Introduction

In this assignment, we will explore a small dataset consisting of data about board games from *Board-GameGeek* (BGG). According to Wikipedia, BGG is "an online forum for board gaming hobbyists and a game database that holds reviews, images, and videos for over 125,600 different tabletop games, including European-style board games, wargames, and card games.". In this assignment, the focus is on "game database" aspect of BGG. More specifically, the students will perform a small investigation of the different mechanics of the many games in the database. That investigation will be assisted by existing libraries for data science in Python. It will also require you to interact with the public APIs of Gemini since we want to investigate how much one of its large language models "knows" about these board game mechanics. To conduct this assignment, it will be necessary to obtain a key to access the Gemini APIs.

In addition to implementing the functionality described above, in this assignment, the students will perform quality assurance activities. More specifically, to write automated tests for some parts of the system and to use a linter to identify simple problems related to code quality and functionality. Furthermore, the students will analyze the solution they have designed and implemented to identify code smells and instances of code that do not adhere to the principles studies in the lectures. For the cases where you do identify these problems, you are expected to eliminate them by refactoring.

This assignment has three main goals: (i) to reinforce your programming skills; (ii) to practice testing, refactoring, and code review with support from a linter; and (iii) to make you explore library and API documentation, to gain the skill of learning unfamiliar technology independently. Before starting work on this assignment, make sure you have read the entire specification. The deliverables for this assignment are the code you will produce, as well as a report.

Part 1: Obtaining the libraries (10 points)

In this project, you will need to leverage external Python libraries and APIs. More specifically, you will need to leverage Gemini's library, which is used to programmatically send requests to and receive responses from Gemini. You can install that library using `conda` or `pip`. Furthermore, to process the provided dataset in CSV format, using a library such as `Pandas` or `NumPy` can save you a lot of time. The former is recommended, but the latter also gets the job done. You can also do all the data processing manually, but that is likely to require more effort from you. The Python SDK for the Gemini API is contained in the google-generativeai package. Install the dependency using pip:

```
pip install -q -U google-generativeai
```

Make sure that you have `google-generativeai` version 0.8.3 or later. You can check that in conda with either

```
conda list google-generativeai
```

or

```
pip show google-generativeai
```

To be able to use Gemini, you will need to acquire an API key. The API key is generated by Gemini and enables it to recognize the programs you build as valid API users. Getting one is very easy once you have registered with Gemini. Just go to https://aistudio.google.com/app/apikey and then press "Create API key".

There are many online tutorials about how to access Gemini's API. To get your first contact with Gemini, we recommend that you use this one. In this project, we are particularly interested in its text completion capabilities (as a way to answer questions). Be sure to experiment with Gemini programmatically, e.g., by writing a program that reads input from the keyboard, sends it to Gemini, and prints the completion it provides. This will be necessary for the next part. Bear in mind that the usage of Gemini's models is free up to a certain limit, which should be more than enough for this project. This is the model you should be using.

In your report, you should mention which libraries and/or frameworks you have used. If none, that should be stated explicitly. You should also mention problems you faced while working on this part, e.g., incorrect Gemini's version, Gemini unavailable, incorrect documentation, problems generating or using your API key, etc.

Part 2: System Development (40 pts)

This part comprises multiple smaller Python implementation tasks, to help you organize your work. For all these tasks, the program should still work even if the dataset changes in terms of having some board games add to or removed from it. In other words, it should be general enough to work with whatever games exist in the dataset, as long as its structure is preserved. You are free to create additional methods, classes, and fields as you deem necessary. You must make your implementation *object-oriented.*

Task 0. Create a Github repository for your project. The repository name must be INFOB2SOM-2024-GroupID. For example, the repository name for G 100 must be INFOB2SOM-2024-G100. Add all the members of the group, TAs and instructors as collaborators in this project. The Github usernames of the TAs and instructors will be shared in a separate announcement. Make this repository private. The team is expected to conduct their work using this repo to store the artefacts (code, documents) they produce for this assignment and update it regularly.

