MP5

December 2, 2025

1 Machine Exercise 5

Jhon Mark L. Acopiado

University of the Philippines Diliman

Masters of Engineering in Artificial Intelligence

AI221:Classical Machine Learning

1.1 Part 1: Country Data from HELP International

1.1.1 1.I. Load Dataset

[1]:

```
# Source Code From : https://www.kaggle.com/datasets/rohan0301/
 ↪unsupervised-learning-on-country-data/data
# Install dependencies as needed:
# pip install kagglehub[pandas-datasets]
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "Country-data.csv"

# Load the latest version
df = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "rohan0301/unsupervised-learning-on-country-data",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documentation for more information:
  # https://github.com/Kaggle/kagglehub/blob/main/README.
 ↪md#kaggledatasetadapterpandas
)

print("First 5 records:", df.head())
```

```
C:\Users\jhon\AppData\Local\Temp\ipykernel_20780\2839170658.py:11:
DeprecationWarning: Use dataset_load() instead of load_dataset(). load_dataset()
```

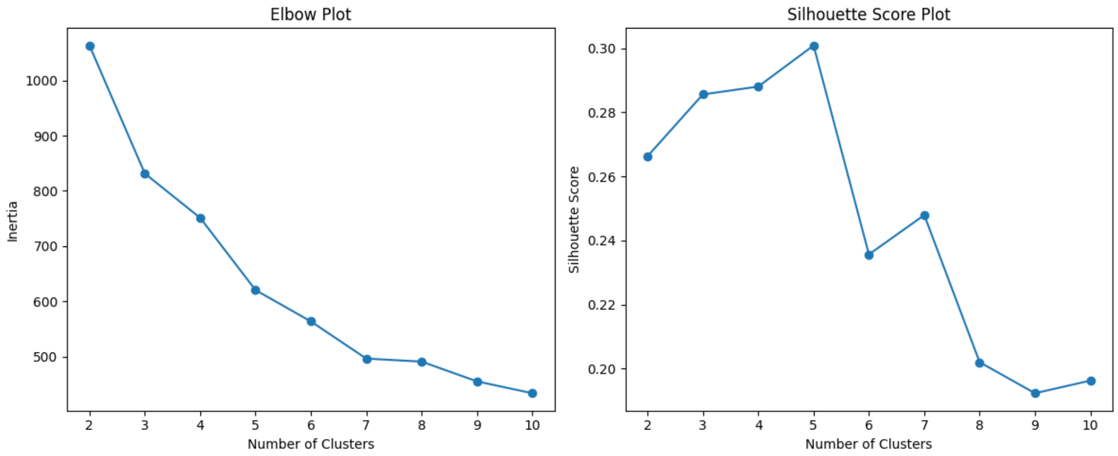
```
will be removed in a future version.
  df = kagglehub.load_dataset(

First 5 records:                country  child_mort  exports  health  imports
income  \
0          Afghanistan        90.2     10.0    7.58     44.9     1610
1              Albania        16.6     28.0    6.55     48.6     9930
2              Algeria        27.3     38.4    4.17     31.4    12900
3               Angola       119.0     62.3    2.85     42.9     5900
4  Antigua and Barbuda        10.3     45.5    6.03     58.9    19100

   inflation  life_expec  total_fer   gdpp
0       9.44        56.2       5.82    553
1       4.49        76.3       1.65   4090
2      16.10        76.5       2.89   4460
3      22.40        60.1       6.16   3530
4       1.44        76.8       2.13  12200
```

[2]:
```python
# Prompt: Store Features and Target Variable
X = df.drop(columns=['country'])
Y = df['country']
```

1.1.2 1.II. K-Means

Item 1.a. Normalize the features data using Standard Scaler. Then, perform K-means clustering on all features. Display the elbow plot (Inertia vs. no. of clusters) and the silhouette score plot. What number of clusters is recommended?

[3]:
```python
# Prompt: Normalize the features of X using StandardScaler
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

import matplotlib.pyplot as plt

# Define the range for the number of clusters
cluster_range = range(2, 11)
inertia = []
silhouette_scores = []

# Perform KMeans clustering for each number of clusters
for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))


def plot_elbow_and_silhouette(cluster_range, inertia, silhouette_scores):
    # Plot the elbow plot (Inertia vs. number of clusters)
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(cluster_range, inertia, marker='o')
    plt.title('Elbow Plot')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Inertia')

    # Plot the silhouette score plot
    plt.subplot(1, 2, 2)
    plt.plot(cluster_range, silhouette_scores, marker='o')
    plt.title('Silhouette Score Plot')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Silhouette Score')

    plt.tight_layout()
    plt.show()

# Call the function
plot_elbow_and_silhouette(cluster_range, inertia, silhouette_scores)
```



Item 1.a. What number of clusters is recommended?

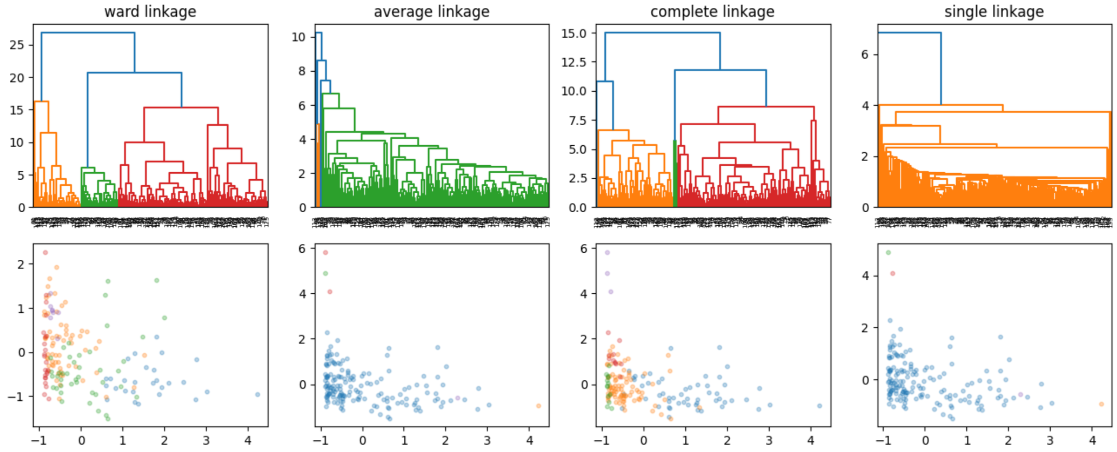
Based on the elbow plot and Silhouette Score, the recommended number of clusters is either 5 or 7. For both 5 and 7 clusters, the Silhouette Score increases, indicating that the data points are well-clustered and distinct from other clusters. The elbow plot shows the relationship between the number of clusters and the inertia. At 7 clusters, the rate of decrease in inertia slows down significantly compared to 5 clusters.

If I were to choose only one, I would select 7 clusters.

1.1.3 1.III. Hierarchical clustering

Item 1.b. Perform hierarchical clustering on the normalized data set and compare the results of various linkage methods. Which one would you recommend? Why is this recommended clustering informative?

[4]:
```
# Soure Code is From AI221 Github Repository:␣
 ↪AI221\Clustering_Anomaly_Detect\kmeans_agglo_blobs.ipynb
from scipy.cluster.hierarchy import linkage, dendrogram, cut_tree
from time import time

fig3 = plt.figure(figsize=(16,6))
ctr = 1
t0 = time()
for method in ('ward', 'average', 'complete', 'single'):
    Z = linkage(X_scaled, method=method)
    ax = fig3.add_subplot(240 + ctr)
    ax.set_title("%s linkage" % method)
    dendrogram(Z)
    cutree = cut_tree(Z, n_clusters=7).flatten()
    ax = fig3.add_subplot(240 + ctr + 4)
    for j in range(0,5):
        ax.scatter(X_scaled[cutree == j,0], X_scaled[cutree == j,1], s=10,␣
 ↪alpha=0.3)
    ctr += 1

plt.show()
print(f"Elapsed Time: {time()-t0} sec")
```



```
Elapsed Time: 3.20088267326355 sec
```

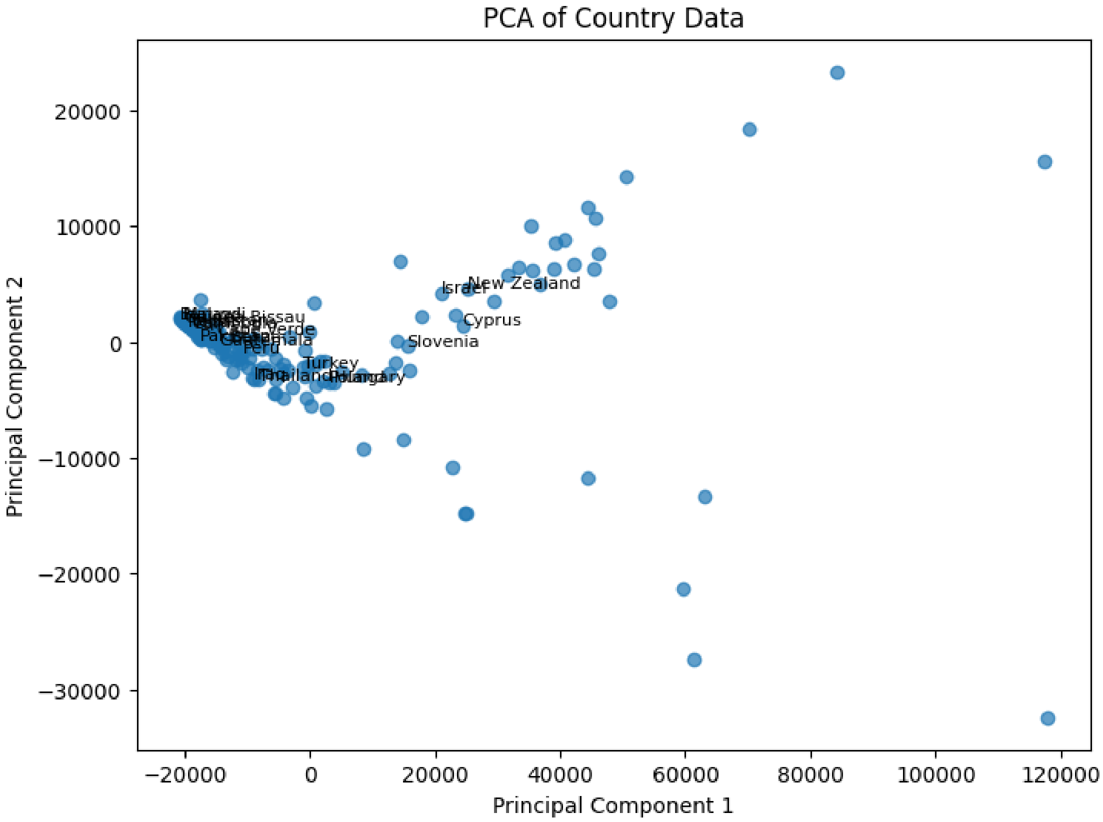
Item 1.b. Which one would you recommend? Why is this recommended clustering informative?

