

How to Use Chat-GPT and Python to Build a Knowledge Graph in Neo4j Based on Your Own Articles

A graph containing structured knowledge from more than 120 articles on mathematics and data science



tds

[Kasper Müller](#)

.

Published in

[Towards Data Science](#)

.

8 min read

.

Aug 27



In this article, I will show how you can structure and explore the content of your own articles using graph technology and some programming.

The idea of using NLP techniques for structuring unstructured data is not new, however, the latest progress in LLMs (Large Language Models) has sparked countless opportunities for doing just that. The accessibility for amateurs through the booming technology Chat-GPT has created a lot of attention towards LLMs and generator models.

In fact, generative AI is on the agenda in many companies already!

The way we will work with the technology in this article is through the programming language Python using OpenAI's developer API. We will work on data from Medium (meta huh?) and build a knowledge graph. That may sound like a mouthful, but it is actually surprisingly easy to get started with.

Getting started

First things first. The plan of attack is the following.

1. Get the API to work and access it through Python.
2. Use a sample text to do prompt engineering ensuring that the GPT-4 model understands what you want from it.
3. Download your articles from Medium (you can of course use other pieces of text if you want) and pre-process the data.
4. Extract and collect output from Chat-GPT.

5. Post-process the output from Chat-GPT
6. Write code to structure the data further into a graph using the Cypher query language.
7. Play around with your new best friend and explore your articles.

Without further ado, let's get started by quickly setting up the basic tech.

Setup


We need to have the programming language Python and the graph database Neo4j installed on our local computer.

The first thing to do is to ensure that you have a plus account at OpenAI so that you can use GPT-4. The second thing you should make sure of is that you have [signed up](#) for the API use. Once that is in place, you need to generate an [API key](#). Then you need to *pip install openai*.

Before connecting to ChatGPT, let's go to the browser and try to find the right way to ask about this task. This is called prompt engineering and it is very important to get right. By trying out different ways to ask using a random article of mine as an example, I found that the right way to ask was to provide a detailed and guided prescript before giving it the actual text.

I ended up with the following prescript:

KA

You are a mathematician and a scientist helping us extract relevant information from articles about mathematics. 

The task is to extract as many relevant relationships between entities to mathematics, physics, or history and science in general as possible.

The entities should include all persons, mathematical entities, locations etc.

Specifically, the only entity tags you may use are:

Mathematical entity, Person, Location, Animal, Activity, Programming language, Equation, Date, Shape, Property, Mathematical expression, Profession, Time period, Mathematical subject, Mathematical concept, Discipline, Mathematical theorem, Physical entity, Physics subject, Physics.

The only relationships you may use are:

IS, ARE, WAS, EQUIVALENT_TO, CONTAINS, PROPOSED, PARTICIPATED_IN, SOLVED, RELATED_TO, CORRESPONDS_TO, HAS_PROPERTY, REPRESENTS, IS_USED_IN, DISCOVERED, FOUND, IS_SOLUTION_TO, PROVED, LIVED_IN, LIKED, BORN_IN, CONTRIBUTED_TO, IMPLIES, DESCRIBES, DEVELOPED, HAS_PROPERTY, USED_FOR

As an example, if the text is "Euler was located in Sankt Petersburg in the 17 hundreds", the output should have the following format: Euler: Person, LIVED_IN, Skt. Petersburg: Location

If we have "In 1859, Riemann proved Theorem A", then as an output you should return Riemann: Person, PROVED, Theorem A: Mathematical theorem

I am only interested in the relationships in the above format and you can only use what you find in the text provided. Also, you should not provide relationships already found and you should choose less than 100 relationships and the most important ones.

You should only take the most important relationships as the aim is to build a knowledge graph. Rather a few but contextual meaningful than many nonsensical.

Moreover, you should only tag entities with one of the allowed tags if it truly fits that category and I am only interested in general entities such as "Shape HAS Area" rather than "Shape HAS Area 1".

