DGL 2025 Coursework 1

Introduction

Welcome to the first DGL coursework! In this assignment, you will explore how graph neural networks (GNNs) can be used for different tasks, including **graph classification** and **node classification**. GNNs are powerful models that allow us to work with graph-structured data, making them useful in various real-world applications such as social networks, biological data analysis, and recommendation systems. In this coursework, you will tackle three key challenges:

1. **Graph-Based Classification** You will implement a GNN that can classify entire graphs while dealing with an interesting dataset. You will explore different graph-level aggregation techniques, analyze class distributions, and experiment with ways to improve model performance.
2. **Node-Based Classification in a Heterogeneous Graph:** Here, you will work with a graph that has **multiple types of nodes**, each with different feature dimensions. Your task is to design a GNN that can correctly classify these nodes while considering their heterogeneity.
3. **Investigating Topology in Node-Based Classification:** In this section, you will analyze how the structure (or topology) of a graph affects the performance of GNNs. You will explore key graph measures such as **node degree**, **modularity**, and **Ollivier-Ricci curvature** and examine their impact on model performance.

What You Will Learn

By the end of this coursework, you will:

- Understand how GNNs can be used for graph and node classification.
- Implement and experiment with different aggregation methods.
- Analyze the impact of class and dataset structure on GNN performance.
- Work with heterogeneous graphs where nodes have different types and feature dimensions.
- Investigate the role of graph topology in node classification tasks.

Submission and Evaluation

The grading will be based on:

- Correctness and completeness of your implementations.
- Performance improvements achieved through your optimizations.

- Clarity of explanations and analysis in your report.
- Adherence to the provided submission guidelines.

IMPORTANT NOTE: The notebooks are strictly for code implementation. All explanations, discussions, plots, and figures must be included in your report.

IMPORTANT NOTE: Use concise answers where possible.

This coursework is designed to be hands-on and exploratory, so feel free to experiment, analyze, and optimize your models. Most importantly, have fun learning about GNNs!

1 Graph Classification (20 points)

In this question, we will explore graph classification using the provided dataset. Specifically, we will implement graph-level classification using graph convolutional networks (GCNs), analyze the dataset, and experiment with ways to improve the model and training process to achieve higher classification performance.

1.1 Graph-Level Aggregation and Training (5 points)

1.1.a Graph-Level GCN (2 points)

Implement three graph-level aggregation methods: **sum**, **mean**, and **max**. A GCN implementation is provided, and your task is to adapt it into a graph-level GCN by integrating different aggregation functions.

1.1.b Graph-Level Training (1 point)

Make modifications to the training script to:

1. Train the model and record the training loss and validation accuracy for each epoch.
2. Plot the test f1 for all three aggregation methods.

   Use the provided functions:

   - `train_model`
   - `plot_training_and_validation`

1.1.c Training vs. Evaluation F1 (2 point)

Additionally, compare training F1 scores vs. evaluation F1 scores. What differences do you observe? Which aggregation function performs best and why (**sum, mean, or max**)?

1.2 Analyzing the Dataset (5 points)

You may notice that the model performs worse on the evaluation dataset. The goal of this task is to analyze the dataset and identify potential issues that might affect the model's performance.

1.2.a Plotting (3 points)

- Plot the topologies of the graphs.
- Plot the feature distributions.
- Plot the label distributions.

1.2.b Discussion (2 points)

- Analyze the generated plots.
- What are your observations?

1.3 Overcoming Dataset Challenges (20 points)

In this section, you will attempt to address the challenges identified by improving the model or training process.

1.3.a Adapting the GCN (5 points)

Modify your GCN implementation from Q1.1 to accept the number of layers and output dimension (i.e. the *graph embedding dimension*) as parametes. Experiment with different hyperparameters, such as:

- Number of layers.
- Hidden dimension size.

You will implement:

- Implement GCN as described above.
- Experiment with hyperparameters and report the results and plots.

Hint: The `ModuleList` class might be useful.

1.3.b Improving the Model (5 points)

Identify and implement **three different methods** to overcome the challenges discovered in Q1.2. Your goal is to achieve the highest possible score. You may experiment with:

- Model architecture modifications.
- Data preprocessing techniques.
- Hyperparameter tuning.
- Loss function adjustments.

1.3.c Evaluating the Best Model (5 points)

Plot the performance of your best model over **100 epochs or more**, averaged over multiple random (i.i.d.) runs (**at least 10 runs**) to produce a smoothed training and evaluation accuracy curve. **Note:** Your score will be based on the smoothed curve.

