

'Node',

{

RELS: {

type: '\*',

orientation: 'UNDIRECTED'

}

}

)

Graphical user interface, text, application, Teams

Description automatically generated

1. Create n2v embedding:

CALL gds.beta.node2vec.write('all\_nodes',

{

embeddingDimension: 10,

writeProperty: 'n2v\_all\_nodes'

}

)

Graphical user interface, text, application

Description automatically generated

1. Create FastRP embeddings without any tunings on the embeddings:

CALL gds.fastRP.write(

'all\_nodes',

{embeddingDimension: 10, writeProperty: 'frp\_all\_nodes'}

)

Graphical user interface, text, application

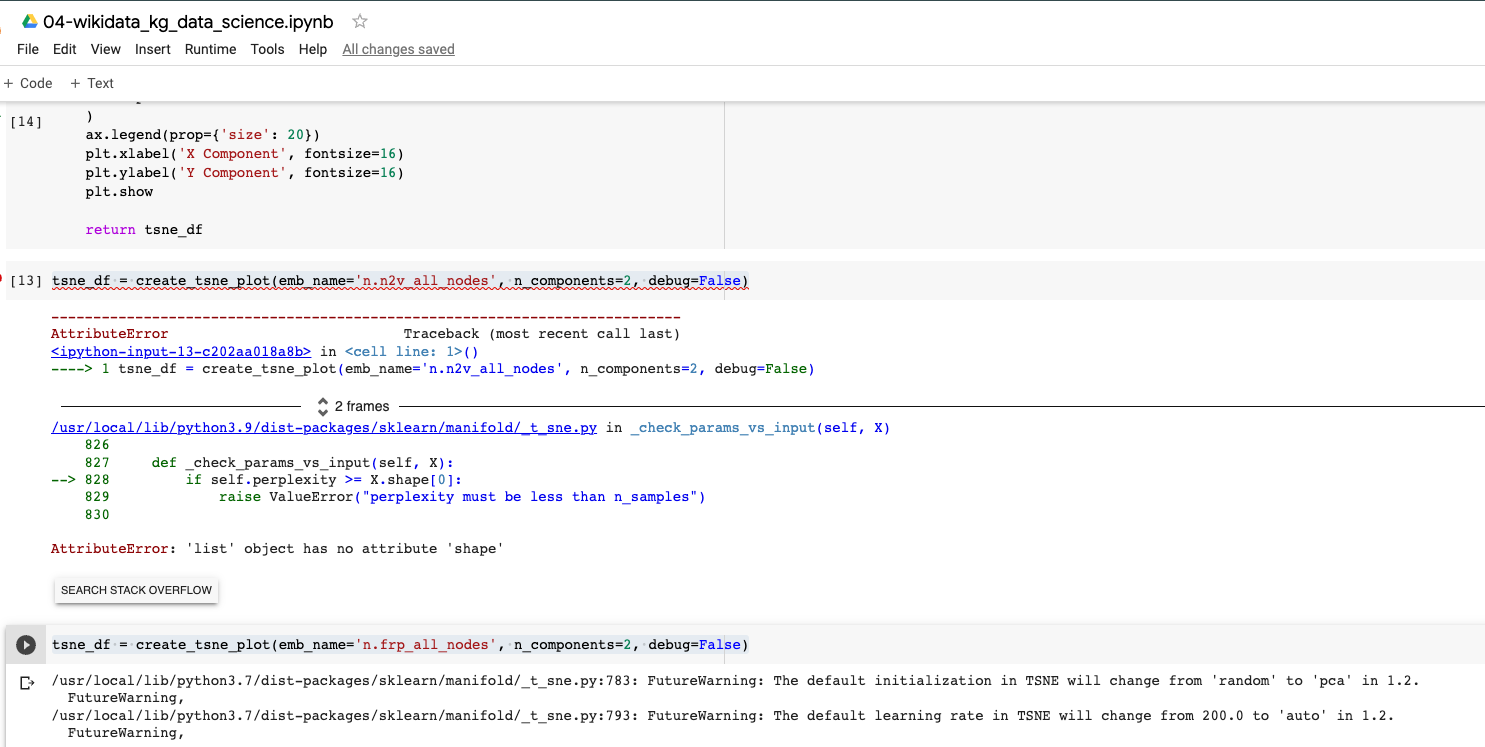
Description automatically generated

1. Now we run 04-wikidata\_kg\_data\_science.ipynb. Notebook 03 was how the graph was populated. Notebook 04 is how you do ‘fun stuff’ with the graph.

For the Notebook take the values for the ‘url’ and ‘pwd’ from ‘Connection details’ under ‘Blank sandbox – graph data science’

### Notebook 04

Purpose: Do data science on KG generated in Notebook 03.  
Connect to the graph data science sandbox instance that received the graph file output from Notebook 03. We visualize t-SNE embeddings for node2vec and then for fastRP. The visualizations don’t look great because of the size of the graph. We then do some binary classification.

While executing Notebook 04, when you the following error:  
  


Change the create\_tsne\_plot() function to what you see in the screenshot below.

Graphical user interface, text, application, email

Description automatically generated

You will mainly change the code from

X\_emb **=** TSNE(n\_components**=**n\_components)**.**fit\_transform(list(model\_df['vec']))

To

X\_emb = TSNE(n\_components=n\_components, learning\_rate='auto', init='random').fit\_transform(np.array(list(model\_df['vec'])))

Change the modeler() function to what you see in the screenshot below:  
Graphical user interface, text, application, email

Description automatically generated