Data Visualization Assignment

by **Abangan, Jaerian Peter** & **Berbo, Christ Derek**

Objective

This documentation implements various **data visualization techniques** using the following datasets;

- **bar_assignment.csv** (Horizontal stacked bar chart)
- **sankey_assignment.csv** (Sankey diagram)
- **networks_assignment.csv** (Network graph)

Flow

1. Load and process the data using python with Plotly.
2. Generate the graph visualizations
    1. Bar Chart
    2. Sanky Diagram
    3. Network Graph
3. Analyze results.
4. Collate all the graph visualizations into 1 pdf.

Bar Graph Analysis

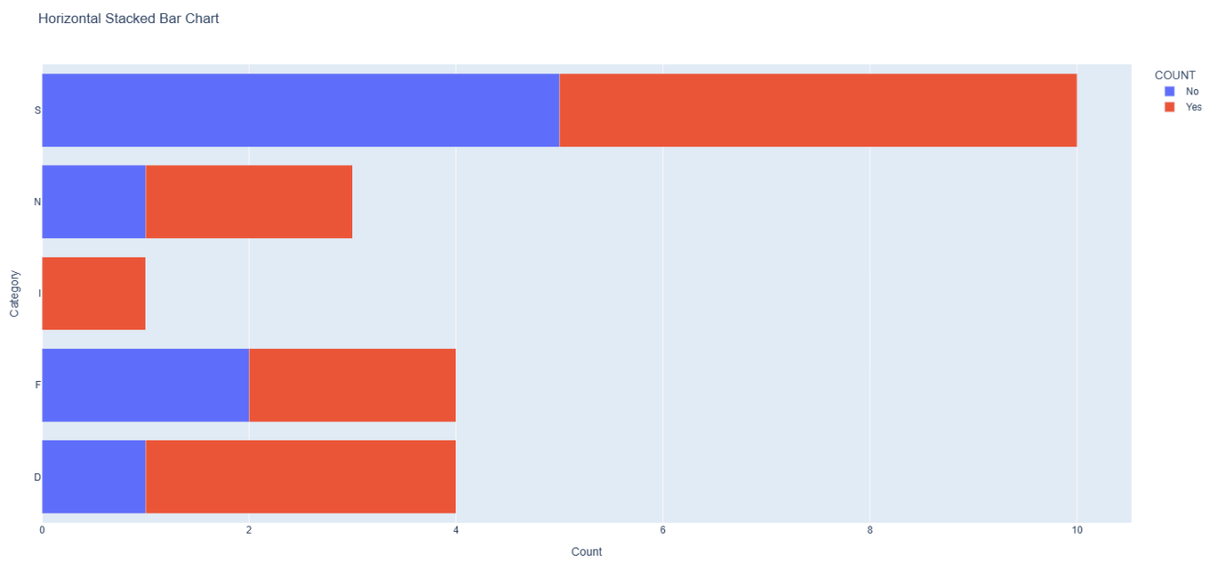
Objectives:

- Create a horizontal stacked bar chart
- Transform 1 into “Yes” and 0 into “No”
- Follow the plot specification for bar plot

Code Snippet:

```python
import pandas as pd
import plotly.express as px

# Get or Load Data set
bar_df = pd.read_csv("bar_assignment.csv")

# Transform 'COUNT': 1 -> "Yes", 0 -> "No"
bar_df["COUNT"] = bar_df["COUNT"].map({1: "Yes", 0: "No"})

# Aggregate counts for stacked bar plot
bar_plot_data = bar_df.groupby(["LABEL", "COUNT"]).size().unstack(fill_value=0)

# Create a horizontal stacked bar chart using Plotly
fig_bar = px.bar(
    bar_plot_data,
    orientation='h',
    title="Horizontal Stacked Bar Chart",
    labels={"value": "Count", "LABEL": "Category"},
)

# Update layout for consistent font and styling
fig_bar.update_layout(
    font=dict(family="Arial", size=12),
    barmode="stack"
)

# Show bar chart
fig_bar.show()
```

Graph Diagram:



Sankey Diagram Analysis

Objectives:

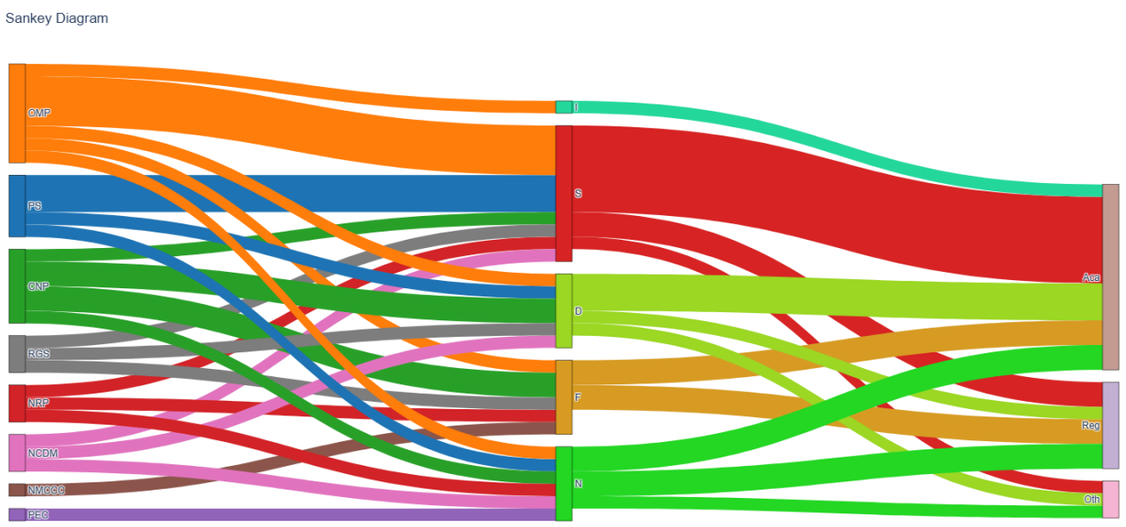
- Create a Sankey Diagram that connects ('PS', 'OMP', 'CNP', 'NRP', 'NMCCC', 'PEC', 'NCDM', 'RGS') to the LABELS to ('Reg', 'Aca', 'Oth')
- Follow the Path Specifications

Code Snippet:

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

# Load the Sankey dataset
sankey_df = pd.read_csv("sankey_assignment.csv")

# Define source and target nodes
source_cols = ['PS', 'OMP', 'CNP', 'NRP', 'NMCCC', 'PEC', 'NCDM', 'RGS']
target_cols = ['Reg', 'Aca', 'Oth']

# Define unique colors for nodes
node_color_map = {
    "PS": "#1f77b4",    # Blue
    "OMP": "#ff7f0e",   # Orange
    "CNP": "#2ca02c",   # Green
    "NRP": "#d62728",   # Red
    "NMCCC": "#8c564b", # Brown
    "PEC": "#9467bd",   # Purple
    "NCDM": "#e377c2",  # Pink
    "RGS": "#7f7f7f",   # Gray
    "Reg": "#c5b0d5",   # Light Purple
    "Aca": "#c49c94",   # Beige
    "Oth": "#f7b6d2",   # Light Pink
}

# Assign colors for middle nodes dynamically
middle_nodes = sankey_df["LABEL"].unique()
for i, label in enumerate(middle_nodes):
    node_color_map[label] = f"hsl({(i * 40) % 360}, 70%, 50%)"  # Auto-generate colors

# Create a dictionary to accumulate flows from one node to another
flow_dict = {}

# Step 1: Connect source columns to the middle node (LABEL)
for index, row in sankey_df.iterrows():
    mid_node = row['LABEL']
    for src in source_cols:
        value = row[src]
        if value > 0:
            key = (src, mid_node)
            flow_dict[key] = flow_dict.get(key, 0) + value

# Step 2: Connect middle node (LABEL) to target columns
for index, row in sankey_df.iterrows():
    mid_node = row['LABEL']
    for tgt in target_cols:
        value = row[tgt]
        if value > 0:
            key = (mid_node, tgt)
            flow_dict[key] = flow_dict.get(key, 0) + value

# Create a list of unique nodes from all links
nodes = set()
for (src, tgt), val in flow_dict.items():
    nodes.add(src)
    nodes.add(tgt)
nodes = list(nodes)

# Assign colors based on node type
node_colors = [node_color_map.get(node, "lightgray") for node in nodes]

# Create a mapping from node name to index for Plotly
node_to_idx = {node: i for i, node in enumerate(nodes)}

# Build the lists for the Sankey diagram links
link_source = []
link_target = []
link_value = []
link_color = []

for (src, tgt), val in flow_dict.items():
    link_source.append(node_to_idx[src])
    link_target.append(node_to_idx[tgt])
    link_value.append(val)
    link_color.append(node_color_map.get(src, "rgba(150,150,150,0.6)"))  # Match link color to source node

# Create the Sankey diagram
fig_sankey = go.Figure(go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=nodes,
        color=node_colors
    ),
    link=dict(
        source=link_source,
        target=link_target,
        value=link_value,
        color=link_color
    )
))
```