The input text is the following:

 Comments

As an example, I gave it a snippet from the article about the Gamma function that I wrote a long time ago:

In the late 1720s, Leonhard Euler was thinking about how to extend the factorial to non-integer values. This was the start of a rich theory used all over the scientific world. A theory of one of the most important functions in mathematics. Leonhard Euler is, without doubt, one of the greatest mathematicians in history. To give you an idea of Euler's powers here follows some examples that show his brilliance. First of all, Euler had an outstanding memory! He was able to recite Virgil's Aeneid from beginning to end, detailing in what line every page of the edition he owned began and ended. To give you some context, the Aeneid comprises a total of 9,896 lines! Euler was also extremely productive. He produced about 30,000 pages in his lifetime and it is estimated that he accounted for about a third of all published scientific papers in the 18th century!!! He even published papers after he died! Many of those pages were written while he was blind, and for that reason, Euler has been called the Beethoven of mathematics. Beethoven could not hear his music. Likewise, Euler could not see his calculations. Actually, Euler was quite optimistic about the loss of his vision. He is known to have said something like: "In this way I will have fewer distractions". One should think that this would slow him down, but in fact, when he became blind, his production went up! Euler also had phenomenal computational powers. On one occasion, two students disagreed over the result of the sum of 17 terms in a series because their results differed in the fifth decimal place. Euler computed the correct result in his mind in a few seconds. This anecdote was referred to by his colleague Nicolas de Condorcet who at Euler's death wrote a lengthy eulogy in which he declares that Euler is "one of the greatest and most extraordinary men that nature has ever produced". So Euler was a great mathematician, to say the least, and he was thinking about how to extend the factorial function. I will show you what he came up with and the surprising properties that followed. Later in the article, I'll reveal how we would give meaning to $1/2!$ and what the value of this symbol is.

What it came up with was the following:



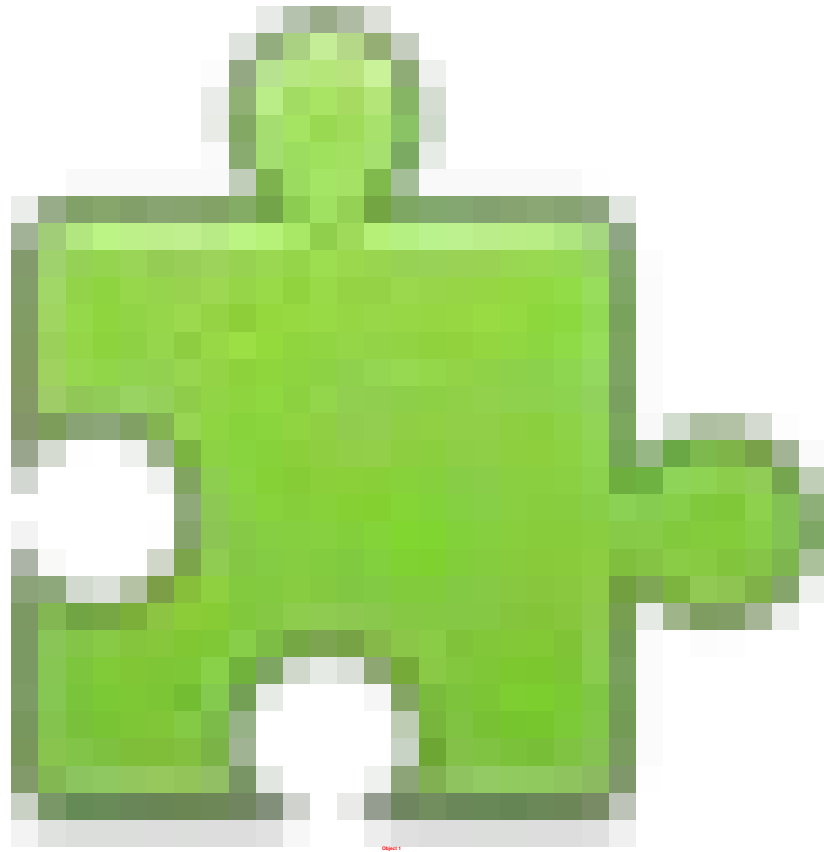
Here are the relationships extracted from the text:

1. Leonhard Euler: Person, WAS, thinking: Activity
2. Leonhard Euler: Person, CONTRIBUTED_TO, theory: Mathematical concept
3. Leonhard Euler: Person, IS, mathematician: Profession
4. Euler: Person, HAD, memory: Property
5. Euler: Person, WAS, productive: Activity
6. Euler: Person, PRODUCED, papers: Physical entity
7. Euler: Person, IS, Beethoven of mathematics: Profession
8. Beethoven: Person, COULD_NOT, hear music: Activity
9. Euler: Person, COULD_NOT, see calculations: Activity
10. Euler: Person, HAD, computational powers: Property
11. students: Person, DISAGREED, result: Mathematical entity
12. Euler: Person, COMPUTED, result: Mathematical entity
13. Nicolas de Condorcet: Person, WROTE, eulogy: Physical entity
14. Nicolas de Condorcet: Person, DECLARED, Euler: Person
15. Euler: Person, WAS, great mathematician: Profession
16. Euler: Person, WAS, thinking: Activity
17. factorial function: Mathematical entity, HAS_PROPERTY, extendability: Property

I've tried to adhere to the given constraints and only included relationships that seem relevant and general enough for building a knowledge graph.

