Semantic Data Technologies Report MOD004979

Assessment Title:
Semantic Modelling for Disaster Management Dataset

Team:
Mohamed Talha Hussain (2367793)
Abdul Khadar Mohammad (2334343)
Sahil Tambe (2353465)

Table of Contents

1. Introduction

Disasters are part of challenges in human life, its management has evolved significantly with the development of robust technologies. By leveraging tools like semantic data, natural language processing (NLP), we can improve decision-making processes, enabling efficient responses. This report explores the intersection of such technologies and their application in disaster management.

This project focuses on the development and application of a semantic model for disaster management. The objective is to create an ontology-based representation of disaster management data, addressing critical questions such as resource allocation, disaster preparedness, and response strategies. The dataset used in this project includes various aspects of disaster management, ranging from emergency response to recovery and mitigation measures. The design section introduces the ontology, including its classes, properties, individuals, and axioms. The requirement mapping section displays how the ontology can answer specific disaster management-related questions, linking each query to the relevant classes and properties within the ontology. In the evaluation phase, five different queries will be run on the Fuseki server, with a focus on ensuring that the system returns the correct and expected results. The implementation will also involve the use of Natural Language Processing (NLP) techniques.

2. Concept and Requirement Analysis

1.1 Aim of the Project

The aim of this project is to enhance disaster management capabilities by employing semantic web technologies and NLP to structure and analyze disaster-related data effectively. Specifically, the project focuses on developing ontologies to formalize disaster concepts, using SPARQL queries to extract meaningful insights from large datasets, and incorporating NLP to process unstructured data. Integrating these approaches with analytics tools like Jupyter Notebooks, the project seeks to improve real-time decision-making and disaster response strategies (Cohen et al., 2015).

1.2 Gap in the Knowledge (Rationale)

Disaster management systems, despite significant evolution over time, still face major challenges in effectively integrating, interpreting, and utilizing data across various domains such as meteorology, healthcare, logistics, and infrastructure. Many existing systems rely on siloed data repositories or struggle to integrate real-time information, which leads to slow, fragmented decision-making and increased vulnerability to errors. The lack of interoperability between systems, compounded by unstructured or semi-structured data sources, exacerbates these challenges. The adoption of semantic models offers a promising solution by creating a unified, context-aware framework where data is not only connected but enriched with meaning, enabling the capture of complex relationships between disaster events, geographic regions, affected populations, and relief efforts in ways that traditional databases cannot. However, developing and deploying such models remains a challenge due to data heterogeneity,

which complicates the creation of a coherent understanding across disparate platforms. Moreover, the absence of real-time data-sharing protocols between national and international agencies further hinders disaster response. Given the global nature of many disasters, such as pandemics, wildfires, and cyclones, semantic modeling can enhance interoperability, ensuring that data from multiple agencies can be meaningfully combined, interpreted, and queried, thus improving disaster response coordination (Schulze et al., 2016; Bernstein et al., 2016; Manning et al., 2014).

1.3 List of Questions

No.	Questions
1	Display the number of deaths and injuries caused by the Viral diseases in Asia ?
2	List the number of disasters occurred in each region?
3	Name the top 3 types of disasters and their locations with highest magnitude ?
4	What is the total number of people left homeless due to disasters ?
5	List the 3 major disasters of 2022 with their countries ?
6	Which disasters have been responded by agencies ?
7	List all disaster events with their names, start years, and end years.
8	Which disaster types have the highest average number of affected individuals?
9	Which are the most recent disaster events, along with their names and end Years ?
10	Which disaster events, along with their names, have overlapping time periods ?
11	Which regions and subregions have the highest number of disaster events ?
12	Which pairs of disaster types overlap in their timeframes ?
13	Identify Most Common Disaster Types by Region and Year ?
14	Most Recent Disaster for Each Disaster SubType ?
15	Retrieve Disaster Events with Multiple Locations and Cross-Region Impacts ?
16	Find the Third Most Recent Disaster by SubType ?
17	Identify Regions with the Highest Casualties ?
18	List Disaster Events with Total Regions Affected and Region Names ?

Table 1: List of Questions

3. Design (Semantic Model for Disaster Management Dataset)

1.4Ontology Summary

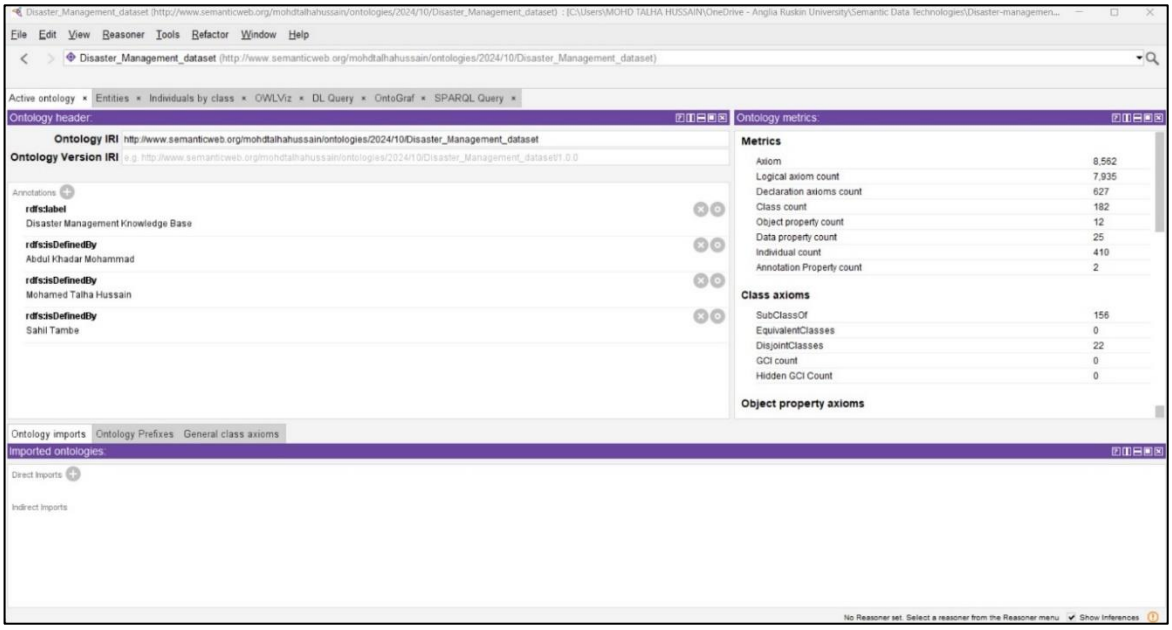


Figure 1: Movie Dataset Ontology Overview

The ontology consists of 182 classes and 156 subclass relationships establishing hierarchies and 22 disjoint axioms. With 12 object properties and 25 data properties, it captures both inter-class relationships and literal attributes of entities. The ontology includes 410 individuals, serving as instances to exemplify the defined concepts. Notably, 2,400 object property assertions and 4,941 data property assertions are utilized to link these individuals with properties and values, ensuring detailed representation. Metadata annotations, such as contributors and a clear label (“Disaster Management Knowledge Base”).

1.5 Classes

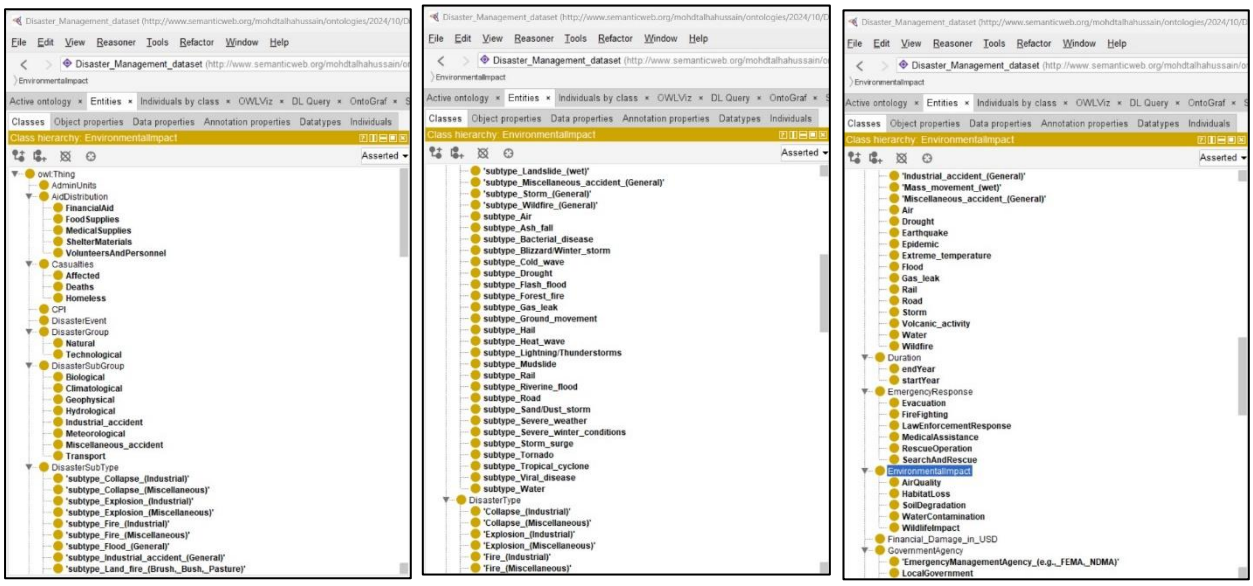


Figure 2 : Class Hierarchy

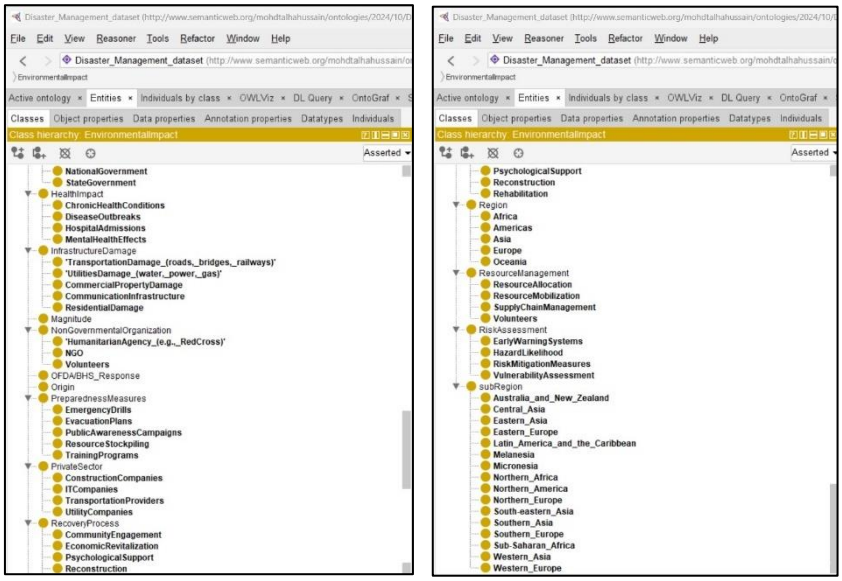


Figure 3 : Class Hierarchy

The set of classes capture various facets of disaster events, their impacts, and the responses to them. These classes are organized into multiple categories, reflecting the broad scope of disaster management. DisasterEvent and DisasterGroup classify disasters by type (e.g., Natural, Technological) and subgroups (e.g., Biological, Climatological, Geophysical). These events are further detailed with specific DisasterSubTypes, such as earthquakes, floods, wildfires, and industrial accidents, enabling precise categorization of the disaster's nature. The dataset also includes classes related to the affected populations, such as casualties, deaths, and the homeless, as well as the emergency response efforts, including evacuation, firefighting, rescue operations, and medical assistance. These elements are tied together by the CPI (Casualty, Property, and Infrastructure Impact) metrics, which assess the severity of the event's consequences.

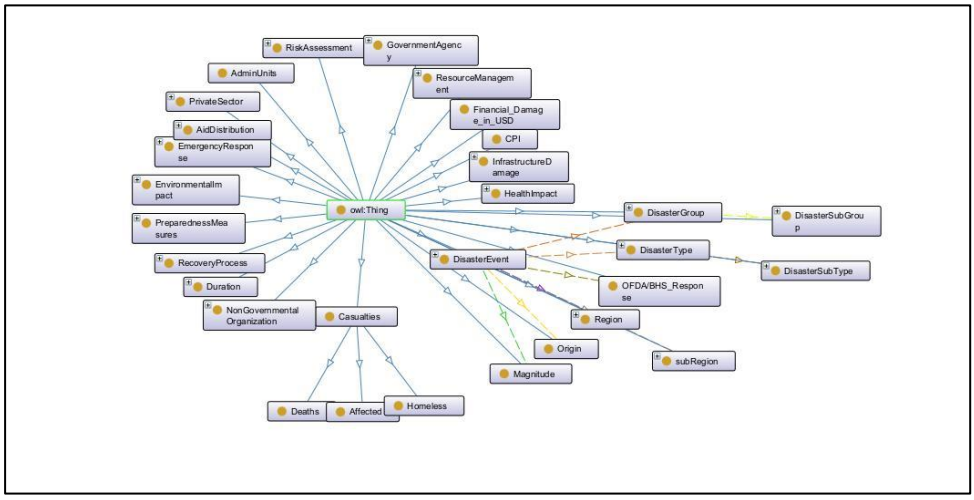


Figure 4 : OntoGraph

In addition to capturing the immediate impact of disasters, the dataset includes classes addressing recovery and preparedness measures, such as resource allocation, recovery

processes, and training programs. The region and subRegion classes, which divide the world into geographic areas like Africa, Asia, and Europe, help contextualize disaster data on a global scale. Key infrastructural and environmental impacts, such as air quality, soil degradation, and water contamination, are also represented, alongside financial aid and resource distribution efforts managed by government agencies, NGOs, and private sectors. By organizing disaster management data in this way, the dataset facilitates comprehensive analysis, offering a foundation for improving disaster response, resource management, and long-term recovery planning.

1.6 Properties

1.6.1 Object Properties

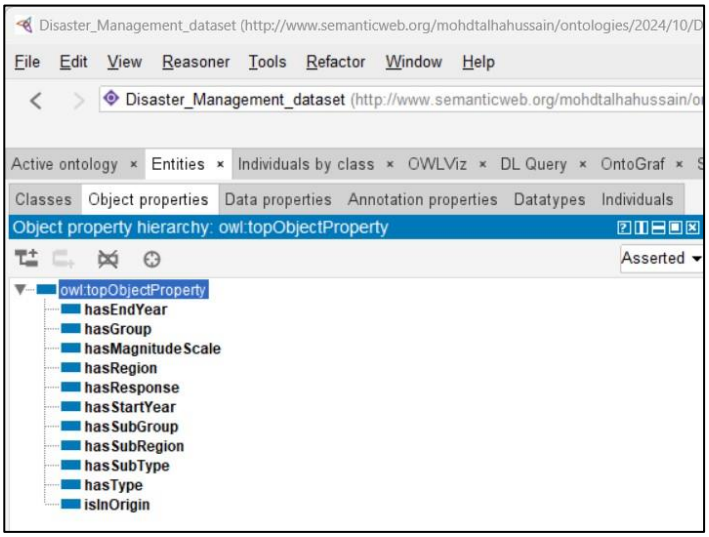


Figure 5 : Object properties

The object properties in the Prodigy for Disaster Management dataset, such as hasEndYear, hasGroup, hasRegion, and hasResponse, define relationships between disaster events and various contextual factors. These properties link disaster events to temporal, geographical, and organizational aspects, enhancing the semantic understanding of the data. For example, hasEndYear associates a disaster event with its conclusion date, while hasGroup categorizes the event under broader disaster types (e.g., natural, technological). Other properties like hasRegion, hasSubRegion, and hasType further enrich the data by connecting the disaster to specific regions, subregions, and its primary type (e.g., flood, earthquake). hasResponse ties the event to the responses and interventions that were implemented, such as rescue operations or medical aid. These object properties help create a connected, detailed model of disaster events, allowing for in-depth analysis of their characteristics, impacts, and responses.