```
93
94  # Update the layout with title and consistent font
95  fig_sankey.update_layout(
96      title_text="Sankey Diagram",
97      font=dict(family="Arial", size=12)
98  )
99
100 # Display the Sankey diagram
101 fig_sankey.show()
```

Graph Diagram:



Network Graph Analysis

Objectve:

- Create the network graph
- D,F,I,N,S should created as a pentagram located at the center of the graph showing connection with each other.
- The others should be outside of the the pentagram, still showing connections to other nodes.
- The node color should be:

Blue: [D,F,I,N,S], Green: ['BIH', 'GEO', 'ISR', 'MNE', 'SRB', 'CHE', 'TUR', 'UKR', 'GBR', 'AUS', 'HKG', 'USA'], Yellow: ['AUT', 'BEL', 'BGR', 'HRV', 'CZE', 'EST', 'FRA', 'DEU', 'GRC', 'HUN', 'IRL', 'ITA', 'LVA', 'LUX', 'NLD', 'PRT', 'ROU', 'SVK', 'SVN', 'ESP']

Code Snippet:

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

# Load dataset
file_path = "networks_assignment.csv"  # Update the path if needed
df = pd.read_csv(file_path, index_col=0)  # Set first column as index

# Define core and peripheral nodes
core_nodes = ["D", "F", "I", "N", "S"]
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"]

# Create graph
G = nx.Graph()

# Add all core nodes to the graph
for node in core_nodes:
    G.add_node(node)

# Add edges from dataset
for src, row in df.iterrows():
    for tgt, weight in row.items():
        if weight > 0:
            G.add_edge(src, tgt, weight=weight)

# Define node positions
def get_positions(core_nodes, peripheral_nodes, core_radius=1.5, peripheral_radius=2.5):
    pos = {}

    # Place core nodes in a pentagon
    pentagon_order = [0, 1, 2, 3, 4]  # Regular order
    for i, index in enumerate(pentagon_order):
        angle = index * (2 * np.pi / 5)  # Divide circle into 5 parts
        pos[core_nodes[i]] = (core_radius * np.cos(angle), core_radius * np.sin(angle))

    # Place peripheral nodes in a circle
    angle_step = 2 * np.pi / len(peripheral_nodes)
    for i, node in enumerate(peripheral_nodes):
        pos[node] = (peripheral_radius * np.cos(i * angle_step), peripheral_radius * np.sin(i * angle_step))

    return pos

peripheral_nodes = set(G.nodes) - set(core_nodes)
pos = get_positions(core_nodes, peripheral_nodes)

# **Manually add pentagram edges** (star inside pentagon)
pentagram_edges = [(core_nodes[i], core_nodes[(i + 2) % 5]) for i in range(5)]

# **Manually add pentagon edges** (outer shape of core nodes)
pentagon_edges = [(core_nodes[i], core_nodes[(i + 1) % 5]) for i in range(5)]

# Combine both sets of blue edges
blue_edges = pentagram_edges + pentagon_edges

# Remove black edges that connect to core nodes
black_edges = [edge for edge in G.edges() if edge not in blue_edges]

# Assign colors
node_color_map = {
    "core": "blue",
    "green": "green",
    "yellow": "yellow",
    "default": "gray"
}
node_colors = []
for node in G.nodes:
    if node in core_nodes:
        node_colors.append(node_color_map["core"])
    elif node in green_nodes:
        node_colors.append(node_color_map["green"])
    elif node in yellow_nodes:
        node_colors.append(node_color_map["yellow"])
    else:
        node_colors.append(node_color_map["default"])

# Create edge traces (black for normal edges, blue for pentagram + pentagon)
def create_edge_trace(edges, color, width):
    edge_x = []
    edge_y = []
    for edge in edges:
        x0, y0 = pos[edge[0]]
        x1, y1 = pos[edge[1]]
        edge_x.extend([x0, x1, None])
        edge_y.extend([y0, y1, None])
    return go.Scatter(
        x=edge_x, y=edge_y,
        line=dict(width=width, color=color),  # Set custom width
        hoverinfo="none",
        mode="lines"
    )
```

```
edge_trace_black = create_edge_trace(black_edges, "black", 1.5)  # Normal edges (thinner)
edge_trace_blue = create_edge_trace(blue_edges, "blue", 3)  # Pentagram edges (thicker)


# Create node traces
node_x = [pos[node][0] for node in G.nodes]
node_y = [pos[node][1] for node in G.nodes]
node_text = list(G.nodes)

node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode="markers+text",
    marker=dict(size=10, color=node_colors),
    text=node_text,
    textposition="top center",
    hoverinfo="text"
)

# Create figure
fig = go.Figure(data=[edge_trace_black, edge_trace_blue, node_trace])
fig.update_layout(
    title="Network Graph with Pentagram and Pentagon",
    showlegend=False,
    hovermode="closest",
    margin=dict(b=0, l=0, r=0, t=40),
    xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
    yaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
    width=800,
    height=600
)

fig.show()
```

Graph Diagram: