

# Graph Neural Network

- **Introduction**

- Graphs can be used to represent most real-world data. Objects can be denoted as nodes of the graph and edges can be used to represent relationships between them.
- Graphs are used in almost every field. In social networks, graphs are used to provide online recommendations, implement news feeds and calculate page rank. In the field of neuroscience, neurons are denoted by nodes and connections between them as edges. These graphs are then used to analyse the functionality of brain networks.
- GNNs are being used in real-world scenarios like Social Networks, Drug Discovery, Fraud Detection, Protein Folding, Contagion dynamics etc...
- There are many more applications in other fields. These prove the importance of working with graphs. Graph neural networks (GNNs) are a powerful class of deep learning models that are specifically designed for graph-structured data and they have been shown to be highly effective in capturing the structural information and dependencies present in graphs.
- GNNs have proven to be highly valuable and suitable for a wide range of real-world applications due to their ability to model and analyse graph-structured data, such as node classification, graph classification, graph generation, and link prediction, among others.
- GNNs can capture both local and global graph information, and adapt to varying graph structures and sizes and they can handle graphs with varying number of nodes, edges, and node features, making them suitable for graphs with different levels of complexity and heterogeneity.

- **Importance of graph classification**

- Graph classification tasks are of significant importance in various real-world applications due to their ability to capture and analyse complex relationships and dependencies among entities represented as graphs.

- Predicting Graph Properties
  - Understanding Graph Data
  - Enabling Graph-Based Machine Learning
  - Handling Heterogeneous Graphs
  - Scalability
  - Real-World Applications
- Accurate graph classification can lead to improved decision-making, better resource allocation, and enhanced performance in various real-world scenarios.
- **Why GNNs are used over SNNs**
  - Graph neural networks are specifically designed to operate on graph-structured data, which is different from the tabular or sequential data which are designed for traditional neural networks.
  - GNNs are used over traditional NNs for graph-structured data because they are capable of capturing the complex relationships and structures in graphs, learning both node-level and graph-level representations, handling graph variability, scalability, incorporating domain-specific features, and providing interpretable results. These capabilities make GNNs highly suitable for a wide range of real-world applications that involve graph-structured data.

## **Implementation**

- In the Graph classification problem, given is a family of graphs and a group of different categories, and we aim to classify all the graphs into the given categories and predict some output.
- Here we just Load and preprocess the graph data by reading graph data from a file and converting it into a suitable format that can be fed into a GNN. The graph data should typically include information about nodes, edges, adjacency matrices and their attributes because the graph consists of  $G = (V, E)$  Nodes(V), Edges(E) and attributes.

- Consider one problem statement which has graph(G) data, Adjacency matrix, Number of Nodes, Label(y) and feature vector for each node.

Adjacency matrix

$$\begin{matrix} & \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

Feature matrix

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

output(B)

$$\begin{bmatrix} 2 & 0 \\ 4 & 0 \\ 2 & 0 \\ 3 & 0 \\ 3 & 0 \end{bmatrix}$$

- Here we Divide the graph data into train and test sets to evaluate the performance of the GNN model and we Use the training set to train the GNN model and test set to evaluate the performance of the trained GNN model.

- Training set:**

- Here GNN model needs to be trained on labelled graph data. Here we just feed the graph data into the GNN architecture, computing the forward and backward passes, and updating the model weights based on the chosen optimization algorithm (e.g., stochastic gradient descent, Adam, etc.).

- Forward Propagation**

- Step-1: **Aggregation Step**

- We just take the **Adjacency matrix(5x5)** and **Feature matrix(5x2)** (feature vector for each node is filled into a matrix of size(Number of Nodes)) we take these two matrices and do dot product store in **B(5x2)**(output matrix).

Adjacency matrix

$$\begin{matrix} & \begin{matrix} A & B & C & D & E \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

Feature matrix

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

output(B)

$$\begin{bmatrix} 2 & 0 \\ 4 & 0 \\ 2 & 0 \\ 3 & 0 \\ 3 & 0 \end{bmatrix}$$

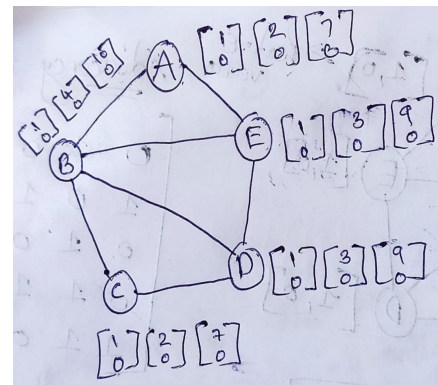
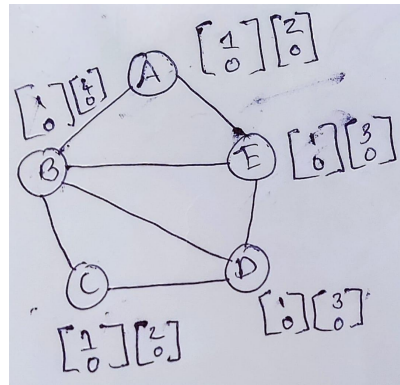
- By observing the output matrix(**B**) we understand that those values that are present in the matrix are the node degrees for each node.(Here Degree 2 is for node A, Degree 4 is for node B, so on..).

#### ■ Step-2: Neighbourhood Aggregation Step

- Here we just take the **output Matrix(B)**(5x2) from the aggregation step and **weight matrix**(2x2)(here weight matrix the values are taken randomly and the size of the matrix is decided based on the size of feature vector dimension) we take these two matrices and do dot product by adjusting the dimension of matrix '**B**' and store in **X(5x2)**.

The image shows a handwritten matrix multiplication. On the left is matrix B (5x2) with values:
$$B(5 \times 2) = \begin{bmatrix} 2 & 0 \\ 4 & 0 \\ 2 & 0 \\ 3 & 0 \\ 3 & 0 \end{bmatrix}$$
In the middle is the weight matrix (2x2) with values:
$$\text{weight}(2 \times 2) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
On the right is the resulting matrix X (5x2) with values:
$$X(5 \times 2) = \begin{bmatrix} 2 & 0 \\ 4 & 0 \\ 2 & 0 \\ 3 & 0 \\ 3 & 0 \end{bmatrix}$$
The equation is written as:
$$B(5 \times 2) \cdot \text{weight}(2 \times 2) = X(5 \times 2)$$

- By observing the B matrix we understand that those values that are present in the matrix are the sum of node degrees for the current node and its neighbourhood where each node aggregates the information from its neighbouring nodes.
- If we continue these two steps it will cover the whole graph and aggregate the information from its neighbouring nodes. Below graph shows the updated feature vectors for each node.



### ■ Step-3: Relu

- The ReLU function is applied element-wise to the (output) $\mathbf{X}$  of the message passing or node update step in the GNN, where each node's aggregated information from its neighbours is combined with its own features.
- Relu function allows it to learn complex patterns and representations from the graph data. It ensures that only positive values are passed forward, while values less than zero are set to zero. This helps in capturing important features and discarding irrelevant or negative information from the graph data.
- By applying the Relu function the output matrix will be shown below.

$$\text{Relu} = \max(0, x)$$

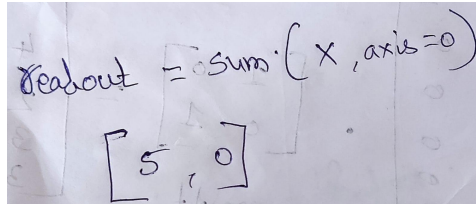
$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

### ■ Step-4: Readout

- In the Graph level representation Readout function will sum up the node representations(feature matrix ' $\mathbf{X}$ ') across the graph

element-wise to generate a fixed-size graph-level representation.

- This captures the global information of the graph. This graph-level representation is then used as input to a classifier to make the final prediction for the graph classification task.



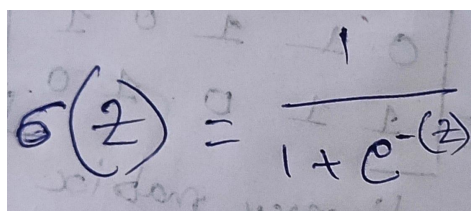
A handwritten formula on a piece of paper. It reads:  $\text{Readout} = \text{sum}(X, \text{axis}=0)$ . Below this, there is a small matrix representation  $\begin{bmatrix} 5 & 0 \end{bmatrix}$ .

#### ■ Step-5: Linear Function ( $y = AX + b$ )

- The Linear function  $y = mx + c$ , where  $y$  is the predicted output,  $A$  is the slope,  $X$  is the input, and  $b$  is the bias, is a simple linear function commonly used in regression problems. In graph classification problems, the goal is to predict the class of an entire graph, rather than predicting continuous values as in regression.
- Here  $\text{output}(X)(1 \times 2)$  from the Readout function and  $A$  **matrix** $(1 \times 2)$  (here in 'A' matrix the values are taken randomly and the size of the matrix is decided based on the size of feature vector dimension) we take these two matrices and do dot product and add to Bias( $b$ ) and store in  $z$  (Scalar value).

#### ■ Step-6: Sigmoid

- Sigmoid function can be used as an activation function in Graph Neural Networks (GNNs) to transform the outputs of the model and make predictions between 0 and 1.
- Here we input the ' $Z$ ' and we get a value between 0 to 1.



A handwritten formula on a piece of paper. It reads:  $\sigma(z) = \frac{1}{1 + e^{-z}}$ .

- **Step-7: Binary classification**

- Here we just input the sigmoid output(**P**) value to get the predicted class between either **0** or **1**. This can be done by applying some conditions.
- Condition: if **P** > **0.5** then **1** and **P** < **0.5** then **0**.so predicted label becomes either **0** or **1**.

- **Loss function**

- Here we use the Binary cross entropy loss function which is commonly used for binary class graph classification problems where the goal is to predict binary outcome.
- It is defined as here **y**(label), **z**(linear function)
- $Loss = y * (\log(1 + e^{(-z)})) + (1 - y) * (\log(1 + e^{(z)}))$
- Now we check if the loss is zero or close to zero. If not then we have to decrease the loss by updating the parameters '**W**', '**A**', '**b**' and this is done by backpropagation.

- **Backward Propagation**

- **Step-1: Initialization**
  -
- **Step-2: Initializing Epsilon**
  - **(0, 0) position**
- **Step-3: Adding Epsilon**
  -
- **Step-4: Forward Propagation**
  -
- **Step-5: Loss function**
  -
- **Step-5: Normalisation**
  -
- **Step-6: Repeat from Step-3**
  -
- 

- **Optimization**

- **Testing set:**

- **Datasets**

- TUDataset is a collection of benchmark datasets for learning with graphs classifications, specifically for graph machine learning and graph neural network(GNN).It provides a standardised way to evaluate the performance of GNN models on a wide range of graph classification tasks.
- It is widely used in the research community as a benchmark for evaluating the performance of GNNs on various graph-related tasks.we use this dataset for graph classification, It includes a diverse set of datasets with



different graph structures, sizes, and properties, covering various domains such as social networks, bioinformatics, and chemistry.

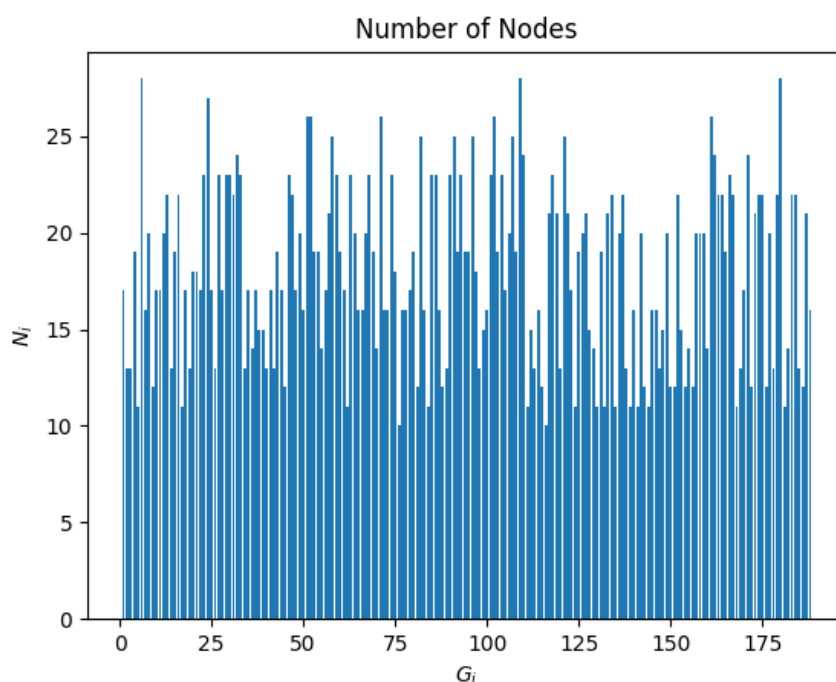
- These datasets help us to evaluate the performance of different GNN models, compare their results, and develop new graph-based machine learning methods.
  - MUTAG Dataset
  - NCI1 Dataset
  - PROTEINS Dataset
  - ENZYMES Dataset
  - MCF-7 Dataset
  - YEAST Dataset
  - PCT\_FM Dataset
  - NCI109 Dataset
- The TUDataset library provides a Python API to easily access and load the datasets and it provides functions for processing and analysing graph data, it helps in splitting the graph data into a training set and testing set so that we can access it easily.

- **Dataset statistics :**

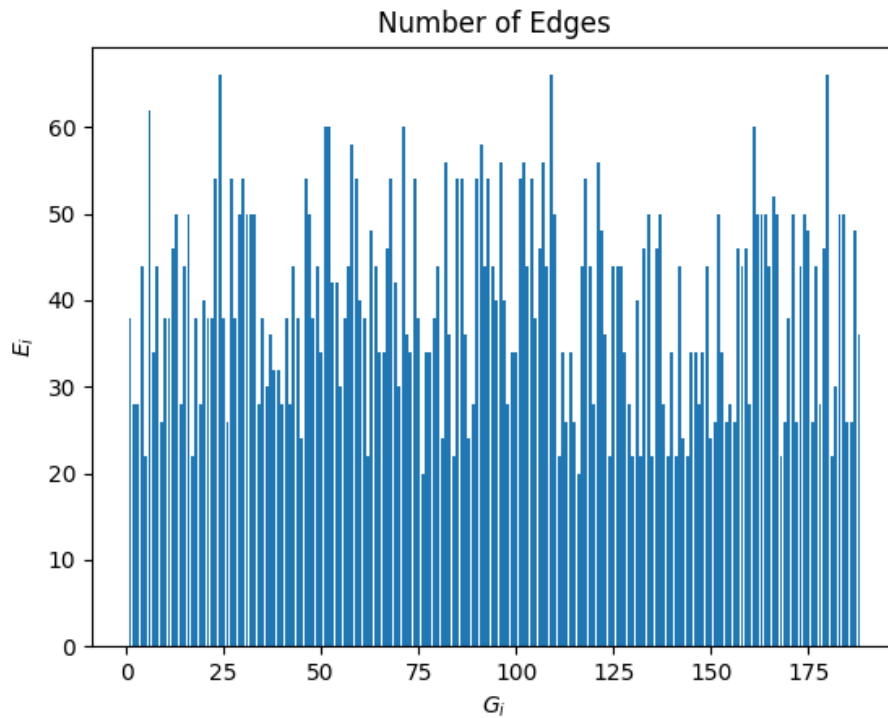
<b>DATASET</b>	<b>Number of Graphs</b>	<b>Number of Features</b>	<b>Training Set</b>	<b>Testing Set</b>	<b>Training Accuracy</b>	<b>Testing Accuracy</b>
<b>Mutag</b>	188	7	150	38	86.6%	68.4%
<b>NCI1</b>	4,110	37	3,300	810	68.5%	67.2%
<b>PROTEINS</b>	1,113	3	888	225	71.9%	73.3%
<b>ENZYMES</b>	600	3	480	120	37.0%	33.3%
<b>MCF-7</b>	27,770	46	22,000	5,770	91.8%	91.4%
<b>YEAST</b>	79,601	74	64,000	15,601	88.1%	88.3%
<b>PTC_FM</b>	349	18	280	69	72.1%	65.2%
<b>NCI109</b>	4,127	38	3,300	827	67.4%	62.9%

- **MUTAG Dataset:**

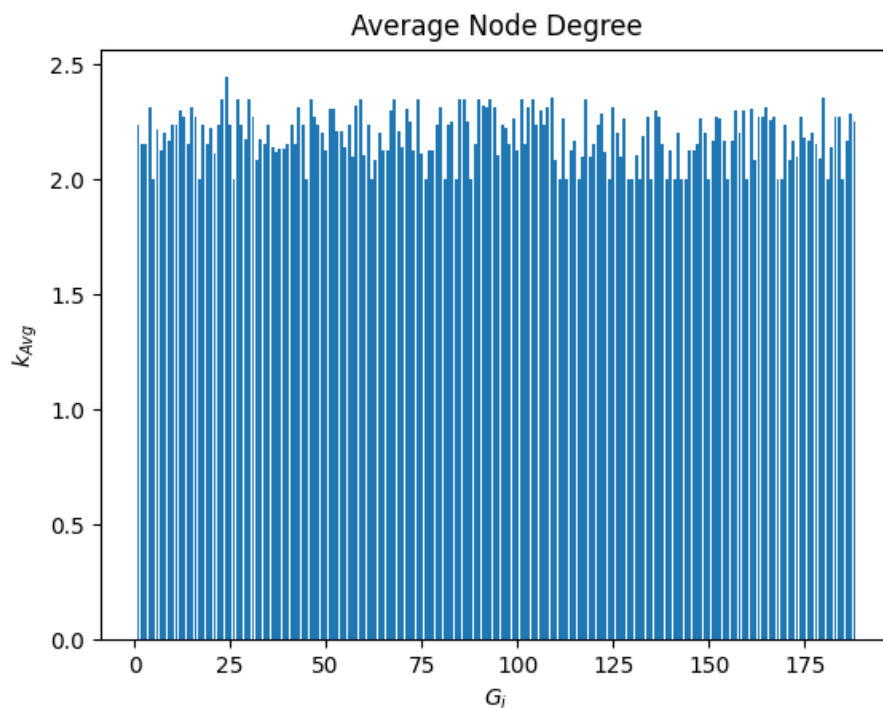
- The MUTAG dataset is a collection of 188 chemical compounds that are divided into two classes based on their mutagenic effect on a bacterium.
  - Chemical data is represented as graphs
  - Nodes represent atoms
  - Edges represent chemical bonds
- The goal of the GNN research on this dataset is typically to train a model to predict the mutagenic class of chemical compounds based on their graph representation.
- MUTAG dataset is loaded into a GNN model, the model is trained using Graph Convolutional Network, Graph Neural Network and so on. The goal is to learn the patterns or representations from the graph data that can effectively predict the mutagenic class of the chemical compounds such as mutagenic (1) or non-mutagenic (0), based on the graph structure and associated node/edge features.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



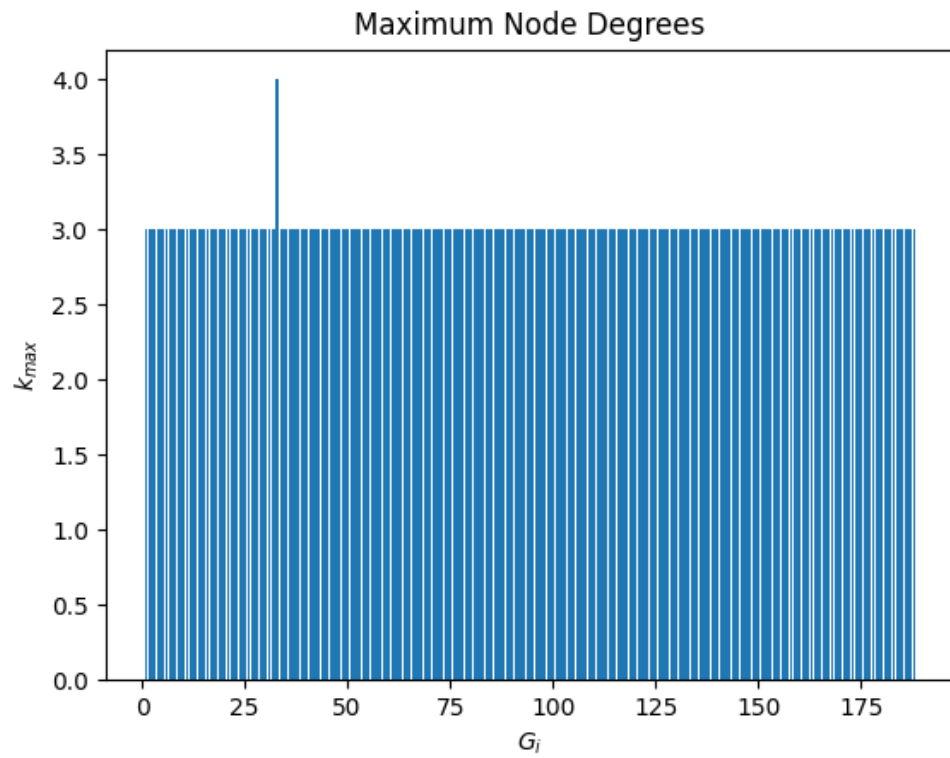
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



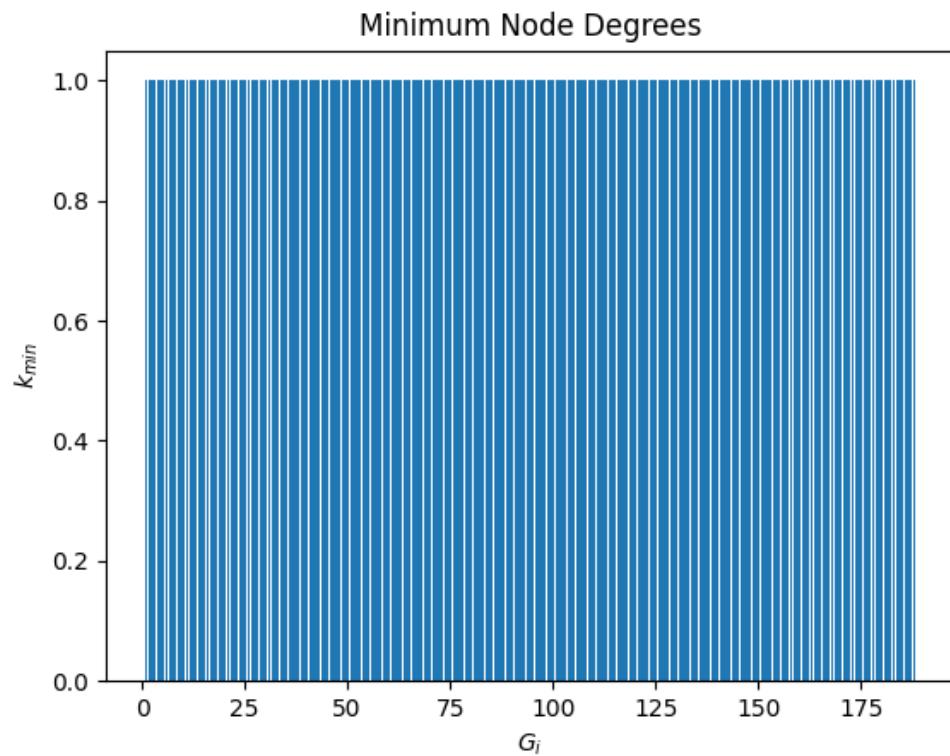
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