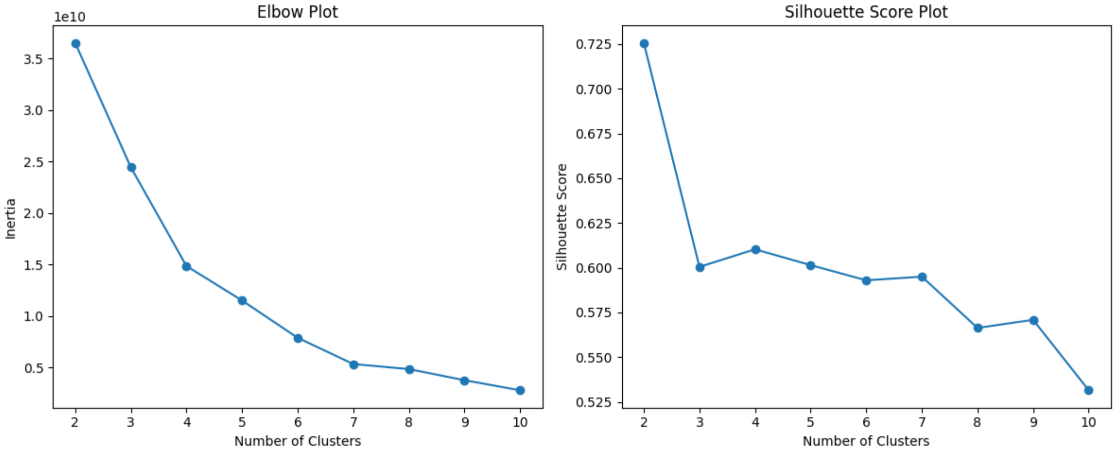
I recommend using the Ward linkage method. For this analysis, I set cut_tree to 7 clusters to align with my response to the previous item. The visualization demonstrates that the Ward linkage method is more informative compared to other methods. Ward linkage minimizes the variance within clusters, resulting in compact and well-separated clusters. This makes it particularly effective when the data has a Gaussian-like distribution or when compact clusters are desired. The method ensures that the clusters formed are homogeneous and interpretable, making it a suitable choice for this dataset.

1.1.4 1.IV. PCA (2 features) + K-means

Item 1.c. This time, perform PCA to reduce the features data into 2D. Then, perform K-means clustering on the reduced data set just as in item (a). Display the elbow plot and silhouette score plot as well.

```
[5]: # Prompt: Perform PCA on X to reduce its dimensions to 2 components
from sklearn.decomposition import PCA
import random
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Prompt: Display a scatter plot of the PCA-reduced data with some points labeled
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.7)
# Prompt: Randomly select some countries to label
selected_countries = random.sample(list(df['country']), 20)


for i, country in enumerate(df['country']):
    if country in selected_countries:
        plt.text(X_pca[i, 0], X_pca[i, 1], s=country, fontsize=8)
plt.title('PCA of Country Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
# Repeat KMeans clustering on PCA-reduced data
cluster_range = range(2, 11)
inertia = []
silhouette_scores = []





# Perform KMeans clustering for each number of clusters
for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(X_pca)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_pca, kmeans.labels_))



plot_elbow_and_silhouette(cluster_range, inertia, silhouette_scores)
```





Item 1.c. What number of clusters is recommended? You can label some countries in the 2D mapping.

I would recommend 7 clusters as it is the number where the elbow plot shows a significant decrease in inertia. Additionally, the Silhouette score for 7 clusters is higher than that of its neighboring cluster numbers, indicating well-defined and distinct clusters.

Item 1.d. Based on the recommended no. of clusters in item (b), make your own descriptions of each cluster. What range of values of features are unique to each cluster?

To support my answer, let me get insights from the table, and plots below. I will base my descriptions on the mean values of the feature variables for each cluster. Below are the descriptions for each cluster:

- **Cluster 1**: This cluster is characterized by high health values (10.51) and low inflation (1.33). These countries likely have strong healthcare systems and stable economies.
- **Cluster 2**: Defined by the lowest income (3017.69), life expectancy (59.04), and GDP per capita (1455.76). These countries may face significant economic and health challenges.
- **Cluster 3**: Notable for the highest child mortality (130.00), inflation (104.00), and fertility rates (5.84). These countries may have developing economies with high population growth.
- **Cluster 4**: Distinguished by the highest exports (176.00), imports (156.67), life expectancy (81.43), and GDP per capita (57566.67). These countries are likely developed nations with strong trade and high standards of living.
- **Cluster 5**: Characterized by the highest income (67171.43) and lowest health values (3.28). These countries may have high wealth but face challenges in healthcare access or outcomes.
- **Cluster 0**: Represents countries with average values across most features, indicating balanced development without extremes in any specific area.
- **Cluster 6**: Defined by unique outliers or anomalies in the dataset, possibly representing countries with rare or exceptional conditions not captured by other clusters.

The following blocks of code support the analysis by providing detailed insights into the clustering results:

- Mean Values of Features for Each Cluster: The code calculates the mean values of each feature for every cluster, allowing us to identify patterns and differences across clusters.

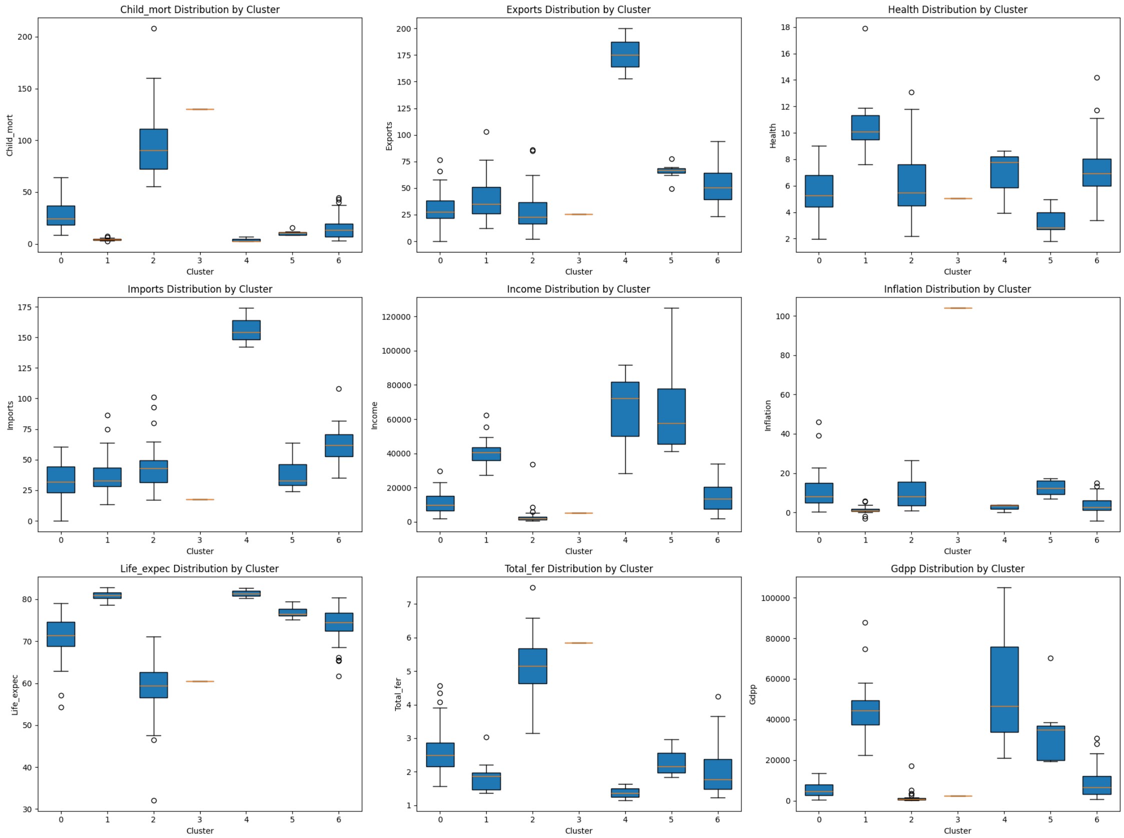
- Box Plots for Feature Distributions: The code generates box plots to visualize the distribution of each feature within the clusters. This helps in understanding the variability and spread of the data for each feature across different clusters.
- Clusters with Highest and Lowest Mean Values: The code identifies which cluster has the highest and lowest mean value for each feature, highlighting the distinguishing characteristics of the clusters.
- Visualization of Cluster Grouping: Finally, the code includes a plot that visually represents how the clusters group as the number of clusters increases. This provides a clear picture of the clustering structure and how the data is segmented.

