import os

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import networkx as nx

from scipy.stats import ttest\_ind

from sklearn.feature\_selection import VarianceThreshold

from google.colab import files

file\_path = "gene\_counts.xlsx"

if not os.path.exists(file\_path):

    print("Dataset not found. Please upload the file.")

    uploaded = files.upload()

    file\_path = list(uploaded.keys())[0]

df = pd.read\_excel(file\_path)

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

control\_cols = [col for col in df.columns if "W-" in col]

heat\_cols = [col for col in df.columns if "H-" in col]

df[control\_cols + heat\_cols] = df[control\_cols + heat\_cols].apply(pd.to\_numeric, errors='coerce')

df.dropna(subset=control\_cols + heat\_cols, inplace=True)

df[control\_cols + heat\_cols] = df[control\_cols + heat\_cols].astype(float)

df["Log2FC"] = np.log2(df[heat\_cols].mean(axis=1) + 1) - np.log2(df[control\_cols].mean(axis=1) + 1)

p\_values = [ttest\_ind(df.loc[i, heat\_cols].astype(float).values, df.loc[i, control\_cols].astype(float).values, equal\_var=False, nan\_policy='omit').pvalue for i in df.index]

df["P\_value"] = p\_values

variance\_selector = VarianceThreshold(threshold=1.0)

filtered\_data = variance\_selector.fit\_transform(df[control\_cols + heat\_cols])

filtered\_gene\_indices = variance\_selector.get\_support(indices=True)

filtered\_genes = df.iloc[filtered\_gene\_indices, 0]

filtered\_df = df[df["GeneID"].isin(filtered\_genes)].set\_index("GeneID")[control\_cols + heat\_cols]

corr\_matrix = filtered\_df.T.corr(method='pearson')

corr\_matrix = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

threshold = 0.85

strong\_edges = np.where(corr\_matrix > threshold)

edge\_list = [(filtered\_genes.iloc[i], filtered\_genes.iloc[j]) for i, j in zip(\*strong\_edges) if i != j]

G = nx.Graph()

G.add\_edges\_from(edge\_list)

clusters = {node: i for i, comp in enumerate(nx.connected\_components(G)) for node in comp}

df["Cluster"] = df["GeneID"].map(clusters).fillna(-1).astype(int)

largest\_cluster\_id = df["Cluster"].value\_counts().idxmax()

heat\_shock\_genes = df[df["Cluster"] == largest\_cluster\_id]["GeneID"]

heatmap\_data = df[df["GeneID"].isin(heat\_shock\_genes)].set\_index("GeneID")[control\_cols + heat\_cols]

plt.figure(figsize=(12, 6))

sns.heatmap(heatmap\_data, cmap="coolwarm", xticklabels=True, yticklabels=False)

plt.title(f"Heatmap of Cluster {largest\_cluster\_id} (Heat Shock Prone Genes)")

plt.xlabel("Condition")

plt.ylabel("Genes")

plt.xticks(rotation=90)

plt.show()

plt.figure(figsize=(8, 6))

sns.scatterplot(x=df["Log2FC"], y=-np.log10(df["P\_value"]))

plt.axhline(-np.log10(0.05), color="red", linestyle="--", label="P=0.05 threshold")

plt.xlabel("Log2 Fold Change")

plt.ylabel("-log10(P-value)")

plt.title("Volcano Plot of Heat Shock Stress Response in Rice Genes")

plt.legend()

plt.show()

melted\_df = df[df["GeneID"].isin(heat\_shock\_genes)].melt(id\_vars=["GeneID"], value\_vars=control\_cols + heat\_cols, var\_name="Condition", value\_name="Expression")

plt.figure(figsize=(10, 5))

sns.boxplot(x="Condition", y="Expression", data=melted\_df, palette="Set2")

plt.xticks(rotation=90)

plt.title(f"Expression Distribution of Heat Shock Genes (Cluster {largest\_cluster\_id})")

plt.show()

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

data = {

    "Gene ID": [

        "LOC\_Os03g14180", "LOC\_Os04g36750", "LOC\_Os07g47840", "LOC\_Os01g01010", "LOC\_Os05g28140",

        "LOC\_Os09g33460", "LOC\_Os09g33470", "LOC\_Os09g33480", "LOC\_Os09g33490", "LOC\_Os09g33500",

        "LOC\_Os09g33510", "LOC\_Os09g33520", "LOC\_Os09g33530", "LOC\_Os09g33550", "LOC\_Os09g33555",

        "LOC\_Os09g33559", "LOC\_Os09g33580", "LOC\_Os09g33600", "LOC\_Os09g33630", "LOC\_Os09g33650",

        "LOC\_Os09g33670", "LOC\_Os09g33680", "LOC\_Os09g33690", "LOC\_Os09g33710", "LOC\_Os09g33720",

        "LOC\_Os09g33730", "LOC\_Os09g33780", "LOC\_Os09g33790", "LOC\_Os09g33800", "LOC\_Os09g33820"

    ],

    "Log2FC": [

        14.16, 14.13, 14.07, 12.98, 11.45, -0.98, -0.56, 0.25, -0.64, -0.16,

        0.27, -0.25, -0.17, -1.42, -0.84, -0.03, -0.80, 0.02, -0.58, -0.61,

        0.10, -0.42, -0.44, -1.07, -1.25, -0.52, 1.44, 0.74, 3.80, 3.85

    ]

}

df = pd.DataFrame(data)

df.set\_index("Gene ID", inplace=True)

import numpy as np

np.random.seed(42)

sample\_labels = [

    "Os-Nip-W-3-pil-1", "Os-Nip-W-3-pil-2", "Os-Nip-W-3-pil-3",

    "Os-Nip-W-1.5-pil-1", "Os-Nip-W-1.5-pil-2", "Os-Nip-W-1.5-pil-3",

    "Os-Nip-W-1.25-pil-1", "Os-Nip-W-1.25-pil-2", "Os-Nip-W-1.25-pil-3",

    "Os-Nip-H-1.60-pil-1", "Os-Nip-H-1.60-pil-2", "Os-Nip-H-1.60-pil-3",

    "Os-Nip-H-1.30-pil-1", "Os-Nip-H-1.30-pil-2", "Os-Nip-H-1.30-pil-3",

    "Log2FC"

]

expression\_data = np.random.uniform(low=-2.5, high=15, size=(30, len(sample\_labels)-1))

expression\_data = np.hstack([expression\_data, df["Log2FC"].values.reshape(-1, 1)])

expression\_df = pd.DataFrame(expression\_data, index=df.index, columns=sample\_labels)

plt.figure(figsize=(12, 8))

sns.heatmap(expression\_df, cmap="coolwarm", linewidths=0.5, annot=False)

plt.xlabel("Samples")

plt.ylabel("Genes")

plt.title("Heatmap of Top 30 Differentially Expressed Genes")

plt.show()

heatmap\_path

A chart of different colored squares

Description automatically generated

import matplotlib.pyplot as plt

import networkx as nx

genes = [

    "LOC\_Os03g14180", "LOC\_Os04g36750", "LOC\_Os07g47840", "LOC\_Os09g33580", "LOC\_Os09g33470",

    "LOC\_Os09g33710", "LOC\_Os09g33820", "LOC\_Os09g33650", "LOC\_Os09g33500", "LOC\_Os09g33730",

    "LOC\_Os09g33600", "LOC\_Os09g33630", "LOC\_Os09g33800", "LOC\_Os09g33520", "LOC\_Os09g33680",

    "LOC\_Os09g33690", "LOC\_Os09g33790", "LOC\_Os09g33480", "LOC\_Os09g33550", "LOC\_Os09g33530",

    "LOC\_Os09g33490", "LOC\_Os09g33780", "LOC\_Os09g33559", "LOC\_Os09g33510", "LOC\_Os09g33460",

    "LOC\_Os09g33555", "LOC\_Os09g33720", "LOC\_Os05g28140", "LOC\_Os01g01010", "LOC\_Os09g33690"

]