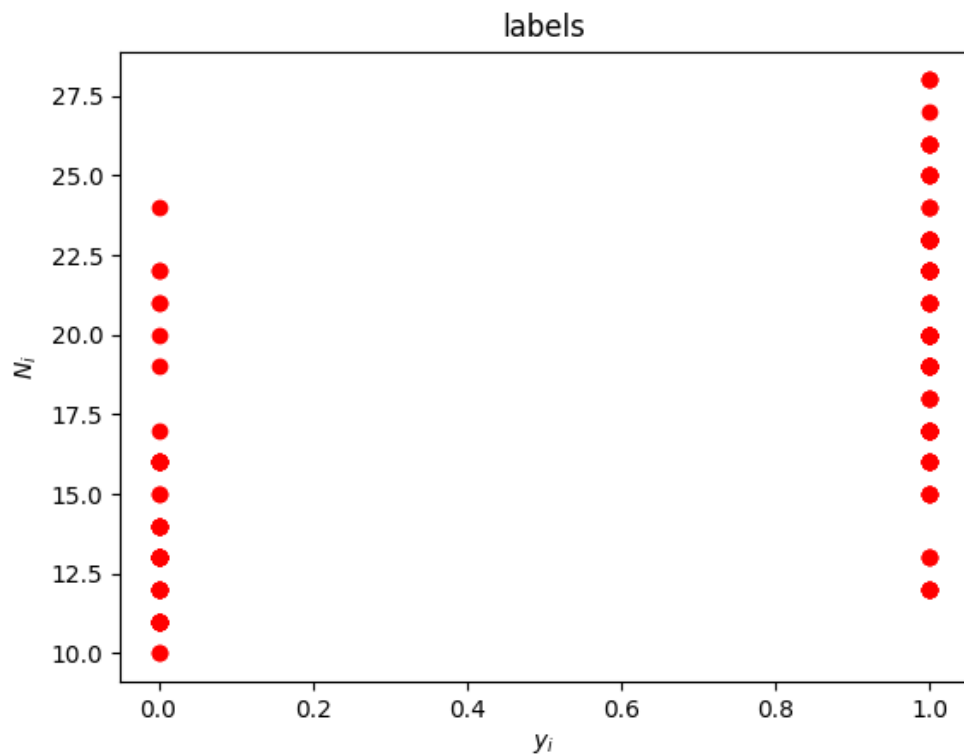
- Plotting a Maximum node degree for each graph



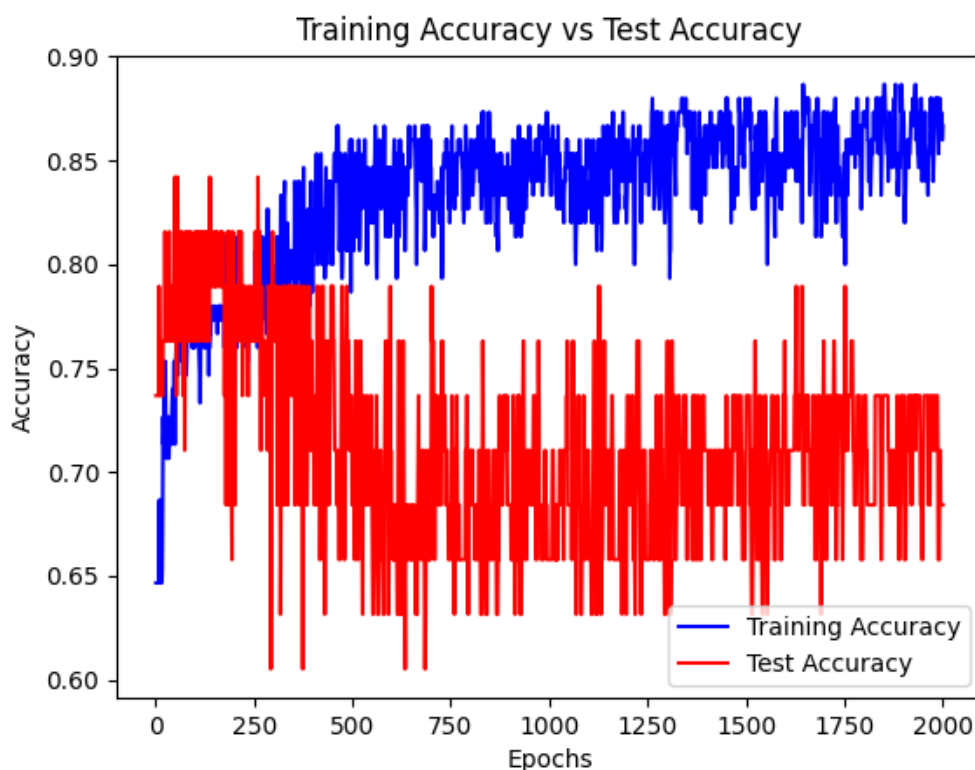
- Plotting a Minimum node degree for each graph



- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



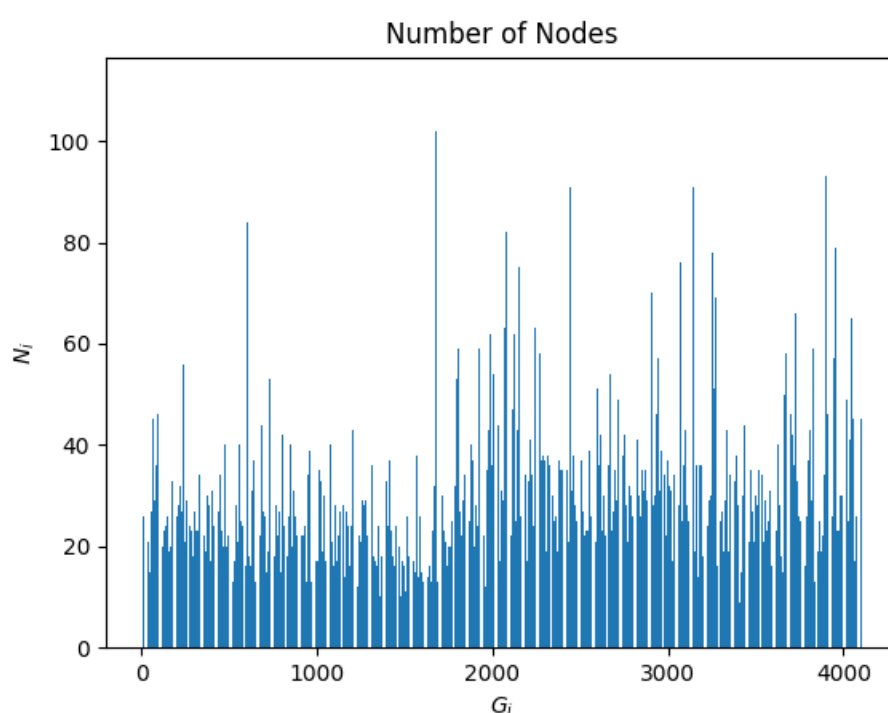
- Plotting a Loss for each graph tells us how loss is varying over every epoch
- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch



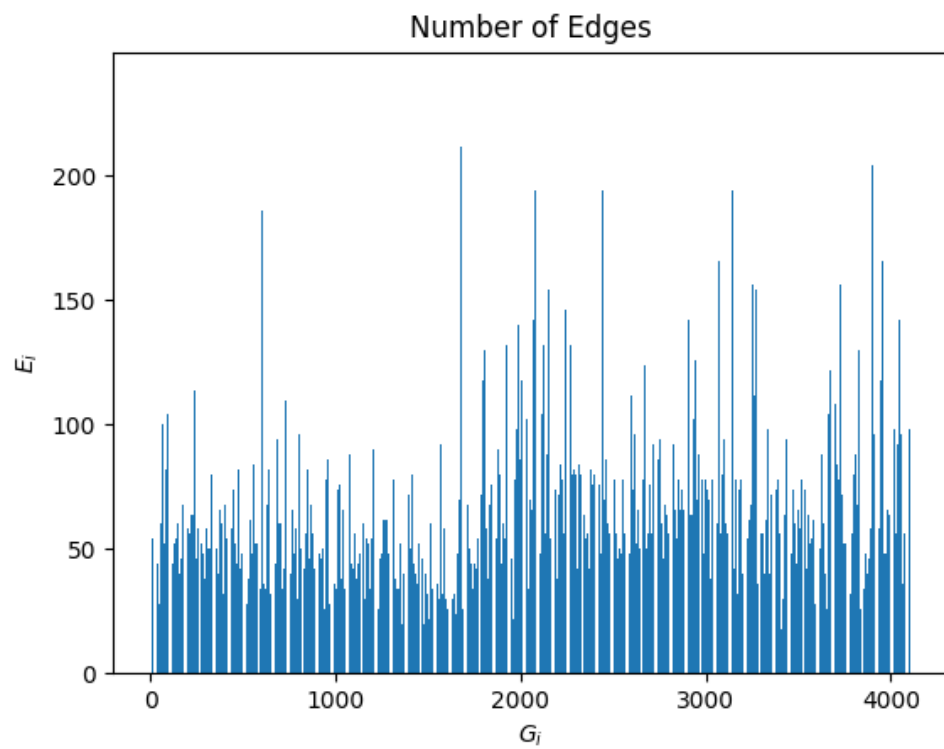
- **NCI1 Dataset:**

- The NCI1 dataset is a collection of 4110 chemical compounds that come under the cheminformatics domain that are divided into two classes based on the anti-cancer screens where the chemicals are assessed as positive or negative to cell lung cancer.
  - Chemical data is represented as graphs
  - Nodes represent atoms
  - Edges represent chemical bonds
- The NCI1 dataset is a binary classification dataset, where compounds are labelled as either 0 or 1. The label "1" indicates that the compound has anti-cancer activity, while the label "0" indicates that the compound does not have anti-cancer activity.
- The NCI1 dataset consists of a total of 4,110 compounds, which are divided into 3,500 compounds for training and 610 compounds for testing and it is mainly used for graph classification, chemical property prediction.

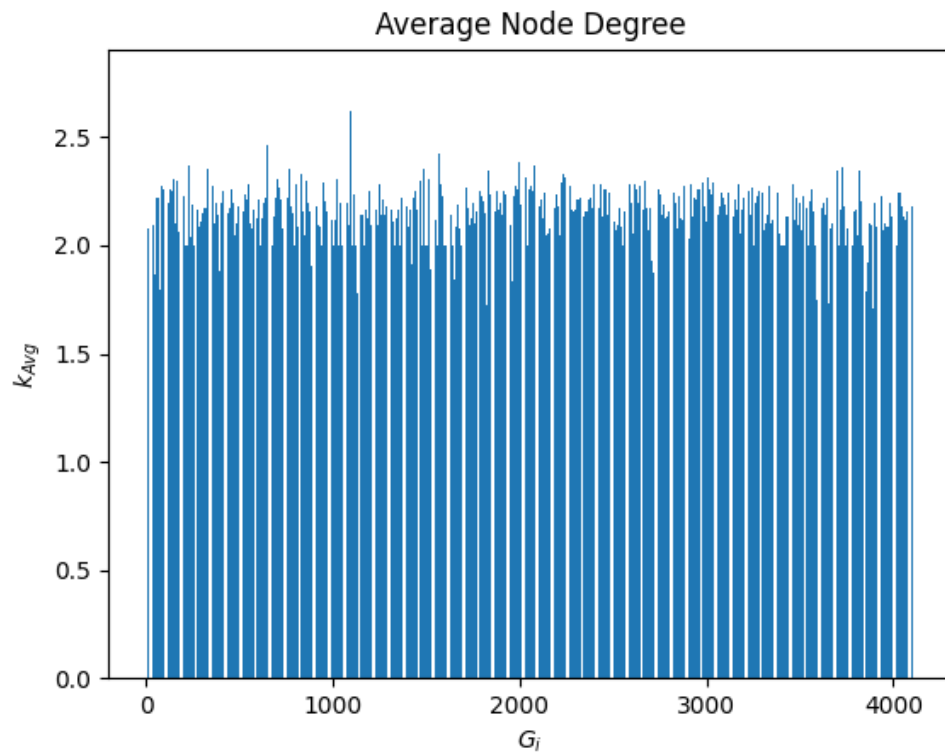
- The NCI1 dataset is the complex and variable nature of chemical compounds, which makes it difficult to accurately predict their anti-cancer activity. Compounds in the dataset can have different sizes, atom types, and bond types so that model has to learn more in order to produce good results.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.

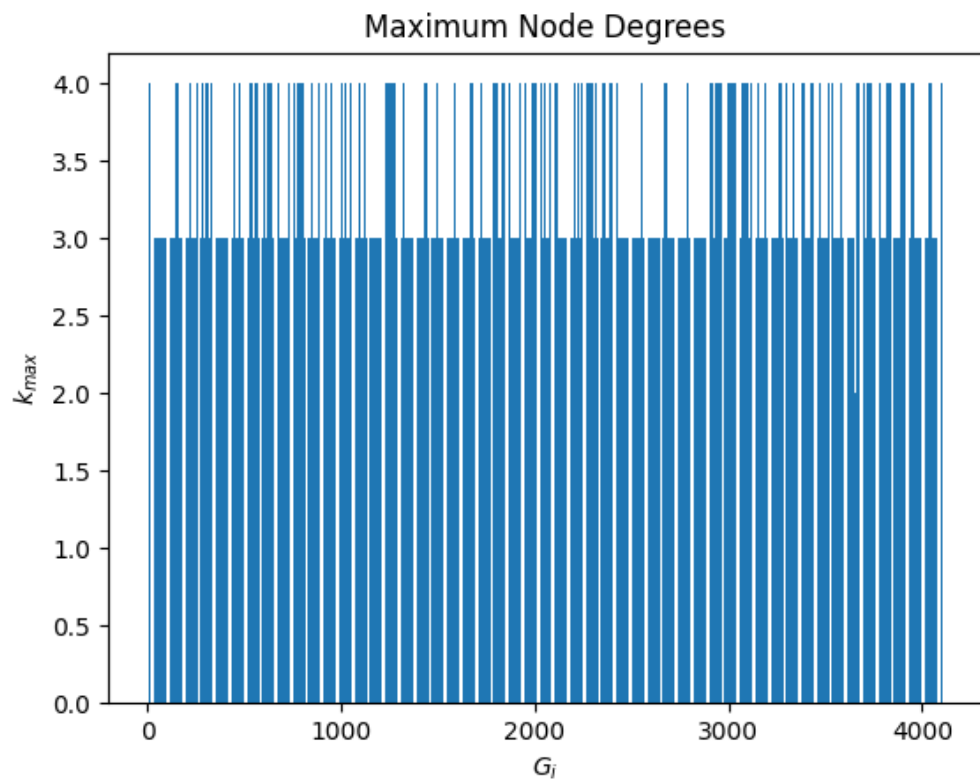


- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.

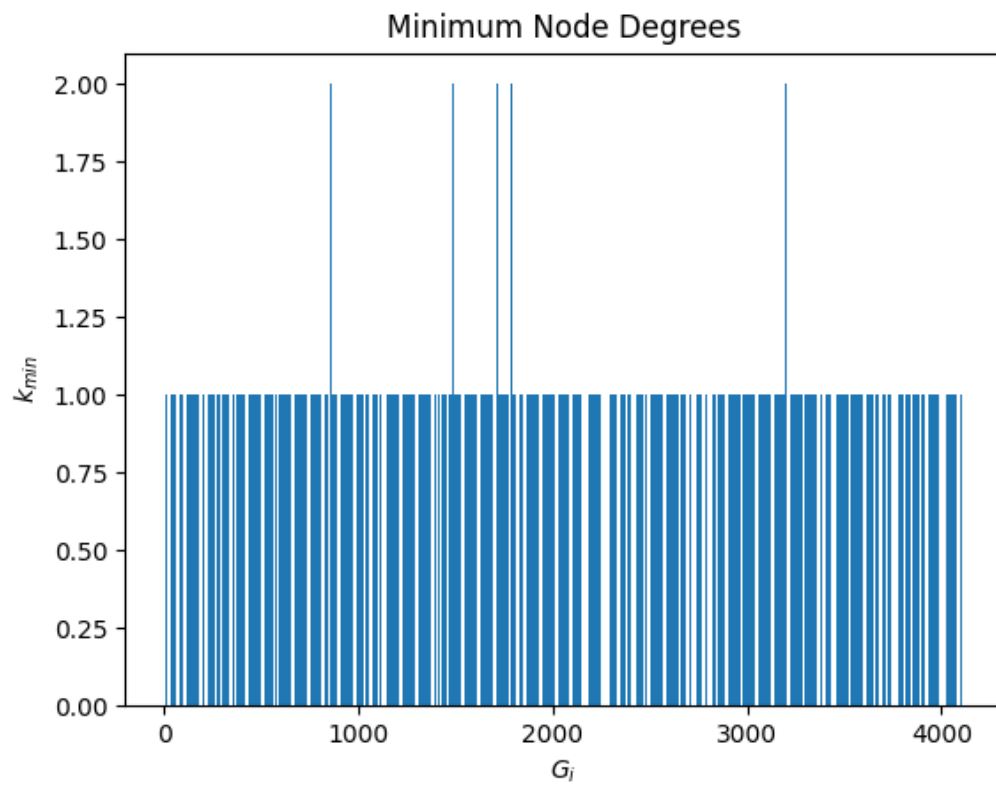




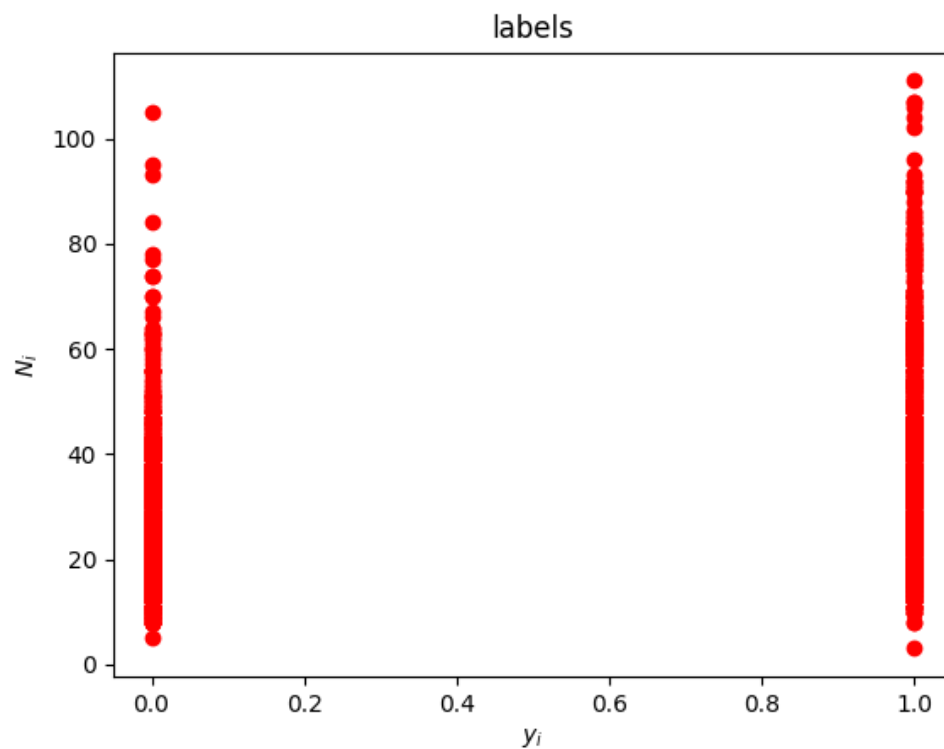
- Plotting a Maximum node degree for each graph



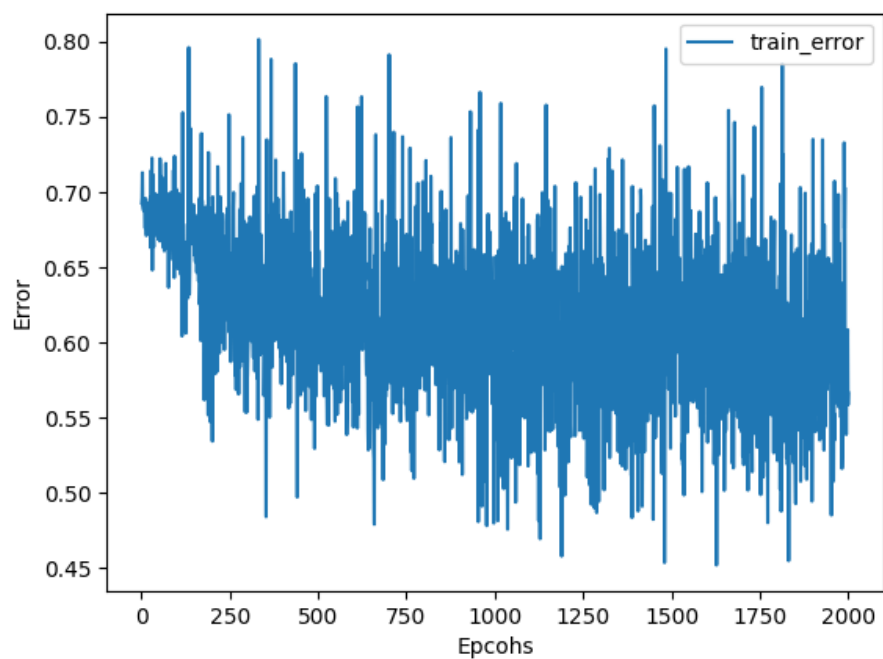
- Plotting a Minimum node degree for each graph



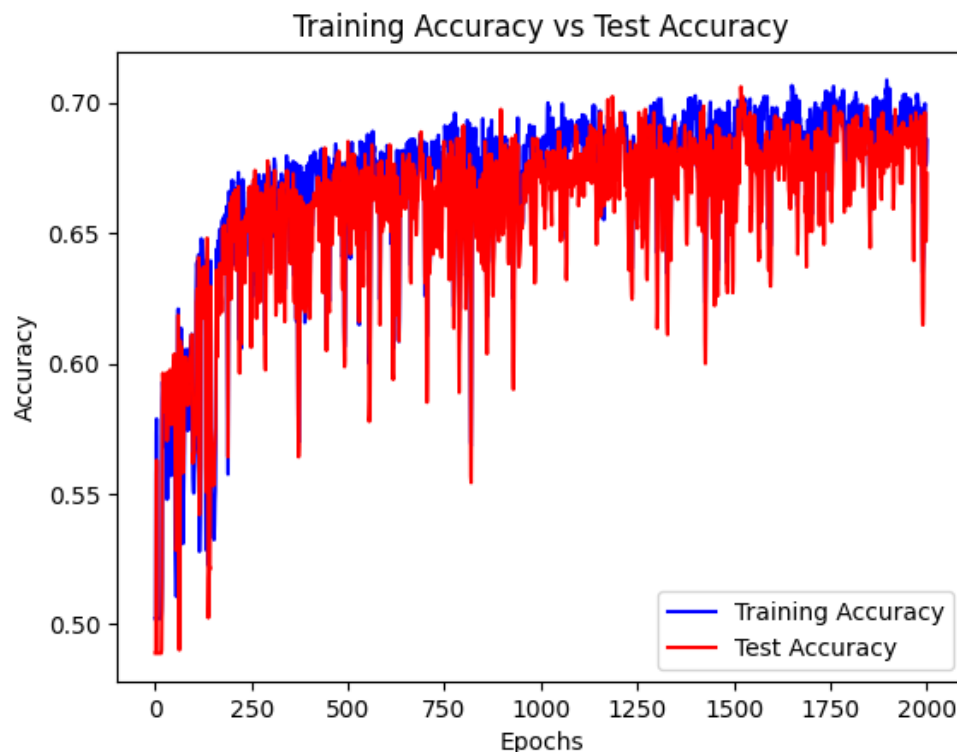
- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



- Plotting a Loss for each graph tells us how loss is varying over every epoch



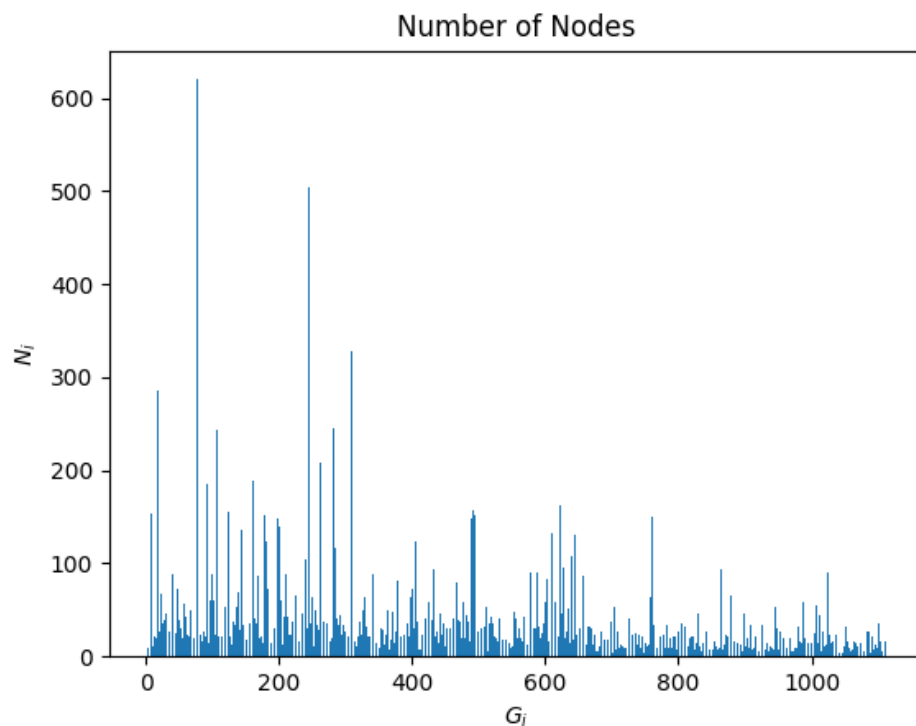
- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch



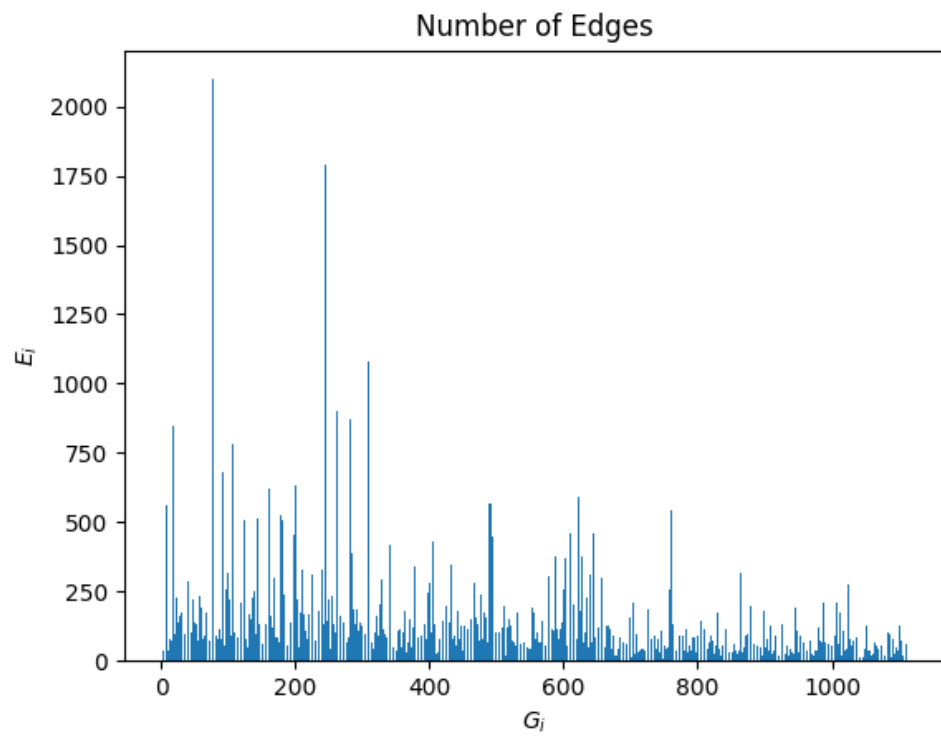
- **PROTEINS Dataset:**

- Proteins is a dataset of proteins that are classified as enzymes or non-enzyme that are divided into two classes based on Angstroms. These datasets are widely used in bioinformatics and computational biology to study protein structure, function, interactions, and evolution.
  - Chemical data is represented as graphs(1113)
  - Nodes represent amino acids
  - Edges represent bonds
  - Node features: Atom-level features such as element type
- The Proteins dataset is a binary classification dataset, where the label indicates whether the protein is an enzyme or not.
- Enzyme : Graphs with label '1' represent proteins that are enzymes and these proteins do possess enzymatic activity.