1.6.2 Data Properties

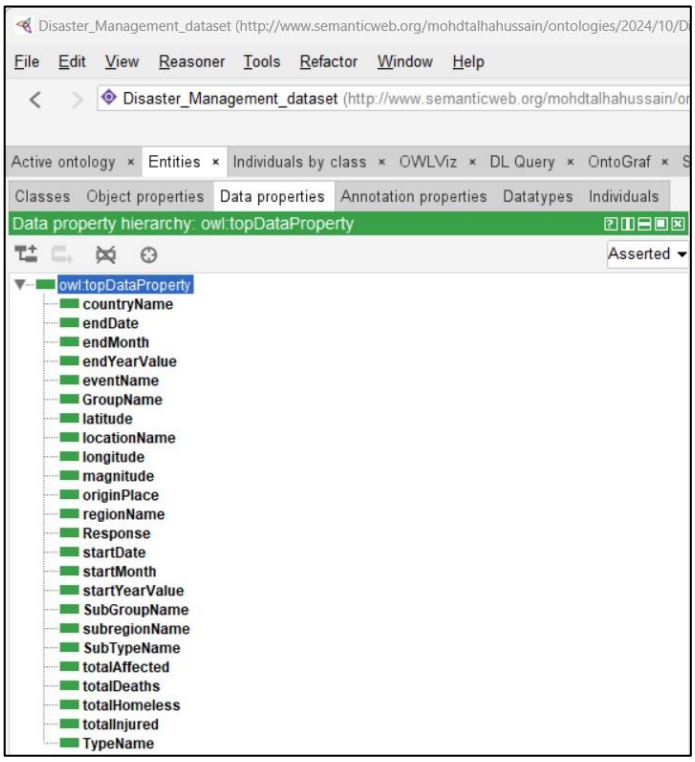


Figure 6 : Data Properties

The data properties, such as countryName, eventName, magnitude, and latitude, capture key factual and quantitative details about each disaster. These properties offer specific data points like the name of the country affected, the event's magnitude, and the geographical coordinates (latitude and longitude) of the origin, providing precise location-based information. Other data properties like startDate, endDate, and startYearValue offer temporal details about the disaster's onset and conclusion, essential for tracking the event's duration and timelines. Similarly, properties like totalAffected, totalDeaths, and totalHomeless quantify the human and infrastructural toll of the disaster. By using these data properties, the system can store and process large volumes of structured information in a way that supports efficient querying and decision-making for disaster management purposes.

1.7 Individuals

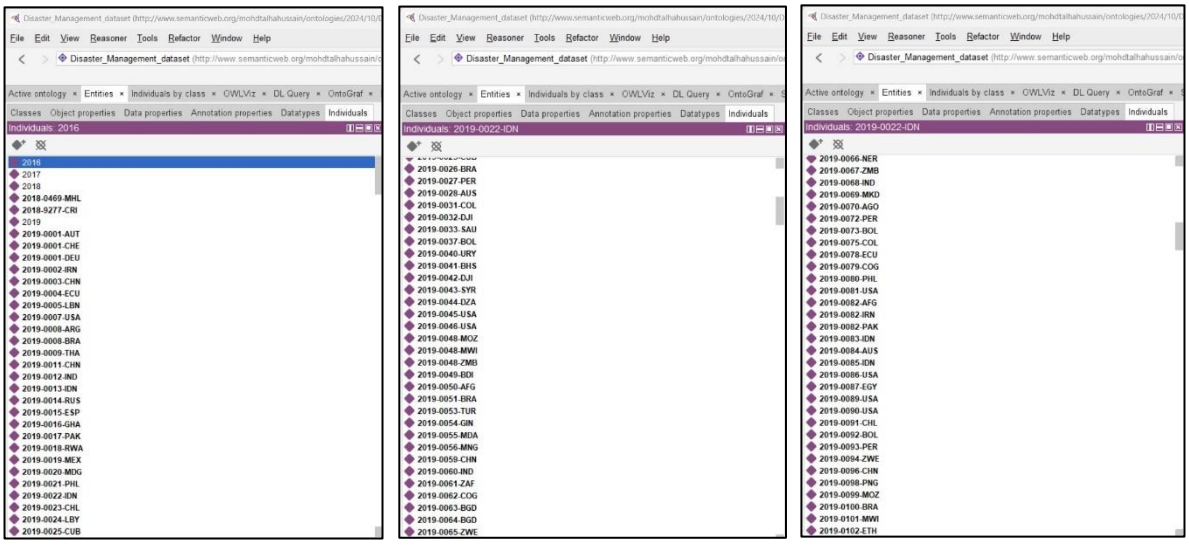


Figure 7 : Individuals

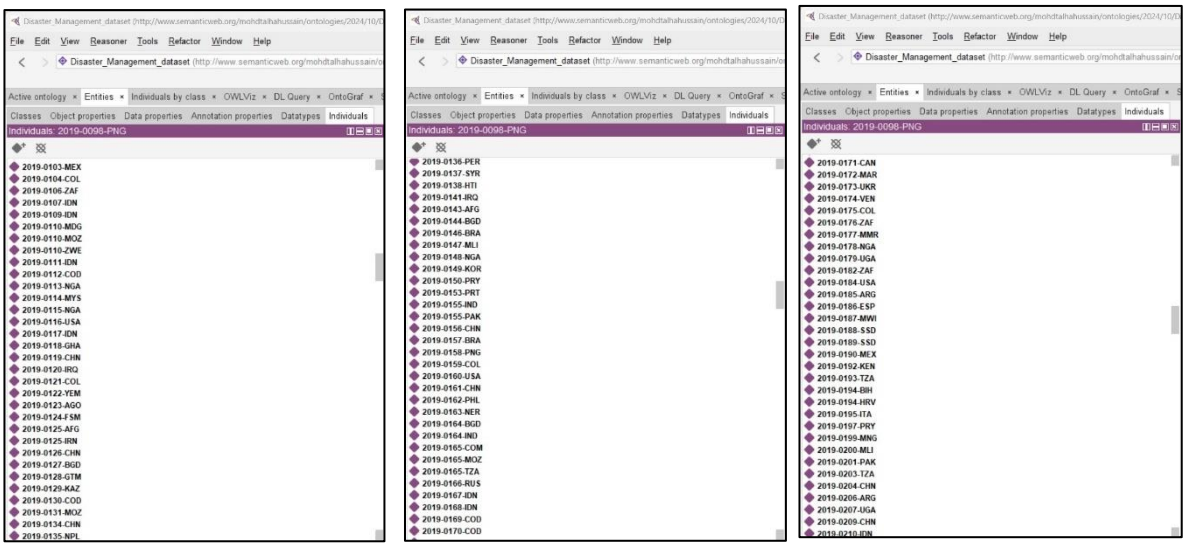


Figure 8 : Individuals

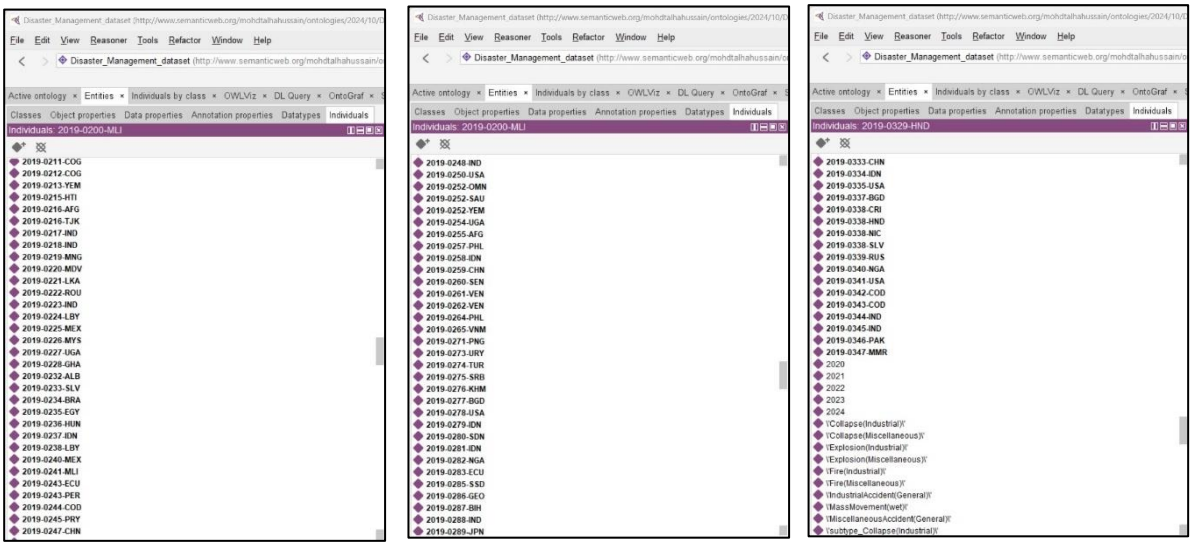


Figure 9 : Individuals

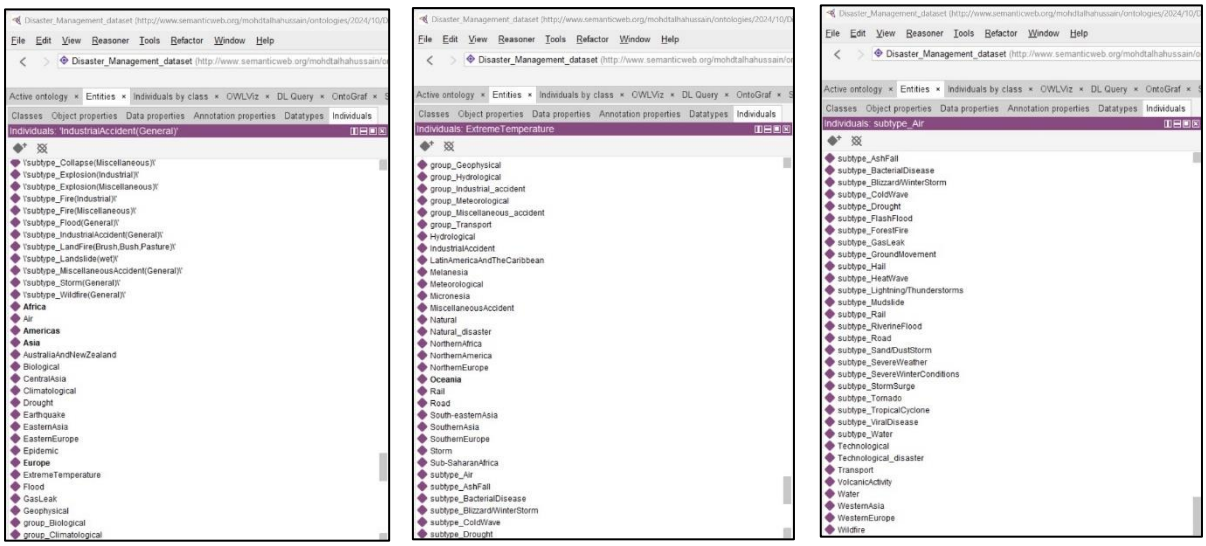


Figure 10 : Individuals

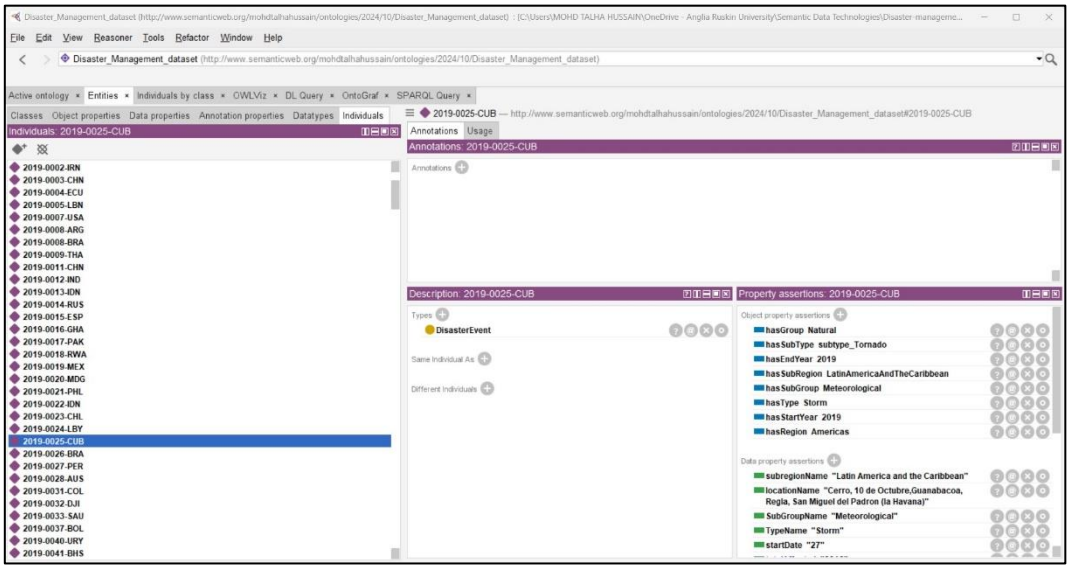


Figure 11 : Individuals

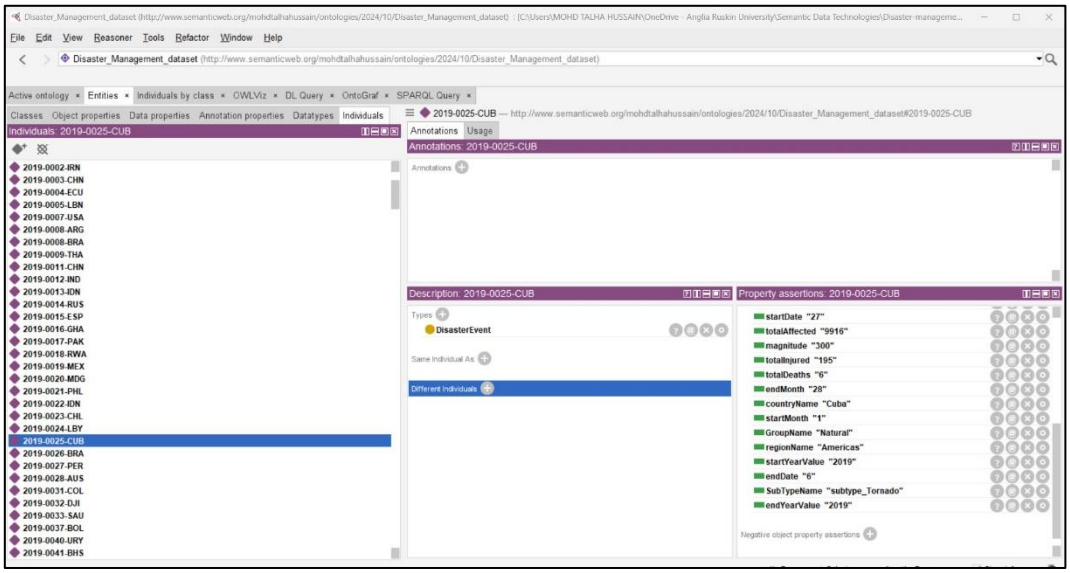
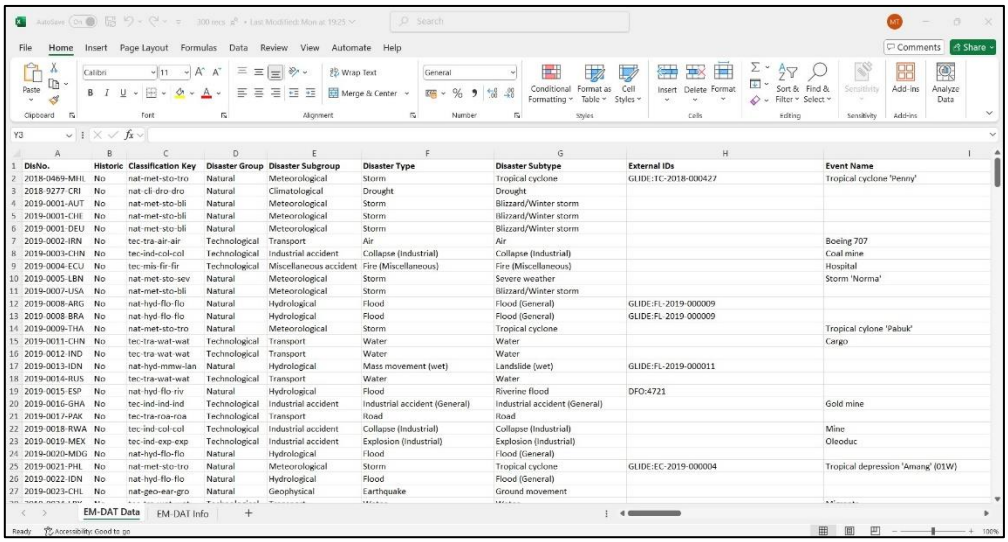


