Analyze Your Website with NLP and Knowledge Graphs

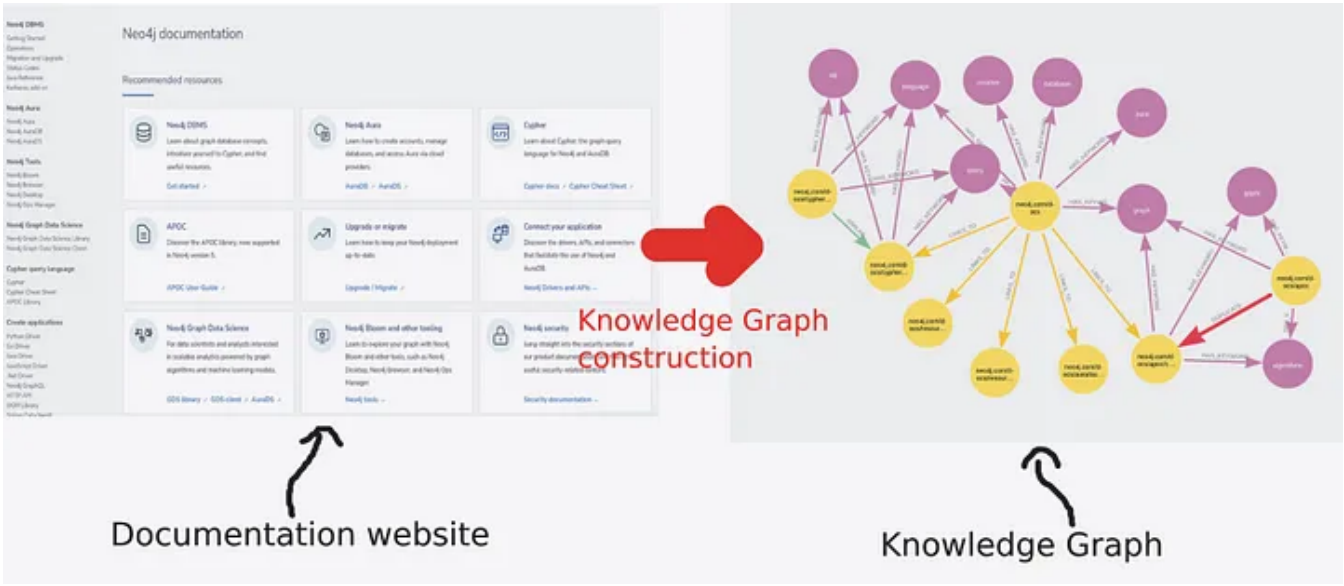
Combine various NLP techniques to construct a knowledge graph representing your website

Tomaz Bratanic · Following
Published in Towards Data Science
14 min read · Jan 5

Listen Share More

A website is a reflection of the company. For the most part, it is used to inform users about various products and services and drive sales. However, the website grows and changes over time and many minor and major changes are introduced. As a result, it is not uncommon to end up with a disorganized website that fails to accomplish its original mission. Therefore, it makes sense to regularly evaluate the structure and content of the website to make it as optimized as possible. Optimizing websites is a huge business, and consequently, there are multiple commercial tools to help you with SEO and other suggestions. However, I will show you how you can create a comprehensive and detailed representation of the content on your website with a little bit of coding knowledge, which will allow you to analyze and improve it.

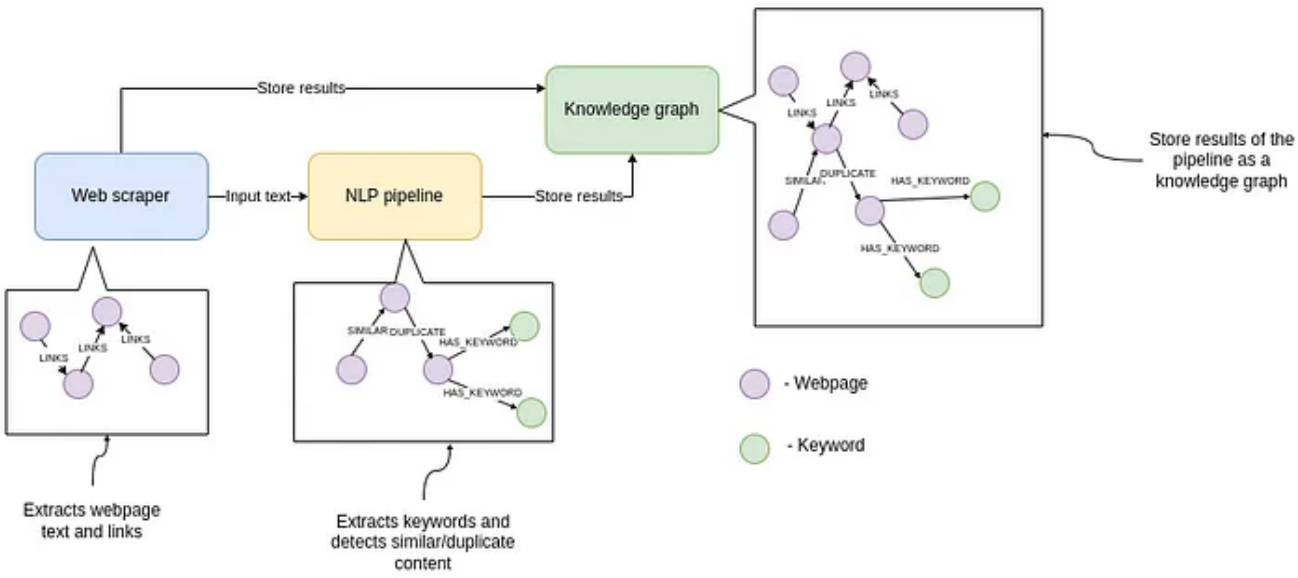
You can extract the structure of the website using any of the available web scrapers. Additionally, it makes sense to not only evaluate the structure but also the content of the website by utilizing various natural language processing techniques. Since most websites are copyrighted, I have decided to use the Neo4j documentation website as an example in this tutorial. The content of the documentation website is available under the CC 4.0 license. However, you can apply a similar workflow to any web page you desire.



Extracting information from documentation to construct a knowledge graph. Image by the author.

It might seem a bit magical (if you ignore my arrows) how you might construct a knowledge graph using information from your website. Throughout this post, I aim to bring more clarity to information extraction and provide you with tools you can use on your own. I have used similar approaches with medical documents, news, or even crypto reports, and now we'll analyze a website with the help of NLP and knowledge graphs.

Data collection and modeling workflow



Data collection and modeling workflow. Image by the author.

The data collection and preprocessing consist of three parts.

