

ClinicalAI: Instruction Manual

How to build the clinicalai knowledge graph, and query it with natural language using a large language model

Prerana S K | ClinicalAI | 19-07-2024

# **Brief Overview of the Project**

90% of clinical experiments fail, due to insufficient background work and research. This can be improved with the help of better pre-planning and more informed decision making. ClinicalAI is a project that aims to bring together the logical deduction capabilities and structure provided by a KG (Knowledge Graph) with the natural language interpretation capabilities of an LLM. This is a step-by-step instruction manual that goes through all the steps (from scratch) of completing the set-up for this project. In the following sections, we first see how to collect and clean data from <https://www.clinicaltrials.gov/>. Then we see how a schema is built on Neo4j’s AuraDB, and finally how to integrate it with GPT-4.

\*\*\* All code can be found at <https://github.com/Narpear/HeCiX>

## **Section-A (collecting and cleaning the dataset)**

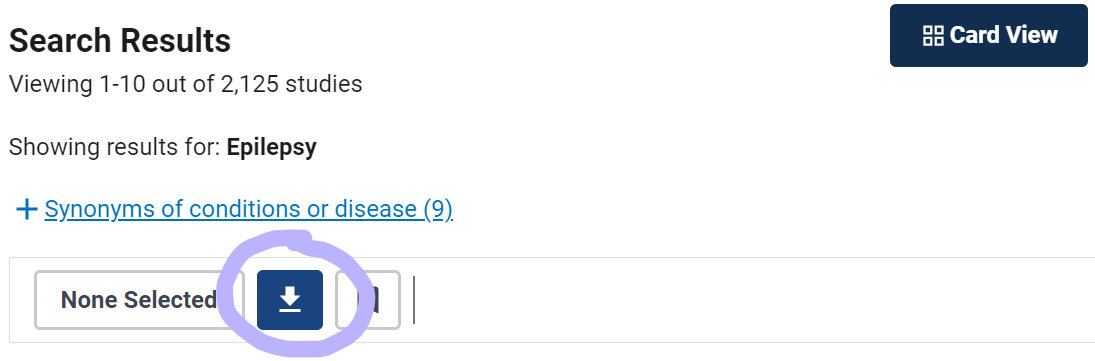
### **Collecting the Dataset**

The first step of our project involves collecting our data and cleaning it. For this proof-of-concept project, we acquire data from <https://www.clinicaltrials.gov/> of six diseases, 200 records each. The six diseases we use are vitiligo, melanoma, atopic dermatitis, alopenia areata, vitiligo, and epilepsy. (On reverse engineering, we realized that these have common nodes such as principal investigator, conditions, etc. and therefore picked these six diseases.)

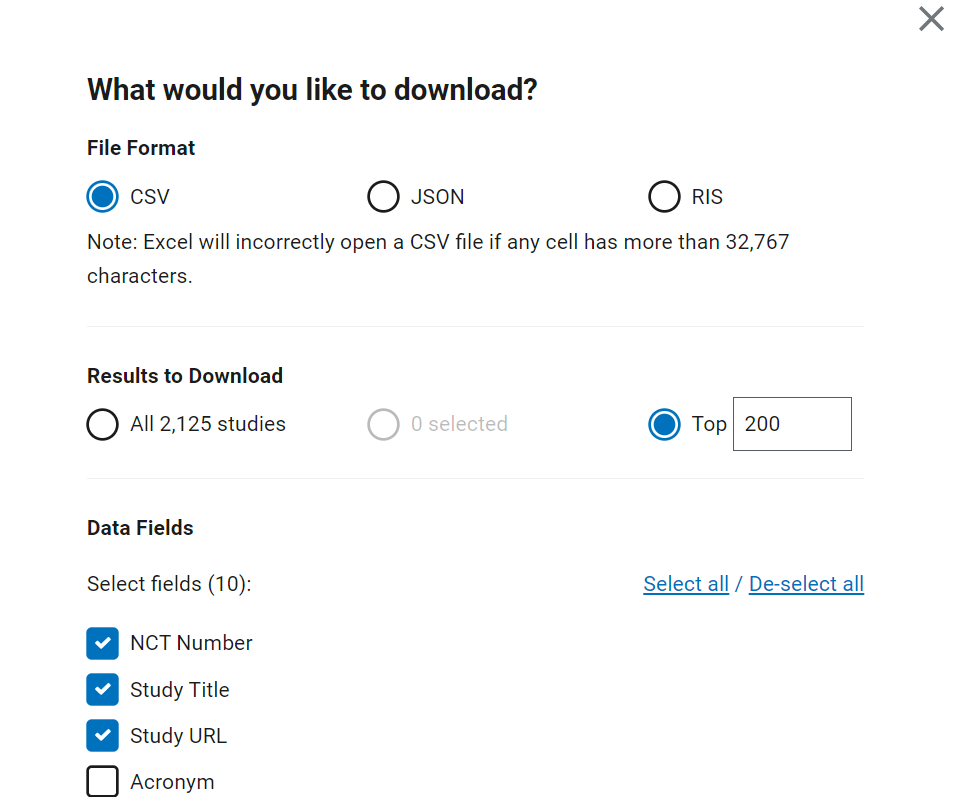
To do this, we visit <https://www.clinicaltrials.gov/> and type in the name of the disease of interest. In our case, this is epilepsy. The other fields can be left blank.