[6]:

```
import seaborn as sns

cluster_labels = KMeans(n_clusters=7, random_state=42).fit_predict(X_scaled)
df_with_clusters = df.copy()
df_with_clusters['Cluster'] = cluster_labels  # Add cluster labels to the␣
 ↪DataFrame
cluster_summary = df_with_clusters.drop(columns=['country']).groupby('Cluster').
 ↪agg(['mean'])
print(display(cluster_summary))

# Prompt: Plot all feature distributions by cluster in the same figure
plt.figure(figsize=(20, 15))
num_features = len(df_with_clusters.columns[1:-1])  # Exclude 'country' and␣
 ↪'Cluster' columns
for i, feature in enumerate(df_with_clusters.columns[1:-1], 1):
    plt.subplot((num_features + 2) // 3, 3, i)  # Arrange subplots in a grid
    for cluster in sorted(df_with_clusters['Cluster'].unique()):
        cluster_data = df_with_clusters[df_with_clusters['Cluster'] == cluster]
        plt.boxplot(cluster_data[feature], positions=[cluster], widths=0.6,␣
 ↪patch_artist=True)
    plt.title(f'{feature.capitalize()} Distribution by Cluster')
    plt.xlabel('Cluster')
    plt.ylabel(feature.capitalize())

plt.tight_layout()
plt.show()

# Prompt: Determine which cluster has the highest and lowest values for each␣
 ↪column
highest_lowest_clusters = {}

for column in df_with_clusters.columns[1:-1]:  # Exclude 'country' and␣
 ↪'Cluster' columns
    cluster_means = df_with_clusters.groupby('Cluster')[column].mean()
    highest_cluster = cluster_means.idxmax()
    lowest_cluster = cluster_means.idxmin()
    highest_lowest_clusters[column] = {
        'highest_cluster': highest_cluster,
        'highest_value': cluster_means[highest_cluster],
        'lowest_cluster': lowest_cluster,
        'lowest_value': cluster_means[lowest_cluster]
    }

# Display the results
for feature, clusters in highest_lowest_clusters.items():
    print(f"Feature: {feature}")
    print(f"  Highest Cluster: {clusters['highest_cluster']} (Value:␣
 ↪{clusters['highest_value']:.2f})")
    print(f"  Lowest Cluster: {clusters['lowest_cluster']} (Value:␣
 ↪{clusters['lowest_value']:.2f})")


# Prompt: Visualize KMeans clustering results for cluster counts from 2 to 10
inertia_values = []
silhouette_avgs = []
fig2 = plt.figure(figsize=(20,20))
for i in range(2,11):
    kmeans = KMeans(n_clusters=i, n_init=10).fit(X_scaled)
    cluster_labels = kmeans.labels_
    centroids = kmeans.cluster_centers_
    inertia_values.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    silhouette_avgs.append(silhouette_avg)
    ax = fig2.add_subplot(330 + i - 1)
    for j in range(0,i):
        ax.scatter(X_scaled[cluster_labels == j,0], X_scaled[cluster_labels ==␣
 ↪j,1], s=10, alpha=0.3)
    ax.scatter(centroids[:,0],centroids[:,1], s=50, color='b', edgecolor='k')
    ax.set_title(f"Inertia = {kmeans.inertia_}\nSilhouette Score =␣
 ↪{silhouette_avg}\nNumber of clusters = {i}")
```

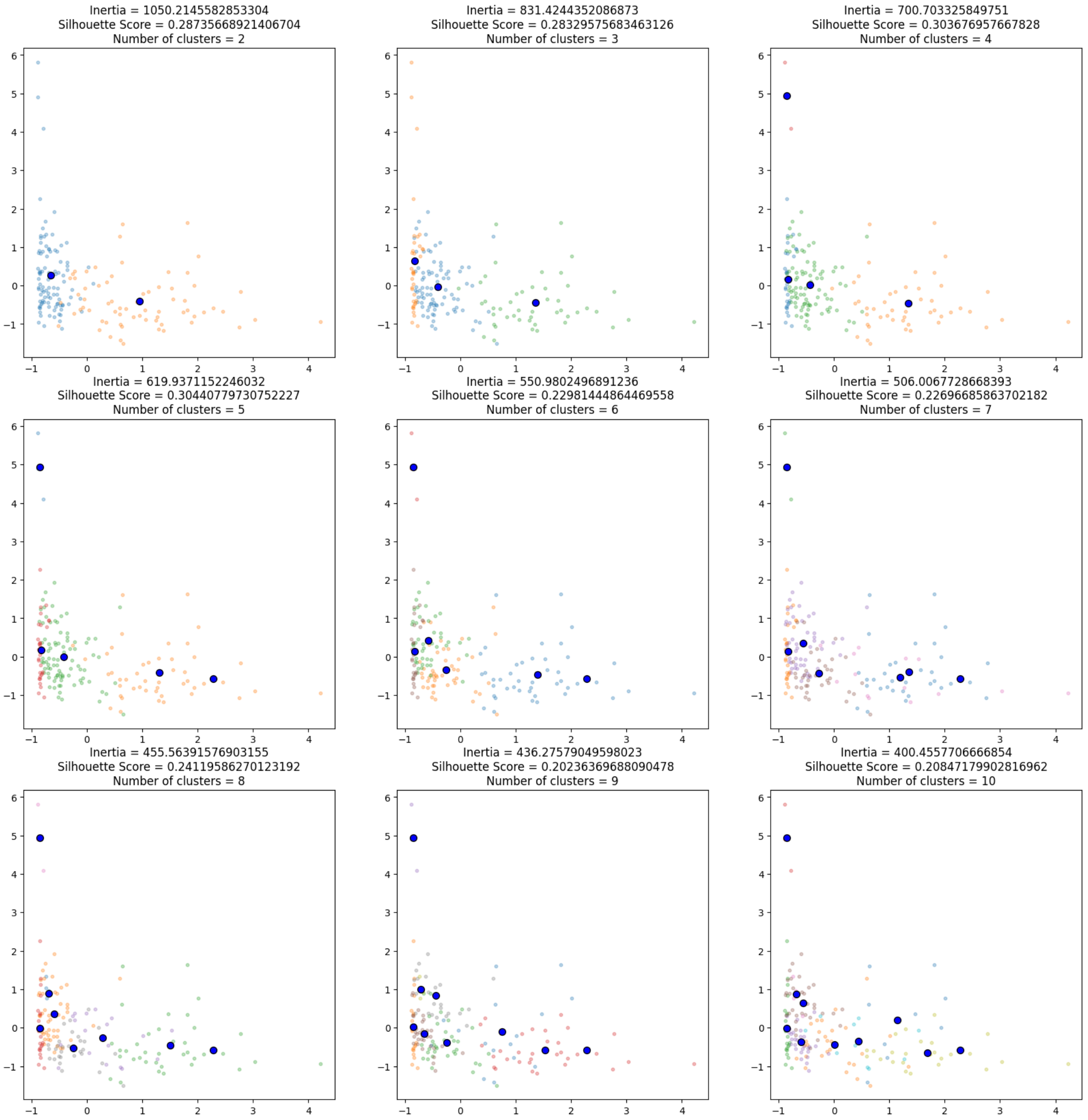
```
         child_mort     exports     health     imports        income  \
               mean        mean       mean        mean          mean
Cluster
0         29.430435   30.621500   5.569565   33.577520  10946.956522
1          4.295652   40.730435  10.513478   38.247826  40265.217391
2         96.009524   27.988333   6.334286   43.811905   3017.690476
3        130.000000   25.300000   5.070000   17.400000   5150.000000
4          4.133333  176.000000   6.793333  156.666667  64033.333333
5         10.700000   65.557143   3.281429   38.700000  67171.428571
6         15.306667   51.824444   7.238889   62.400000  14206.888889

          inflation life_expec total_fer          gdpp
```

```
              mean       mean      mean          mean
Cluster
0        10.601435  71.139130  2.590652   5526.000000
1         1.334913  80.891304  1.810870  45417.391304
2         9.820357  59.042857  5.129286   1455.761905
3       104.000000  60.500000  5.840000   2330.000000
4         2.468000  81.433333  1.380000  57566.666667
5        12.517143  76.928571  2.287143  34057.142857
6         3.771511  73.928889  2.001556   8703.244444
None
```



```
Feature: child_mort
  Highest Cluster: 3 (Value: 130.00)
  Lowest Cluster: 4 (Value: 4.13)
Feature: exports
  Highest Cluster: 4 (Value: 176.00)
  Lowest Cluster: 3 (Value: 25.30)
Feature: health
  Highest Cluster: 1 (Value: 10.51)
  Lowest Cluster: 5 (Value: 3.28)
Feature: imports
```

```
  Highest Cluster: 4 (Value: 156.67)
  Lowest Cluster: 3 (Value: 17.40)
Feature: income
  Highest Cluster: 5 (Value: 67171.43)
  Lowest Cluster: 2 (Value: 3017.69)
Feature: inflation
  Highest Cluster: 3 (Value: 104.00)
  Lowest Cluster: 1 (Value: 1.33)
Feature: life_expec
  Highest Cluster: 4 (Value: 81.43)
  Lowest Cluster: 2 (Value: 59.04)
Feature: total_fer
  Highest Cluster: 3 (Value: 5.84)
  Lowest Cluster: 4 (Value: 1.38)
Feature: gdpp
  Highest Cluster: 4 (Value: 57566.67)
  Lowest Cluster: 2 (Value: 1455.76)
```



1.1.5 1.V. 2D PCA + Anomaly Detetion

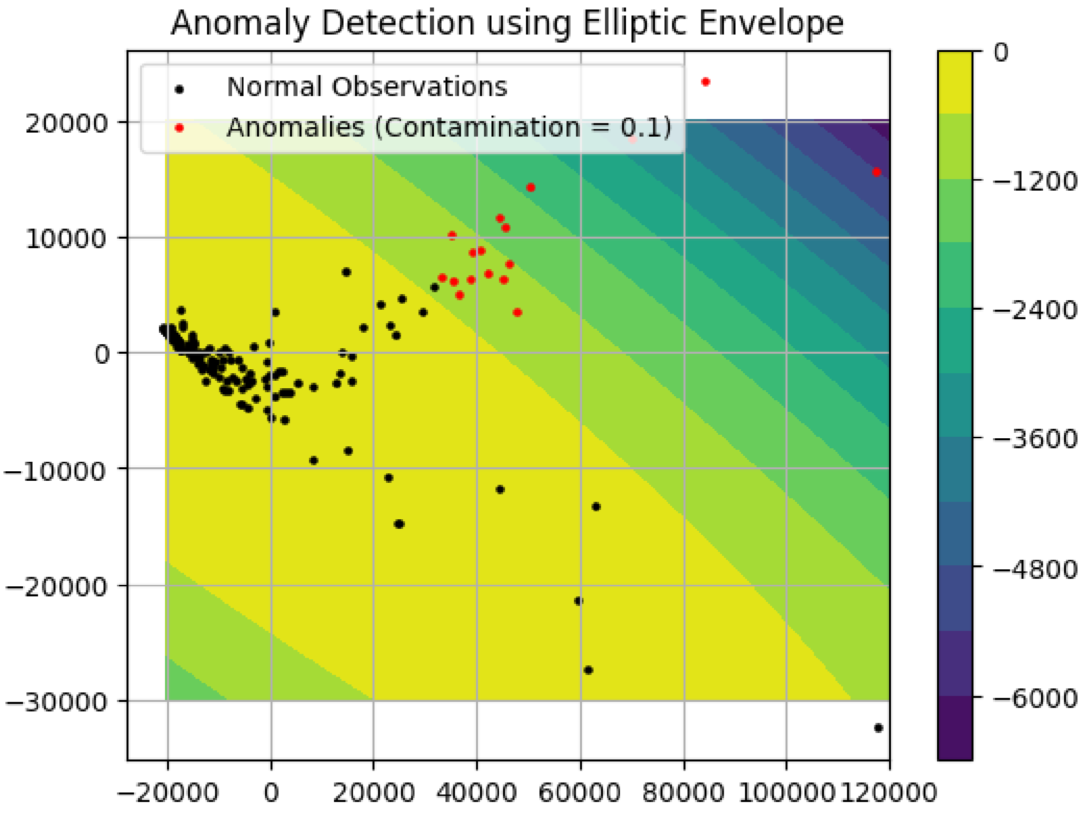
Item 1.e. Based on the 2D PCA mapping, perform anomaly detection using any method.

