**Mata Kuliah Analisis Multivariat**

**Laporan Tugas Analisis Diskriminan: Klasterisasi Golongan Berdasarkan Blood Pressure dan Age**

**Dosen Pengampu:  Wahyu Sri Utami, S.Si., M.Sc.**



**Disusun oleh:**

* Fadilah Ratu Azzahra (5232811015)
* Lathif Ramadhan  (5231811022)

**PROGRAM STUDI SAINS DATA  
FAKULTAS SAINS DAN TEKNOLOGI  
UNIVERSITAS TEKNOLOGI YOGYAKARTA   
YOGYAKARTA**

**2024**

# Daftar Isi

[Daftar Isi II](#_Toc200362769)

[Deskripsi Dataset IV](#_Toc200362770)

[1. Analisis Diskriminan dengan Perhitungan Manual (Excel) 1](#_Toc200362771)

[1.1. Sheet “Data\_Raw” 1](#_Toc200362772)

[1.2. Sheet “Perhitungan\_Awal” 2](#_Toc200362773)

[1.2.1. Identifikasi Data Kelompok 2](#_Toc200362774)

[1.2.2. Hitung Jumlah Observasi SI (n₁ dan n₂) 3](#_Toc200362775)

[1.2.3. Hitung Vektor Rata-Rata (Mean Vector) untuk Setiap Grup 3](#_Toc200362776)

[1.2.4. Hitung Matriks Varians-Kovarians untuk Setiap Grup (S₀ dan S₁) 3](#_Toc200362777)

[1.2.5. Hitung Matriks Varians-Kovarians Gabungan (S\_pooled) 3](#_Toc200362778)

[1.2.6. Hitung Invers dari Matriks S\_pooled (S\_pooled⁻¹) 3](#_Toc200362779)

[1.2.7. Hitung Vektor Koefisien Diskriminan (b) 3](#_Toc200362780)

[1.3. Sheet “Model\_Diskriminan” 3](#_Toc200362781)

[1.3.1. Fungsi Diskriminan (BENTUK Z = b₁X₁ + b₂X₂) 4](#_Toc200362782)

[1.3.2. Hitung Rata-Rata Skor Diskriminan Per Group (CENTROIDS Z̄₀ dan Z̄₁) 4](#_Toc200362783)

[1.3.3. Hitung CutOff Score (Z\_c) 4](#_Toc200362784)

[1.4. Sheet “Simulasi\_Prediksi” 4](#_Toc200362785)

[1.5. Sheet “Evaluasi” 5](#_Toc200362786)

[1.6. Kesimpulan 6](#_Toc200362787)

[2. Analisis Diskriminan dengan Perhitungan Manual (Python) 8](#_Toc200362788)

[2.1. Library Importation and Data Loading 8](#_Toc200362789)

[2.2. Data Preparation for Manual LDA 10](#_Toc200362790)

[2.2.1. Mengidentifikasi Variabel Fitur dan Target 10](#_Toc200362791)

[2.2.2. Memisahkan Data Berdasarkan Kelas 10](#_Toc200362792)

[Step 1: Calculate Mean Vectors for Each Class (Hitung Vektor Rata-rata untuk Setiap Kelas) 11](#_Toc200362793)

[2.1.a. Menghitung vektor rata-rata untuk Kelas 0 12](#_Toc200362794)

[2.1.b. Menghitung vektor rata-rata untuk Kelas 1 12](#_Toc200362795)

[2.1.c. Menghitung vektor rata-rata keseluruhan (overall mean) untuk semua data 12](#_Toc200362796)

[Step 2: Calculate Scatter Matrices (Hitung Matriks Sebaran) 12](#_Toc200362797)

[2.2.a. Within-Class Scatter Matrix (S\_W) (Matriks Sebaran Dalam Kelas) 13](#_Toc200362798)

[2.2.b. Between-Class Scatter Matrix (S\_B) (Matriks Sebaran Antar Kelas) 14](#_Toc200362799)

[Step 3: Solve the Generalized Eigenvalue Problem (Selesaikan Masalah Eigenvalue Umum) 14](#_Toc200362800)

[Step 4: Project Data onto the Discriminant Axis (LD Scores) (Proyeksikan Data ke Sumbu Diskriminan (Skor LD)) 17](#_Toc200362801)

