Assessment 2

Knowledge Representation & Reasoning

CS551J

Dr. Mingjun Zhong

Yassin Dinana

52094480

8th March 2021

Introduction:

Knowledge Graphs is a programmatic method that is used for data interlinking and presentation. It is also described as a digital structure that links and builds relationships between facts, it usually includes an ontology, which is a set of concepts covering one subject that shows the relationship between its entities, where all the entities could be keywords, images, locations, or organizations, all those entities are interconnected forming the knowledge graph.

Different companies such as Google, Microsoft, eBay, and Airbnb use knowledge graphs to enhance their search engines and to understand their customers, business decisions, and gather data analytics.

The knowledge graph is represented as a set of nodes and edges.

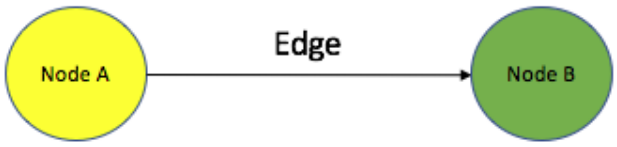

Figure 1: Representation of a triple

Looking at figure 1 above, it represents a triple – where node A represents one entity and node B represents a second entity where both entities are linked by an edge which represents the relationship between both entities.

This paper will be mainly focusing on designing and encoding a text knowledge graph and its analysis. The subject for the knowledge graph to be built for this paper is the *Covid-19 Vaccine*, the knowledge graph will collect and extract text information from different web pages in order to gather data that will help build the knowledge graph. Different sentences related to the Covid-19 Vaccine will be developed and then the triples, also known as (subject and object) will be extracted to build the knowledge graphs.

Python will be used as the programming language for this project, different frameworks and libraries are also used such as DBpedia and RDFLibrary which will be deeply explained later on this paper.

1.1-

In order to collect and gather the appropriate resources related to Covid-19 from web articles, different approved URLs related to the Covid-19 Vaccine are added to a list in the beginning of the program.

```
import requests

urls = ['https://www.nhs.uk/conditions/coronavirus-covid-19/coronavirus-vaccination/coronavirus-vaccine/',
        'https://www.who.int/emergencies/diseases/novel-coronavirus-2019/covid-19-vaccines',
        'https://www.cdc.gov/coronavirus/2019-ncov/vaccines/index.html',
        'https://www.dshs.state.tx.us/covidvaccine/',
        'https://www.bbc.com/news/world-europe-56252028',
        'https://www.health.gov.au/initiatives-and-programs/covid-19-vaccines',
        'https://www.afro.who.int/news/africa-getting-ready-roll-out-covid-19-vaccines',
        'https://www.thelancet.com/journals/laninf/article/PIIS1473-3099(20)30800-8/fulltext',
        'https://www.businessinsider.com/vaccine-wars-rich-countries-battle-doses-poorer-miss-out-2021-1']

html_page = page.content

req = []

for url in urls:
  page = requests.get(url)
  req.append(page)
  print(page.text)

  html_page_g = []

for res in req:
  html_page_g_temp = res.content
  html_page_g.append(html_page_g_temp)
```

Figure 2: Importing URLs related to the Covid-19 vaccine and looping on the list

Looking at figure 2 above, a list is created where it contains different links related to the Covid-19 vaccine are added to a python list. Those URLs include all the text information that will be gathered in order to build a knowledge graph. As it can be seen, a library named Requests in being imported to the notebook, this library is an HTTP library that makes it easy to send requests to the HTTP web pages, the reason why a request must be sent to the links in the list is because it is important to extract the content of all the URLs in order to collect the articles and text related to the Covid-19 Vaccine.

```
<div class="flex-parent">
  <div class="user-input-container">
    <label
      id="focusedLabel"
      class="headline-regular"
      for="newsletter-module-email"
    >Email address</label>
    <input
      class="headline-regular"
      type="email"
      autocapitalize="off"
      name="newsletter-module-email"
      id="newsletter-module-email"
      placeholder="Email address"
      autocomplete="email"
      required="required"
```

Figure 3: Snippet of the raw HTML output of the URLs provided in the list

As there are many URLs in the list, all of them needs to be accessed, that is the reason why there are *For Loops* created to access all the links, such loops that can be seen in figure 2 include sending HTTP requests to all the URLs, printing all the text in the webpages, where the HTTP format of the text will be printed when running the cell.