Hints:

- Set `verbose=False` in the `train_model` function to suppress excessive logging.
- Use `np.mean` with an axis parameter to compute the average performance over multiple runs.

1.3.d Final Analysis and Explanation (5 points)

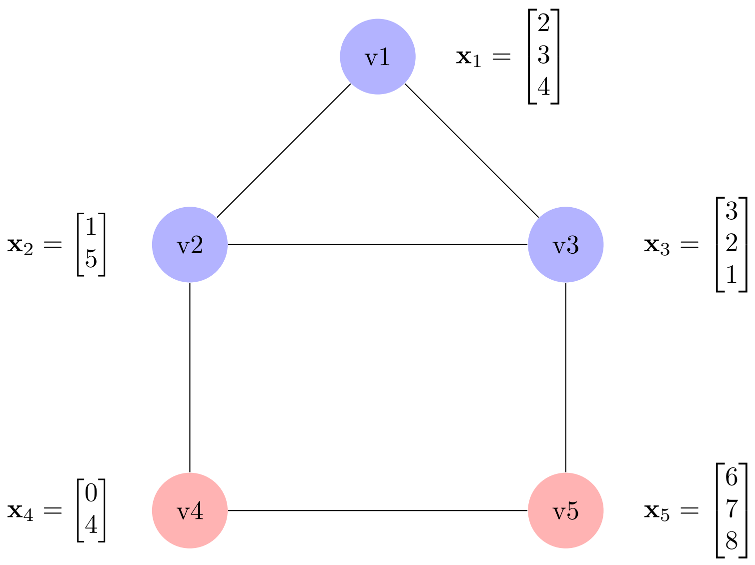
Provide a detailed explanation of your process and findings. Specifically, address the following:

- What modifications did you make to the model, loss function, and dataset?
- What hyperparameter tuning strategies did you explore? What were your findings?
- Present your findings with plots and results. Attempt to explain observed patterns in model performance.

Tip: Clear, concise sentences and bullet points are sufficient for this section.

2 Node Classification in a Heterogeneous Graph (40 points)

Consider a **heterogeneous graph** where there are 2 types of nodes, t_1 and t_2 with d_1 and d_2 feature dimensions, respectively. Each node belongs to one of two classes: c_1 and c_2. The task is to train a GNN for node classification on a heterogeneous graph. Below is an illustration of a graph with nodes colored based on classes. The graph is undirected.


v1
$\mathbf{x}_1 = \begin{bmatrix}2\\3\\4\end{bmatrix}$
$\mathbf{x}_2 = \begin{bmatrix}1\\5\end{bmatrix}$
v2
v3
$\mathbf{x}_3 = \begin{bmatrix}3\\2\\1\end{bmatrix}$
$\mathbf{x}_4 = \begin{bmatrix}0\\4\end{bmatrix}$
v4
v5
$\mathbf{x}_5 = \begin{bmatrix}6\\7\\8\end{bmatrix}$

In this graph:

- Nodes v_1, v_3, and v_5 are of t_1 and nodes v_2 and v_4 are of t_2.
- Nodes v_1, v_2, and v_3 have the class label c_1 and nodes v_4 and v_5 have the class label c_2.
- The feature vector for each node is shown next to it. For example, node v_1 has the feature vector $\mathbf{x}_1 = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$.

2.1 Dataset

In this question, we use a **synthetic dataset** which has already been generated and split into train and validation subsets.

- The training graph has $N_1^{tr} = 250$ nodes for node type t_1 and $N_2^{tr} = 250$ nodes for node type t_2.
- The validation graph has $N_1^{val} = 100$ nodes for node type t_1 and $N_2^{val} = 100$ nodes for node type t_2.
- t_1 and t_2 nodes have $d_1 = 20$ and $d_2 = 30$ features, respectively.

The `load_data_from_json` function has already been implemented for you, which reads, loads, and parses the JSON file and returns:

- $\mathbf{A} \in \mathbb{R}^{N \times N}$: Adjacency matrix where $N = N_1 + N_2$.
- $\mathbf{X_1} \in \mathbb{R}^{N_1 \times d_1}$: Node feature matrix for t_1 nodes.
- $\mathbf{X_2} \in \mathbb{R}^{N_2 \times d_2}$: Node feature matrix for t_2 nodes.
- `mapping`: A Python dictionary object such that:

```
mapping[i] = {
   "node_type": str ("type1" or "type2"),
   "feat_index": int index into the relevant feature matrix
}
```

2.1.a Problem Challenge (1 point)

Explain why this is a challenging problem and how it differs from a standard node classification task.

