Exercise 04 - Object-Oriented Programming

Deadline

November 23, 2020 - 12:00/noon (UTC+1:00)

Submission Guidelines

1. Clone your repository to your local workstation/computer if you have not done so
2. Structure and work on your submission.
   - Create a directory (folder) for the exercise sheet, e.g. name the directory ”exercise04” for Exercise 04.
   - Place all relevant files/scripts in this newly made directory
   - Do not include the `.git` folder! Add `.gitignore` to your git repo!
3. Commit your work to the git repository.
4. Create a git tag.
   - Tag name must be equivalent to ”SubmissionExerciseSheet04”.
   - Tag can be made via the command line or using the GitLab GUI

Grading (10 pts):

Task	1	2	3	4
Points	4	2	3	1

Using Previous Homework

This exercise requires you to build on the work done for Exercise03. If you did not do or finish Exercise03, or do not wish to use your work, you may use the provided NetworkX script in the sample_files folder.

General Style Tips

- Try to use "encapsulation" as often as possible -- use dunders ("_") to indicate which methods are private/protected and which aren't
  - Make methods private/protected if no one needs to access them directly
- Some methods may not need to use a class attribute at all and and execute code only on the parameters passed to the function itself. These are known as "static methods" and when these are defined within a class, you should use the `@staticmethod` decorator to label them as such
  - PyCharm tells you when to do this. Listen to PyCharm, it knows a lot!

Tasks

Task 1 *Give It Class* (4 pts)

Since most of the methods in your script pertain to generating a network and require using the same data (e.g. file paths, graph object, etc), it would be wise to wrap all of these methods up in a class so that the commonly used data can be defined as attributes and are easily accessible by the methods.

1. Create a new class called `Network` and move all of your methods related to importing/generating/exporting your graph into this new class. This will serve as an abstract class for what we will do next (1 pt)
   - Make sure your `Network` class has **at least** two attributes: `self.nodes` (representing your node dictionary) and `self.graph` (representing your edges/networkx graph)
     - These attributes should be empty for now upon initialization (i.e. set them to `None`)
2. Modify your `__init__` method to allow users to pass either a PPI file, or a node list and edge list. (1 pt)

- Modify your code so that if a user initializes this class with a PPI file, or a node list and edge list, it automatically compiles a node/ID dictionary and assigns it to the `self.nodes` attribute AND generates a NetworkX graph which is then assigned to the `self.graph` attribute
- Add some lines of defensive code which warns the user that there is a conflict if 3 files are uploaded (i.e. the PPI file and node+edge lists). This defensive code should raise a warning and tell the user what the problem is.

3. Create a new class called `Analyzer` that inherits from the `Network` class. Move all of your methods related to generating a graph image and finding the shortest path to this new class (1 pt)
   - Make sure to update your CLI methods to use these new classes!

> At this point you should have two classes: `Network` and `Analyzer`. Make sure your `Network` class has methods to do the following:
>
> 1. Import a new PPI file and overwrite the current `self.nodes` and `self.graph` attributes
> 2. Import a new node/edge list and overwrite the current `self.nodes` and `self.graph` attributes
> 3. Generate a node list and edge list and write them to specified locations
>
> ---
>
> Because your `Analyzer` class inherited from the `Network` class, it should be able to perform the above functions as well as do the following:
>
> - Generate a graph image
> - Find the shortest path between two given HGNC symbols

4. Add a new CLI command called `compile` that will generate a node/edge lists from a PPI file (0.5 pt)
   - This should take 3 parameters (excluding the name of the command): PPI file location, node list output location, and edge list output location
     - Example: `python my_script.py compile /path/to/ppis.csv ./nodes.tsv ./edges.tsv`
5. Update your methods so that the edges in the generated image can be labeled with the type of relation it is. This information can be extracted from the PPI file if it was not already. (0.5 pt)
   - This should be an optional network feature i.e. users can choose whether the edge types are displayed in the image or not

Task 2 *Natural Extension* (2 pts)

Up until now, the networks have only dealt with protein-protein interactions and therefore all nodes in the network are considered proteins. But as bioinformaticians, you know that proteins are translated from RNA, and RNA is transcribed from DNA. Therefore, it makes sense that the user should be able to enrich their network with this information if they choose to.

1. Create a new class method called `enrich_network` that automatically generates RNA and DNA nodes for every protein node in the network. (1 pt)
   - Nodes should be color coded so that they can easily be differentiated:
     - Red = proteins
     - Blue = RNA
     - Green = DNA
   - Edges should be generated between protein and RNA nodes and be labeled with `translated` and newly created edges between RNA and DNA nodes should be labeled with `transcribed`
   - Modify your `create` CLI command so that it has a boolean flag click option so users can enable this "enriched" output
   - Example: `python my_script.py create --enrich /path/to/ppis.csv graph.png`
   - *Note*: This may require you to change your methods related to finding and labeling shortest path to use different colors
2. Update your methods so that users can compile node/edge lists that include these new nodes and edges (1 pt)
   - The node list should now have a new column describing what type of molecule it is representing
   - The edge lists should the new `transcribed` and `translated` edges
   - Don't forget to update your new `compile` CLI command so you can enable this new output from the command line!
   - Example: `python my_script.py compile --enrich /path/to/ppis.csv ./nodes.tsv ./edges.tsv`

Task 3 *Summarize It For Me* (3 pts)

Users may want to know some quick facts about the network they are working with. Here you will modify your code so it can gather useful statistics about a compiled network

1. Create a new class called `Statistics` that inherits from the `Network` class (0.5 pts)
2. Add a method called `summary_statistics` to your `Statistics` class that can compile a pandas DataFrame of information about your network (1 pt)
    - It should contain the following information
        - Number of nodes in the network
            - Number of each type of node (protein, RNA, DNA)
        - Number of edges in the network
            - Number of each type of edge
        - The graph density
        - Average degree connectivity of the graph
3. Write a method called `export_stats` in the `Statistics` class that can export your summary statistics as either a JSON file or a separated file to a specified location/file (0.5 pts)
    - Users should be able to choose their delimiter (how it will be separated e.g. comma, tab, space, etc) if it's not a JSON file
4. Finally, add a CLI command called `stats` that will read a PPI or node/edge lists and generate summary statistics for the user (1 pt)
    - There should an option to export the statistics to a file
        - The type of file that it will be exported as (e.g. JSON, CSV, TSV, etc) should be determined by the file path's extension
            - e.g. If the export file path is `/home/tmp/stats.csv` then it should be exported as a CSV file with a "," delimiter
    - There should also be an option to print a table of the summary statistics directly to the terminal (STDOUT)
    - Users should be able to control if it is an "enriched" graph or not
    - Example test command: `python my_script.py stats /path/to/ppis.csv --print-table --enrich --export stats.json`

Task 4 *Modularize Your Methods* (1 pt)

Add this point, you have two separate parts of your code: code for handling the (images and stats) and the CLI related methods. It is time to break these up.

1. Separate your code into two files: all code relating to the CLI should go in a new file called `cli.py` and your network related code should be saved in a file called `network.py` (1 pt)
    - Make sure the CLI still imports the needed classes/methods properly and still works
    - I should now be able to run `python cli.py compile --enrich /path/to/ppis.csv ./nodes.tsv ./edges.tsv`
    - *Side Note* - If you wish to split your code/classes into additional modules, you may do so, but all CLI methods should reside in a file named `cli.py`