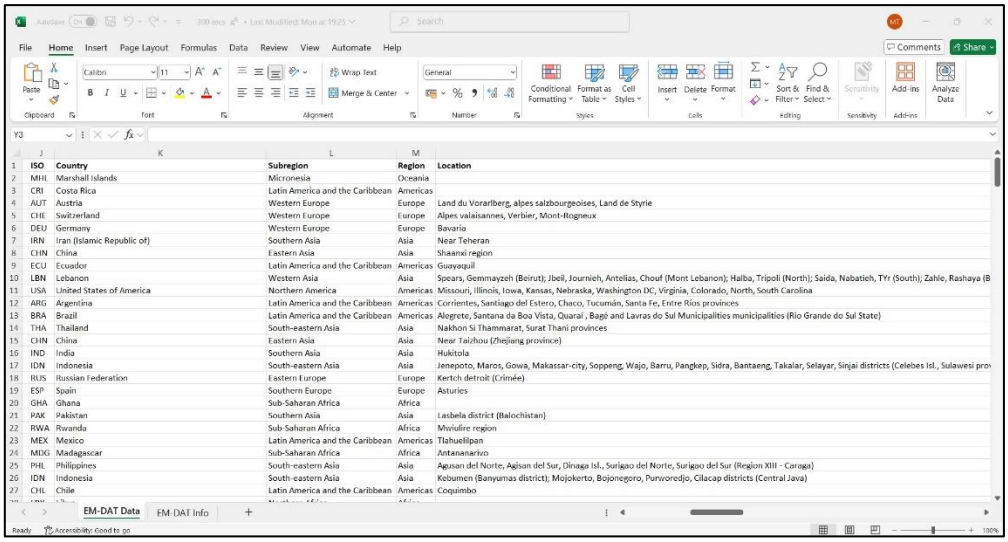
Figure 12: Individuals

The individuals are categorized by their type (e.g., Natural or Technological), with examples such as Floods (Hydrological) or Earthquakes (Geophysical), and further breaks them down by subtypes (e.g., Tropical Cyclones under Meteorological, or Landslides under Geophysical). It also provides information on the group types (e.g., Meteorological or Industrial Accidents) that classify disasters into broader categories, helping to identify the root causes and responses to various disaster scenarios. Furthermore, the dataset includes detailed information on the regions and subregions affected by these events. For example, disasters in Africa (Region) and Southern Africa (Subregion) or in Asia with subregions like Southeast Asia are specified, making it easier to geographically map and analyze disaster trends



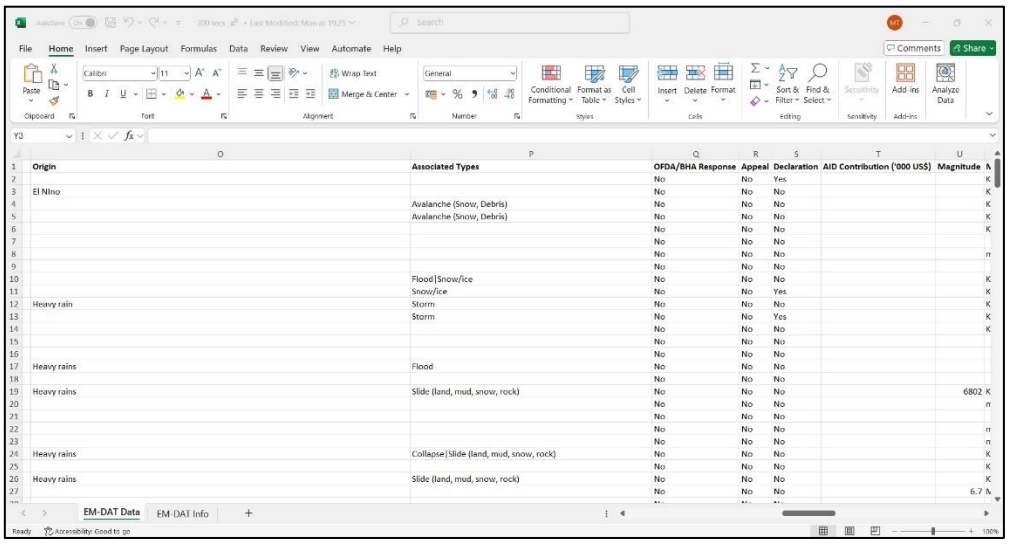
DisNo.	Historic	Classification Key	Disaster Group	Disaster Subgroup	Disaster Type	Disaster Subtype	External IDs	Event Name
2018-0469-MHL	No	nat-met-sto-tro	Natural	Meteorological	Storm	Tropical cyclone	GLIDE:TC-2018-000427	Tropical cyclone 'Penny'
2018-9277-CRI	No	nat-cli-dro-dro	Natural	Climatological	Drought	Drought		
2019-0001-AUT	No	nat-met-sto-bli	Natural	Meteorological	Storm	Blizzard/Winter storm		
2019-0001-CHE	No	nat-met-sto-bli	Natural	Meteorological	Storm	Blizzard/Winter storm		
2019-0001-DEU	No	nat-met-sto-bli	Natural	Meteorological	Storm	Blizzard/Winter storm		
2019-0002-IRN	No	tec-tra-air-air	Technological	Transport	Air	Air		Boeing 707
2019-0003-CHN	No	tec-ind-col-col	Technological	Industrial accident	Collapse (Industrial)	Collapse (Industrial)		Coal mine
2019-0004-ECU	No	tec-mis-fir-fir	Technological	Miscellaneous accident	Fire (Miscellaneous)	Fire (Miscellaneous)		Hospital
2019-0005-LBN	No	nat-met-sto-sev	Natural	Meteorological	Storm	Severe weather		Storm 'Norma'
2019-0007-USA	No	nat-met-sto-bli	Natural	Meteorological	Storm	Blizzard/Winter storm		
2019-0008-ARG	No	nat-hyd-flo-flo	Natural	Hydrological	Flood	Flood (General)	GLIDE:FL-2019-000009	
2019-0008-BRA	No	nat-hyd-flo-flo	Natural	Hydrological	Flood	Flood (General)	GLIDE:FL-2019-000009	
2019-0009-THA	No	nat-met-sto-tro	Natural	Meteorological	Storm	Tropical cyclone		Tropical cylone 'Pabuk'
2019-0011-CHN	No	tec-tra-wat-wat	Technological	Transport	Water	Water		Cargo
2019-0012-IND	No	tec-tra-wat-wat	Technological	Transport	Water	Water		
2019-0013-IDN	No	nat-hyd-mmw-lan	Natural	Hydrological	Mass movement (wet)	Landslide (wet)	GLIDE:FL-2019-000011	
2019-0014-RUS	No	tec-tra-wat-wat	Technological	Transport	Water	Water		
2019-0015-ESP	No	nat-hyd-flo-riv	Natural	Hydrological	Flood	Riverine flood	DFO:4721	
2019-0016-GHA	No	tec-ind-ind-ind	Technological	Industrial accident	Industrial accident (General)	Industrial accident (General)		Gold mine
2019-0017-PAK	No	tec-tra-roa-roa	Technological	Transport	Road	Road		
2019-0018-RWA	No	tec-ind-col-col	Technological	Industrial accident	Collapse (Industrial)	Collapse (Industrial)		Mine
2019-0019-MEX	No	tec-ind-exp-exp	Technological	Industrial accident	Explosion (Industrial)	Explosion (Industrial)		Oleoduc
2019-0020-MDG	No	nat-hyd-flo-flo	Natural	Hydrological	Flood	Flood (General)		
2019-0021-PHL	No	nat-met-sto-tro	Natural	Meteorological	Storm	Tropical cyclone	GLIDE:EC-2019-000004	Tropical depression 'Amang' (01W)
2019-0022-IDN	No	nat-hyd-flo-flo	Natural	Hydrological	Flood	Flood (General)		
2019-0023-CHL	No	nat-geo-ear-gro	Natural	Geophysical	Earthquake	Ground movement		

Figure 13 : Excel data



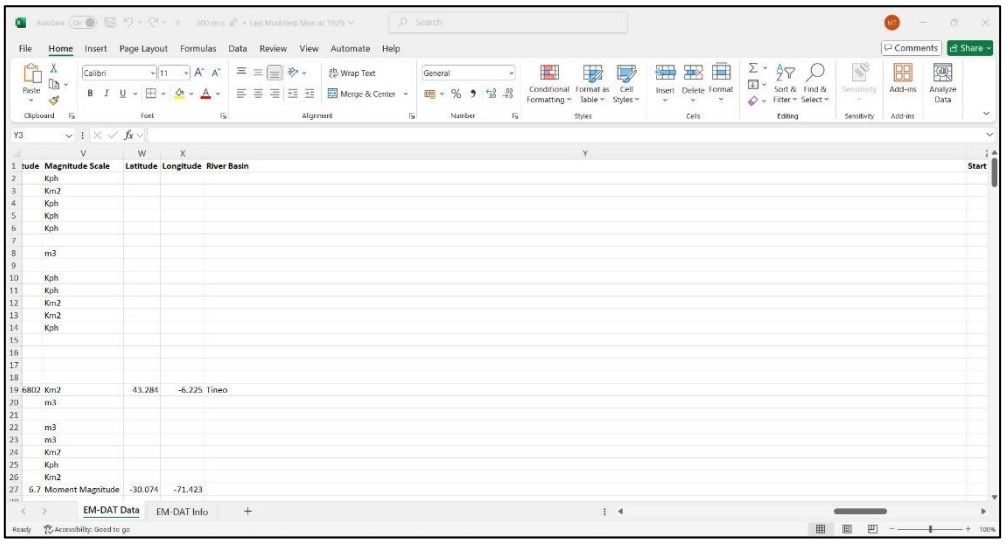
ISO	Country	Subregion	Region	Location
MHL	Marshall Islands	Micronesia	Oceania	
CRI	Costa Rica	Latin America and the Caribbean	Americas	
AUT	Austria	Western Europe	Europe	Land du Vorarlberg, alpes salzbourgeoises, Land de Styrie
CHE	Switzerland	Western Europe	Europe	Alpes valaisannes, Verbier, Mont-Rogneux
DEU	Germany	Western Europe	Europe	Bavaria
IRN	Iran (Islamic Republic of)	Southern Asia	Asia	Near Teheran
CHN	China	Eastern Asia	Asia	Shaanxi region
ECU	Ecuador	Latin America and the Caribbean	Americas	Guayaquil
LBN	Lebanon	Western Asia	Asia	Spears, Gemmayzeh (Beirut); Jbeil, Journieh, Antelias, Chouf (Mont Lebanon); Halba, Tripoli (North); Saida, Nabatieh, TYr (South); Zahle, Rashaya (B
USA	United States of America	Northern America	Americas	Missouri, Illinois, Iowa, Kansas, Nebraska, Washington DC, Virginia, Colorado, North, South Carolina
ARG	Argentina	Latin America and the Caribbean	Americas	Corrientes, Santiago del Estero, Chaco, Tucumán, Santa Fe, Entre Ríos provinces
BRA	Brazil	Latin America and the Caribbean	Americas	Alegrete, Santana da Boa Vista, Quaraí , Bagé and Lavras do Sul Municipalities municipalities (Rio Grande do Sul State)
THA	Thailand	South-eastern Asia	Asia	Nakhon Si Thammarat, Surat Thani provinces
CHN	China	Eastern Asia	Asia	Near Taizhou (Zhejiang province)
IND	India	Southern Asia	Asia	Hukitola
IDN	Indonesia	South-eastern Asia	Asia	Jenepoto, Maros, Gowa, Makassar-city, Soppeng, Wajo, Barru, Pangkep, Sidra, Bantaeng, Takalar, Selayar, Sinjai districts (Celebes Isl., Sulawesi prov
RUS	Russian Federation	Eastern Europe	Europe	Kertch detroit (Crimée)
ESP	Spain	Southern Europe	Europe	Asturies
GHA	Ghana	Sub-Saharan Africa	Africa	
PAK	Pakistan	Southern Asia	Asia	Lasbela district (Balochistan)
RWA	Rwanda	Sub-Saharan Africa	Africa	Mviulire region
MEX	Mexico	Latin America and the Caribbean	Americas	Tlahuelilpan
MDG	Madagascar	Sub-Saharan Africa	Africa	Antananarivo
PHL	Philippines	South-eastern Asia	Asia	Agusan del Norte, Agisan del Sur, Dinaga Isl., Surigao del Norte, Surigao del Sur (Region XIII - Caraga)
IDN	Indonesia	South-eastern Asia	Asia	Kebumen (Banyumas district); Mojokerto, Bojonegoro, Purworedjo, Cilacap districts (Central Java)
CHL	Chile	Latin America and the Caribbean	Americas	Coquimbo

Figure 14: Excel data



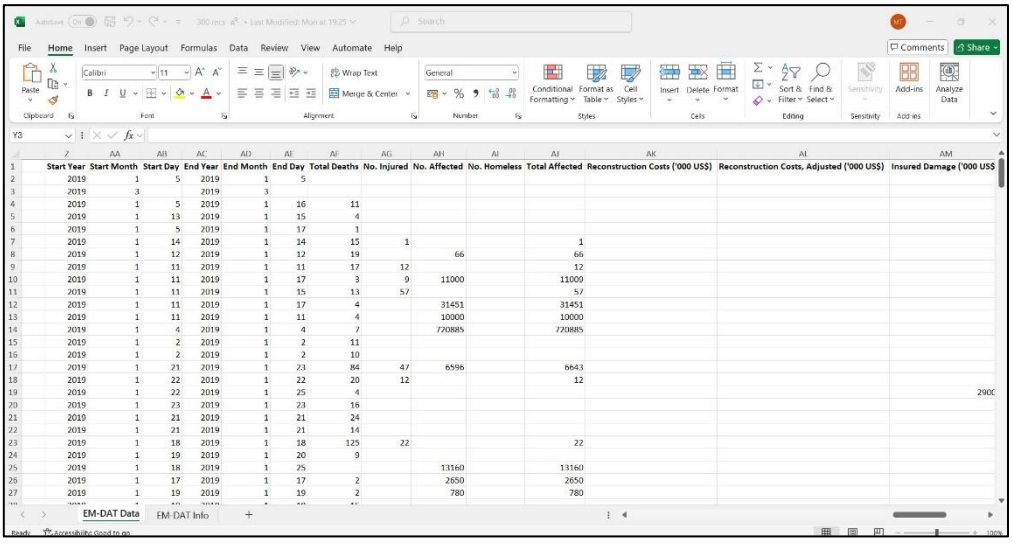
	O	P	Q	R	S	T	U
1	Origin	Associated Types	OFDA/BHA Response	Appeal	Declaration	AID Contribution ('000 US$)	Magnitude
2			No	No	Yes		K
3	El Nino		No	No	No		K
4		Avalanche (Snow, Debris)	No	No	No		K
5		Avalanche (Snow, Debris)	No	No	No		K
6			No	No	No		K
7			No	No	No		
8			No	No	No		n
9			No	No	No		
10		Flood|Snow/Ice	No	No	No		K
11		Snow/Ice	No	No	Yes		K
12	Heavy rain	Storm	No	No	No		K
13		Storm	No	No	Yes		K
14			No	No	No		K
15			No	No	No		
16			No	No	No		
17	Heavy rains	Flood	No	No	No		
18			No	No	No		
19	Heavy rains	Slide (land, mud, snow, rock)	No	No	No		6802 K
20			No	No	No		n
21			No	No	No		
22			No	No	No		n
23			No	No	No		n
24	Heavy rains	Collapse|Slide (land, mud, snow, rock)	No	No	No		K
25			No	No	No		K
26	Heavy rains	Slide (land, mud, snow, rock)	No	No	No		K
27			No	No	No		6.7 M

