## Data Preprocessing

import pandas as pd

# mental-illnesses-prevalence  
df = pd.read\_csv('mental-illnesses-prevalence.csv') # Extract data in 2019  
df\_2019 = df[df['Year'] == 2019] # Save the result as a new CSV file  
df\_2019.to\_csv('mental-illnesses-prevalence\_2019.csv', index=False)

# depressive-disorders-prevalence-vs-gdp-per-capita  
df1 = pd.read\_csv('depressive-disorders-prevalence-vs-gdp-per-capita.csv') df1\_2019 = df1[df1['Year'] == 2019]   
df1\_2019.to\_csv('depressive-disorders-prevalence-vs-gdp-per-capita\_2019.csv', index=False)

# Read the three CSV files  
df\_2019 = pd.read\_csv('mental-illnesses-prevalence\_2019.csv')  
df1\_2019 = pd.read\_csv('depressive-disorders-prevalence-vs-gdp-per-capita\_2019.csv')  
df2 = pd.read\_csv('dealing-with-anxiety-depression-comparison.csv')  
df2 = df2.drop(columns=['Year'])  
  
# Rename the 'Entity' column to 'Country' in each DataFrame  
df\_2019.rename(columns={'Entity': 'Country'}, inplace=True)  
df1\_2019.rename(columns={'Entity': 'Country'}, inplace=True)  
df2.rename(columns={'Entity': 'Country'}, inplace=True)

# Drop rows where the 'code' column doesn't have data  
df\_2019.dropna(subset=['Code'], inplace=True)  
df1\_2019.dropna(subset=['Code'], inplace=True)  
df2.dropna(subset=['Code'], inplace=True)

df2.rename(columns={  
 'Share - Question: mh8b - Engaged in religious/spiritual activities when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Engaged\_in\_Religious\_Activities',  
 'Share - Question: mh8e - Improved healthy lifestyle behaviors when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Improved\_Healthy\_Lifestyle',  
 'Share - Question: mh8f - Made a change to work situation when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Changed\_Work\_Situation',  
 'Share - Question: mh8g - Made a change to personal relationships when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Changed\_Personal\_Relationships',  
 'Share - Question: mh8c - Talked to friends or family when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Talked\_to\_Friends\_Family',  
 'Share - Question: mh8d - Took prescribed medication when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Took\_Medication',  
 'Share - Question: mh8h - Spent time in nature/the outdoors when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Spent\_Time\_in\_Nature',  
 'Share - Question: mh8a - Talked to mental health professional when anxious/depressed - Answer: Yes - Gender: all - Age group: all': 'Talked\_to\_Mental\_Health\_Professional'  
}, inplace=True)  
  
# Performing an inner join  
merged\_inner = df\_2019.merge(df1\_2019, on=['Country', 'Code', 'Year'] , how='inner').merge(df2, on=['Country'], how='inner')

merged\_inner = merged\_inner.drop(columns=['Code', 'Year', 'Continent',  
 'Depressive disorders (share of population) - Sex: Both - Age: Age-standardized\_y'])

# Save the combined DataFrame to a new CSV file  
merged\_inner.to\_csv('mental\_health\_issues.csv', index=False)

## Unsupervised learning techniques

import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.preprocessing import StandardScaler  
from sklearn.decomposition import PCA  
from sklearn.cluster import KMeans  
from sklearn.metrics import silhouette\_score  
from sklearn.metrics import confusion\_matrix, accuracy\_score  
from sklearn.cluster import AgglomerativeClustering  
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

# Load the data  
data = pd.read\_csv('mental\_health\_issues.csv')  
  
# Basic statistics and missing values  
print(data.isnull().sum())

Country 0  
Schizophrenia\_Disorders 0  
Depressive\_Disorders 0  
Anxiety\_Disorders 0  
Bipolar\_Disorders 0  
Eating\_Disorders 0  
GDP\_per\_Capita 1  
Engaged\_in\_Religious\_Activities 1  
Improved\_Healthy\_Lifestyle 0  
Changed\_Work\_Situation 0  
Changed\_Personal\_Relationships 0  
Talked\_to\_Friends\_Family 0  
Took\_Medication 0  
Spent\_Time\_in\_Nature 0  
Talked\_to\_Mental\_Health\_Professional 0  
dtype: int64

# Fill missing values with the mean of the column  
data['GDP\_per\_Capita'].fillna(data['GDP\_per\_Capita'].mean(), inplace=True)  
data['Engaged\_in\_Religious\_Activities'].fillna(data['Engaged\_in\_Religious\_Activities'].mean(), inplace=True)

# PCA

# Drop specific columns  
data\_clean = data.drop(columns=['GDP\_per\_Capita'])  
# Initialize the StandardScaler  
scaler = StandardScaler()  
# Scale the data excluding the 'Country' column  
scaled\_data = scaler.fit\_transform(data\_clean.iloc[:, 1:])  
# Convert scaled data back to DataFrame  
scaled\_data = pd.DataFrame(scaled\_data, columns=data\_clean.columns[1:])

# Handling missing values by filling with the median  
for col in data.columns[1:]: # Assuming the first column is 'Country' and doesn't need imputation  
 if data[col].isnull().any():  
 data[col].fillna(data[col].median(), inplace=True)# Perform PCA  
pca = PCA()  
pca\_out = pca.fit\_transform(scaled\_data)

pd.DataFrame(pca.components\_.T  
 , index=scaled\_data.columns  
 , columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11', 'PC12', 'PC13'])

# Perform PCA  
pca = PCA()  
pca\_out = pca.fit\_transform(scaled\_data\_df)  
# Create plot  
fig, ax1 = plt.subplots(figsize=(11, 8))  
# Set limits  
ax1.set\_xlim(-3.5, 3.5)  
ax1.set\_ylim(-3.5, 3.5)  
# Plot country names for PC1 and PC2  
for i, country in enumerate(data\_clean['Country']):  
 ax1.annotate(country, (pca\_out[i, 0], pca\_out[i, 1]))  
# Plot Principal Component loading vectors using a twin axis.  
ax2 = ax1.twinx().twiny()  
ax2.set\_ylim(-1, 1)  
ax2.set\_xlim(-1, 1)  
# Plot principal components as arrows  
for i, (comp, var) in enumerate(zip(pca.components\_.T, scaled\_data\_df.columns)):  
 color = 'r' if i < 5 else 'b' # Blue for the first 5 components, gray for the rest  
 ax2.arrow(0, 0, comp[0], comp[1], head\_width=0.03, head\_length=0.01, ec=color)  
 ax2.text(comp[0]\*1.15, comp[1]\*1.15, var, color=color)  
  
ax1.set\_xlabel("PC1")  
ax1.set\_ylabel("PC2")  
plt.show()

fig, axes = plt.subplots(1, 2, figsize=(15, 6))  
ticks = np.arange(len(pca.explained\_variance\_ratio\_)) + 1  
# Plot for the proportion of variance explained by each component  
ax = axes[0]  
ax.plot(ticks,  
 pca.explained\_variance\_ratio\_,  
 marker='o')  
ax.set\_xlabel('Principal Component')  
ax.set\_ylabel('Proportion of Variance Explained')  
ax.set\_ylim([0, 1])  
ax.set\_xticks(ticks)  
# Annotating each point with the variance explained  
for i, v in enumerate(pca.explained\_variance\_ratio\_):  
 ax.text(ticks[i], v + 0.02, f"{v:.2f}", ha='center', va='bottom')  
# Plot for the cumulative proportion of variance explained  
ax = axes[1]  
cumulative\_variance = np.cumsum(pca.explained\_variance\_ratio\_)  
ax.plot(ticks,  
 cumulative\_variance,  
 marker='o')  
ax.set\_xlabel('Principal Component')  
ax.set\_ylabel('Cumulative Proportion of Variance Explained')  
ax.set\_ylim([0, 1])  
ax.set\_xticks(ticks)  
# Annotating each point with the cumulative variance explained  
for i, v in enumerate(cumulative\_variance):  
 ax.text(ticks[i], v + 0.02, f"{v:.2f}", ha='center', va='bottom')  
plt.tight\_layout()  
plt.show()

# KNN

# Calculate WCSS for different numbers of clusters  
wcss = []  
for i in range(1, 9):  
 kmeans = KMeans(n\_clusters=i, n\_init=10, random\_state=42)  
 kmeans.fit(scaled\_data)  
 wcss.append(kmeans.inertia\_)  
# Plot the WCSS against the number of clusters  
plt.figure(figsize=(10, 6))  
plt.plot(range(1, 9), wcss, marker='o', linestyle='--')  
plt.title('Elbow Method for Optimal Number of Clusters')  
plt.xlabel('Number of Clusters')  
plt.ylabel('Total Within-Cluster Sum of Squares (WCSS)')  
plt.show()

Choose 3 cluster since the change from adding 1 more clusters diminish. 3 would be the optimal number of clusters and paint the resulting groups on the principal components.