Even though it clearly didn't really understand the task, it did okay, especially with the format. However, sometimes it creates duplicates, and note that it hallucinated some entities and relationships even though we asked it not to. Annoying disobedient machine! We will deal with this later.

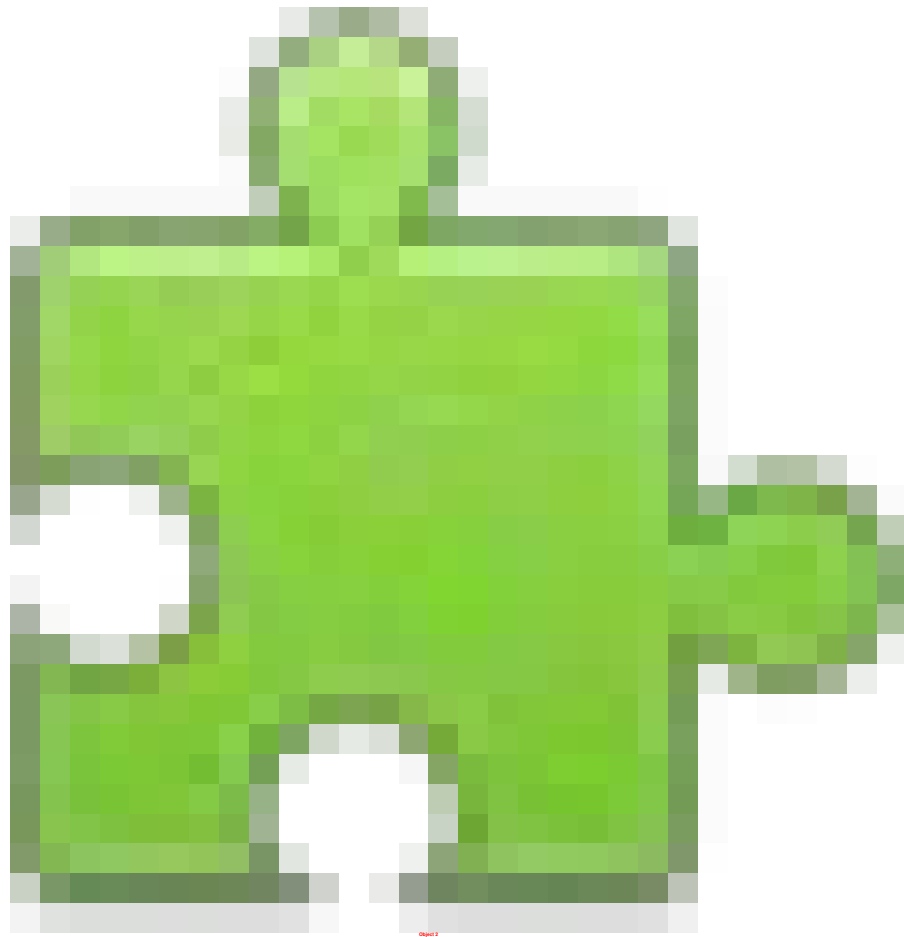
For future uses, we will store this prescript in a Python file called *prompt_input.py*.



Now that the basic setup is in place, let's test if it actually works.

If the code is only for you and only on your local machine, you can hardcode the API key in the Python file, otherwise you can set it as an environment variable or place it in a config file that you don't push anywhere!

Let's test if this setup works. We create a file called *connect.py* containing the basic connection to ChatGPT from Python.



We verify that this works!

Data

I need to fetch articles from my Medium account. At the time of writing, I have published 123 articles, but the download feature from Medium returns 259 files! This is because it classifies comments and drafts as posts too. We only want the published articles, but that is not the only problem. The files are HTML files! That is of course great if you want to read them in a browser, but not if you want to work with the pure text.

Well, nice try, Medium, but that can't stop a data scientist armed with programming languages and dirty tricks!

