Mehlam Saifudeen (mms330) and Harshita Kumar (hxk613)

Question 1

In this question, you will perform pathway-based enrichment analysis of the disease **Hepatomegaly**. The purpose of this method is to determine whether a particular set of genes is enriched for membership in a specific biological pathway. We will use the genes associated with Hepatomegaly and a database of known pathways and their associated genes (KEGG_2021 Human database for this question).

(a) The DisGeNet gene set file that contains the sets of genes associated with each disease is provided with this assignment. Extract the genes associated to Hepatomegaly from DisGeNet disease-gene network. Report the number of genes you found.

(b) Compare the list of disease-associated genes to the genes in each pathway in the KEGG_2021 database. Test whether disease-associated and pathway genes overlap significantly more than would be expected by chance. Identify and report the top 5 pathways that are significantly enriched for disease-associated genes. Report your enrichment results by providing visuals (bar graph, table, clustergram, etc.).

(c) Locate the neighbors of the disease-associated genes in the $hippie - ppi$ network by mapping the disease-associated genes to the $hippie - ppi$ network. Perform another enrichment analysis as you did in section (b) on an "extended set" of genes, where the "extended set" is obtained by merging the disease-associated genes with the genes that code for the interacting patners of the proteins coded by the Hepatomegaly genes. Report your results the same way you did in section (b).

(d) Compare results from sections (b) and (c). What was the purpose of performing enrichment analysis with an "extended set" of genes? How did the addition of interacting partners affect the enrichment results? What have we discovered from these analyses? Are your findings consistent with the biological knowledge of Hepatomegaly?

Answer.

a. The code snippet below extracts the set of genes associated with Hepatomegaly from the DisGeneNet disease gene network.

```
# Load the CSV file as a DataFrame
disease_genes_df = pd.read_csv('disgenet-genesets.csv', header=None)

# Extract the genes from a specific row (The one with the disease Hepatomegaly)
print(disease_g (variable) disease_genes_df: Any
Hepatomegaly = disease_genes_df.iloc[0].dropna().drop(0)

# Report the number of genes found
num_genes = len(Hepatomegaly)
print(f'Number of genes found for Hepatomegaly: {num_genes}')
```

✓ 0.5s

Below is the output of the list of genes:
[A1BG, AHR, ALB, ALDH1B1, CAMK2A, CRYGD, MAPK14, CYBA, CYP1A1, CYP1A2, CYP1B1, ADAM3A, DSCAM, FGF12, NR5A2, HDC, IGKC, LEP, LEPR,

MYO1B, TRIM37, NFE2L2, NOS2, NOS3, PEPD, PPARA, RELA, TGFB1, NR1H2, KLF11, COPS3, ABCC3, NR1I2, ATG5, ADAMTSL2, KEAP1, NR1I3, NR1H4, ADAMTS5, AKR7A3, MLC1, DAPK2, SLC29A3, SESN2, CYGB, GALNT13, MBOAT2, GPR155, SBSPON, PRSS35, CABCOCO1, STAC3, MIR192, ACOT1]

The number of genes found for **Hepatomegaly is 54 genes.**

b. In order to perform this task we first imported the pandas and gseapy libraries. From the gseapy libraries, we obtained the tabulate and the barplot and dot-plot libraries. The code snippet above shows how I have extracted the genes associated with Hepatomegaly along with reporting them in the form of a list. Next, I created an empty data frame and extracted all the data from the disgenet CSV file. Using this data frame and the list of Hepatomegaly genes, I conducted enrichment analysis using a package from the gseapy library. As asked in the question I reported the top 5 pathways that are significantly associated with genes. The code snippet below shows how this was conducted and the results table as well.

```python
import gseapy as gp
from tabulate import tabulate
from gseapy.plot import barplot, dotplot
```

Python

```python
enriched_genes = gp.enrichr(gene_list=hepatomegaly_genes, gene_sets=['KEGG_2021_Human'])
print(tabulate(enriched_genes.results.head(5)))
```