Figure 15 : Excel data



	V	W	X	Y	
1	tude Magnitude Scale	Latitude	Longitude	River Basin	Start
2	Kph				
3	Km2				
4	Kph				
5	Kph				
6	Kph				
7					
8	m3				
9					
10	Kph				
11	Kph				
12	Km2				
13	Km2				
14	Kph				
15					
16					
17					
18					
19	6802 Km2	43.284	-6.225	Tineo	
20	m3				
21					
22	m3				
23	m3				
24	Km2				
25	Kph				
26	Km2				
27	6.7 Moment Magnitude	-30.074	-71.423		

Figure 16 : Excel data



	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM
1	Start Year	Start Month	Start Day	End Year	End Month	End Day	Total Deaths	No. Injured	No. Affected	No. Homeless	Total Affected	Reconstruction Costs ('000 US$)	Reconstruction Costs, Adjusted ('000 US$)	Insured Damage ('000 US$
2	2019	1	5	2019	1	5								
3	2019	3		2019	3									
4	2019	1	5	2019	1	16	11							
5	2019	1	13	2019	1	15	4							
6	2019	1	5	2019	1	17	1							
7	2019	1	14	2019	1	14	15	1			1			
8	2019	1	12	2019	1	12	19		66		66			
9	2019	1	11	2019	1	11	17	12			12			
10	2019	1	11	2019	1	17	3	9	11000		11009			
11	2019	1	11	2019	1	15	13	57			57			
12	2019	1	11	2019	1	17	4		31451		31451			
13	2019	1	11	2019	1	11	4		10000		10000			
14	2019	1	4	2019	1	4	7		720885		720885			
15	2019	1	2	2019	1	2	11							
16	2019	1	2	2019	1	2	10							
17	2019	1	21	2019	1	23	84	47	6596		6643			
18	2019	1	22	2019	1	22	20	12			12			
19	2019	1	22	2019	1	25	4							2900
20	2019	1	23	2019	1	23	16							
21	2019	1	21	2019	1	21	24							
22	2019	1	21	2019	1	21	14							
23	2019	1	18	2019	1	18	125	22			22			
24	2019	1	19	2019	1	20	9							
25	2019	1	18	2019	1	25			13160		13160			
26	2019	1	17	2019	1	17	2		2650		2650			
27	2019	1	19	2019	1	19	2		780		780			

Figure 17 : Excel data

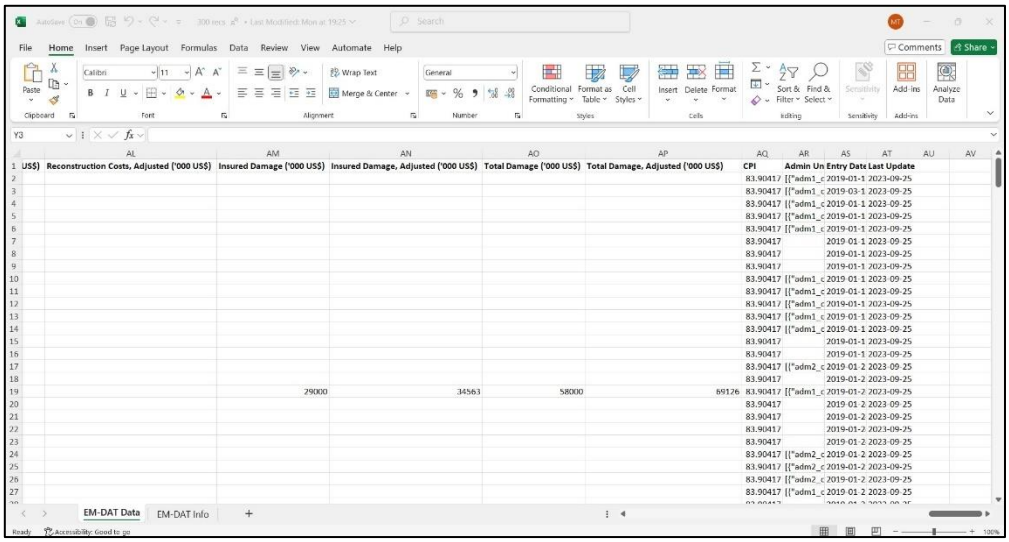


Figure 18: Excel sheets

The provided dataset offers a comprehensive and organized collection of disaster-related data, spanning both natural and technological events. The various columns within the dataset include essential details about individual disasters, such as unique identifiers, event names, disaster types and subtypes, external IDs, and specific classifications. For example, the dataset covers a broad spectrum of disasters, from storms and floods to industrial accidents and earthquakes, capturing critical information like the triggering factors, magnitude, and geographic details (such as latitude and longitude). Additional columns record the associated disaster responses, including declarations and aid contributions, which can aid in evaluating disaster preparedness and response efforts. This structured data is valuable for researchers, policymakers, and emergency responders, enabling them to analyze disaster patterns, frequencies, and severity, and ultimately enhance decision-making for future events.

In addition to disaster-specific data, the dataset also includes geographic information, linking countries with their corresponding regions and subregions, further contextualizing the disasters in relation to their global and local locations. This geographic data, along with information on disaster impacts, such as deaths, injuries, affected populations, and reconstruction costs, serves as a crucial resource for various analyses. By organizing this data into tabular format, the dataset provides a practical tool for identifying trends, mapping disaster occurrence, and assessing the long-term effects of various events. With columns detailing damage costs, insured damage, and other economic factors, this data can be leveraged to study economic losses and the effectiveness of disaster management strategies, supporting better preparedness and response strategies across regions.

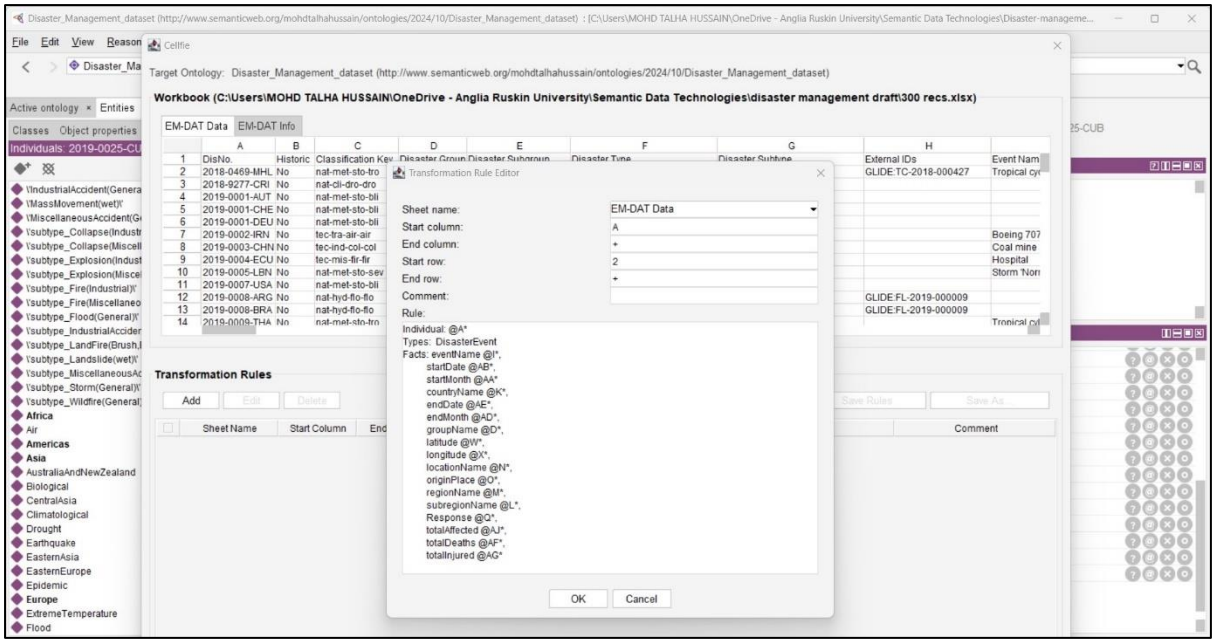


Figure 19: Rules

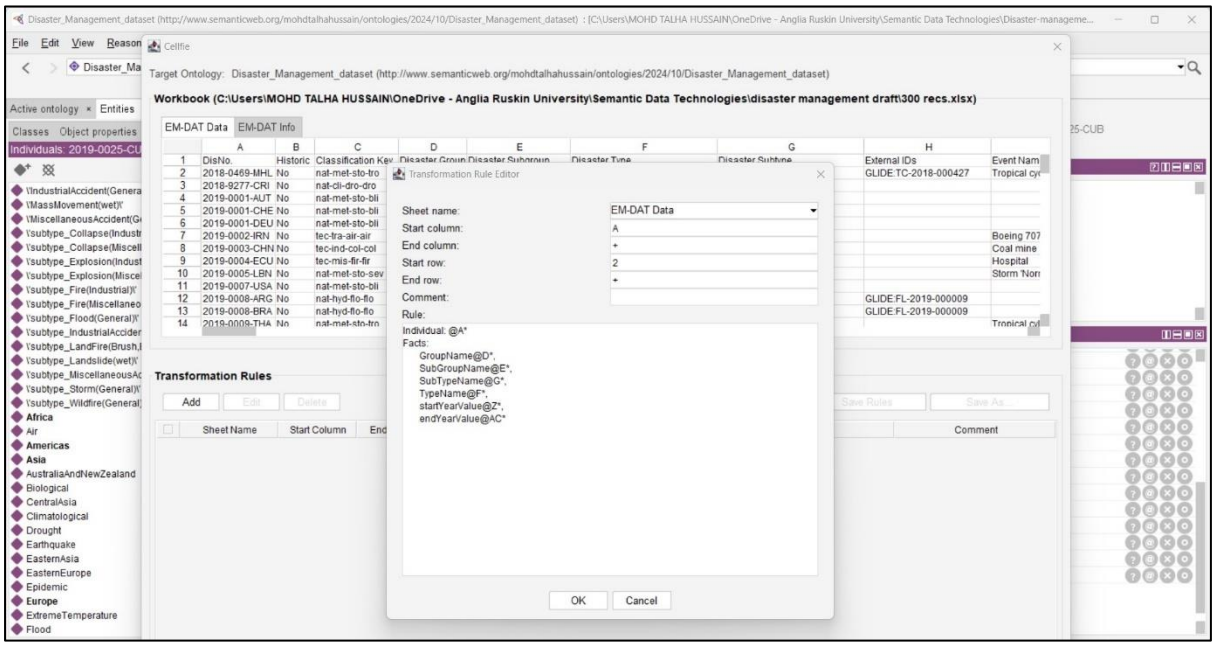


Figure 20: Rules

The provided rules outline the process of importing disaster-related data into Protégé using the Celfie plugin, a tool that facilitates the conversion of spreadsheet data into a structured ontology format. The "Individual" refers to a specific disaster event or entity, with a unique identifier denoted as @A*, representing a distinct disaster. The dataset consists of various facts associated with each disaster, including key attributes like the eventName (e.g., "Hurricane Katrina"), startDate (e.g., "2005-08-29"), and countryName (e.g., "USA"). These data points are mapped to the corresponding fields in Protégé, ensuring that each disaster event is properly represented within the ontology. Additionally, geographical information such as latitude and longitude (e.g., latitude: 30.6954, longitude: -89.8496) along with the regionName (e.g., "North America") and

subregionName (e.g., "United States") is included, allowing for precise mapping and analysis of disaster events in relation to global regions.
Furthermore, the rules include metadata related to the disaster's severity, such as the totalAffected, totalDeaths, and totalInjured (e.g., total affected: 1,000,000 people). The dataset also captures the Response to the disaster, such as relief efforts by agencies (e.g., Red Cross). Additional facts like GroupName (e.g., "Meteorological"), SubGroupName (e.g., "Cyclone"), and SubTypeName (e.g., "Hurricane") are essential for categorizing disasters into broader groupings and types, facilitating in-depth analysis and classification. The startYearValue (e.g., "2005") and endYearValue (e.g., "2006") are also critical for tracking the timeline of the disaster event. These rules ensure that each disaster event is systematically represented, organized by relevant categories, and ready for further analysis in Protégé, making it a powerful tool for disaster research and response planning.

1.8 Axioms

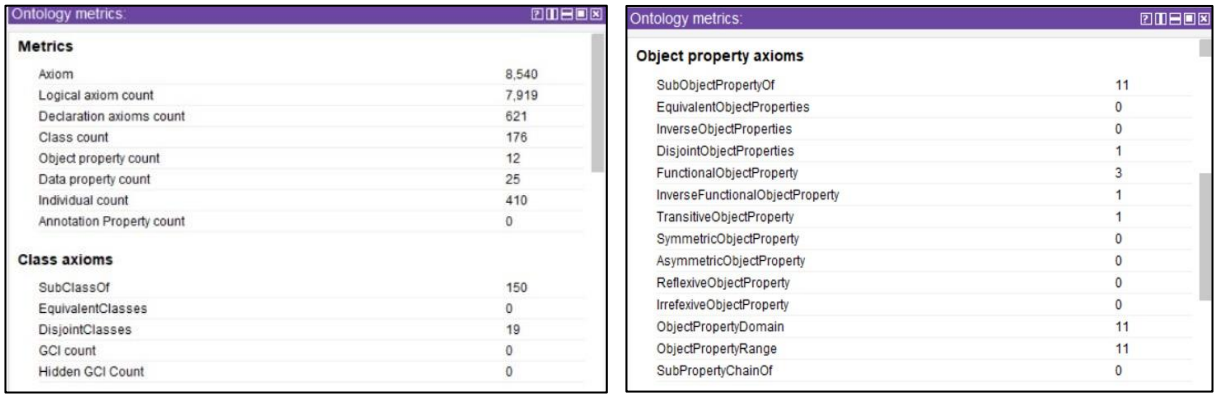


Figure 21: Axioms

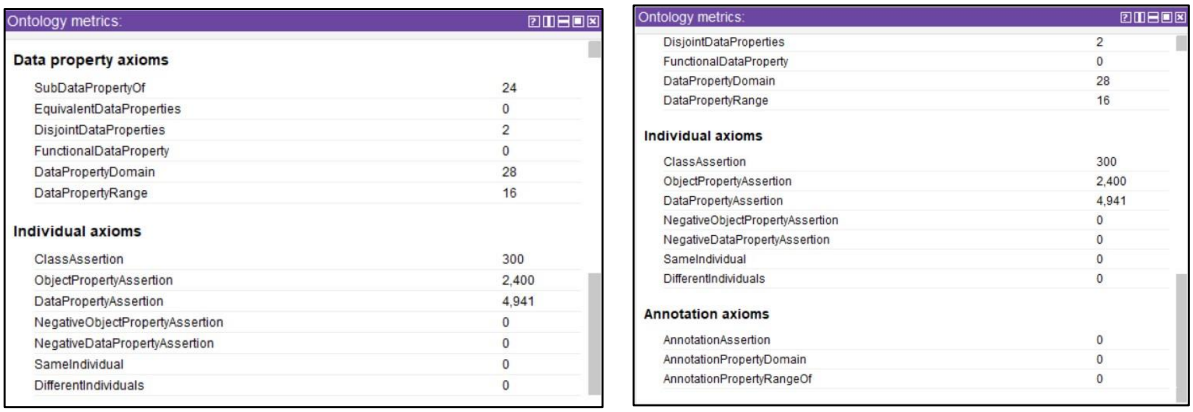


Figure 22: Axioms

The pictures represent various axioms in the ontology. The ontology contains 11 SubObjectPropertyOf axioms and a single count for several other properties, including DisjointObjectProperties, FunctionalObjectProperty, InverseFunctionalObjectProperty, and TransitiveObjectProperty. For data property axioms, there are 24 SubDataPropertyOf axioms, 2 DisjointDataProperties, 28 DataPropertyDomain axioms, and 16 DataPropertyRange axioms. The ontology's individual axioms include a

substantial number of ObjectPropertyAssertion (2,400) and DataPropertyAssertion (4,941), alongside 300 ClassAssertions. Annotation axioms are not used in this ontology, as indicated by their zero counts. Class axioms reveal 150 SubClassOf relationships and 19 DisjointClasses with no equivalent classes or general concept inclusions (GCI).