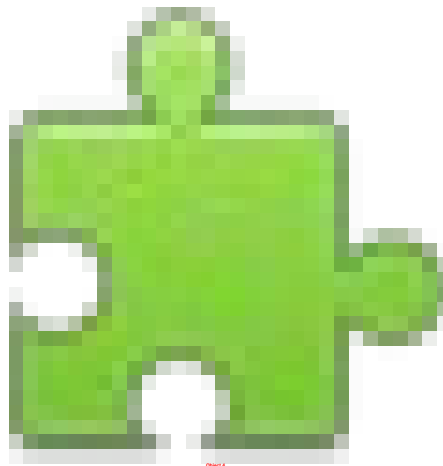
We also note that the file names of the downloaded files are quite messy. A standard name is for example “2020-12-11_The-Most-Beautiful-Equation-in-the-World-5ab6e49c363.html”

Let's store these files in a folder called *raw*.

We write a small module called *extract_text_from_html.py* with some functionality to extract the text from these files:



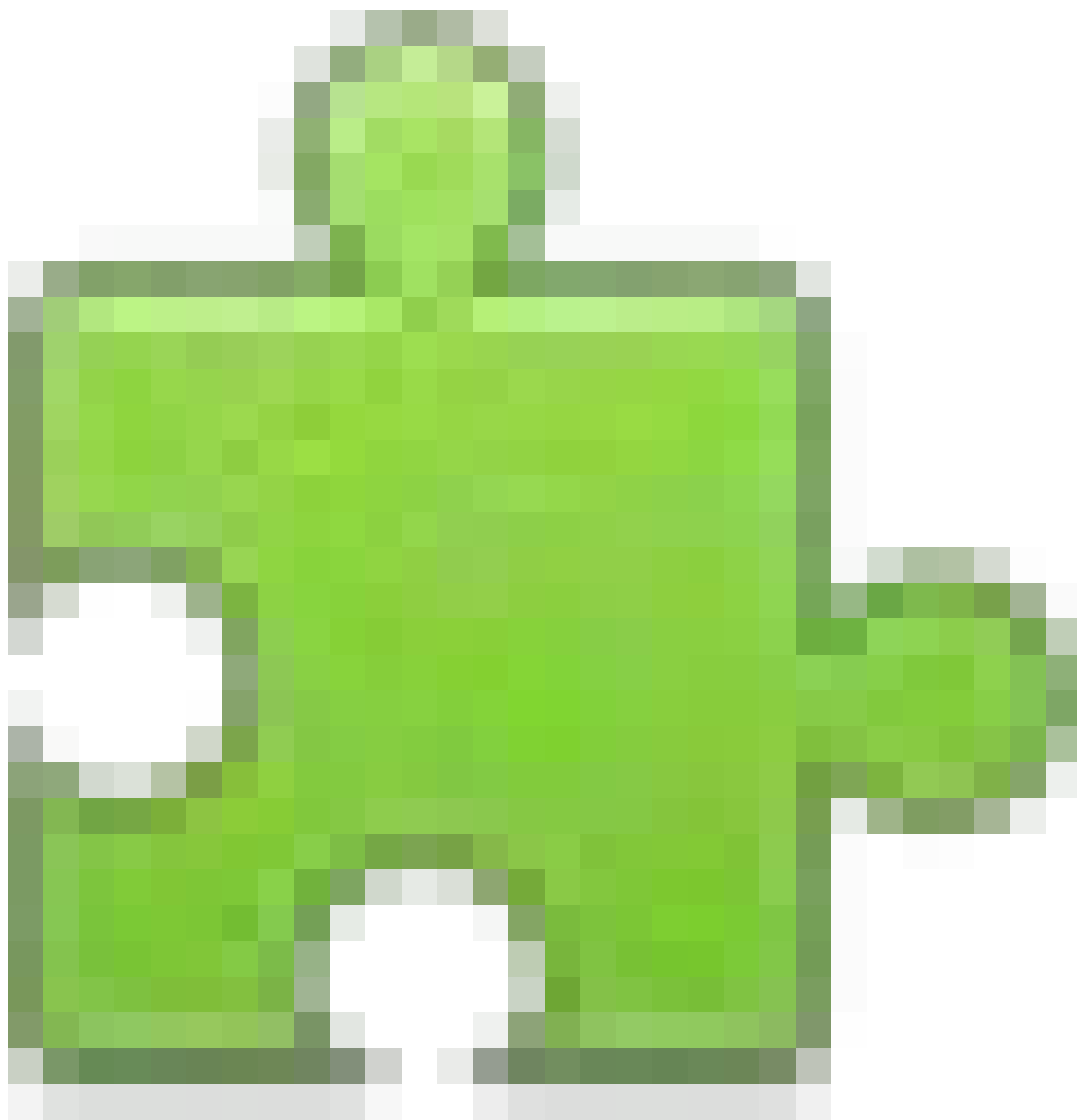
Before we can use it to actually get results from ChatGPT, we need to be able to split the text up into batches. The reason is that GPT-4 has a token limit. Luckily, this is easy. In a file called *preprocess.py*, we write:



Now we are ready to actually get some data from ChatGPT.

We write a file called *process_articles.py* where **loop** through the articles, retrieve the **titles** from the frightening file names, extract the actual **text** from the HTML files, run each batch of text through **ChatGPT**, **collect** the results from the files, and **save** the outputs from the model in new files that we store in a folder called *data*. We also save the actual texts in a folder called **cleaned** for later use.

Phew!! that was a lot. But actually, the code is simple because we have already done some of the work in other files.



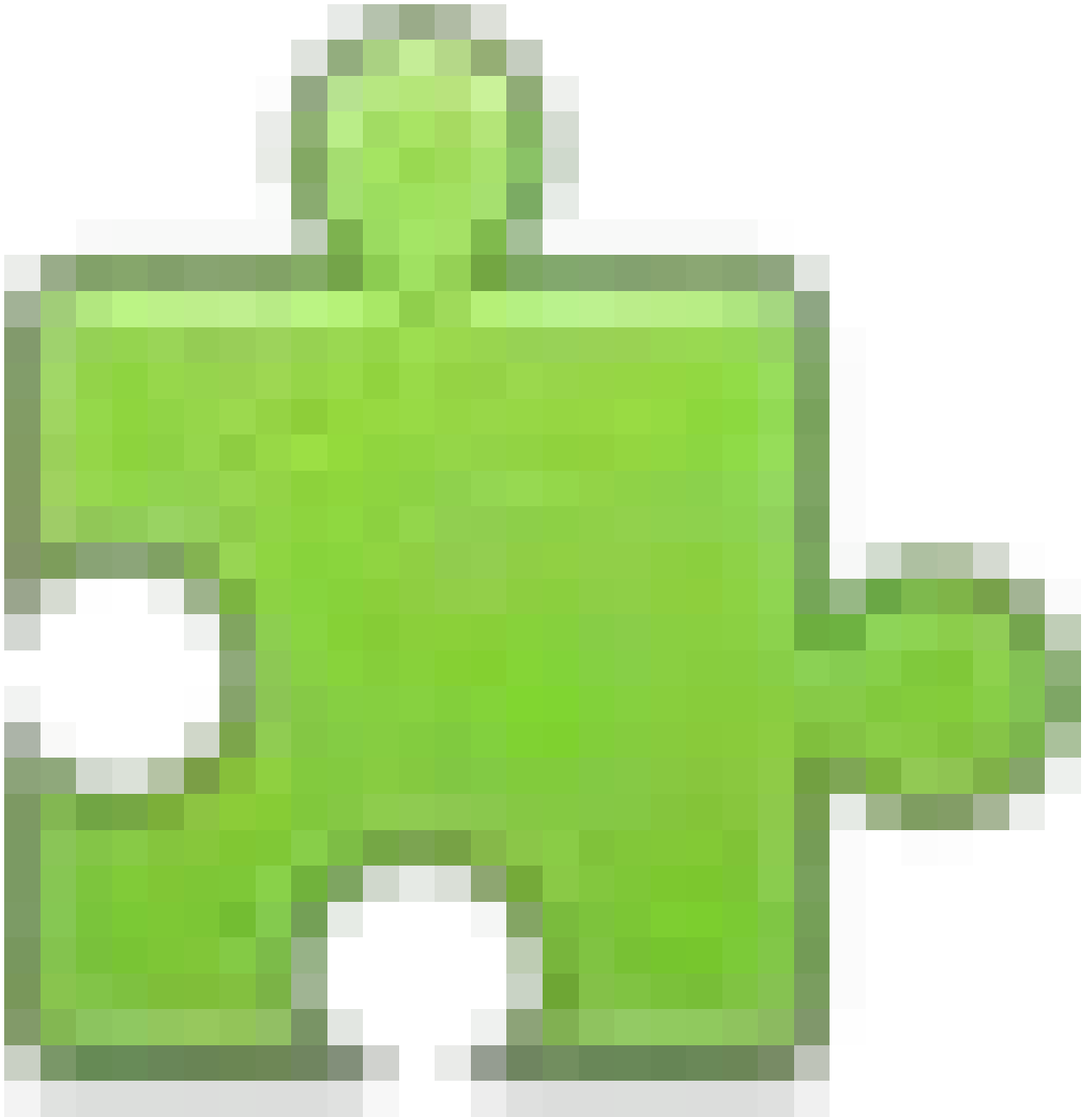
The above code might take a while to execute as the GPT-4 model is relatively slow compared to the other sub-performing models available. We make sure to use a caching setup so that if the program crashes, then we don't start all over, we just start where we left off.