After running the code cell in figure 2, the HTML text of all the web pages is printed as an output, looking at figure 3 above, it is a snippet the output, which is just raw HTML. As the HTML output has been printed, this means that the resources and text related to the URLs provided has been collected and gathered. The aim is to extract the text from the raw HTML and convert it to sentences. This will be shown later in the report.

1.2-

As this process can be named web scraping, BeautifulSoup is then used, it is a package that is used to parse HTML and XML files, it is used in this step of the project to extract the data needed out of the URLs provided in figure 2. We use BeautiulSoup next as an HTML parser in order to extract all the text from the HTML output, which is seen in figure 3 looking at figure 4 below, where a variable named *html_page* (seen in figure 2) that includes the content of the URLs is passed to the parsing package.

```
from bs4 import BeautifulSoup

soup = bs4.BeautifulSoup(html_page, 'html.parser')
all_text = soup.get_text() # get all the text.
```

Figure 4: Using BeautifulSoup to parse the HTML data and extract the text

Looking at figure 4 above, the BeautifulSoup which is an NLP package receives the html_page, which includes all the pages content – the code snippet in figure 4 above, parses the HTML output and extracts all the text from the webpages. This means that all the information has been obtained. The code snippet in figure 4 is then used in a loop so it can cover all the URLs provided similarly to the For Loops provided in figure 2.

```
nltk.download('punkt')
sentences = tokenize.sent_tokenize(all_text) #Convert all text to sentences using Tokenize

#Saving all the senteces into a CSV file and name is "output.csv"
with open('output.csv', 'w', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['sentence'])
    for row in sentences:
        writer.writerow([str(row)])
```

Figure 5: Using NLTK to convert all text to sentences then save the sentences into a CSV file

Figure 5 above does the next step, after obtaining all the text from the HTML output using BeautifulSoup, it is now possible to convert the obtained text into sentences and save the sentences into CSV files. In order to do this conversion, we use the Natural Language ToolKit, better known

as (NLTK) is a platform downloaded in the python notebook which will extract the HTML text and convert it to human-readable sentences this is also done using the tokenize function in the NLTK library.

Looking at figure 5, the code snippet used to convert all the text into sentences can be seen as well as opening and saving all the sentences into a CSV file.

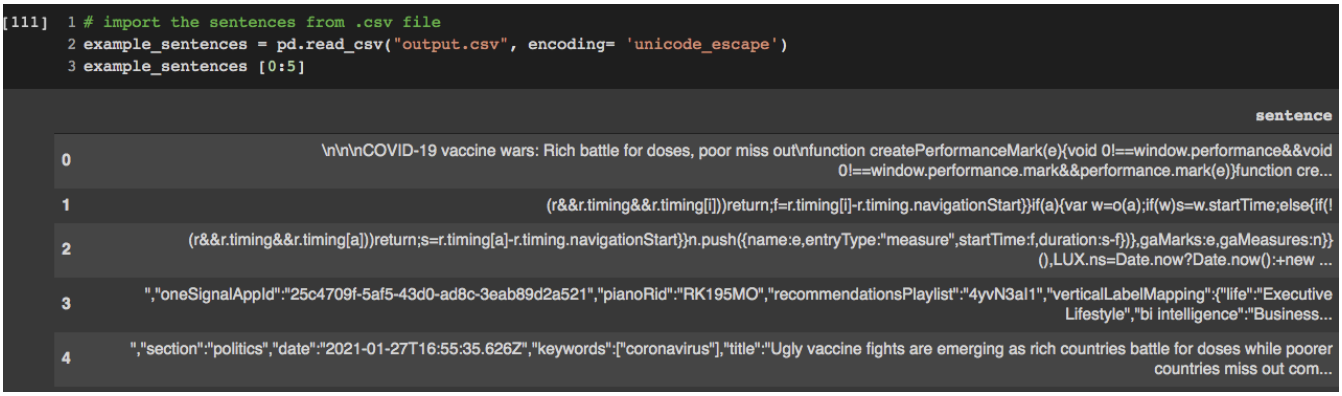

Figure 6: Reading the first 5 sentences in the CSV file and printing them

Looking at figure 6, the first 5 sentences in the CSV file are displayed and read. It can be seen that the sentences are full of noise and unwanted characters. The next part of this paper will be discussing the cleaning of the data and then displayed the cleaned sentences as well as comparing the difference.