2.1.b Real-World Analogy (1 point)

- Provide a real-world example where this problem may arise.
- Explain what each node type and class label could represent.
- Ensure the analogy aligns with the dataset structure.
- Highlight why understanding the relationships between node types and classes is critical for this task.

2.1.c Interpretation of the Dataset: Plotting the Graph (2 points)

- Implement the `plot_graph` function which visualizes the graph structure.
- Color nodes by class labels and display their types.
- What patterns do you observe in terms of node distributions across types and classes?

2.1.d Interpretation of the Dataset: Plotting the Node Feature Distributions (2 points)

Implement the `plot_node_feature_distributions` function.

- This function should plot the distribution of the node features.
- Do not consider the distribution of a specific feature x_i, consider the mean of the feature vector $\mathbf{x}$ for each node.
- All distributions should appear on the **same figure**, so you can compare the feature distributions of t_1 and t_2 nodes as well as c_1 and c_2 nodes in Question 2.1.e.
- Check Figure 1 for reference on how your plot should look like.

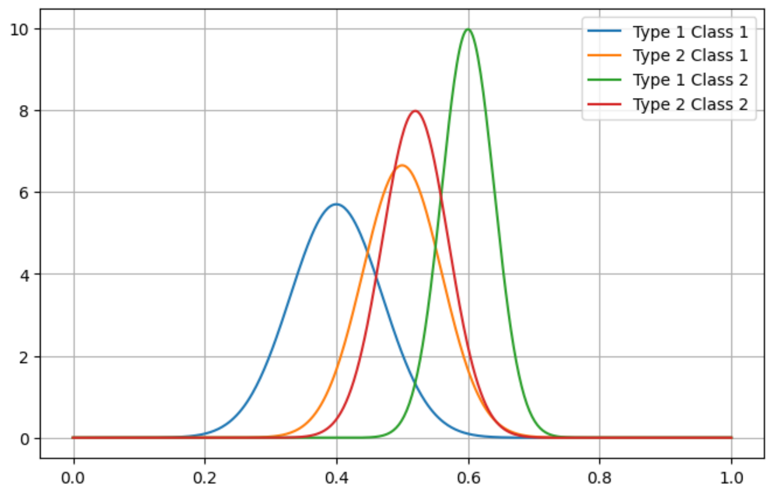


Figure 1: Example figure illustrating the distribution of feature values for different node types (t_1, t_2) and classes (c_1, c_2).

2.1.e Interpretation of the Dataset: Discussion (2 points)

- Interpret the node feature distributions.
- Discuss any patterns, similarities, or differences you notice.
- What observations can you make regarding the distribution of features across different node types (1 point) and classes (1 point)?

2.2 Naive Solution: Padding

In the naive solution, we address the challenge of heterogeneous node feature dimensions by zero-padding all node feature vectors to match the maximum feature dimension. This ensures uniformity in the input dimensions for processing with a GNN.

Naive Solution: Let the feature vector of a node v_i be denoted as $\mathbf{x}_i$, with a dimension d_i. The maximum feature dimension across all nodes in the graph is defined as:

$$d_{\text{max}} = \max\{d_i \mid i \in \text{nodes}\}.$$

To create uniform dimensions, each node's feature vector is zero-padded as follows:

$$\mathbf{x}_i^{\text{padded}} = \begin{bmatrix} \mathbf{x}_i \\ \mathbf{0} \end{bmatrix},$$

where:

- $\mathbf{x}_i \in \mathbb{R}^{d_i}$ is the original feature vector of node i.
- $\mathbf{0} \in \mathbb{R}^{d_{\text{max}} - d_i}$ is a zero vector of length $d_{\text{max}} - d_i$, added to pad the feature vector to a common dimension d_{max}.