Task 1. Create a class named `BoardGameMechanicsAnalyzer`. The constructor to this class should take two parameters (besides self, of course): the path of the dataset to be analyzed and your Gemini API key. The constructor should read the dataset from disk and store it in main memory, using the Pandas framework. It should also eliminate all the rows where the name, year of publication, or list of mechanics of the game is missing (null or nan). In the report, briefly discuss how you have chosen to represent the dataset internally to `BoardGameMechanicsAnalyzer` and how you have eliminated the rows with missing data.

Task 2. Add functionality to your class to ask Gemini whether the mechanics listed for a certain game in the BGG dataset are actually mechanics that are associated with that game. This should work for any game in the dataset. For example, when asked about which of the mechanics listed in the dataset for the game Gloomhaven ('Action Queue', ' Action Retrieval', ' Campaign / Battle Card Driven', and 16 others), Gemini stated that only ten of them actually apply to the game. Your program should both send the query to Gemini and collect its response. Bear in mind that the prompt you send to Gemini strongly impacts the answer it produces. Experiment with different possibilities before committing to a specific one.

Task 3. Add functionality to your class to process the result produced by Gemini in response to the query. It should identify which mechanics Gemini considers that the game does have and contrast that against the set of mechanics listed on BGG. The data at BGG can be considered ground truth. Therefore, the ratio between the number of correctly predicted mechanics listed by Gemini and the total number of predictions indicates the Accuracy of Gemini in determining the mechanics for that specific game.
`Accuracy =  (# Correct predictions by Gemini)/(# Total predictions)`

Task 4. Make it possible for users of instances of BoardGameMechanicsAnalyzer to determine Gemini's accuracy in listing the mechanics of multiple board games. It should be possible to provide a list of (game name, year published) pairs. It should also be possible to inquire about all the games in the dataset. Bear in mind that Gemini limits the number of requests that can be made per minute. For example, for model="gemini-1.5-flash" which has a fast and versatile performance across a diverse variety of tasks, the rate limit is 1 million tokens per minute, 15 requests per minute, and 1500 requests per day. You can view the exact rate limits per model on this page. In your report, indicate Gemini's mean accuracy for the 200 highest-rated games in the dataset.

Task 5. Finally, determine the ten mechanics that Gemini most consistently and accurately assigns to games. Do the same for the ones that are being assigned least consistently. Are there mechanics on BGG that Gemini never attributes to any games? Add these pieces of information to your report. You can utilize Python's collections module for this purpose. Besides providing a working implementation, in the report, you should document the organization of your code in terms of how responsibilities have been assigned to program elements (classes, methods, etc.).

Part 3: System Verification and Quality Assurance (25 Pts)

To ensure your system functions as intended, perform the following tasks:

1. Test Design and Automation
   - Design at least three test cases for your system. Each test case should include:
     - The user role.
     - Preconditions.
     - Test setup.
     - Step-by-step instructions to execute the test.
     - The expected outcome for successful execution.
     - Automate these tests using the unittest framework and execute them.
     - Document the results:
     - Indicate whether each test passed or failed.
     - If a test fails, explain the issue and how it was resolved.
     - Failure Scenario: At least one test case must simulate a failure scenario (e.g., Gemini API is unreachable, or the dataset is unavailable).
2. Non-Deterministic Behavior and Mocking
   - Acknowledge that systems using Gemini may exhibit non-deterministic behavior (e.g., varying responses to identical prompts).
   - Create at least one test case that uses unittest.mock to produce deterministic results.
   - Mock functionality dependent on Gemini to facilitate reliable testing.
   - Refer to the unittest.mock documentation to implement this.
3. Test Coverage Analysis
   - Use Coverage.py to measure test coverage and specify the coverage criteria used.
   - Include a detailed description of your test cases and test coverage in your report.
4. Linting with myPy and Pylint
   - Analyze your system using the myPy and Pylint linting tools.
   - Document in your report:
     - The issues identified by each tool.
     - Their significance and implications.
     - How you addressed (or chose not to address) these issues.
     - If more than 10 issues are identified, focus on the 10 most important ones and provide a summary of the remaining issues.

