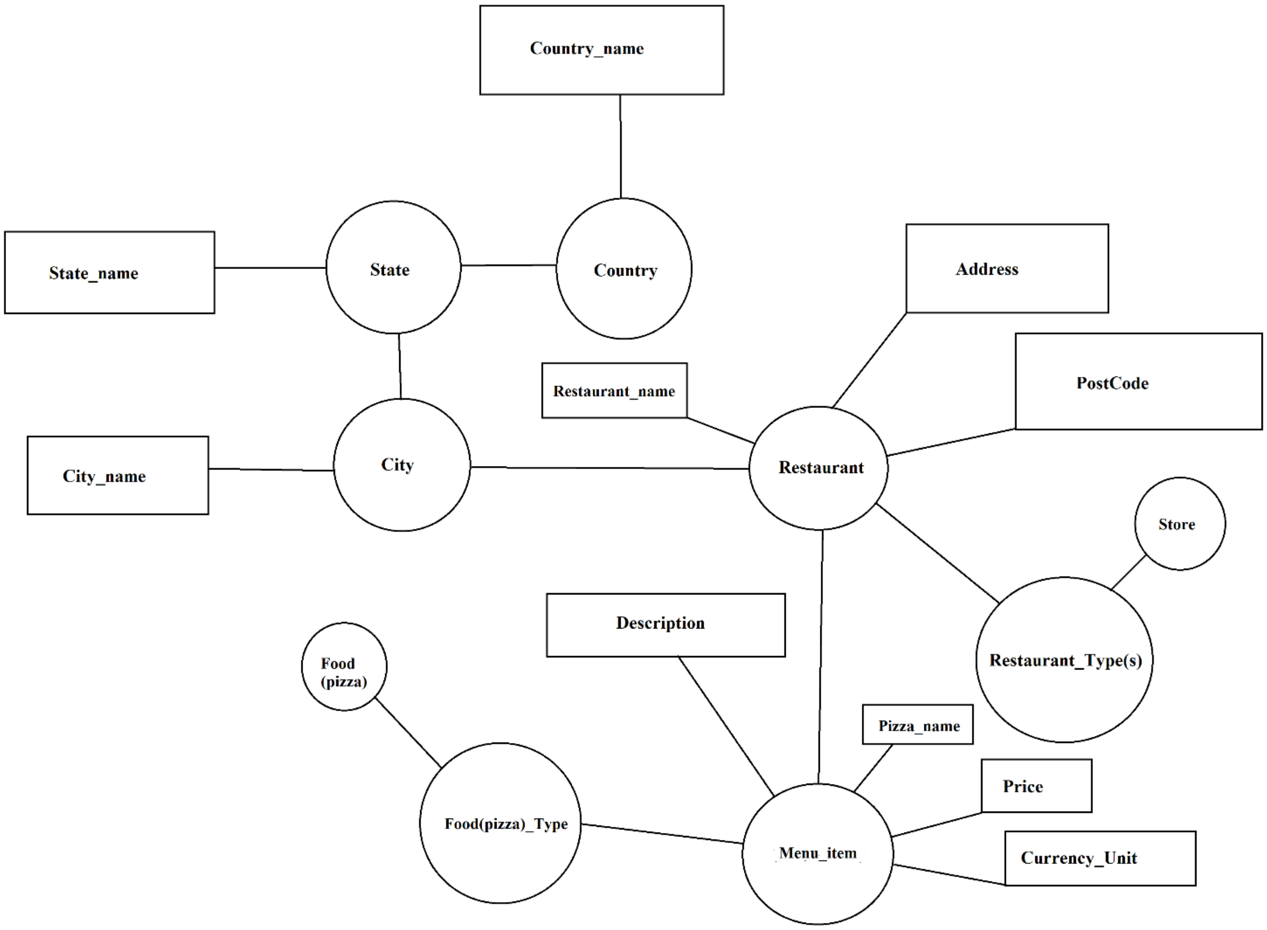
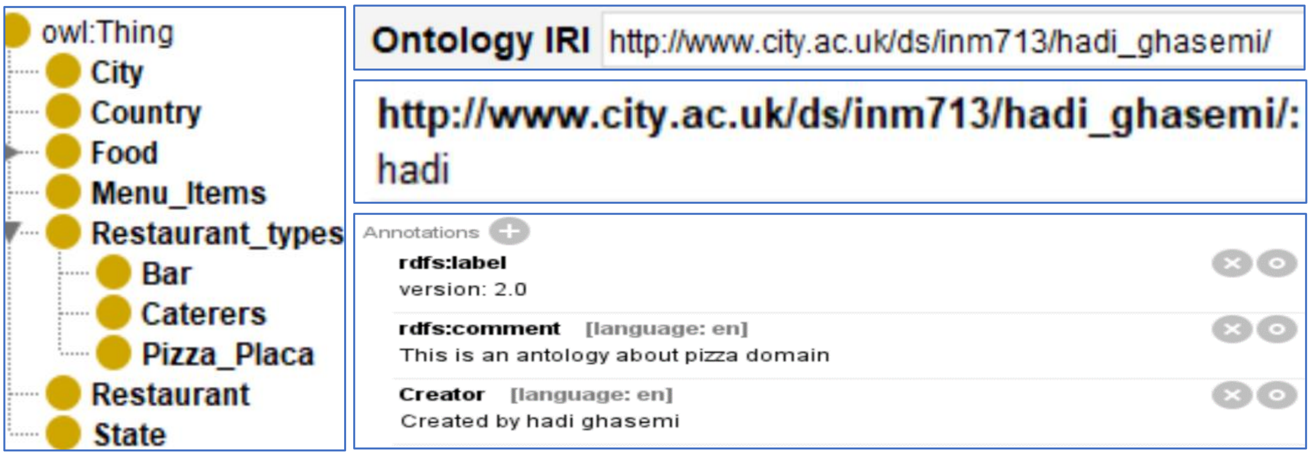