Cleaning the data:

There are methods implemented to clean the text data and to remove the noise. Different functions are created where each function cleans a part in the data that can be seen in figure 6 above, such methods include:

- Stripping the HTML tags
- Removing accented characters
- Removing contractions (such as Don't, I've)
- Lemmatizing the text
- Removing stop words

Different libraries were imported in order to clean the data, such libraries include BeautifulSoup, UnicodeData, and Contractions. The library NLTK is also being used to remove the stop words in the sentences.

After each function has been implemented to clean the sentences, a main function named "normalized_corpus" is then implemented where it includes a For Loop that normalizes the text and completes the pre-processing of the data. Text normalization is the process of transforming the sentences in order to allow the consistency of the text data before applying it to build the knowledge graph. All the cleaning function implemented as well as the normalization function can be seen in figure 7 below.

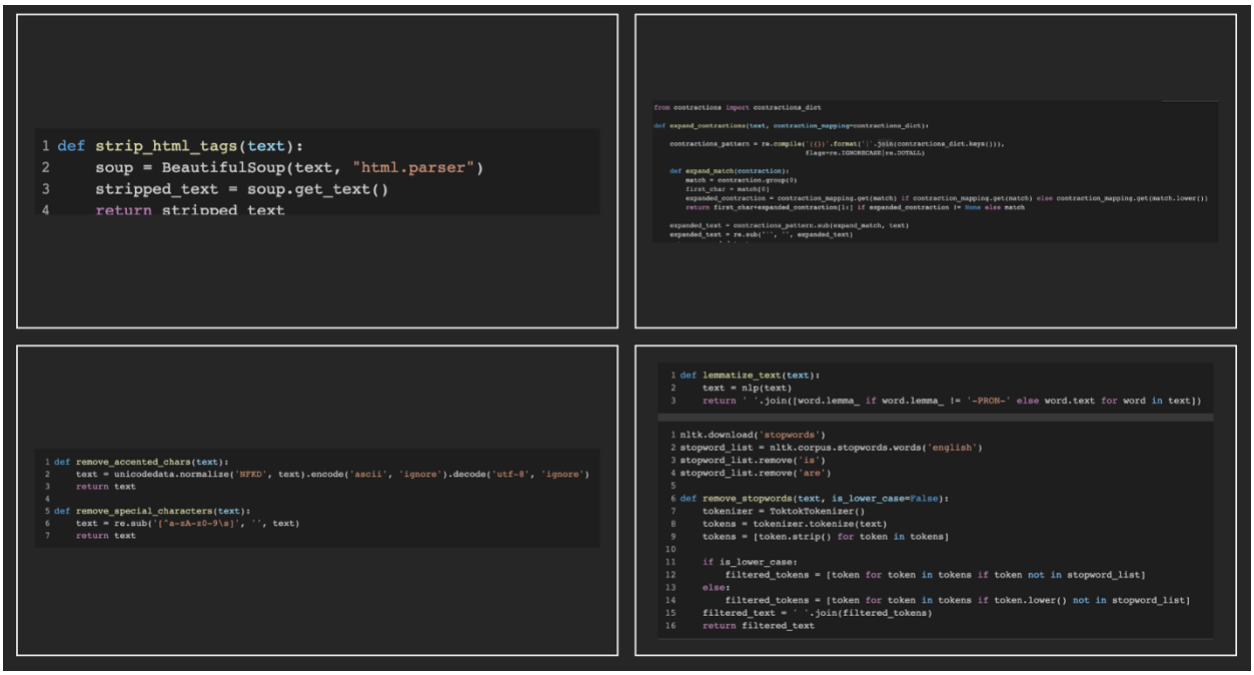

Figure 7: Functions used to clean the sentences

After running all the functions above, as stated, they are combined in the Normalized Corpus function in order to normalize the cleaned data and complete the pre-processing, the normalize corpus function can be seen in figure 8 below:

```
def normalize_corpus(corpus, html_stripping=True, contraction_expansion=True,
                     accented_char_removal=True, text_lower_case=True,
                     text_lemmatization=True, special_char_removal=True,
                     stopword_removal=True):

    normalized_corpus = []
    # normalize each document in the corpus
    for doc in corpus:
        # strip HTML
        if html_stripping:
            doc = strip_html_tags(doc)
        # remove accented characters
        if accented_char_removal:
            doc = remove_accented_chars(doc)
        # expand contractions
        if contraction_expansion:
            doc = expand_contractions(doc)
        # lowercase the text
        if text_lower_case:
            doc = doc.lower()
        # remove extra newlines
        doc = re.sub(r'[\r|\n|\r\n]+', ' ',doc)
        # insert spaces between special characters to isolate them
        special_char_pattern = re.compile(r'([{.(-)!}])')
        doc = special_char_pattern.sub(" \\1 ", doc)
        # lemmatize text
        if text_lemmatization:
            doc = lemmatize_text(doc)
        # remove special characters
        if special_char_removal:
            doc = remove_special_characters(doc)
        # remove extra whitespace
        doc = re.sub(' +', ' ', doc)
        # remove stopwords
        if stopword_removal:
            doc = remove_stopwords(doc, is_lower_case=text_lower_case)

        normalized_corpus.append(doc)
    return normalized_corpus
```