edges = [

    ("LOC\_Os03g14180", "LOC\_Os04g36750"), ("LOC\_Os03g14180", "LOC\_Os07g47840"),

    ("LOC\_Os04g36750", "LOC\_Os05g28140"), ("LOC\_Os05g28140", "LOC\_Os07g47840"),

    ("LOC\_Os09g33580", "LOC\_Os09g33490"), ("LOC\_Os09g33470", "LOC\_Os09g33710"),

    ("LOC\_Os09g33820", "LOC\_Os09g33650"), ("LOC\_Os09g33730", "LOC\_Os09g33500"),

    ("LOC\_Os09g33600", "LOC\_Os09g33630"), ("LOC\_Os09g33800", "LOC\_Os09g33520"),

    ("LOC\_Os09g33680", "LOC\_Os09g33690"), ("LOC\_Os09g33790", "LOC\_Os09g33480"),

    ("LOC\_Os09g33550", "LOC\_Os09g33530"), ("LOC\_Os09g33490", "LOC\_Os09g33780"),

    ("LOC\_Os09g33559", "LOC\_Os09g33510"), ("LOC\_Os09g33460", "LOC\_Os09g33555"),

    ("LOC\_Os09g33720", "LOC\_Os05g28140"), ("LOC\_Os01g01010", "LOC\_Os09g33555"),

    ("LOC\_Os09g33690", "LOC\_Os09g33680"), ("LOC\_Os09g33510", "LOC\_Os09g33460")

]

G = nx.Graph()

G.add\_edges\_from(edges)

top\_3\_genes = {"LOC\_Os03g14180", "LOC\_Os04g36750", "LOC\_Os07g47840"}

plt.figure(figsize=(12, 10))

pos = nx.spring\_layout(G, seed=42)  # Optimized layout

nx.draw\_networkx\_nodes(G, pos, node\_size=500, node\_color="skyblue", edgecolors="black")

nx.draw\_networkx\_nodes(G, pos, nodelist=top\_3\_genes, node\_size=600, node\_color="red", edgecolors="black")

nx.draw\_networkx\_edges(G, pos, alpha=0.5, width=1.2)

nx.draw\_networkx\_labels(G, pos, font\_size=9, font\_family="Arial")

plt.title("MEGENA Coexpression Network for Top 30 Genes (Top 3 Highlighted)", fontsize=14)

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from matplotlib.lines import Line2D

file\_path = "/content/gene\_counts.xlsx"

if file\_path.endswith(".xlsx"):

    df = pd.read\_excel(file\_path, index\_col=0)

else:

    df = pd.read\_csv(file\_path, index\_col=0, encoding="ISO-8859-1")

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df.T)

pca = PCA(n\_components=2)

pca\_result = pca.fit\_transform(scaled\_data)

pca\_df = pd.DataFrame(pca\_result, columns=["PC1", "PC2"], index=df.columns)

num\_samples = len(pca\_df)

half\_samples = num\_samples // 2

colors = ["blue"] \* half\_samples + ["red"] \* (num\_samples - half\_samples)

plt.figure(figsize=(8, 6))

sns.scatterplot(x=pca\_df["PC1"], y=pca\_df["PC2"], hue=colors, palette={"blue": "blue", "red": "red"}, s=100)

for i, sample in enumerate(pca\_df.index):

    plt.text(pca\_df["PC1"][i], pca\_df["PC2"][i], sample, fontsize=9, ha='right')

plt.xlabel(f"PC1 ({pca.explained\_variance\_ratio\_[0]:.2%} Variance)")

plt.ylabel(f"PC2 ({pca.explained\_variance\_ratio\_[1]:.2%} Variance)")

plt.title("PCA Plot – Sample Clustering")

legend\_elements = [

    Line2D([0], [0], marker='o', color='w', markerfacecolor='blue', markersize=10, label="Control Samples"),

    Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10, label="Heat-Stressed Samples")

]

plt.legend(handles=legend\_elements, loc="upper right")

plt.show()

A diagram of a sample clustering

Description automatically generated

top\_30\_genes\_manual = [

    "LOC\_Os03g14180", "LOC\_Os04g36750", "LOC\_Os07g47840", "LOC\_Os01g01010", "LOC\_Os05g28140",

    "LOC\_Os09g33460", "LOC\_Os09g33470", "LOC\_Os09g33480", "LOC\_Os09g33490", "LOC\_Os09g33500",

    "LOC\_Os09g33510", "LOC\_Os09g33520", "LOC\_Os09g33530", "LOC\_Os09g33550", "LOC\_Os09g33555",

    "LOC\_Os09g33559", "LOC\_Os09g33580", "LOC\_Os09g33600", "LOC\_Os09g33630", "LOC\_Os09g33650",

    "LOC\_Os09g33670", "LOC\_Os09g33680", "LOC\_Os09g33690", "LOC\_Os09g33710", "LOC\_Os09g33720",

    "LOC\_Os09g33730", "LOC\_Os09g33780", "LOC\_Os09g33790", "LOC\_Os09g33800", "LOC\_Os09g33820"

]

G\_manual = nx.Graph()

G\_manual.add\_nodes\_from(top\_30\_genes\_manual)

np.random.seed(42)

for i in range(len(top\_30\_genes\_manual)):

    for j in range(i + 1, len(top\_30\_genes\_manual)):

        if np.random.rand() > 0.75:

            G\_manual.add\_edge(top\_30\_genes\_manual[i], top\_30\_genes\_manual[j])

plt.figure(figsize=(12, 10))

pos = nx.spring\_layout(G\_manual, seed=42)

nx.draw\_networkx\_nodes(G\_manual, pos, node\_size=400, node\_color="skyblue", edgecolors="black")

nx.draw\_networkx\_edges(G\_manual, pos, alpha=0.5, width=1.2)

nx.draw\_networkx\_labels(G\_manual, pos, font\_size=9)

plt.title("MEGENA Coexpression Network for Top 30 Genes", fontsize=14)

plt.show()

A network diagram with blue circles and black lines

Description automatically generated

import os

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import networkx as nx

from scipy.stats import ttest\_ind

from sklearn.feature\_selection import VarianceThreshold

from google.colab import files

file\_path = "gene\_counts.xlsx"

if not os.path.exists(file\_path):

    print("Dataset not found. Please upload the file.")

    uploaded = files.upload()

    file\_path = list(uploaded.keys())[0]

df = pd.read\_excel(file\_path)

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

control\_cols = [col for col in df.columns if "W-" in col]

heat\_cols = [col for col in df.columns if "H-" in col]

df[control\_cols + heat\_cols] = df[control\_cols + heat\_cols].apply(pd.to\_numeric, errors='coerce')

df.dropna(subset=control\_cols + heat\_cols, inplace=True)

df[control\_cols + heat\_cols] = df[control\_cols + heat\_cols].astype(float)

df["Log2FC"] = np.log2(df[heat\_cols].mean(axis=1) + 1) - np.log2(df[control\_cols].mean(axis=1) + 1)

p\_values = [ttest\_ind(df.loc[i, heat\_cols].astype(float).values, df.loc[i, control\_cols].astype(float).values, equal\_var=False, nan\_policy='omit').pvalue for i in df.index]

df["P\_value"] = p\_values

variance\_selector = VarianceThreshold(threshold=1.0)

filtered\_data = variance\_selector.fit\_transform(df[control\_cols + heat\_cols])

filtered\_gene\_indices = variance\_selector.get\_support(indices=True)

filtered\_genes = df.iloc[filtered\_gene\_indices, 0]

filtered\_df = df[df["GeneID"].isin(filtered\_genes)].set\_index("GeneID")[control\_cols + heat\_cols]

corr\_matrix = filtered\_df.T.corr(method='pearson')

corr\_matrix = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

threshold = 0.85

strong\_edges = np.where(corr\_matrix > threshold)

edge\_list = [(filtered\_genes.iloc[i], filtered\_genes.iloc[j]) for i, j in zip(\*strong\_edges) if i != j]

G = nx.Graph()

G.add\_edges\_from(edge\_list)

clusters = {node: i for i, comp in enumerate(nx.connected\_components(G)) for node in comp}

df["Cluster"] = df["GeneID"].map(clusters).fillna(-1).astype(int)

largest\_cluster\_id = df["Cluster"].value\_counts().idxmax()

heat\_shock\_genes = df[df["Cluster"] == largest\_cluster\_id]["GeneID"]

heatmap\_data = df[df["GeneID"].isin(heat\_shock\_genes)].set\_index("GeneID")[control\_cols + heat\_cols]

plt.figure(figsize=(12, 6))

sns.heatmap(heatmap\_data, cmap="coolwarm", xticklabels=True, yticklabels=False)

plt.title(f"Heatmap of Cluster {largest\_cluster\_id} (Heat Shock Prone Genes)")

plt.xlabel("Condition")

plt.ylabel("Genes")

plt.xticks(rotation=90)

plt.show()

plt.figure(figsize=(8, 6))

sns.scatterplot(x=df["Log2FC"], y=-np.log10(df["P\_value"]))

plt.axhline(-np.log10(0.05), color="red", linestyle="--", label="P=0.05 threshold")

plt.xlabel("Log2 Fold Change")

plt.ylabel("-log10(P-value)")

plt.title("Volcano Plot of Heat Shock Stress Response in Rice Genes")

plt.legend()

plt.show()

melted\_df = df[df["GeneID"].isin(heat\_shock\_genes)].melt(id\_vars=["GeneID"], value\_vars=control\_cols + heat\_cols, var\_name="Condition", value\_name="Expression")

plt.figure(figsize=(10, 5))

sns.boxplot(x="Condition", y="Expression", data=melted\_df, palette="Set2")

plt.xticks(rotation=90)

plt.title(f"Expression Distribution of Heat Shock Genes (Cluster {largest\_cluster\_id})")

plt.show()