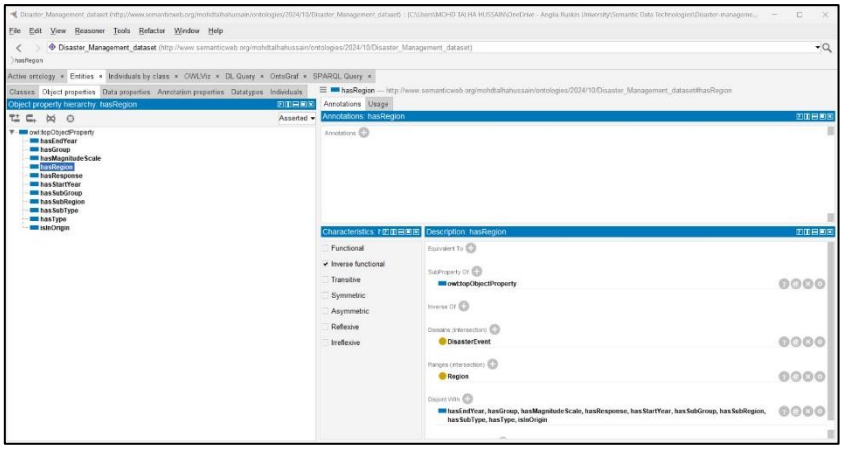


Figure 23: Axioms

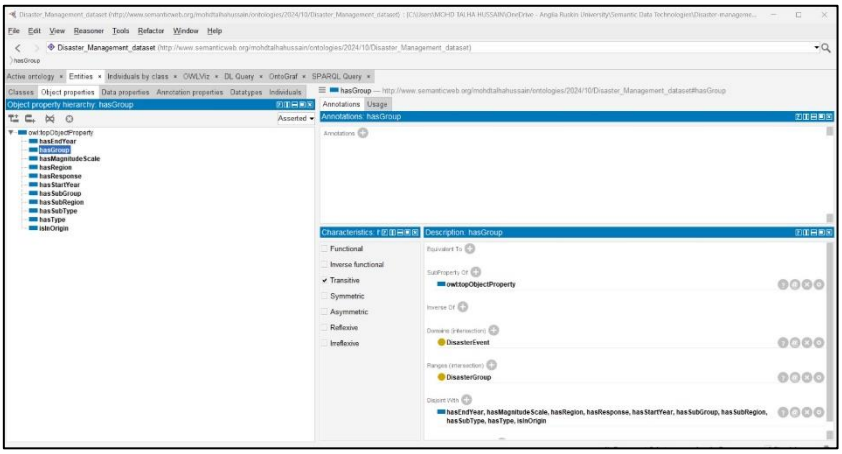


Figure 24: Axioms

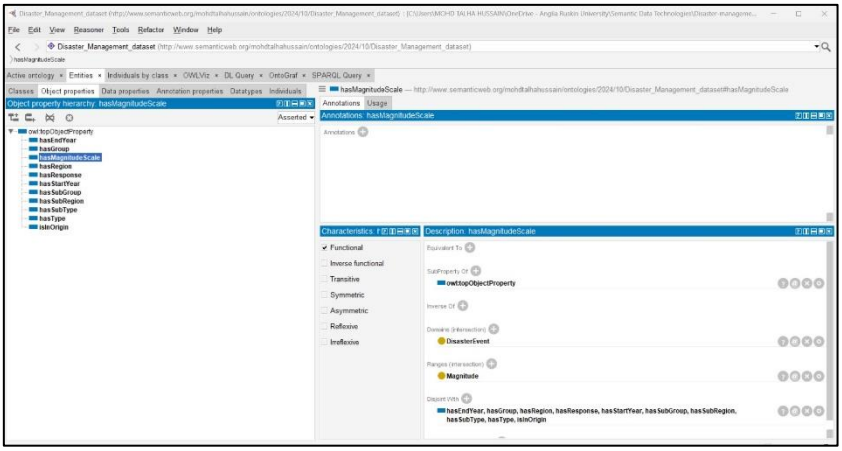


Figure 25: Axioms

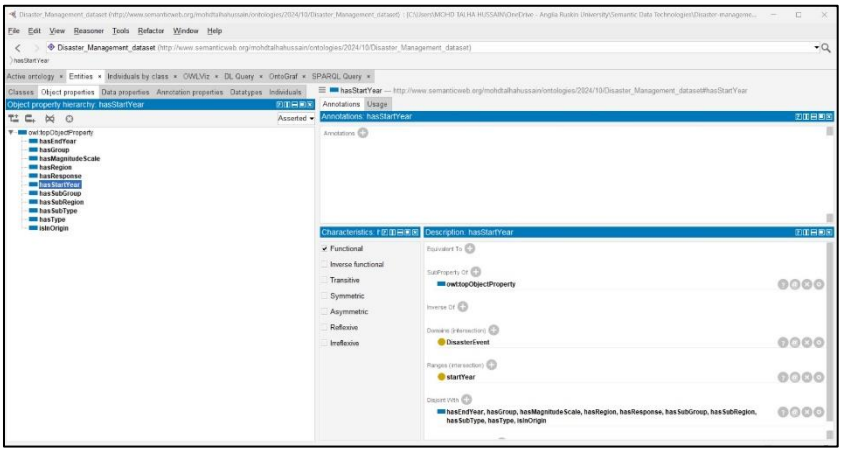


Figure 26: Axioms

The properties such as hasStartYear, hasRegion, and hasMagnitudeScale has its specific characteristics and domains. The hasStartYear property links a DisasterEvent to its starting year, marked as Functional, ensuring each event has a single start year. The hasRegion property connects a DisasterEvent to a Region and is Inverse Functional, signifying that a specific region corresponds uniquely to one disaster event. Similarly, hasMagnitudeScale relates a DisasterEvent to its Magnitude, also defined as Functional, allowing only one magnitude per event. These properties, with well-defined domains, ranges, and disjoint relationships, maintain logical consistency, supporting accurate semantic modeling for disaster analysis.

4. Requirements Mapping

1.9Mapping Table

Question	**Mapping**
1. Display the number of deaths and injuries caused by the Viral diseases in Asia?	This question can be answered using the DisasterEvent, DisasterSubtype class and data properties (eventName, regionName, totalDeaths, SubTypeName, totalInjured).
2. List the number of disasters occurred in each region?	This question can be answered using the DisasterEvent, Region class and data properties (regionName).
3. What are the top 3 types of disasters with the highest magnitude?	This question can be answered using the DisasterEvent class and data properties (eventName, magnitude).
4. What is the total number of people left homeless due to disasters?	This question can be answered using the DisasterEvent, Casualties class and data properties (totalHomeless).

5. List the 3 major disasters of 2022 with their countries?	This question can be answered using the DisasterEvent class and data properties (eventName, startYearValue, countryName).
6. Which disasters have been responded by agencies ?	This question can be answered using the DisasterEvent, Response class and data properties (OFDA response).
7. List all disaster events with their names, start years, and end years ?	This question can be answered using the DisasterEvent, Duration class and data properties (eventName, startYearValue, endYearValue).
8. Which disaster types have the highest average number of affected individuals?	This question can be answered using the DisasterEvent , Casualties class and data properties (eventName, startYearValue, endYearValue).
9. Which are the most recent disaster events, along with their names and end years?	This question can be answered using the DisasterEvent class and data properties (eventName and endYearValue)
10. Which disaster events, along with their names, have overlapping time periods?	This question can be answered using DisasterEvent class and data properties (eventName, hasStartYear, endYearValue)
11. Which regions and subregions have the highest number of disaster events?	This question can be answered using DisasterEvent, Region class and data Properties (regionName, subRegionName)
12. Which pairs of disaster types overlap in their timeframes?	This question can be answered using disasterEvent, Duration class and data properties (startYearValue, endYearValue)
13. Identify Most Common Disaster Types by Region and Year ?	This question can be answered using the DisasterEvent, Region, Type class and the data properties regionName, startYearValue, and TypeName.
14. Most Recent Disaster for Each Disaster SubType ?	This question can be answered using the DisasterSubType and DisasterEvent classes, the data properties SubTypeName and startYearValue.
15. Retrieve Disaster Events with Multiple Locations and Cross-Region Impacts.	This question can be answered using the DisasterEvent class and the data properties eventName, locationName, and regionName.
16. Find the Third Most Recent Disaster by SubType.	This question can be answered using the DisasterEvent and DisasterSubType classes, the data properties SubTypeName, eventName, and

	startYearValue, and the object property hasSubType.
17. Identify Regions with the Highest Casualties.	This question can be answered using the Region, Casualities class and the data properties regionName, totalDeaths, and totalInjured.
18. List Disaster Events with Total Regions Affected and Region Names.	This question can be answered using the DisasterEvent class and the data properties eventName and regionName.

Table 2: Requirement Mapping

5. Evaluation and Use

4.1 Use case 1

Question:
Display the number of deaths and injuries caused by the Viral diseases in Asia?

SPARQL Code and Output:

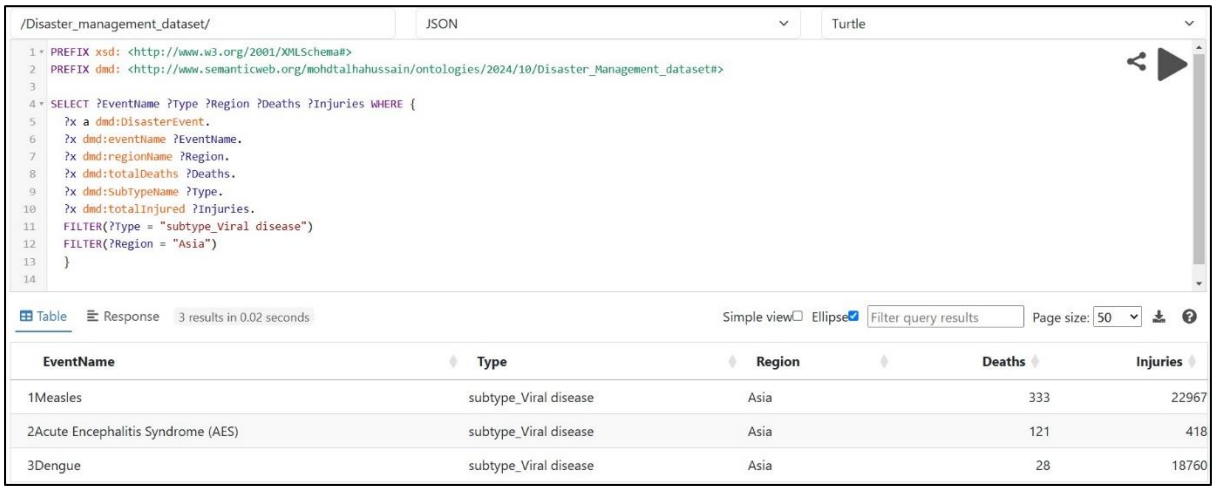


Figure 27: SPARQL query and result

Explanation:
To answer this query, DisasterEvent instances can be filtered by the region Asia and the subtype Viral. Then, the values of totalDeaths and totalInjured are counted for all matching events. The result gives the total number of deaths and injuries caused by viral diseases in Asia.

4.2 Use case 2

Question:
List the number of disasters occurred in each region?

SPARQL Code and Output:

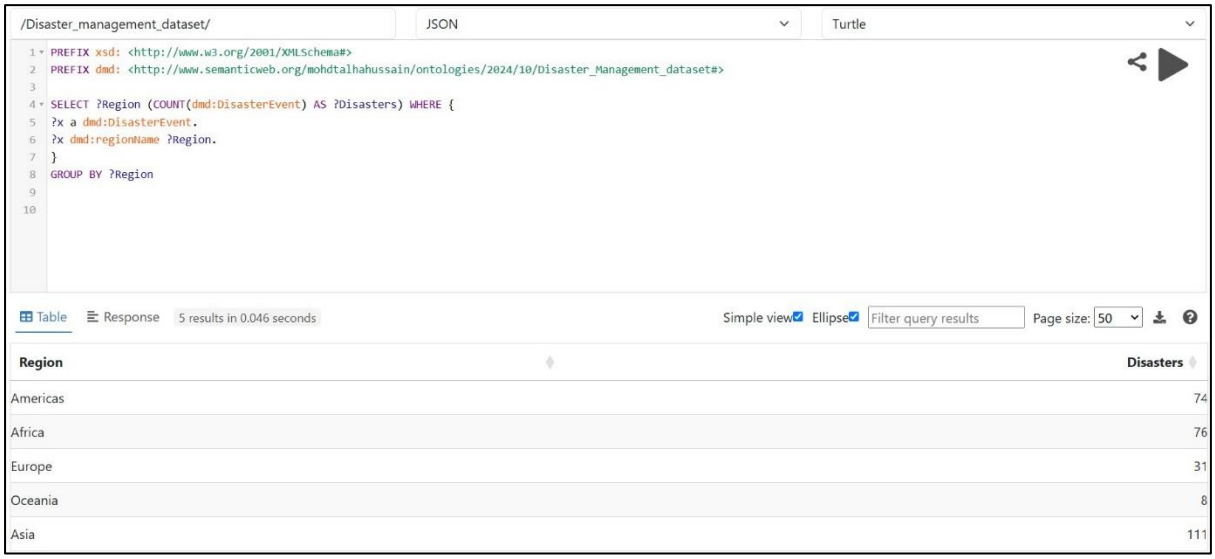


Figure 28: SPARQL query and result

Explanation:

To answer this query, we would group the DisasterEvent instances by the regionName property. Then, we count the number of disasters (represented by eventName) that occurred in each region are counted. This the count of how many disasters occurred in each region.

4.3 Use case 3

Question:

Name the top 3 types of disasters and their locations with highest magnitude?

SPARQL Code and Output:

Figure 29: SPARQL query and result

Explanation:

To answer this query, DisasterEvent instances are filtered and ordered by the magnitude property in descending order. The LIMIT 3 clause would ensure that only the top 3 disasters with the highest magnitude are returned resulting most severe disasters based on their magnitude.