Figure 8: Normalize Corpus function combining all the cleaning functions and normalizing

In order to visualise the output of the sentences cleaning, the SpaCy library is also imported in the code, it is a python library that is mainly used for Natural Language Processing. By looking at figure 6, it can be seen that the variable including all the un-cleaned variables is called "example_sentences". After the sentences cleaning has been implemented, it is known that mainly the language in which the sentences are implemented is English, the following SpaCy line is being implemented:

nlp=spacy.load('en_core_web_sm')

The line above is an English pipeline optimized for the CPU where is includes many components such as parser, lemmatizer, and enter. It constructs the English language object and loads it in the model.

```
import unicodedata
from nltk.tokenize.toktok import ToktokTokenizer
nlp=spacy.load('en_core_web_sm')

clean_sentences =    normalize_corpus(full_sentences, html_stripping=True, contraction_expansion=True,
                     accented_char_removal=True, text_lower_case=True,
                     text_lemmatization=True, special_char_removal=True,
                     stopword_removal=True)

def clean_sentences_new(n):
    n=int(n[0:20])
    return n
```

Figure 9: Importing SpaCy and Clean Sentences variable created where it includes cleaned sentences

Another variable is implemented which is called "*clean_sentences*" where it includes all the sentences after they have been cleaned, this variable will be used to print the cleaned sentences and visualize the difference between the cleaned sentences and the sentences before being pre-processed and cleaned as seen in figure 6.

```
clean_sentences[0:7]
```

```
['israel aim immunize entire country march success',
 'insider logo word insider',
 'indicate click perform search',
 'copyright 2021',
 'rr timingr timing[ ] returns r timing[ ] r timing navigationstart n push name e entrytypemeasurestarttime
 'visit business insider homepage story',
 'newsletter start morning 10 thing politic need know today']
```

Figure 10: Cleaned Sentences printed

Looking at figure 10 above, it shows the sentences from the CSV file after they have been cleaned, the sentences before cleaning can be seen in figure 6 where they included many unwanted characters, HTML tags, and many strop words. Comparing it to the sentences after being cleaned, it can be seen that the sentences are well outputted. The clean sentences can now be implemented as an input to the knowledge graphs to create entities. Creating entities in the next part of this report where from all the cleaned sentences developed from the URLs, entities will be extracted in the form of triples structured data.

1.3-

This part of the report will involve extracting the triples required for created the Covid-19 Vaccine knowledge graph. First, we extract the dependency tags for one of the sentences that were cleaned in figure 10. This is done using the SpaCy library.

```
import spacy
nlp = spacy.load('en_core_web_sm')

doc = nlp("israel aim immunize entire country march success")

for tok in doc:
  print(tok.text, "...", tok.dep_)
```

```
israel ... nsubj
aim ... ROOT
immunize ... xcomp
entire ... amod
country ... compound
march ... compound
success ... dobj
```

Figure 11: Extracting the dependencies for one of the cleaned sentences

It can be seen in the output that the sentence has been broken down in different dependencies such as compound words, object, and roots. The root of the sentence which is also the verb of the sentence in the representation of the edge of the knowledge graph as it can be seen in figure 1. Such dependencies would be represented in the following form:

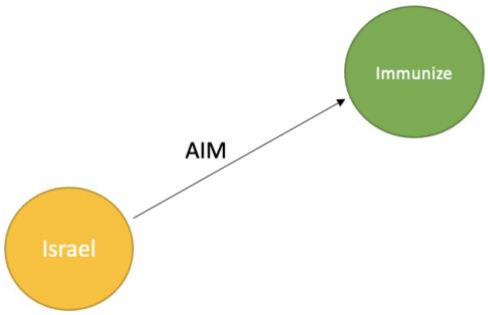

Figure 12: Knowledge Graph for the sentence provided

The knowledge graph in figure 12 above represents the first part of the sentence which is “Israel aim immunize”