- Web scraper: A Python script that walks through the documentation web page and collects links and text
- NLP pipeline: Extracts keywords from text and calculate text embeddings to detect similar/duplicate content
- Knowledge graph: Store results as a knowledge graph for further analysis

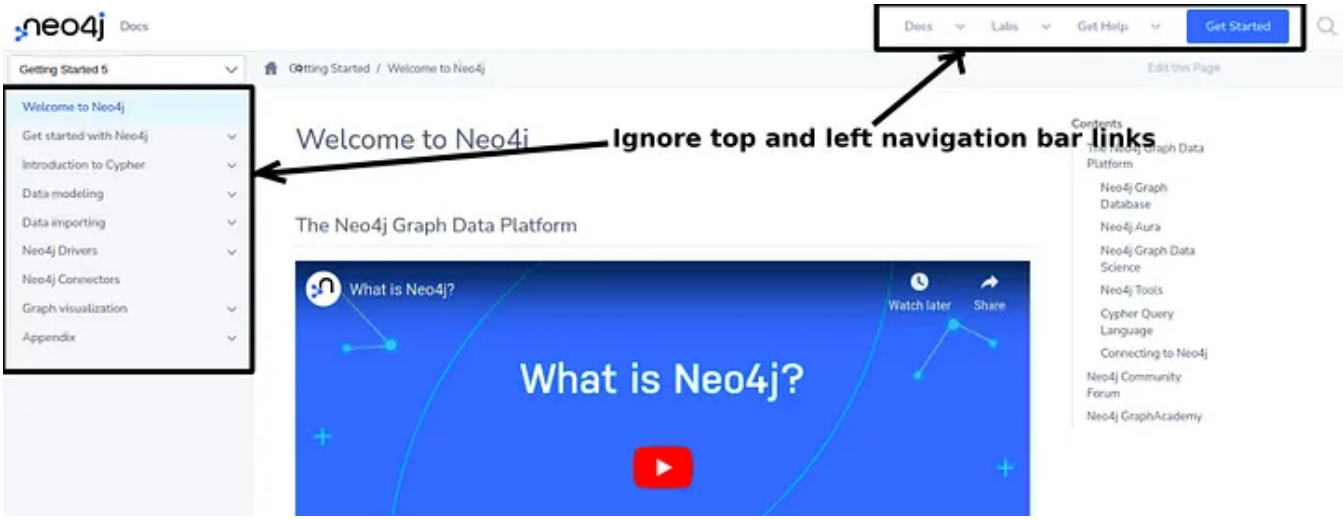
The code for the data collection and preprocessing is available on GitHub as a Jupyter notebook.

You don't have to run the data collection and processing yourself since it takes a couple of hours. I have prepared a Neo4j dump that you can use if you want to follow along with the analysis later in the post.

Web scraper

I usually use Python Selenium for web scraping, but you can use any of other libraries or languages you want to extract relevant information from websites. I won't go into too many details about the code, as the goal of this post is not to teach you how to scrape websites. However, you can examine the Jupyter notebook that handles the web scraping.

Specifically for the Neo4j documentation website, I avoided scraping the links from the left and top navigation bars as that would introduce much noise in the graph since most of the pages have the same navigation bars present.



Links from navigation bars are ignored during scraping. Image by the author.

With the Neo4j documentation website, I wanted to capture how a user could traverse the documentation without using the navigation bar. Otherwise, we would introduce noise in the knowledge graph as all the pages would be linking to the same pages in the navigation bars. Additionally, I have focused on extracting text and links from only documentation web pages, so some product or marketing pages were not scraped for their content.

Natural language processing

The natural language processing step involves extracting keywords and calculating text embeddings to detect similar and duplicate content. Before even considering training your own NLP model, it is always a good idea to check the HuggingFace model repository and see if any publically available models are a good fit for your use case.

After a bit of research, I have found a keyword extraction model made available by **Yankı Ekin Yüksel** that we will use. I really love how simple it is to load and run a model using transformers and HuggingFace.

The following code loads the keyword extraction model and prepares a NLP pipeline.

```
tokenizer = AutoTokenizer.from_pretrained("yanekyuk/bert-uncased-keyword-extrac
model = AutoModelForTokenClassification.from_pretrained(
    "yanekyuk/bert-uncased-keyword-extractor"
)

nlp = pipeline("ner", model=model, tokenizer=tokenizer)
```

You don't have to download models or worry about file paths. Instead, you can simply define the model name as the argument of the tokenizer and the model, and the transformers library does all the work for you.

The pipeline returns tokens, which are not necessarily a word. Therefore, we need to construct the words back from tokens after the NLP pipeline finishes.

```
def extract_keywords(text):
    """
    Extract keywords and construct them back from tokens
    """
    result = list()
    keyword = ""
    for token in nlp(text):
        if token['entity'] == 'I-KEY':
            keyword += token['word'][2:] if \
              token['word'].startswith("##") else f" {token['word']}"
        else:
            if keyword:
                result.append(keyword)
            keyword = token['word']
    # Add the last keyword
    result.append(keyword)
    return list(set(result))

extract_keywords("""
Broadcom agreed to acquire cloud computing company VMware in a $61 billion (€57
""") # ['cloud computing', 'vmware', 'broadcom']
```

The example shows that the model extracted **cloud computing**, **vmware**, and **broadcom** from the given text. The results seem fitting for our use case as we are analyzing the Neo4j documentation, which should contain many technological keywords.

Next, we also need to calculate text embeddings that will help us identify similar and duplicate content. Again, I've search the HuggingFace model repository a bit and came across a sentence transformers model that can be used to identify similar sentences or paragraphs. Again, the model can be loaded and used with as little as three lines of codes.

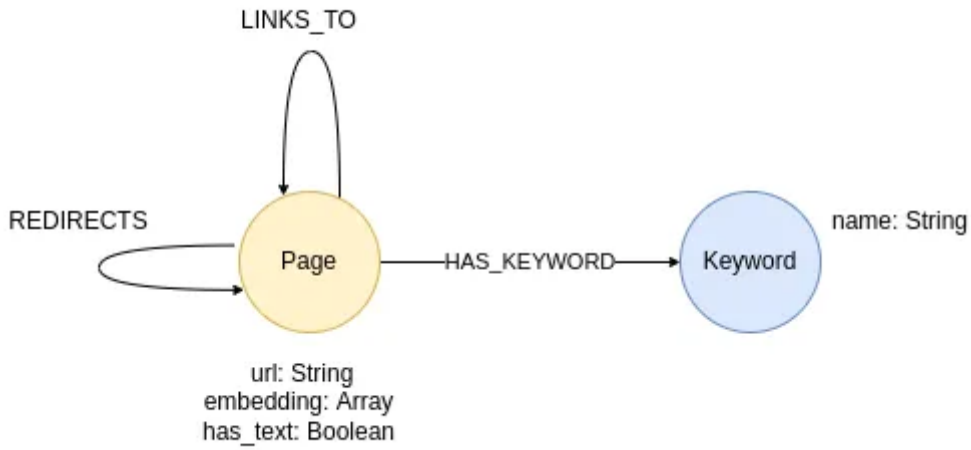
```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

def generate_embeddings(text):
    embeddings = model.encode(text)
    return [float(x) for x in embeddings.tolist()]
```

We need to convert the result to a float of lists as Neo4j Driver doesn't support NumPy arrays.

