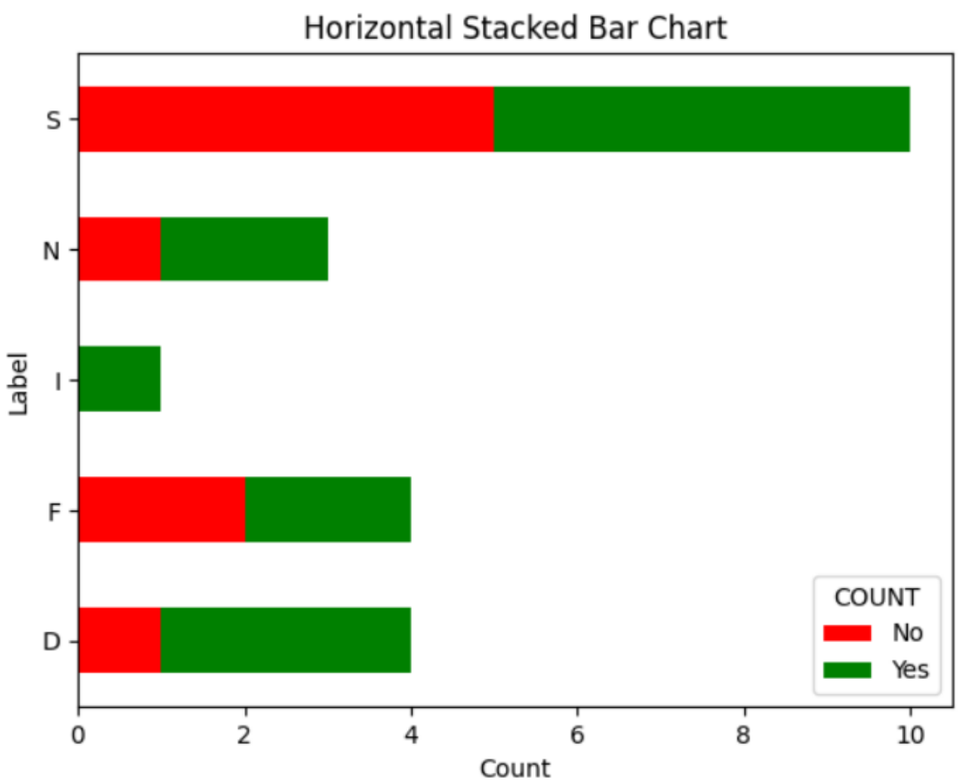
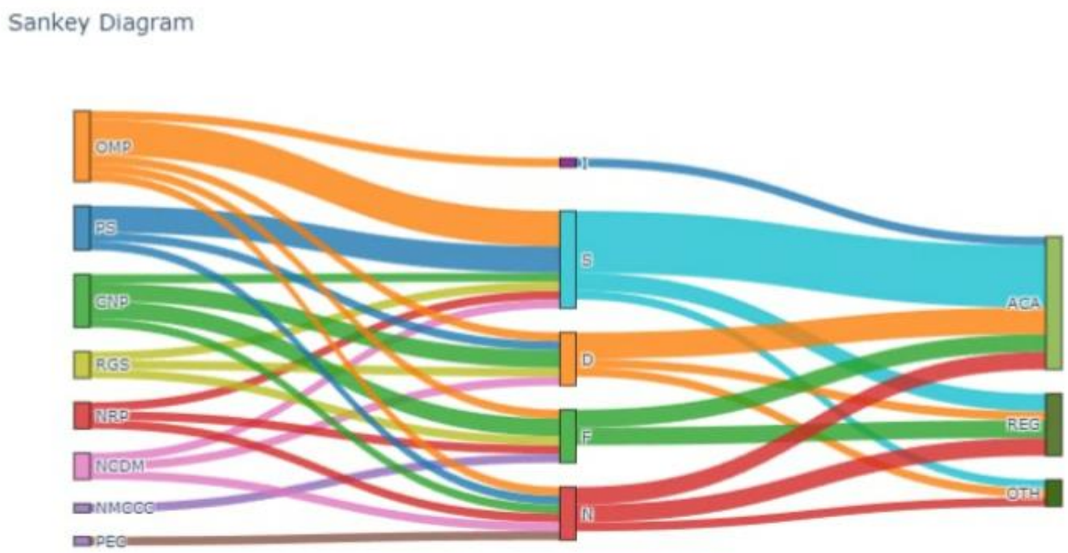
```python
import pandas as pd
import matplotlib.pyplot as plt

# Read the data from the CSV file
df = pd.read_csv('bar_assignment.csv')

# Transform 1 into "Yes" and 0 into "No"
df['COUNT'] = df['COUNT'].map({1: 'Yes', 0: 'No'})

# Create a crosstab to prepare for the stacked bar chart
crosstab = pd.crosstab(df['LABEL'], df['COUNT'])

# Plot the horizontal stacked bar chart
crosstab.plot(kind='barh', stacked=True, color=['red', 'green'])
plt.xlabel('Count')
plt.ylabel('Label')
plt.title('Horizontal Stacked Bar Chart')
plt.legend(title='COUNT')
plt.show()
```




```python
import pandas as pd
import plotly.graph_objects as go

# Load CSV
df = pd.read_csv('sankey_assignment.csv', index_col=0)

# Clean column and index names
df.columns = df.columns.str.strip().str.upper()
df.index = df.index.str.strip().str.upper()

sources = ['PS', 'OMP', 'CNP', 'NRP', 'NMCCC', 'PEC', 'NCDM', 'RGS']
middle_labels = ['S', 'I', 'D', 'F', 'N']
targets = ['REG', 'ACA', 'OTH']

# Extract connections from sources to middle labels
data = []
for middle in middle_labels:
    for source in sources:
        value = df.at[middle, source] if source in df.columns else 0
        if value > 0:
            data.append((source, middle, value))

# Extract connections from middle labels to targets, this is for the blocks in the middle
for middle in middle_labels:
    for target in targets:
        value = df.at[middle, target] if target in df.columns else 0
        if value > 0:
            data.append((middle, target, value))

# Create label mapping
labels = sources + middle_labels + targets
label_map = {label: i for i, label in enumerate(labels)}
```

```
# Define custom colors for bars
node_colors = {
  'PS': 'rgba(31, 119, 180, 0.8)',
    'CNP': 'rgba(44, 160, 44, 0.8)',
    'OMP': 'rgba(255, 127, 14, 0.8)',
    'NRP': 'rgba(214, 39, 40, 0.8)',
    'NMCCC': 'rgba(148, 103, 189, 0.8)',
    'PEC': 'rgba(148, 103, 189, 0.8)',
    'NCDM': 'rgba(227, 119, 194, 0.8)',
    'RGS':'rgba(188, 189, 34, 0.8)',
    'D': 'rgba(255, 127, 14, 0.8)',
    'N': 'rgba(214, 39, 40, 0.8)',
    'F': 'rgba(44, 160, 44, 0.8)',
    'I': 'rgba(128, 0, 128, 0.8)',
    'S': 'rgba(23, 190, 207, 0.8)',
    'ACA': '#98BF63',
    'REG': '#607D3B',
    'OTH': ' #466D1D'

}

# Assign node colors
node_color_list = [node_colors.get(label, 'lightgray') for label in labels]  # Default to lightgray

# Assign link colors
colors = [
    'rgba(31, 119, 180, 0.8)', 'rgba(255, 127, 14, 0.8)', 'rgba(44, 160, 44, 0.8)',