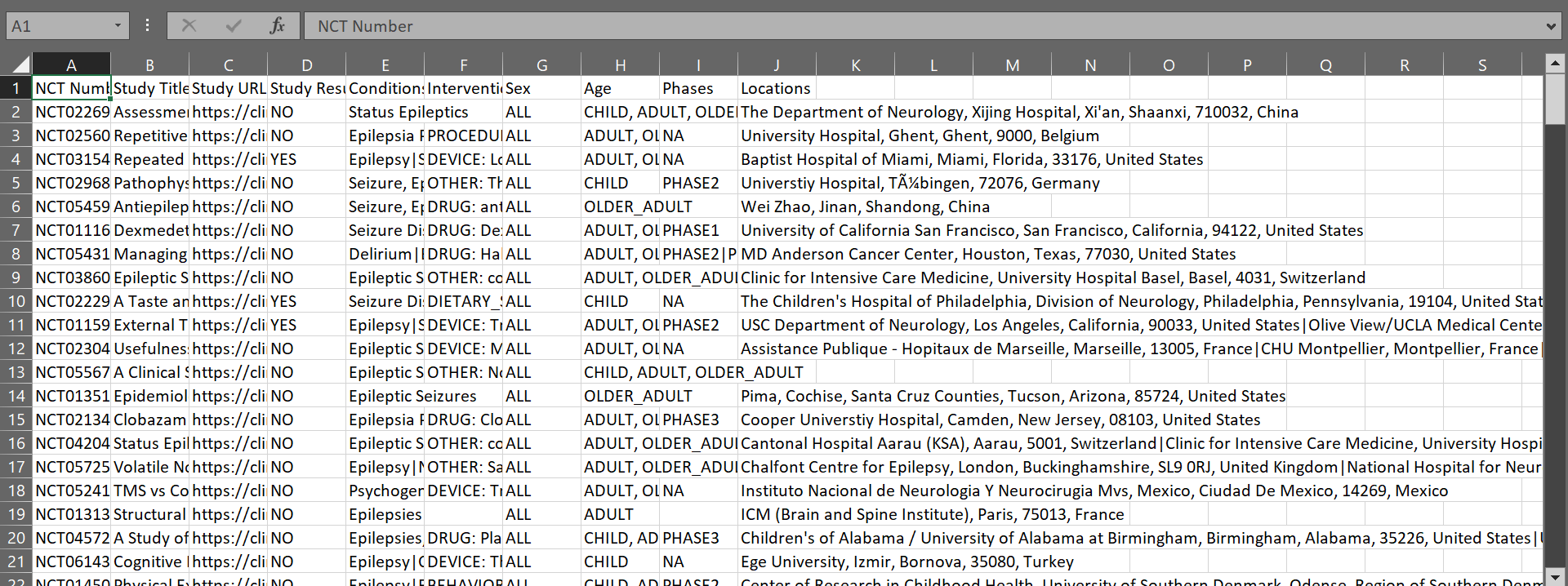
We then click on the download button on the page.