Now (several hours in agony later) we have a structured dataset of results from GPT-4. Perfect. Now we “just” need to build a graph from it.

Building the knowledge graph

We will merge the preprocess and graph creation process into one single function. This is normally not very advisable (separation of concerns and all), but because we need to look at a “relationship and entity” — level in the preprocessing anyway, we might as well create the nodes and relationships in the graph while we have our hands dirty.

Let us create a small API containing a driver so we can talk to our graph.



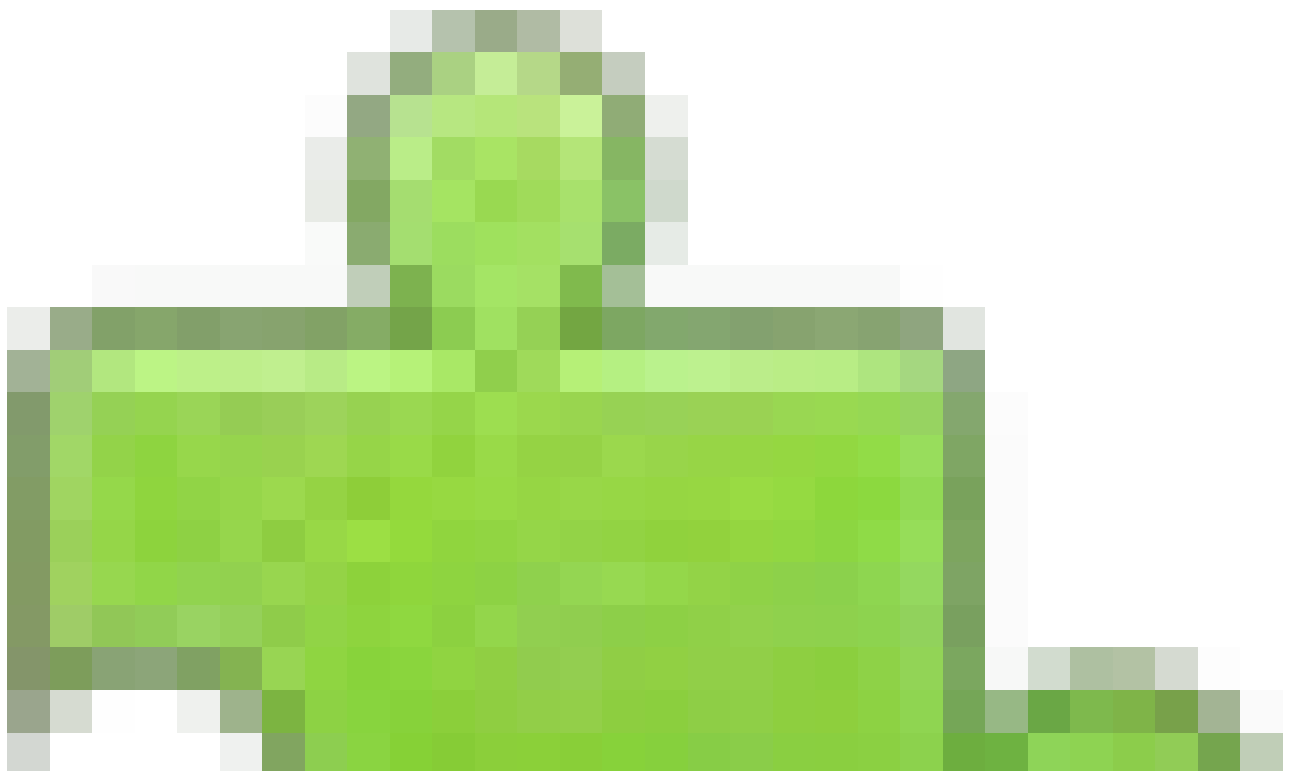
Open

We need to **loop** over the results, make sure that the entities are **not** too long, **clean** the results, and define the **nodes** and **relationships** in the output from the gpt-model, we **don't** want to call the graph with the same query multiple times, we want the entities to be connected to the original