Figure 1:Initial ontology modeling

In the first task, it was requested to create a small ontology for a dataset that models the domain of pizza data. Since this modeling is done manually and by a protege, it was required to prepare a small model that generally presents key concepts. For example, regarding the types of restaurants and types of pizza, only a few examples will be mentioned, and the model will only have general concepts, with no presentation of specific examples and individuals.

To build the model, various ideas are considered. For instance, one option is to consider a restaurant or a shop as the main entity and add other concepts based on their relationships with it. Another approach is to consider hierarchical relationships starting from the country, where the country includes states, states include cities, cities include buildings, restaurants being an instance of buildings, and restaurants include menus, menus include dishes, and dishes include pizzas.



Ontology IRI http://www.city.ac.uk/ds/inm713/hadi_ghasemi/

http://www.city.ac.uk/ds/inm713/hadi_ghasemi/:hadi

rdfs:label
version: 2.0

rdfs:comment [language: en]
This is an antology about pizza domain

Creator [language: en]
Created by hadi ghasemi

Figure 2: Initial ontology created by protege

We have decided to consider this ontology with two main entities: restaurants and food. Food includes various types, with pizza being one of them, and restaurants also have different types. Each restaurant is connected to its geographical attributes, such as address, postal code, city, etc. Food, in this case, with pizza as the only type, is linked to its characteristics, such as pizza type, price, currency unit, etc., through relationships.

The relationship of each pizza, as a type of food, with a restaurant is defined as a specific relationship. To reduce redundancy and achieve more structure, a restaurant is only associated with the city it is located in, and information about the state and country of the restaurant can be indirectly accessed through the relationship of that city with its state and country.

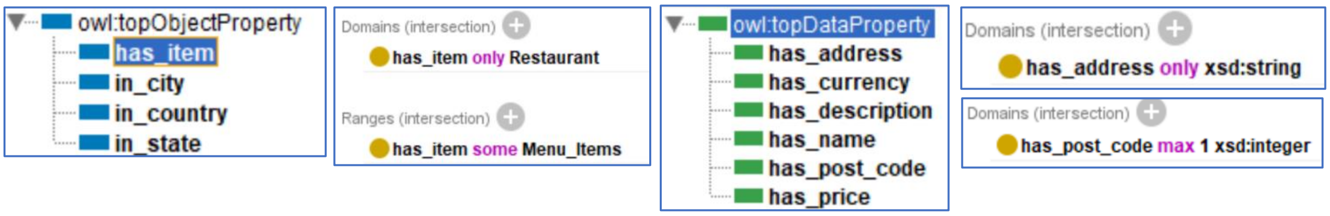
The reason for adopting this architecture for the ontology is its similarity to the ontology of pizza from Stanford University. We chose this approach because we wanted to design an ontology that is compatible and well-connected with other ontologies in its domain. This way, if necessary in the future, we can integrate them to benefit from each other's information.

By structuring our ontology in a way that aligns with established standards and practices, we ensure that it can easily interact with other ontologies and datasets, promoting interoperability and data exchange. This approach facilitates collaboration and data sharing within the domain, making it easier to combine and leverage knowledge from various sources and ontologies effectively.