As there are many different sentences where the entities (subject, predicate, and object) must be extracted, therefore, function “get_entities” is created. The function is broken and seen in the figure below.

```
def get_entities(sent):

# chunk 1 (2 entities are provided)
  ent1 = ""
  ent2 = ""

  previous_tok_dep = ""    # dependency for previous token
  previous_tok_text = ""   # the text of the previous token (from the sentence)

  prefix = ""
  modifier = ""
```

Figure 13: Extracting Entities Function – part 1

As the function broken apart into different chunks – figure 13 above shows the first chunk of the function, where function get_entities takes the parameter "sent". It can be seen that there are two entities to be extracted, where each entity will be taking a value of 0 or 1. Two empty variables are then created, those variables are "*previous_tok_dep*" and "*previous_tok_text*", those variables will be holding the dependency tags of the previous word, the dependency tags such as the ones provided in figure 11.

Two other empty variables are initialised which are "prefix" and "modifier", those variables will be holding the text that is found in the subject and object.

```
# chunk 2
    # if token is a punctuation mark then move on to the next token
  for tok in nlp(sent):
    if tok.dep_ != "punct":
      # Check for compound words in the token
      if tok.dep_ == "compound":
        prefix = tok.text
        # If yes, add the current word to it
        if previous_tok_dep == "compound":
          prefix = previous_tok_text + " "+ tok.text

      # check if the token is a modifier
      if tok.dep_.endswith("mod") == True:
        modifier = tok.text
        # If yes, add the current word to it
        if previous_tok_dep == "compound":
          modifier = previous_tok_text + " "+ tok.text
```

Figure 14: Extracting Entities Function – part 2

As for the second part of the function, a For Loop is created which loops on all the tokens of each sentence, it can be seen in the first "if" statement that if the token is a punctuation which is presented as "…" then ignore it and move to the next token. Hence, if it is found that the token is a compound word as discussed earlier, this means it has to be stored in the prefix variable which created in chunk 1 of this function as seen in figure 13. In the last part, it is checking if the token is a modifier or an object, if this is true then the token saved in the prefix is then added to it. This loop is looping on all the cleaned sentences extracted.

```
# chunk 3
    if tok.dep_.find("subj") == True:
      ent1 = modifier +" "+ prefix + " "+ tok.text
      prefix = ""
      modifier = ""
      previous_tok_dep = ""
      previous_tok_text = ""

# chunk 4
    if tok.dep_.find("obj") == True:
      ent2 = modifier +" "+ prefix +" "+ tok.text

# chunk 5
    previous_tok_dep = tok.dep_
    previous_tok_text = tok.text

return [ent1.strip(), ent2.strip()]
```

Figure 15: Extracting Entities Function – full.

Looking at figure 15 above, it finalises the function, looking at chunk 3 in the figure checks to find a subject, if the subject is found and is true – it will be applied to the first entity variable that is initialised in figure 13 – any other variable but the subject won't be added and will set the program to reorganise.

Chunk 4 in the figure above, does the same but it only looks for an object and not a subject. It then stores the object in the variable of entity 2 which is created in figure 13 – it reorganises and resets if any other variable is found.

Finally, chunk 5 representing the last part of the function just updates all the dependencies of the previous tokens when it finds and extracts the subject and the object in a sentence.

```
1 get_entities("israel aim immunize entire country march success")
```

```
['israel', 'entire country march success']
```

Figure 16: Testing the extraction on a sentence

Looking at figure 16 above, the sentence is passed to the function "get_entities" and as it can be seen in output the subject and object are extracted. This proves that the extraction method is working.

After the entities worked on one example as seen in figure 16, it can now be tested on all the clean sentences and visualise the subjects and objects of the sentences as an output. For such task, an empty python list is created which gets appended when the entity pairs are found. After running the program, the output is the list of all the entity pairs. The output is asked to print the entity pairs of the first 20 sentences.

```
entity_pairs = []
for i in tqdm(clean_sentences):
    if type(i) != float:
        entity_pairs.append(get_entities(i))

entity_pairs[0:20]
```

```
100%|██████████| 50/50 [00:00<00:00, 74.09it/s]
[['israel', 'entire country march success'],
 ['insider logo', 'insider'],
 ['', 'click perform search'],
 ['', ''],
 ['global healthcare strategy story', 'vtl6vsddvideonielsenapidp5982de1d31c5'],
 ['', ''],
 ['politic  need', ''],
 ['', ''],
 ['instagram icon', ''],
 ['germanys health minister', 'fair  share'],
 ['eu canada', 'donate vaccine poor country'],
 ['france germany spain', 'administer vaccine'],
 ['nl', 'polska'],
 ['else _ uspapi push', 'slice var r'],
 ['european union week', 'drug giant uk vaccine supply'],
 ['', ''],
 ['rich  country', 'ahead vaccination program'],
 ['anti tabloid press uk', 'heavily  row'],
 ['', ''],
 ['', '']]
```