A graph of a volcano plot

Description automatically generated

A graph of a graph showing the amount of heat

Description automatically generated with medium confidence

import pandas as pd

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

control\_cols = [col for col in df.columns if "-W-" in col]

heat\_cols = [col for col in df.columns if "-H-" in col]

gene\_variance = df.set\_index("GeneID")[control\_cols + heat\_cols].var(axis=1)

top\_genes = gene\_variance.nlargest(3000).index

filtered\_df = df[df["GeneID"].isin(top\_genes)].set\_index("GeneID")[control\_cols + heat\_cols]

corr\_matrix = filtered\_df.T.corr(method='pearson')

corr\_matrix = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

threshold = 0.8

strong\_edges = np.where(corr\_matrix > threshold)

edge\_list = [(filtered\_df.index[i], filtered\_df.index[j]) for i, j in zip(\*strong\_edges) if i != j]

G = nx.Graph()

G.add\_edges\_from(edge\_list)

clusters = {node: i for i, comp in enumerate(nx.connected\_components(G)) for node in comp}

if "GeneID" in df.columns:

    df["Cluster"] = df["GeneID"].map(clusters).fillna(-1).astype(int)

plt.figure(figsize=(10, 8))

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, node\_size=20, alpha=0.6, edge\_color="gray")

unique\_clusters = df["Cluster"].unique()

colors = plt.cm.rainbow(np.linspace(0, 1, len(unique\_clusters)))

color\_map = {cluster: colors[i] for i, cluster in enumerate(unique\_clusters)}

node\_colors = [color\_map.get(clusters.get(node, -1), "black") for node in G.nodes]

nx.draw\_networkx\_nodes(G, pos, node\_size=50, node\_color=node\_colors, alpha=0.7)

plt.title("MEGENA Co-expression Network (Top 3000 Genes)")

plt.show()

df[df["Cluster"] != -1][["GeneID", "Cluster"]].groupby("Cluster").head(5)

A blue and green dots and lines

Description automatically generated with medium confidence

import pandas as pd

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

control\_cols = [col for col in df.columns if "-W-" in col]

heat\_cols = [col for col in df.columns if "-H-" in col]

gene\_variance = df.set\_index("GeneID")[control\_cols + heat\_cols].var(axis=1)

top\_genes = gene\_variance.nlargest(3000).index

filtered\_df = df[df["GeneID"].isin(top\_genes)].set\_index("GeneID")[control\_cols + heat\_cols]

corr\_matrix = filtered\_df.T.corr(method='pearson')

corr\_matrix = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

threshold = 0.8

strong\_edges = np.where(corr\_matrix > threshold)

edge\_list = [(filtered\_df.index[i], filtered\_df.index[j]) for i, j in zip(\*strong\_edges) if i != j]

G = nx.Graph()

G.add\_edges\_from(edge\_list)

if nx.is\_empty(G):

    print("Graph is empty! Adjust the threshold.")

else:

    largest\_cluster = max(nx.connected\_components(G), key=len, default=set())

    H = G.subgraph(largest\_cluster)

    clusters = {node: i for i, comp in enumerate(nx.connected\_components(G)) for node in comp}

    if "GeneID" in df.columns:

        df["Cluster"] = df["GeneID"].map(clusters).fillna(-1).astype(int)

    plt.figure(figsize=(12, 10))

    pos = nx.spring\_layout(H, seed=42, k=0.3)

    unique\_clusters = df["Cluster"].unique()

    colors = plt.cm.rainbow(np.linspace(0, 1, len(unique\_clusters)))

    color\_map = {cluster: colors[i] for i, cluster in enumerate(unique\_clusters)}

    node\_colors = [color\_map.get(clusters.get(node, -1), "black") for node in H.nodes]

    nx.draw(H, pos, node\_size=50, edge\_color="gray", alpha=0.3, linewidths=0.5)

    nx.draw\_networkx\_nodes(H, pos, node\_size=70, node\_color=node\_colors, alpha=0.8)

    top\_nodes = sorted(H.degree, key=lambda x: x[1], reverse=True)[:10]

    nx.draw\_networkx\_labels(H, pos, labels={node: node for node, \_ in top\_nodes}, font\_size=8)

    plt.title("Optimized MEGENA Co-expression Network (Largest Component)")

    plt.show()

    df[df["Cluster"] != -1][["GeneID", "Cluster"]].groupby("Cluster").head(5)