Figure 1 illustrates the general and initial concept for building the ontology. Subsequently, efforts will be made to implement this concept in Protégé. To prevent the image from becoming cluttered and simplify the ontology, the naming of properties has been omitted in the figure 1. The hierarchy of created classes is as below:

- Food
    - Pizza
        - American_pizza
        - Italian_pizza
        - …

- Restaurant
- Menu_item



owl:topObjectProperty
has_item
in_city
in_country
in_state

Domains (intersection)
has_item only Restaurant

Ranges (intersection)
has_item some Menu_Items

owl:topDataProperty
has_address
has_currency
has_description
has_name
has_post_code
has_price

Domains (intersection)
has_address only xsd:string

Domains (intersection)
has_post_code max 1 xsd:integer

Figure 3: Object and data properties and their restrictions

- City
- State
- Country
- Restaurant_types
    - Bar
    - Caterers
    - …

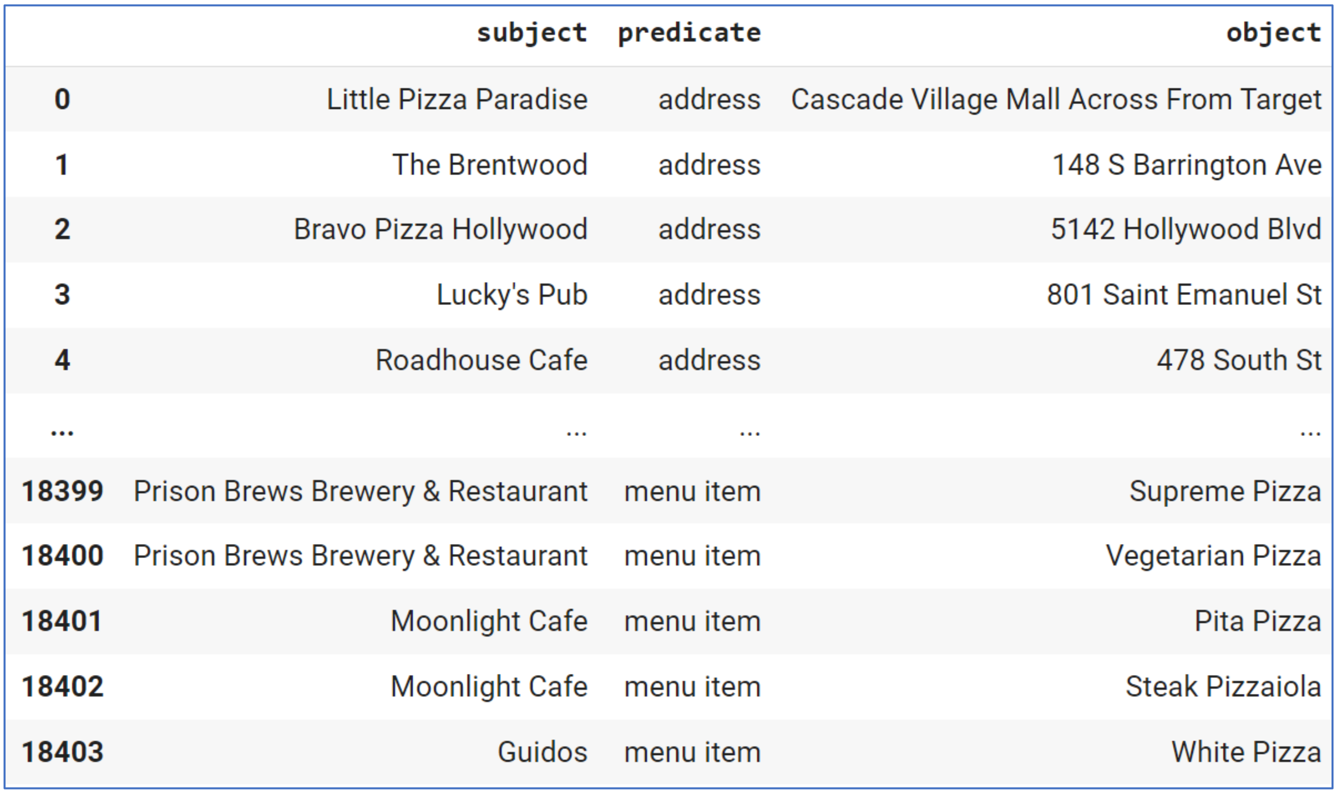
In continuation, object properties and data properties were created, and their ranges and domains were defined where necessary (figure 3). Cardinality constraints were also applied. Finally, the ontology was extracted in Turtle format and saved as "ontology.ttl." This ontology represents a simple model of the pizza domain, implemented in stage 3. A more detailed model, along with individual data, will be automatically generated as a knowledge graph using Python.