Python

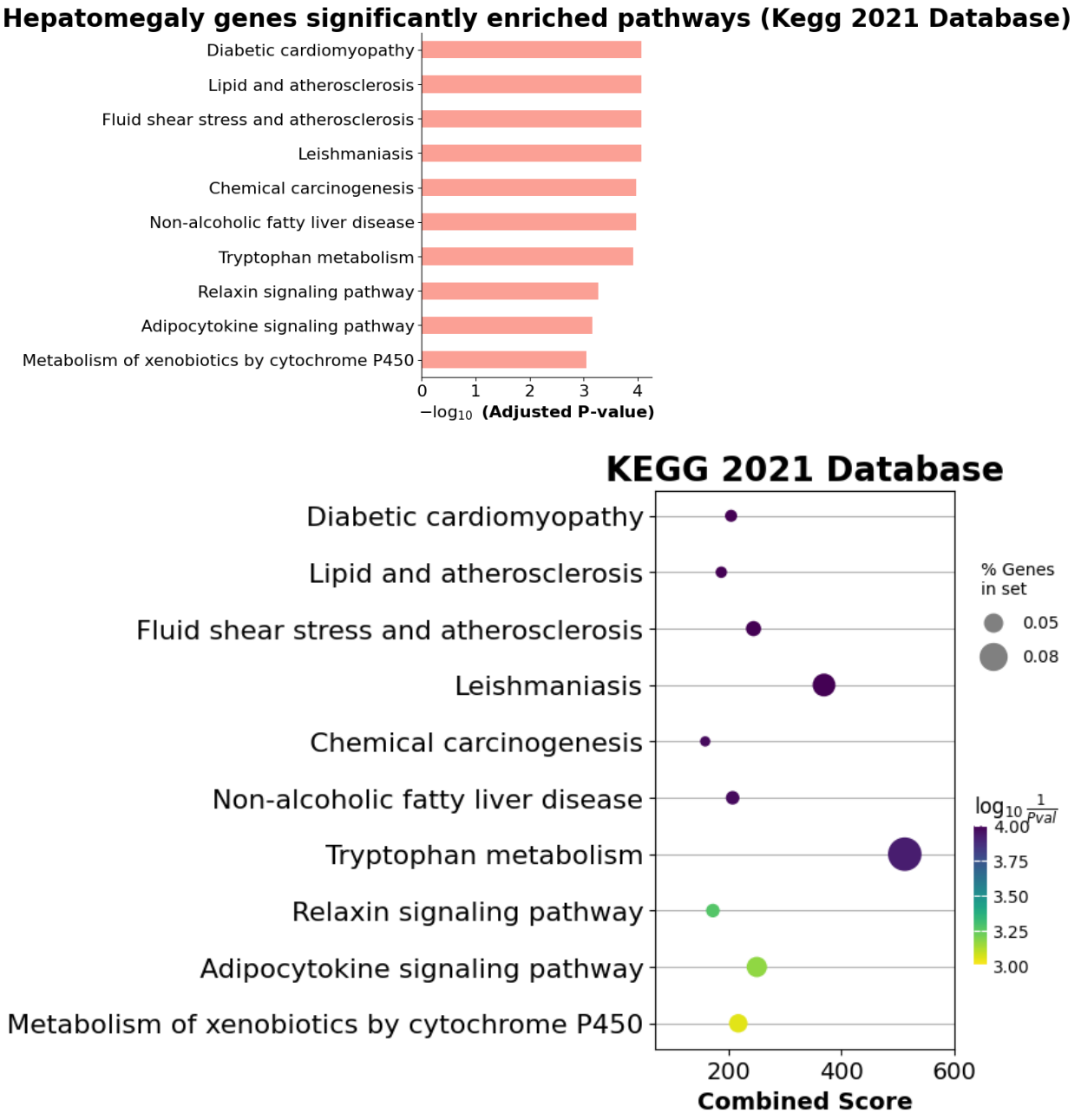
0	KEGG_2021_Human	Diabetic cardiomyopathy	7/203	1.18312e-06	8.65035e-05	0	0	15.0076	204.814	TGFB1;NOS3;CAMK2A;CYBA;MAPK14;PPAR
1	KEGG_2021_Human	Lipid and atherosclerosis	7/215	1.73554e-06	8.65035e-05	0	0	14.1332	187.465	NOS3;CAMK2A;CYP1A1;CYBA;MAPK14;REL
2	KEGG_2021_Human	Fluid shear stress and atherosclerosis	6/139	1.98517e-06	8.65035e-05	0	0	18.6212	244.493	NOS3;KEAP1;CYBA;MAPK14;RELA;NFE2L2
3	KEGG_2021_Human	Leishmaniasis	5/77	2.02347e-06	8.65035e-05	0	0	28.1661	369.277	TGFB1;NOS2;CYBA;MAPK14;RELA
4	KEGG_2021_Human	Chemical carcinogenesis	7/239	3.49936e-06	0.000106463	0	0	12.6557	158.993	NR1I3;CYP1A2;CYP1A1;CYP1B1;AHR;PPA

+ Code + Markdown

```python
barplot(enriched_genes.res2d, title='Hepatomegaly genes significantly enriched pathways (Kegg 2021 Database)')
dotplot(enriched_genes.res2d, title='KEGG 2021 Database', size=35)
```

Python

Next, I used the barplot and dot plot packages to show the results I obtained. This is included in the snippet above. Below are the results:

