PROJECT SPECIFICATION

Zhang Weiwei and Di Silvestro Irene
22nd April 2022

AIM: given two datasets named disease_evidences.tsv and gene_evidences.tsv derived from DisGeNET COVID-19 data collection as input, write a program able to read the two files, compute operations on them exploiting classes, Object-Oriented programming and Python packages (Pandas, Flask and ABC) and present the user the relevant outcomes on a web application.

WORKFLOW

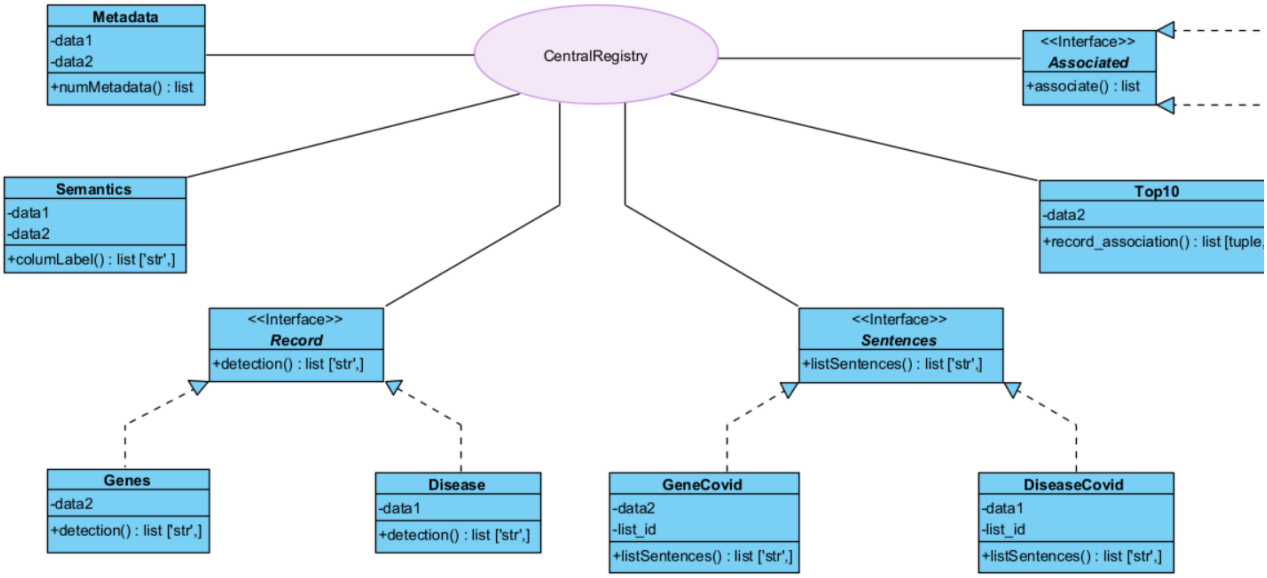
- After reading the given files and having understood the instructions to build our web application, first we focus on how to define our normal classes, one per each operation and then, when possible, we try to link some classes together by implementing a couple of abstract classes serving as template from which we can generate subclasses to perform similar operations in a faster and more efficient way. Then, we move to the design of the CRC cards basing on the classes we are going to implement in part 2.

  Below, there are the examples of an abstract and a normal class:

Abstract	**Record**	Data Genes, Disease
		• disease_evidences • gene_evidences

Genes	Data,Record
• Recording the number of different genes detected in literature. • The list should be sorted in ascending order	• gene_evidences

- Then, we build the complete UML diagram by connecting the abstract classes with their realizations (through blue dashed lines) and normal ones with their associations (through dark straight lines). All the classes are characterized by their respective attributes and methods.



- The next step consists of the implementation of three parts: part1.py with the two dataset readers using pandas and the registry listing the different types of operations specified in part2.py with their relative classes. While part3.py contains the codes to build the web-based user interface and its Flask application. Such a UI provides a list of choices, where each choice enables an analytical objective.