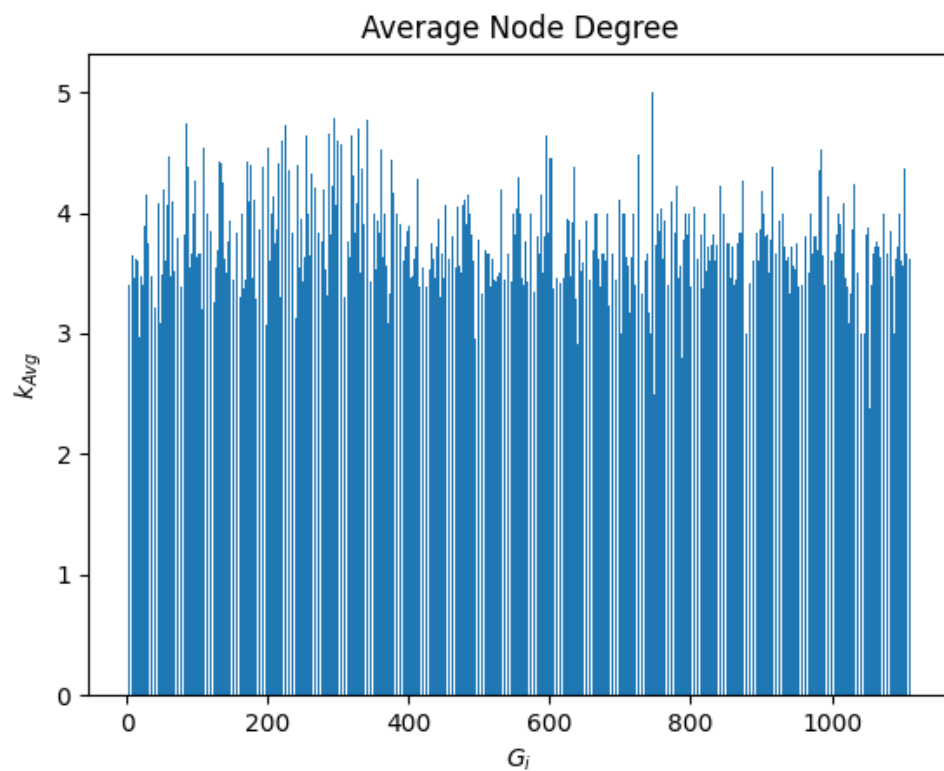
- Non-Enzyme : Graphs with label '0' represent proteins that are not enzymes and these proteins do not possess enzymatic activity and are typically involved in other cellular functions or structural roles within cells.
- Proteins dataset is used for tasks such as protein classification, enzyme prediction, and other protein-related analysis tasks using graph-based machine learning models, such as graph neural networks (GNNs).
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



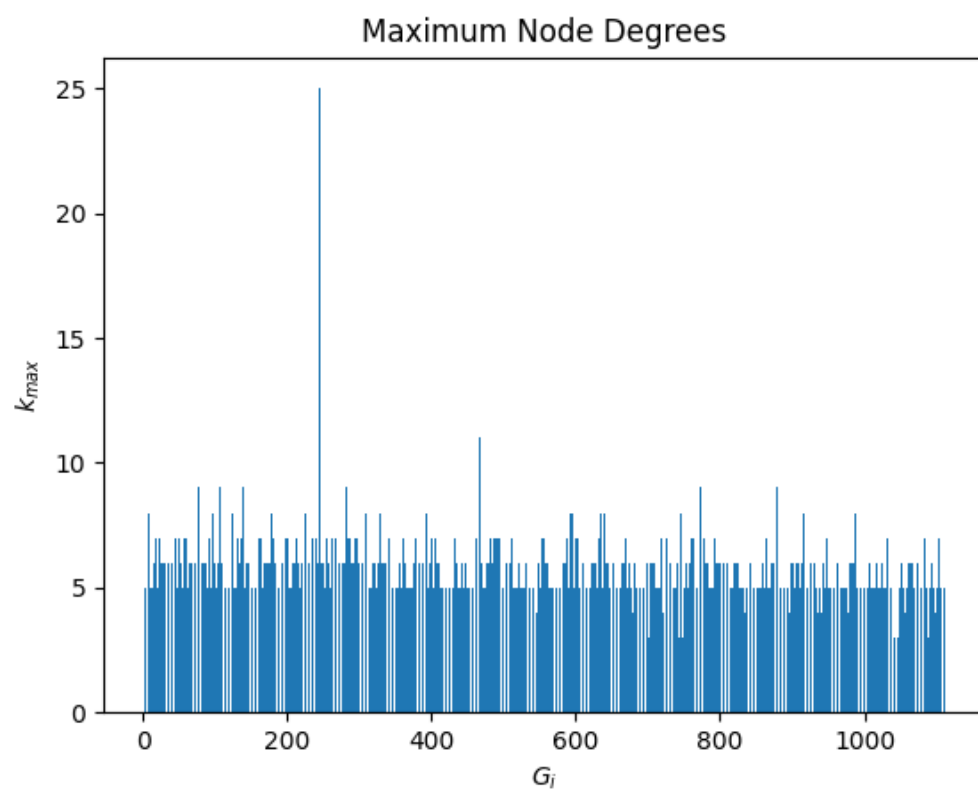
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



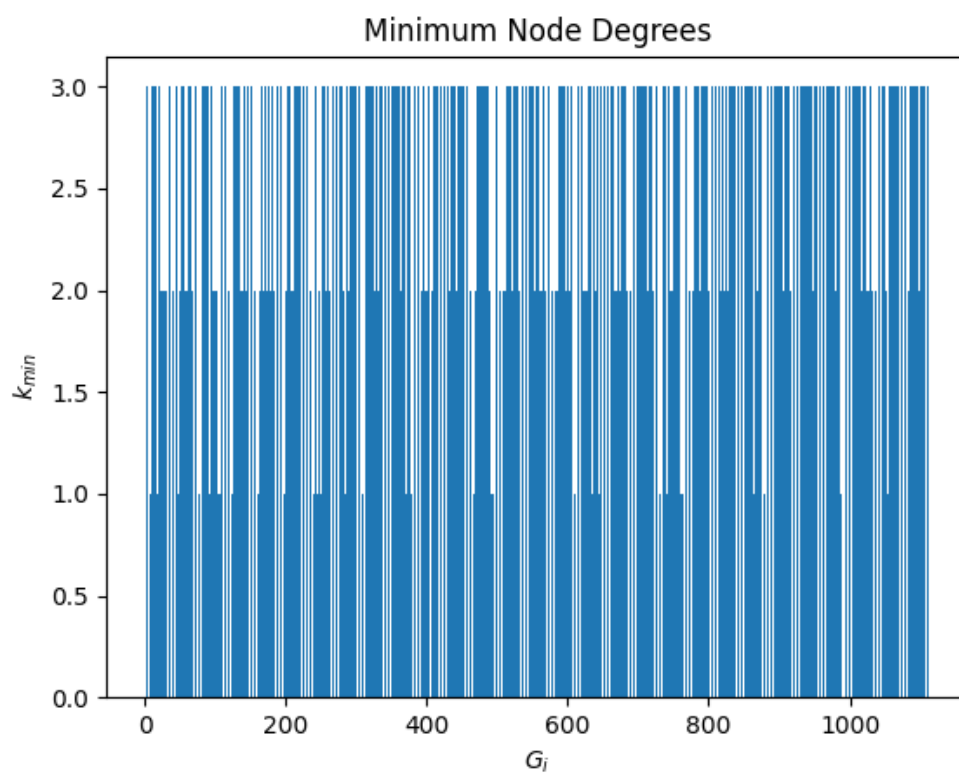
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



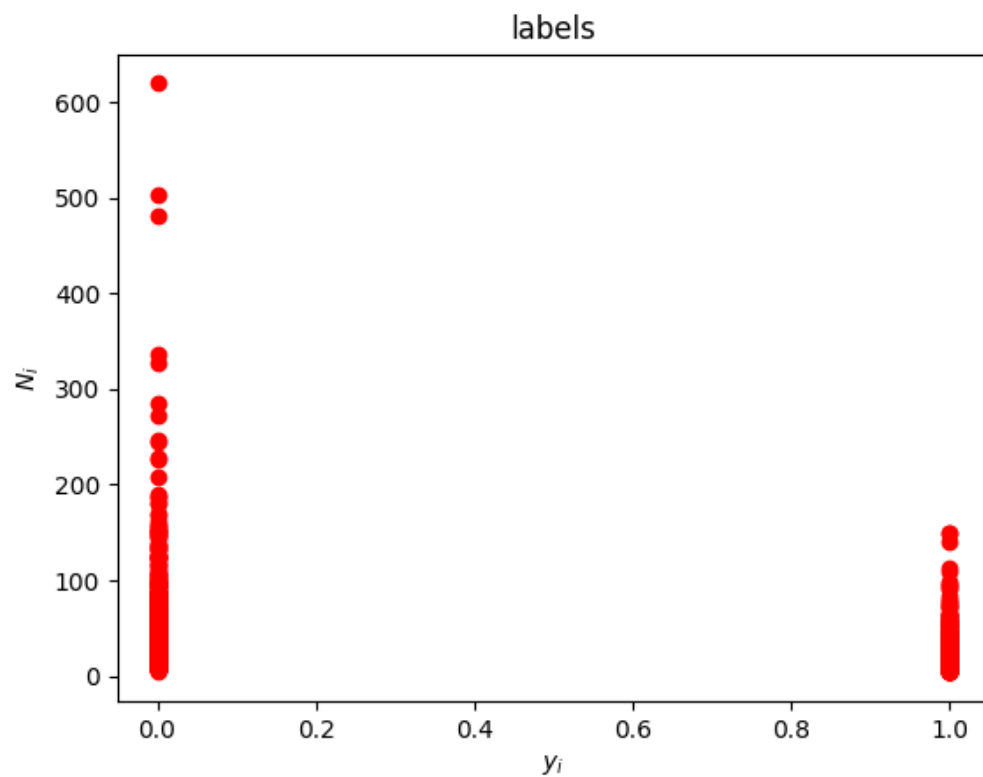
- Plotting a Maximum node degree for each graph



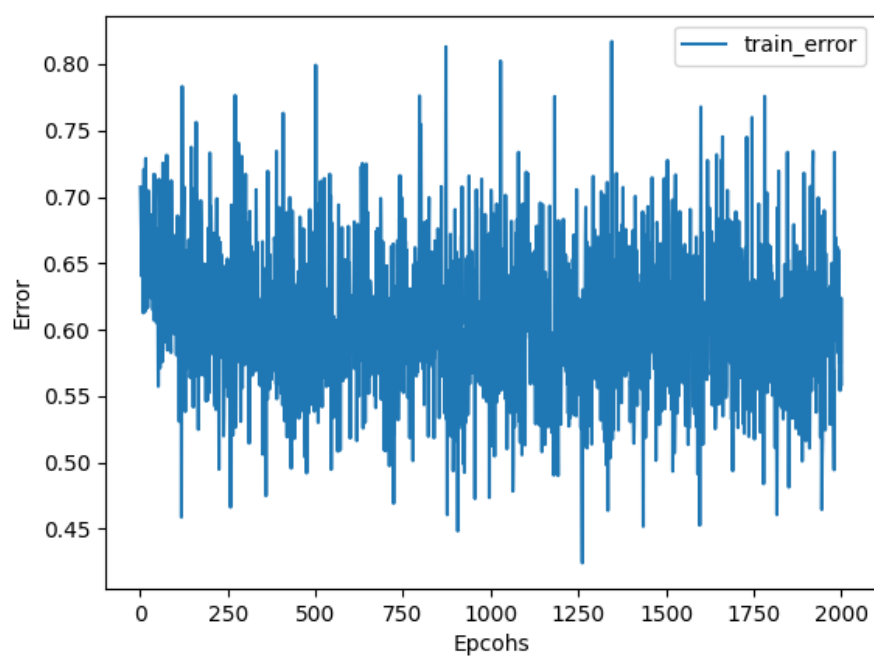
- Plotting a Minimum node degree for each graph



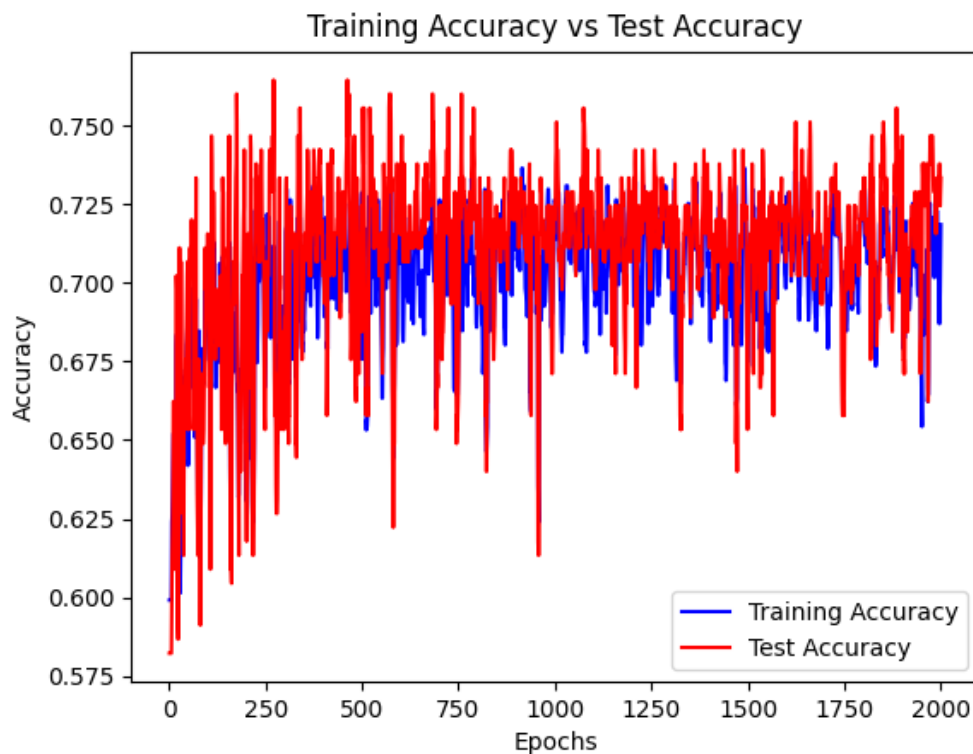
- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



- Plotting a Loss for each graph tells us how loss is varying over every epoch



- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch

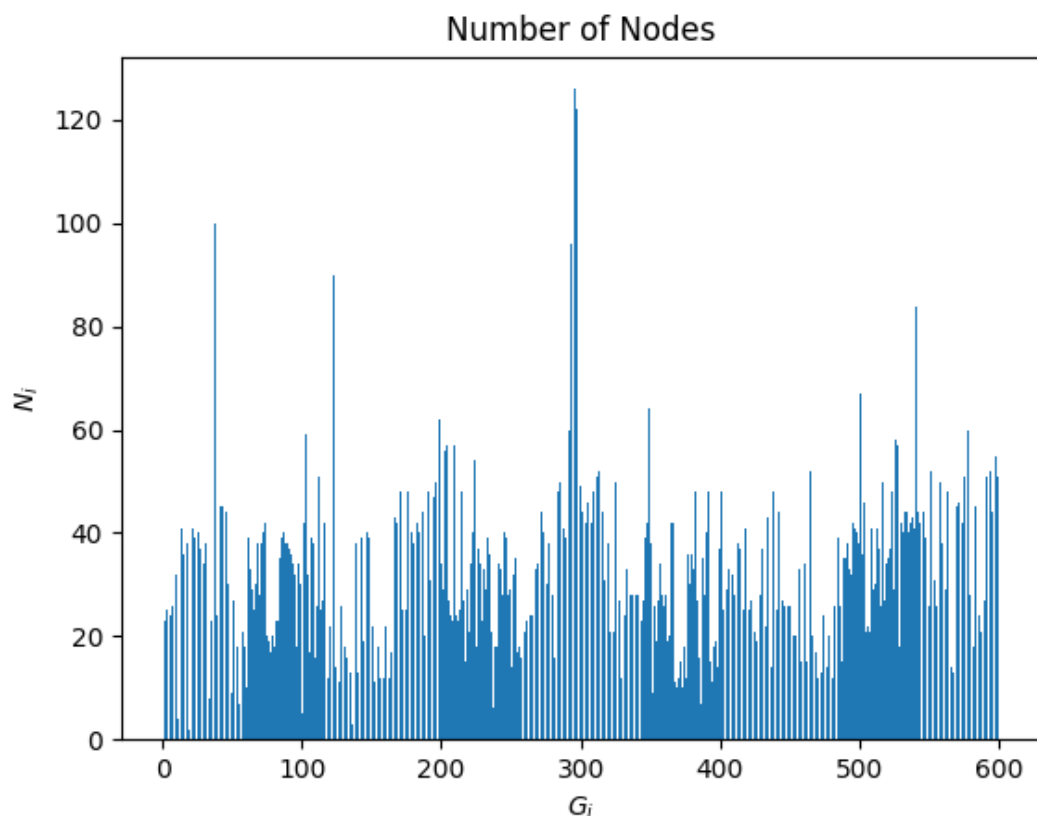


- **ENZYMES Dataset:**

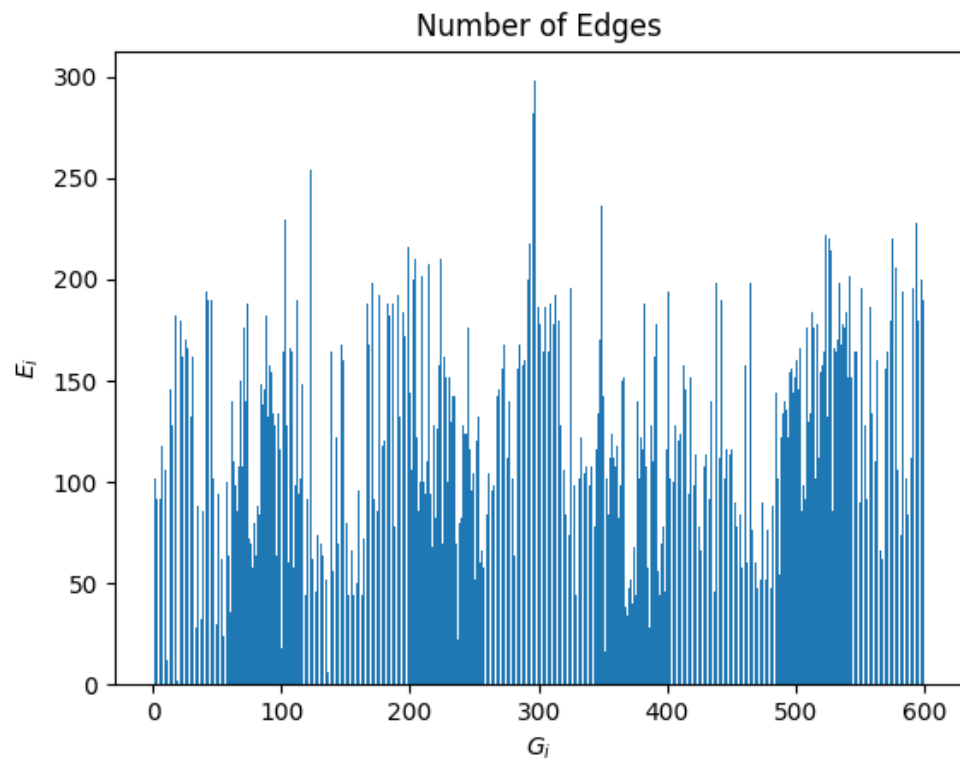
- The Enzymes dataset is a collection of '600' enzyme graphs, where each graph represents the structural information of an enzyme molecule. Enzymes are biological molecules that function as catalysts in living organisms.
  - Graphs represent molecular structure of enzyme
  - Nodes represent atom
  - Edges represent chemical bond
  - Node features: Atom-level features such as element type
- The Enzymes dataset is used for classification tasks where the goal is to predict the functional class or family of enzymes based on their molecular structure.