4.4 Use case 4

Question:

What is the total number of people left homeless due to disasters ?

SPARQL Code and Output:

Figure 30: SPARQL query and result

Explanation:

To answer this query, the values of the totalHomeless property are summed across all DisasterEvent instances, giving the total number of people who became homeless due to disasters. The query would essentially aggregate the total number of homeless individuals across all events.

4.5 Use case 5

Question:

List the 3 major disasters of 2022 with their countries?

SPARQL Code and Output:

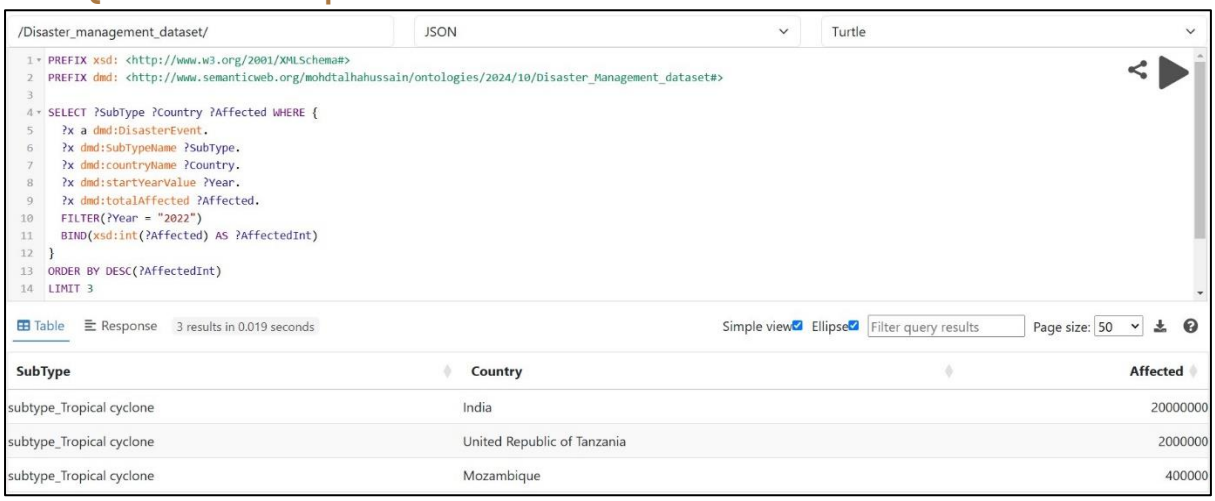


Figure 31: SPARQL query and result

Explanation:

This query is answered by filtering the DisasterEvent instances by the startYearValue property, looking for events that started in 2022. The results are sorted based on

totalAffected. To identify the "major" disasters of that year. Using LIMIT 3 ensures that only the top 3 disasters are returned.

6. Implementation

1.10 Connection to the Dataset:

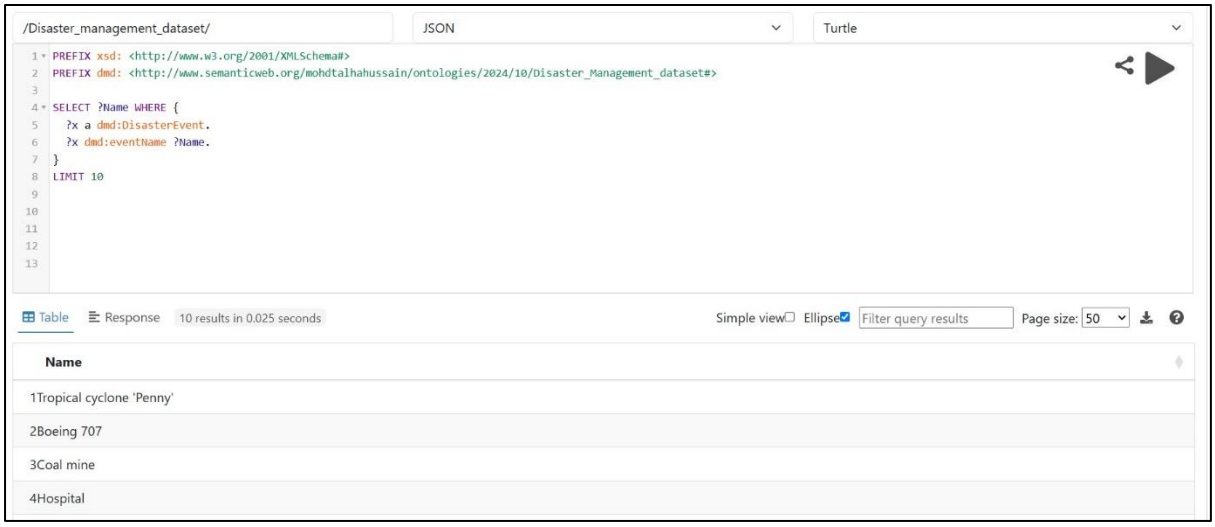


Figure 31: Fuseki server

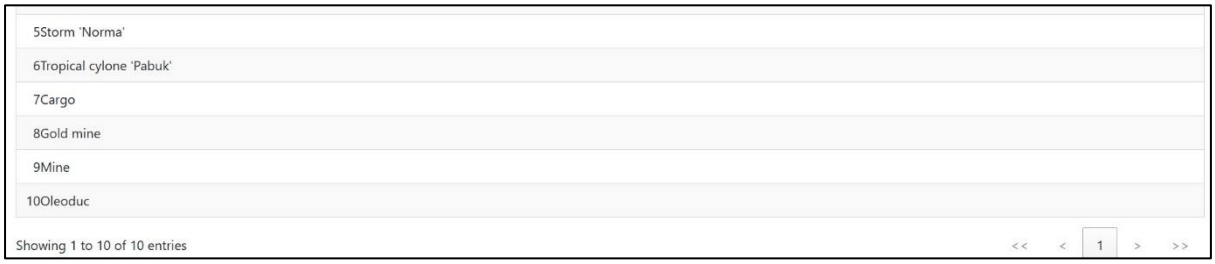


Figure 32: Fuseki server

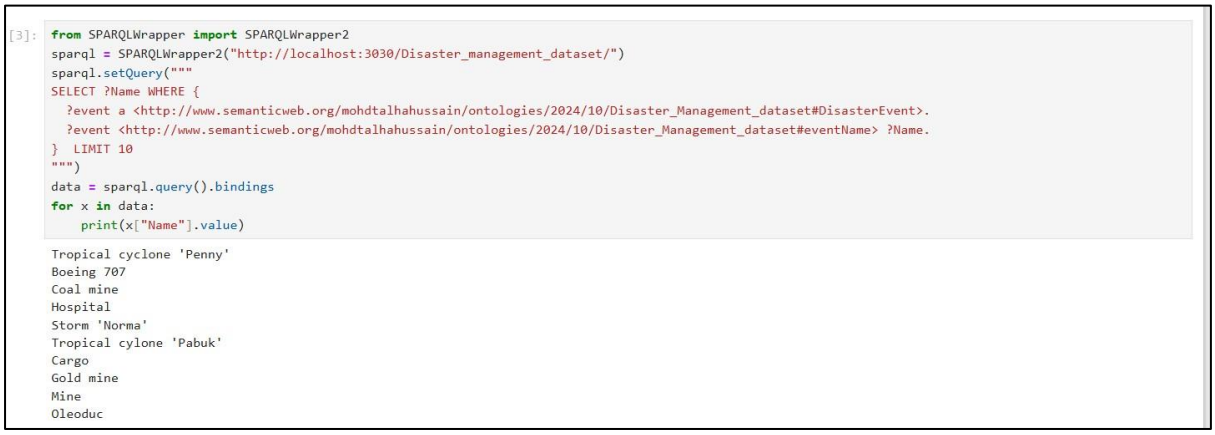


Figure 33: Jupyter Notebook

The code interacts with a local SPARQL endpoint that holds a disaster management dataset, the connection is established by implementing Python library called SPARQLWrapper2.The core of the operation is a SPARQL query, which is a query

language for querying datasets stored in the RDF (Resource Description Framework) format. The query is designed to retrieve the names of disaster events from the dataset. Specifically, it selects DisasterEvents and extracts the name of each event.After setting up the query, the code sends it to the SPARQL endpoint. The results of the query are returned as a collection of records, each containing a disaster event's name.The code then iterates through these results and prints the names of the events. It fetches and displays the first ten event names from the dataset. The output prints the names of various disaster events, which includes entries like "Tropical cyclone 'Penny'," "Storm 'Norma'," "Boeing 707," and others.

1.11 NLP Techniques:

5.2.1 Normalization

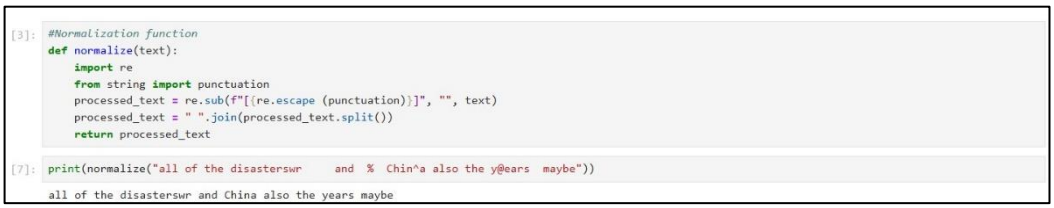


Figure 34: Normalization

Normalization cleans text by performing two main tasks. First, it removes all punctuation marks from the input string using the re.sub() function, which eliminates symbols like %, ^, and @. For example, the input "all of the disasterswr and % Chin^a also the y@ears maybe" becomes "all of the disasterswr and China also the years maybe and Chin a also the years maybe". Second, the function normalizes whitespace by splitting the text into words and then joining them back together with a single space, ensuring uniform spacing. After both operations, the cleaned text is returned with punctuation removed and spaces normalized, making it easier to process. In the example, the final output is "all of the disasters and China also the years maybe ".

5.2.2 Tokenization

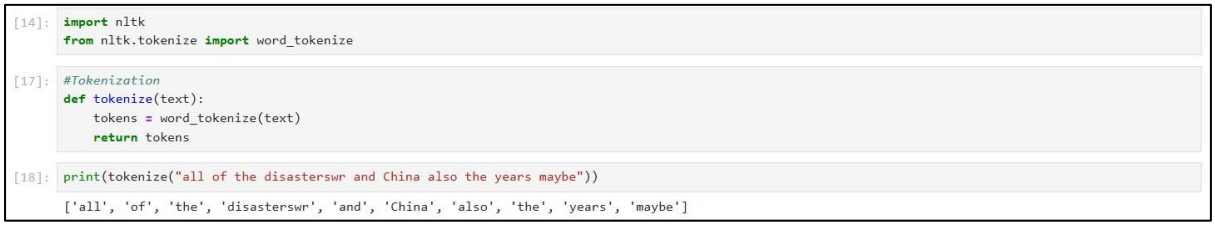


Figure 35: Tokenization

Tokenization means to break a text into smaller parts like words. The tokenize function takes a string "all of the disasterswr and China also the years maybe" as input and splits into individual words, resulting in ['all', 'of', 'the', 'disasterswr', 'and', 'China', 'also', 'the', 'years', 'maybe']. This individual components are then utilized for further natural language processes.

5.2.3 Stop Word Removal

```
[19]: import nltk
      #nltk.download('stopwords')
      from nltk.corpus import stopwords
      stop_words = set(stopwords.words('english'))
      print(stop_words)

      {'after', 'his', 'does', 'having', 'both', 'weren', 'be', "it's", 'are', 'himself', 'or', 'myself', 'no', 're', 'he', 'it', 'some', 'wasn', "you're", 'yo
      urselves', 's', 'themselves', 'should', 'them', 'mightn', 'theirs', 'she', 'they', 'won', "aren't", 'as', 'you', 'if', "you've", 'because', 'against', 't
      hen', 'any', 'each', 'i', 'other', 'below', 'being', 'again', 'why', 'isn', 'hers', 'same', 'did', "hasn't", 'which', 'him', 'me', 'by', "mightn't", 'd',
      'aren', 'had', 'of', 've', 'this', 'to', "weren't", 'where', "wasn't", 'am', 'only', "mustn", 'your', 'until', 'own', 'll', "isn't", "won't", 'y', 'ove
      r', 'a', 'what', "you'd", 'haven', 'during', "she's", 'and', 'such', 'just', 'few', 'once', 'were', 'm', 'has', 'here', 'more', 'hadn', 'yourself', 't',
      'these', 'how', 'not', 'with', 'very', "didn't", 'those', 'from', 'who', 'shouldn', 'above', 'all', "shan't", 'on', 'ourselves', "that'll", 'ma', 'its',
      'was', 'an', 'doing', 'that', 'under', 'can', "couldn't", "should've", "you'll", 'our', "doesn't", 'shan', 'have', "needn't", 'nor', 'do', 'than', 'her',
      'wouldn', 'through', 'further', "shouldn't", 'hasn', 'whom', 'between', 'now', 'didn', 'will', 'down', 'yours', 'is', 'been', "haven't", 'about', 'we',
      'couldn', 'herself', 'their', 'itself', 'needn', "don't", 'too', 'ours', 'my', 'o', 'don', 'while', 'up', 'there', 'most', 'but', 'ain', 'for', 'doesn',
      'at', 'in', 'into', 'so', 'before', "mustn't", "hadn't", "wouldn't", 'out', 'the', 'when', 'off'}
```

Figure 36: Stop word removal

```
[23]: #Stopword Removal
      def strip_stopwords(tokens):
          stop_words = set(stopwords.words('english'))
          cleaned_tokens = [token for token in tokens if token.lower() not in stop_words]
          return cleaned_tokens

[24]: print(strip_stopwords(['all', 'of', 'the', 'disasterswr', 'and', 'China', 'also', 'the', 'years', 'maybe']))

      ['disasterswr', 'China', 'also', 'years', 'maybe']
```

Figure 37: Stop word removal

Stopword removal is a technique used to filter out common words (like "the," "is," and "in") that don't carry significant meaning in text analysis. The strip_stopwords function takes a list of tokens (words) as input and removes those that are found in a predefined list of stopwords for the English language, provided by the NLTK library. For instance, the input list ['all', 'of', 'the', 'disasterswr', 'and', 'China', 'also', 'the', 'years', 'maybe'] is processed and common words like "all", "the" and "of" are removed resulting as ['disasterswr', 'China', 'also', 'years', 'maybe'].

5.2.4 Stemming (PorterStemmer vs WordNetLemmatizer)

a) PorterStemmer

```
[26]: #Stemming
      import nltk
      from nltk.stem import PorterStemmer
      def stem(tokens):
          ps = PorterStemmer()
          stems = [ps.stem(x) for x in tokens ]
          #sentence_with_stemmed_words = ' '.join(stems)
          return stems

[29]: print(stem(['disasters', 'China', 'also', 'years', 'maybe']))

      ['disast', 'china', 'also', 'year', 'mayb']
```

Figure 38: PorterStemmer

Stemming, as implemented in the PorterStemmer, removes prefixes and suffixes to get a root form. In the given code stemming transforms ['disasters', 'China', 'also', 'years', 'maybe'] into ['disast', 'china', 'also', 'year', 'mayb']. Stemming sometimes produces less non-standard or incomplete word forms like ‘disast’ in this case.

b) WordNetLemmatizer

```
[40]: import nltk
      from nltk.stem import WordNetLemmatizer
      #Lemmatization
      def lemmatize(tokens):
          lem = WordNetLemmatizer()
          lems = []
          for token in tokens:
              temp = lem.lemmatize(token, 'n')
              lems.append(lem.lemmatize(temp, 'v'))
          return lems

[41]: print(lemmatize(['disasters', 'China', 'also', 'years', 'maybe']))

      ['disaster', 'China', 'also', 'year', 'maybe']
```

Figure 39: WordNetLemmatizer

Lemmatization implements dictionaries and rules to reduce words to their base form, ensuring meaningful word in the language. In the case of ['disasters', 'China', 'also', 'years', 'maybe'], lemmatization transforms the words into ['disaster', 'China', 'also', 'year', 'maybe'], thus the word ‘disaster’ is not reduced further. This method is more precise as it produces linguistically correct base forms.