Then, we choose the selected fields and download our data. We chose to download the data in .csv format, and decided to download 200 records for the purpose of our study. The fields selected are NCT Number, Study Title, Study URL, Study Results, Conditions, Interventions, Sex, Age, Phases, and Locations.



The downloaded .csv file should look something like this



Since this does not include information about Principal Investigators, our next step is to scrape this data from the webpage dedicated to each clinical trial.

### **Scraping Principal-Investigators from Individual Webpages**

To scrape this data from webpages of each clinical trial, we write some basic python code. This code can be found at <https://github.com/Narpear/HeCiX/tree/main/clinical_trials>.

There are four python files here that help with scraping data we need.

* **download\_html.py**

(<https://github.com/Narpear/HeCiX/blob/main/clinical_trials/download_html.py>)

This function downloads and saves the formatted HTML content of a specific clinical trial page from clinicaltrials.gov using its NCT number.

* **get\_div.py**

(<https://github.com/Narpear/HeCiX/blob/main/clinical_trials/get_div.py>)

This function extracts and returns the principal investigator names from HTML pages locally saved by downloag\_html.py, identified by its NCT number.

* **delete\_html.py**

(<https://github.com/Narpear/HeCiX/blob/main/clinical_trials/delete_html.py>)

This function deletes a specific HTML file named after an NCT number if it exists in the current directory, or prints a message if the file does not exist.

* **append\_pis.py**

(<https://github.com/Narpear/HeCiX/blob/main/clinical_trials/append_pis.py>)

This script processes a CSV file of clinical trials, downloads each trial's webpage, extracts principal investigator names, updates a dataframe with this information, deletes the downloaded HTML, and saves the results to a new CSV file.

The file titled append\_pis.py is the main file that is to be run, and the others are basically helper functions. The paths used within the code should be changed, to match the locations of the files on other systems.

### **Combining Collected Data and Cleaning Up**

The data collected from different diseases are to be put in a single .csv file. This is done by appending the name of the disease the trial is associated with, at the beginning of each record. This can be found here: <https://github.com/Narpear/HeCiX/blob/main/clinical_trials/6_Diseases_With_PI%20-%20Sheet1.csv>.

The columns with multiple values (principal investigators, conditions, locations, and interventions) are split into separate columns. These cells are left blank for records that do not have multiple values. This can be found here: <https://colab.research.google.com/drive/12cW196ALr-UwzGjPGPlFM4trC1o6gLkK?usp=sharing> and the code to separate columns can be found here <https://colab.research.google.com/drive/12cW196ALr-UwzGjPGPlFM4trC1o6gLkK?usp=sharing>.

Finally, special and unidentified characters are replaced with true values from the actual webpages, by manual cleaning.

The processed .csv file that’s ready for use can be found at <https://github.com/Narpear/HeCiX/blob/main/clinical_trials/Cleaned%20Final%20Dataset.csv>

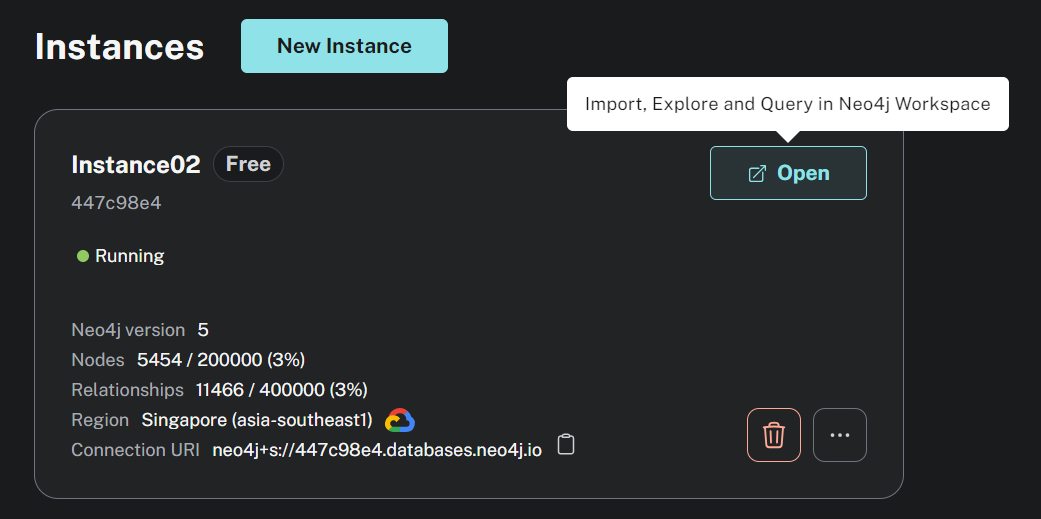
This completes dataset collection and cleaning.

