Below are three programming tasks. Complete the tasks that fit the GRA you are applying for. You can complete multiple tasks.

**Generative AI:** We are not against using generative AI models if they make our code better and increase our efficiency. If you decide to use any of the available generative AI models for code, inform us which model you use, make sure to highlight what is your code and what is the model’s code, be able to explain all code including that from GTP.

**Deadline:** January 22nd, 5pm

Good Luck!

**Task 1 – Generation Data Science and ML**

1. Write a program in Python to scrape about 100 web pages from a single health-related website. For each page calculate at least 5 NLP-related statistics and store all the results in a file.
2. Create one more file named aggregated\_results (1 record) which is the average of the 100 statistics.
3. Create a simple local web application in Java using a web framework like Spring/Play containing one or more web pages. It should take input of a text file or text in the request and then generate nlp statistics of the text similar to the earlier task.
4. Return the comparison of the statistics between the input text and statistics from the earlier task(aggregated\_results) in a txt file or on another web page ( in a table format)

Some examples of statistics are: 1) the number of sentences, 2) the number of words, 3) the number of nouns (anything with a part-of-speech that starts with NN)

Write your code assuming that it will be used as part of a larger project and that someone else may also be calling your methods/classes.

To submit:

* create a public github repo.
* Make sure to include any external jars (or a MAVEN file) in your repo
* Include a README in markup that outlines what you did.
* Follow the best practices of code development and reproducibility. Code should be reproducible on any system
* If possible record a demo and add that in the GitHub repo as well.

The goal of this is to get a feeling for your experience and comfort level with some basic programming tools, so feel free to improvise as you see fit.  
This task checks your Python, web scraping, Java, and web development skills.

**Task 2 – Mainly ML**

Use the included dataset - “Problem\_Dataset.xlsx” to build an ensemble of multiple NLP models. The goal is to learn to label a sentence with A1, A2, etc.

Subtask 1: There should be at least 1 model for each binary classification label (AKA diagnostic criteria A1, A2, A3, B1, B2, B3, B4). All models are expected to be deep learning based models. Use a train:validation split as 80:20 for each of the following variants.

These are the deliverables:

* 1. training script or methods (You may choose to write as classes or simple functions) to clean the data, build train & save models.
* 2. An evaluation script or a clean python method to load & run all 7 models (from local directory) as ensemble and give predictions for each input text as an array of length 7 (where each index holds a binary value for corresponding label)
* Obtain precision, recall, F1-score along with predictions and save them locally.

Subtask 2: Now modify the above code in training and prediction methods to build & train a multi-label classification model. Report precision, recall, F1-score along with predictions and save them locally.

Subtask 3: Now, make use of the BERT model with required customization. Fine-tune the new model to report precision, recall, F1-score along with predictions and save them locally. Report precision, recall, F1-score along with predictions and save them locally.

Subtask 4:

Plot the Label-wise comparison (all 7 labels) for precision, recall and F1-scores from all 3 scenarios(Ensembler Vs. Muliti-label model vs. BERT model).

Make assumptions wherever required and explain the choice/particular decision or assumption (if any).

Write clean python code for each of the above tasks in a single jupyter notebook (.ipynb file) with required headings and brief explanations. Please do extensive commenting for code readability. Also DO SHARE the list of external/additional python modules and dependencies used in the notebook as requirements.txt file since the notebook shall be reproducible in Google colab on our end.

**Task 3 – Frontend/full stack**

**Frontend Development:**

1. Create a user-friendly HTML form with a text area where users can enter text to be processed.
2. Apply basic CSS styling to the form. Consider using Bootstrap for improved visual appeal.
3. Use JavaScript to validate user input on the front end, ensuring it's not empty and contains at least one sentence. You can utilize jQuery for DOM manipulation.

**Backend Development:**

1. Set up a Flask web application with routes for serving the HTML form and handling form submissions.
2. Implement server-side validation for the user's input text to ensure data integrity.
3. To do the text processing (sentence splitting, tokenization, part-of-speech tagging) use the Stanford CoreNLP library (<https://stanfordnlp.github.io/CoreNLP/>). Write your code assuming that it will be used as part of a larger project and that someone else may also be calling your methods/classes.
4. Modify your Flask application to use the NLP model to process user input text.
5. Create API endpoints for sending and receiving data between the front end and back end.

**AWS Deployment and Hosting:**

1. Deploy your Flask application on an AWS EC2 instance. Configure security groups and open necessary ports for access.
2. Connect the front end and back end. Modify the frontend to make POST requests to the Flask backend running on the AWS EC2 instance.

**Result Display:**

1. Display NLP model results on the front end. You can present the results as text or use visualizations for more advanced displays.

**GitHub Repository and Documentation:**

1. Create a public GitHub repository to host your project code. Include all necessary files, such as HTML, CSS, JavaScript, Flask app code, and any model files or data.
2. Write a README.md file in your repository. The README should explain how to run the project, describe its functionality, and provide any other relevant information.

Where possible, include screenshots in the README to show working code.