[8]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor
from sklearn.neighbors import KernelDensity
from sklearn.ensemble import IsolationForest
from sklearn.datasets import make_blobs
from sklearn.svm import OneClassSVM
```

1.V.a. Elliptic Envelope

[9]:

```
# Source Code is From AI221 Github Repository:␣
 ↪AI221\Clustering_Anomaly_Detect\anomaly_detect_methods.ipynb
Xp, Yp = np.meshgrid(np.linspace(-20000,120000),np.linspace(-30000,20000))
XY = np.vstack([Xp.ravel(), Yp.ravel()]).T

envelope = EllipticEnvelope(random_state=0, contamination=0.1).fit(X_pca)
Zp = envelope.score_samples(XY)
Zp = Zp.reshape(Xp.shape)

# Get the anomalous data points
y_pred = envelope.predict(X_pca)
normals = X_pca[y_pred == 1,:]
anomals = X_pca[y_pred == -1,:]

cntr = plt.contourf(Xp, Yp, Zp, levels=10, cmap='viridis')
plt.scatter(normals[:,0], normals[:,1], s=5, color='k', label='Normal␣
 ↪Observations')
plt.scatter(anomals[:,0], anomals[:,1], s=5, color='r', label='Anomalies␣
 ↪(Contamination = 0.1)')
plt.title('Anomaly Detection using Elliptic Envelope')
plt.colorbar(cntr)
plt.legend()
plt.grid()
plt.show()

envelope_anomalies = df.iloc[np.where(y_pred == -1)[0]]
```

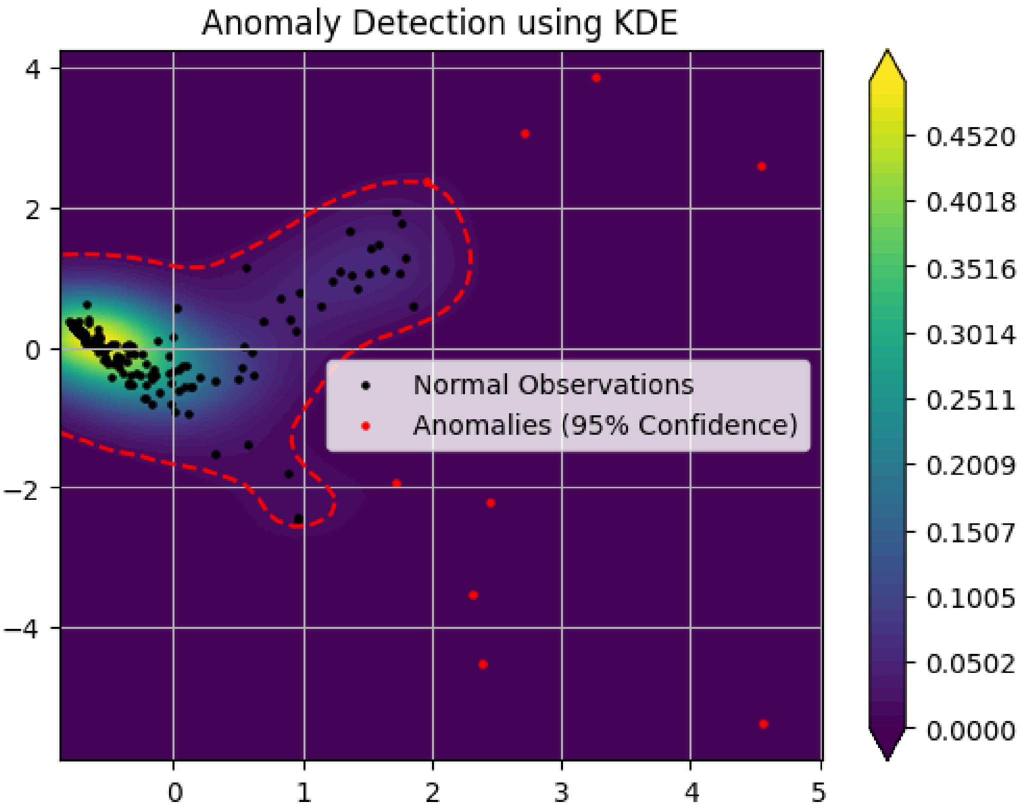


1.V.b Kernel Density

[10]:

```
# Source Code is From AI221 Github Repository:␣
 ↪AI221\Clustering_Anomaly_Detect\anomaly_detect_methods.ipynb
# Generate the KDE surface as Z
X_pca_scaled  = scaler.fit_transform(X_pca)
kde = KernelDensity(kernel='gaussian',bandwidth=0.4).fit(X_pca_scaled)


Xp, Yp = np.meshgrid(
    np.linspace(X_pca_scaled[:, 0].min() * 1.1, X_pca_scaled[:, 0].max() * 1.1,␣
 ↪50),
    np.linspace(X_pca_scaled[:, 1].min() * 1.1, X_pca_scaled[:, 1].max() * 1.1,␣
 ↪50)
)
XY = np.vstack([Xp.ravel(), Yp.ravel()]).T

Zp = np.exp(kde.score_samples(XY))
Zp = Zp.reshape(Xp.shape)

# Establish a confidence level of 95% (or 5% cutoff)
# for the UCL using the quantile of kde_scores.
scores = kde.score_samples(X_pca_scaled)
threshold = np.quantile(scores,0.05)
print(f"Threshold (KDE) = {np.exp(threshold)}")

# Get the anomalous data points
normals = X_pca_scaled[scores > threshold,:]
anomals = X_pca_scaled[scores <= threshold,:]

cntr = plt.contourf(Xp, Yp, Zp, levels=np.linspace(Zp.min(), Zp.max(), 50),␣
 ↪cmap='viridis', extend='both')
plt.contour(Xp, Yp, Zp, levels=[np.exp(threshold)], colors='red',␣
 ↪linestyles='dashed', linewidths=1.5)
plt.scatter(normals[:,0], normals[:,1], s=5, color='k', label='Normal␣
 ↪Observations')
plt.scatter(anomals[:,0], anomals[:,1], s=5, color='r', label='Anomalies (95%␣
 ↪Confidence)')
plt.title('Anomaly Detection using KDE')
plt.colorbar(cntr)
plt.legend()
plt.grid()
plt.show()