In our heteregonous graph, there are two types of nodes, t_1 with feature dimension $d_1 = 20$ and t_2 with feature dimension $d_2 = 30$. The maximum feature dimension is:

$$d_{\text{max}} = \max\{d_1, d_2\} = 30.$$

For a node v_i of type t_2, with features:

$$\mathbf{x}_i = \begin{bmatrix} 5 \\ 7 \\ \vdots \\ 2 \end{bmatrix} \in \mathbb{R}^{20},$$

the zero-padded feature vector becomes:

$$\mathbf{x}_i^{\text{padded}} = \begin{bmatrix} 5 \\ 7 \\ \vdots \\ 2 \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{30}.$$

2.2.a Limitations of Naive Solution (2 points)

- List two **two key limitations** of the naive solution.
- Consider aspects such as model expressiveness, computational efficiency, or any other relevant factors. Provide clear justifications for your points.

2.3 Node-Type Aware GCN (18 points)

2.3.a Implementation (10 points)

In this task, **design and implement** a node-type aware 2-layer GCN model that explicitly accounts for two distinct node types and performs binary node classification. Your design should adhere to the following requirements:

- Consider two types of nodes without using padding **(8 points)**.
- Limit your model to **two layers**, as in the naive solution.
- You may either use the provided `GCNLayer` and `GraphNeuralNetwork` classes or implement your own classes. However, the final model architecture must be implemented within the `HeteroGCN` class.
- While you are allowed to introduce additional learnable parameters, your approach must **explicitly** account for different node types rather than merely increasing the number of learnable parameters.
- Determine the optimal hyperparameters for your model, such as hidden dimensions, learning rate, scheduler patience, etc. These hyperparameters do not need to match those of the naive model, as each design has its own optimal configuration.
- Your model must achieve superior **validation F1 score (2 points)** compared to the naive approach.
- You may use different versions of adjacency matrices in your implementation:
  - Unmodified adjacency matrices, i.e., `A_train`.
  - Self-looped adjacency matrices, i.e., `A_train_self_loops`.
  - Normalized adjacency matrices, i.e., `A_train_normalized`.

2.3.b Discussion (8 points)

In your report, you are expected to:

- Explain your design choices, including:
  - The reasoning and logic behind your solution. (1 point)
  - The limitations of the naive solution that your model addresses. (1 point)
  - The advantages and potential drawbacks of your approach. (2 points)
- Include the loss curves for both the naive approach and your model. These should be automatically generated using the `plot_loss` function provided to you. (2 points)
- Describe your hyperparameter tuning process, including the methodology and reasoning behind your choices. (2 points)

2.4 Exploring Attention (8 points)

2.4.a Implementation (4 points)

In this question, you will explore the attention-based aggregation mechanism in GCNs. Attention-based aggregation is explained in Lecture 3.5.

- Implement an attention-based aggregation mechanism for binary node classification using GCNs. (2 points)
- Limit your model to **two layers**, as in the Question 2.2 and Question 2.3
- Base your implementation on the naive padding approach introduced in Question 2.2.
- **Do not** implement a node-type aware special design as in Question 2.3.
- Your implementation must achieve better **validation F1 score (2 points)** compared to the naive GCN in Question 2.2.
- Determine the optimal hyperparameters for your model, such as hidden dimensions, learning rate, scheduler patience, etc. These hyperparameters do not need to match those of the previous models, as each design has its own optimal configuration.

2.4.b Discussion (4 points)

Include in your report the following:

- Explain the attention-based aggregation shortly. (1 point)
- Loss curve for your model, which should be automatically generated using the provided `plot_loss` function. (1 point)
- Description of your hyperparameter tuning process, including the methodology and reasoning behind your choices. (2 points)

Implementation Notes

- A baseline 2-layer Graph Convolutional Network (GCN) has already been implemented for you in the Jupyter Notebook as a naive solution.
- The functions `train_model` and `evaluate_model` are provided for you. **Do not modify their definitions**, as doing so will result in a penalty. These functions are essential for training and evaluating models throughout this question.
- The `train_model` and `evaluate_model` functions accept any number of keyword arguments, which will be passed to your model's `forward` method. Please refer to their docstrings and observe their usage in the naive approach for proper implementation.

2.5 Overall Discussion (4 points)

Your report must include the following:

- A bar plot comparing the **precision, recall, and F1 scores** of the three models, generated using the `plot_results` function. Explain your results. (2 points)

- An explanation of why the Node-Type Aware GCN outperforms the naive GCN. (1 point)
- An explanation of why the attention-based aggregation GNN outperforms the naive GCN. If your implementation could not outperform Naive GCN, do not worry, you can get full points in this question by clearly explaining the reason. (1 point)

3 Investigating Topology in Node-Based Classification Using GNNs (30 points)

In this section, we will explore the impact of graph topology on node-based classification using GNNs. The experiments will focus on analyzing different topological measures, visualizing their distributions, and evaluating GCN performance on 2 graphs with different topologies.