In connection with the pizza domain and modeling, the question arises as to what types of restaurants exist in the dataset. To answer this question, text analysis was performed on the relevant column in the dataset, and the types of restaurants were extracted separately. Each restaurant entity can be connected to multiple restaurant types through a many-to-many relationship because each restaurant may belong to multiple categories.

A similar issue arises concerning pizza, and it was addressed by determining that each pizza has a specific type, such as Sicilian or Margherita, which needs to be identified. To identify this issue, the name of each pizza was analyzed and processed in Python.

Regarding the cardinality of relationships, the question was raised about what cardinality relationships should have. To address this, some constraints were identified through data frame analysis in pandas. For example, some restaurants do not have postal codes, so the cardinality of the postal code should be 0-1. However, each menu item is definitely in one restaurant, and each restaurant may have multiple menu items, making the cardinality of this relationship 1 on the restaurant side and * on the menu item side.

Since types like restaurants, shops, buildings, etc., are considered non-rigid classes, it was decided not to establish inheritance relationships with specific restaurant types such as bar, Italian, and others. The reason for this decision is that the restaurant type is a general concept, and it's possible for the rigidity property to be lost, as per which rigid classes must not be subclasses of non-rigid classes. In other words,



	subject	predicate	object
0	Little Pizza Paradise	address	Cascade Village Mall Across From Target
1	The Brentwood	address	148 S Barrington Ave
2	Bravo Pizza Hollywood	address	5142 Hollywood Blvd
3	Lucky's Pub	address	801 Saint Emanuel St
4	Roadhouse Cafe	address	478 South St
...	...	...	...
18399	Prison Brews Brewery & Restaurant	menu item	Supreme Pizza
18400	Prison Brews Brewery & Restaurant	menu item	Vegetarian Pizza
18401	Moonlight Cafe	menu item	Pita Pizza
18402	Moonlight Cafe	menu item	Steak Pizzaiola
18403	Guidos	menu item	White Pizza

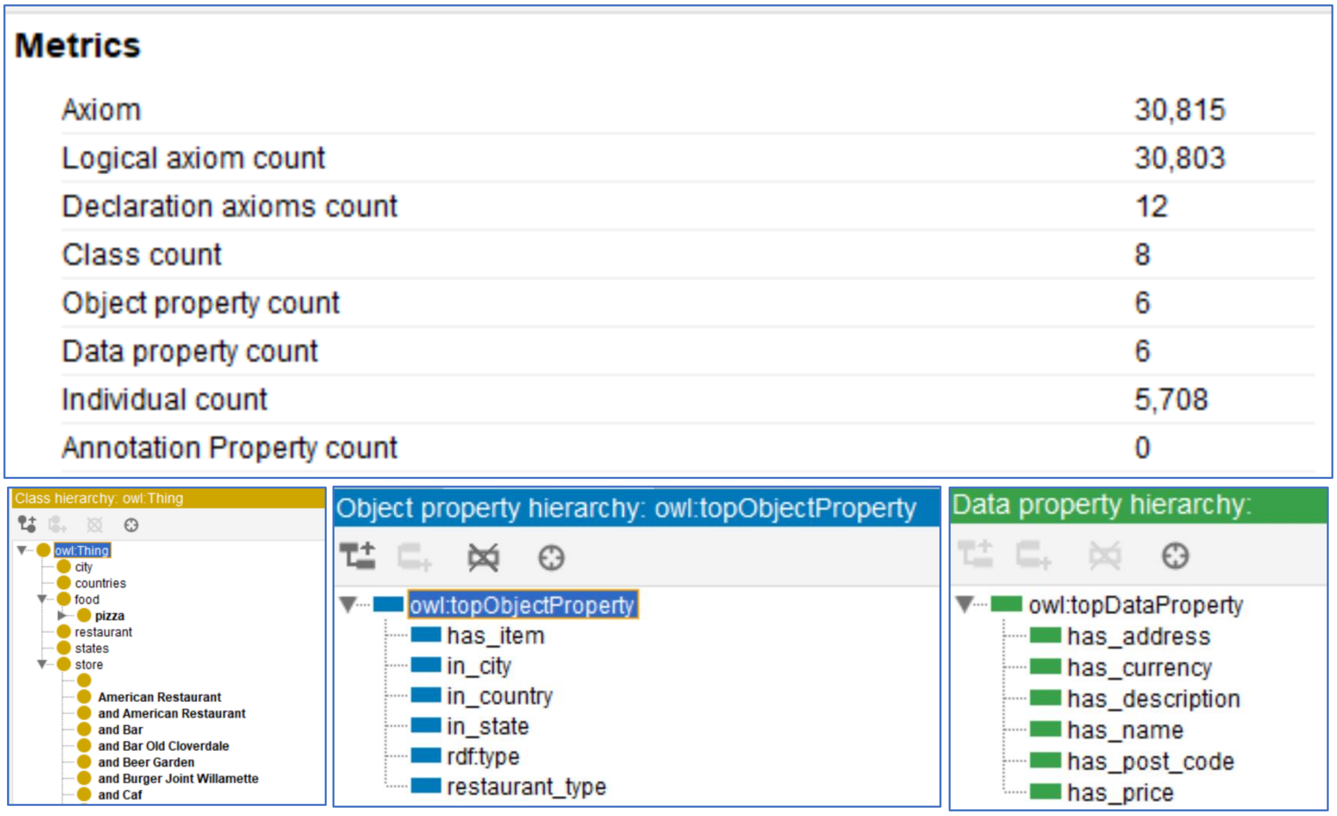
Figure 4: triples table

non-rigid classes may not have well-defined boundaries, and their characteristics or categories can change over time, making it inappropriate to create strict inheritance relationships with them. This approach allows for greater flexibility and adaptability in representing the evolving nature of these classes within the ontology. Indeed, since concepts like food and pizza are considered rigid classes, inheriting from them does not pose any issues related to the rigidity property.

To establish identity feature when distinguishing pizzas with the same name, there was indeed an issue. Some shops might have food items listed as "pizza" on their menus, which resulted in a lack of differentiation between instances. However, by creating unique and restaurant-related names for pizza entity instances, the problem was addressed. This ensures that each instance of pizza has a unique identity in the data, minimizing confusion in distinguishing pizzas with the same name. also the property of unity applies to the pizza ontology. In other words, all entities within the ontology are considered "whole entities" where all their parts are related to each other in the same way.

In the 3rd stage, it was determined that there are 65 records missing postal code values. However, it should be noted that these 65 records belong to menu items of restaurants that do not have a postal code. The number of restaurants may not necessarily be 65, and it is likely to be significantly lower because each restaurant may have multiple items in the menu.

Further, text analysis was performed on the specifications of the pizza items in the restaurants' menus. More than 50 types of pizza were extracted, and a new column was added to the table to indicate the



Metrics

Axiom	30,815
Logical axiom count	30,803
Declaration axioms count	12
Class count	8
Object property count	6
Data property count	6
Individual count	5,708
Annotation Property count	0

Figure 5: Created knowledge graph in task3

type of each pizza item in a restaurant. This process involved several steps to classify the 2500 unidentified pizza types into approximately 2300 categorized pizzas and 1300 uncategorized pizzas.

Furthermore, the restaurant categories were identified by splitting the content of the category column using commas, and they were added one-to-one to the tripples. This approach allows for a systematic and structured representation of restaurant categories, making it easier to classify and organize restaurant entities. As depicted in Figure 4, the triple table was ultimately extracted to be used in the next stage for building a knowledge graph. This triple table likely represents the relationships and associations between entities and their attributes in a structured format, which can serve as a foundation for constructing a knowledge graph to represent the pizza domain more comprehensively and semantically.

Now, after performing the necessary processing on the SPARQLWrapper library table, rdflib was installed to use them for knowledge graph construction. As requested, well-known and popular knowledge graphs were used to extract entities related to countries, cities, and states. We wrote some functions that are used to retrieve the URIs of entities from different knowledge bases (DBpedia, Wikidata, and Google's Knowledge Graph) based on their labels or alternative labels.

During the stages of building the knowledge graph from the dataset table, the ontology created in Task 1 served as a guideline. The goal was to stay as faithful as possible to the names, URI addresses, classes, and relationships defined in that ontology. However, there might be a few minor differences in the implementation.

The knowledge graph created, named RDF.ttl, is stored for further use (figure 5). Additionally, the code for this section is available with the .ipynb extension because Google Colab was used for processing in this part. This approach allows for easy access to and execution of the code within a Jupyter Notebook environment, which is commonly used for data processing and analysis tasks.