articles, and then we need to make sure that each entity and relationship cooked up by Chat-GPT is actually **in** the text so we don't build a graph of **machine dreams**!

The last of the above requirements is to raise the probability that we can trust the graph even though it is not fool proof if you think about it.

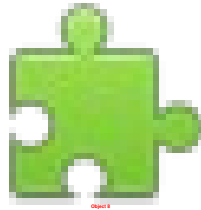
No biggie! We write the following:



There are of course many ways to create a schema for a knowledge graph. It is not always easy to see what should be nodes and what should be relationships but since we don't want relationships between relationships, we went for the above.

Moreover, I chose a minimalistic approach for this article. Normally, we would enrich the nodes and relationships with more properties.

Now we just need a main point of entry.



That is it. Now we have ourselves a knowledge graph containing information from my articles on Medium. In fact, we have about 2000 nodes and 4500 relationships.

Exploring the graph

So what can we do with this thing? What should we ask of it?

Let's try to find out how many articles the different persons were found in. We have the following:

```
neo4j$ MATCH (p:Person)->(a:Article) RETURN p.name as Name, COUNT(a) as Articles ORDER BY Articles DESC
```

	Name	Articles
1	"Euler"	25
2	"Riemann"	11
3	"Leibniz"	7
4	"Gauss"	6
5	"Euclid"	6
6	"Ramanujan"	4
7	"Newton"	4
8	"Lagrange"	3
9	"Dirichlet"	3
10	"Einstein"	3
11	"Hilbert"	3
12	"Fourier"	3
13	"Archimedes"	3
14	"Cantor"	3

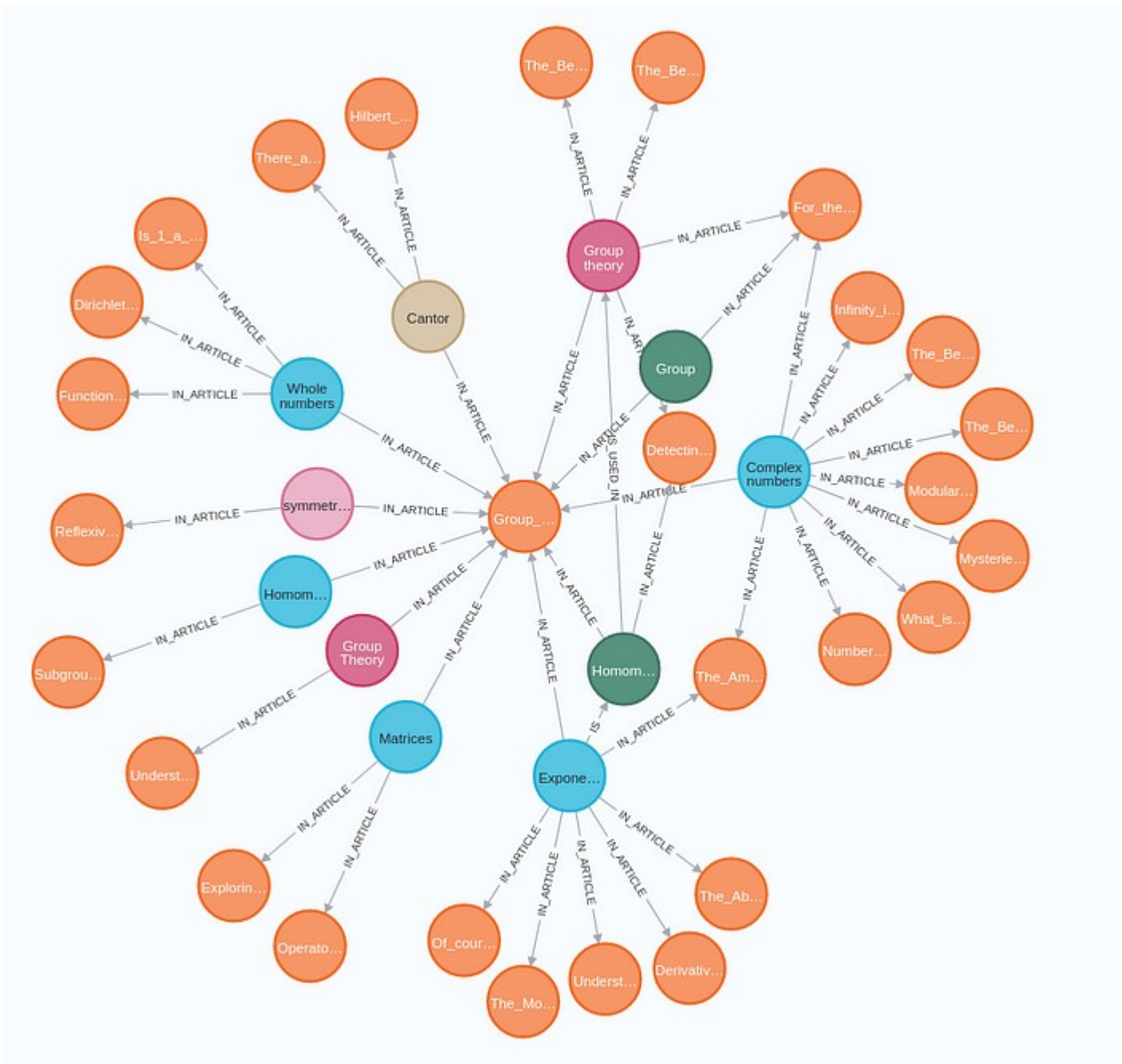
Not surprisingly, Euler tops the list but I am surprised that Ramanujan and Newton were found in 4 of my articles. I could of course find the titles of the articles if I wanted but let's move on. Okay, so this was fun but you don't need a graph to figure this out.