3.1 Analyzing the Graphs (10 points)

3.1.a Topological and Geometric Measures (3 points)

- Examine two topological measures (*Node Degree* and *Betweenness Centrality*) and one geometric measure (*Ollivier-Ricci Curvature*).
- Include a definition for each measure.
- Explain what each measure quantifies and how it contributes to understanding the structure of a graph.

3.1.b Visualizing and Comparing Topological and Geometric Measures of Two Graphs (3 points)

Implement the following three functions to visualize and compare structural properties of two given graphs (reference Figures 2a, 2b, and 2c as examples of expected outputs):

- `plot_node_degree_distribution_two_graphs`
- `plot_betweenness_centrality_distribution_two_graphs`
- `plot_ollivier_ricci_curvature_distribution_two_graphs`

Afther implementing these functions:

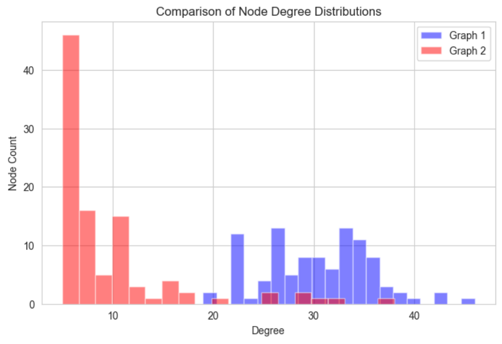
- Generate visualizations for both graphs.
- Examine the topological and geometrical characteristics of the graphs, compare their structures, discuss similarities and differences.

3.1.c Visualizing the Graphs (2 points)

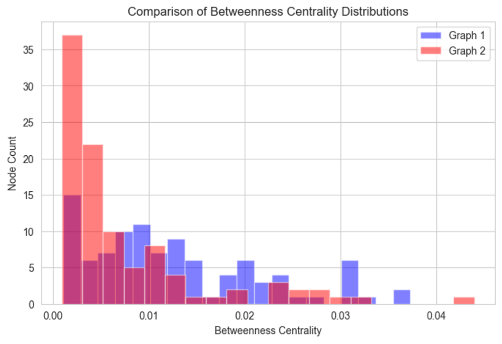
Implement the `plot_graph` function to generate visual representations of both graphs. Comment on the topologies/structures of the graphs.

3.1.d Visualizing Node Feature Distributions (2 points)

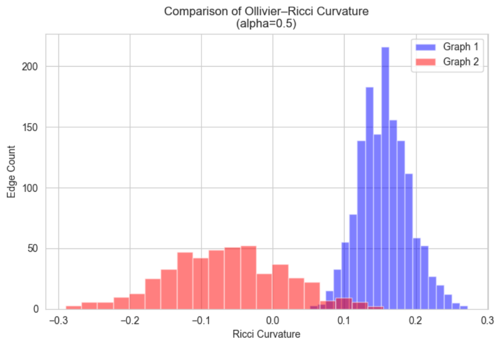
- Implement the function `plot_node_feature_dist_by_class_two_graphs` to visualize the average node feature distribution per class for two given graphs.
- Do not consider the distribution of a specific feature x_i, consider the mean of the feature vector $\mathbf{x}$ for each node.
- The function should generate a plot similar to Figure 3.



(a) Node degree distribution



(b) Betweenness centrality distribution



(c) Ollivier-Ricci curvature distribution

Figure 2: Comparison of graph measures (for reference only)

- Analyze the results. Discuss any observed overlaps between the node feature distributions across different classes. Are the features well-separated, or do they exhibit significant overlap? Compare the distributions between the two graphs and provide insights into their differences and similariteis.

3.2 Evaluating GCN Performance on Different Graph Structures (10 points)

3.2.a Implementation of Layered GCN (2 points)

- Implement a GCN where layers can be passed as input parameter and that can return all embedding layers. (If you will not be able to do so, to get results implement one class per layer).
- Train the GCN on G_1 and G_2 independently. Plot your results. (Note: you don't need to get a high performance at this stage, just make sure it trains).

3.2.b Plotting of t-SNE Embeddings for Graph Neural Networks (2 points)

- Use t-SNE to visualize the node embeddings for the final layer of the GCN for each graph for both training and evaluation data, use the labels as class labels for the tsne plot.