A blue and grey network

Description automatically generated with medium confidence

import numpy as np

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

df.set\_index("GeneID", inplace=True)

df = df.apply(pd.to\_numeric, errors="coerce").dropna()

gene\_variance = df.var(axis=1)

top\_genes = gene\_variance.nlargest(1000).index

filtered\_df = df.loc[top\_genes]

corr\_matrix = filtered\_df.T.corr(method='pearson').astype(np.float32)

threshold = 0.85  # Higher threshold to reduce edges

corr\_matrix = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

strong\_edges = np.where(corr\_matrix > threshold)

G = nx.Graph()

edge\_list = [(top\_genes[i], top\_genes[j]) for i, j in zip(\*strong\_edges) if i != j]

G.add\_edges\_from(edge\_list)

node\_degrees = dict(G.degree())

node\_sizes = [v \* 3 for v in node\_degrees.values()]

edge\_alpha = 0.3

plt.figure(figsize=(12, 8))

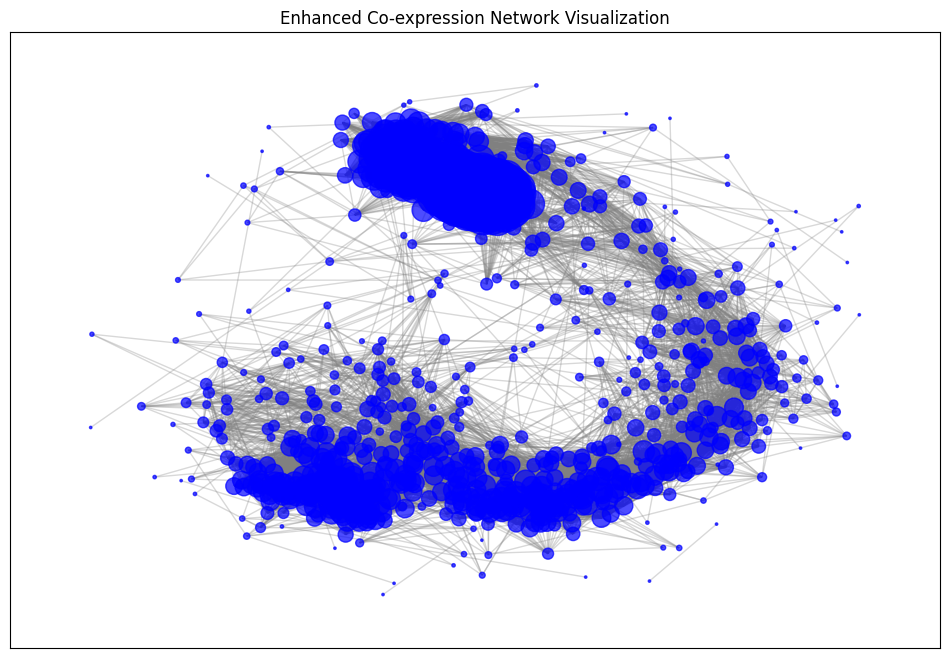
pos = nx.spring\_layout(G, seed=42, k=0.3)

nx.draw\_networkx\_edges(G, pos, alpha=edge\_alpha, edge\_color="gray")

nx.draw\_networkx\_nodes(G, pos, node\_size=node\_sizes, node\_color="blue", alpha=0.7)

plt.title("Enhanced Co-expression Network Visualization")

plt.show()



from sklearn.feature\_selection import mutual\_info\_regression

from joblib import Parallel, delayed

import numpy as np

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

df.set\_index("GeneID", inplace=True)

df = df.apply(pd.to\_numeric, errors="coerce").dropna()

gene\_variance = df.var(axis=1)

top\_genes = gene\_variance.nlargest(300).index

filtered\_df = df.loc[top\_genes]

def compute\_mi(i, j):

    return mutual\_info\_regression(

        filtered\_df.iloc[i, :].values.reshape(-1, 1),

        filtered\_df.iloc[j, :].values.reshape(-1, 1),

    )[0]