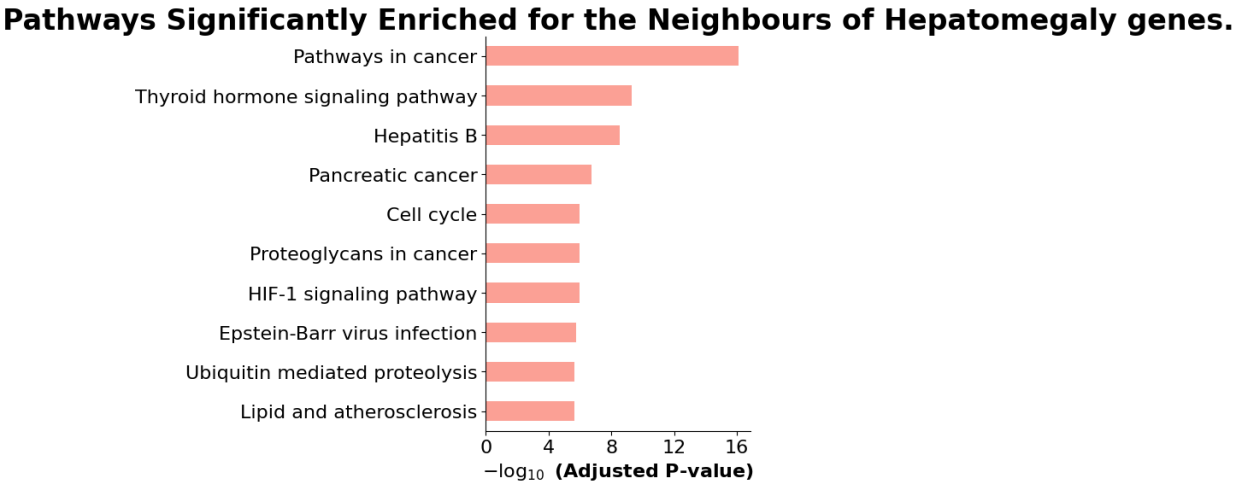


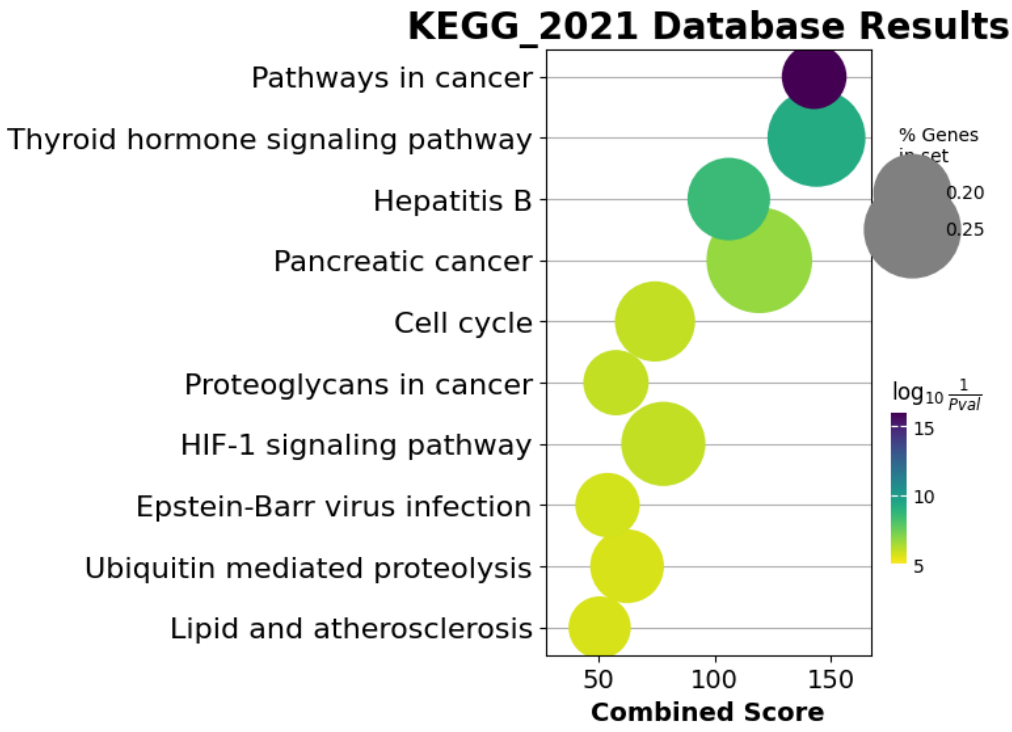
c. To locate the neighbors of the disease-associated genes in the hippie-ppi network, the hippie-ppi network was mapped into a graph. Furthermore, any self-loops were omitted. Then, the function findnode was used to map the Hepatomegaly genes to that graph. It was realized that 28 out of the 54 genes were found in the network. This is expected as about half of them can be expected to be in the basic human interaction network, while the rest are probably specific to the disease. The indices which were not found were removed. Then, the neighbor function was used to find the genes that neighbor

each of these in the graph. A total of 1483 genes were found. The code for this can be found below:

```
HipData = readtable('/Users/harshitakumar/Documents/Spring 2023/CSDS 459/hippie-ppi.csv');
hipGraph = graph(HipData.(1), HipData.(2), 'OmitSelfLoops');
indices = findnode(hipGraph, hepData);
indices = nonzeros(indices);
hipGrahNeighbours = [];
for i = 1:length(indices)
    hipGrahNeighbours =  [hipGrahNeighbours; neighbors(hipGraph, indices(i))];
end
```

Then, this list was transferred over to the Python IDE to perform the enrichment analysis similar to the above in part (b). The bar plots and dot plots were regenerated below:





d. As can be seen from parts b and c, the enrichment analysis revealed different results. Therefore part c has more significantly enriched pathways than the analysis with only disease-associated genes. This suggests that incorporating the interacting partners of the disease-associated genes into the analysis can provide additional insights into the underlying biological mechanisms. An extended gene set can be useful because it can help confirm results as the increase in the number of genes creates smaller errors or variations in p-Values to create more significant enrichment. It also helps capture additional genes that could be relevant to the disease pathways. Thus, we might expect to see higher enrichment scores. This is consistent with biological knowledge however, as either with a smaller gene set or an extended one, we would expect the same pathways to be enriched. For hepatomegaly, this makes sense as it is associated with metabolism, biosynthesis, and signaling, making it a symptom of many liver diseases. This is what we see above.

Question 2

(a) A set of nodes in a network is said to be modular if the nodes are densely connected to each other while being sparsely connected to the rest of the network. Formulate a statistic to objectively quantify the modularity of a given set of nodes in a given network. Don't forget to define the variables you use.

(b) Empirically assess whether the statistic you formulated for quantifying network modularity depends on the number of nodes in the given set. You can do this by selecting random sets of given sizes and plotting modularity as a function of size, also considering variance. Provide a plot of modularity value vs. the number of edges to show the relationship between size and modularity, and explain what can be interpreted from this plot.

(c) You have already obtained the associated gene set of Hepatomegaly from Question 1. For Hepatomegaly, assess the modularity of the proteins coded by the genes associated with the disease, using the statistic you formulated in section (a). Using permutation tests, assess the statistical significance of the network modularity of Hepatomegaly. Make sure that you account for the number of genes associated while performing this analysis. Report the number of edges, number of nodes, modularity score, and p-value. Interpret your results from a biological perspective.