Knowledge graph construction

After the web scraper and natural language processing steps are done, we can go ahead and construct a knowledge graph. You might have already guessed that we are going to be using Neo4j to store our knowledge graph. You can use a free cloud instance or setup a local environment.

The graph schema after the initial import is defined as the following.



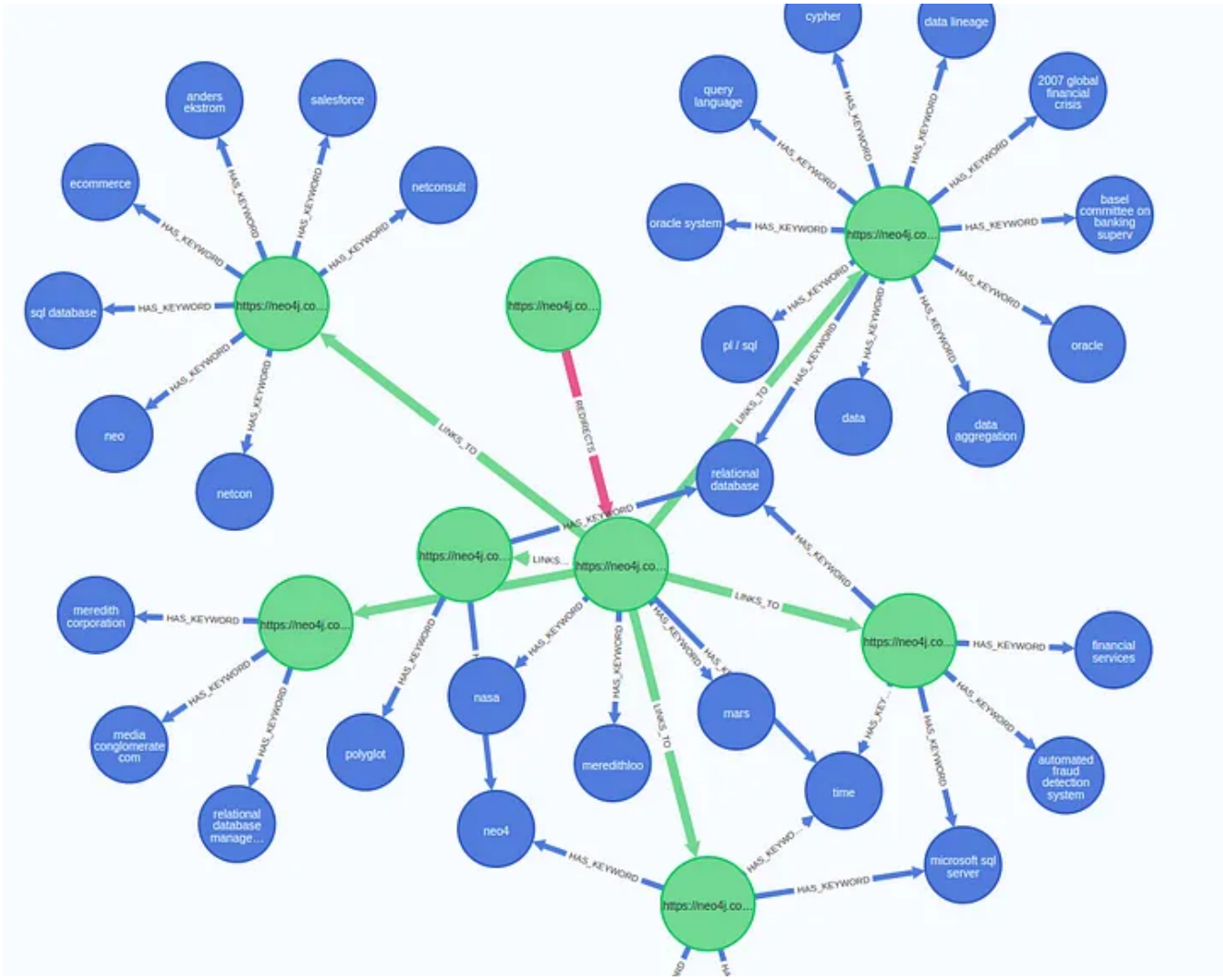
Initial graph schema. Image by the author.

In the center of our graph are web pages. We know their URL address, text embedding value, and whether or not the web scraper extracted text from the page. Pages can also link or redirect to other pages, which is represented with according

Check out the preprocessing notebook if you are interested in the code implementation of the data import. Otherwise, we will jump straight to the network analysis part.

Network Analysis

I have prepared a Neo4j database dump if you don't want to scrape the Neo4j documentation but would still like to follow the network analysis examples.



Sample subgraph of the Neo4j documentation website knowledge graph. Image by the author.

I will walk you through some website network analysis examples that I found interesting. We will be using the Graph Data Science Python client, which is ideal tool to perform network analysis with Neo4j from Python.

The Jupyter Notebook with all the relevant code for the network analysis is available on GitHub.

Overall statistics

First of, we will begin by evaluating the size of our dataset by counting the number of nodes and relationships with the `apoc.meta.stats` procedure.

```
gds.run_cypher("""
CALL apoc.meta.stats()
```

```
YIELD labels, relTypesCount
""")
```

Our knowledge graph has 15370 pages and 4199 keywords, along with 62365 links and 723 redirects. I was not expecting this many pages. However, considering that the documentation covers multiple products across multiple versions, it makes sense that the number of pages on a website can explode. Additionally, many links point to pages outside the Neo4j website.

Next, we will evaluate from how many pages we successfuly retrieved content information.

```
gds.run_cypher("""
MATCH (p:Page)
RETURN p.has_text AS has_text,
       count(*) AS count
""")
```

We have successfully retrieved the content and calculated text embeddings from 9688 web pages. The web scraper focused on recovering content from the documentation website while mostly ignoring the structure and text of the product and similar pages. Therefore, there are 2972 on the Neo4j website for which we haven't retrieved content. Finally, the Neo4j website links to 2710 outside its primary domain. Pages outside of the Neo4j documentation website were explicitly ignored during web scraping.

Out of curiosity, we can list ten random outside web pages that Neo4j links to the most.

```
gds.run_cypher("""
MATCH (p:Page)
WHERE p.has_text IS NULL
RETURN p.url AS page,
       count{(p)<-[:LINKS_TO|REDIRECTS]-()} AS links
ORDER BY links DESC
```

```
    LIMIT 5
    """)
```

Results

The most linked web page is actually a localhost URL, which is the default address of the Neo4j Browser. Following are some links to APOC releases on GitHub. Finally, it seems that Neo4j has some products or services that support integrations with Microsoft NLP and AWS cloud APIs, as otherwise, they probably wouldn't link them in their documentation.