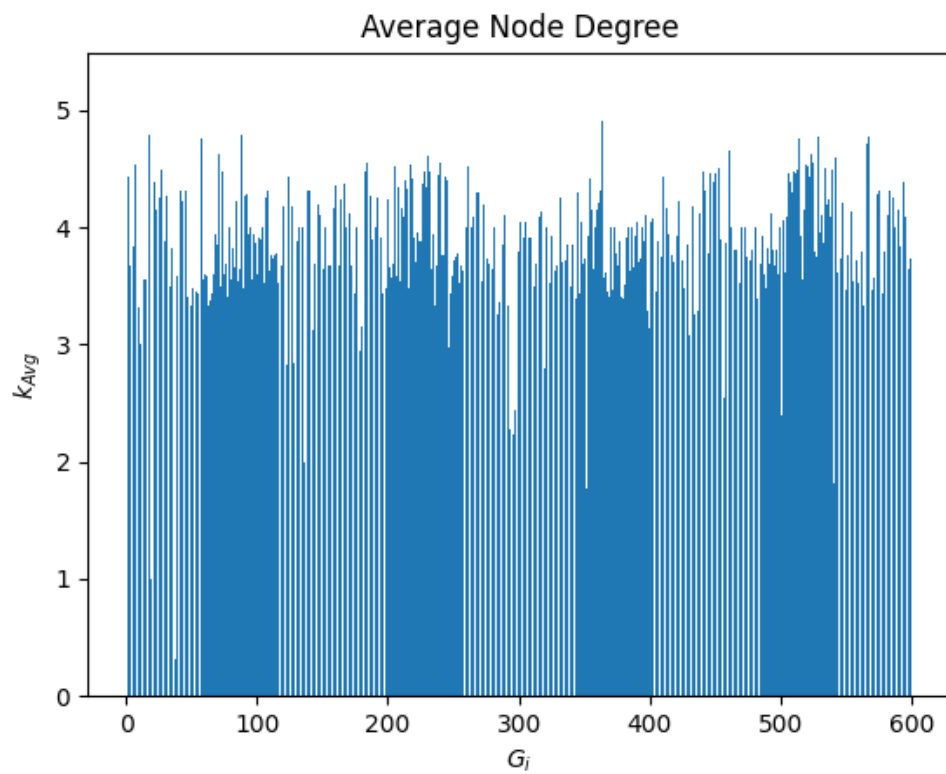
- Dataset typically includes labels for each graph. These labels are used as the target labels(6) for the classification task, and the goal is to train a deep learning model that can accurately predict the functional class or family of enzymes based on their molecular structure.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



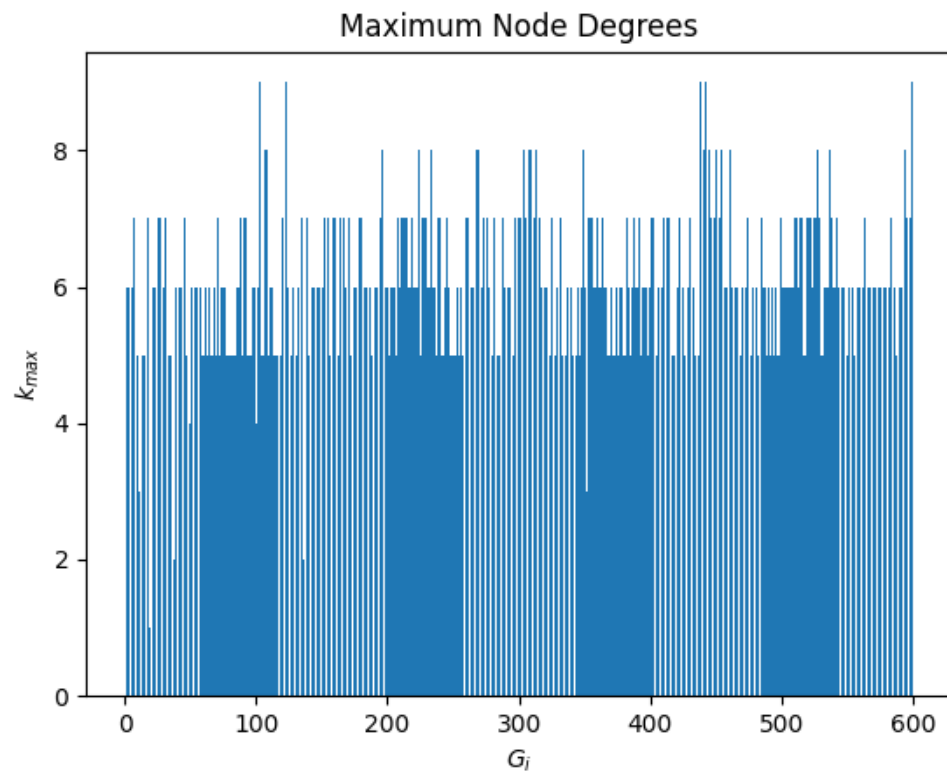
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



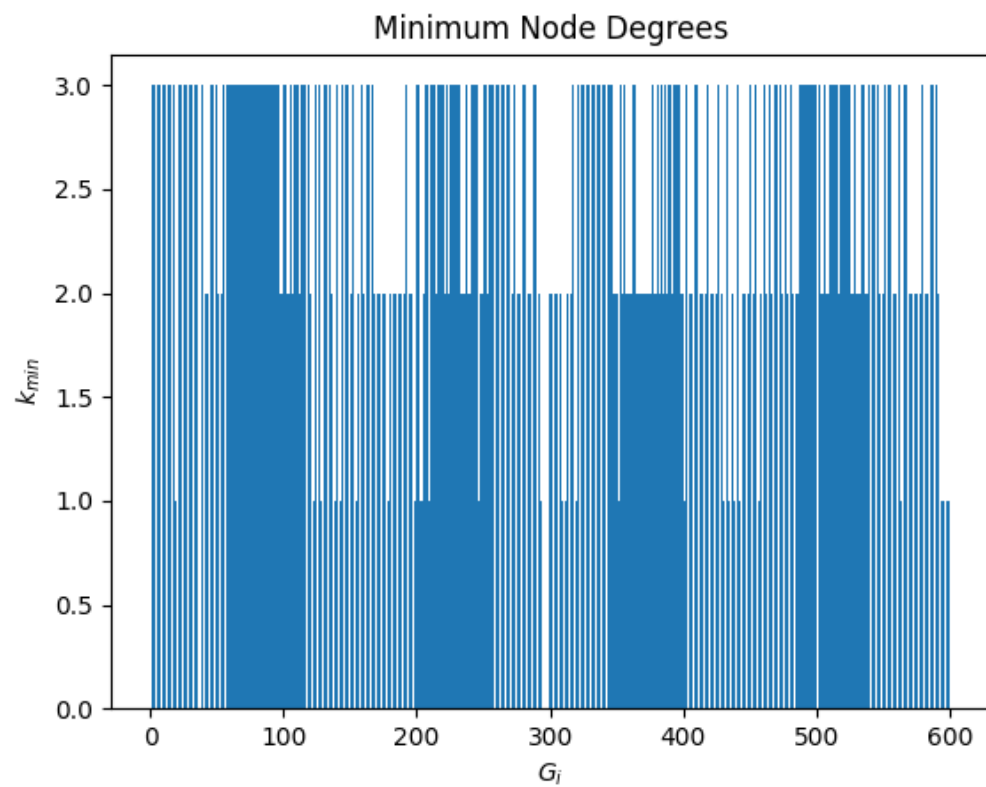
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



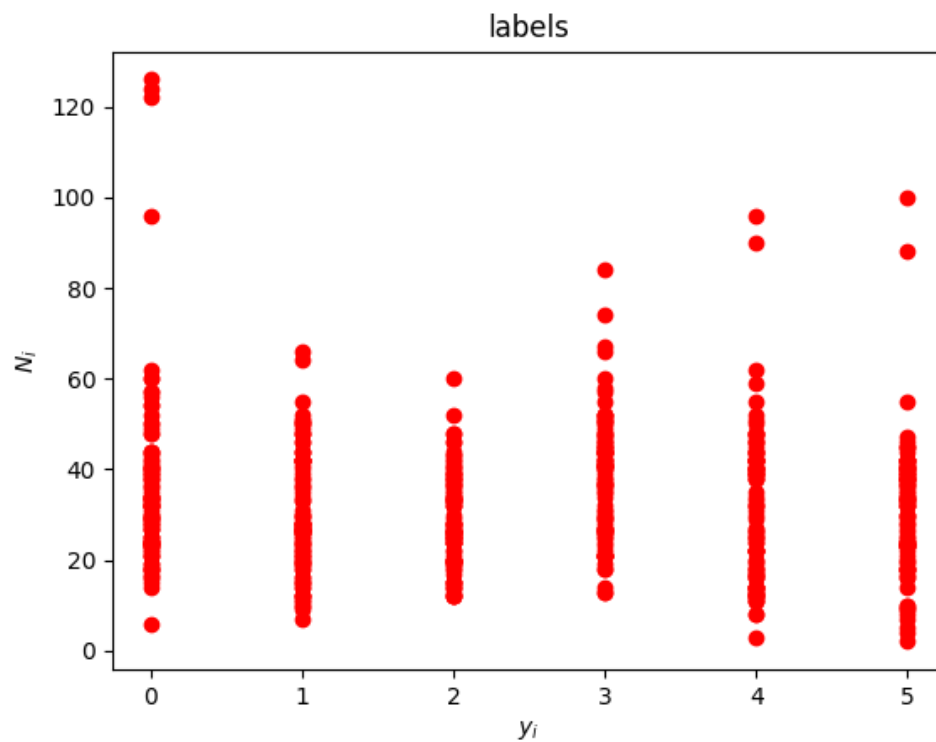
- Plotting a Maximum node degree for each graph



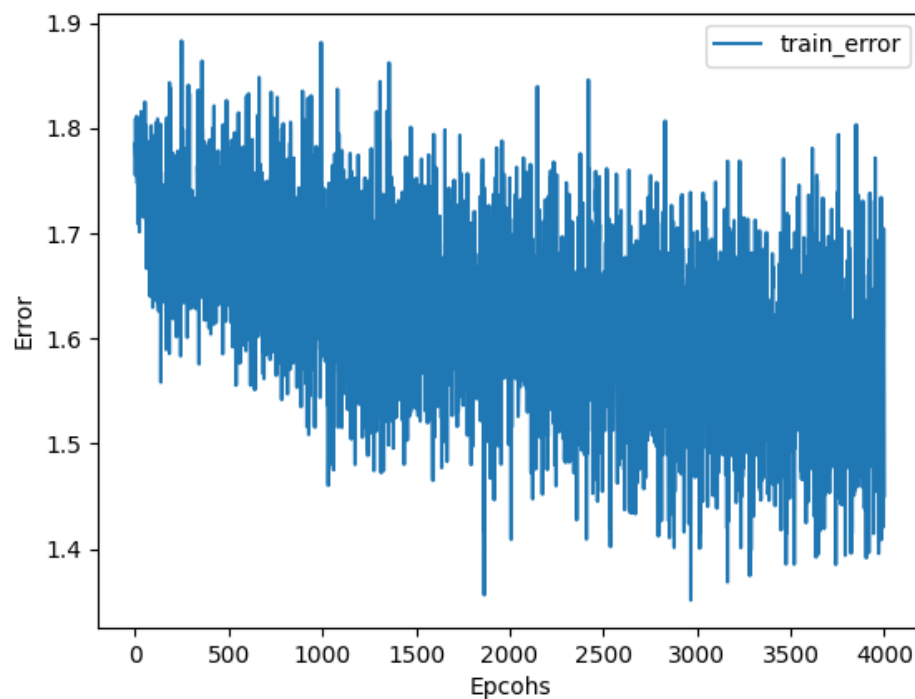
- Plotting a Minimum node degree for each graph



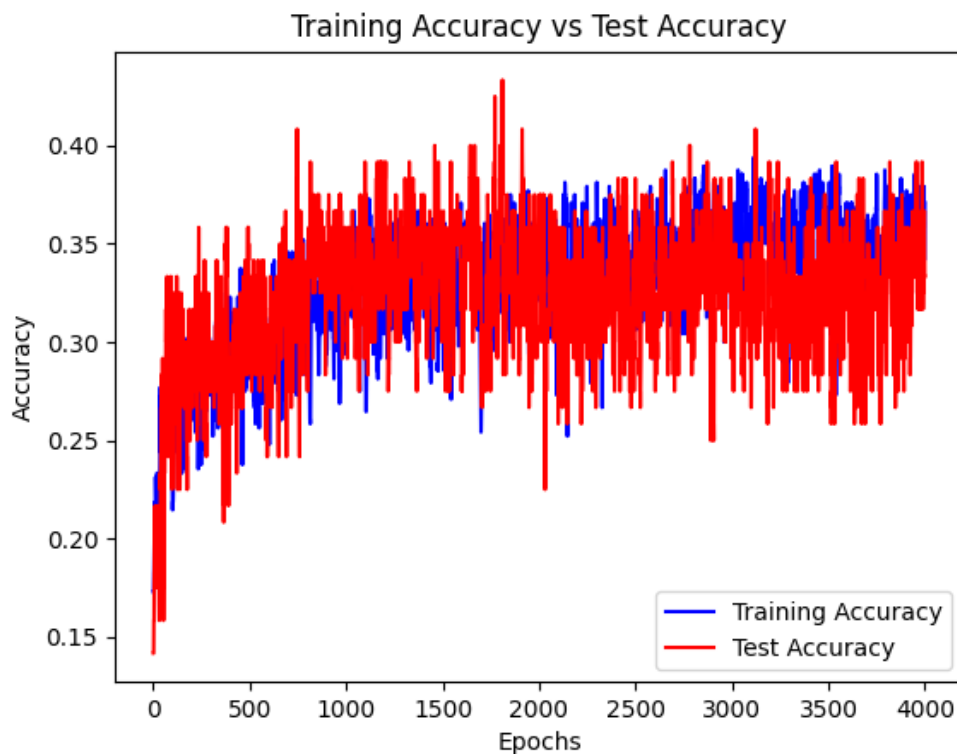
- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



- Plotting a Loss for each graph tells us how loss is varying over every epoch

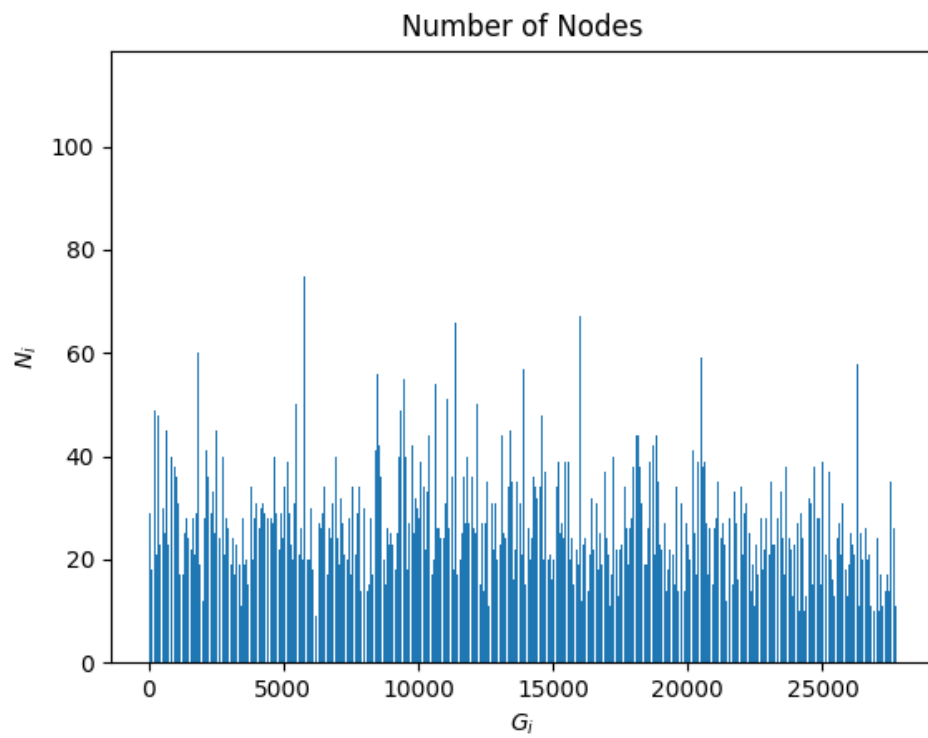


- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch

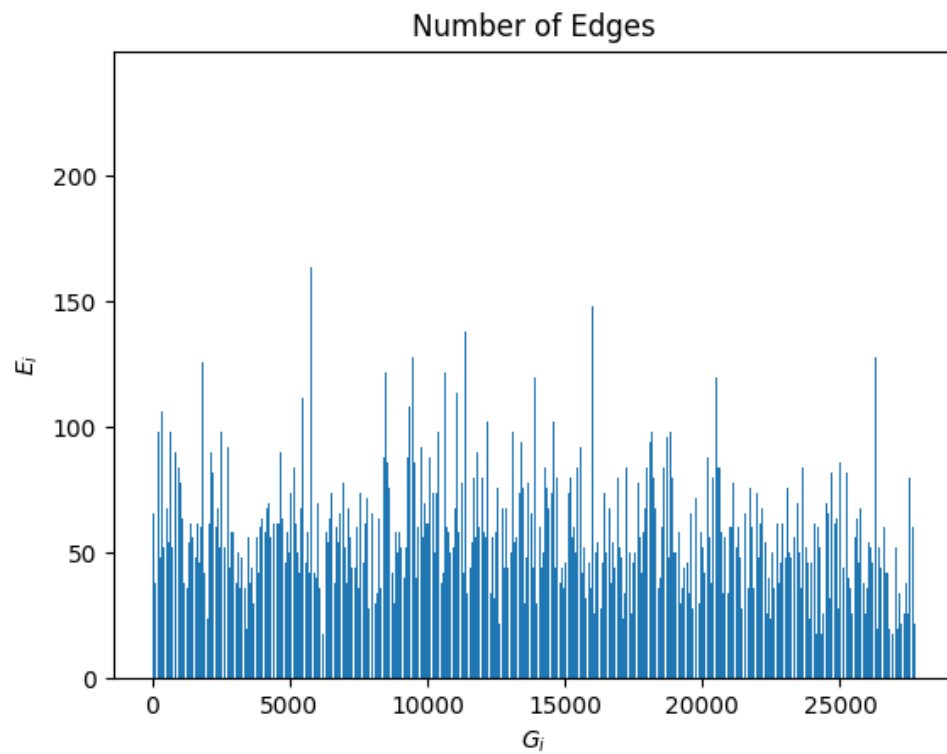


- **MCF-7 Dataset :**

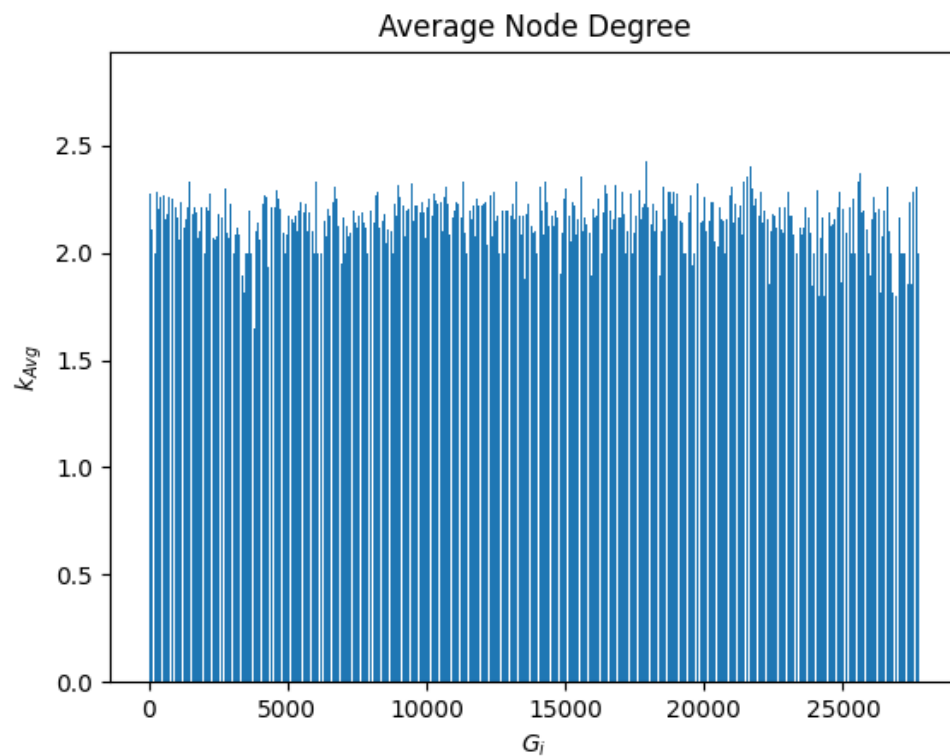
- The MCF-7 dataset is a synthetic dataset where its data is generated artificially for research purposes. The MCF-7 dataset cell line which is popularly used in human breast cancer cell line in studies on cancer.
- It is frequently used to research breast cancer and create machine learning algorithms for classification and predicting tasks.
  - Graphs represent Molecular structures(molecule)
  - Atoms represented as nodes
  - Chemical bonds represent edges
- This dataset is typically used for classification tasks, where the goal is to predict a binary or multi-class label for each molecule. The target variable in the dataset represents the label of each molecule, such as whether it is cancerous or not.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



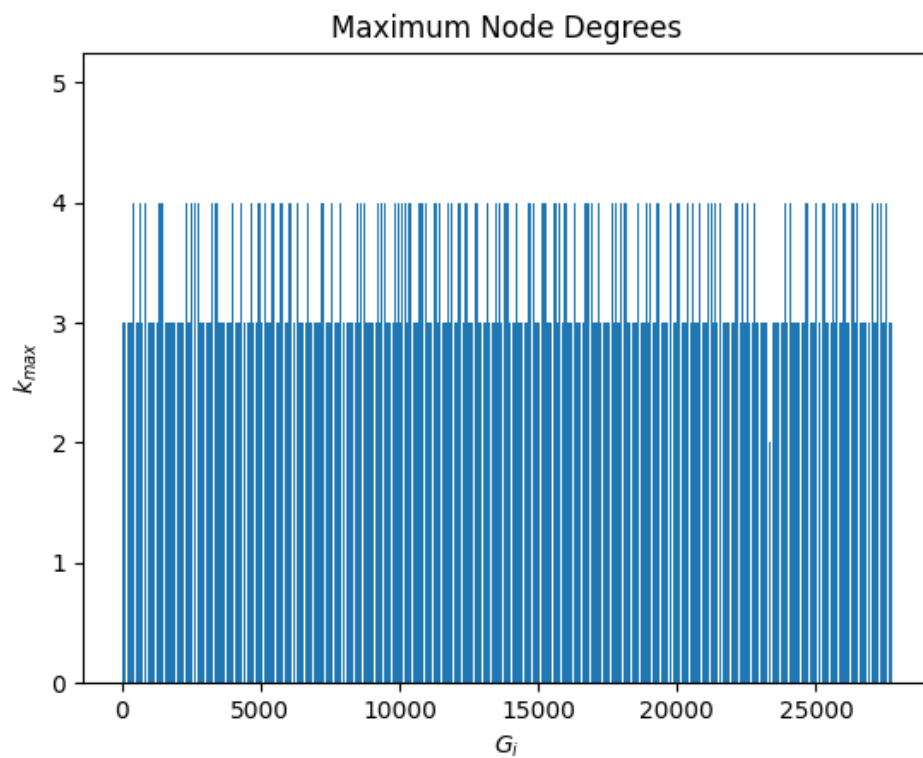
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



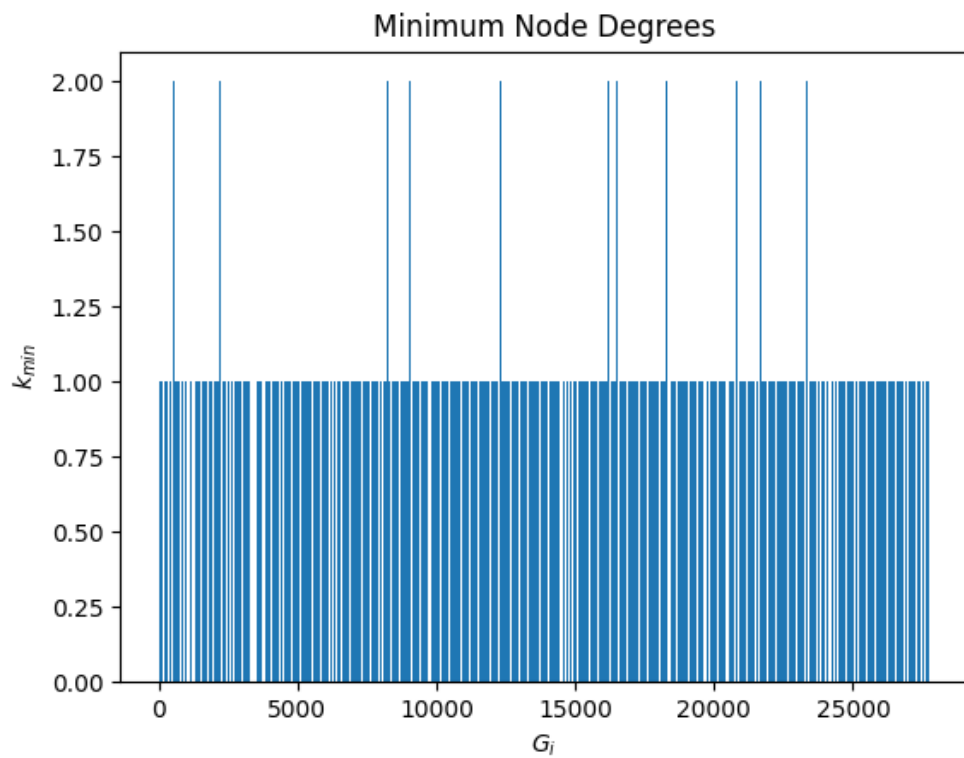
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