Figure 17: Entity pair output of the first 20 clean sentences

Looking at figure 17 above, it can be seen that the first 20 sentences entity pairs are implemented as an output. It can be seen in the code that the clean sentences are passed in the for loop and not the original sentences extracted from the HTML (before being cleaned).

Looking at figure 12, It can be seen that the two nodes are connected using a verb, this is usually known as the relation between the nodes.

```
def get_relation(sent):

  doc = nlp(sent)
  matcher = Matcher(nlp.vocab)

  pattern = [{'DEP':'ROOT'},
             {'DEP':'prep','OP':"?"},
             {'DEP':'agent','OP':"?"},
             {'POS':'ADJ','OP':"?"}]

  matcher.add("matching_1", None, pattern)

  matches = matcher(doc)
  k = len(matches) - 1

  span = doc[matches[k][1]:matches[k][2]]

  return(span.text)
```

```
get_relation("israel aim immunize entire country march success")
```

```
'aim'
```

Figure 18: Extracting the edge (verb) of the sentences

Looking at figure 18 above, this function extracts the edge / verb from the sentences, it does this by trying to extract the root word which was discussed earlier. It checks for the preposition following it, if yes, then it is added to the root word. An example from the cleaned sentences can be found in figure 18, where the output of the sentence is "aim", which is the verb of the sentence.

Finally, after extracting the entities, we now have the subjects, objects, and edges of all the cleaned sentences. Using the pandas library, a data frame is built where it includes all the entities of all the cleaned sentences.

The figure below shows the output including the entities of all the first 30 cleaned sentences.

```
subjects = [item[0] for item in entity_pairs]
objects = [item[1] for item in entity_pairs]

df_graph = pd.DataFrame({'subjects':subjects, 'objects':objects, 'edge':relations})
df_graph[0:30]
```

	subjects	objects	edge
0	israel	entire country march success	aim
1	insider logo	insider	word
2		click perform search	indicate
3			copyright
4	global healthcare strategy story	vtl6vsddvideonielsenapidp5982de1d31c5	tell global
5			story
6	politic need		know
7			country
8	instagram icon		camera
9	germanys health minister	fair share	insist
10	eu canada	donate vaccine poor country	pledge
11	france germany spain	administer vaccine	add
12	nl	polska	za
13	else _ uspapi push	slice var r	try
14	european union week	drug giant uk vaccine supply	clash
15			policy
16	rich country	ahead vaccination program	mean rich
17	anti tabloid press uk	heavily row	splash
18			x
19			word
20	partner offer	service privacy policy	accept
21	wrong vaccine nationalism	single coronavirus nothing	citizen
22	india italia japan	focus digital firm focus transformation	maxico
23	furious britain	order time	delay
24	ability	ehemail	indicate
25	close interaction	notification	indicate
26	icon world globe	international option	indicate different
27	global supply chain	rich country	depend
28	fm	button	get
29	twitter icon shape person	shoulder	rss

Figure 19: Extracted entities of the cleaned sentences

1.4-

In this part of the report, the RDF framework will be used to create the knowledge graph. This takes into consideration all the entities that have been extracted as well as connecting the python program to DBpedia, which is a website that extracts the structured content from the information in Wikipedia.

"rdflib" is the library used for such task. The next steps will be walking through it step by step.

```
1 import rdflib
2 from rdflib import Graph    #Library to use RDF
3 from rdflib.parser import Parser
4
5 g = rdflib.Graph()
```

Figure 20: Importing the RDF library and creating a graph

Looking at figure 20 above, the RDF libraries are imported as well as the parser from the library. Then, a graph given the name "g" is initialised and will be the platform to work on.

```
[53]  1 result = g.parse('http://dbpedia.org/resource/Coronavirus')

[51]  1 print(f'graph has {len(result)} facts')

      graph has 1171 facts
```

Figure 21: Parsing the graph to DBpedia and printing its facts

The next step is parsing the RDF graph from an external source, looking at the URL provided in figure 21, it is a link from DBpedia which has structured information about the covid-19 and the vaccines. Opening the link will lead to the page on DBpedia which includes the entities and structured data about this topic.

Each graph has different facts which are intended in the entities, for the graph parsed in the DBpedia page, the graph has 1171 facts, facts is also known as the triples in the graph, which is important to know in order to visualise the size of the graph and the number of information it holds.

```
1 for index, (sub, pred, obj) in enumerate(g):
2   print(sub, pred, obj)
3   if index == 10:
4     break
```

Figure 22: Iterating on the triples in the graph

Looking at figure 22, it can be seen that there is a for loop that iterates on the triples of the graph, the ripples are the subjects, predicates, and the objects and breaking at index number 10.

The output of such iteration will generate the URIs for the first 10 triples. After visualising all the triples which are shown as URIs. The figure below shows the output form.

```
1 for index, (sub, pred, obj) in enumerate(g):
2   print(sub, pred, obj)
3   if index == 10:
4     break
```

```
http://dbpedia.org/resource/Coronavirus http://www.w3.org/2002/07/owl#sameAs http://rdf.freebase.com/ns/m.01cpyy
http://dbpedia.org/resource/Coronavirus http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Virus
http://dbpedia.org/resource/Yellowhead_disease http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Coronavirus
http://dbpedia.org/resource/Coronavirus http://dbpedia.org/property/wikiPageUsesTemplate http://dbpedia.org/resource/Template:See_also
http://dbpedia.org/resource/2019–20_West_Bromwich_Albion_F.C._season http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Coronavirus
http://dbpedia.org/resource/Richard_Tsoi http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Coronavirus
http://dbpedia.org/resource/Li-Meng_Yan http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Coronavirus
http://dbpedia.org/resource/Coronavirus http://dbpedia.org/property/from Q290805
http://dbpedia.org/resource/Baltic_Pride http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Coronavirus
http://dbpedia.org/resource/Coronavirus http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Orthornavirae
http://dbpedia.org/resource/IDX-184 http://dbpedia.org/ontology/wikiPageWikiLink http://dbpedia.org/resource/Coronavirus
```

Figure 23: Output when enumerating over the graph triples (first 10 indices)

Serializing using the Turtle Method:

The next step is to represent the graph by using the serialization methods, the process of serialization is RDF convers the object into a persistent form. It is also a method for transmitting and storing all the triples information in the graph. In this project, the used serialization methods are turtle methods decoding to u8. The turtle serialization method is part of the SPARQL. Figure 24 below shows the serialization method and its output. The output presented is a huge string including all the triples from DBpedia, it is also visible in all available languages which can be visualised in the output terminal, it can later be passed to a variable and saved to a file later on.

```
1 print(g.serialize(format='ttl').decode('u8'))
```

```
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://dbpedia.org/resource/2002> dbo:wikiPageWikiLink <http://dbpedia.org/resource/Coronavirus> .

<http://dbpedia.org/resource/2002–2004_SARS_outbreak> dbo:wikiPageWikiLink <http://dbpedia.org/resource/Coronavirus> .

<http://dbpedia.org/resource/2010s_in_health_and_society> dbo:wikiPageWikiLink <http://dbpedia.org/resource/Coronavirus> .

<http://dbpedia.org/resource/2012_Middle_East_respiratory_syndrome_coronavirus_outbreak> dbo:wikiPageWikiLink <http://dbpedia.org/resource/Coronavirus>

<http://dbpedia.org/resource/2012_in_science> dbo:wikiPageWikiLink <http://dbpedia.org/resource/Coronavirus> .

<http://dbpedia.org/resource/2017_Fresno_shooting_spree> dbo:wikiPageWikiLink <http://dbpedia.org/resource/Coronavirus> .
```

Figure 24: Output after serializing using Turtle

Serializing using the N-Triples Method:

The N-triples format is also for storing and transmitting the data, differently, it is a plain text serialisation format in the RDF – it is a part of turtle. Used for encoding the full RDF graph. The process involved the creation of nodes with URI references. It is used to parse the URI in the knowledge graph.

