Project Specification

Efficiently managing and accessing datasets

Bertozzi Jacopo
Martinengo Lucia
Spadazzi Roan
Sperandio Luca

Goals: read two datasets generated from the COVID-19 DisGeNET data collection, perform operations on them and share the final outputs with the user through a web-page application.

Deadline: 21/02/2022 **Completion Time:** 18/02/2022

Work Breakdown:

After reading the provided instructions, we focused our attention on finding correlations that could link the datasets. Our main candidates were the “pmid” values. Later, we designed the CRC cards based on the “prototype classes” written in the instructions [part 2] - examples below.

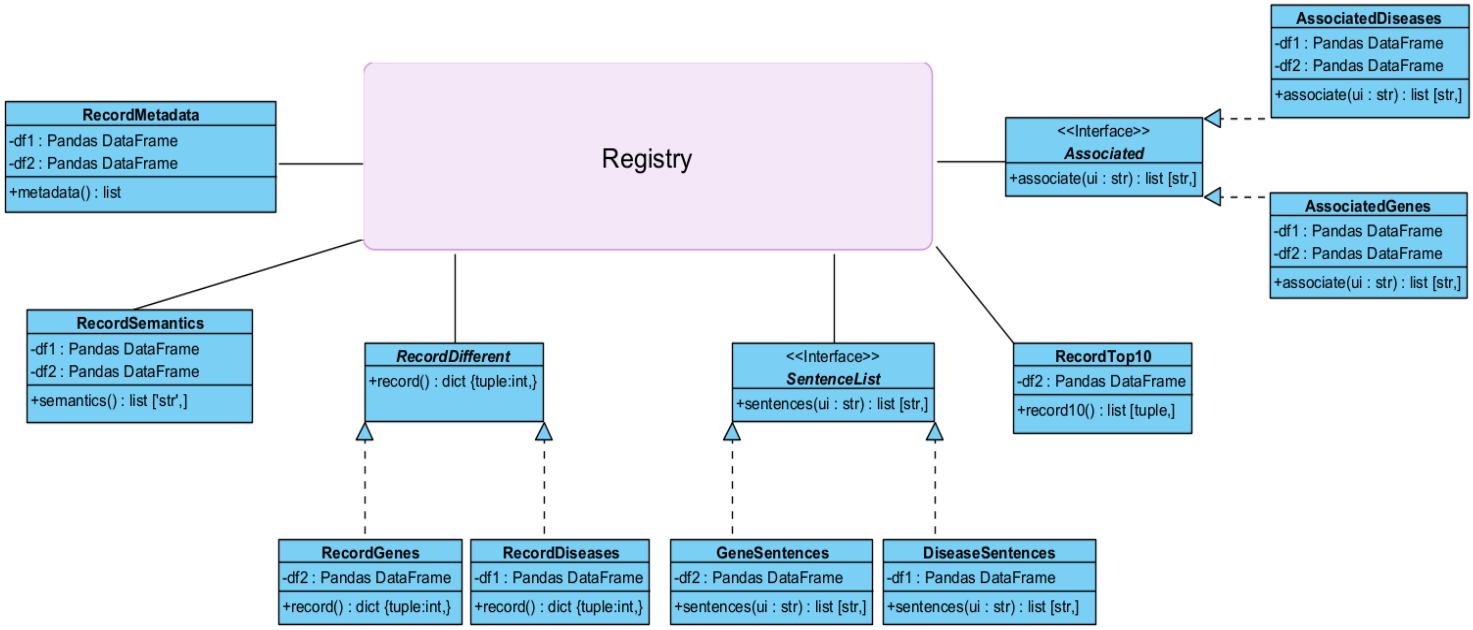
Abstract	**RecordDifferent**	DATA RecordGenes, RecordDiseases
		• gene_evidences • disease_evidences

An example of an abstract CRC card.

RecordMetadata	DATA
• Record the numerical metadata consisting of the number of rows and columns of each of the two files composing the data collection	• disease_evidences • gene_evidences

An example of a normal CRC card (notice the presence of “responsibilities”).

We proceeded by creating a general layout linking all the classes together, and we figured we could implement a couple of abstract classes as a template for similar methods. Using the UML-backbone, we then built the complete diagram by connecting abstract classes (<<interface>>) with their realizations and normal ones with associations. All the classes received their respective attributes and operations.