kernel_density_anomalies = df.iloc[np.where(scores <= threshold)[0]]
```

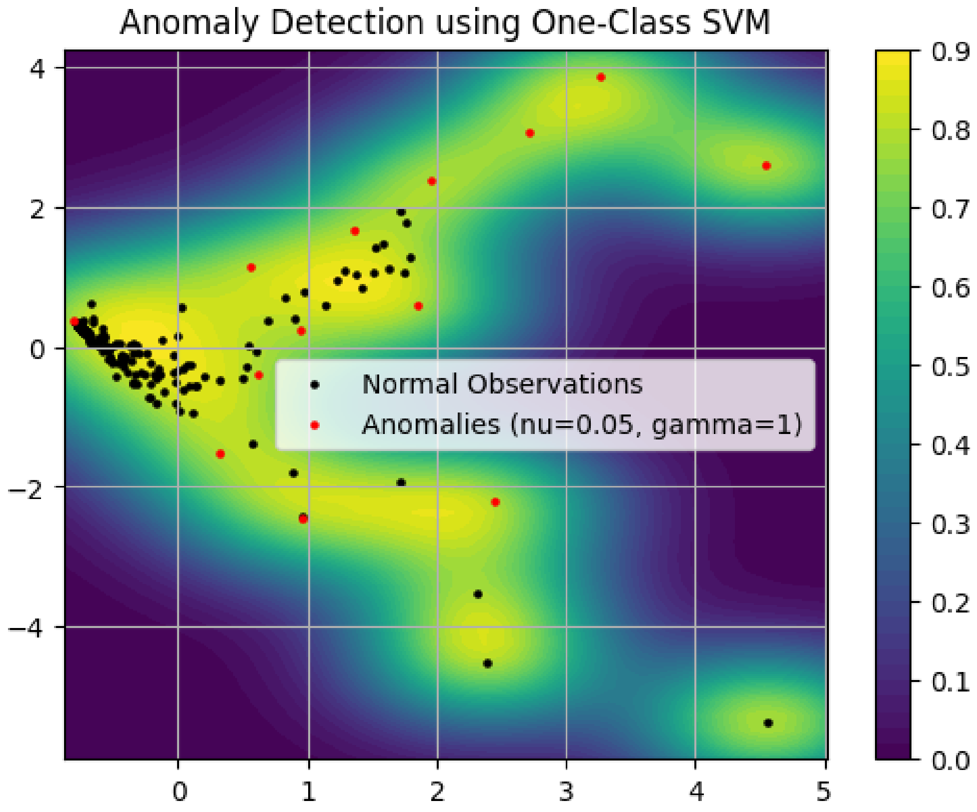
```
Threshold (KDE) = 0.012569474629724105
```



1.V.c. One Class SVM

[11]:

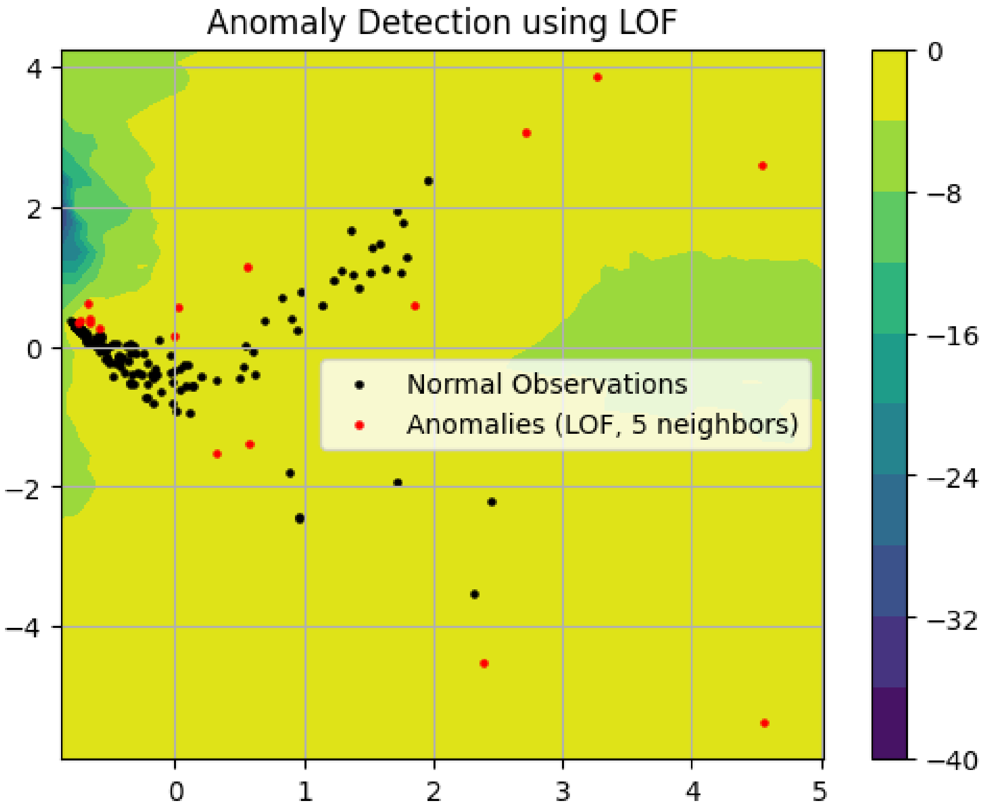
```python
# Source Code is From AI221 Github Repository:␣
  ↪AI221\Clustering_Anomaly_Detect\anomaly_detect_methods.ipynb
ocsvm = OneClassSVM(nu=0.05, gamma=1).fit(X_pca_scaled)
Zp = ocsvm.score_samples(XY)
Zp = Zp.reshape(Xp.shape)

# Get the anomalous data points
y_pred = ocsvm.predict(X_pca_scaled)
normals = X_pca_scaled[y_pred == 1,:]
anomals = X_pca_scaled[y_pred == -1,:]

cntr = plt.contourf(Xp, Yp, Zp, levels=50, cmap='viridis')
plt.scatter(normals[:,0], normals[:,1], s=5, color='k', label='Normal␣
  ↪Observations')
plt.scatter(anomals[:,0], anomals[:,1], s=5, color='r', label='Anomalies (nu=0.
  ↪05, gamma=1)')
plt.title('Anomaly Detection using One-Class SVM')
plt.colorbar(cntr)
plt.legend()
plt.grid()
plt.show()

ocsvm_anomalies = df.iloc[np.where(y_pred == -1)[0]]
```



1.V.d. Local Outlier Factor

```
[12]: # Source Code is From AI221 Github Repository:␣
       ↪AI221\Clustering_Anomaly_Detect\anomaly_detect_methods.ipynb
      lof = LocalOutlierFactor(n_neighbors=5,novelty=True).fit(X_pca_scaled)
      Zp = lof.score_samples(XY)
      Zp = Zp.reshape(Xp.shape)

      # Get the anomalous data points
      y_pred = lof.predict(X_pca_scaled)
      normals = X_pca_scaled[y_pred == 1,:]
      anomals = X_pca_scaled[y_pred == -1,:]

      cntr = plt.contourf(Xp, Yp, Zp, levels=10, cmap='viridis')
      plt.scatter(normals[:,0], normals[:,1], s=5, color='k', label='Normal␣
       ↪Observations')
```

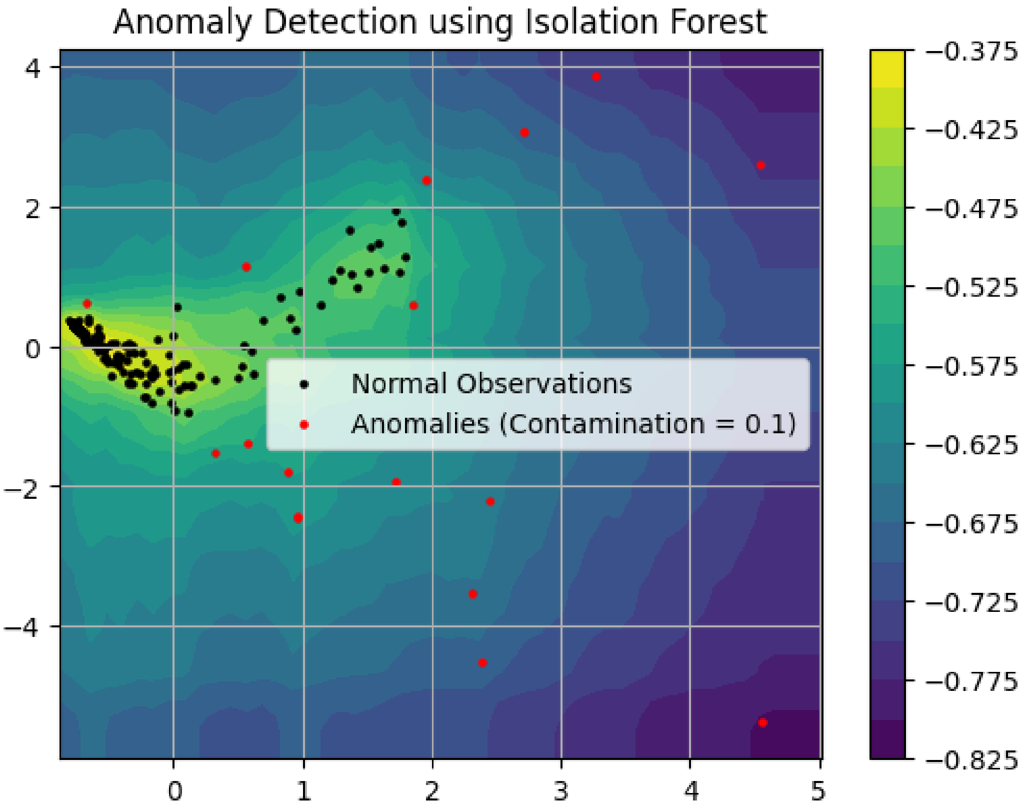
```
plt.scatter(anomals[:,0], anomals[:,1], s=5, color='r', label='Anomalies (LOF,␣
  ↪5 neighbors)')
plt.title('Anomaly Detection using LOF')
plt.colorbar(cntr)
plt.legend()
plt.grid()
plt.show()

lof_anomalies = df.iloc[np.where(y_pred == -1)[0]]
```



1.V.e. Isolation Forest

[13]:
```
# Source Code is From AI221 Github Repository:␣
  ↪AI221\Clustering_Anomaly_Detect\anomaly_detect_methods.ipynb
isoforest = IsolationForest(contamination=0.1).fit(X_pca_scaled)
Zp = isoforest.score_samples(XY)
Zp = Zp.reshape(Xp.shape)

# Get the anomalous data points
y_pred = isoforest.predict(X_pca_scaled)
normals = X_pca_scaled[y_pred == 1,:]
```

```
anomals = X_pca_scaled[y_pred == -1,:]

cntr = plt.contourf(Xp, Yp, Zp, levels=20, cmap='viridis')
plt.scatter(normals[:,0], normals[:,1], s=5, color='k', label='Normal␣
 ↪Observations')
plt.scatter(anomals[:,0], anomals[:,1], s=5, color='r', label='Anomalies␣
 ↪(Contamination = 0.1)')
plt.title('Anomaly Detection using Isolation Forest')
plt.colorbar(cntr)
plt.legend()
plt.grid()
plt.show()