Identify dead links

We will proceed by identifying dead or broken links. A broken link is a link that points to a non-existing web page. Like most websites, Neo4j documentation has a designated 404 web page. The web scraper assigns a "404" text value to any URL that responds with a 404 page.

```
gds.run_cypher("""
MATCH (:Page)-[:LINKS_TO|REDIRECTS]->(:Page{is_404:true})
RETURN count(*) AS brokenLinkCount
""")
```

There are 241 broken links in the dataset. The broken link number sounds small, given that there is a total of 62 thousand links in the database. However, if you performed this analysis on your website, you could always forward the results to the appropriate teams to get the links fixed.

Finding shortest paths

Most websites are designed to gently push the users along the journey to the end goal. For example, if you are running an e-commerce website, the goal is probably a purchase event. With Neo4j, you can analyze all the paths a user might follow to reach the desired destination. Since we are dealing with a documentation website, we can't explore how a user might end up completing a purchase on the website. However, we can apply the same techniques and evaluate the shortest paths between various parts of the website.

The following code finds all the shortest paths between the two given web pages.

```
gds.run_cypher("""
MATCH (start:Page {url:"https://neo4j.com/docs"}),
      (end:Page {url:"https://console.neo4j.io"})
MATCH p=shortestPath((start)-[:LINKS_TO|REDIRECTS*..10]->(end))
RETURN [n in nodes(p) | n.url] AS path
""")
```

The results show that a user must traverse the following web pages in order to reach the Aura console page from the documentation home:

- https://neo4j.com/docs
- https://neo4j.com/docs/aura/auradb
- https://neo4j.com/docs/aura/auradb/getting-started/create-database
- https://console.neo4j.io

Representing your website as a knowledge graph can significantly improve the understanding of designed flows throughout your web page, which in turn can help you optimize them.

Link analysis

Next, we will use centrality algorithms to rank the importance of the web pages. For example, let's say we simply define the rank of web pages as the number of incoming links. In that case, we can utilize the Degree centrality algorithm to rank the importance of the web pages.

To execute any graph algorithms from the Graph Data Science library, we have first to project an in-memory graph.

```
G, metadata = gds.graph.project('structure', 'Page',
  ['LINKS_TO', 'REDIRECTS'])
```

With the Graph Data Science library projection, you have the option to choose a specific subgraph of your knowledge graph you want to evaluate with graph algorithms. In this example, we selected the **Page** nodes and **LINKS_TO** and **REDIRECTS** relationships. For simplicity's sake, we will treat the links and redirects as identical. However, for more in-depth network analysis, we could define some weights and perhaps treat redirects as more important than links.

The following code will calculate the incoming degree centrality, which is simply the number of incoming links or redirects a web page has.

```
df = gds.degree.stream(G, orientation="REVERSE")
df["url"] = [d["url"] for d in gds.util.asNodes(df["nodeId"].to_list())]
df.sort_values("score", ascending=False, inplace=True)
df.head()
```

One thing to note here is that we are using the Graph Data Science Python Client to interact with the database, so the syntax might be slightly different if you are used to Cypher procedure calls. The results are the following:

The developer knowledge base page has 598 incoming links. Many links also point to specific tags of the developer blog and graph gists. I would assume that tons of

documentation sites point to specific examples which can be found in blogs of graph gists. If we were to understand the intended flow more, we could drill down where the links are coming from and so on.

Sometimes the number of incoming links is not a sufficient ranking metric. The founders of Google were aware of this issue as they derived the most famous graph algorithm, PageRank, which takes into account the number of incoming links and where they are coming from. For example, there is a difference if the web page has a direct link from the home page or some periphery documentation page that only some people visit.

The following code will calculate the PageRank score and merge it with the degree dataframe.

```
pr_df = gds.pageRank.stream(G)
pr_df["pagerank"] = pr_df.pop("score")
combined_df = df.merge(pr_df, on="nodeId")
combined_df.sort_values("pagerank", ascending=False, inplace=True)
```

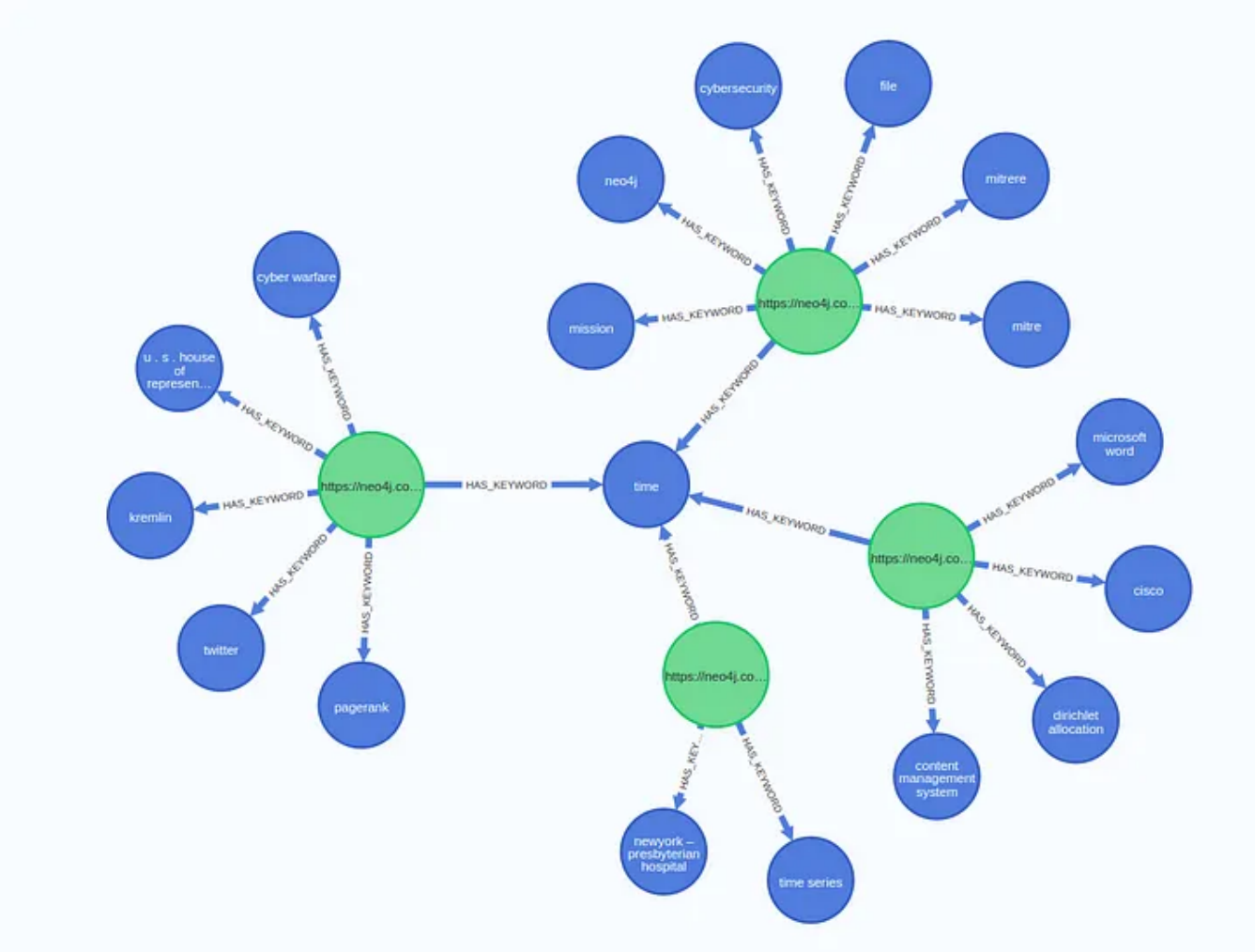
Now we can examine the top five most important web pages judging by the PageRank score.