UML diagram.
Blue dashed lines represent realizations between an abstract class and its relative derivatives.
Straight lines represent a general association between the main Registry and the rest of the classes.

The next step regarded the implementation of [part 1] with the dataset readers using pandas and the Registry class dividing all operations specified in [part 2] with their relative classes. Finally, we coded [part 3] and its Flask application.

Functional specification:

Classes of [part 2]:

- **RecordMetadata**
  - Input: both datasets
  - Output: list of 4 integers
  - Functions: return the number of rows/cols of the DataFrames
  - Technique(s): usage of `.shape()`

- **RecordSemantics**
  - Input: both datasets
  - Output: a list of strings
  - Functions: return the name of the columns
  - Technique(s): usage of `.columns()`

- **RecordDifferent(ABC)**

- **RecordGenes(RecordDifferent)**
    - Input: gene_evidences.tsv
    - Output: dictionary {geneid: frequency}
    - Functions: return all genes with their relative frequencies
    - Technique(s): usage of `sorted()` with a lambda function to order the dictionary based on frequencies

- **RecordDiseases(RecordDifferent)**
    - Input: disease_evidences.tsv
    - Output: dictionary {diseaseid: frequency}
    - Functions: return all diseases with their relative frequencies
    - Technique(s): *same as RecordGenes*

- **SentenceList(ABC)**

- **GeneSentences(SentenceList)**
    - Input: gene_evidences.tsv
    - Output: list of strings
    - Functions: return all sentences associating COVID with genes
    - Technique(s): usage of: `.at[]` to pinpoint the exact location of an element in a DataFrame, the "<span class='disease covid cdisease'" string inside sentences and the re library to clean the string from HTML characters

    - Extra / about regexp: '`<[^<]+?>`' searches for a <…> string made of 1 or more characters `(+)` excluding another < `[^<]`. It searches the smallest possible string `(?)`

- **DiseaseSentences(SentenceList)**
    - Input: disease_evidences.tsv
    - Output: list of strings
    - Functions: return all sentences associating COVID with diseases
    - Technique(s) / extra: *same as GeneSentences*

- **RecordTop10**
    - Input: gene_evidences.tsv
    - Output: list of tuples(diseaseid, geneid)
    - Functions: return the top 10 distinct disease/gene associations
    - Technique(s): usage of: a single dataset to optimize memory usage and computing times; the "<span class='disease' id='" string; the separation of multiple diseases through "_" character; `sorted()` with a lambda function to order the dictionary based on frequencies

- **Associated(ABC)**

- **AssociatedDiseases(Associated)**
    - Input: both datasets
    - Output: list of strings
    - Functions: provide disease list given gene symbol/ID
    - Technique(s): usage of `.at[]` and the "pmid" values to switch from a dataset to another

- **AssociatedGenes(Associated)**
    - Input: both datasets
    - Output: list of strings
    - Functions: provide gene list given disease name/ID
    - Technique(s): *same as AssociatedDiseases*

[part 3]:

- requests classes in all parts involving user-input to correctly perform operations without importing [part 3] in [part 1], which would result in a looping-error.

Improvements & maintenance:

- Grouping of all functions of [part 1] in a unique Registry class that has as parameters the DataFrames, the performable operations and all the functions calling [part 2] methods.

- Handling user input errors by exploiting the length of the final output: if it is equal to 0 the function returns a 'Err404' string that will be later checked in [part 3]. Entering the `if res=='Err404'` brings the user to a specific error page.

- Using Regular Expressions as mentioned in "Technique(s)".

- Taking advantage of specific character sequences inside the sentences to isolate relevant outputs.

- Avoiding nested for loops.

Performance:

- After the previous improvements points 1 to 7 have high efficiency with very short computing times.

- Points 8 and 9 take a few seconds longer because of the double for cycles iterating on large DataFrames.

Aesthetic choices:

- `self.__links` in [part 1] allows to have short and "space-less" links for their relative webpages. Each one corresponds to an operation.

- Implementing clickable "Homepage" and "Go back" buttons to facilitate webpage navigation.

- Using ordered and unordered lists in the webpages based on the output format for a correct visualization.

Final comments:

- Our approach was to work on the same parts simultaneously to avoid reading and explaining each other's code, losing useful time and optimizing algorithm structure & creation.

- All the rest of the project's parts (such as PNGs, CRC Cards, further comments etc.) can be found in this same folder, in the code and the README.md file).