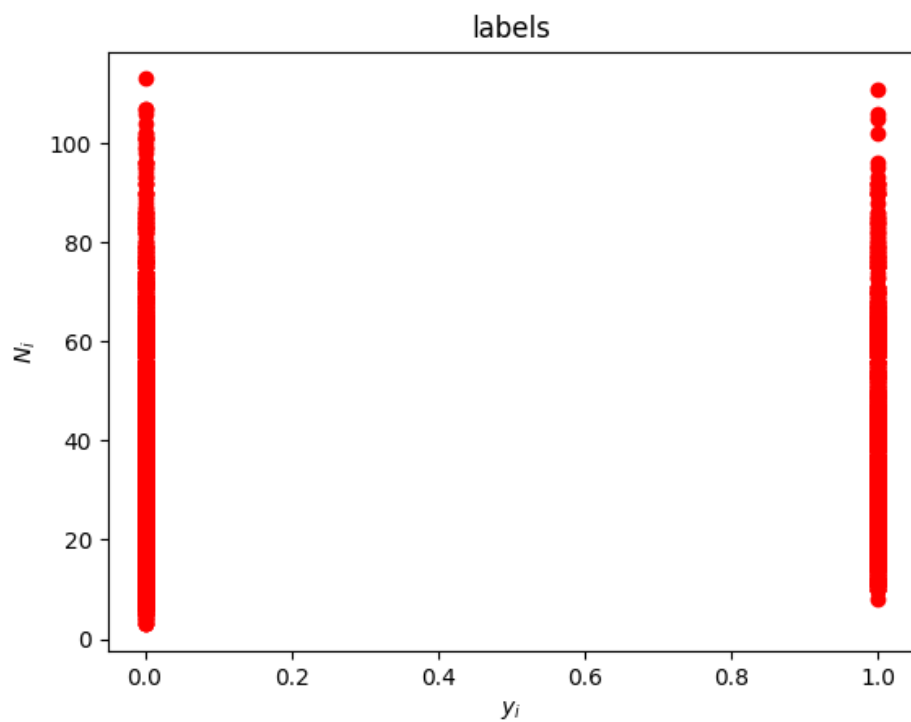
- Plotting a Maximum node degree for each graph



- Plotting a Minimum node degree for each graph\

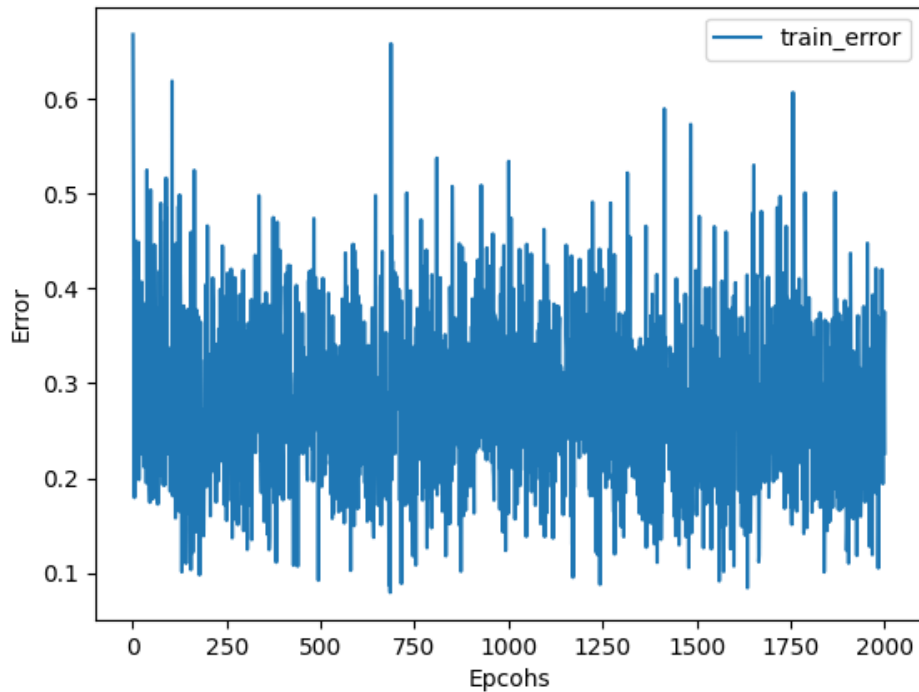


- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.

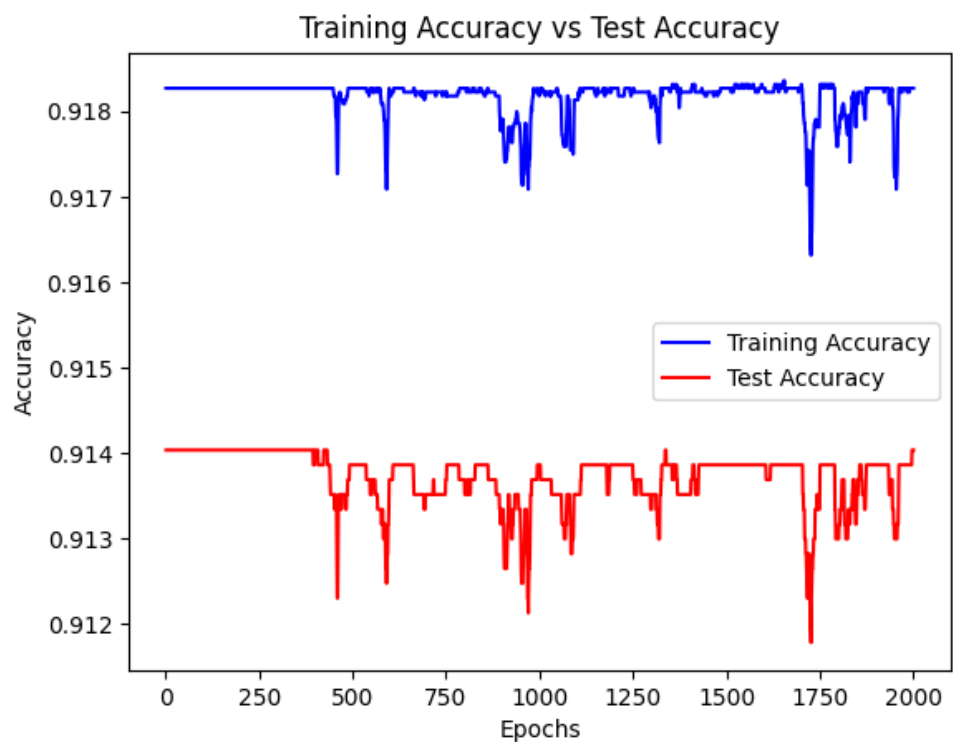




- Plotting a Loss for each graph tells us how loss is varying over every epoch

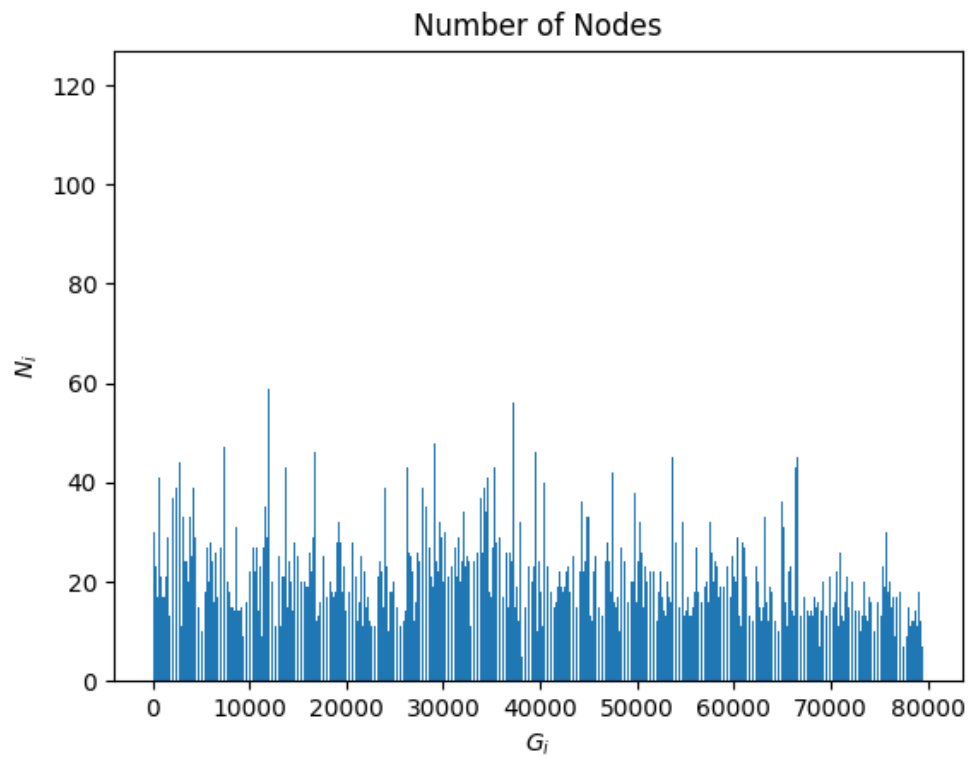


- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch

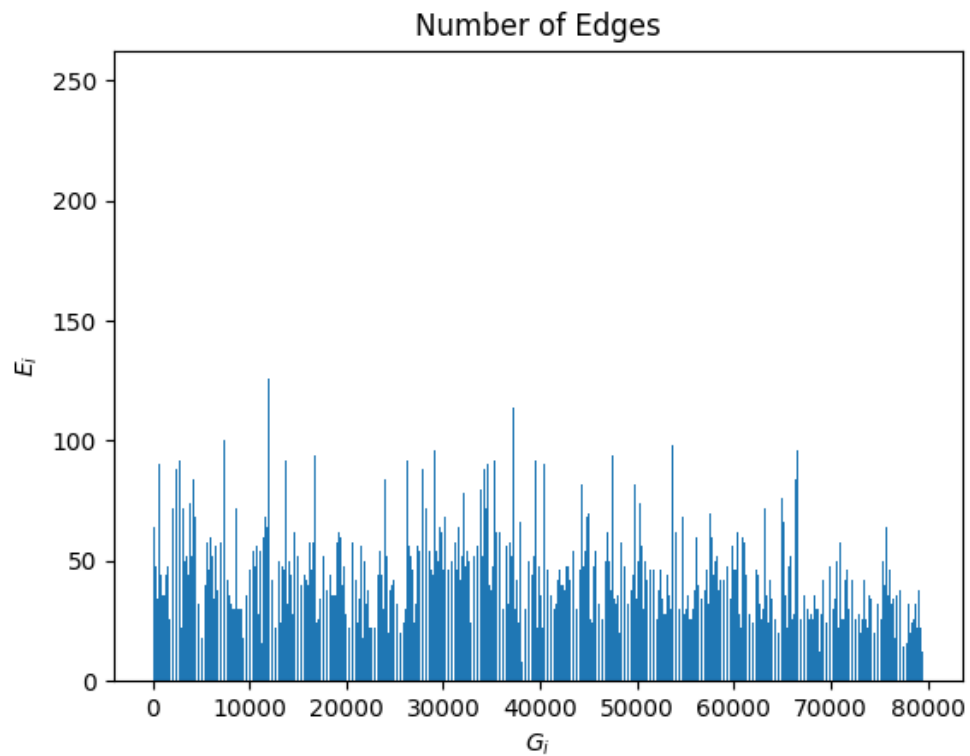


- **YEAST Dataset:**

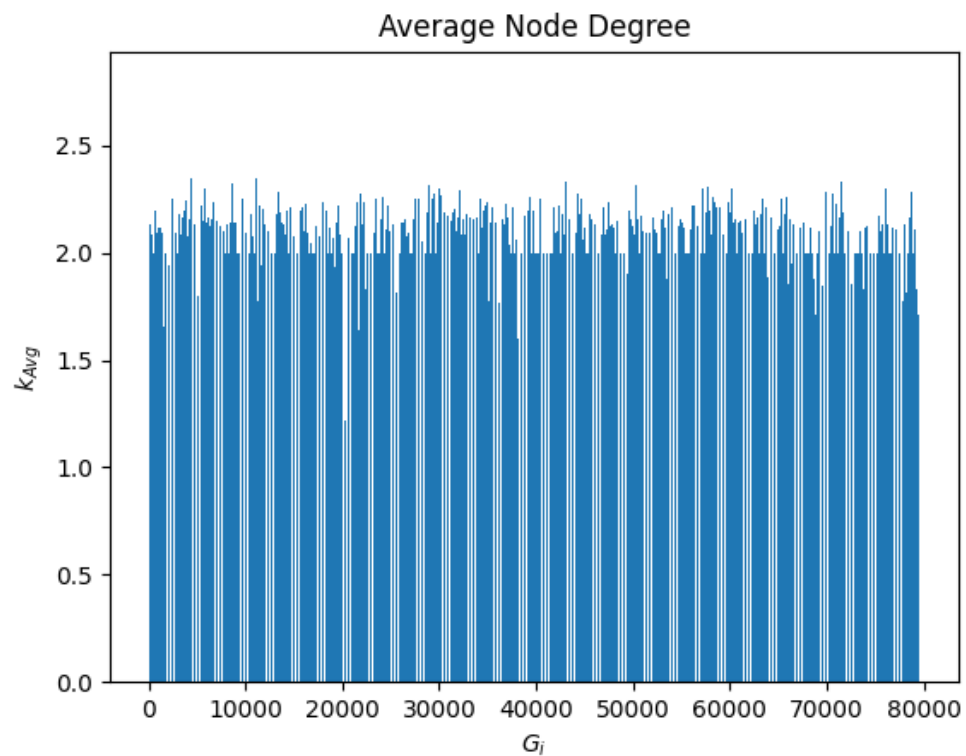
- The YEAST dataset is a protein-protein interaction (PPI) network of the *Saccharomyces cerevisiae* yeast species. It contains information about protein nodes and their interactions.
  - Graphs(79 601) represent Molecule
  - Nodes represent proteins
  - Edges represent interactions between them
  - 0 and 1 tell whether the molecule is yeast or not respectively.
- The dataset is derived from the Yeast Interactome Database (YIDB) and has been used in many studies related to protein function prediction, protein-protein interaction prediction, and graph representation learning.
- The YEAST dataset does not come with graph labels, as it is an unlabeled dataset. Here researchers generated them separately and then combined them with the YEAST dataset.
- Here they obtain graph labels from external sources and use some domain-specific methods to generate graph-level labels based on the protein nodes and their interactions in the PPI network. These graph labels can then be used for tasks such as graph classification or graph regression, depending on your specific use case.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



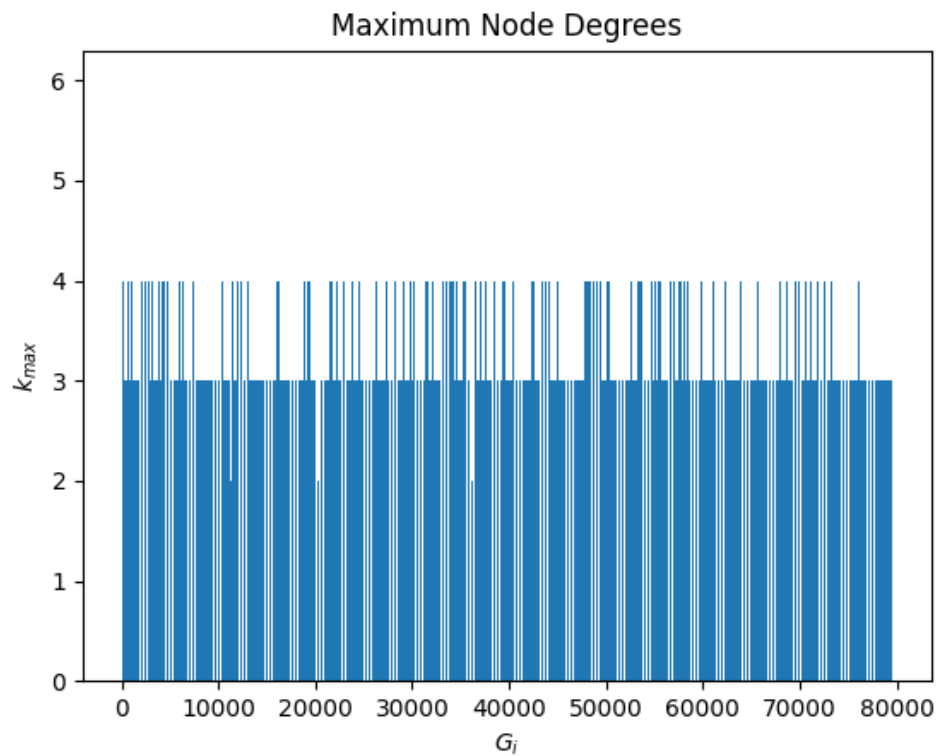
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



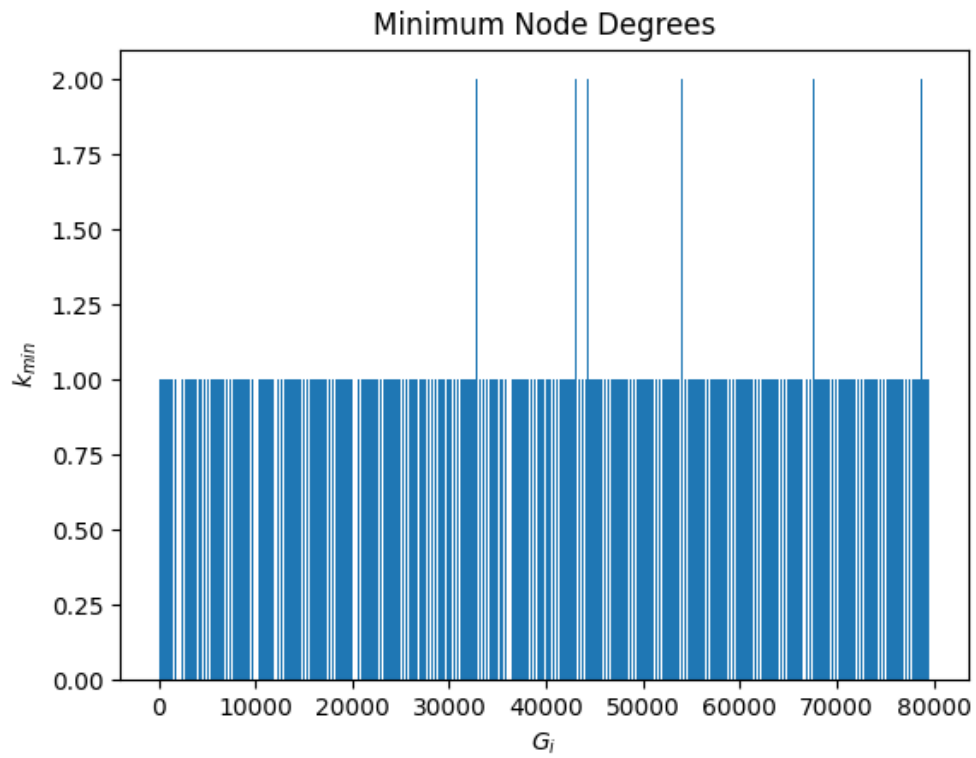
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



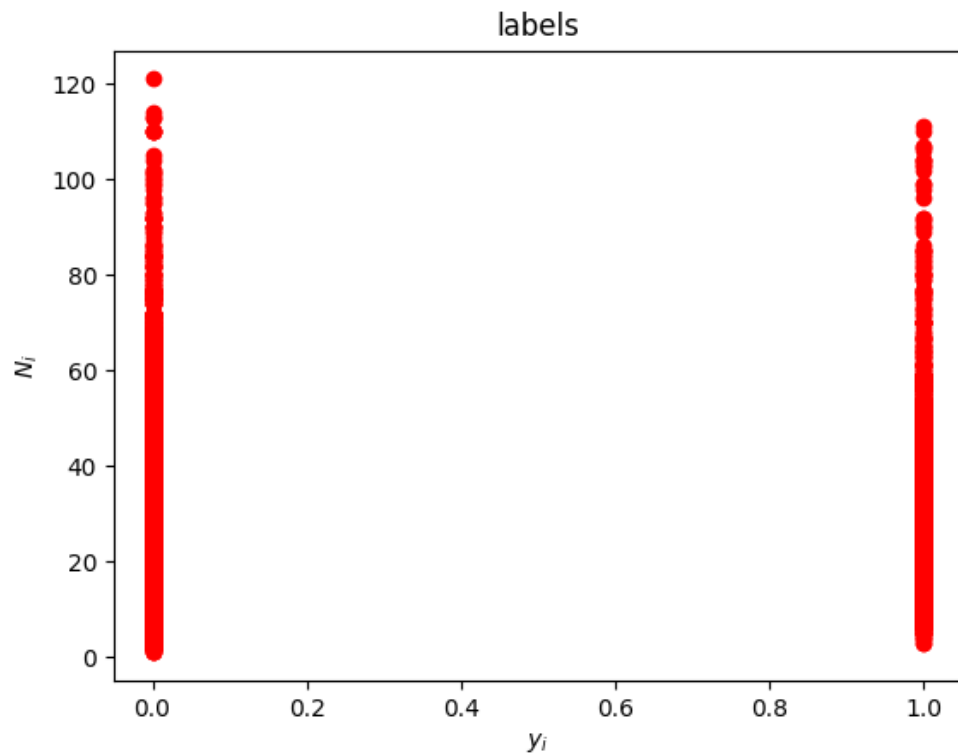
- Plotting a Maximum node degree for each graph



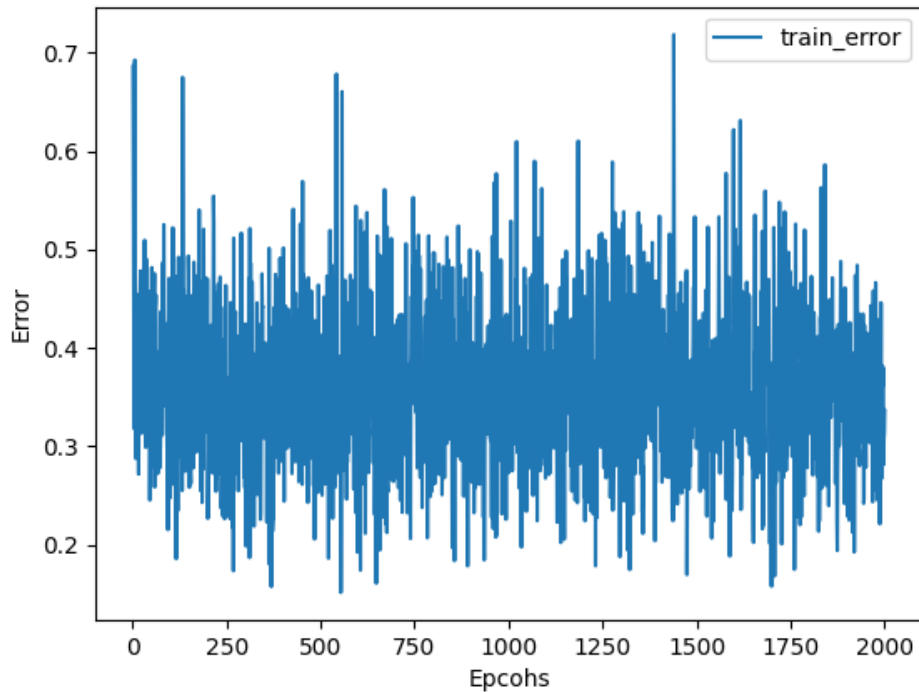
- Plotting a Minimum node degree for each graph