isoforest_anomalies = df.iloc[np.where(y_pred == -1)[0]]
```



Item 1.e. Which countries are deemed to be outliers and why? Which features make them outliers?

Based on the collective results from various anomaly detection models, the analysis reveals the following: Switzerland, Luxembourg, and Norway are identified as anomalies by all five models. Denmark is flagged by four models, while Japan, Brunei, Qatar, Singapore, Libya, the United States, and the Bahamas are identified by three models.

A statistical comparison between the combined anomalies from the five models and the overall dataset highlights significant differences. The anomalies exhibit higher values in exports, imports, income, and GDP per capita (GDPP) compared to the overall dataset. This indicates that the countries identified as anomalies are developed nations.

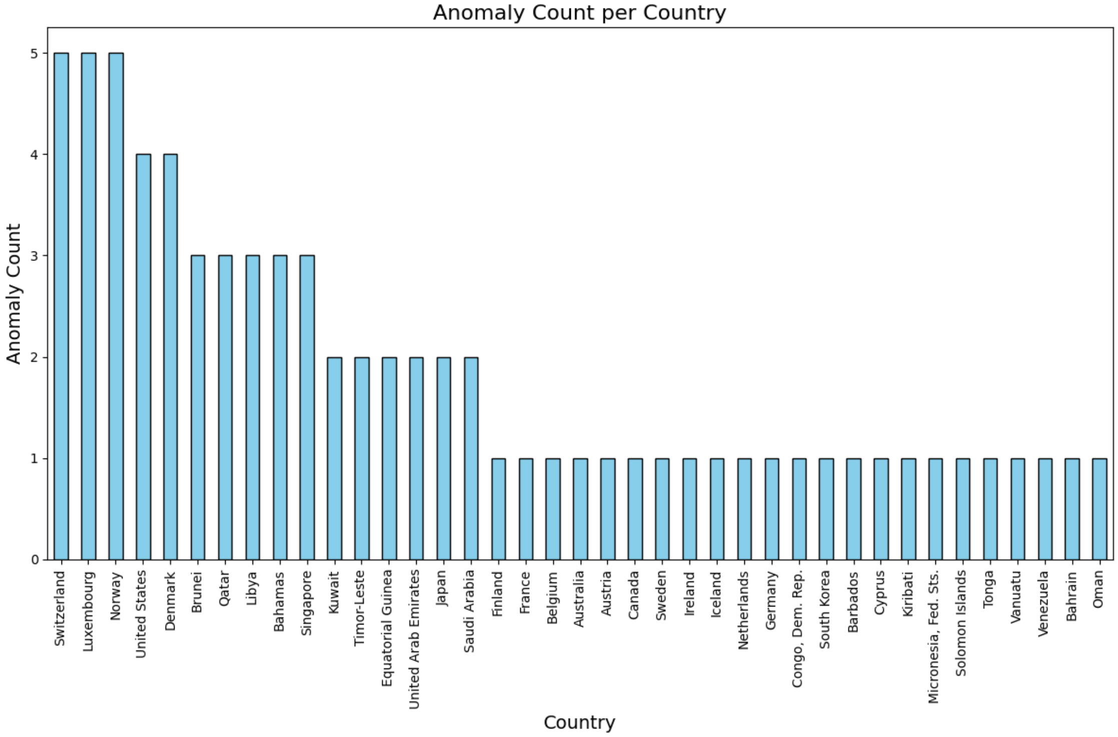
In contrast, the majority of countries in the dataset are either underdeveloped or still developing. Alternatively, it could suggest that the anomaly countries are exceptionally successful, standing out significantly from the rest.

The following code provides supporting evidence for the analysis described above.

[14]:

```python
# Prompt: Create a bar plot showing the count of anomalies detected per country across all methods, x-axis: country, y-axis: count of anomalies
import pandas as pd

import matplotlib.pyplot as plt

# Combine all anomaly dataframes into one
all_anomalies = pd.concat([
    envelope_anomalies['country'],
    kernel_density_anomalies['country'],
    ocsvm_anomalies['country'],
    lof_anomalies['country'],
    isoforest_anomalies['country']
])

# Count the occurrences of each country
anomaly_counts = all_anomalies.value_counts()

# Plot the bar plot
plt.figure(figsize=(12, 8))
anomaly_counts.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Anomaly Count per Country', fontsize=16)
plt.xlabel('Country', fontsize=14)
plt.ylabel('Anomaly Count', fontsize=14)
plt.xticks(rotation=90, fontsize=10)
plt.tight_layout()
plt.show()

all_anomalies_df = pd.concat([
    envelope_anomalies,
    kernel_density_anomalies,
    ocsvm_anomalies,
    lof_anomalies,
    isoforest_anomalies
])

# Prompt: Remove duplicate entries based on country from all_anomalies_df
```

```
all_anomalies_df = all_anomalies_df.drop_duplicates(subset=['country']).
 ↪sort_values(by='country')

print(display(all_anomalies_df))
# Prompt: Show the Mean and Standard Deviation of each feature in␣
 ↪all_anomalies_df
anomaly_stats = all_anomalies_df.describe().loc[['mean', 'std']]
print(display(anomaly_stats))
# Prompt: Show the Mean and Standard Deviation of each feature in the original␣
 ↪dataframe df
original_stats = df.describe().loc[['mean', 'std']]
print(display(original_stats))
```



```
                country  child_mort  exports  health  imports  income  \
7             Australia         4.8     19.8    8.73     20.9   41400
8               Austria         4.3     51.3   11.00     47.8   43200
10              Bahamas        13.8     35.0    7.89     43.7   22900
11              Bahrain         8.6     69.5    4.97     50.9   41100
13             Barbados        14.2     39.5    7.97     48.7   15300
15              Belgium         4.5     76.4   10.70     74.7   41100
23               Brunei        10.5     67.4    2.84     28.0   80600
29               Canada         5.6     29.1   11.30     31.0   40700
37     Congo, Dem. Rep.       116.0     41.1    7.91     49.6     609
```

```
42                  Cyprus         3.6     50.2    5.97     57.5   33900
44                 Denmark         4.1     50.5   11.40     43.6   44000
49       Equatorial Guinea       111.0     85.8    4.48     58.9   33700
53                 Finland         3.0     38.7    8.95     37.4   39800
54                  France         4.2     26.8   11.90     28.1   36900
58                 Germany         4.2     42.3   11.60     37.1   40400
68                 Iceland         2.6     53.4    9.40     43.3   38800
73                 Ireland         4.2    103.0    9.19     86.5   45700
77                   Japan         3.2     15.0    9.49     13.6   35800
81                Kiribati        62.7     13.3   11.30     79.9    1730
82                  Kuwait        10.8     66.7    2.63     30.4   75200
89                   Libya        16.6     65.6    3.88     42.1   29600
91              Luxembourg         2.8    175.0    7.77    142.0   91700
101  Micronesia, Fed. Sts.        40.0     23.5   14.20     81.0    3340
110            Netherlands         4.5     72.0   11.90     63.6   45500
114                 Norway         3.2     39.7    9.48     28.5   62300
115                   Oman        11.7     65.7    2.77     41.2   45300
123                  Qatar         9.0     62.3    1.81     23.8  125000
128           Saudi Arabia        15.7     49.6    4.29     33.0   45400
133              Singapore         2.8    200.0    3.96    174.0   72100
136        Solomon Islands        28.1     49.3    8.55     81.2    1780
138            South Korea         4.1     49.4    6.93     46.2   30400
144                 Sweden         3.0     46.2    9.63     40.7   42900
145            Switzerland         4.5     64.0   11.50     53.3   55500
149            Timor-Leste        62.6      2.2    9.12     27.8    1850
151                  Tonga        17.4     12.4    5.07     60.3    4980
157   United Arab Emirates         8.6     77.7    3.66     63.6   57600
159          United States         7.3     12.4   17.90     15.8   49400
162                Vanuatu        29.2     46.6    5.25     52.7    2950
163              Venezuela        17.1     28.5    4.91     17.6   16500

     inflation  life_expec  total_fer    gdpp
7        1.160        82.0       1.93   51900
8        0.873        80.5       1.44   46900
10      -0.393        73.8       1.86   28000
11       7.440        76.0       2.16   20700
13       0.321        76.7       1.78   16000
15       1.880        80.0       1.86   44400
23      16.700        77.1       1.84   35300
29       2.870        81.3       1.63   47400
37      20.800        57.5       6.54     334
42       2.010        79.9       1.42   30800
44       3.220        79.5       1.87   58000
49      24.900        60.9       5.21   17100
53       0.351        80.0       1.87   46200
54       1.050        81.4       2.03   40600
58       0.758        80.1       1.39   41800
68       5.470        82.0       2.20   41900
```

```
73      -3.220        80.4       2.05   48700
77      -1.900        82.8       1.39   44500
81       1.520        60.7       3.84    1490
82      11.200        78.2       2.21   38500
89      14.200        76.1       2.41   12100
91       3.620        81.3       1.63  105000
101      3.800        65.4       3.46    2860
110      0.848        80.7       1.79   50300
114      5.950        81.0       1.95   87800
115     15.600        76.1       2.90   19300
123      6.980        79.5       2.07   70300
128     17.200        75.1       2.96   19300
133     -0.046        82.7       1.15   46600
136      6.810        61.7       4.24    1290
138      3.160        80.1       1.23   22100
144      0.991        81.5       1.98   52100
145      0.317        82.2       1.52   74600
149     26.500        71.1       6.23    3600
151      3.680        69.9       3.91    3550
157     12.500        76.5       1.87   35000
159      1.220        78.7       1.93   48400
162      2.620        63.0       3.50    2970
163     45.900        75.4       2.47   13500
None
      child_mort    exports    health    imports        income  inflation  \
mean   17.541026  54.279487  8.005128  51.282051  39408.692308   6.893846
std    26.812622  38.638883  3.608391  31.583556  26523.865636   9.778181

      life_expec  total_fer          gdpp
mean   76.123077   2.454359  35158.820513
std     7.033995   1.286196  24695.282429
None
      child_mort    exports    health    imports        income  inflation  \
mean   38.270060  41.108976  6.815689  46.890215  17144.688623   7.781832
std    40.328931  27.412010  2.746837  24.209589  19278.067698  10.570704

      life_expec  total_fer          gdpp
mean   70.555689   2.947964  12964.155689
std     8.893172   1.513848  18328.704809
None
```

1.2 2. Early-Stage Diabetes Risk Prediction

1.2.1 2.I. Load Dataset

[19]:

```
# code source: https://archive.ics.uci.edu/dataset/529/
  ↪early+stage+diabetes+risk+prediction+dataset
from ucimlrepo import fetch_ucirepo

# fetch dataset
early_stage_diabetes_risk_prediction = fetch_ucirepo(id=529)

# data (as pandas dataframes)
X = early_stage_diabetes_risk_prediction.data.features
y = early_stage_diabetes_risk_prediction.data.targets