## **SECTION B (BUILDING THE KNOWLEDGE GRAPH)**

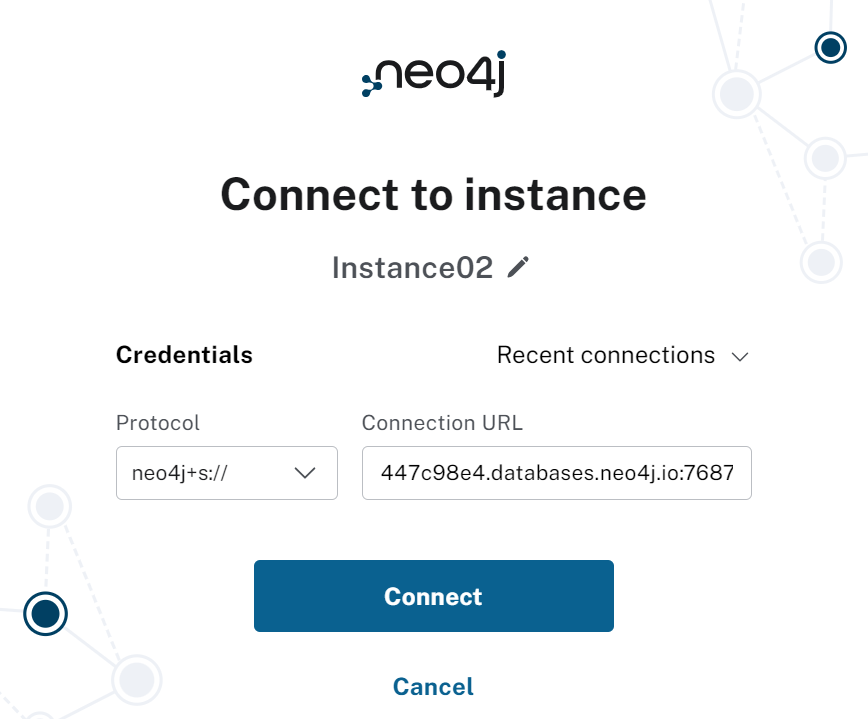
The following steps are all done on AuraDB (<https://neo4j.com/cloud/platform/aura-graph-database/>). First, you must create an instance and note down the username (which is usually neo4j) and the password, or change the password if needed and store it securely.

### **Constructing the Schema**

First, we open our instance of AuraDB.