In conclusion, overall code implements lemmatization instead of stemming because of its higher accuracy and linguistic correctness.

5.2.5 Named Entity Recognition (NER)

```
[56]: lemmatized_words = ['list', 'disaster', '2021', 'die', 'toll']

[57]: import stanza
      #nlp = stanza.Pipeline("en")
      #nlp = stanza.Pipeline("en", download_method=stanza.DownloadMethod.NONE)
      text = " the list for disasters @ in 2021 and died tolls."
      doc = nlp(text)
      if hasattr(doc, 'entities') and len(doc.entities) > 0:
          print(doc.entities[0].text)
          print(doc.entities[0].type)
          lemmatized_words.append(doc.entities[0].type.lower())
      print(lemmatized_words)

      2021
      DATE
      ['list', 'disaster', '2021', 'die', 'toll', 'date']
```

Figure 40: Named Entity Recognition

Named Entity Recognition (NER) is a technique used in NLP to identify and classify key elements, such as names of people, organizations, dates, and locations. In this case, the text "the list for disasters @ in 2021 and died tolls." is processed, and NER identifies the entity "2021" as a DATE. Initially, the list ['list', 'disaster', '2021', 'die', 'toll'] contains lemmatized words. After recognizing "2021" as a date, the entity type "DATE" is appended to the list, updating it to ['list', 'disaster', '2021', 'die', 'toll', 'date'].

5.2.6 Bag of Words (BoW)

```
[58]: # Bag of Words
class_list = [['eventName', 'disaster'], ['region', 'place'], ['Group', 'category']]
prop_list = [['TypeName', 'type'], ['startYearValue', 'year', 'date'], ['countryName', 'country', 'gpe'], ['GroupName', 'group'], ['magnitude', 'magnitud

[60]: def map_to_ontology(lemmatized_words, class_list, prop_list):
    query_classes = []
    query_properties = []

    # Match lemmatized words to classes
    for word in lemmatized_words:
        for class_group in class_list:
            if word.lower() in [cls.lower() for cls in class_group]:
                query_classes.append(class_group[0])  # Select the canonical class (e.g., 'Movie')

    # Match lemmatized words to properties
    for word in lemmatized_words:
        for prop_group in prop_list:
            if word.lower() in [prop.lower() for prop in prop_group]:
                query_properties.append(prop_group[0])  # Select the canonical property (e.g., 'has_director')

    return query_classes, query_properties

# Map lemmatized words to ontology
query_classes, query_properties = map_to_ontology(lemmatized_words, class_list, prop_list)
print(query_classes,query_properties)

['eventName'] ['totalDeaths', 'startYearValue']
```

Figure 41: Bag of words

Bag of Words is a method that transforms text into a set of words, disregarding grammar and word order but keeping track of word frequency. In this case, the code is designed to match lemmatized words from a given list (e.g., ['list', 'disaster', '2021', 'die', 'toll']) to predefined classes and properties in an ontology. The class list and property list contain pairs such as [['eventName', 'disaster'], ['region', 'place']] and [['TypeName', 'type'], ['startYearValue', 'year', 'date']], where words like "disaster" and "2021" are mapped to their respective classes and properties like eventName and startYearValue. For example, after processing the lemmatized words, the code identifies "disaster" as belonging to the class eventName and "2021" as corresponding to the property startYearValue.

5.2.7 Speech Recognition

```
[116]: import speech_recognition as sr
# Initialize recognizer class (for recognizing the speech)
recognizer = sr.Recognizer()
#Microphone as the audio source
with sr.Microphone() as source:
    print("Please say something...")
    #Recognizer sensitivity
    recognizer.adjust_for_ambient_noise(source, duration=1)
    #AudioData object
    audio = recognizer.listen(source)
    print("Recognizing...")
    #Google's speech recognition
    try:
        text = recognizer.recognize_google(audio)
        print("Your Query: ", text)
    except sr.UnknownValueError:
        print("Google Speech Recognition could not understand the audio")
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition service; {e}")
    df = process_data(text)
    print(df)

Please say something...
Recognizing...
Your Query:  give me the disasters of Asia and people died
                          eventName totalDeaths regionName
0                       Boeing 707           15       Asia
1                         Coal mine          19       Asia
2                     Storm 'Norma'           3       Asia
3           Tropical cylone 'Pabuk'           7       Asia
4                             Cargo          11       Asia
```

Figure 42: Speech Recognition

Speech recognition is a technology that enables devices to process and understand human speech, converting spoken language into text. In this case, the speech_recognition library is used, which leverages Google's speech recognition service to recognize and transcribe speech from a microphone of the device. The system adjusts for ambient noise. Once the speech is captured, the recognize_google method converts audio into text. For example, the speech input "give me the disasters of Asia and people died" is transcribed into text and processed by a data function (process_data). This function then extracts relevant data such as event names, death tolls, and regions from a dataset, producing an output like ['Boeing 707', 'Coal mine', 'Storm "Norma"'] alongside corresponding data like the region "Asia" and death totals.

1.12 Overall Code:

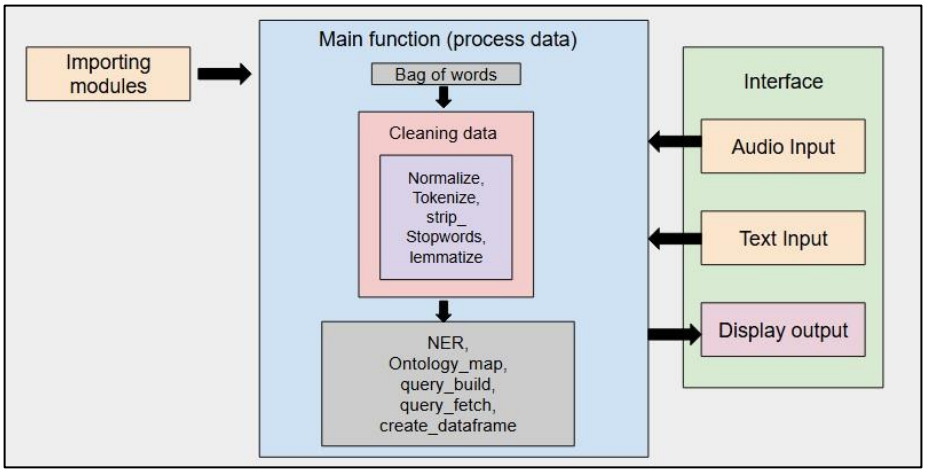


Figure 43: Code Architecture

The picture represents the architecture of the overall code from processing input to the final result. Initially, required modules are imported to enable necessary functionalities. The core processing occurs in the Main Function, which calls all other necessary functions like Bag of Words etc. This is followed by a comprehensive data cleaning phase, including normalization, tokenization, removal of stopwords, and lemmatization to refine the data for further analysis. Later Named Entity Recognition (NER), ontology mapping, and query formulation are applied. Next, the DataFrame is created for storage and display of results. Finally, two interfaces are created allowing audio and text inputs and processed results are displayed.

1. Importing the modules

```
[31]: import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import re
from string import punctuation
import SPARQLWrapper
from SPARQLWrapper import SPARQLWrapper2
import sys
import os
import contextlib
import stanza
import logging

# Suppress output during the NLP pipeline initialization
@contextlib.contextmanager
def suppress_stdout_stderr():
    with open(os.devnull, 'w') as devnull:
        with contextlib.redirect_stdout(devnull), contextlib.redirect_stderr(devnull):
            yield
```

Figure 44: Importing Modules

2. Defining Bag of words

```
[52]: # Bag of Words
def bag_of_words():
    class_list = [['eventName', 'disaster'], ['region', 'place'], ['Group', 'category']]
    prop_list = [['TypeName', 'type'], ['startYearValue', 'year', 'date'], ['countryName', 'country', 'gpe'], ['GroupName', 'group'], ['magnitude', 'magn
    return class_list, prop_list
```

Figure 45: Defining bag of words

3. Defining functions for Cleaning Data (Normalize -> tokenize -> stopword removal -> lemmatization)

```
[33]: def clean_data(data):
    text = data
    import nltk
    from nltk.tokenize import word_tokenize
    from nltk.stem import WordNetLemmatizer
    from nltk.corpus import stopwords
    import re
    from string import punctuation

    def normalize(text):
        processed_text = re.sub(f"[{re.escape(punctuation)}]", "", text)
        processed_text = " ". join(processed_text. split())
        return processed_text

    def strip_stopwords(tokens):
        stop_words = stopwords.words("english")
        clean_words = []
        for word in tokens:
            if word not in stop_words:
                clean_words.append(word)
        #print(clean_words)
        return clean_words
```

Figure 46: Cleaning data

```
    def lemmatize(text):
        text = normalize(text)
        lem = WordNetLemmatizer()
        lems = []
        tokens = word_tokenize(text)
        #print(tokens)
        clean_tokens = strip_stopwords(tokens)
        for token in clean_tokens:
            temp = lem.lemmatize(token, 'n')
            lems.append(lem.lemmatize(temp, 'v'))
        return lems

    lemmatized_words = lemmatize(text)
    return lemmatized_words
```

Figure 47: Cleaning data

4. Named Entity Recognition (NER)

```
[28]: def ner(new_text, lemmatized_words):
          cleaned_tokens = lemmatized_words
          import stanza
          # nlp = stanza.Pipeline("en")
          logging.getLogger("stanza").setLevel(logging.WARNING)
          with suppress_stdout_stderr():
              nlp = stanza.Pipeline("en", download_method=stanza.DownloadMethod.NONE)
          doc = nlp(new_text)
          if hasattr(doc, 'entities') and len(doc.entities) > 0:
              # print(doc.entities[0].text)
              # print(doc.entities[0].type)
              cleaned_tokens.append(doc.entities[0].type.lower())
          #     lemmatized_words.append(doc.entities[0].text)
          return cleaned_tokens, doc
```

Figure 48: NER

5. Ontology Mapping (lemmatized words -> Bag of Words -> Classes and Properties)

```
[54]: def ontology_map(lemmatized_words, cls_lst, prp_lst):
          class_list, prop_list =  cls_lst, prp_lst
          lemmatized_words = lemmatized_words
          # Function to map lemmatized words to ontology classes and properties
          def map_to_ontology(lemmatized_words, class_list, prop_list):
              query_classes = []
              query_properties = []

              # Match lemmatized words to classes
              for word in lemmatized_words:
                  for class_group in class_list:
                      if word.lower() in [cls.lower() for cls in class_group]:
                          query_classes.append(class_group[0])  # Select the canonical class (e.g., 'eventName')

              # Match lemmatized words to properties
              for word in lemmatized_words:
                  for prop_group in prop_list:
                      if word.lower() in [prop.lower() for prop in prop_group]:
                          query_properties.append(prop_group[0])  # Select the canonical property (e.g., 'Type')

              return query_classes, query_properties

          # Map lemmatized words to ontology
          query_classes, query_properties = map_to_ontology(lemmatized_words, class_list, prop_list)
          return query_classes, query_properties
```

Figure 49: Ontology mapping

Parts 1 to 6 of the code are the implementation of the various NLP techniques through functions which are mentioned in the previous parts of the report, thus the code remains same.

6. Query building

```
[8]: def query_build(query_classes, query_properties, doc):
         import SPARQLWrapper
         query_classes, query_properties, doc = query_classes, query_properties, doc
         # Build the SPARQL query dynamically based on the mapped classes and properties
         # Initialize the SPARQL query
         sparql_query = """
         SELECT """ + " ".join([f"?{cls}" for cls in query_classes] + [f"?{prop}" for prop in query_properties]) + """
         WHERE {
           ?event a <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#DisasterEvent>.
         """

         # Add the class and property conditions dynamically
         sparql_query += "\n".join([f"?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#{cls}> ?{cls}." for c
         sparql_query += "\n".join([f"?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#{prop}> ?{prop}." for
         if hasattr(doc, 'entities') and len(doc.entities) > 0:
             if doc.entities[0].type == "GPE":
                 if hasattr(doc, 'entities') and len(doc.entities) > 0:
                     cvalue = doc.entities[0].text
                     #print(doc.entities[0].type)
                     sparql_query += (f"FILTER(?countryName = \"{cvalue}\")")
             # Regular expression to match a year (4 digits)
             year_pattern = r"\b(\d{4})\b"
             # Search for a year in the input string
             match = re.search(year_pattern, text)
```

Figure 50: Query building

```
        if match:
            # Extracted year
            yvalue = int(match.group(1))
            sparql_query += (f"FILTER(?startYearValue = \"{yvalue}\")")
        if doc.entities[0].type == "LOC":
            lvalue = doc.entities[0].text
            sparql_query += (f"FILTER(?regionName = \"{lvalue}\")")
    sparql_query += """
    } LIMIT 20
    """
    return sparql_query
```

Figure 51: Query building

7. Fetching Data from fuseki server

```
[58]: def query_fetch(sparql_query):
          sparql_query = sparql_query
          # Connect to Fuseki server and execute the query
          sparql = SPARQLWrapper2("http://localhost:3030/Disaster_management_dataset/")
          sparql.setQuery(sparql_query)
          data = sparql.query().bindings
          return data
```

Figure 52: Fetching data

The query_build function dynamically constructs SPARQL query by breaking it as a string and concatenating values based on classes and query_properties.The values are inserted as variables using f-strings. The function also adds additional filtering conditions. Then the constructed query is sent to the server as a set of variable bindings and results are retrieved.

8. Creating DataFrame from the results

```
[69]: def create_dataframe(data, query_classes, query_properties):
          data, query_classes, query_properties = data, query_classes, query_properties
          import pandas as pd
          events = []
          for result in data:
              event_data = []
              for cls in query_classes:
                  event_data.append(result[f"{cls}"].value if f"{cls}" in result else None)
              for prop in query_properties:
                  event_data.append(result[f"{prop}"].value if f"{prop}" in result else None)
              events.append(event_data)
          # Dynamically assign column names for the DataFrame
          columns = []
          for cls in query_classes:
              columns.append(f"{cls}")  # Adding columns for query_classes
          for prop in query_properties:
              columns.append(f"{prop}")  # Adding columns for query_properties

          df = pd.DataFrame(events, columns=columns)
          return df
```

Figure 53: Creating dataframe

Dataframe is created from the values retrieved from result, the columns are dynamically created and values are appended through iteration for for loop.

9. Main function

```
[112]: def process_data(text):
           class_list, prop_list = bag_of_words()
           lemmatized_words =  clean_data(text)
           nere, doc = ner(text, lemmatized_words)
           query_classes, query_properties = ontology_map(lemmatized_words, class_list, prop_list)
           sparql_query = query_build(query_classes, query_properties, doc)
           data = query_fetch(sparql_query)
           df = create_dataframe(data, query_classes, query_properties)
           # return lemmatized_words, query_classes, query_properties, sparql_query, df
           return df
```

Figure 54: Main function

10.A) Audio Input

```
[116]: import speech_recognition as sr
# Initialize recognizer class (for recognizing the speech)
recognizer = sr.Recognizer()
#Microphone as the audio source
with sr.Microphone() as source:
    print("Please say something...")
    #Recognizer sensitivity
    recognizer.adjust_for_ambient_noise(source, duration=1)
    #AudioData object
    audio = recognizer.listen(source)
    print("Recognizing...")
    #Google's speech recognition
    try:
        text = recognizer.recognize_google(audio)
        print("Your Query: ", text)
    except sr.UnknownValueError:
        print("Google Speech Recognition could not understand the audio")
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition service; {e}")
    df = process_data(text)
    print(df)
```

Figure 55: Audio input

10.B) User Interface

```
[117]: text = input("Enter your question")
df = process_data(text)
print(df)
```

Figure 56: User interface

Two interfaces are created one for the audio input and the other for text input.The audio input uses Google's speech recognition module and converts it to text whereas the text interface directly sends text to the main function. Finally, the main function calls all other functions passing parameters and returns the result.