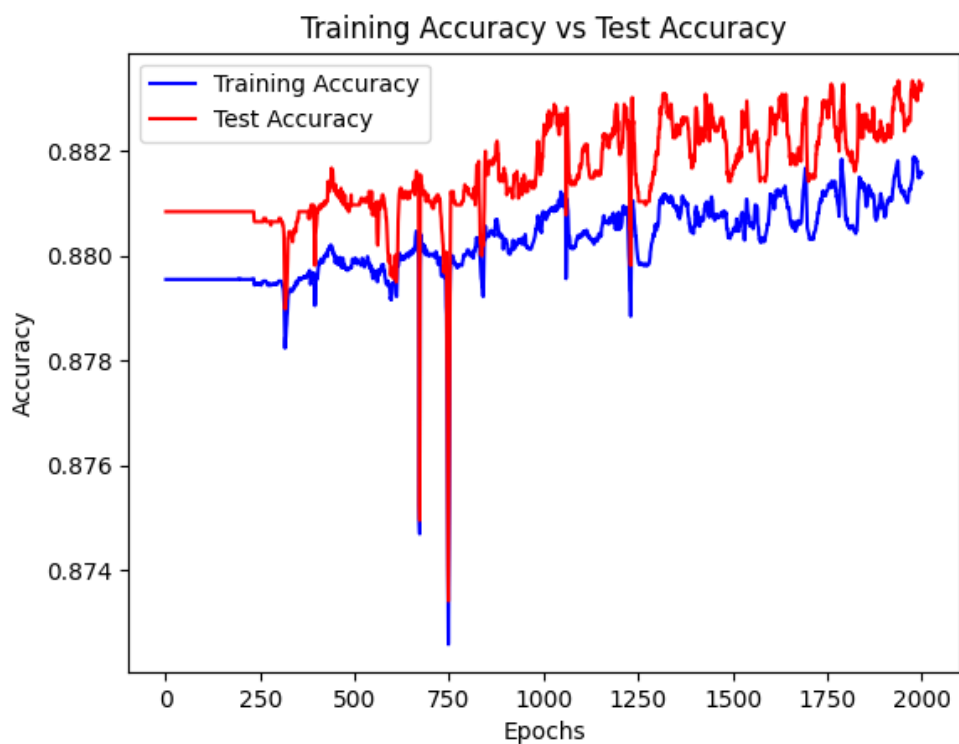
- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



- Plotting a Loss for each graph tells us how loss is varying over every epoch

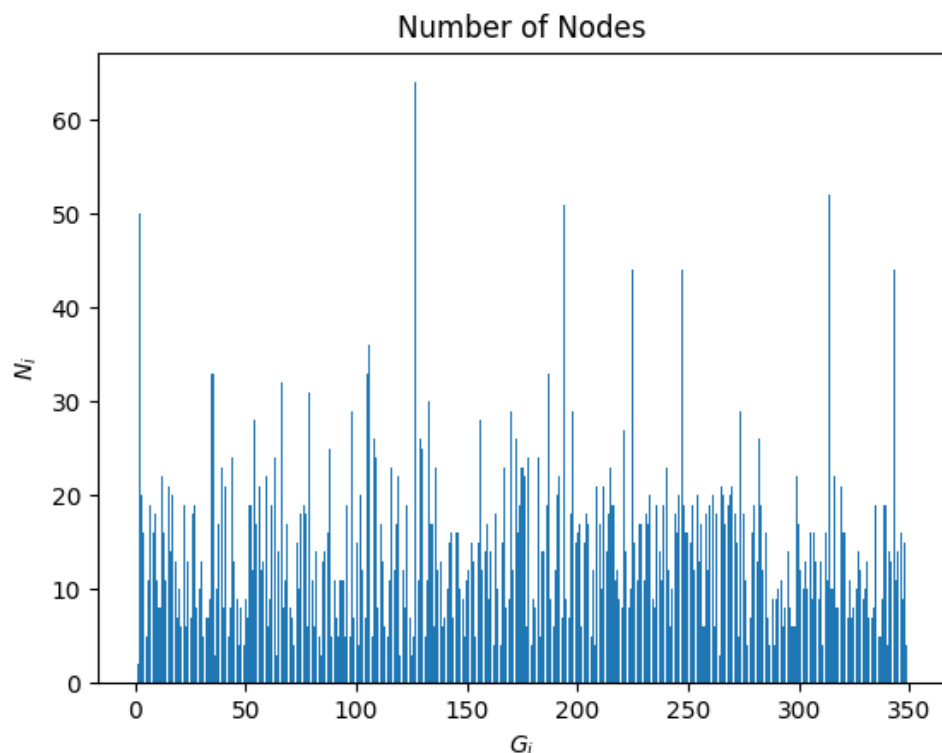


- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch

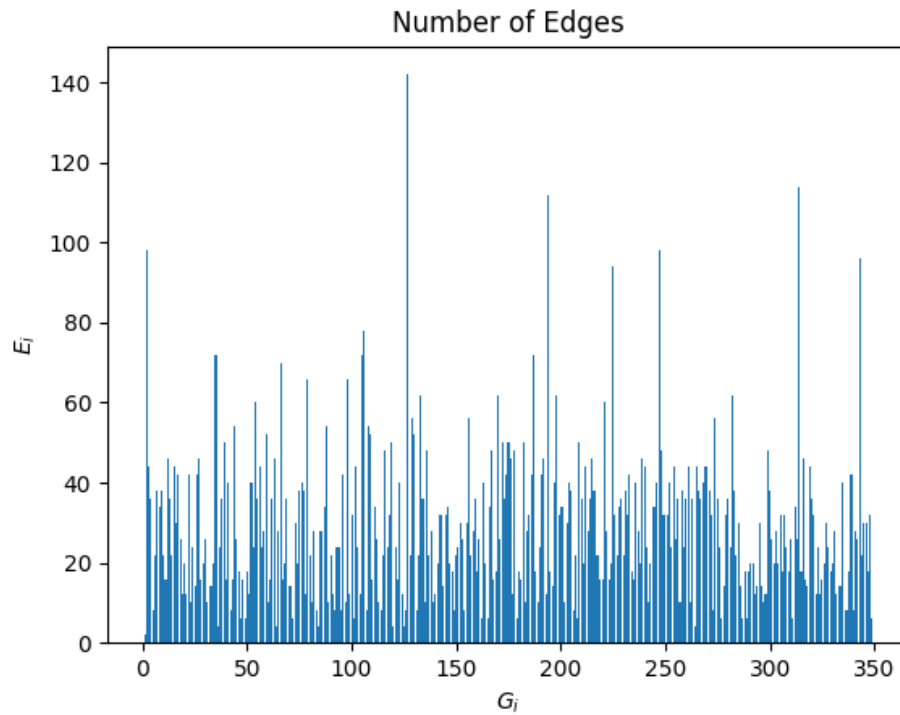


- **PTC\_FM Dataset:**

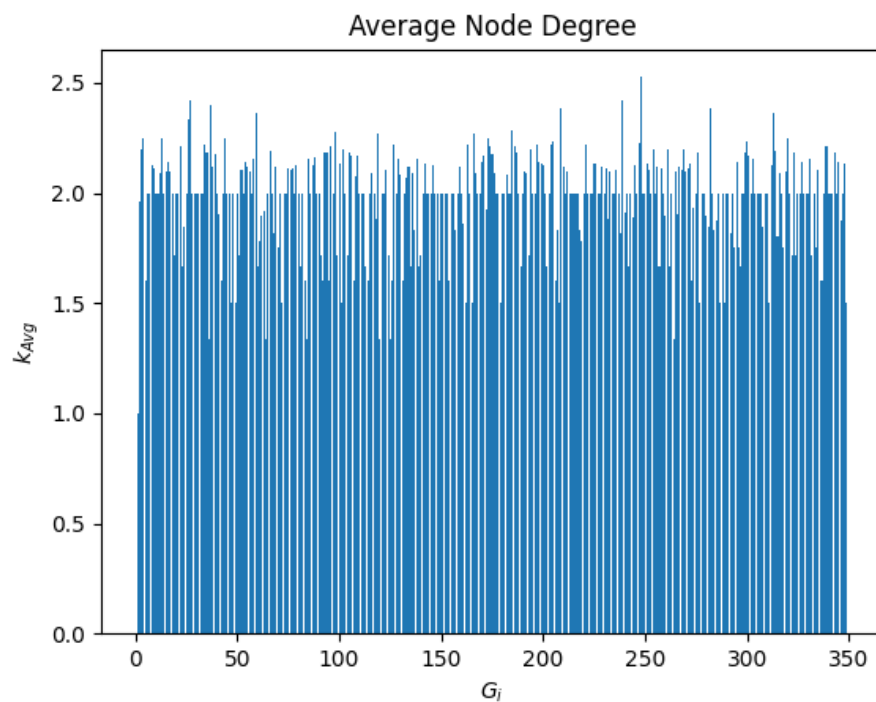
- The PTC\_FM dataset is a collection of 349 graphs, which is a popular library for deep learning on graph-structured data and it contains graphs that represent protein contact maps derived from Free Modeling (FM) experiments. Contact maps are used in computational protein modelling to represent the spatial proximity between pairs of amino acids in a protein structure.
  - Graph represents a protein contact map
  - Nodes represent amino acids
  - Edges Represent contact between pair of amino acids
- The PTC\_FM dataset is used for tasks such as contact prediction, protein folding prediction, or protein structure prediction, The goal is to predict the three-dimensional structure of proteins.
- If the structure is correct then 1 or then 0.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.

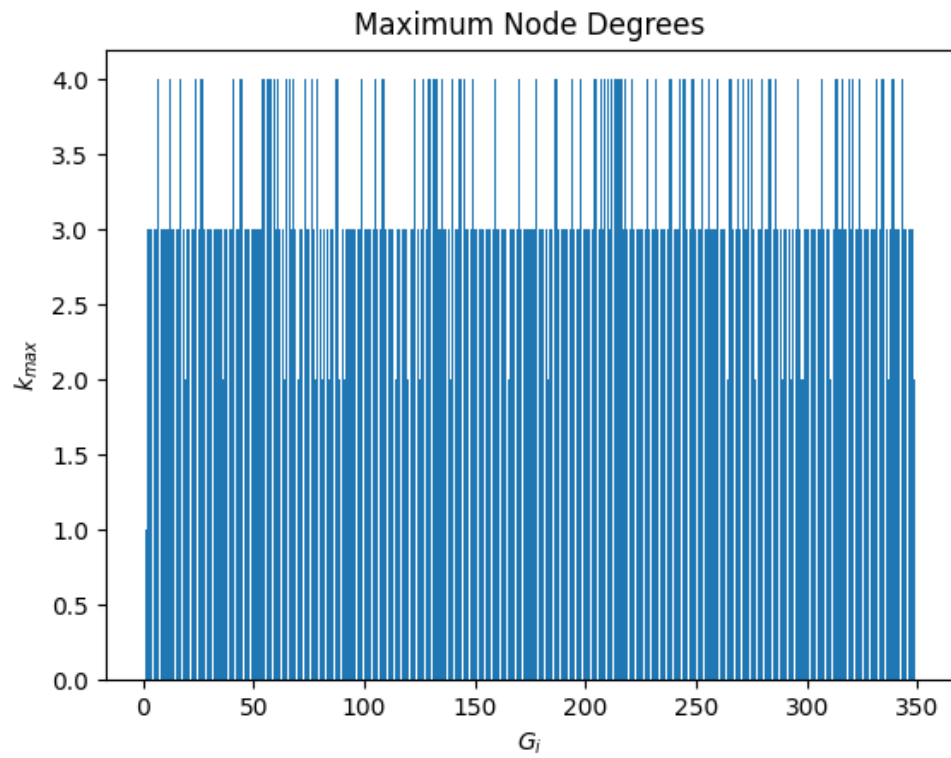


- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.

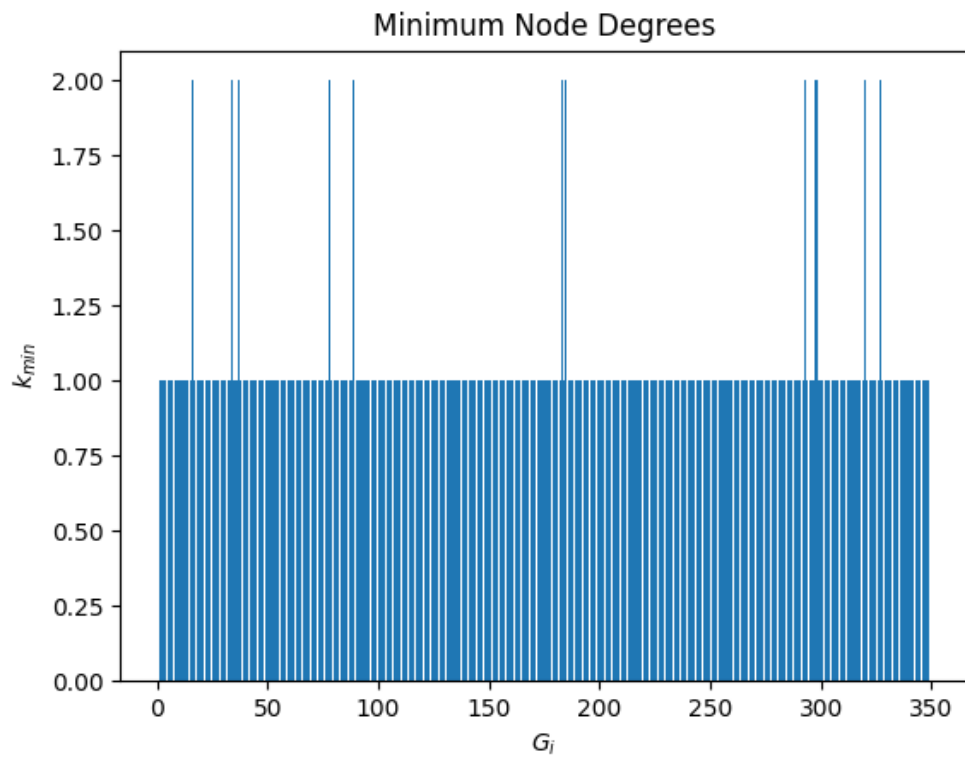




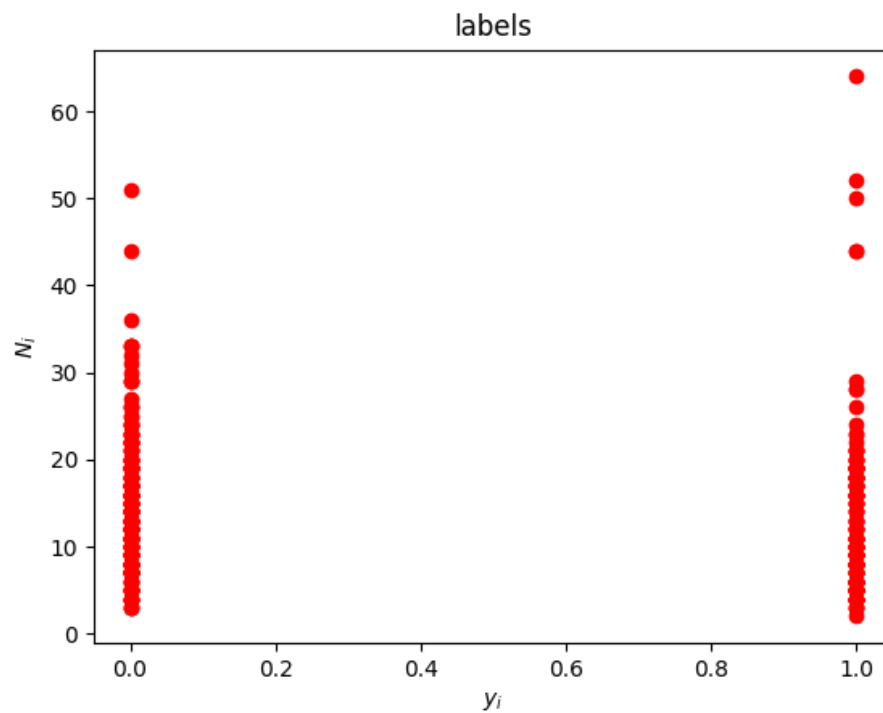
- Plotting a Maximum node degree for each graph



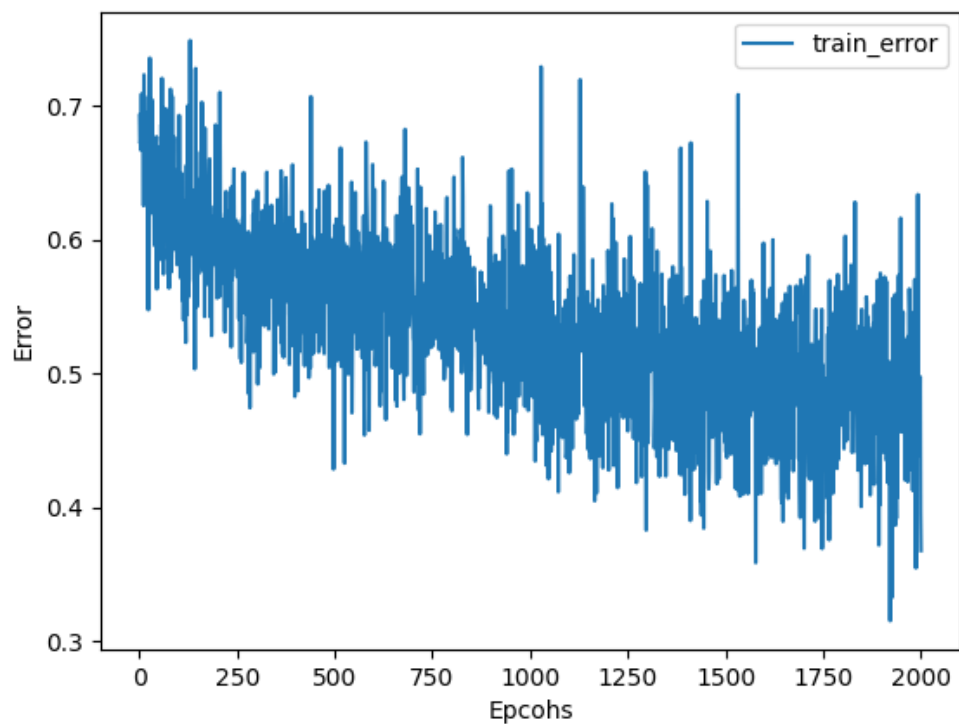
- Plotting a Minimum node degree for each graph



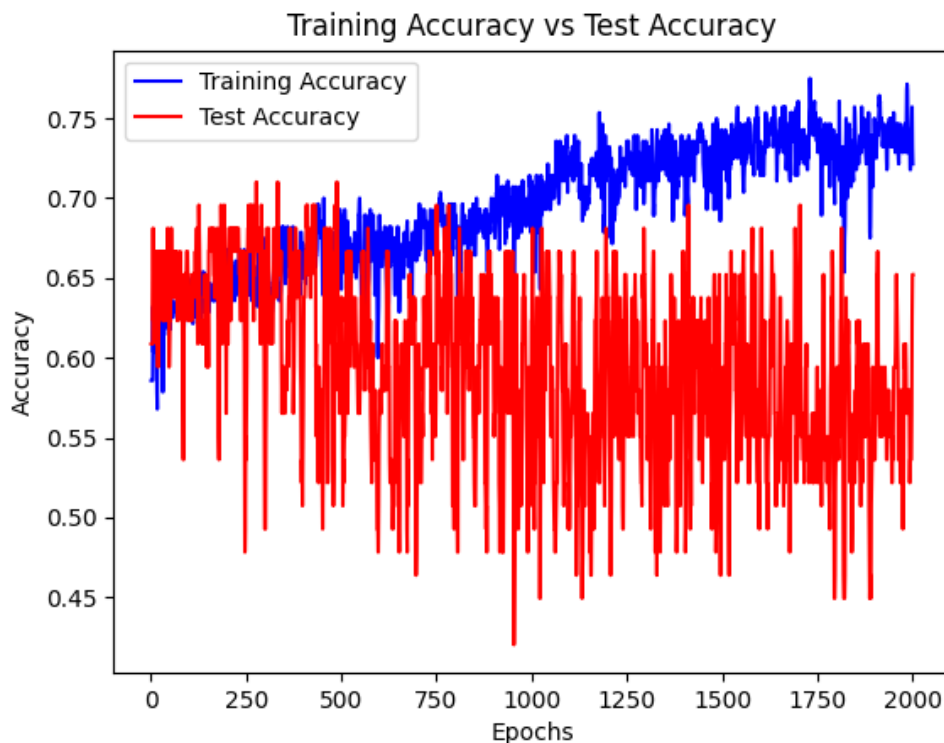
- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



- Plotting a Loss for each graph tells us how loss is varying over every epoch



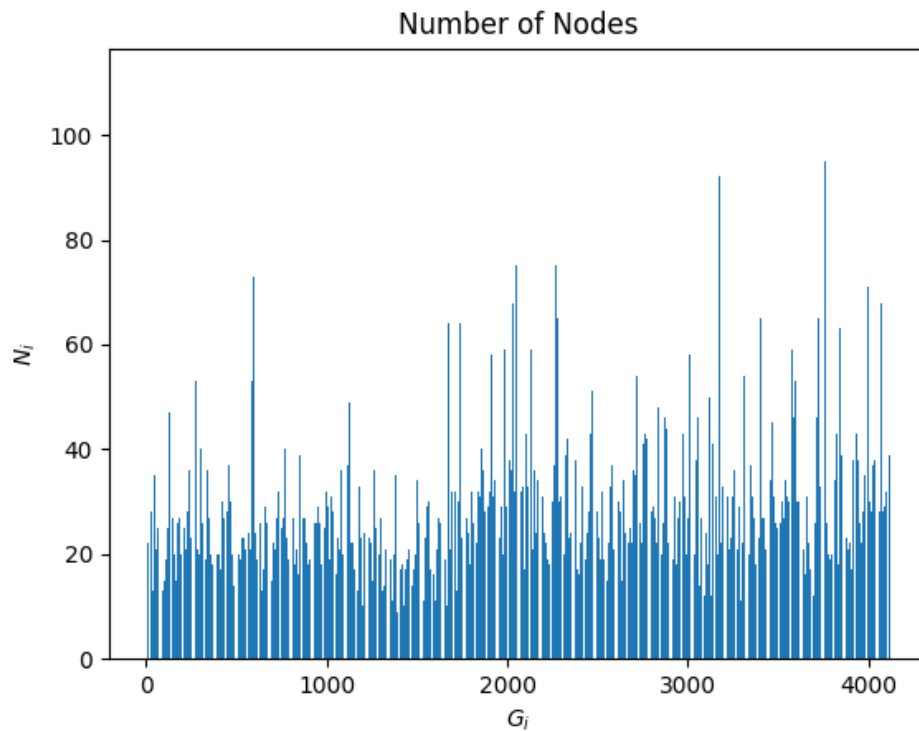
- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch



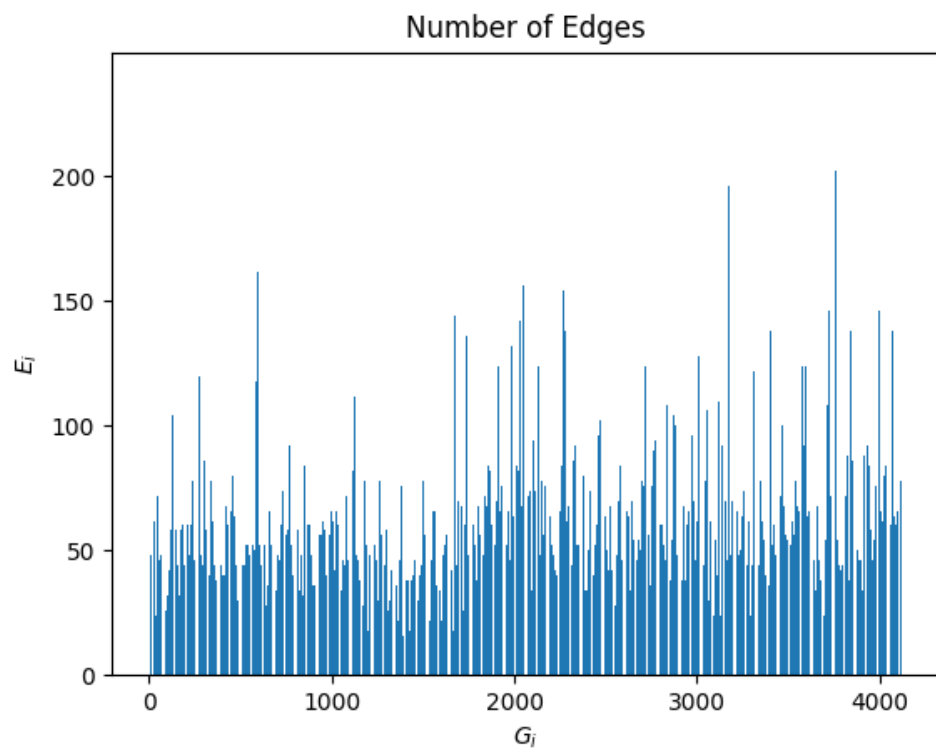
- **NCI109 Dataset:**

- The NCI109 dataset is a collection of 4127 graphs which are used for graph-based machine learning and it is derived from the National Cancer Institute (NCI) Anticancer Screen, which is a publicly available database of biological activity of compounds screened against various cancer cell lines. The NCI109 dataset has been pre processed and transformed into a graph-based format suitable for machine learning tasks.
  - Graphs represent Chemical compound
  - Nodes represent atoms
  - Edges represent chemical bonds
- The graphs present in this dataset are fully connected such that there is an edge between every pair of nodes in the graph and the class distribution is usually imbalanced, with a small number of active compounds compared to inactive compounds.