# K-means clustering vs PCA  
# Initialize PCA, choosing to keep two principal components  
pca = PCA(n\_components=7)  
pca\_data = pca.fit\_transform(scaled\_data)  
# Perform K-means clustering using the PCA results  
kmeans = KMeans(n\_clusters=3, n\_init=10, random\_state=42)  
clusters = kmeans.fit\_predict(pca\_data)  
# Add clustering information back to the original DataFrame  
data['Cluster'] = clusters  
# Check the distribution of countries in each cluster  
print(data['Cluster'].value\_counts())  
  
# Plot the first two principal components  
plt.figure(figsize=(8, 6))  
sns.scatterplot(x=pca\_data[:, 0], y=pca\_data[:, 1], hue=data['Cluster'], palette='viridis')  
plt.title('PCA and k-mean Clustering of Dataset')  
plt.xlabel('Principal Component 1')  
plt.ylabel('Principal Component 2')  
plt.legend(title='Cluster')  
plt.show()  
# Print the explained variance ratio  
print("Explained Variance Ratio:", pca.explained\_variance\_ratio\_)  
# Plotting the results for GDP\_per\_Capita  
plt.figure(figsize=(10, 6))  
sns.barplot(x=cluster\_summary.index, y=cluster\_summary['GDP\_per\_Capita'])  
plt.title('Average GDP per Capita by Cluster')  
plt.xlabel('Cluster')  
plt.ylabel('Average GDP per Capita')  
plt.show()

# Calculate the mental health issues for each cluster  
cluster\_means = data.groupby('Cluster')[[ 'Schizophrenia\_Disorders', 'Depressive\_Disorders','Anxiety\_Disorders', 'Bipolar\_Disorders', 'Eating\_Disorders']].mean()  
# Convert mean values to probabilities within each cluster  
cluster\_probs = cluster\_means.div(cluster\_means.sum(axis=1), axis=0)  
# Display the results  
print("Probabilities of Mental Health Issues for Each Cluster:")  
cluster\_probs

# Calculate the average GDP per capita and unemployment rate for each cluster  
cluster\_means = data.groupby('Cluster')[['GDP\_per\_Capita']].mean()  
cluster\_means

# Calculate the mean values for each behavioral variable within each cluster  
cluster\_means = data.groupby('Cluster')[['Engaged\_in\_Religious\_Activities', 'Improved\_Healthy\_Lifestyle', 'Changed\_Work\_Situation', 'Changed\_Personal\_Relationships', 'Talked\_to\_Friends\_Family', 'Took\_Medication', 'Spent\_Time\_in\_Nature', 'Talked\_to\_Mental\_Health\_Professional']].mean()  
# Convert mean values to probabilities (normalized to sum to 1 within each cluster)  
cluster\_probs = cluster\_means.div(cluster\_means.sum(axis=1), axis=0)  
  
# Transpose the DataFrame to make clusters as columns and variables as rows  
cluster\_probs\_transposed = cluster\_probs.T  
  
# Display the results  
print("Probabilities of Behavioral Variables for Each Cluster:")  
cluster\_probs\_transposed

# Define thresholds for income levels  
low\_threshold = 11000  
medium\_threshold = 45000  
# # Create a new column 'Income\_Level' based on the thresholds  
data.loc[data['GDP\_per\_Capita'] < low\_threshold, 'Income\_Level'] = 'Low'  
data.loc[(data['GDP\_per\_Capita'] >= low\_threshold) & (data['GDP\_per\_Capita'] < medium\_threshold), 'Income\_Level'] = 'Medium'  
data.loc[data['GDP\_per\_Capita'] >= medium\_threshold, 'Income\_Level'] = 'High'  
# Convert Income\_Level to numerical values  
income\_mapping = {'Low': 1, 'Medium': 2, 'High': 3}  
data['Income\_Level\_Num'] = data['Income\_Level'].map(income\_mapping)