1.13 Code Validation:

The flow of code is represented in below diagram, out of 8 stages input data goes through stage 1 – 7 and output is received in stage 8.

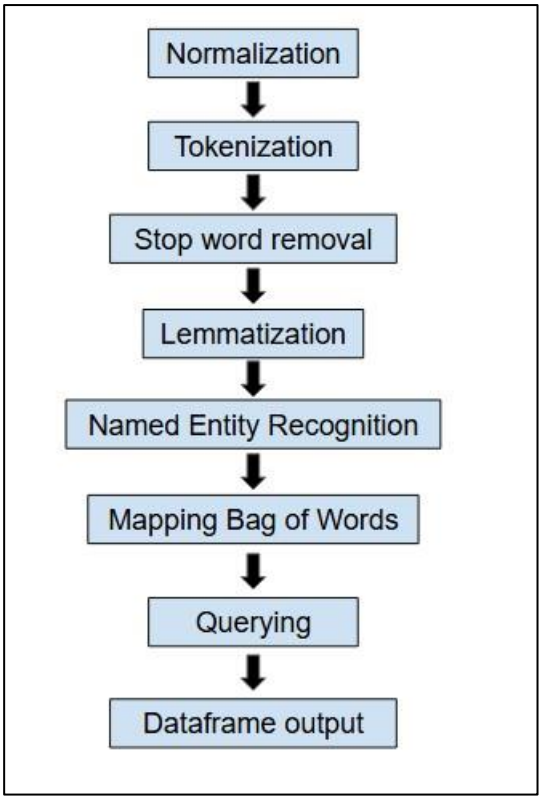


Figure 57: Flow of code

```
import logging
from IPython.display import display, Markdown
# Suppress output during the NLP pipeline initialization
@contextlib.contextmanager
```

Figure 58: Code validation

```
def clean_data(data):
    text = data
    def normalize(text):
        processed_text = re.sub(f"[{re.escape(punctuation)}]", "", text)
        processed_text = " ". join(processed_text. split())
        return processed_text

    def strip_stopwords(tokens):
        stop_words = stopwords.words("english")
        clean_words = []
        for word in tokens:
            if word not in stop_words:
                clean_words.append(word)
        #print(clean_words)
        return clean_words
    def lemmatize(text):
        text = normalize(text)
        lem = WordNetLemmatizer()
        lems = []
        tokens = word_tokenize(text)
        #print(tokens)
        clean_tokens = strip_stopwords(tokens)
        for token in clean_tokens:
            temp = lem.lemmatize(token, 'n')
            lems.append(lem.lemmatize(temp, 'v'))
        return text, tokens, clean_tokens, lems

    text, tokens, clean_tokens, lemmatized_words = lemmatize(text)
    return text, tokens, clean_tokens, lemmatized_words
```

Figure 59: Code validation

9. Main function

[58]:
```
def process_data(text):
    class_list, prop_list = bag_of_words()
    text, tokens, clean_tokens, lemmatized_words =  clean_data(text)
    display(Markdown('**1. Normalized output:**'))
    print(text)
    display(Markdown('**2. Tokenized Output:**'))
    print(tokens)
    display(Markdown('**3.Stopwords removed output:**'))
    print(clean_tokens)
    display(Markdown('**4. Lemmatized output:**'))
    print(lemmatized_words)
    ner_tokens, doc = ner(text, lemmatized_words)
    query_classes, query_properties = ontology_map(ner_tokens, class_list, prop_list)
    sparql_query = query_build(query_classes, query_properties, doc)
    data = query_fetch(sparql_query)
    df = create_dataframe(data, query_classes, query_properties)
    display(Markdown('**5. NER output:**'))
    print(ner_tokens)
    display(Markdown('**6. Bag of Words Mapped output:**'))
    print(query_classes, query_properties)
    display(Markdown('**7. Query output:**'))
    print(sparql_query)
    display(Markdown('**8. Dataframe output:**'))
    print(df)
    # return df
```

Figure 60: Code validation

In order to validate the code few changes have been made as shown in the above figures. The display(Markdown(...)) and print() functions are used to present intermediate steps.

5.4.1 NL Query 1

10.B) User Interface

[60]:
```
display(Markdown('**Enter your question:**'))
text = input()
process_data(text)
# df = process_data(text)
# print(df)
```

Enter your question:

the list for disas$ters @ in 2021 yfg and died people.

1. Normalized output:

the list for disasters in 2021 yfg and died people

2. Tokenized Output:

['the', 'list', 'for', 'disasters', 'in', '2021', 'yfg', 'and', 'died', 'people']

3.Stopwords removed output:

['list', 'disasters', '2021', 'yfg', 'died', 'people']

4. Lemmatized output:

['list', 'disaster', '2021', 'yfg', 'die', 'people']

5. NER output:

['list', 'disaster', '2021', 'yfg', 'die', 'people', 'date']

6. Bag of Words Mapped output:

['eventName'] ['totalDeaths', 'startYearValue']

Figure 61: NL Query 1

7. Query output:

```
SELECT ?eventName ?totalDeaths ?startYearValue
WHERE {
  ?event a <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#DisasterEvent>.
 ?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#eventName> ?eventName.?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#totalDeaths> ?totalDeaths.
?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#startYearValue> ?startYearValue.FILTER(?startYearValue = "2021")
    } LIMIT 20
```

8. Dataframe output:

	eventName	totalDeaths	startYearValue
0	Measles	72	2021
1	Chemical factory	78	2021
2	Cholera	713	2021
3	Building	25	2021
4	Helicopter	13	2021
5	Mine	14	2021
6	Chemical factory	10	2021
7	Buildings	24	2021

Figure 62: NL Query 1

5.4.2 NL Query 2

Figure 63: NL Query 2

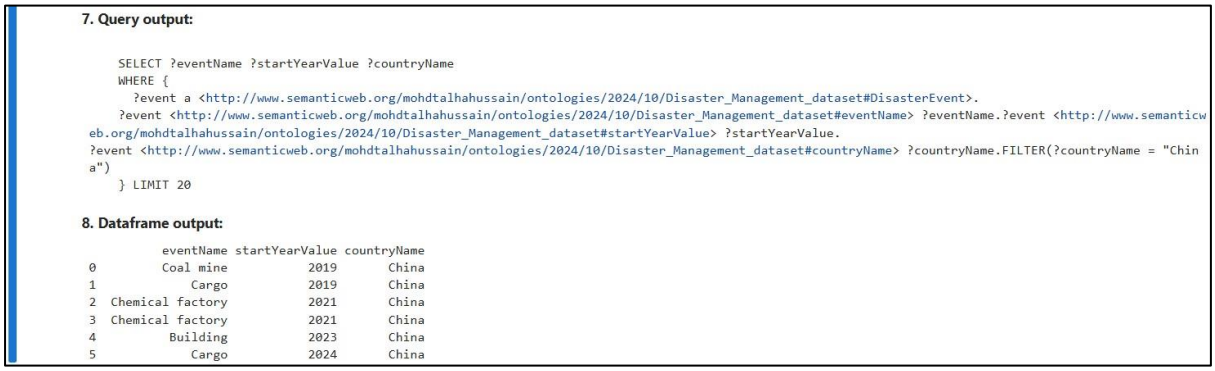


Figure 64: NL Query 2

5.4.3 NL Query 3

Figure 65: NL Query 3

7. Query output:

```
SELECT ?eventName ?magnitude ?regionName
WHERE {
  ?event a <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#DisasterEvent>.
 ?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#eventName> ?eventName.?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#magnitude> ?magnitude.
?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#regionName> ?regionName.FILTER(?regionName = "Africa")
} LIMIT 20
```

8. Dataframe output:

	eventName	magnitude	regionName
0	Cyclone 'Idai'	140	Africa
1	Cyclone 'Kenneth'	185	Africa
2	Cyclone 'Kenneth'	300	Africa

Figure 66: NL Query 3

5.4.4 NL Query 4

10.B) User Interface

```
[64]: display(Markdown('**Enter your question:**'))
text = input()
process_data(text)
# df = process_data(text)
# print(df)
```

Enter your question:

types of acci%dents killi#ng people India?

1. Normalized output:

types of accidents killing people India

2. Tokenized Output:

['types', 'of', 'accidents', 'killing', 'people', 'India']

3.Stopwords removed output:

['types', 'accidents', 'killing', 'people', 'India']

4. Lemmatized output:

['type', 'accident', 'kill', 'people', 'India']

5. NER output:

['type', 'accident', 'kill', 'people', 'India', 'gpe']

6. Bag of Words Mapped output:

[] ['TypeName', 'totalDeaths', 'countryName']

Figure 67: NL Query 4

7. Query output:

```
SELECT ?TypeName ?totalDeaths ?countryName
WHERE {
  ?event a <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#DisasterEvent>.
 ?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#TypeName> ?TypeName.
?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#totalDeaths> ?totalDeaths.
?event <http://www.semanticweb.org/mohdtalhahussain/ontologies/2024/10/Disaster_Management_dataset#countryName> ?countryName.FILTER(?countryName = "India")
} LIMIT 20
```

8. Dataframe output:

	TypeName	totalDeaths	countryName
0	Water	10	India
1	Explosion (Industrial)	10	India
2	Fire (Miscellaneous)	17	India
3	Storm	50	India
4	Storm	50	India
5	Extreme temperature	90	India
6	Epidemic	121	India
7	Extreme temperature	22	India
8	Fire (Miscellaneous)	20	India
9	Flood	52	India
10	Storm	10	India
11	Storm	50	India
12	Road	29	India
13	Flood	1900	India
14	Road	35	India
15	Road	11	India

Figure 68: NL Query 4

5.4.5 NL Query 5

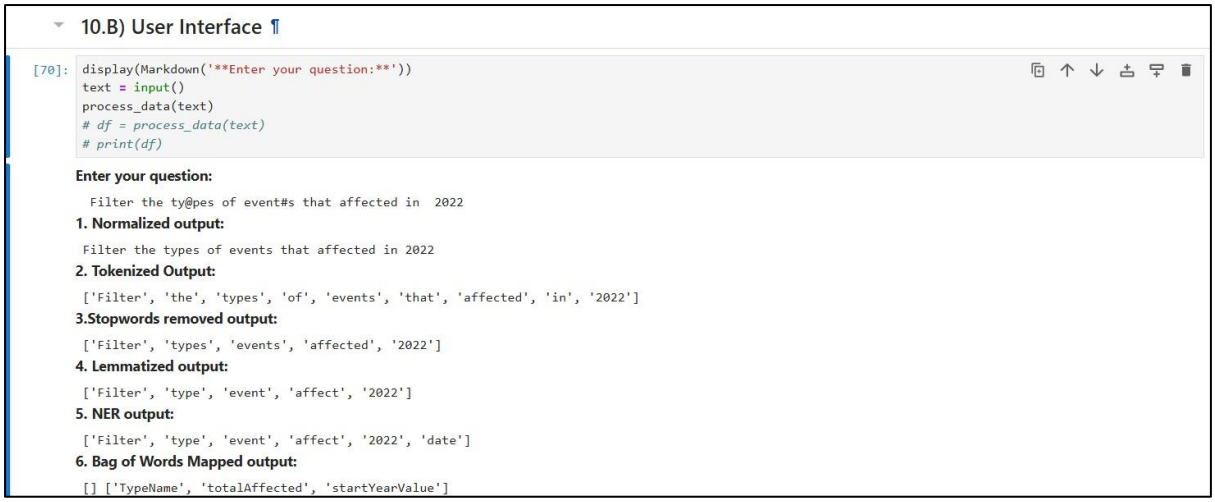


Figure 69: NL Query 5

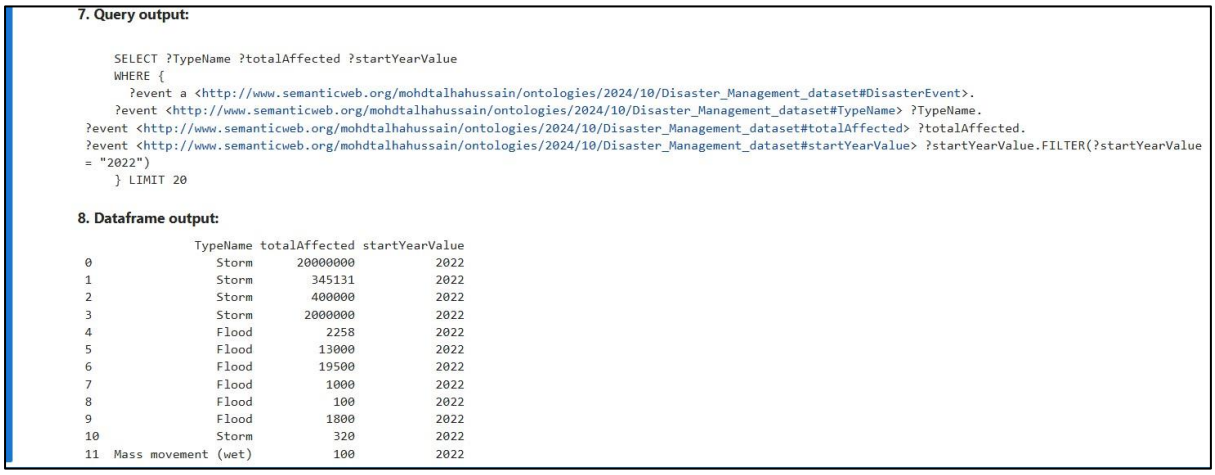


Figure 70: NL Query 5

7. Conclusion and Future Work

This project has demonstrates the integration of semantic modeling and natural language processing (NLP) techniques in disaster management systems. The semantic approach, grounded in the literature of disaster risk reduction and information retrieval (Bernstein et al., 2016; Manning et al., 2014; Schulze et al., 2016), provides a more structured and interoperable solution compared to traditional systems. The methodology and technological integration presented here not only optimize the management of disaster data but also facilitate improved decision-making processes, which are critical during emergency response and recovery phases.

Despite the promising results, the system could be expanded to integrate with other disaster management tools, including early warning systems, resource allocation platforms, and real-time monitoring systems. By connecting this semantic-based query system with other data sources and decision support tools, a more holistic and dynamic disaster management platform could be developed, ultimately contributing to better preparedness, faster responses, and more effective recovery efforts.

In conclusion, while this project has laid a solid foundation for utilizing semantic modelling and NLP recognition in disaster management, it opens up numerous possibilities for future advancements that can enhance the system's capabilities and its practical application in real-world disaster scenarios.

8. References

1. Bernstein, A., Choy, H. & Yang, Y., 2016. The role of semantics in disaster management. *International Journal of Disaster Risk Reduction*, 19, pp.120-129.
2. Manning, C.D., Raghavan, P. & Schütze, H., 2014. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press.
3. Schulze, M., Panos, D. & Wrede, B., 2016. Semantic interoperability in disaster management: Integrating heterogeneous information sources. In: *Proceedings of the International Conference on Information Systems for Crisis Response and Management (ISCRAM)*, pp.142-152.
4. Cohen, J., Manion, L. & Morrison, K., 2015. *Research Methods in Education*. 8th ed. London: Routledge.
5. Jurafsky, D. and Martin, J.H., 2021. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd ed. Pearson.