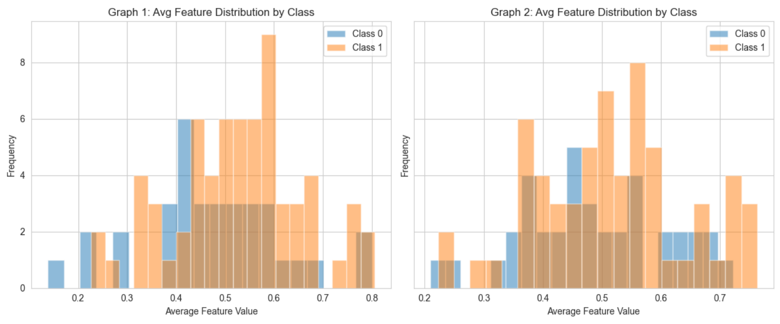


Figure 3: Example of node feature distribution per class.

3.2.c Training the Model on Merged Graphs $G = G_1 \cup G_2$ (3 points)

- Implement a GCN that trains on both graphs at once. Plot the training curve.
- Compare the performance of $G = G_1 \cup G_2$ against training on G_1 and G_2, respectively.
- Plot TSNE plots.

3.2.d Joined vs. Independent Training (3 points)

- Compare training results, embeddings, and overall observations between independent and merged training.
- Formulate a hypothesis explaining your observations.
- Discuss different implementation options for training on both graphs, including your choice. (e.g. what model modifications are possible; what training modifications are possible?)
- Provide relevant plots and tables where appropriate.
- Bonus: Implement a modification (or alternative approach) for training on $G_1 \cup G_2$ and compare outcomes.

3.3 Topological changes to improve training (10 points)

3.3.a Plot the Ricci Curvature for Each Edge (1 points)

- Plot a barplot of the the Ricci curvature per edge, where the x-axis are the edges and y-axis is the ricci curvature.

3.3.b Investigate Extreme Case Topologies (4 points)

- How would you modify a topology so that the graph structure is ignored, making the GNN behave like an MLP?
- What would an ideal graph structure be for optimal training and testing if labels were available (both during training and testing)?
- What are you observations? Analyze the scores and what does it mean about the dataset?
- Provide relevant plots and results.

3.3.c Improving Graph Topology for Better Learning (5 points)

- Implement two graph modifications.
- Describe your modifications.
- What motivated your implementation?
- If it helped give a hypothesis on why you think that is? If it did not give a hypothesis why you think it did not help.
- Provide relevant plots and results.

4 Submission Instructions

Your coursework submission must be in the form of a **ZIP file** containing the following:

- Three completed Jupyter Notebooks with **generated cell outputs**.
- **A well-structured report** following the designated template.
- The data folder must remain the same as provided to you; any changes or modifications to the data will be penalized.
- `requirements.txt` must be updated if you used any additional libraries or changed the versions of the libraries provided in the original `requirements.txt`.

4.1 File Organization

Ensure your ZIP file includes the following structure:

```
submission.zip
|-- notebook_Q1.ipynb
|-- notebook_Q2.ipynb
|-- notebook_Q3.ipynb
|-- report.pdf
|-- data folder (same as we provided to you)
|-- requirements.txt
```

4.2 Important Notes

- Your Jupyter Notebooks should be fully executed, with outputs visible.
- Your report should follow the given structure, using clear headings and properly formatted sections.
- If you used any external sources, ensure proper citations and references are included.

4.3 Penalties

Deductions will be applied for the following issues:

- **Logs not displaying as expected**. Ensure that any required logging information is correctly shown.
- **Not following the report template**. This includes missing sections, unclear headings, or disorganized content.
- **Uncommented code**. Each function and major block of code should have appropriate comments explaining its purpose.
- **Non-executable code blocks**. All code cells in the notebooks must be runnable without errors.
- **Missing references and citations**. If external sources are used, proper references must be included.

- **Modified Data**. You are not allowed to modify the data we provided to you.
- **Clarity in Figures**. Ensure that your figures are clear and well-presented, with visible legends, appropriate color schemes, descriptive titles, and properly labeled axes.

4.4 Final Checklist Before Submission

Before submitting, please ensure:

- Your Jupyter Notebooks are fully executed, and outputs are visible.
- The report is well-structured and follows the template.
- No unnecessary files are included in the ZIP file.
- The submission follows the required file structure.

Failure to adhere to these instructions may result in penalties. If you have any questions, please reach out before the submission deadline.