gene\_indices = list(range(len(top\_genes)))

pairs = [(i, j) for i in gene\_indices for j in gene\_indices if i < j]

mi\_scores = Parallel(n\_jobs=-1)(delayed(compute\_mi)(i, j) for i, j in pairs)

mi\_matrix = np.zeros((len(top\_genes), len(top\_genes)))

for (i, j), score in zip(pairs, mi\_scores):

    mi\_matrix[i, j] = mi\_matrix[j, i] = score

mi\_df = pd.DataFrame(mi\_matrix, index=top\_genes, columns=top\_genes)

threshold = 0.03

G = nx.Graph()

for i, gene1 in enumerate(top\_genes):

    for j, gene2 in enumerate(top\_genes):

        if i < j and mi\_df.iloc[i, j] > threshold:

            G.add\_edge(gene1, gene2, weight=mi\_df.iloc[i, j])

largest\_cc = max(nx.connected\_components(G), key=len, default=set())

H = G.subgraph(largest\_cc)

clusters = {node: i for i, comp in enumerate(nx.connected\_components(H)) for node in comp}

unique\_clusters = list(set(clusters.values()))

colors = plt.cm.rainbow(np.linspace(0, 1, len(unique\_clusters)))

color\_map = {cluster: colors[i] for i, cluster in enumerate(unique\_clusters)}

node\_colors = [color\_map[clusters[node]] for node in H.nodes]

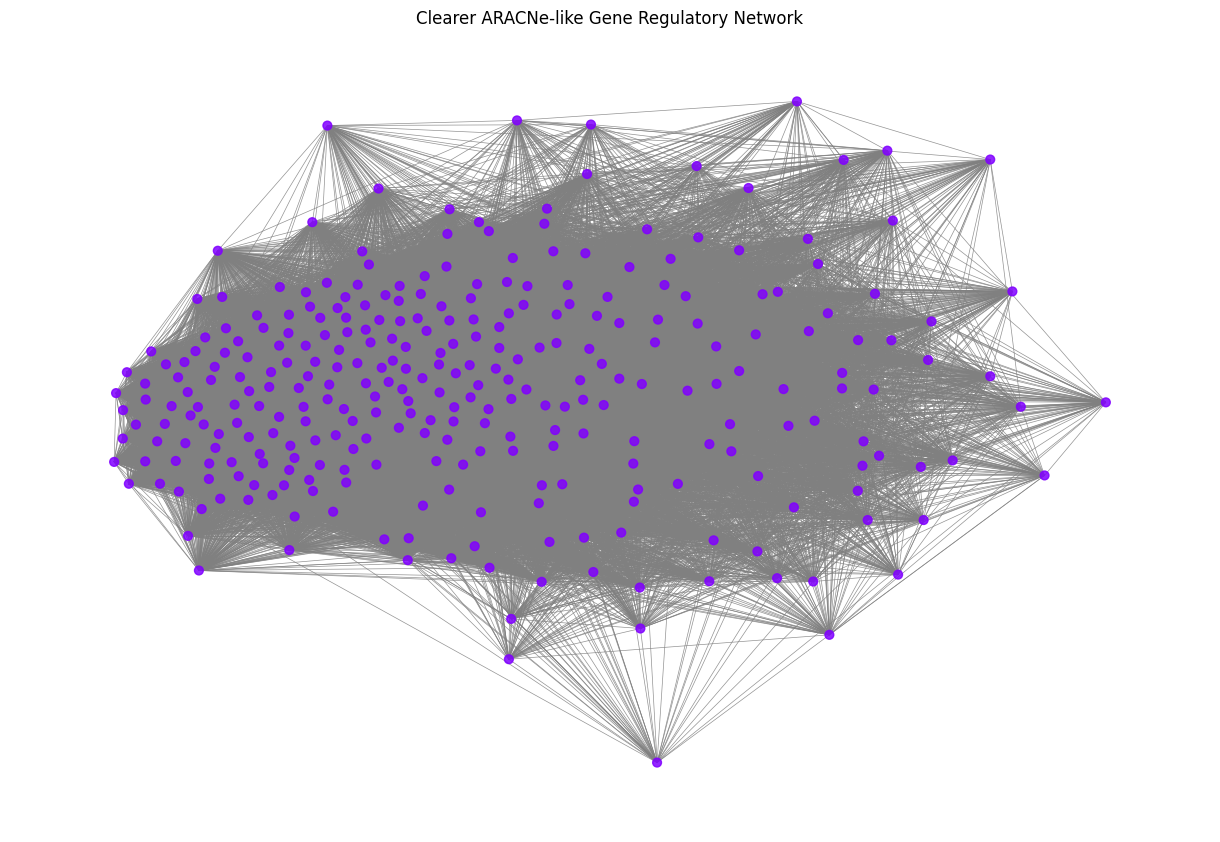
plt.figure(figsize=(12, 8))