# metadata
print(early_stage_diabetes_risk_prediction.metadata)

# variable information
print(early_stage_diabetes_risk_prediction.variables)
```

```
{'uci_id': 529, 'name': 'Early Stage Diabetes Risk Prediction',
'repository_url': 'https://archive.ics.uci.edu/dataset/529/early+stage+diabetes+
risk+prediction+dataset', 'data_url':
'https://archive.ics.uci.edu/static/public/529/data.csv', 'abstract': 'This
dataset contains the sign and symptpom data of newly diabetic or would be
diabetic patient. ', 'area': 'Computer Science', 'tasks': ['Classification'],
'characteristics': ['Multivariate'], 'num_instances': 520, 'num_features': 16,
'feature_types': ['Categorical', 'Integer'], 'demographics': ['Age', 'Gender'],
'target_col': ['class'], 'index_col': None, 'has_missing_values': 'no',
'missing_values_symbol': None, 'year_of_dataset_creation': 2020, 'last_updated':
'Mon Mar 04 2024', 'dataset_doi': '10.24432/C5VG8H', 'creators': [],
'intro_paper': {'ID': 397, 'type': 'NATIVE', 'title': 'Likelihood Prediction of
Diabetes at Early Stage Using Data Mining Techniques', 'authors': 'M. M. F.
Islam, Rahatara Ferdousi, Sadikur Rahman, Humayra Yasmin Bushra', 'venue':
'Computer Vision and Machine Intelligence in Medical Image Analysis', 'year':
2019, 'journal': None, 'DOI': '10.1007/978-981-13-8798-2_12', 'URL': 'https://ww
w.semanticscholar.org/paper/9329dec57c5f13f195220ffa7077fd0029983f07', 'sha':
None, 'corpus': None, 'arxiv': None, 'mag': None, 'acl': None, 'pmid': None,
'pmcid': None}, 'additional_info': {'summary': 'This has been col-\r\nlected
using direct questionnaires from the patients of Sylhet Diabetes\r\nHospital in
Sylhet, Bangladesh and approved by a doctor.', 'purpose': None, 'funded_by':
None, 'instances_represent': None, 'recommended_data_splits': None,
'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'Age
1.20-65\t\t\r\nSex 1. Male, 2.Female\t\t\r\nPolyuria 1.Yes,
2.No.\t\t\r\nPolydipsia 1.Yes, 2.No.\t\t\r\nsudden weight loss 1.Yes,
2.No.\t\t\r\nweakness 1.Yes, 2.No.\t\t\r\nPolyphagia 1.Yes, 2.No.\t\t\r\nGenital
thrush 1.Yes, 2.No.\t\t\r\nvisual blurring 1.Yes, 2.No.\t\t\r\nItching 1.Yes,
2.No.\t\t\r\nIrritability 1.Yes, 2.No.\t\t\r\ndelayed healing 1.Yes,
2.No.\t\t\r\npartial paresis 1.Yes, 2.No.\t\t\r\nmuscle sti\x0bness 1.Yes,
2.No.\t\t\r\nAlopecia 1.Yes, 2.No.\t\t\r\nObesity 1.Yes, 2.No.\t\t\r\nClass
1.Positive, 2.Negative.\t\t\r\n', 'citation': None}}
                  name     role         type demographic description units  \
0                  age  Feature      Integer         Age        None  None
1               gender  Feature  Categorical      Gender        None  None
2             polyuria  Feature       Binary        None        None  None
3           polydipsia  Feature       Binary        None        None  None
4   sudden_weight_loss  Feature       Binary        None        None  None
5             weakness  Feature       Binary        None        None  None
6           polyphagia  Feature       Binary        None        None  None
7       genital_thrush  Feature       Binary        None        None  None
8      visual_blurring  Feature       Binary        None        None  None
9              itching  Feature       Binary        None        None  None
10        irritability  Feature       Binary        None        None  None
11     delayed_healing  Feature       Binary        None        None  None
12     partial_paresis  Feature       Binary        None        None  None
13    muscle_stiffness  Feature       Binary        None        None  None
14            alopecia  Feature       Binary        None        None  None
15             obesity  Feature       Binary        None        None  None
16               class   Target       Binary        None        None  None

   missing_values
0              no
1              no
2              no
3              no
4              no
5              no
6              no
7              no
8              no
9              no
10             no
11             no
12             no
13             no
14             no
15             no
16             no
```

1.2.2 2.II. Encoding, and Dataset Split

Item 2.a. Make the necessary encoding for categorical inputs. Split the data into 80% Training and 20% Testing with stratification.

[33]:

```
# Encode categorical variables
from sklearn.calibration import LabelEncoder
```

```
X = pd.get_dummies(X, drop_first=True, dtype=int)
# Encode y

le = LabelEncoder()
y = le.fit_transform(y)


# Split the dataset into training and testing sets (stratify = y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

1.2.3 2.III. Best Model Using

Item 2.b. Using Optuna, find the best model between the MLP Classifier, Random Forest Classifier, XGBoost Classifier, Logistic Regression, Naïve Bayes Classifier, SVM Classifier (SVC), and kNN Classifier. Set Optuna to maximize the 10-fold cross-validation score (cross_val_score). You are free to design the search space for hyper-parameters in these models.

[37]:

```
import optuna
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score

# Set the logging level to WARNING to suppress info logs
optuna.logging.set_verbosity(optuna.logging.WARNING)

def objective(trial):
    classifier_name = trial.suggest_categorical("classifier",
                                                 ["MLP", "RandomForest",␣
 ↪"XGBoost", "LogisticRegression", "NaiveBayes", "SVM", "kNN"])

    if classifier_name == "MLP":
        hidden_layer_sizes_choices = ["(50,)", "(100,)", "(50, 50)"]
        params = {
            "hidden_layer_sizes": eval(trial.
 ↪suggest_categorical("hidden_layer_sizes", hidden_layer_sizes_choices)),
            "activation": trial.suggest_categorical("activation", ["relu",␣
 ↪"tanh"]),
            "solver": trial.suggest_categorical("solver", ["adam", "sgd"]),
            "alpha": trial.suggest_float("alpha", 1e-5, 1e-1, log=True),
        }
        model = MLPClassifier(**params, max_iter=1000, random_state=42,␣
↪early_stopping=True)

    elif classifier_name == "RandomForest":
        params = {
            "n_estimators": trial.suggest_int("n_estimators", 50, 300, step=50),
            "max_depth": trial.suggest_int("max_depth", 3, 20),
            "min_samples_split": trial.suggest_int("min_samples_split", 2, 10),
        }
        model = RandomForestClassifier(**params, random_state=42)

    elif classifier_name == "XGBoost":
        params = {
            "max_depth": trial.suggest_int("max_depth", 3, 15),
            "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3,␣
↪log=True),
            "n_estimators": trial.suggest_int("n_estimators", 50, 300, step=50),
        }
        model = XGBClassifier(**params, eval_metric="logloss", random_state=42)

    elif classifier_name == "LogisticRegression":
        params = {
            "C": trial.suggest_float("C", 1e-3, 1e3, log=True),
            "solver": trial.suggest_categorical("k-solver", ["liblinear",␣
↪"lbfgs"]),
        }
        model = LogisticRegression(C=params["C"], solver=params["solver"],␣
↪max_iter=1000, random_state=42)

    elif classifier_name == "NaiveBayes":
        model = GaussianNB()

    elif classifier_name == "SVM":
        params = {
            "C": trial.suggest_float("C", 1e-3, 1e3, log=True),
            "kernel": trial.suggest_categorical("kernel", ["linear", "rbf",␣
↪"poly"]),
        }
        model = SVC(**params, random_state=42)

    elif classifier_name == "kNN":
        params = {
            "n_neighbors": trial.suggest_int("n_neighbors", 3, 15),
            "weights": trial.suggest_categorical("weights", ["uniform",␣
↪"distance"]),
        }
        model = KNeighborsClassifier(**params)

    # Perform 10-fold cross-validation
    score = cross_val_score(model, X_train, y_train, cv=10, scoring="accuracy").
 ↪mean()
    return score

# Create and run the Optuna study
study = optuna.create_study(direction="maximize", sampler=optuna.samplers.
 ↪TPESampler(seed=42))
study.optimize(objective, n_trials=500, show_progress_bar=True)

# Get the best model and hyperparameters
best_params = study.best_params
best_classifier = best_params.pop("classifier")

if best_classifier == "MLP":
    hidden_layer_sizes = eval(best_params["hidden_layer_sizes"])
    best_model = MLPClassifier(**best_params,  max_iter=1000, random_state=42,␣
 ↪early_stopping=True)
elif best_classifier == "RandomForest":
    best_model = RandomForestClassifier(**best_params, random_state=42)
elif best_classifier == "XGBoost":
    best_model = XGBClassifier(**best_params, eval_metric="logloss",␣
 ↪random_state=42)
elif best_classifier == "LogisticRegression":
    best_model = LogisticRegression(C=best_params["C"],␣
 ↪solver=best_params["k-solver"], random_state=42)
elif best_classifier == "NaiveBayes":
    best_model = GaussianNB()
elif best_classifier == "SVM":
    best_model = SVC(**best_params, random_state=42)
elif best_classifier == "kNN":
    best_model = KNeighborsClassifier(**best_params)

# Train the best model on the training data
best_model.fit(X_train, y_train)

# Evaluate on the test data
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average="weighted")

print(f"Best Classifier: {best_classifier}")
print(f"Best Hyperparameters: {best_params}")
print(f"Test Accuracy: {accuracy}")
print(f"Test F1-Score: {f1}")
```

```
  0%|          | 0/500 [00:00<?, ?it/s]

Best Classifier: RandomForest
Best Hyperparameters: {'n_estimators': 50, 'max_depth': 17, 'min_samples_split':
2}
Test Accuracy: 0.9807692307692307
Test F1-Score: 0.9808511271925906
```

Item 2.b. What is the Accuracy and F1-score on the Test Data of the best model?

The best classifier is RandomForest (n_estimators=50, max_depth=17, min_samples_split=2), havning Test Accuracy of 98% and Test F1-Score of 98%.

1.2.4 2.IV. Random Forest

This section focuses on two key objectives: optimizing the Random Forest model through hyperparameter tuning and comparing its performance with the paper's reported best Random Forest F1-score of 98%.

Item 2.c. In the paper, the best model was found to be Random Forest, having a weighted average F1 score of 0.98. Using your own hyper-parameter search, can you find a better Random Forest model with higher F1 score?