    'rgba(214, 39, 40, 0.8)', 'rgba(148, 103, 189, 0.8)', 'rgba(140, 86, 75, 0.8)',
    'rgba(227, 119, 194, 0.8)', 'rgba(188, 189, 34, 0.8)', 'rgba(23, 190, 207, 0.8)'
]

def get_color(index):
    return colors[index % len(colors)]

link_colors = [get_color(label_map[s]) for s, t, v in data]

# Create Sankey diagram
fig = go.Figure(go.Sankey(
    node=dict(
        pad=15, thickness=10, line=dict(color='black', width=0.5),
        label=labels, color=node_color_list  # Apply custom node colors
    ),
    link=dict(
        source=[label_map[s] for s, t, v in data],
        target=[label_map[t] for s, t, v in data],
        value=[v for s, t, v in data],
        color=link_colors
    )
))

fig.update_layout(title_text="Sankey Diagram", font_size=12)
fig.show()
```



```python
import plotly.graph_objects as go
import pandas as pd
import networkx as nx
import numpy as np

# Load adjacency matrix from CSV
df = pd.read_csv("networks_assignment.csv", index_col=0)

# Convert to edge list
edges = df.stack().reset_index()
edges.columns = ["source", "target", "weight"]
edges = edges[edges["weight"] > 0]

# Create graph
G = nx.from_pandas_edgelist(edges, source="source", target="target", edge_attr="weight")

# Define node groups
pentagram_nodes = {'D', 'F', 'I', 'N', 'S'}
blue_nodes = pentagram_nodes
green_nodes = {'BIH', 'GEO', 'ISR', 'MNE', 'SRB', 'CHE', 'TUR', 'UKR', 'GBR', 'AUS', 'HKG', 'USA'}
yellow_nodes = {'AUT', 'BEL', 'BGR', 'HRV', 'CZE', 'EST', 'FRA', 'DEU', 'GRC', 'HUN', 'IRL', 'ITA', 'LVA', 'LUX',
'NLD', 'PRT', 'ROU', 'SVK', 'SVN', 'ESP'}

# Assign colors
node_colors = {node: ("#728FCE" if node in blue_nodes else
                      "#50C878" if node in green_nodes else
                      "#FFE135" if node in yellow_nodes else "gray") for node in G.nodes}

# Define positions manually
pos = {}

# Set pentagram positions using a star shape
radius = 2
center_x, center_y = 0, 0
angles = np.linspace(np.pi / 2, np.pi / 2 + 2 * np.pi, 6)[:-1]

pentagram_positions = {node: (center_x + radius * np.cos(a), center_y + radius * np.sin(a))
                       for node, a in zip(pentagram_nodes, angles)}

pos.update(pentagram_positions)

# Outer nodes
outer_nodes = set(G.nodes) - pentagram_nodes
np.random.seed(42)

min_radius = 3
max_radius = 5
node_spacing = 0.9
placed_positions = []

for node in outer_nodes:
    while True:
        angle = np.random.uniform(0, 2 * np.pi)
        radius = np.random.uniform(min_radius, max_radius)
        new_pos = (radius * np.cos(angle), radius * np.sin(angle))

        if all(np.linalg.norm(np.array(new_pos) - np.array(existing)) > node_spacing for existing in