Answer.

a. In order to objectively quantify the modularity of a given set of nodes in a given network, we can use the concept of modularity as a measure of the quality of a network partition. Modularity is defined as the fraction of the edges in the network that fall within the given set of nodes minus the expected fraction if the edges were distributed at random.
Mathematically, the modularity of a set of nodes S in a network G = (V, E) is defined as
$Q(S) = 0.5M * \Sigma [A_{ij} - \{(k_i - k_j) / 2m\}]$ for nodes i, j in S

where:

M = total number of edges in the network
A_{ij} = 1 if there is an edge between nodes i and j, and 0 otherwise
ki = the degree of the node i (Number of edges incident to i)
kj = the degree of node j
Σ = the sum of the values of the function inside the parenthesis for all pairs of nodes i and j in S

The modularity Q(S) ranges from -1 to 1, where values close to 1 indicate strong modularity, values closer to 0 indicate no significant modularity and values which are closer to -1 indicate anti-modularity. This means that if the value of the modularity is closer to 1 then the nodes in S are strongly connected to each other

and sparsely connected to the rest of the network and vice-versa for when the value of the modularity is closer to -1.

Hence, to objectively quantify the modularity of a given set of nodes in a given network, we can calculate the modularity Q(S) using the formula shown above and interpret the results accordingly. Thus the statistic formulated above can objectively quantify the modularity of a given set of nodes in a network.

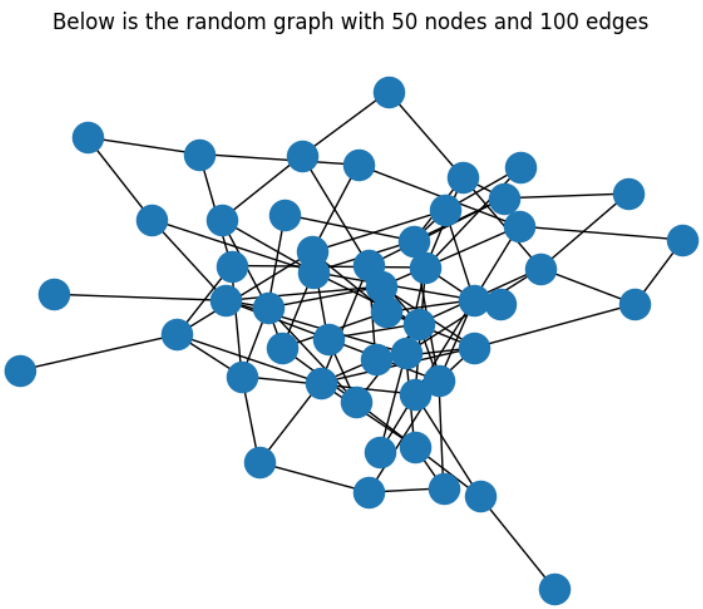
b. To assess whether the statistic formulated above for quantifying network modularity depends on the number of nodes in a given set, I first started off by defining the modularity statistic as a function in jupyter notebook. Below is the code snippet for the same.

```
# Define the modularity statistic as a function
def modularity(S, G):
    m = G.number_of_edges()
    ki = dict(G.degree())
    A = nx.to_numpy_array(G)
    S_indices = [i for i, n in enumerate(G.nodes()) if n in S]
    S_size = len(S_indices)
    S_mask = np.zeros((G.number_of_nodes(), 1))
    S_mask[S_indices] = 1
    kis = ki.values()
    kis = np.asarray(list(kis))
    ki_sum = np.sum(kis)
    expected_frac = (ki_sum / (2 * m)) ** 2
    A_S = np.multiply(A, np.dot(S_mask, S_mask.T))
    m_S = np.sum(kis[S_indices]) / 2
    modularity_S = np.sum(A_S) / (2 * m) - expected_frac * (m_S / m) ** 2
    return modularity_S
```

Next, we created a random network with 50 nodes and 100 edges and defined ranges of the sizes for the random sets of nodes. An empty array was created to store the modularity values later. The code snippet below shows how this was executed:

```
# create a random network with 50 nodes and 100 edges
G = nx.gnm_random_graph(50, 100, directed = False)

# define the range of sizes for the random sets of nodes
sizes = np.arange(5, 25, 5)

# Define the number of repetitions for each size
num_reps = 50

# Initialize an array to store the modularity values
modularities = np.zeros((len(sizes), num_reps))

# Print the graph and all the values below
nx.draw(G)
plt.title("Below is the random graph with 50 nodes and 100 edges")
plt.show()
print(modularities)
```