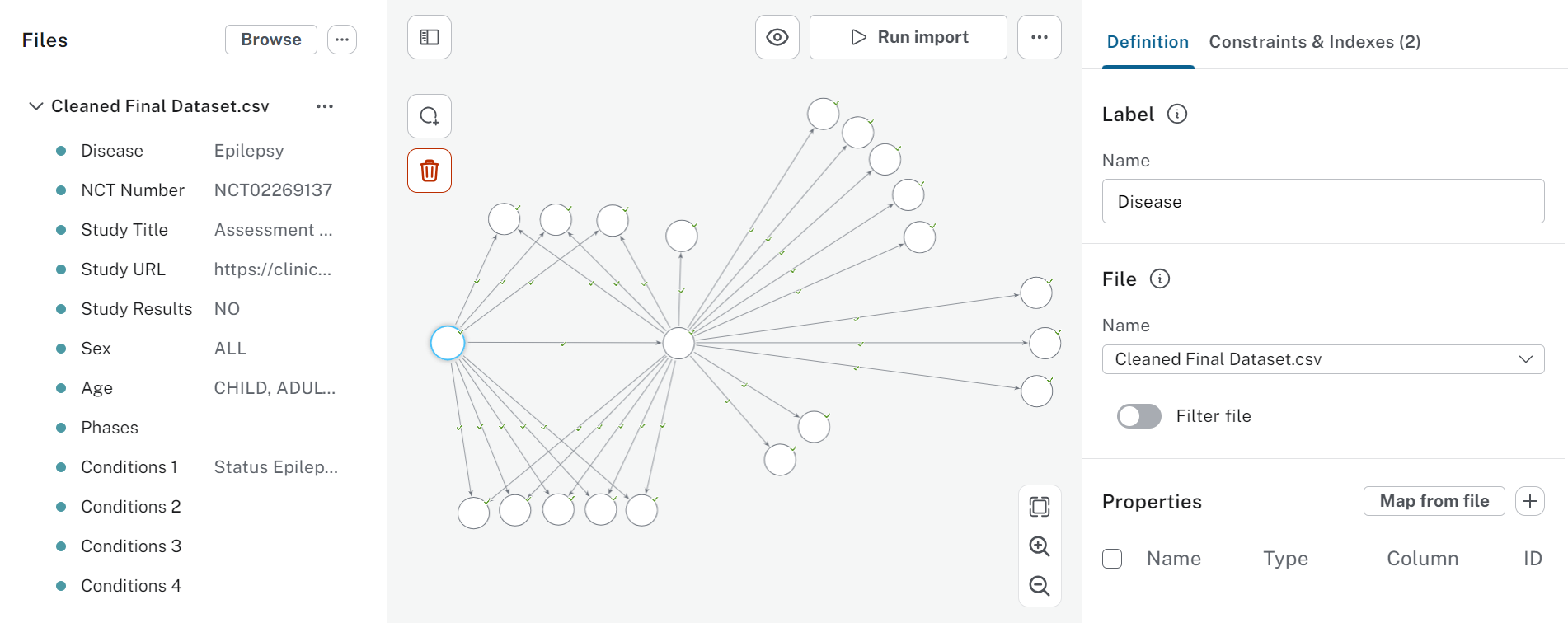


The URI of the instance is the protocol, followed by the connection URL. Make sure to store the URI safely as well, since it is important for connecting to the database later.



For example, the URI here would be neo4j+s://447c98e4.databases.neo4j.io:7687

Import the .csv file and build the schema and map the columns of the csv to the nodes and edges of the knowledge graph. This is done by creating a node/relationship, then double clicking on it and adding the required properties to it.



Once the schema is built, ‘Run import’ to populate the schema and finish constructing the knowledge graph.

This completes building the knowledge graph.

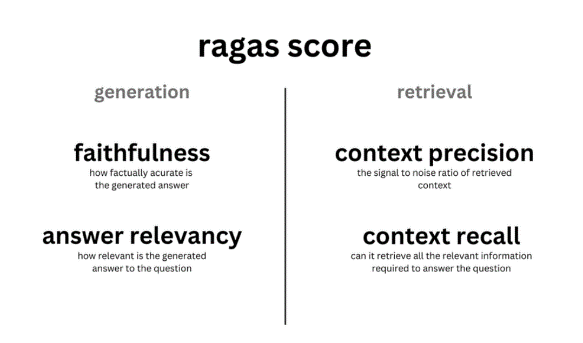
## **SECTION C (Connecting the knowledge graph to the llm)**

<https://github.com/Narpear/HeCiX/blob/main/KG_Aug_LLM_GPT4.ipynb> is the code file used to connect the LLM to the Knowledge Graph. The KG once connected to GPT-4, can be queried using natural language. LangChain is used to connect the KG with the LLM. Natural language queries sent to the system are converted into CQL (Cypher Query Language) by GPT-4. GraphCypherQAChain is a tool from the LangChain ecosystem that’s used to query graph databases. This sends the cypher query to the AuraDB database and retrieves the result. The answer is stored as the context. The cell output is captured, and send to GPT-4 once again (along with the question), for further formatting and receiving the output in a more polished and human-readable (natural language) format.

Before running the entire code, make sure to save environment variables (and also enter the OpenAI API key).

## **SECTION D (evaluating the rag model)**

<https://github.com/Narpear/HeCiX/blob/main/Prer_RAGAS_Evaluation.ipynb> contains code to evaluate the RAG model using the RAGAS library. Here are the metrics we have used, and what they mean:



And we are done!