Let's try something else. Let's see how many articles mention both Riemann and Euler.

```
neo4j$ MATCH (:Person {name: 'Euler'})→(a:Article)←(:Person {name: 'Riemann'}) RETURN a.name
```

	a.name
1	"Beyond_Infinity"
2	"Primes_in_Arithmetic_Progressions____The_Genius_of_Dirichlet"
3	"Mysteries_of_Infinity"
4	"The_Hunt_for_a_Number"
5	"The_Battle_of_the_Cubic_Equation_and_the_Dawn_of_the_Complex_Numbers"
6	"The_Riemann_Hypothesis_Made_Real"
7	"Infinity_in_Numbers____Leonhard_Euler"

Let's see how many of Euler's discoveries my articles mention.



The result is displayed here as 27 other articles connected by the non-orange nodes in the above image. Even though this is merely a toy example, one could imagine how this could just as well show how business-related documents are related by various sensitive keywords important within some disciplines such as GDPR or Audit.

Takeaways

Obviously, this work should be seen as what we call a “proof of concept”. We can’t use my articles for anything really, but if this had been texts from a company containing information about their customers and employees from emails to word files, pdfs, and so on, this could be used to map out how customers are related and which employees work closely together.

This in turn would give us a 360 view of how data flows through the organization as a whole, who is the most important person for a specific type of information flow, who is the right one to reach

out to if you want to know about a specific topic or document, the client we contacted in our department was once contacted by another department, etc.

Extremely valuable information. Of course, we can't use ChatGPT for this because we don't know what happens to the data we send it. Therefore it is not a good idea to ask it about sensitive or business-critical information. What we need to do is to download another LLM (large language model) that only lives on our laptop. A local LLM. We can even fine-tune it on our own data. This is done as we speak by many companies already for building chatbots, assistants, and so on.

But using it to build a knowledge graph over your unstructured data is next-level if you ask me and I think I have shown that it is more than doable!

If your company wants to know how we can use your data to weave a spiderweb of possibilities for your business, then please reach out to [me](#) or my colleague [Kenneth Nielsen](#).

Thank you for reading.

If you like to read articles like this one on Medium, you can [get a membership](#) for full access. To join the community, simply click [here](#).

[Mathematics](#)

[Python](#)

[ChatGPT](#)

[Graph Database](#)

[Machine Learning](#)



Written by Kasper Müller

[44K Followers](#)

·Writer for

[Towards Data Science](#)

Mathematician, programmer and writer interested in the mysteries of the Universe, fascinated by the human mind, music and things that I don't understand.