Below is the random graph that was generated with 50 nodes and 100 edges



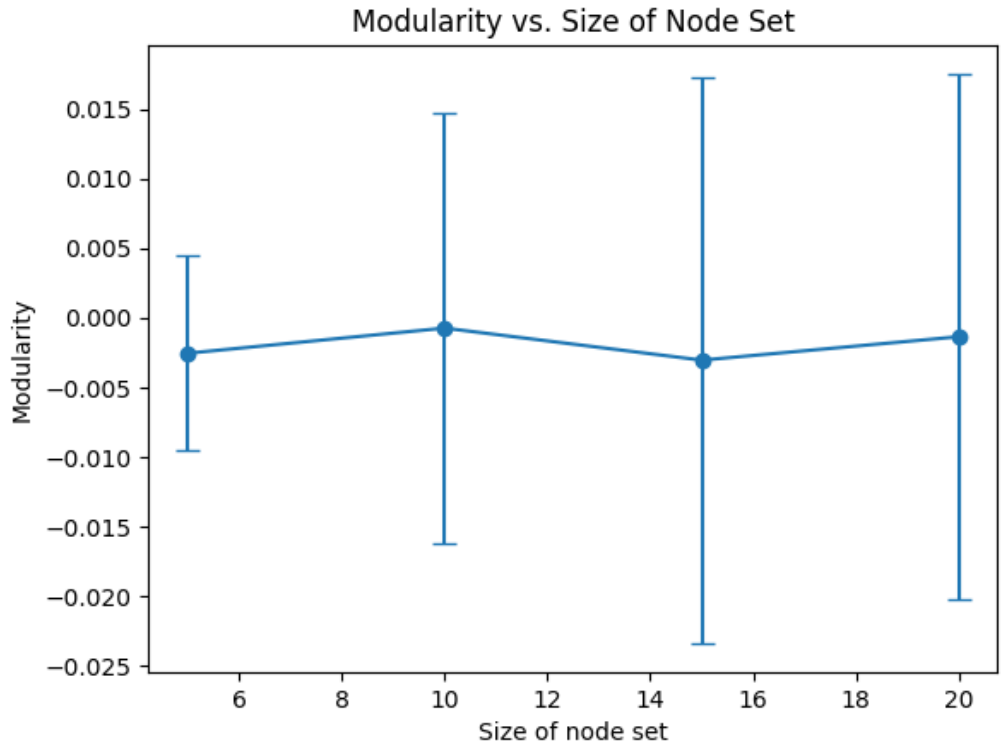
Next, we looped over the sizes and repetitions of each and computed the modularity for each random set of nodes. The mean and the standard deviation of the same were calculated as well. Below is the code snippet to show how this was done:

```python
# Loop over the sizes and repetitions and compute the modularity for each random set of nodes
for i, size in enumerate(sizes):
    for j in range(num_reps):
        S = set(np.random.choice(G.nodes(), size, replace=False))
        modularities[i, j] = modularity(S, G)

# Calculate the mean and standard deviation of the modularity values for each size
mean_modularities = np.mean(modularities, axis=1)
std_modularities = np.std(modularities, axis=1)

print(mean_modularities)
print(std_modularities)
```

```
[-0.0025665 -0.000756  -0.003059  -0.0013735]
[0.00701688 0.01548873 0.0203394  0.01888036]
```

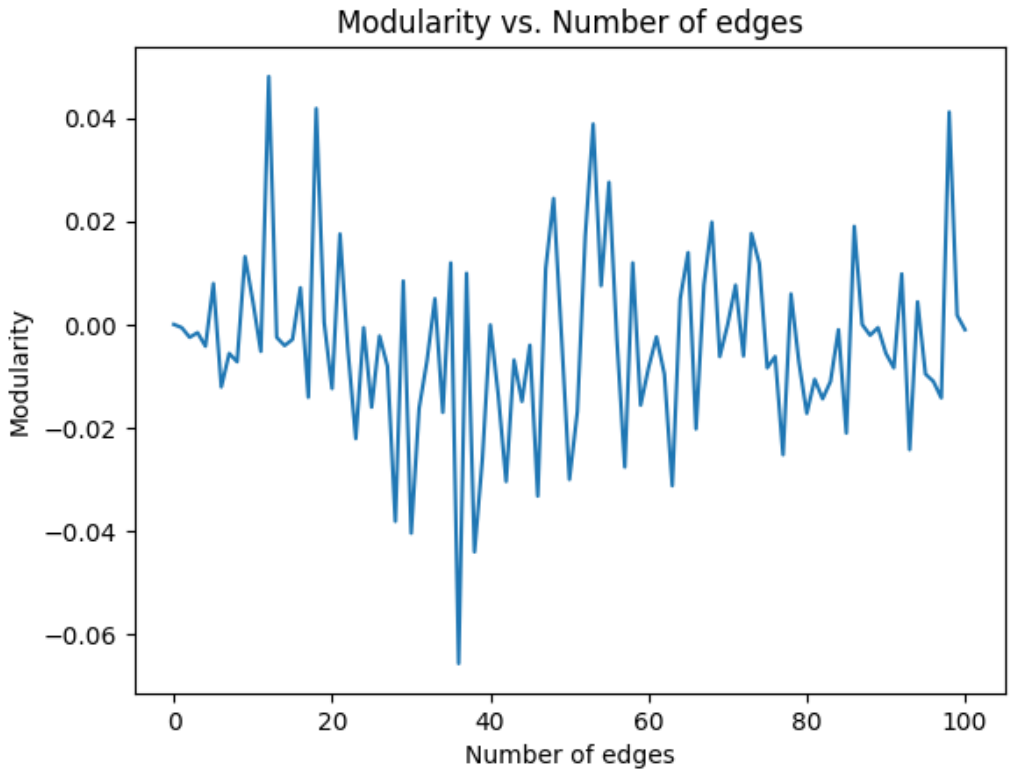
```python
# Plot the modularity as a function of size
plt.errorbar(sizes, mean_modularities, yerr=std_modularities, fmt='o-', capsize=5)
plt.xlabel('Size of node set')
plt.ylabel('Modularity')
plt.title('Modularity vs. Size of Node Set')
plt.show()
```

Below is the graph that shows modularity vs the Size of a node-set



Below is the code snippet and the plot that shows the Modularity against the number of edges for the entire random network generated.

```
# Plot the modularity as a function of the number of edges
num_edges = np.arange(G.number_of_edges() + 1)
modularity_vs_edges = np.zeros(len(num_edges))
for i, num_edge in enumerate(num_edges):
    G_sub = nx.subgraph(G, np.random.choice(G.nodes(), num_edge, replace=True))
    modularity_vs_edges[i] = modularity(set(G_sub.nodes()), G)
plt.plot(num_edges, modularity_vs_edges, '-')
plt.xlabel('Number of edges')
plt.ylabel('Modularity')
plt.title('Modularity vs. Number of edges')
plt.show()
```



From the plots above we can see that the modularity value is close to 0. This is because there are no underlying structures or modules to the network. As we go on and add more edges to the network, the modularity values may slightly increase or decrease due to either the chance of connections forming groups of nodes or more edges being added between groups respectively. Nevertheless, the overall trend should be a stable level of modularity.

