KEN 4256 Knowledge Graphs Assignment 1

Georgios Ioannidis, Giacomo Rivetti, Guilherme De Sequeira Antunes

February 13, 2026

1 Introduction

1.1 Goal

The primary objective of this project was to design and implement a Knowledge Graph (KG) that integrates heterogeneous data sources, more specifically recipes, nutritional facts, and restaurant information, alongside unstructured user reviews. The system aims to establish semantic interoperability, enabling complex querying that links dietary information with commercial food availability and user sentiment.

1.2 Scope

The Knowledge Graph implementation is built using `RDFLib` in Python, orchestrating the integration of three structured CSV datasets (Recipes, Nutrition, and Restaurants) with unstructured user reviews.

The technical scope of the solution includes:

- **Vocabulary Standardization:** Adopting `schema.org` as the foundational vocabulary to ensure semantic interoperability across the disjoined datasets.
- **NLP Pipeline:** Implementing a rule-based sentiment analysis pipeline using `VaderSentiment`. This allowed for the efficient extraction of sentiment polarity from user reviews, linking subjective user feedback to structured recipe data.
- **External Enrichment:** Extending the graph's semantic depth by linking recipe ingredients to external entities in **Wikidata** via SPARQL wrappers, facilitating future cross-domain querying.

2 Methodology

2.1 Task 1: Develop a vocabulary for semantic interoperability

Four core classes were identified from the datasets: **Recipe**, **Nutrition**, **Restaurant**, and **Review**. Two additional classes were created because Tasks 3.3 and 3.4 would be difficult to solve without them: **Ingredient** and **Cuisine**. Key object properties connect these classes: **hasNutrition** links Recipe to Nutrition, **servesCuisine** connects Restaurants and Recipes to Cuisine, **UserReview** links Reviews with Recipes, and **recipeIngredient** links Recipes to their Ingredients.

Schema.org was used as the primary vocabulary due to its wide coverage of food-related entities and semantic web interoperability. Reused terms include **Recipe**, **Restaurant**, **calories**, **proteinContent**, **sugarContent**, **fiberContent**, **ingredients**, **cookTime**, **datePublished**, **priceRange**, and **AggregateRating**.

The schema is implemented using **rdflib** in Python. Classes are declared as `owl:Class` instances, and properties as `owl:DatatypeProperty` or `owl:ObjectProperty` with `rdfs:domain` and `rdfs:range` constraints to ensure type safety.

2.2 Task 2: Construct the KG using structured data

A collision occurred between the names of nutrition items and recipes, which caused the Turtle serialization to produce mixed-up instances. To resolve this, we created separate namespaces based on our base namespace `http://kg-course.io/food-nutrition/`:

- `http://kg-course.io/food-nutrition/` (FOD) — general entities
- `http://kg-course.io/food-nutrition/nutrition/` (NUTR) — nutrition instances
- `http://kg-course.io/food-nutrition/recipe/` (REC) — recipe instances

We encountered issues with data cleaning at the beginning of the assignment. Standard `strip` and `replace` methods were insufficient, especially for very long or messy strings. To address this, we created a URI sanitization function called **clean_for_uri**, which removes problematic characters such as parentheses, commas, and spaces, and encodes data points so they can safely be used as RDF URIs.

Additionally, a function called **extract_first_image_url** was implemented to extract only the first image URL for each recipe. This was necessary because some image fields contained multiple URLs, and the separation was inconsistent. Full URLs containing commas were being incorrectly split, so extracting the first URL ensured proper processing.

All relevant columns needed for the knowledge graph (to make Task 4 solvable) were converted to RDF triples. For each entity:

- A unique URI is created from its name.
- The entity is typed via `rdf:type`.
- Column values are assigned appropriate XSD datatypes (e.g., float, string).

Recipe-to-Nutrition linking uses exact name matching between recipe names and nutrition entries, creating **hasNutrition** triples when matches are found. Recipe-to-Cuisine linking uses partial keyword matching: each recipe keyword is checked against restaurant cuisine fields, and matches generate **servesCuisine** triples linking recipes to cuisine type URIs.

AI assistance helped in creating the **clean_for_uri** function, ensuring URI consistency. The output from Claude was carefully examined and tested. AI also helped with the partial matching of recipe keywords with cuisines, improving the accuracy of the **servesCuisine** links. Beyond that, AI tools were used for grammar, clarity, and coherence checks to improve the quality of the report.

2.3 Task 3: Enrich the KG with unstructured data and external KGs

The primary objective of this task was to populate the Knowledge Graph (KG) using information extracted from unstructured textual sources (specifically the `Reviews.txt` file). This process involved extracting user sentiment, recognizing entities (ingredients), and performing semantic interlinking with external knowledge bases like Wikidata. We utilized the **spaCy** library with the `CoreWebSm` model to analyze the review texts.

For each review, a polarity score was calculated using **VADER (Sentiment Intensity Analyzer)**. Results were categorized into positive, neutral, and negative labels. To comply with Linked Data principles, ingredients were not stored as plain text (Literals) but as **URIRefs**.

The resulting Turtle file provides a semantically enriched view of the dataset. By transforming "dead text" into "clickable nodes" (URIs), we enabled complex queries that can correlate user sentiment with global ingredient properties defined on Wikidata.

2.4 Task 4: SPARQL queries

The Queries Implemented were designed to balance complexity and utility, focusing on common user needs like allergens, health, and cost efficiency where the following order was followed:

1.Identify allergen-risk recipes (mango).
2.List healthy pies under 2 hours.
3.Find Chinese restaurants with delivery in New Delhi.

4.Calculate average cost in Davenport for Asian cuisine.
5.Recommend top 5 low-calorie desserts.
6.Top 10 highly-rated beverages with sugar content.
7.Highest-rated protein-rich recipes served in UK restaurants.
8.Top 5 healthiest recipes (based on NDS) and restaurant availability.

The output of our queries matches the expected result for all cases for which we were able to provide a query, balancing the computational cost with the core objective of this assignment which was to obtain the results, however, to save space we limited the queries to values of 10 and 5 results in some cases. Further down you will be able to see some of queries as well as the output results of them.

2.4.1 Results for Query 4

Uses the AVG operation to calculate the cost of two for all queried restaurants.

```
# 4.4
run_query("4.4", "Average Cost (Davenport Asian)", """
SELECT (AVG(?price) AS ?Average_Cost) WHERE {
    ?rest a schema:Restaurant ; schema:city ?city ;
        schema:priceRange ?price ; schema:servesCuisine ?c .
    FILTER(STR(?city) = "Davenport")
    FILTER(REGEX(STR(?c), "Asian|Chinese|Thai", "i"))
}
""")
```

Figure 1: SPARQL Query 4: Finding the average cost of dining for two at restaurants in Davenport serving Asian cuisine.

```
================================================
Task 4.4: Average Cost (Davenport Asian)
================================================
Average_Cost
         2.0
```

Figure 2: SPARQL query results for Query 4: Finding the average cost of dining for two at restaurants in Davenport serving Asian cuisine.

2.4.2 Results for Query 5

To sort the recipes we based the query on specific keywords ('Dessert' and 'Easy') and nutritional constraints (less than 300 calories). The results are sorted by calories in ascending order to recommend the lightest options first, the link between schema:Recipe and schema:Nutrition was also used to access the calorie data.

```
# 4.5
run_query("4.5", "Easy Desserts (<300cal, after 2000)", """
SELECT ?Name ?Calories ?Image WHERE {
    ?recipe a schema:Recipe ; schema:givenName ?Name ; schema:recipeCategory ?cat ;
            schema:keywords ?k ; schema:hasNutrition ?n ; schema:image ?Image ; schema:datePublished ?date .
    ?n schema:calories ?Calories .
    FILTER(REGEX(STR(?cat), "dessert", "i") && REGEX(STR(?k), "easy", "i") && ?Calories < 300)
    FILTER(CONTAINS(STR(?date), "200") || CONTAINS(STR(?date), "201") || CONTAINS(STR(?date), "202"))
} ORDER BY ASC(?Calories) LIMIT 5
""")
```

Figure 3: SPARQL query results for Query 5: Recommending the top 5 low-calorie desserts labeled as 'Easy' and published after 2000.

```
================================================
Task 4.5: Easy Desserts (<300cal, after 2000)
================================================
                    Name Calories
Image
    Ice Cream Substitute      0.0
       frogs in the pond      0.0 https://img.sndimg.com/food/image/upload/w_555,h_416,c_fit,fl_progressive,q_95/v1/
img/recipes/11/56/4/picalRHtl.jpg
Julie's Chocolate Trifle     10.5
         Pumpkin Fritters     13.7
    Crystallized Flowers     17.2
```

Figure 4: SPARQL query results for Query 5: Recommending the top 5 low-calorie desserts labeled as 'Easy' and published after 2000.

2.4.3 Results for Query 6

Retrieves recipes classified as 'Beverages' and extracts their sugar content from the nutrition entity integrating unstructured data results by linking the recipe to its sentiment score (schema:reviewRating), we opted by ordering the results by sentiment descending to showcase the most highly-rated beverages.

```
# 4.6
run_query("4.6", "Top Beverages (Sentiment)", """
SELECT ?Name ?PrepTime ?Sugar (SAMPLE(?Sentiment) AS ?SentimentSample) WHERE {
    ?recipe a schema:Recipe ; schema:givenName ?Name ; schema:keywords ?k ;
            schema:cookTime ?PrepTime ; schema:hasNutrition ?n .
    ?n schema:sugarContent ?Sugar .
    ?review a schema:Review ; schema:UserReview ?recipe ; schema:reviewRating ?Sentiment .
    FILTER(CONTAINS(LCASE(?k), "beverage"))
} GROUP BY ?Name ?PrepTime ?Sugar
ORDER BY DESC(?Sugar) LIMIT 10
""")
```

Figure 5: SPARQL Query 6: Identifying the top 10 highly-rated beverages with their preparation time and sugar content.

```
==================================================
Task 4.6: Top Beverages (Sentiment)
==================================================
                                   Name PrepTime Sugar SentimentSample
                    Instant Coffee Kahlua     PT2H 602.0        positive
                            Best Lemonade     PT5M  77.2        positive
                    Elly May's Wedding Punch     PT3H  50.2        positive
Copycat Coffee House Whipped Cappuccino      nan  40.4        positive
                         Champagne Punch      nan  38.1        positive
                      Bobbing Apple Punch      nan  30.6        positive
                            Hot Cocoa Mix      nan  28.7        positive
                      Strawberry Lemonade      nan  22.8        positive
                            Hot Cocoa Mix      nan  10.2        positive
                            Coffee Frappe      nan   9.5        positive
```

Figure 6: SPARQL query results for Query 6: Identifying the top 10 highly-rated beverages with their preparation time and sugar content.

3 Ontology Diagram

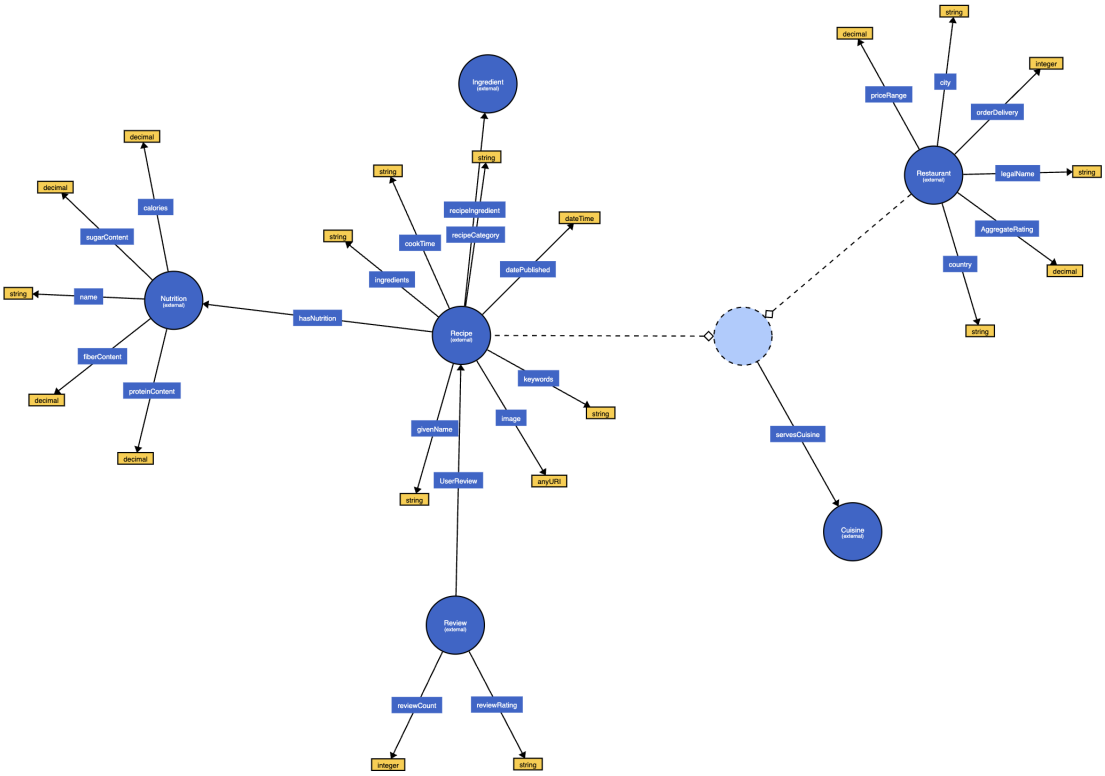

Figure 7: Ontology diagram representing the vocabulary of the Knowledge Graph.

4 Reflection

During the development phase, the team faced some challenges regarding environment compatibility and with wikidata linking, however to address these limitations and meet the project deadline, we adjusted our workflow:

1. **Switch to Google Colab:** Instead of fighting with local installations, we ran the extraction scripts in the cloud. This gave us a working environment immediately without needing to configure compilers.

2. **Using VaderSentiment:** We decided to replace the heavier machine learning models with `VaderSentiment` due to being a lighter library that does not require complex dependencies and easier to integrate while providing a good fast/accurate balance.

These adjustments allowed us to successfully build the Knowledge graph that was intended to build in order to provide answer to all of the questions.

5 References

RDFLib documentation: https://rdflib.readthedocs.io/en/stable/
spaCy: https://spacy.io/
Schema.org: https://schema.org/
Wikidata: https://www.wikidata.org/
DBpedia: https://www.dbpedia.org/
Vader Sentiment Analysis:
https://github.com/cjhutto/vaderSentiment
RDF Data Modelling Tutorial:

https://sinaahmadi.github.io/posts/rdf-data-modelling-a-tutorial.html
GraphDB: https://graphdb.ontotext.com/
RDFLib Endpoint: https://github.com/vemonet/rdflib-endpoint