[59]:
```
def objective_rf(trial):

    params = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 1000, step=3),
        "max_depth": trial.suggest_int("max_depth", 3, 200),
        "min_samples_split": trial.suggest_int("min_samples_split", 2, 50),
    }
    model = RandomForestClassifier(**params, random_state=42)

    # Perform 10-fold cross-validation
    score = cross_val_score(model, X_train, y_train, cv=10, scoring="accuracy").
 ↪mean()
    return score

# Create and run the Optuna study
study_rf = optuna.create_study(direction="maximize", sampler=optuna.samplers.
 ↪TPESampler(seed=42))
study_rf.optimize(objective_rf, n_trials=100, show_progress_bar=True)

# Get the best model and hyperparameters
best_params_rf = study_rf.best_params



best_model_rf = RandomForestClassifier(**best_params_rf, random_state=42)
```

```
# Train the best model on the training data
best_model_rf.fit(X_train, y_train)

# Evaluate on the test data
y_pred_rf = best_model_rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf, average="weighted")

print(f"Best Hyperparameters: {best_params_rf}")
print(f"Test Accuracy: {accuracy_rf}")
print(f"Test F1-Score: {f1_rf}")
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

```
Best Hyperparameters: {'n_estimators': 985, 'max_depth': 117,
'min_samples_split': 2}
Test Accuracy: 0.9903846153846154
Test F1-Score: 0.9904061137656939
```

Item 2.c. Using your own hyper-parameter search, can you find a better Random Forest model with higher F1 score?

Improvemnt to 99%, the optimized Random Forest model (n_estimators=985, max_depth=117, min_samples_split=2) produced above show a slight increase in the F1-score of 98% compared to the result reported in the paper.

Item 2.d. Based on the best Random Forest model in item (c), perform any feature importance method to explain the model. What insights can we get from the model?

The results indicate that the top three features significantly influencing the model's performance are: polyuria_Yes, polydipsia_Yes, and gender_Male. These features play a critical role in the model's predictions and provide valuable insights into the factors driving its performance.

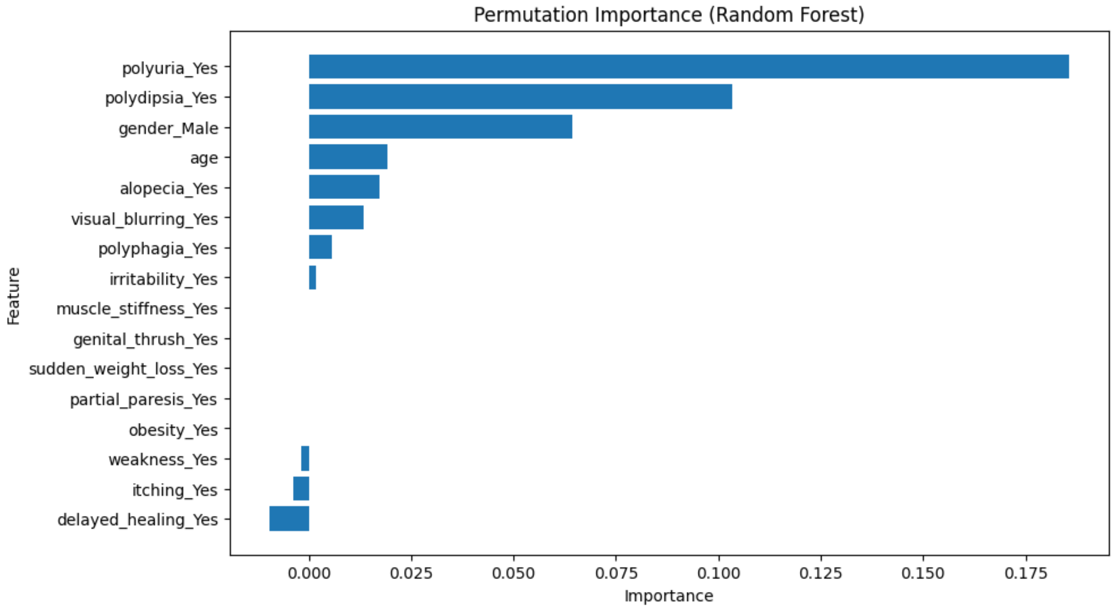
Insights:

- Polyuria (Excessive Urination): This is identified as a key feature, highlighting its significance as an early symptom of diabetes.
- Polydipsia (Excessive Thirst): Similarly, this feature underscores its importance as another early indicator of diabetes.
- Gender (Male): While being male appears as a top feature, this likely reflects the dataset's distribution, where a higher proportion of diabetes cases are observed in males. This does not necessarily imply a causal relationship but rather a pattern in the data.

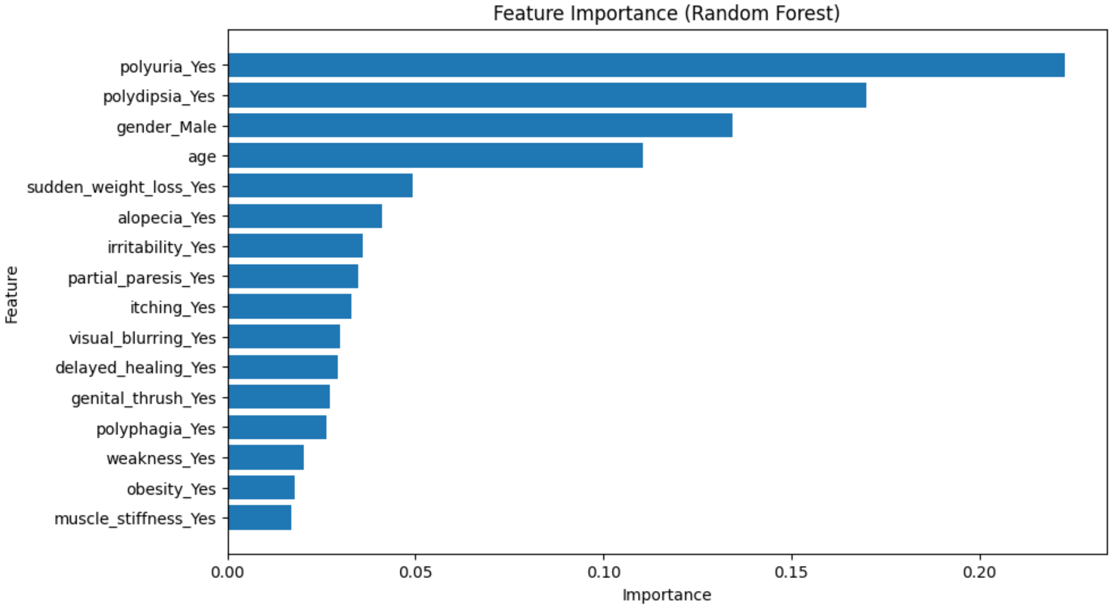
The following blocks of code support the answer provided:

[55]:
```
from sklearn.inspection import permutation_importance

# Calculate permutation importance
```

```
perm_importance = permutation_importance(best_model_rf, X_test, y_test,␣
 ↪scoring="f1_weighted")

# Create a DataFrame for better visualization
perm_importance_df = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": perm_importance.importances_mean
}).sort_values(by="Importance", ascending=False)

# Plot the permutation importance
plt.figure(figsize=(10, 6))
plt.barh(perm_importance_df["Feature"], perm_importance_df["Importance"])
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Permutation Importance (Random Forest)")
plt.gca().invert_yaxis()
plt.show()
```



[56]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Get feature importance from the trained Random Forest model
importances = best_model_rf.feature_importances_

# Create a DataFrame for better visualization
feature_importance_df = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df["Feature"], feature_importance_df["Importance"])
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Feature Importance (Random Forest)")
plt.gca().invert_yaxis()
plt.show()
```



[57]:

```
# This snippet of code is from MEX1 (household electricy consumption)
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, accuracy_score

alphas = np.logspace(-3, 3, 10)
best_alpha = None
best_val_score = -np.inf
best_model = None

X_val = X_test
y_val = y_test
for alpha in alphas:
    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('ridge', Ridge(alpha=alpha))
    ])
    pipe.fit(X_train, y_train)
    val_pred = pipe.predict(X_val)
    val_score = r2_score(y_val, val_pred)
    if val_score > best_val_score:
        best_val_score = val_score
        best_alpha = alpha
        best_model = pipe

# Final evaluation
train_score =  r2_score(y_train, best_model.predict(X_train))
test_score = r2_score(y_test, best_model.predict(X_test))

ridge = best_model.named_steps['ridge']

# Get the feature names for X1 to X8
feature_names = np.array(X.columns[:8])

# Get absolute value of coefficients for the best Ridge model
coef_abs = np.abs(ridge.coef_[:8])

# Get indices of top 5 features
top5_idx = np.argsort(coef_abs)[-5:][::-1]
top5_features = feature_names[top5_idx]

print("Top 5 features among X1 to X8:", list(top5_features))
```

```
Top 5 features among X1 to X8: ['polyuria_Yes', 'polydipsia_Yes', 'gender_Male',
'genital_thrush_Yes', 'sudden_weight_loss_Yes']
```

1.2.5 2.V. Random forest on larger n_estimators

This section is an additional task where I explore the impact of using a larger value for n_estimators.

[ ]:

```
def objective_rf_estimators(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 5000, 20000,␣
 ↪step=500),
        "max_depth": trial.suggest_int("max_depth", 3, 200),
        "min_samples_split": trial.suggest_int("min_samples_split", 2, 50),
    }
    model = RandomForestClassifier(**params, random_state=42)

    # Perform 10-fold cross-validation
    score = cross_val_score(model, X_train, y_train, cv=10, scoring="accuracy").
 ↪mean()
    return score
# Create and run the Optuna study
study_rf_estimators = optuna.create_study(direction="maximize", sampler=optuna.
 ↪samplers.TPESampler(seed=42))
study_rf_estimators.optimize(objective_rf_estimators, n_trials=10,␣
 ↪show_progress_bar=True)
# Get the best model and hyperparameters
best_params_rf_estimators = study_rf_estimators.best_params


best_model_rf_estimators = RandomForestClassifier(**best_params_rf_estimators,␣
 ↪random_state=42)
# Train the best model on the training data
best_model_rf_estimators.fit(X_train, y_train)

# Evaluate on the test data
y_pred_rf_estimators = best_model_rf_estimators.predict(X_test)
accuracy_rf_estimators = accuracy_score(y_test, y_pred_rf_estimators)
f1_rf_estimators = f1_score(y_test, y_pred_rf_estimators, average="weighted")

print(f"Best Hyperparameters: {best_params_rf_estimators}")
print(f"Test Accuracy: {accuracy_rf_estimators}")
print(f"Test F1-Score: {f1_rf_estimators}")
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```