The modularity remains close to the value of 0 because as the network grows, it becomes more connected and nodes become more likely to have connections

with nodes in different groups which makes it harder to identify distinct modules. At the same time, there is a chance that connections form within groups as well which makes it a little easier to identify distinct modules. Although there is a much lesser chance of this happening.
Overall such a plot for modularity in a random network can be used as a benchmark for comparison to more complex networks that exhibit stronger modular structures.

c. To assess the modularity of the proteins coded by the genes associated with a disease, we can use the same statistic that we formulated in part a. We can also use permutation tests to assess the statistical significance of the network modularity.
In the code snippet below, we first load the gene set associated with a disease which in this case is Hepatomegaly, and the protein-protein interaction network. We then extract the subgraph induced by the disease genes and calculate its modularity using the same statistic formulated earlier.
Next, we perform a permutation test to assess the statistical significance of the modularity of the disease gene set. We randomly permute the disease gene set multiple times and calculate the modularity of the subgraph induced by the permuted gene set. Next, we calculate the p-value as the proportion of the permutations where the modularity is greater than or equal to the modularity of the original subgraph. Finally, we print the number of nodes and edges in the subgraph, the modularity score, and the p-value. This gives us an idea of how modular the subgraph induced by the disease genes is, and whether the modularity score is statistically significant.

```
print("This is the number of nodes :", ppi.number_of_nodes(), "This is the number of edges : ", ppi.number_of_edges())
```
✓ 0.2s

```
This is the number of nodes : 17336 This is the number of edges :  383481
```

```
def unidentifiedGene(arr):
    genes_to_remove = []
    for x in arr:
        try:
            list(ppi.nodes).index(x)
        except:
            genes_to_remove.append(x)
    return set(arr) - set(genes_to_remove)


Hepatomegaly_gene_set = unidentifiedGene(Hepatomegaly)
print(Hepatomegaly_gene_set)
```
✓ 0.1s

```
{'NR1I2', 'DSCAM', 'NOS2', 'STAC3', 'FGF12', 'ACOT1', 'HDC', 'NR1H2', 'PEPD', 'TGFB1', 'MYO1B', 'AHR', 'NR5A2', 'ATG5', 'NR1H
```

```
# Create the communities to be used for analysis
Hepatomegaly_subset = defined_subsets(ppi, Hepatomegaly_gene_set)
```
✓ 0.0s

```
modularity_Hepatomegaly = nx_comm.modularity(ppi, Hepatomegaly_subset)
```
✓ 1.2s

Below is the modularity of Hepatomegaly

```
print("This is the Modularity of Hepatomegaly :", modularity_Hepatomegaly)
```
✓ 0.0s

```
This is the Modularity of Hepatomegaly : 9.654497306616665e-05
```

```
valid_partition = nx_comm.kernighan_lin_bisection(ppi)
print(nx_comm.modularity(ppi,valid_partition))
```
✓ 17.5s

```
0.05766273164497051
```