The score column represent the number of incoming links and redirects, while the pagerank columns hold the PageRank score.

Interestingly, only the developer knowledge base page retains its position when using PageRank instead of Degree centrality. It seems that GraphConnect was vital as it is still the second most important web page. As a web UX designer, you might take this information and try to change the structure of the website so that perhaps the latest GraphConnect would be more important. Remember, we are only scratching the surface with this network analysis. However, you could find interesting patterns and then drill down to understand the web page flow and optimize it.

Keyword analysis and co-occurrence topic clustering

In the last part of this analysis, we will take a look at the keyword analysis.



Graph representation of web pages and their keywords. Image by the author.

Having the right keywords on the right web pages is one of the critical aspects of search engine optimization. We can get a high-level overview of the page by examining its most frequent keywords.

```
gds.run_cypher("""
MATCH (k:Keyword)
RETURN k.name AS keyword,
       count {(k)<-[:HAS_KEYWORD]-()} AS mentions
ORDER BY mentions DESC
LIMIT 5
""")
```

Results

The results look exactly what we might expect. The web page talks about nodes, neo4j, graphs, and Java. Not sure why the clipboard is there. Perhaps there are a lot of “Copy to clipboard” sections throughout the documentation.

We can drill down a bit and look at the most frequent keywords for web pages where the “graph-data-science” is present in the URL address. This way, we filter primarily for Neo4j Graph Data Science library documentation.

```
gds.run_cypher("""
MATCH (p:Page)-[:HAS_KEYWORD]->(k:Keyword)
WHERE p.url CONTAINS "graph-data-science"
RETURN k.name AS keyword,
       count(*) AS mentions
```

```
    ORDER BY mentions DESC
    LIMIT 5
    """)
```

Results

It looks very similar to the overall keyword presence, except the keyword "algorithm" shows up more frequently here. Now, we could go ahead and drill down keywords by other sections or by individual pages. A knowledge graph is a fantastic tool for either drill-down analysis or to analyze the distribution of keywords and content through designated user flows. Additionally, if you used an NLP model that

is able to detect both short- and long-tail keywords, it would greatly help with any SEO analysis and optimization.

Lastly, we can also perform a keyword co-occurrence clustering with only a couple of lines of code. A keyword co-occurrence clustering can be understood as a task of identifying topics, where the topics consist of multiple keywords that frequently co-occur in the text corpus.

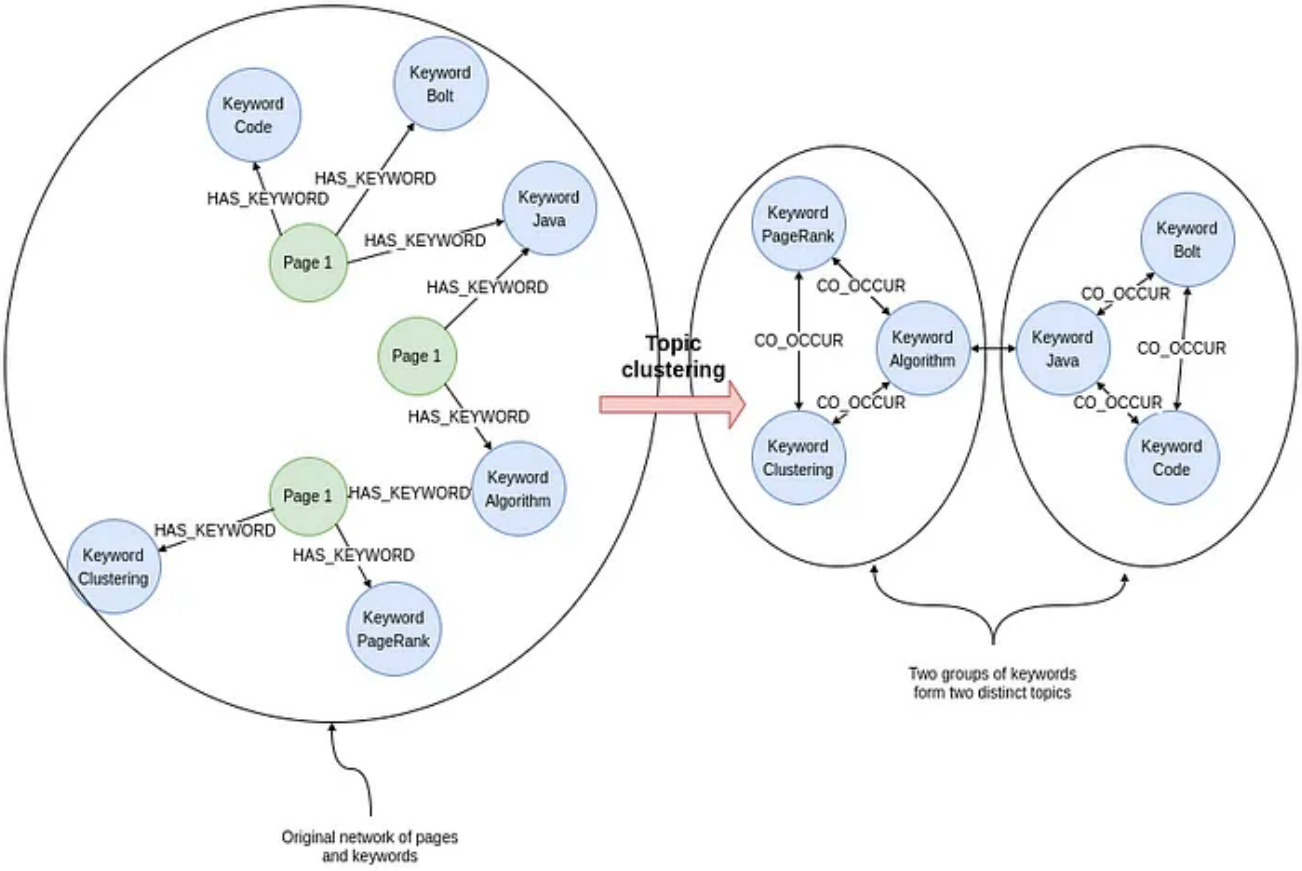


Diagram of the co-occurrence topic clustering output. Image by the author.

The workflow of the keyword co-occurrence clustering or topic clustering in Neo4j is the following:

1. Project an in-memory graph with relevant information.
2. Create a CO_OCCUR relationship between keyword that commonly appear together in text.
3. Run a community detection algorithm like the Louvain method to identify communities or clusters of keywords.