The output of the serialization method using the N-triples can be find in the next figure below where a separate graph is created

```
1 g1 = rdflib.Graph()
2 result = g1.parse('https://dbpedia.org/resource/COVID-19_vaccine')
```

```
[86] 1 from rdflib.namespace import RDF, RDFS, FOAF
     2
     3 for topic in g1.subjects(RDF.type, FOAF.topic):
     4   for ent in g1.objects(topic, RDFS.label):
     5     print(ent)
```

```
[87] 1 print(g1.serialize(format='n3').decode('utf-8'))
```

```
<http://dbpedia.org/resource/Virus-like_particle>,
<http://dbpedia.org/resource/Virus_inactivation>,
<http://dbpedia.org/resource/Visp>,
<http://dbpedia.org/resource/Western_African_Ebola_virus_epidemic>,
<http://dbpedia.org/resource/Western_world>,
<http://dbpedia.org/resource/World_Health_Organization> ;
dbp:align "left"@en ;
dbp:description "604800.0"^^<http://dbpedia.org/datatype/second>,
    "--09-14"^^xsd:gMonthDay,
    "Dose-dependent neutralizing antibody response on two-dose schedule; undetermined durability. Adverse effects: fever, fatigue, headache, muscle ache
    "Double-blind, randomized, placebo-controlled to evaluate efficacy and safety in Brazil ; Chile ; Indonesia ; Turkey"@en,
    "Interventional; randomized, double-blind, placebo-controlled"@en,
    "Interventional; randomized, placebo-controlled study for efficacy, safety, and immunogenicity"@en,
    "Interventional; randomized, placebo-controlled study for efficacy, safety, and immunogenicity."@en,
    "Neutralizing antibodies at day 14 after 2 injections;  Adverse effects: injection site pain and fever, which were mild and self-limiting; no seriou
    "Neutralizing antibody and T cell responses. Adverse effects: moderate over 7 days: 74% had fever, pain, fatigue"@en,
    "Non-randomized, single-center, two doses"@en,
    "Partially observer-blind, multicenter, controlled, dose-confirmation"@en,
    "Preprint. IgG and neutralizing antibody response with adjuvant after booster dose. Adverse effects: short-duration, low grade, local pain, headache
    "Preprint. Immunogenicity eliciting 92% seroconversion at lower dose; Adverse effects: mild in severity, pain at injection site"@en,
    "Randomised, double-blind, placebo-controlled, dose-ranging"@en,
```

Figure 25: Output after serializing using N-Triples method

Finally, after implementing all those steps, the knowledge graph is then designed using all the information provided by this project.

Figure 26 below, shows the final knowledge graph using MatplotLibrary, it is human-readable, and it is easy to spot the nodes and their edges.

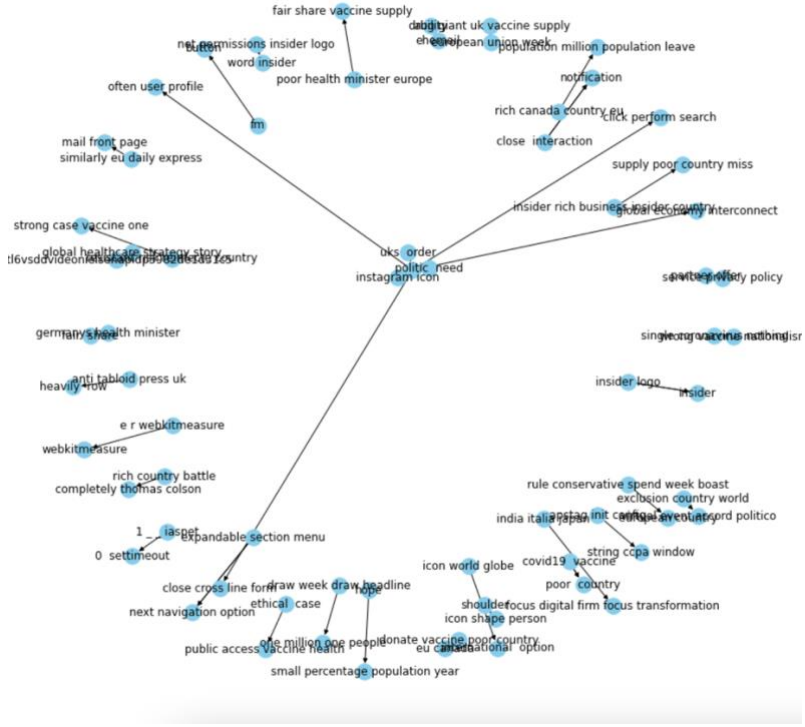

Figure 26: Knowledge Graph implemented by the system

Please find the notebook with the code in the following link:

https://drive.google.com/file/d/1TMzUjOjpIQ9yZgTiCqKncYlDpCLHDjbR/view?usp=sharing

I have also sent you the notebook by email. Please let me know if there are any issues.