Using Permutation Tests to Assess Statistical Significance

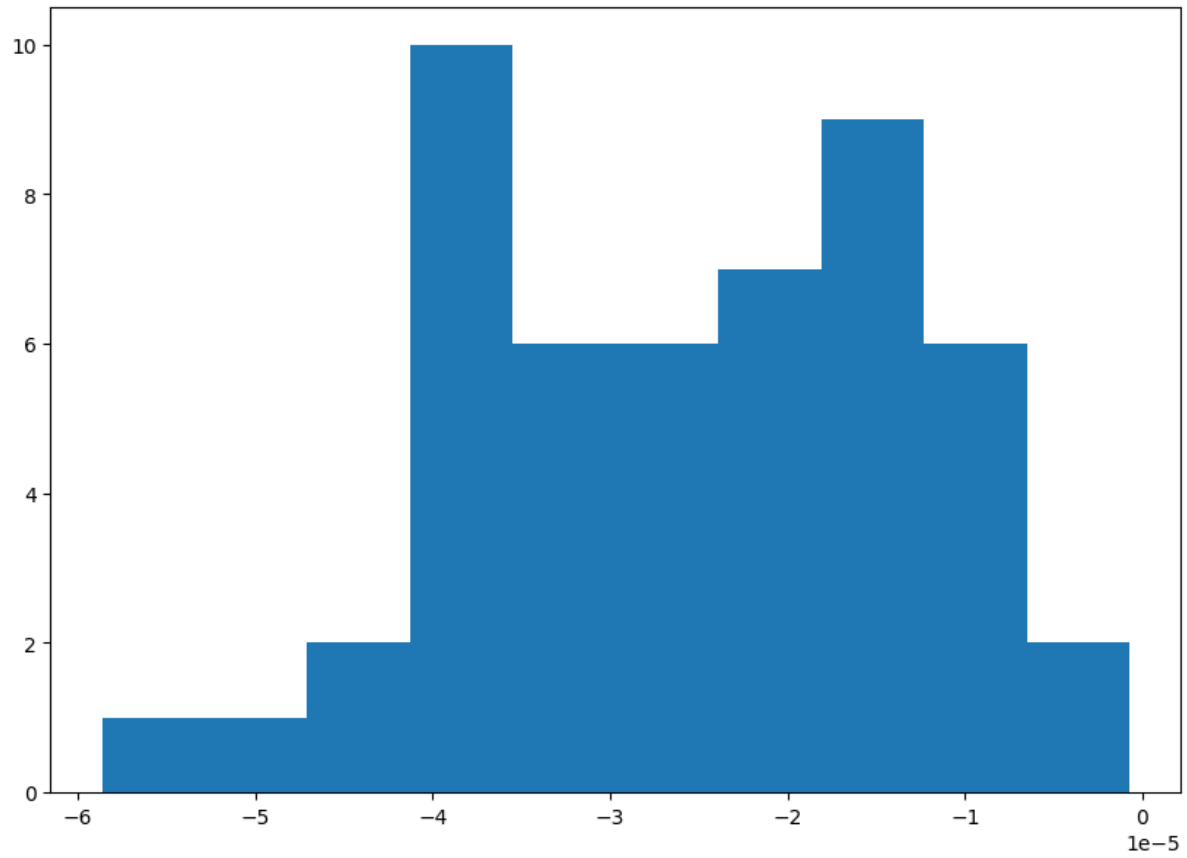
```
def random_subsets_permutation_tests(G, size):
    # subset of entire environment
    node_environment = set(G.nodes)
    #node_environment = set(node_environment);
    #create a subset of size increment
    random_community = sample(list(G.nodes()), size)
    #remove these nodes from the partition environment
    node_environment.difference_update(random_community)
    random_community = set(random_community)
    return [random_community, node_environment]
```
✓ 0.0s

```
# generate a random set that is the same size as the Hepatomegaly Data set
size = len(Hepatomegaly_subset[0])
sample_subset = random_subsets_permutation_tests(ppi,size)
sample_subset_modularity = nx_comm.modularity(ppi,sample_subset)
print("Below is the modularity of a random subset of the ppi data set :", sample_subset_modularity)
```
✓ 1.6s

```
Below is the modularity of a random subset of the ppi data set : 3.970194283470678e-05
```

```
# Generate sets of random permuations with both data sets
def permutation_modularities(G, disease_subset, sample_subset, size, num_permutations):
    modularities = []
    merged_arr = list(disease_subset[0].union(sample_subset[0]))
    for x in range(num_permutations):
        permutation = sample(merged_arr, size)
        permutation_sorted = sorted(permutation)
        save = defined_subsets(ppi, permutation_sorted)
        modularities.append(nx_comm.modularity(ppi, save))
    return modularities
```
✓ 0.0s

```
def plot_modularity_diff(mod_arr, disease_modularity):
    for i in range(len(mod_arr)):
        mod_arr[i] = mod_arr[i] - disease_modularity

    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(mod_arr)

    # Show plot
    plt.show()
```
✓ 0.0s



From the results obtained, we can suggest that the random network has a higher level of modularity compared to proteins coded by the genes associated with the disease Hepatomegaly which has the same size as the network. This indicates that the proteins associated with the disease are less densely connected while being more connected to the rest of the network, which is a characteristic of a modular network.
The p-value obtained from the permutation test suggests that the modularity score of the disease-associated proteins is statistically significant, which means that the observed modularity score is unlikely to have occurred by chance itself.

From a biological perspective, this could suggest that the proteins coded by the disease-associated genes may function together in a few of the biological pathways or processes that are relevant to the disease. Additionally, identifying these modular sub-networks could potentially lead to the discovery of new drug targets or therapeutic strategies for treating the disease. In addition to this, based on the modularity values, the findings are consistent with the biological findings. The modularities for the diseases are distinct from the modularities of random samples by a large margin.

Question 3

(a) Construct a gene-gene interaction network from the given disease-gene interaction network (*DisGeNet*) such that two genes are connected if they are likely to be associated with the same disease. Explain your process and the metrics you define in your process. Report the number of nodes and edges of the graph you construct.
(Some ideas: You can create a graph where each gene is represented by a node and an edge is drawn between two nodes if the corresponding genes are connected according to a threshold you set (for two proteins to be considered connected, they must be associated with a certain number of diseases). The similarity or correlation between the "disease association profiles" of two genes can also be measured using other metrics like Jaccard Similarity, Cosine Similarity, or Mutual Information. Using a correlation threshold, decide whether to connect them.)

(b) Compare your constructed network with the $hippie - ppi$ network in terms of the overlap of the edges. Report, visualize, assess the significance of, and biologically interpret your results.

(c) Using your constructed network, compare neighbor overlaps for each gene to the $hippie - ppi$ network (do a gene-by-gene comparison). Sort and rank the genes with the most neighbor overlaps. List the top 10 genes (e.g., according to statistical significance or a metric of your choice) and the overlaps between their neighbors. Biologically interpret your results.

Answer

a. The goal of the code is to create a gene-gene interaction network based on a given disease-gene interaction network such that two genes are connected if they are likely to be associated with the same disease. In other words, we want to construct a graph where each node represents a gene and each edge represents a connection between the two genes based on their association with common diseases.
   To achieve this goal, the following steps were taken:
   Load the data: I started by loading the data from the CSV file into a pandas data frame. Since the first column contains the names of diseases and the remaining columns contain the associated genes for each disease, we set header = none to indicate no header row
   Create a list of all genes: I did this by creating a set of all genes listed in the data frame by iterating over each row and adding all non-disease elements to the set. This set will be used to create the nodes of the graph.
   Next, I create a dictionary mapping each gene to a set of associated diseases by iterating over each row and adding the corresponding disease to the set for each gene in that row. This dictionary is used to determine the edges of the graph.
   I then created a graph using the networkx library with thresholding as the metrics to be used to determine if a pair of genes is associated with at least two common diseases. If they are, then an edge is added between them. I set the threshold here as 2, but this can be adjusted depending on the connectivity of the graph.