We will begin by projecting an in-memory graph with all the relevant information. We need to project both the Page and the Keyword nodes along with the connecting HAS_KEYWORD relationship. We need to reverse the relationship orientation in the graph projection as we want to examine clusters of co-occurring keywords and not groups of similar web pages.

P.s. If you leave the natural orientation and follow the examples you will identify clusters of similar web pages based on the keywords they mention

```
G, metadata = gds.graph.project(
    "keywords", ["Page", "Keyword"], {"HAS_KEYWORD": {"orientation": "REVERSE"}
)
```

Next, we need to create a **CO_OCCUR** relationship between keywords frequently appearing together on web pages. To solve this task, we will use the Node Similarity algorithm. The Node Similarity uses the Jaccard similarity coefficient by default to calculate the similarity between two nodes.

In this example, each keyword has a set of web pages it is mentioned in. If the Jaccard coefficient between a pair of keywords based on the web pages they appear in is greater than 0.40, then a new **CO_OCCUR** relationship is created between them. We use the mutate mode to store the algorithm's results back to the in-memory projected graph.

```
gds.nodeSimilarity.mutate(
    G, mutateRelationshipType="CO_OCCUR", mutateProperty="score",
    similarityCutoff=0.4
)
```

Finally, we will use the Louvain method algorithm, a community detection algorithm, to identify clusters of keywords. The algorithm outputs each node and its community id. Therefore, we need to group the results by their community id to create a list of keywords that form a topic or a cluster.

```
topic_df = gds.louvain.stream(G, nodeLabels=["Keyword"], relationshipTypes=["CO
topic_df["keyword"] = [
    n["name"] for n in gds.util.asNodes(topic_df["nodeId"].to_list())
]
topic_df.groupby("communityId").agg(
    {"keyword": ["size", list]}
).reset_index().sort_values([("keyword", "size")], ascending=False).head()
```

Results

	communityId	keyword	
		size	list
634	1250	46	[santa, chewbacca, galaxy, republic, jedi, christmas eve, new year ' s day, gandhi jayanti dussehra, caribbean iii, ganesh chaturthi, thanksgiving, makar sankranti republic day, vijayadashami, diwali padwa veterans day, labor day, maha shivratri holi memorial day, christmas day, independence day, boxing day, new year ' s eve, diwali, president ' day, martin luther king day, good friday, caribbean, christmas, veterans day, thanksgiving day, maha shivratri, king iii, president ', padwa, holi memorial day, king iii holiday, reindeer, christmas tree, santa claus, street map, rome, android, tatooine, wookiepedia, alderaan, jorge albarran, santavin, luke skywalker]
1749	3181	35	[mar cabra, mossack fonseca, tax haven, panama, panama papers, apache solr, tika, prizesion, offshore leaks, oxwall source, zurich switzerland, paul kuhne, zurich, swiss leaks, hsbc, hsbc leaks, excel source, president of azerbaijan, panamapaper, mahabharataa, president, data science analytics, quantum analytics, azerbaijan, firepower scandal, cablegate, pentagon papers, ilham aliyev, the case, duncan campbell, baku, heydar aliyev, azerbaijan airlines, dubai, azerfo]
2212	4063	31	[reddit, avengers : infinity war, evelina gabasova, seattle company, craig walls, pokemon, pokemonpher, tom hiddleston, tom hidton, avengers, the avengers, x - men, node classification learning, marvel universe, hulk, spider - man, thanosbra, s . h . i . e . l ., nick fury, black panther, loki, hawkeye, guardians of the galaxy, thanos, infinity stones, marvel cinematic universe, iron man, captain america, doctor strange, thor, natasha romanoff]
1712	3074	31	[netcon, salesforce, anders ekstrom, netconsult, sql database, search engine, musicology, chief technical officer, streaming services, music industry, the orchard, the orchard444j, music distribution, jeremy davies, webflow, stephen o ' grady, media, state, manhattan, marriott marquis times square, tim hanssen, pat patterson, cloudera, streamsetser, hilary mason, jake graham, lauren shin, artificial, change data capture, redmonk, redmon]
1551	2749	27	[maharashtra, airtel, tata, vodafone, mobile operator, gujarat, rajasthan, centurylink, business conglomerate, prodapt, at & t, jhaver group, deutsche telekom, verizon, liberty global, windstream, adtran, virgin media, ebay, forresterner, comcast, fortune 100, accorhotels, jpmorgan chase, atpco, forrester, adeo]

Since the topic clustering workflow we followed is an unsupervised technique, we need to manually assign overall topic names. For example, we can observe that the first and largest topic contains keywords like chewbacca, jedi, christmas day, independence day, and so on. It is a an interesting mix of holidays and Star Wars. We could explore why both holidays and Star Wars are mixed together. Additionally, the second largest topic seems to talk about various panama and paradise papers along with the companies and people involved.

Summary

In my opinion, knowledge graphs and natural language processing techniques are a match made in heaven. As mentioned, I have seen similar approaches to analyzing medical documents, news, or even crypto reports. The idea is to use NLP and other tools to extract valuable information from unstructured data, which is then used to construct a knowledge graph. A knowledge graph offers a friendly and flexible structure of the extracted information that can be used to support various analytical workflows.

All the code of this blog post is available on GitHub.

Neo4j Graph Data Science NLP Deep Dives

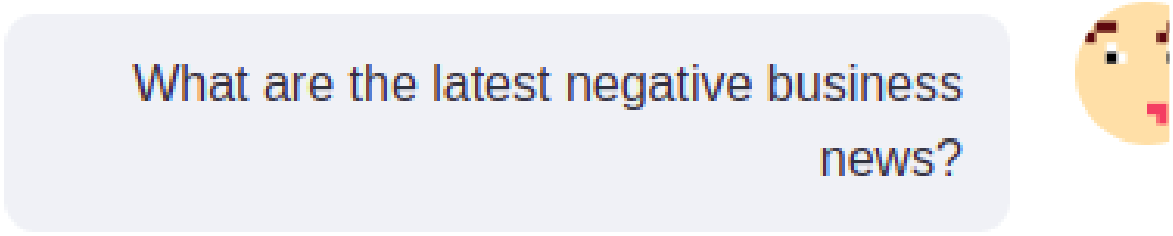
Following

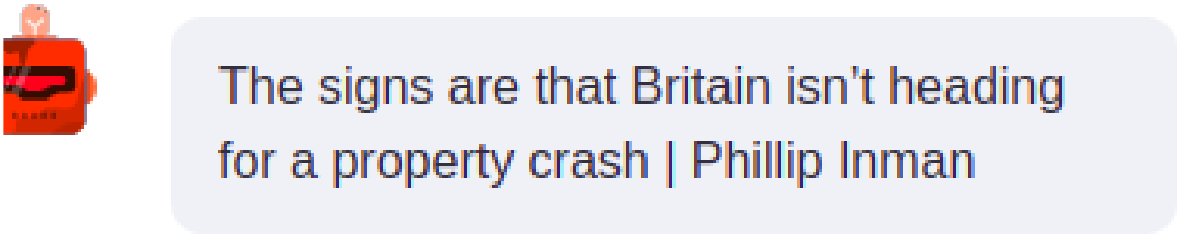
Written by Tomaz Bratanic

5.9K Followers · Writer for Towards Data Science

Data explorer. Turn everything into a graph. Author of Graph algorithms for Data Science at Manning publication. http://mng.bz/GGVN