# Create the confusion matrix  
conf\_matrix = confusion\_matrix(data['Income\_Level\_Num'], kmeans\_pca.labels\_ + 1)  
accuracy = accuracy\_score(data['Income\_Level\_Num'], kmeans\_pca.labels\_ + 1)  
# Plot the confusion matrix  
plt.figure(figsize=(10, 7))  
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',  
 xticklabels=['Cluster 0', 'Cluster 1', 'Cluster 2'],  
 yticklabels=['Low', 'Medium', 'High'])  
plt.xlabel('PCA-Based Clusters')  
plt.ylabel('Income Levels')  
plt.title('Confusion Matrix: PCA-Based Clusters vs. Income Levels')  
plt.show()  
# Display the confusion matrix  
print("Confusion Matrix:")  
print(conf\_matrix)  
print(f"Accuracy: {accuracy:.2f}")

# Hierarchical Clustering

# Define thresholds for income levels  
low\_threshold = 10000  
medium\_threshold = 45000  
# # Create a new column 'Income\_Level' based on the thresholds  
data.loc[data['GDP\_per\_Capita'] < low\_threshold, 'Income\_Level'] = 'Low'  
data.loc[(data['GDP\_per\_Capita'] >= low\_threshold) & (data['GDP\_per\_Capita'] < medium\_threshold), 'Income\_Level'] = 'Medium'  
data.loc[data['GDP\_per\_Capita'] >= medium\_threshold, 'Income\_Level'] = 'High'

# Separate the columns needed for clustering analysis

data\_features = data.drop(columns=['Country', 'Income\_Level', 'GDP\_per\_Capita'])  
# Scale the feature data  
scaler = StandardScaler()  
scaled\_data = scaler.fit\_transform(data\_features)  
# Convert scaled data back to DataFrame  
scaled\_data\_df = pd.DataFrame(scaled\_data, columns=data\_features.columns)  
# Initialize PCA

pca = PCA(n\_components=7)  
pca\_data = pca.fit\_transform(scaled\_data)  
# Perform Hierarchical Clustering  
hc = AgglomerativeClustering(distance\_threshold=0, n\_clusters=None, linkage='average')  
hc.fit(pca\_data)  
# Extract linkage matrix using the linkage method  
linkage\_matrix = linkage(pca\_data, method='average')  
# Cutting the dendrogram at a specific distance to form three clusters  
data['HC\_Cluster'] = fcluster(linkage\_matrix, t=4.8, criterion='distance')

# Create the plot to visualize the dendrogram  
fig, ax = plt.subplots(figsize=(8, 8))  
dendrogram(linkage\_matrix, labels=data\_clean['Country'].values, color\_threshold=4.8, ax=ax)  
ax.axhline(y=4.8, color='black') # Add a line to indicate the cut-off  
plt.title("Hierarchical Clustering Dendrogram")  
plt.xlabel("Country")  
plt.ylabel("Distance")  
plt.show()

# Convert Income\_Level to numerical values  
income\_mapping = {'Low': 1, 'Medium': 2, 'High': 3}  
data['Income\_Level\_Num'] = data['Income\_Level'].map(income\_mapping)

# Create the confusion matrix and calculate accuracy  
conf\_matrix = confusion\_matrix(data['Income\_Level\_Num'], data['HC\_Cluster'])  
accuracy = accuracy\_score(data['Income\_Level\_Num'], data['HC\_Cluster'])  
# Plot the confusion matrix  
plt.figure(figsize=(10, 7))  
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',  
 xticklabels=['Cluster 1', 'Cluster 2', 'Cluster 3'],  
 yticklabels=['Low', 'Medium', 'High'])  
plt.xlabel('Hierarchical Clusters')  
plt.ylabel('Income Levels')  
plt.title('Confusion Matrix: Hierarchical Clusters vs. Income Levels')  
plt.show()  
# Display the confusion matrix and accuracy  
print("Confusion Matrix:")  
print(conf\_matrix)  
print(f"Accuracy: {accuracy:.2f}")