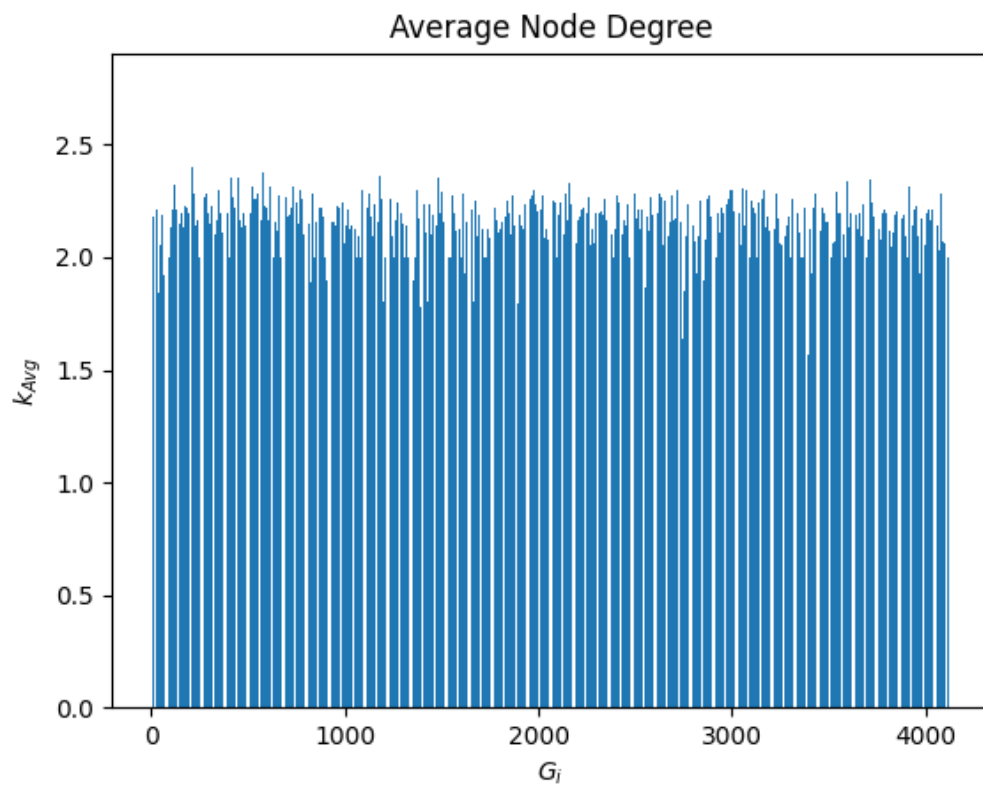
- Each compound is labelled as either active (1) or inactive (0) based on its biological activity against cancer cell lines.
- **Analysis:**
  - Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



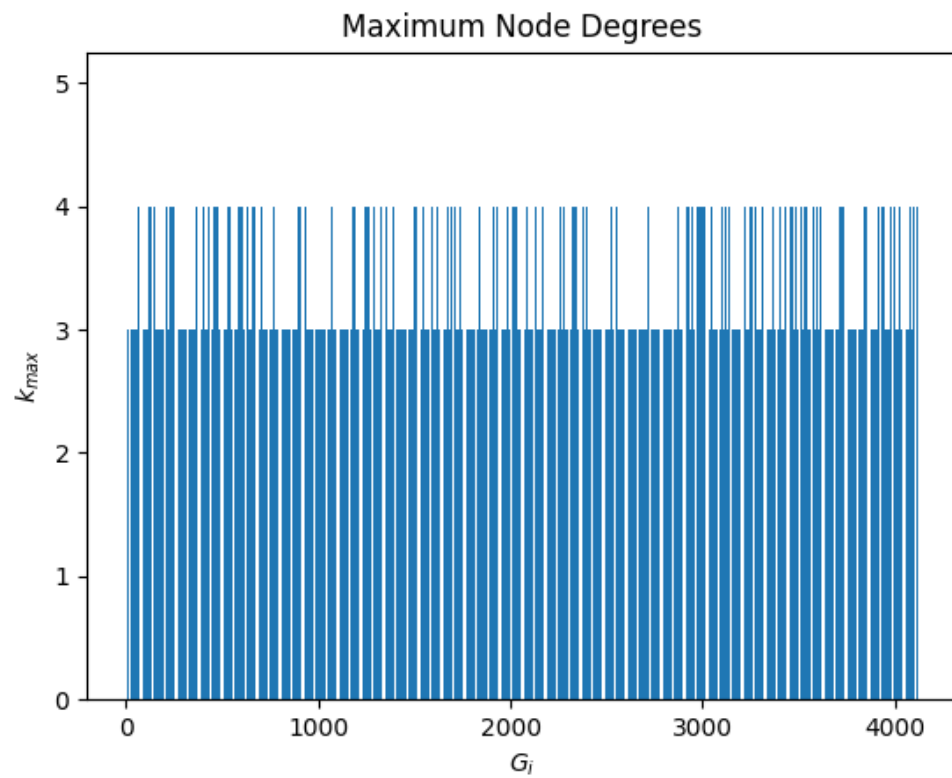
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



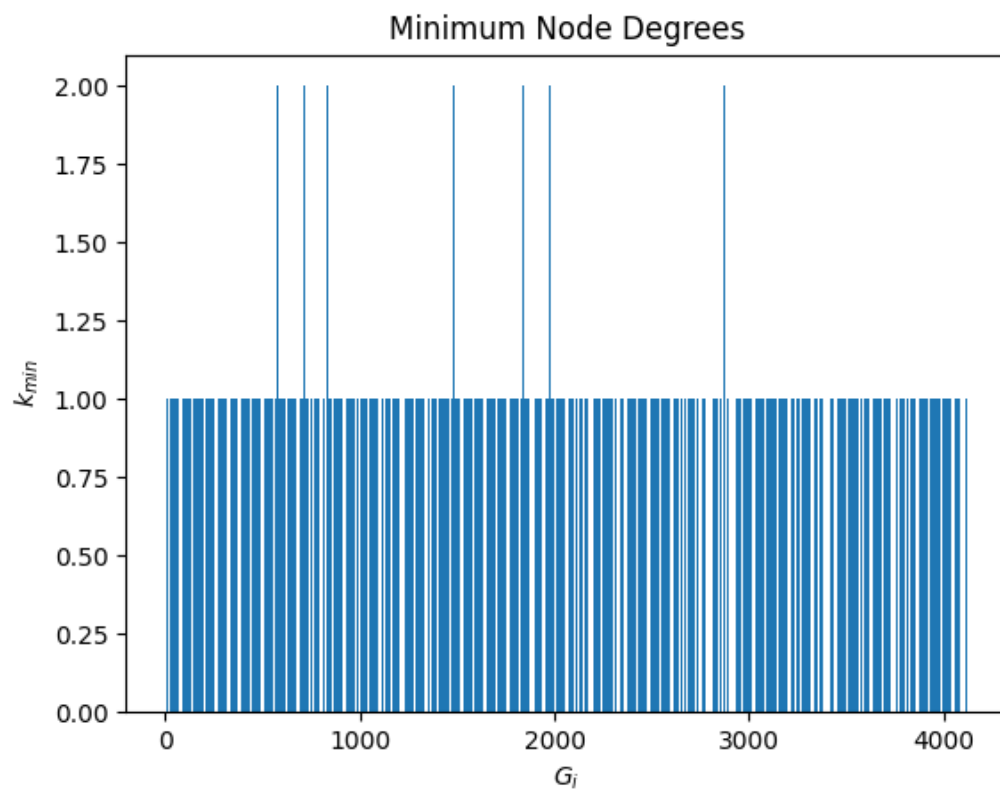
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



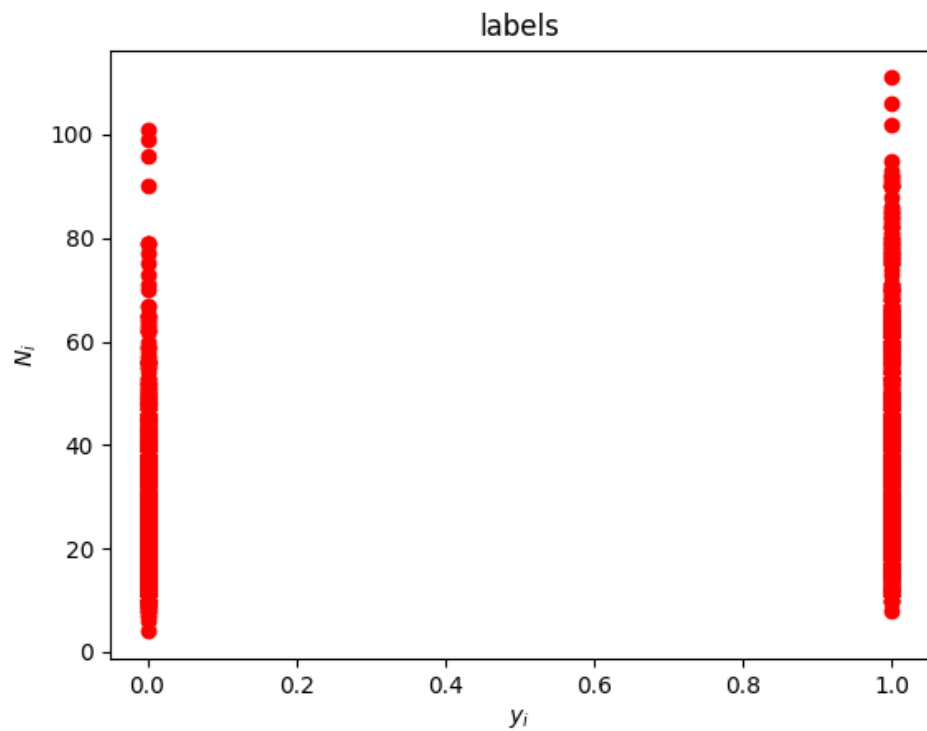
- Plotting a Maximum node degree for each graph



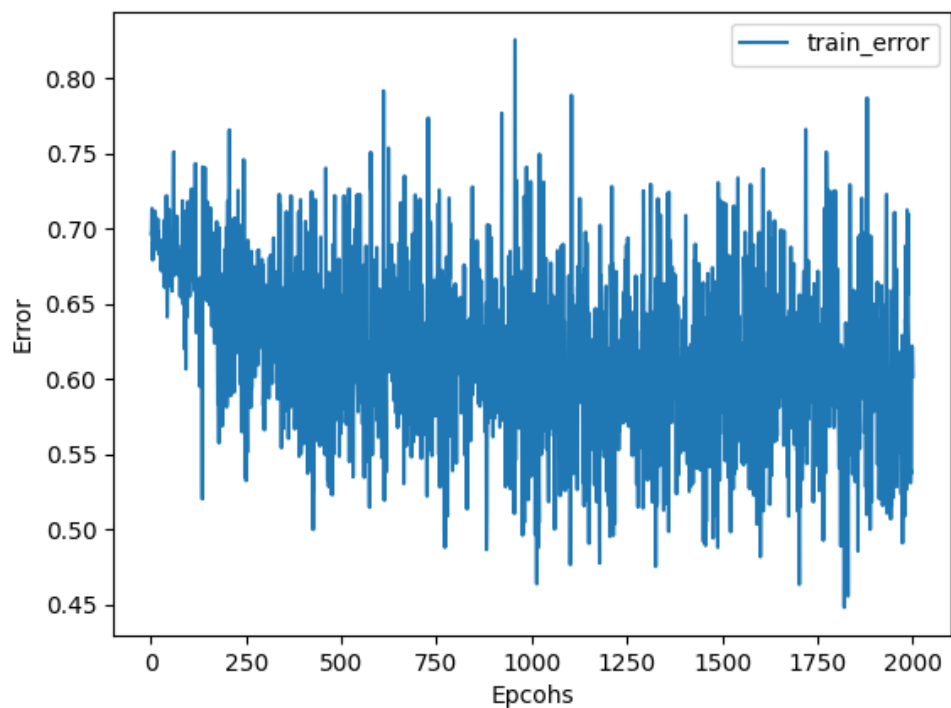
- Plotting a Minimum node degree for each graph



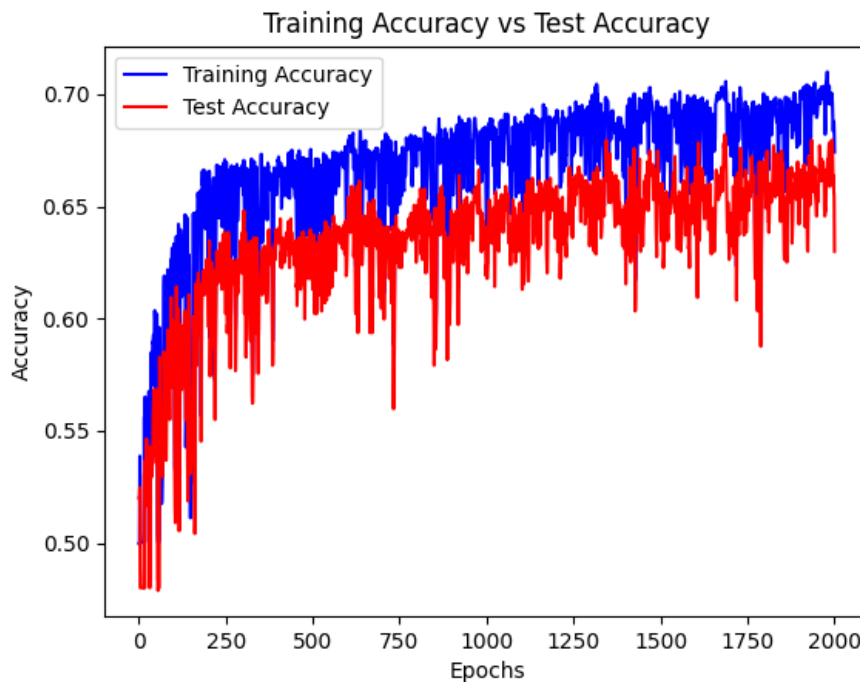
- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.



- Plotting a Loss for each graph tells us how loss is varying over every epoch



- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch



- **Methodology:**

Graph neural networks are a class of neural networks that can operate on different types of Networks and are particularly useful for graph classification problems. Here we used TUD datasets which are extracted from the same background. So to check the performance and to analyse the relation among different types of graphs like random networks, scale free networks, star graphs, complete graphs, cycle graphs, path graphs and wheel graphs which are of different backgrounds. Here real world problems are of different backgrounds which have different graph properties. However real world graphs a bit complex and noisy. Here we have generated a dataset (Collection of these localised and delocalized Networks) for graph classification.

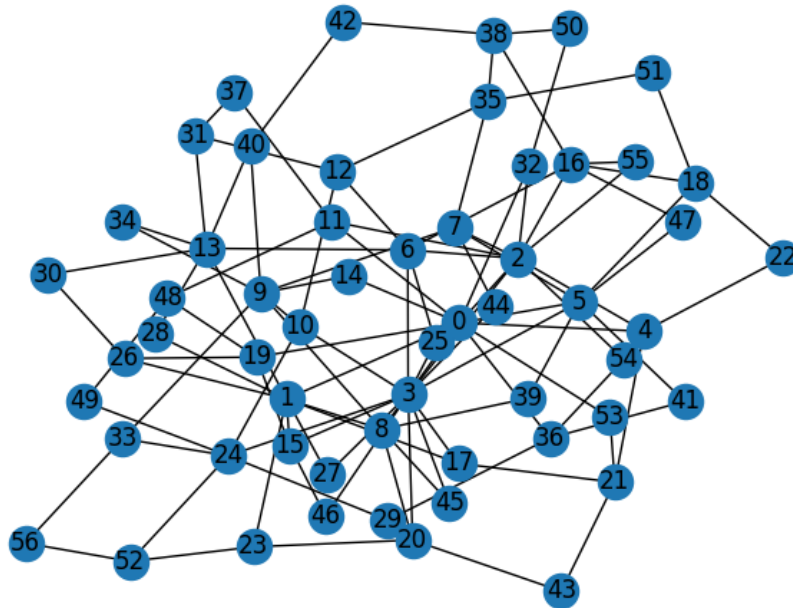
- **Data collection:**

- We generated some barabasi albert graphs of ``n`` nodes (generated randomly between 30 to 100) is grown by attaching new nodes each with ``m`` Edges (chosen randomly between 2 and 3) that are preferentially attached to existing nodes with high degree and extracted adjacency

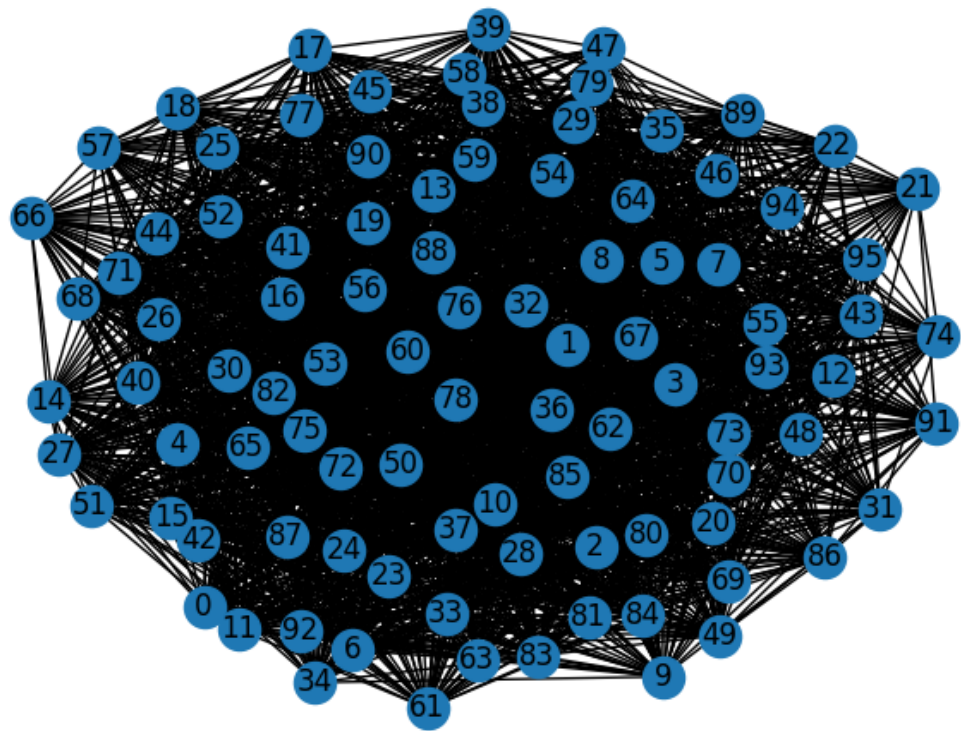


matrix for each graph and created edge\_indices, generated feature matrices for each graph based size of the graph and assign labels.

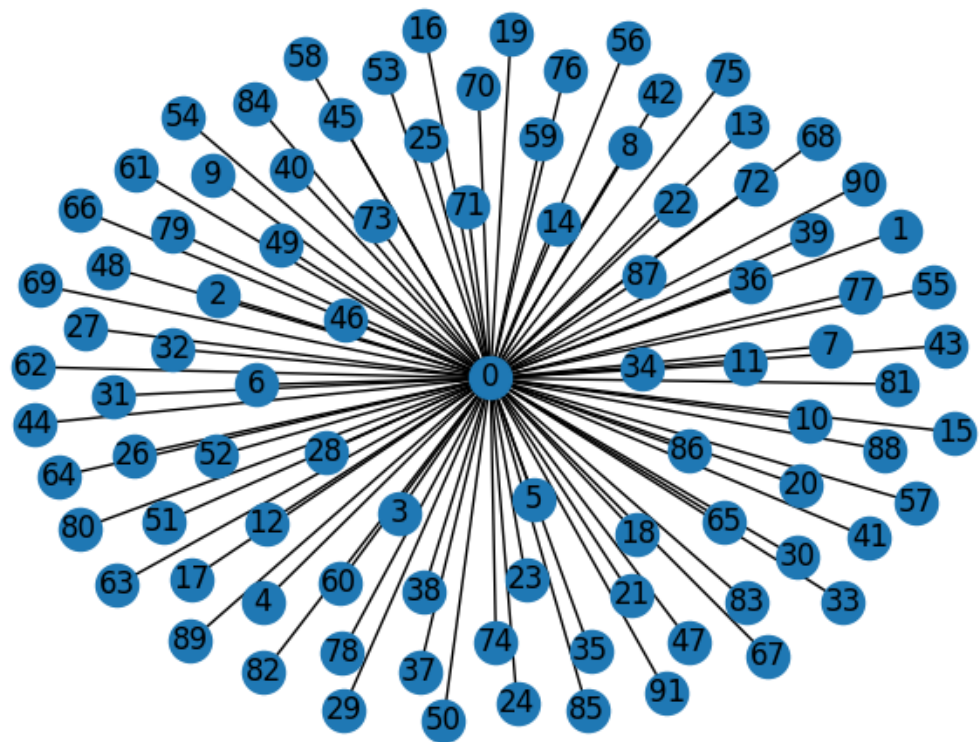
- We generate the same for Erdos renyi graphs(The  $G(n,p)$  model chooses each of the possible edges with probability  $p(0.5 \text{ to } 0.8)$ ) and other random graphs and create edge\_indices, feature matrices and assign labels.
  - Barabasi albert graphs : Label is `` 1 ``