Part 4: Code Review and Refactoring (25 Points)

In this part, you will review the design and implementation of your system to identify and address code smells.

1. Identify Code Smells: Analyze your system to detect code smells. Specify which smells were found, the affected program elements, and explain why these elements qualify as smells. Remember, code smells are contextual and not absolute; their significance depends on the specific circumstances of the design.
2. Refactor the System: Refactor your code to address the identified smells and improve the system's structure. Document the refactorings applied, including the rationale behind each change and the specific code smells it addresses.
3. Report Your Process: In your report, provide a detailed account of the process you followed:
   - Were the refactorings applied manually or with the assistance of a tool?
   - How did you ensure that the refactorings preserved the system's behavior?

This part of the assignment evaluates your ability to critically assess your codebase, apply best practices in refactoring, and communicate your process effectively.

Work Decomposition

Each member of a group must contribute to the project in a fair way. Indicate an effort distribution statement listing the contributions of each member in your report.

Policy on the Use of Generative AI

1. Permitted Uses: Students may use generative AI tools (e.g., ChatGPT, GitHub Copilot) under the following conditions:
   - Learning and Exploration: For understanding concepts, debugging, and generating small code snippets to aid in problem-solving.
   - Documentation and Comments: To assist in drafting documentation, comments, or explanatory notes, provided they are critically reviewed and adjusted by the student.
   - Error Identification: To identify and explain potential errors in their code.
2. Prohibited Uses: The following uses of generative AI are not allowed:
   - Complete Code Generation: Submitting AI-generated code as is, without significant modification or understanding.
   - Code Ownership Misrepresentation: Passing off AI-generated solutions as entirely original work without acknowledging AI assistance.
   - Avoiding Concept Mastery: Relying on AI to bypass core learning objectives, such as understanding algorithms, debugging techniques, or design principles.
3. Acknowledgment of AI Assistance: Students must document their use of AI in the assignment, specifying:
   - The tools used (e.g., ChatGPT, GitHub Copilot).
   - The specific tasks where AI was used (e.g., generating code snippets, clarifying concepts).
   - Any substantial modifications or refinements made to AI-generated output.

This acknowledgment must be included as a section in the report. The groups who do not include this statement needs to explain the use of AI and their understanding of the project in a set meeting.

4. Impact
   - Understanding: Students must demonstrate a clear understanding of all code and concepts in the project, regardless of AI assistance.
   - Plagiarism: Directly submitting AI-generated code or solutions without attribution will be treated as plagiarism and subject to academic integrity policies.
5. Encouragement of Responsible Use: Students are encouraged to:
   - Use AI as a learning partner rather than a shortcut to solutions.
   - Critically evaluate AI-generated outputs for correctness, efficiency, and alignment with the requirements.
   - Seek guidance from instructors or TAs when unsure about the appropriate use of AI in their projects.
6. Limitations of AI: Students should be aware that AI tools:
   - May produce incorrect, incomplete, or inefficient solutions.
   - Do not replace the need for a thorough understanding of fundamental programming concepts.
   - Can introduce biases or errors that need careful scrutiny and correction.

Deadline

The deadline for the assignment is December 8, 2024 at 23:59.

Late Policy

Each group has a total of 3 bonus days that they can use throughout the course. Inform the instructor Negar Alizadeh (`n.s.alizadeh@uu.nl`) how many bonus days you plan to use if you decide to use them before December 6, 2024 at 16:59 (before the end of the work week) for this assignment.

Apart from the bonus days (either due to lack of communication or spending all bonus days), the late policy is -10 points for each late day.

What to submit?

Submit a .zip file, containing:

1. a PDF report describing your work names as GroupID-LabAssignment1.pdf. For example, the report of the group G 100 would be named as G100-LabAssignment1.pdf. For parts 2–4, %20 of your grade is your report. The report should list the group id, member names and their student IDs in the title page. The report can be 3-5 pages long excluding the title page.
2. Your implementation of the entire Board Game Mechanics Analyzer application. Submission of solutions should be made through Blackboard.