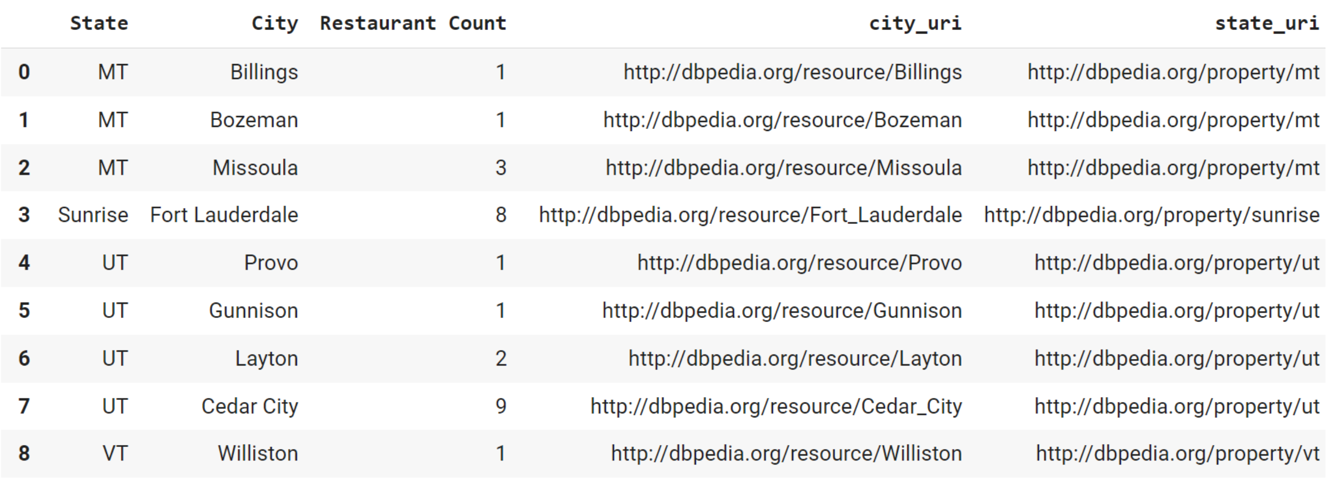
In the next step, to answer the asked queries, sparqle queries were written based on the model of the knowledge graph:

1. Return number of restaurants by city, sorted by state and number of restaurants:

Query:

```
SELECT ?stateName ?state ?city ?cityName (COUNT(?restaurant) AS
?restaurantCount)
WHERE {
  ?restaurant rdf:type n:restaurant ;
              n:has_address ?address ;
              n:in_city ?city .
  ?city n:in_state ?state .
  ?state n:has_name ?stateName .
  ?city n:has_name ?cityName .
}
GROUP BY ?state ?city ?stateName ?cityName
ORDER BY ?state ?restaurantCount
```

Results:



	State	City	Restaurant Count	city_uri	state_uri
0	MT	Billings	1	http://dbpedia.org/resource/Billings	http://dbpedia.org/property/mt
1	MT	Bozeman	1	http://dbpedia.org/resource/Bozeman	http://dbpedia.org/property/mt
2	MT	Missoula	3	http://dbpedia.org/resource/Missoula	http://dbpedia.org/property/mt
3	Sunrise	Fort Lauderdale	8	http://dbpedia.org/resource/Fort_Lauderdale	http://dbpedia.org/property/sunrise
4	UT	Provo	1	http://dbpedia.org/resource/Provo	http://dbpedia.org/property/ut
5	UT	Gunnison	1	http://dbpedia.org/resource/Gunnison	http://dbpedia.org/property/ut
6	UT	Layton	2	http://dbpedia.org/resource/Layton	http://dbpedia.org/property/ut
7	UT	Cedar City	9	http://dbpedia.org/resource/Cedar_City	http://dbpedia.org/property/ut
8	VT	Williston	1	http://dbpedia.org/resource/Williston	http://dbpedia.org/property/vt

2. Return the list of restaurants with missing postcode:
   Query:

```
SELECT  ?name
WHERE {
  ?restaurant rdf:type n:restaurant ;
              n:has_address ?address ;
              n:has_name ?name .
  OPTIONAL {
    ?restaurant n:has_post_code ?postcode .
  }
  FILTER (!bound(?postcode))}
```

Results:

```
Restaurant 1: Ak Diamonds
Restaurant 2: Baldinelli Pizza
Restaurant 3: Five Below
Restaurant 4: Masago
Restaurant 5: Milt's Pizza Place Llc
Restaurant 6: Pizza City
Restaurant 7: San Biagio's Pizza
Restaurant 8: Sir Pizza
Restaurant 9: Two Brothers Deli
Restaurant 10: Valley Lahvosh Baking
Restaurant 11: Villa Rose Pizza
```

3. Return the average prize of a Margherita pizza
   Query:

```
PREFIX pt: <http://www.city.ac.uk/ds/inm713/hadi_ghasemi/pizza_types/>

SELECT (AVG(?price) AS ?averagePrice)
WHERE {
  ?pizza rdf:type pt:margherita;
  n:has_price ?price .
}
```

   Results:

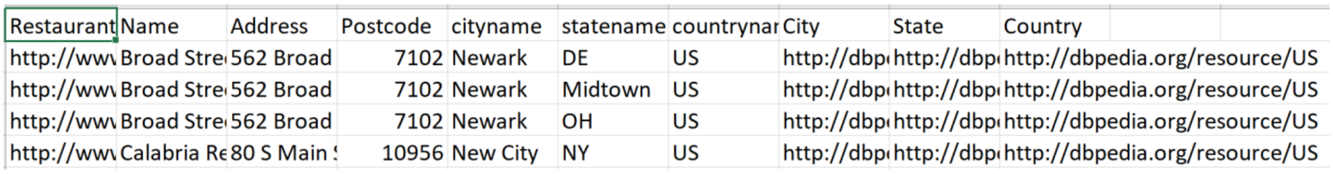
Average Price: 15.54 USD

4. Return all the details of the restaurants that sell pizzas without tomate:

```
PREFIX pt: <http://www.city.ac.uk/ds/inm713/hadi_ghasemi/pizza_types/>

SELECT ?restaurant ?name ?address ?rsname ?city ?cityname ?state ?statename
?country ?countryname ?postcode
WHERE {
  ?restaurant rdf:type n:restaurant ;
              n:in_city ?city ;
              n:has_address ?address;
              n:has_name ?rsname;
              n:has_post_code ?postcode .
  ?city n:has_name ?cityname;
        n:in_state ?state .
  ?state n:has_name ?statename ;
         n:in_country ?country .
  ?country n:has_name ?countryname.
  ?restaurant n:has_item ?menu .
  ?menu rdf:type pt:bianca .}
```

Results:



Restaurant	Name	Address	Postcode	cityname	statename	countrynar	City	State	Country
http://www	Broad Stre	562 Broad	7102	Newark	DE	US	http://dbp	http://dbp	http://dbpedia.org/resource/US
http://www	Broad Stre	562 Broad	7102	Newark	Midtown	US	http://dbp	http://dbp	http://dbpedia.org/resource/US
http://www	Broad Stre	562 Broad	7102	Newark	OH	US	http://dbp	http://dbp	http://dbpedia.org/resource/US
http://www	Calabria Re	80 S Main	10956	New City	NY	US	http://dbp	http://dbp	http://dbpedia.org/resource/US

Perform reasoning:

```
from owlready2 import *
onto_path.append("/content/")
onto = get_ontology("/content/RDF.owl").load()
onto.base_iri = "http://www.city.ac.uk/ds/inm713/hadi_ghasemi/"
sync_reasoner([onto])
onto.save(file = 'Extended_RDF.owl', format = 'rdfxml')
onto.save(file = "Extended_RDF1", format = "ntriples")
```

```
* Owlready2 * Running HermiT...
    java -Xmx2000M -cp /usr/local/lib/python3.10/dist-
packages/owlready2/hermit:/usr/local/lib/python3.10/dist-
packages/owlready2/hermit/HermiT.jar org.semanticweb.HermiT.cli.CommandLine -c
-O -D -I file:////tmp/tmpkrvos7ih
* Owlready2 * HermiT took 13.198657751083374 seconds
* Owlready * (NB: only changes on entities loaded in Python are shown, other
changes are done but not listed)
```

The new ontology developed has been read by Protégé and saved in Turtle format.
To algin the Stanford pizza ontology and created ontology by ours, In the first step, the class and property lists of both ontologies are extracted and considered which one shares a same concept and semantic (table 1).

Table 1: Equivalent entities detected by user

Pizza.owl	Created ontology
Classes	
Pizza:country	Hadi:countries
Pizza:pizza	Hadi:food/pizza
Pizza:food	Hadi:food
Pizza:Margherita	Hadi:pizza_types/margherita
Pizza:Mushroom	Hadi:pizza_types/mushroom
Pizza:siciliana	Hadi:pizza_types/sicilian
Pizza:CheeseyPizza	Hadi:pizza_types/cheese
Pizza:MeatyPizza	Hadi:pizza_types/meat
Pizza:RealItalianPizza	Hadi:pizza_types/italian
Pizza:VegetarianPizza	Hadi:pizza_types/veggie

Then, by using equivalentclass in the rdflib, we align these two ontologies after parsing them and save the result as Ontology aligned_by_user.ttl.

As an alternative method for alignment between these two ontologies, the AML software was used. To do this:

1. The two ontologies were provided as input files to the AML software.
2. The "Automatic Match" operation was performed on these two ontologies within the AML software.
3. The output was saved in RDF format.
4. The RDF output was subsequently converted to Turtle format using Protégé.

This process likely helped in aligning and mapping concepts and relationships between the two ontologies, enabling better interoperability and integration of knowledge between them.