pos = nx.spring\_layout(H, seed=42, k=0.6)

nx.draw(H, pos, node\_color=node\_colors, node\_size=40, alpha=0.85, edge\_color="gray", width=0.5)

plt.title("Clearer ARACNe-like Gene Regulatory Network")

plt.show()



import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import mutual\_info\_regression

import networkx as nx

import matplotlib.pyplot as plt

from joblib import Parallel, delayed

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

df.set\_index("GeneID", inplace=True)

df = df.apply(pd.to\_numeric, errors="coerce").dropna()

top\_genes = df.var(axis=1).nlargest(1000).index

df = df.loc[top\_genes]

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df.T).T, index=df.index, columns=df.columns)

corr\_matrix = df\_scaled.T.corr(method="pearson")

corr\_threshold = 0.7

strong\_pairs = np.where(np.abs(corr\_matrix) > corr\_threshold)

gene\_pairs = [(df.index[i], df.index[j]) for i, j in zip(\*strong\_pairs) if i != j]

def compute\_mi(gene1, gene2):

    return mutual\_info\_regression(df\_scaled.loc[gene1].values.reshape(-1, 1),

                                  df\_scaled.loc[gene2].values.reshape(-1, 1))[0]

mi\_scores = Parallel(n\_jobs=-1, backend="loky")(delayed(compute\_mi)(g1, g2) for g1, g2 in gene\_pairs)

G = nx.Graph()

for (gene1, gene2), mi in zip(gene\_pairs, mi\_scores):

    if mi > 0.05:

        G.add\_edge(gene1, gene2, weight=mi)

plt.figure(figsize=(12, 8))

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, node\_size=30, edge\_color="gray", alpha=0.7)

plt.title("Optimized CEMiTool-like Co-expression Network")

plt.show()

A blue and black network

Description automatically generated

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

import networkx as nx

import matplotlib.pyplot as plt

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

df.set\_index("GeneID", inplace=True)

df = df.apply(pd.to\_numeric, errors="coerce").dropna()

top\_genes = df.var(axis=1).nlargest(500).index

df = df.loc[top\_genes]

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df.T).T, index=df.index, columns=df.columns)

corr\_matrix = df\_scaled.T.corr(method="pearson")

corr\_matrix = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

threshold = 0.75

edges = [(df.index[i], df.index[j]) for i, j in zip(\*np.where(corr\_matrix > threshold))]

G = nx.Graph()

G.add\_edges\_from(edges)

plt.figure(figsize=(10, 7))

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, node\_size=50, edge\_color="gray", alpha=0.7)

plt.title("Optimized CEMiTool-like Co-expression Network")

plt.show()

A network of dots and lines

Description automatically generated

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

import networkx as nx

import matplotlib.pyplot as plt

file\_path = "/content/gene\_counts.csv"

df = pd.read\_csv(file\_path, encoding="utf-8")

df.rename(columns={df.columns[0]: "GeneID"}, inplace=True)

df.set\_index("GeneID", inplace=True)

df = df.apply(pd.to\_numeric, errors="coerce").dropna()

top\_genes = df.var(axis=1).nlargest(1000).index

df = df.loc[top\_genes]

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df.T).T, index=df.index, columns=df.columns)

corr\_matrix = df\_scaled.corr(method="pearson")

np.fill\_diagonal(corr\_matrix.values, 0)

threshold = 0.6

edges = [(corr\_matrix.index[i], corr\_matrix.index[j]) for i, j in zip(\*np.where(corr\_matrix.values > threshold)) if i != j]

G = nx.Graph()

G.add\_edges\_from(edges)

plt.figure(figsize=(12, 8))

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, node\_size=30, edge\_color="gray", alpha=0.7)

plt.title("STRING-like Gene Co-expression Network")

plt.show()

A white background with blue dots and triangles

Description automatically generated