REGISTRY CLASS OF PART_1

- It defines the readers of the two datasets, the list of operations needed to be performed and the list of links of each web page, one per operation.
- If provides a function for each operation calling the result from the imported file of part2.py. This is necessary for the implementation of part3.py.

Grouping all functions of part1.py in a unique CentralRegistry class that takes as parameters the two DataFrames (named data1 and data2 in our program) facilitates the performance of the operations and the calling of functions in part2.py.

CLASSES OF PART_2

Class Metadata:	Input: the two datasets Output: list of 4 integers Functions: return the number of rows/columns of the two DataFrames Technique(s): built-in function .shape() of Pandas
Class Semantics:	Input: the two datasets Output: a list of strings Functions: return the labels of the columns of the two DataFrames Technique(s): built-in function .columns() of Pandas
Class Record(ABC):	**abstract**
Class Genes(Record):	Input: gene_evidences.tsv Output: a list of the geneid Functions: return distinct genes in an ascending order Technique(s): built-in function .groupby() of Pandas
Class Disease(Record):	Input: disease_evidences.tsv Output: a list of the diseaseid Functions: return distinct diseases in an ascending order Technique(s): *same as Genes*
Class Sentences(ABC):	**abstract**
Class GeneCovid(Sentences):	Input: gene_evidences.tsv Output: list of strings Functions: return all sentences associating COVID with genes Technique(s): usage of [‘column_label’] to extract the columns of data of interest and index them to look for evidence of covid exploiting the "<span class='disease covid cdisease'" and "<span class='disease covid cvirus'" strings inside sentences.
Class DiseaseCovid(Sentences):	Input: disease_evidences.tsv Output: list of strings Functions: return all sentences associating COVID with diseases Technique(s): *same as GeneSentences* without the "<span class='disease covid cvirus'" string.
Class Top10:	Input: the two datasets Actual output: list of tuples((geneid, diseaseid): frequency) Output: list of lists[[geneid], [diseaseid], [frequency]] Functions: return the top 10 distinct disease-gene associations Technique(s): exhaustive search exploiting the pmid values
Class Associated(ABC):	**abstract**
Class AssociatedDiseases(Associated):	Input: the two datasets Output: list of strings Functions: provide disease list given gene symbol/ID Technique(s): usage of .loc[] and the pmid values
Class AssociatedGenes(Associated):	Input: both datasets Output: list of strings Functions: provide gene list given disease name/ID Technique(s): *same as AssociatedDiseases*

WEB APPLICATION PART_3

Using Flask and importing the file of part1.py, we create a homepage with the list of provided operations and for each of them, bound to the corresponding class defined in part2.py, we build a web page using distinct HTML files to return the final result.

→ self.__links in part1.py allows to have short links for their relative webpages. Each one corresponds to their relative operation.

→ Implementing clickable “Homepage”, “Go back” and “Continue” buttons allows to facilitate webpage navigation.

EFFICIENCY

From operation 1 to 6, we can observe a high efficiency, as the computing time is very short taking only a few seconds. Points 8 and 9 take a few more seconds because of the double for loops iterating on large DataFrames. While point 7 takes around 10 hours to compute the output due to the exhaustive search approach performed on huge datasets. In this case, we have preferred the optimality of the solution at the expense of the time efficiency. Consequently, we saved the output and assigned it to a variable.

WORKING APPROACH

- logical division of the different parts of the program to build them one after another in order to facilitate their connection increasing the final working efficiency

- reasoning together to understand the general path to follow and then working on the same parts either simultaneously or separately

- all the parts of the project (such as PNGs, CRC Cards, further comments etc.) are found in the same folder

```
def Neurons(hours_of_programming):
        if hours_of_programming > 0:
                return ‘RIP’
        else:
                return ‘see ya next time’
print(Neurons(118))
```