More from Tomaz Bratanic and Towards Data Science





Tomaz Bratanic in Neo4j Developer Blog

Knowledge Graph-Based Chatbot With GPT-3 and Neo4j

Learn how to develop a chatbot that provides answers based on data stored in a knowledge graph

11 min read · Mar 9

352 5

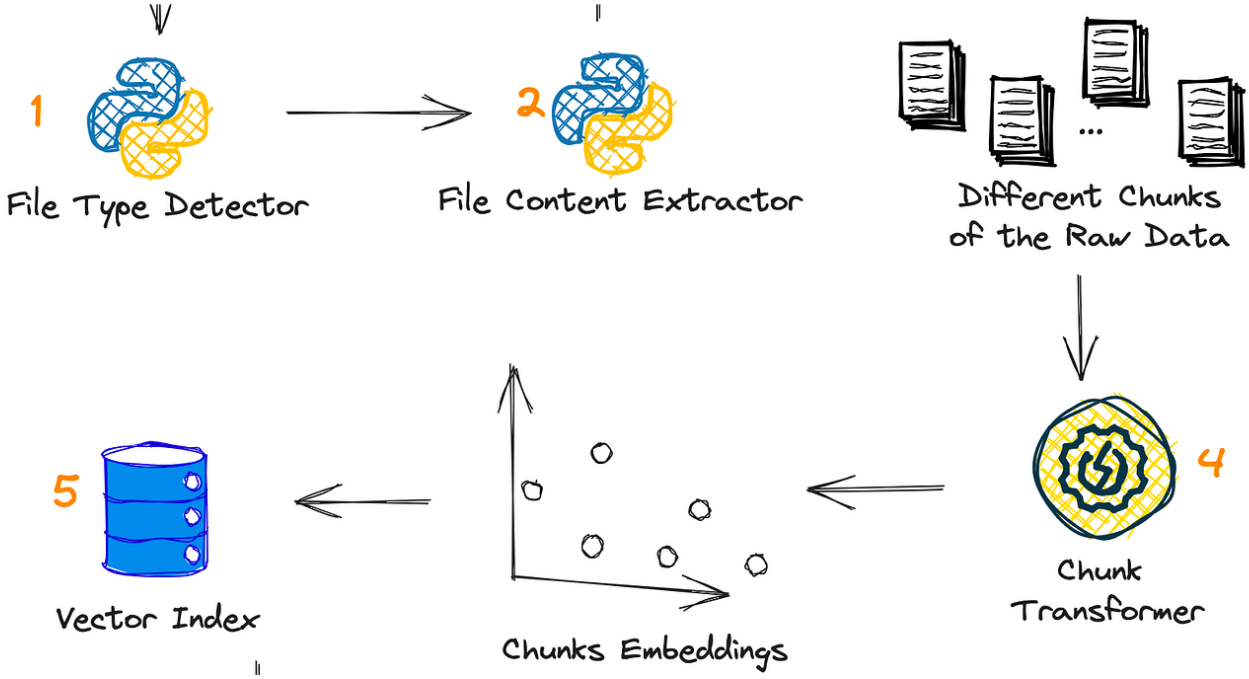
Bex T. in Towards Data Science

130 ML Tricks And Resources Curated Carefully From 3 Years (Plus Free eBook)

Each one is worth your time

· 48 min read · Aug 1

2.9K 10



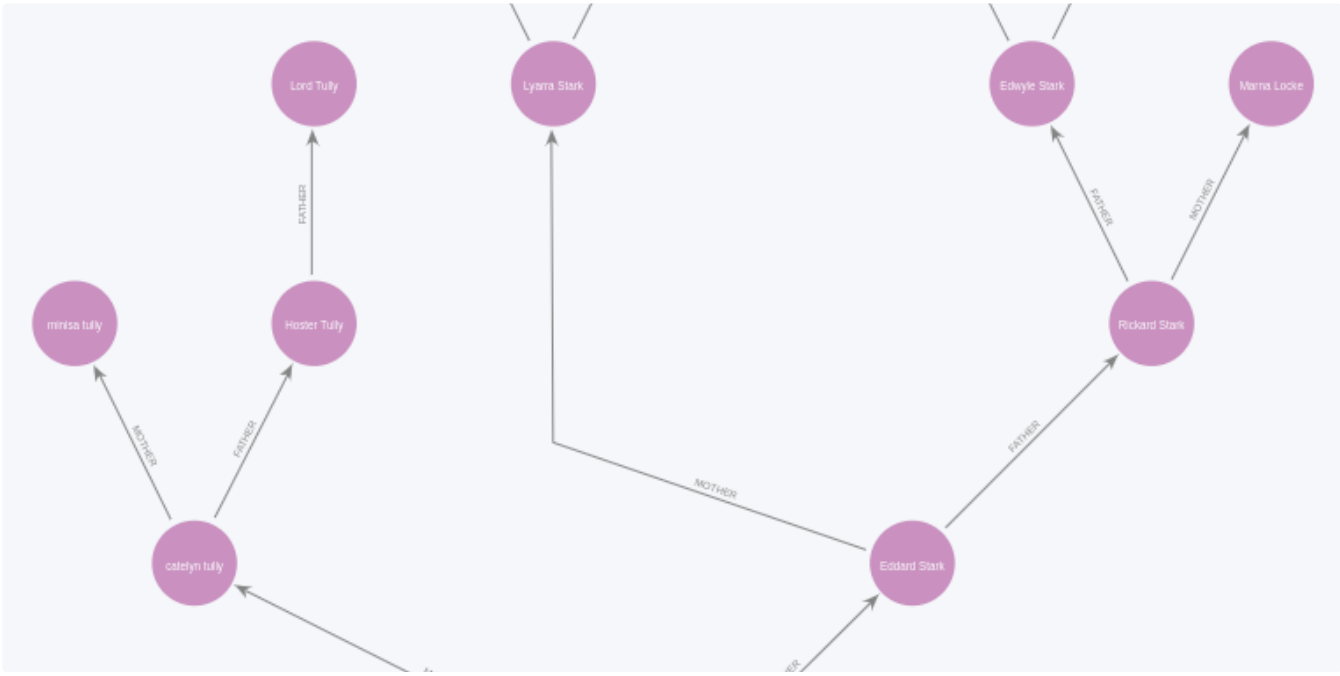
Zoumana Keita in Towards Data Science

How to Chat With Any File from PDFs to Images Using Large Language Models—With Code