As another method, you can execute the LogMap 2 program using the following command:

```
java -jar logmap-matcher-4.0.jar MATCHER
http://krrwebtools.cs.ox.ac.uk/logmap/ontologies/matching_10_09_2023/RDF.owl
http://krrwebtools.cs.ox.ac.uk/logmap/ontologies/matching_02_07_2023/Pizza.owl C:/ true
```

At the moment of writing this report, the above-mentioned command, which should execute correctly according to the LogMap guidelines, does not provide the expected output. In the event of identifying an error and resolving it, the output file will be attached by LogMap as evidence.

For the task of embedding, we use owl2vec_star and by configuring the default.cfg file and using the following code, the program gets executed, and the embedding process is carried out. The ontology created in the previous stages is used as input in the file and three different configurations were tested and three different outputs were produced which are shared as Ontology embeddings{i}.bin(/.txt) which indicates the number of corresponding configuration. Table 2 shows the configurations of different embeddings.

```
python OWL2Vec_Standalone.py -config_file default.cfg
```

Table 2: different configuration details for embeddings

	Conf 1	Conf 2	Conf 3
Embed size	100	80	150
Iterations	10	8	15
Window	5	2	10
Negative	25	15	35
Seed	42	25	100

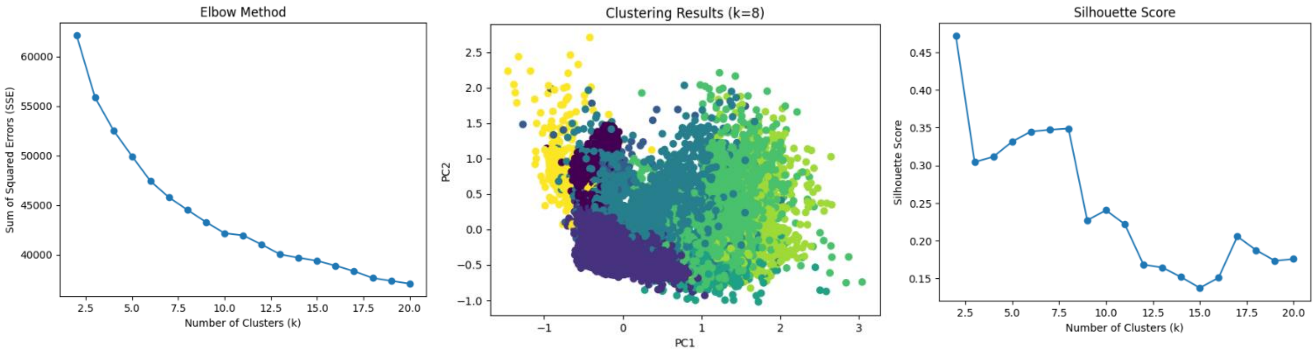


Figure 6: clustering on embedding vectors

In the next step, embeddings generated from the ontology were extracted, and clustering operations were performed on them. Figure 6 shows two silhouette (left side) and elbow (right side) diagrams for different numbers of clusters in the K-means method.

Based on these two diagrams, the number k = 8 was selected as an approximate breaking point in the elbow diagram and as one of the maximum silhouette values in the silhouette diagram. This value of k was chosen as the appropriate number of clusters for the clustering process. The clustering result with 8 clusters is displayed in Figure 6 (middle).

In the next step, 5 pairs of embedded words were selected, and their embedding vectors were compared in terms of cosine similarity. Table 3 displays the cosine similarity results for these pairs of embedded words. The codes of this step exist in the main jupyter file named main.ipynb.

Table 3: similarity of chosen pairs

Entity 1	Entity 2	Cosine similarity
margherita	pizza margherita	0.79
margherita	bianca	0.61
margherita	US	0.39
US	http://dbpedia.org/resource/US	0.43
margherita	http://dbpedia.org/resource/US	0.29

By comparing the obtained similarity scores, we can observe that the term "margherita" has the highest similarity with the phrase "pizza margherita." This is logical since "margherita" within the pizza ontology is expected to have a higher similarity to the specific concept of "pizza margherita" than the general concept of "pizza." Additionally, as the contents of "pizza bianca" can be similar to "pizza margherita," we would expect its similarity score to be higher than the general concept of "pizza" but lower than "pizza margherita," which is indeed the case.

Another interesting point is the similarity score between "margherita" and "us" that is relatively low. However, what is noteworthy here is that the similarity score between "margherita" and its corresponding URI is higher than the similarity between "margherita" and "us." Similarly, the similarity between "us" and its URI is also higher. However, the lowest similarity score is between "margherita" and the URI corresponding to "US".