Lastly, the number of nodes and edges of the graph were reported. The code snippet below shows how these steps were executed. We can see below that step 3 took nearly an hour to run due to a threshold of 2.

```python
# Step 1: Load the data
df = pd.read_csv("disgenet-genesets.csv")

# Step 2: Create a set of all genes
genes = set()
for row in df.itertuples(index=False):
    genes.update(set(row[1:]))

# Step 3: Create a dictionary mapping each gene to a set of associated diseases
gene_disease_map = {}
for row in df.itertuples(index=False):
    disease = row[0]
    for gene in genes:
        if gene in row[1:]:
            if gene not in gene_disease_map:
                gene_disease_map[gene] = set()
            gene_disease_map[gene].add(disease)
```

✓ 53m 42.8s

```python
# Step 4: Create a graph using networkx library with thresholding
G = nx.Graph()
for gene1 in gene_disease_map:
    for gene2 in gene_disease_map:
        if gene1 == gene2:
            continue
        common_diseases = gene_disease_map[gene1].intersection(gene_disease_map[gene2])
        if len(common_diseases) >= 2:  # set the threshold here as 2
            G.add_edge(gene1, gene2)
```

✓ 37.3s

```python
# Step 5: Report the number of nodes and edges of the graph
print(f"Number of nodes: {G.number_of_nodes()}")
print(f"Number of edges: {G.number_of_edges()}")
```

✓ 0.0s

```
Number of nodes: 5548
Number of edges: 898090
```

The number of edges in the graph is 898090 and the number of nodes is 5548.

Question 3(a)

In this question, we first aimed to determine which genes interact with each other by running through each of the diseases and linking each gene to one to create an interaction. This was repeated for each gene:

```
for i =1:height(DisData)
    i
    data = table2cell(DisData(i, 2:end))';
    data = data(~cellfun('isempty',data));
    for j = 1:length(data)
        for k = j+1:length(data)
            geneNetwork = [geneNetwork; [data(j), data(k)]];
        end
    end
end
```

PM

- While we aimed to repeat this for all the diseases, due to limitations of time having run the code already for 2+ hours, only a portion of the genes was able to be included. However, this still created a comprehensive dataset for the analysis.
- Then, the next part was to find out which gene interactions overlap between diseases, and this was done using the group counts function in Matlab. From the data, it was realized that some gene interactions were present in over 33% of the disease, while some were only present in one. The following lines of code were used:

```
countGene = groupcounts(geneNetwork, ["geneNetwork1", "geneNetwork2"]);
graphGene = countGene(find(countGene.GroupCount > 1), :);
gGene = graph(string(graphGene.(1)), string(graphGene.(2)), graphGene.(3));
%     LWidths = 5*gGene.Edges.Weight/max(gGene.Edges.Weight);
%     p = plot(gGene, "k", 'LineWidth',LWidths);
numEdges = height(graphGene);
numNodes = height(gGene.Nodes);
```

- Using this, we selected interactions that were at least present in two diseases.
- Thus, in the graph, there were 2483 nodes found with 157784 edges.
- To create the graph, we gave each interaction a weight (dependent on how many diseases it shows up on). This can be seen from the variable LWidths.

Question 3(b)

- We decided the best way to compare the Hippie-ppi network and our network is by comparing which gene interactions are present in both, and which are present in only one.

```
HipData.Properties.VariableNames = ["geneNetwork1","geneNetwork2"];
commonGenes = intersect(HipData, graphGene(:, 1:2));
HipOnlyGenes = setdiff(HipData, graphGene(:, 1:2));
graphOnlyGenes = setdiff(graphGene(:, 1:2), HipData);
```

- To do this in Matlab, we used the functions “intersect” and “setdiff”.
- Thus, we first found which gene interactions are present in both networks. A total of 468 gene interactions were found to be common between them.
- Then, it was realized that there were over 380000 gene interactions unique to the Hippie-ppi network and over 150000 gene interactions were common to the disease network.
- This does make sense because the Hippie-ppi network looks at protein interactions in homosapiens, while the Disgennet looks at gene interactions specific to a disease. We do expect some overlap as some interactions might also be present without disease, but we expect a lot more interactions to be specific to a disease as that is what causes the disease’s phenotype.

Question 3(c)

- To do the gene-by-gene comparison of the neighbor overlaps, we first found the list of genes that do overlap between the Hippie-PPI network and disease network.

```
genesCommon = unique([commonGenes.(1); commonGenes.(2)]);
```

- Then we ran through the whole list, and identified neighbors on each side of the list for both network graphs we have created. This involved finding genes that are directly connected to each of the genes.
- Then, the overlap score for each gene was calculated using the Jaccard index. This just means that the genes with common neighbors were listed.

```
overlapScore = zeros(1, length(genesCommon));
for i = 1:length(genesCommon)
    gene = genesCommon(i);
    [~, ~, p1] = intersect(gene,HipData{:,1});
    [~, ~, p2] = intersect(gene,HipData{:,2});
    [~, ~, p3] = intersect(gene,graphGene{:,1});
    [~, ~, p4] = intersect(gene,graphGene{:,2});
    neighborHip = [HipData{p1, 2}; HipData{p2, 1}];
    neighborDis = [graphGene{p3, 2};graphGene{p4, 1}];
    overlapScore(i) = length(intersect(neighborHip, neighborDis)) / length(union(neighborHip, neighborDis));
end
```

- However, from our data, it was realized that no genes had an overlap score of more than 0. This is probably because in the disease network, as mentioned above, not all the genes were analyzed completely. This means that there would be a lot of overlaps that our analysis would have missed.
- However, biologically, we expect that genes with a higher overlap score would be ones that are functionally related or they are paths of a pathway.