Complete guide to building an AI assistant that can answer questions about any file

· 9 min read · Aug 5

840 11



Tomaz Bratanic in Towards Data Science

Investigate family connections between House of the Dragon and Game of Thrones characters

Interactively analyze and visualize the family tree using Neo4j

12 min read · Nov 10, 2022

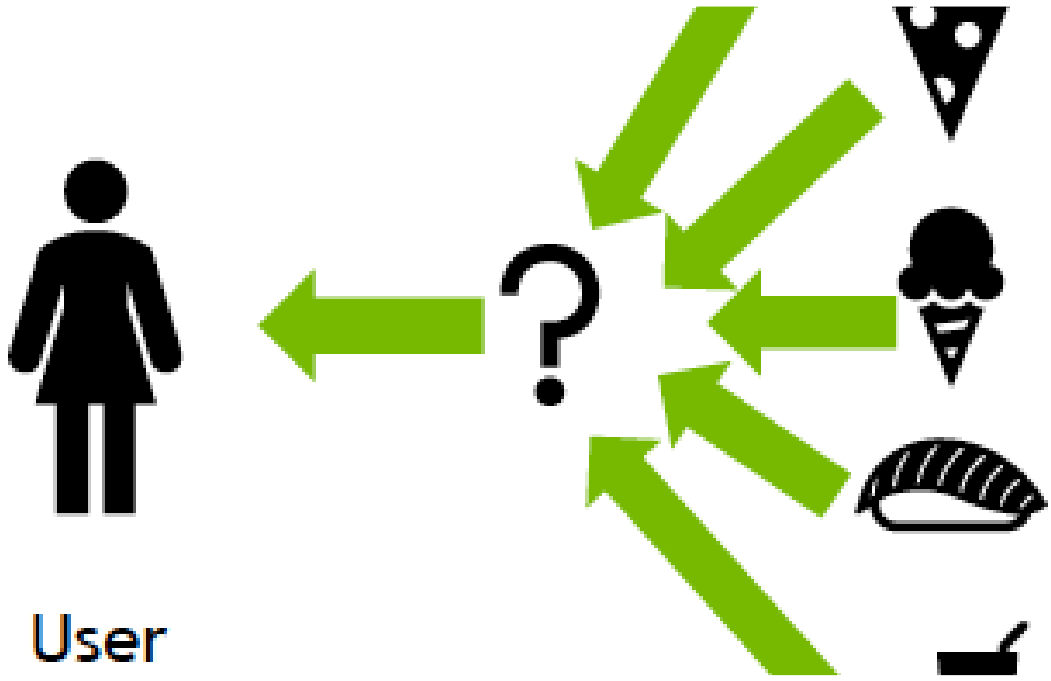
Recommended from Medium

Mohamed El-Bably

GPT Graph: A Simple Tool for Knowledge Graph Exploration

As a developer, exploring and organizing information is a crucial part of the job. That's why I wanted to share with you an open-source...

5 min read · Apr 30



Practical Guides to Machine Learning

10 stories · 316 saves

Maximilian Vogel in MLearning.ai

The ChatGPT list of lists: A collection of 3000+ prompts, examples, use-cases, tools, APIs...

Updated Aug 20, 2023. Added prompt design courses, masterclasses and tutorials.

10 min read · Feb 8

7K 75

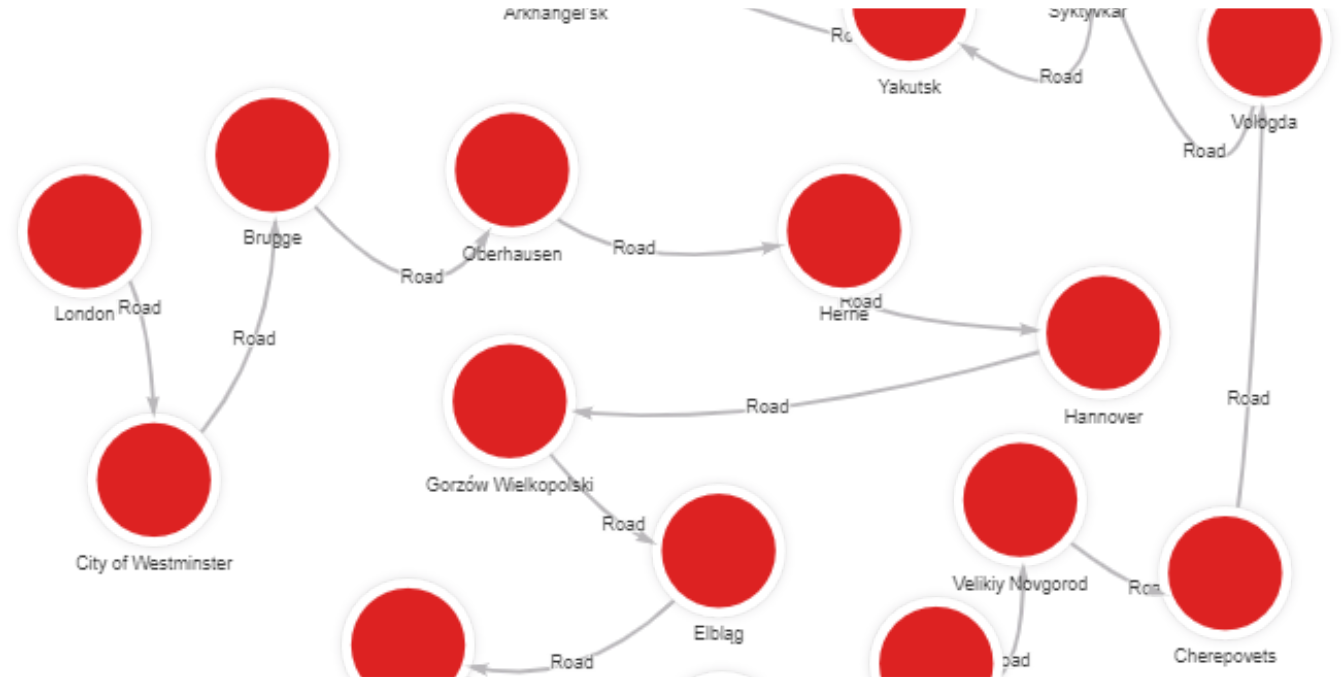
Prathamesh Gadekar in Level Up Coding

Python Libraries for Lazy Data Scientists

Do you feel lethargic today? Use these five libraries to boost your productivity.

7 min read · Apr 7

701 4



Roshmita Dey

Neo4j: Unraveling Complex Relationships in Data with Graph Databases

In the ever-evolving landscape of data management, traditional relational databases have long been the cornerstone of structured...

4 min read · Aug 10

3

Jim the AI Whisperer in The Generator

Is ChatGPT getting dumber? Let's talk about 'AI Drift'

Why is AI losing its edge? AI Drift explained 🚗💨🤖

· 8 min read · Aug 16

1.8K 30

See more recommendations