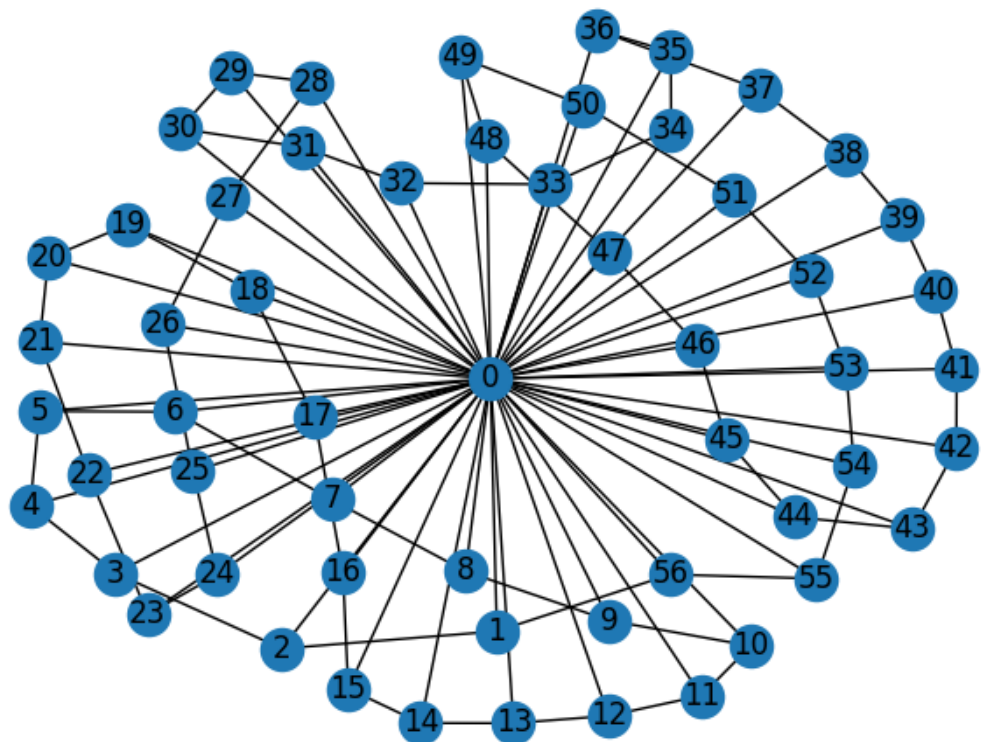
- Erdos renyi graphs : Label is `` 0 ``



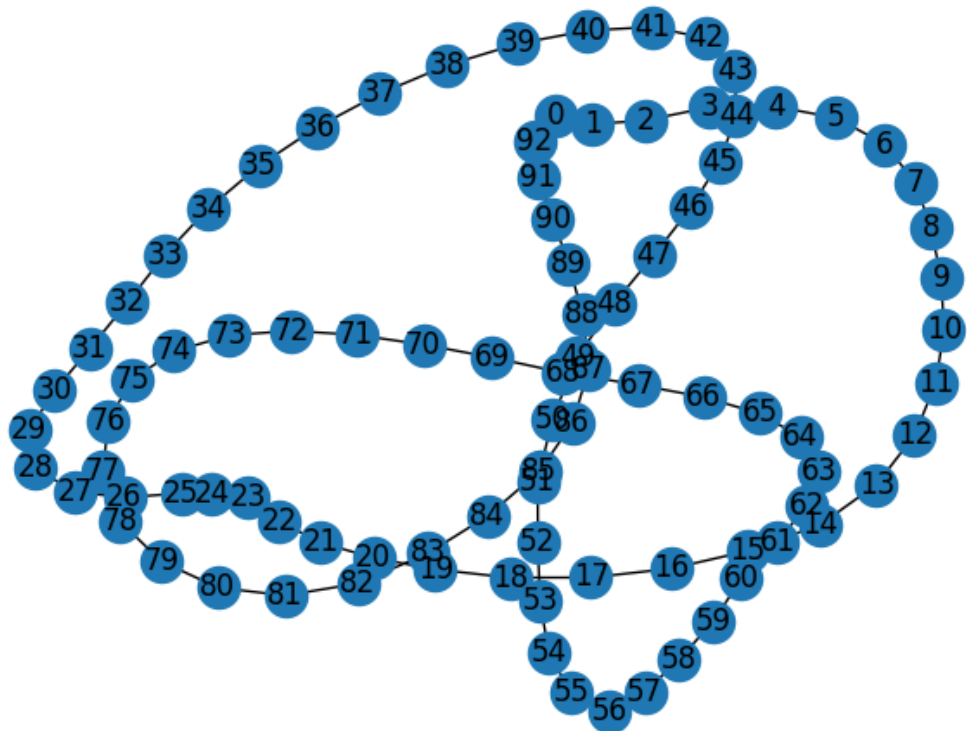
- Star graphs :Label is `` 1 ``



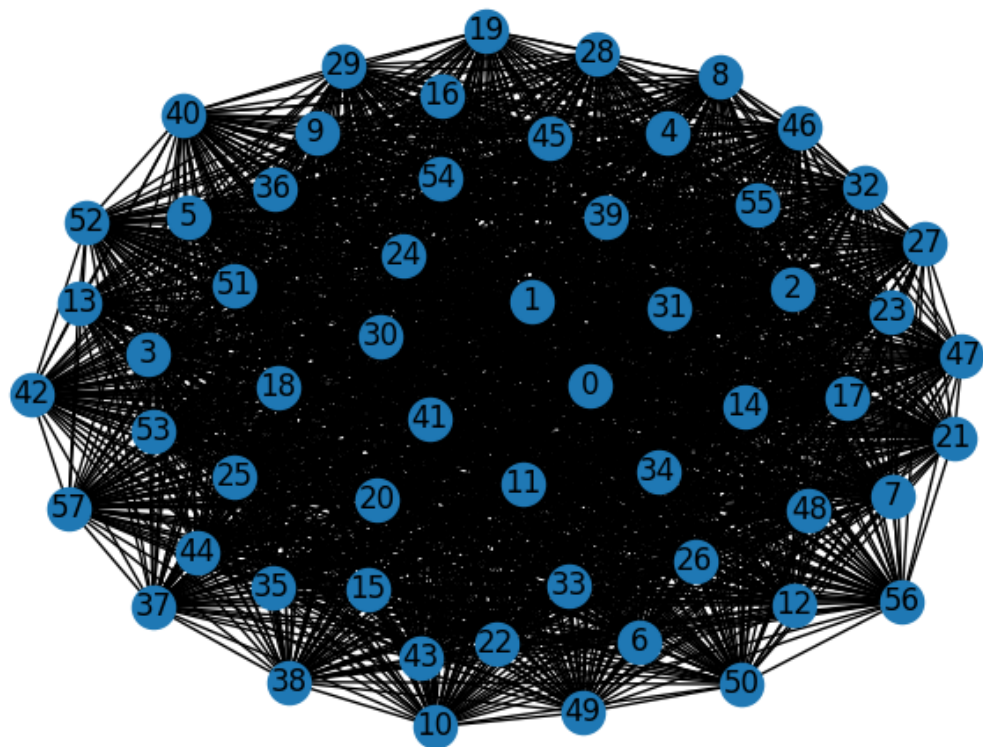
- Wheel graphs :Label is `` 1 ``



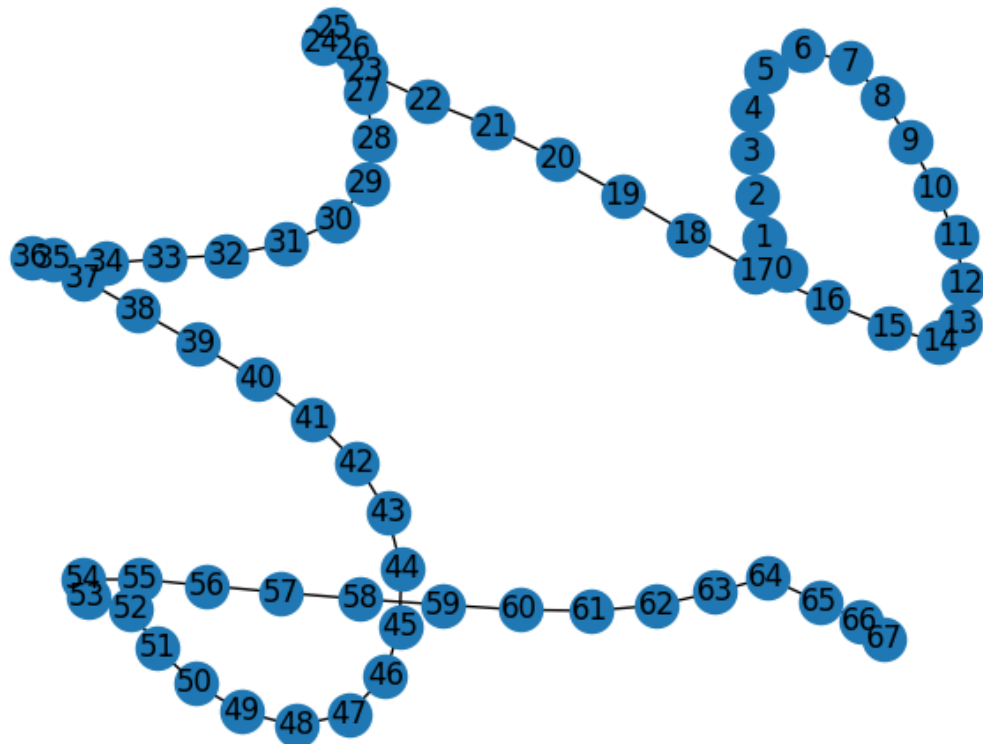
- Cycle graphs : Label is `` 0 ``



- Complete graphs : Label is `` 0 ``



- Path graphs : Label is `` 1 ``



- Data preprocessing :

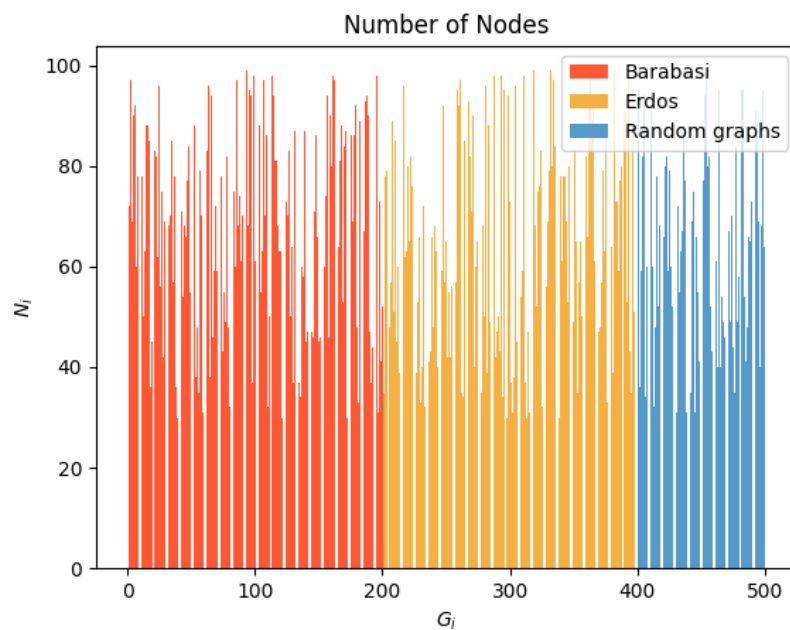
The graph data we have extracted is processed using torch geometric data which is imported from the torch geometric python library consists of Data class and Dataset class where our graph data is converted to torch geometric class so that the processed dataset can be used for training and testing sets.

Steps for Data preprocessing :

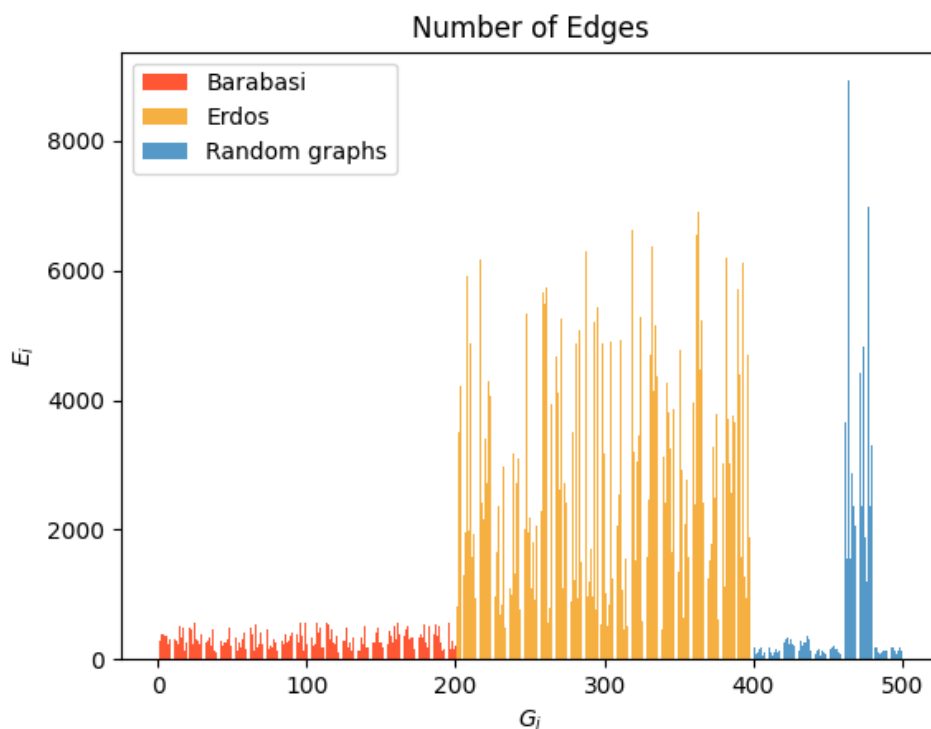
- **Data Loading** : Edge-indices, feature matrices and labels are passed into the **Data** object which is provided by torch geometric library.
- **Cleaning the Data** : The data may contain missing values, outliers, or other inconsistencies. Cleaning the data involves identifying and handling these issues.
- **Feature Normalisation**: The features of the nodes in the graph may have different scales, which can negatively impact model performance. Torch Geometric provides functions to perform normalisation
- **Graph Transformation**: Here some graph models in Torch Geometric require the graph to be transformed into a specific format. Torch Geometric provides functions to perform these transformations.
- **Feature Extraction**: Here graph features may not be in a suitable format for the model. Feature extraction is the process of converting the graph features into a different format that is more compatible with the model. Torch Geometric provides various functions to perform feature extraction.
- Finally **Data** objects are stored in the List (Dataset) and This dataset is shuffled splitted into training set and testing set and these two are passed into **DataLoader** class which is imported from the torch geometric library.
- DataLoader loads a dataset and divide training data and training data into batches and these batches are passed one by one to the model.
- Finally we can perform **graph classification** on these processed data very easily.

- **Analysis:**

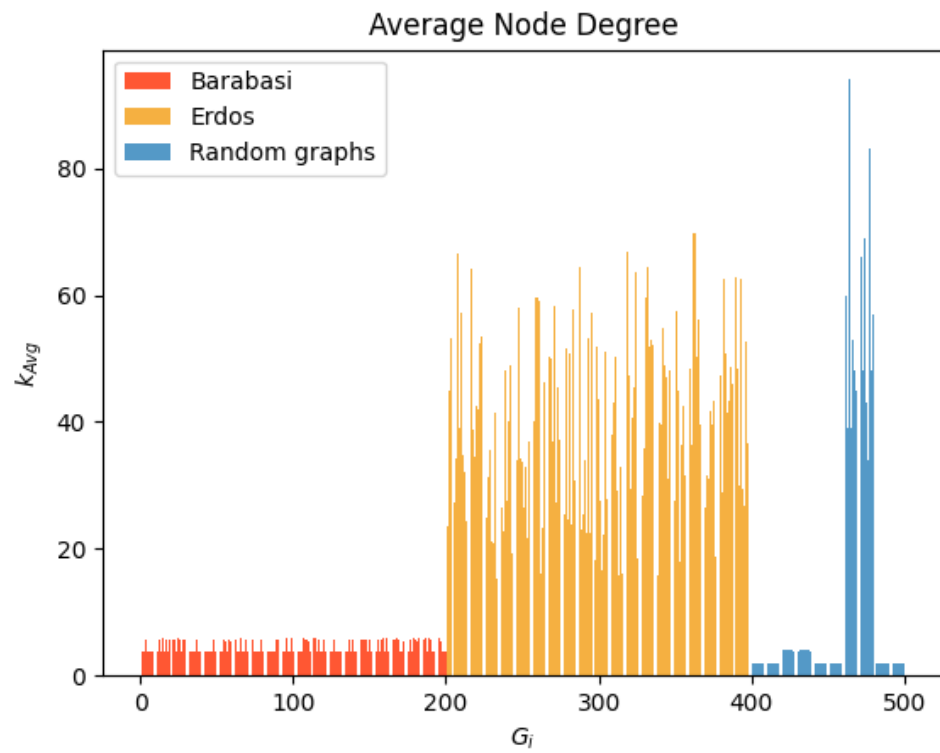
- Plotting Number of nodes of each graph tells us how nodes of each graph are assigned to this dataset and we can visualise it clearly.



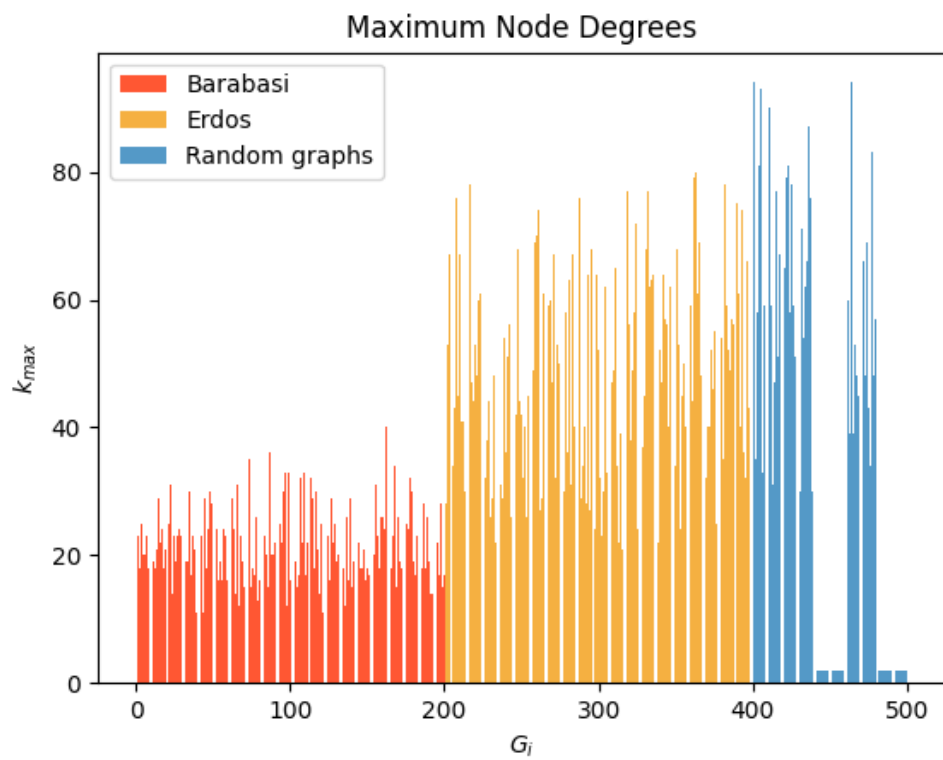
- Plotting Number of edges for each graph tells us how edges for each graph are assigned and distributed among each graph in this dataset.



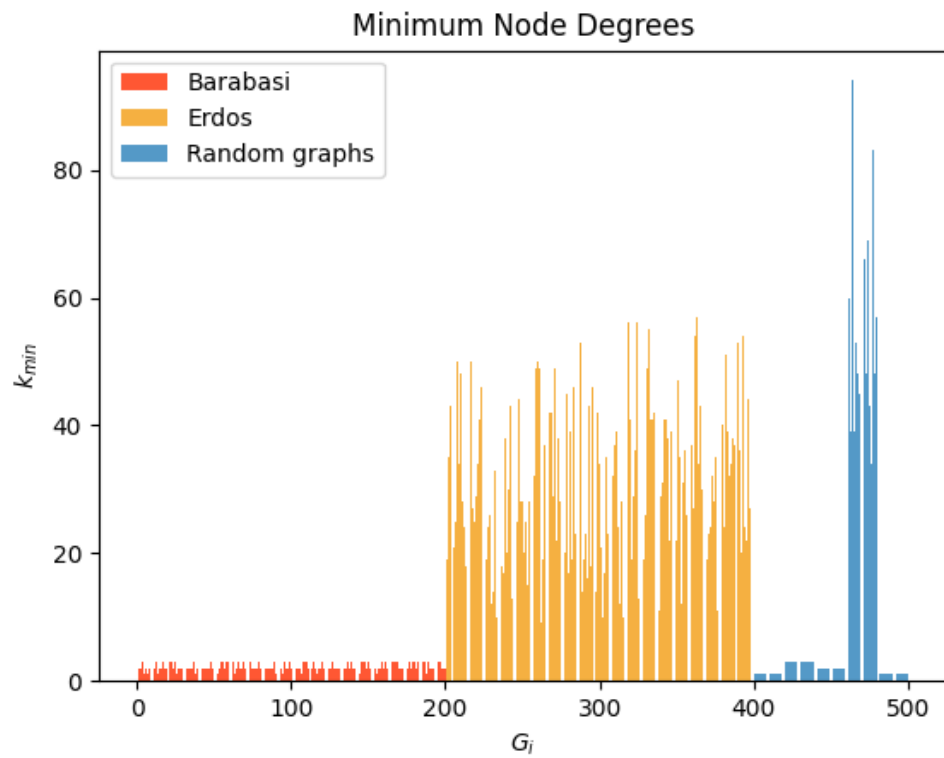
- Plotting the Average Node degree for each graph tells us how edges are distributed among each node in the graph.



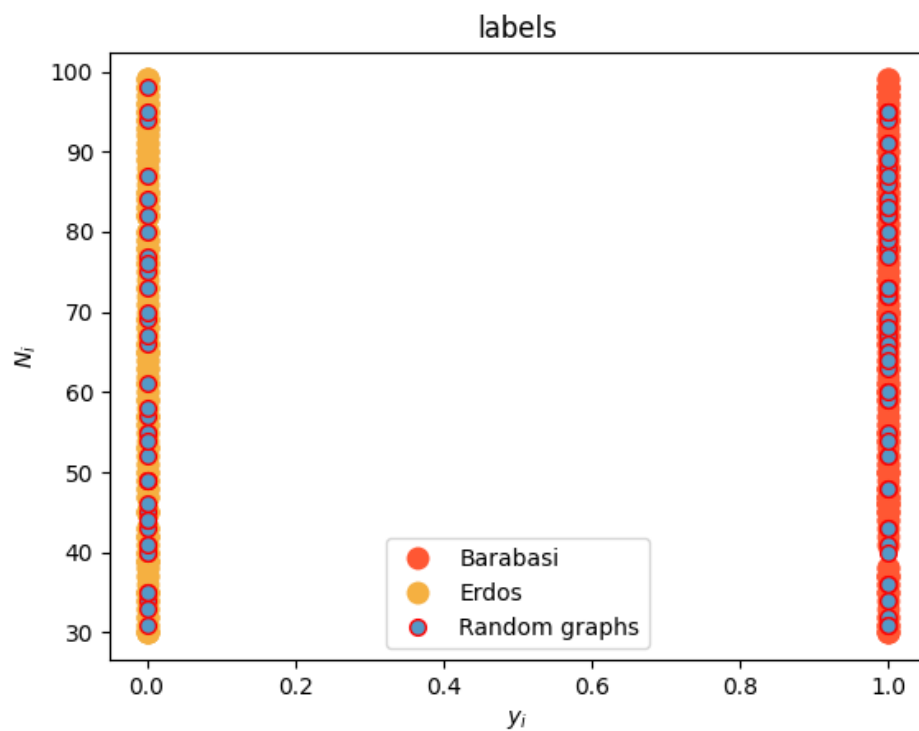
- Plotting a Maximum node degree for each graph



- Plotting a Minimum node degree for each graph

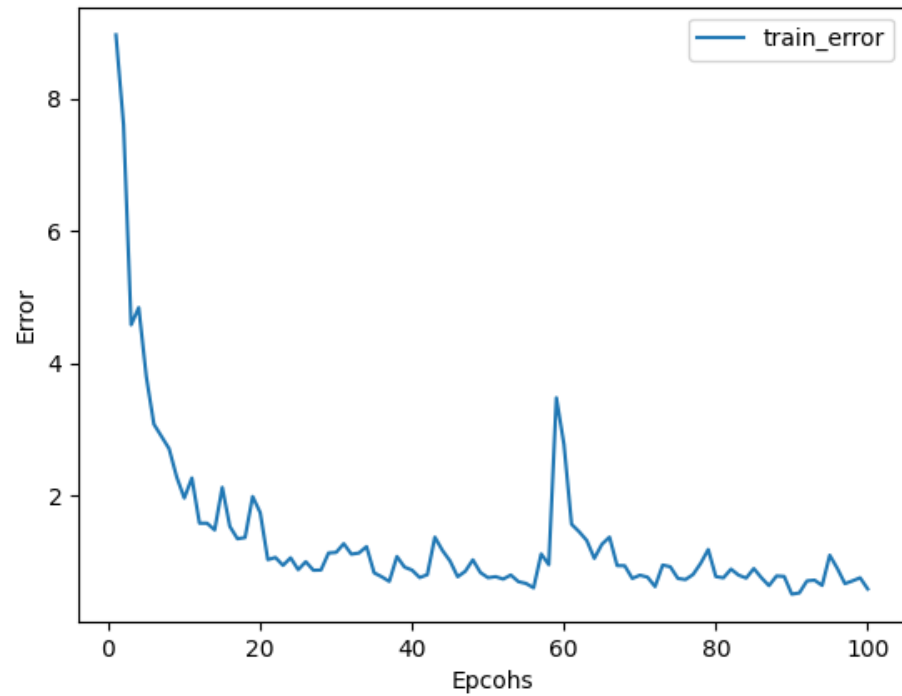


- Plotting a Label for each graph tells us how labels are assigned to graphs of different sizes.





- Plotting a Loss for each graph tells us how loss is varying over every epoch



- Plotting a Accuracy for each graph tells the accuracy of our model over every epoch

