**Preprocessing**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import BorderlineSMOTE

# Load dataset

df = pd.read\_csv("Synthetic\_PCOS\_10k.csv") # Ensure the file is in the same directory

# Drop irrelevant columns

df.drop(columns=["Sl. No", "Patient File No.", "Unnamed: 44"], inplace=True, errors='ignore')

# Handle missing values and type issues

df["II beta-HCG(mIU/mL)"] = pd.to\_numeric(df["II beta-HCG(mIU/mL)"], errors='coerce')

df["AMH(ng/mL)"] = pd.to\_numeric(df["AMH(ng/mL)"], errors='coerce')

df.dropna(thresh=2, inplace=True)

df.fillna(0, inplace=True)

# Round values

for col in ["BMI", "FSH/LH", "Waist:Hip Ratio"]:

if col in df.columns:

df[col] = df[col].round(2)

# Save cleaned dataset (before SMOTE)

df.to\_csv("PCOS\_original\_cleaned\_1.csv", index=False)

# Split features and target

X = df.drop(columns=["PCOS (Y/N)"])

y = df["PCOS (Y/N)"]

# Check class distribution before SMOTE

print("Class distribution before SMOTE:")

print(y.value\_counts())

# Apply Borderline SMOTE

smote = BorderlineSMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(X, y)

# Check class distribution after SMOTE

print("\nClass distribution after SMOTE:")

print(y\_resampled.value\_counts())

# Scale features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_resampled)

# Create final DataFrame

df\_processed = pd.DataFrame(X\_scaled, columns=X.columns)

df\_processed["PCOS (Y/N)"] = y\_resampled.reset\_index(drop=True)

# Save final preprocessed file

df\_processed.to\_csv("PCOS\_after\_preprocessing\_1.csv", index=False)

print("Files saved: 'PCOS\_original\_cleaned\_1.csv' and 'PCOS\_after\_preprocessing\_1.csv'")

**Selecting TOMIM Features**

import pandas as pd

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

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Load the preprocessed dataset

df = pd.read\_csv("PCOS\_after\_preprocessing\_1.csv")

# Step 2: Separate features and target

X = df.drop(columns=["PCOS (Y/N)"])

y = df["PCOS (Y/N)"]

# Step 3: Calculate Mutual Information Scores

mi\_scores = mutual\_info\_classif(X, y, random\_state=42)

# Step 4: Create a DataFrame of features and their MI scores

mi\_df = pd.DataFrame({

'Feature': X.columns,

'MI Score': mi\_scores

})

# Step 5: Sort features by MI score

mi\_df = mi\_df.sort\_values(by='MI Score', ascending=True).reset\_index(drop=True)

# Step 6: Compute the threshold (mean score) and select top features

threshold = mi\_df["MI Score"].mean()

tomim\_selected\_features = mi\_df[mi\_df["MI Score"] > threshold]["Feature"].tolist()

# Step 7: Print results

print("\n📊 All Features with MI Scores:")

print(mi\_df.sort\_values(by="MI Score", ascending=False).to\_string(index=False))

print(f"\n📌 Mean MI Score Threshold: {threshold:.4f}")

print(f"✅ Selected Features ({len(tomim\_selected\_features)}):")

print(tomim\_selected\_features)

# Step 8: Clean Visualization with Seaborn

sns.set(style="whitegrid")

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

# Color coding: selected vs not selected

colors = ['#4CAF50' if score > threshold else '#B0BEC5' for score in mi\_df['MI Score']]

# Plot horizontal bar chart

plt.barh(mi\_df['Feature'], mi\_df['MI Score'], color=colors)

plt.axvline(threshold, color='red', linestyle='--', linewidth=2, label=f'Mean Threshold: {threshold:.4f}')

# Title and labels

plt.title("Feature Importance (Mutual Information - TOMIM)", fontsize=16, pad=15)

plt.xlabel("Mutual Information Score", fontsize=13)

plt.ylabel("Features", fontsize=13)

plt.legend()

plt.tight\_layout()

plt.show()

**Selecting TOPCA Features**

import pandas as pd

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import numpy as np

# Load preprocessed dataset

df = pd.read\_csv("PCOS\_after\_preprocessing\_1.csv")

# Split features and target

X = df.drop(columns=["PCOS (Y/N)"])

y = df["PCOS (Y/N)"]

# Step 1: Apply PCA (initially full components)

pca = PCA()

X\_pca = pca.fit\_transform(X)

# Step 2: Plot cumulative explained variance

cumulative\_variance = np.cumsum(pca.explained\_variance\_ratio\_)

plt.figure(figsize=(8, 5))

plt.plot(cumulative\_variance, marker='o', color='green')

plt.axhline(y=0.95, color='red', linestyle='--', label='95% Variance Threshold')

plt.xlabel("Number of Principal Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("Explained Variance by PCA Components")

plt.grid(True)

plt.legend()

plt.tight\_layout()

plt.show()

# Step 3: Determine number of components to retain 95% variance

num\_components = np.argmax(cumulative\_variance >= 0.95) + 1

print(f"\n✅ Number of components to retain 95% variance: {num\_components}")

# Step 4: Re-fit PCA with only required components

pca\_optimal = PCA(n\_components=num\_components)

pca\_optimal.fit(X)

# Step 5: Get original feature scores from absolute component weights

components = np.abs(pca\_optimal.components\_)

feature\_scores = components.sum(axis=0)

# Step 6: Map features to their scores and select top features

feature\_importance = pd.DataFrame({

'Feature': X.columns,

'PCA Score': feature\_scores

}).sort\_values(by="PCA Score", ascending=False).reset\_index(drop=True)

# Store top features based on number of components

topca\_selected\_features = feature\_importance.head(num\_components)["Feature"].tolist()

# Print selected features

print(f"\n📌 Selected {len(topca\_selected\_features)} features using TOPCA:")

print(topca\_selected\_features)