placed_positions):
            pos[node] = new_pos
            placed_positions.append(new_pos)
            break

# Extract node positions
node_x = [pos[n][0] for n in G.nodes]
node_y = [pos[n][1] for n in G.nodes]
node_colors_plot = [node_colors[n] for n in G.nodes]

# Extract edges
edge_x, edge_y, edge_hovertext = [], [], []
for edge in G.edges:
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_x.extend([x0, x1, None])
    edge_y.extend([y0, y1, None])
    edge_hovertext.append(f"{edge[0]} → {edge[1]}: {G[edge[0]][edge[1]]['weight']}")

# Create figure
fig = go.Figure()

# Add edges
fig.add_trace(go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=1, color='black'),
    mode='lines',
    hovertext=edge_hovertext,
    hoverinfo="text"
))

# Edge endpoint circles
# Adjust edge-end circle positioning to touch node borders precisely
# Adjust edge-end circle positioning to touch node borders precisely
edge_end_x = []
edge_end_y = []
```

```
# Fine-tuned node marker radius for precise placement
node_marker_radius = 28 * 0.012  # Slightly reduced scaling factor

for edge in G.edges:
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]

    # Compute edge direction
    dx, dy = x1 - x0, y1 - y0
    length = np.sqrt(dx**2 + dy**2)

    if length > 0:  # Avoid division by zero
        dx /= length
        dy /= length

        # Move circles even closer to the nodes
        edge_end_x.extend([x0 + dx * node_marker_radius, x1 - dx * node_marker_radius])
        edge_end_y.extend([y0 + dy * node_marker_radius, y1 - dy * node_marker_radius])

# Add small circular markers at edge endpoints
fig.add_trace(go.Scatter(
    x=edge_end_x, y=edge_end_y,
    mode='markers',
    marker=dict(size=6, color='black'),  # Small circles at edge endpoints
    hoverinfo='none',
    showlegend=False
))

# Add nodes
fig.add_trace(go.Scatter(
    x=node_x, y=node_y,
    mode='markers+text',
    marker=dict(size=27, color=node_colors_plot),
    text=[node if node in pentagram_nodes else "" for node in G.nodes],
    hovertext=[f"{node} (Degree: {G.degree[node]})" for node in G.nodes],
    hoverinfo="text",
    textposition="middle center",
    textfont=dict(size=12, color="white")
))

# Update layout
fig.update_layout(
    title="Network Graph with Pentagram Structure",
    showlegend=False,
    xaxis=dict(showgrid=False, zeroline=False, showticklabels=False, scaleanchor="y"),
    yaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
    plot_bgcolor='white',
    font_size=12,
    width=700,
    height=700
)

fig.show()
```