[Step 5: Determine a Threshold for Classification (Tentukan Ambang Batas untuk Klasifikasi) 18](#_Toc200362802)

[Step 6: Formulate the Discriminant Function and Classify Data (Rumuskan Fungsi Diskriminan dan Klasifikasikan Data) 20](#_Toc200362803)

[Step 7: Reviewing the Formulated Manual Discriminant Function (Meninjau Rumus Fungsi Diskriminan Manual yang Telah Dirumuskan) 25](#_Toc200362804)

[Step 8: Making Predictions on New Data (Manual) (Membuat Prediksi pada Data Baru (Manual)) 26](#_Toc200362805)

[2.3. Kesimpulan 29](#_Toc200362806)

[3. Analisis Diskriminan Menggunakan Library Scikit-Learn (Python) 30](#_Toc200362807)

[3.1. Import Libraries 30](#_Toc200362808)

[3.2. Load Dataset 30](#_Toc200362809)

[3.3. Data Preprocessing and Preparation 31](#_Toc200362810)

[3.3.1. Memisahkan Fitur (X) dan Target (y) 32](#_Toc200362811)

[3.3.2. Membagi Data menjadi Set Pelatihan dan Pengujian 33](#_Toc200362812)

[3.3.3. Penskalaan Fitur (Feature Scaling) 33](#_Toc200362813)

[3.4. Model Building: Linear Discriminant Analysis (LDA) 34](#_Toc200362814)

[3.4.1. Menginisialisasi model LDA 34](#_Toc200362815)

[3.4.2. Melatih model LDA menggunakan data pelatihan yang sudah diskalakan 35](#_Toc200362816)

[3.5. Model Evaluation 35](#_Toc200362817)

[3.5.1. Membuat prediksi pada data pengujian 36](#_Toc200362818)

[3.5.2. Menghitung Akurasi 36](#_Toc200362819)

[3.5.3. Menampilkan Confusion Matrix 36](#_Toc200362820)

[3.5.4. Menampilkan Laporan Klasifikasi 37](#_Toc200362821)

[3.6. Extracting Discriminant Function Coefficients 37](#_Toc200362822)

[3.6.1. Mengekstrak Koefisien 38](#_Toc200362823)

[3.6.2. Mengekstrak Intercept 38](#_Toc200362824)

[3.6.3. Menyajikan Fungsi Diskriminan Linier 38](#_Toc200362825)

[3.7. Visualization 39](#_Toc200362826)

[3.7.1. Distribusi Fitur per Kelas 40](#_Toc200362827)

[3.7.2. Scatter Plot Fitur (Scaled) dengan Batas Keputusan LDA 40](#_Toc200362828)

[3.7.3. Distribusi Skor Diskriminan (LD Scores) 41](#_Toc200362829)

[3.8. Making Predictions on New Data 43](#_Toc200362830)

[3.8.1. Menyiapkan Data Baru 43](#_Toc200362831)

[3.8.2. Menerapkan Penskalaan Fitur 44](#_Toc200362832)

[3.8.3. Membuat Prediksi 44](#_Toc200362833)

[3.9. Kesimpulan 45](#_Toc200362834)

[Kesimpulan Akhir 47](#_Toc200362835)

# Deskripsi Dataset

**Sumber Data dan Konteks Awal**

Dataset yang digunakan dalam analisis diskriminan ini bersumber dari data rekam medis yang dikumpulkan oleh National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK). Data asli mencakup catatan medis pasien perempuan keturunan Indian Pima berusia minimal 21 tahun, dengan fokus utama pada prediksi kejadian diabetes berdasarkan berbagai parameter diagnostik.

Link sumber dataset: <https://www.kaggle.com/datasets/whenamancodes/predict-diabities>

**Modifikasi Dataset untuk Analisis Diskriminan**

Untuk keperluan analisis diskriminan saat ini, dataset asli telah melalui proses modifikasi. Sebelumnya, pada analisis klastering, dataset ini dieksplorasi untuk mengidentifikasi kelompok-kelompok alami berdasarkan karakteristik pasien. Hasil dari analisis klastering tersebut, yang optimalnya menghasilkan dua kelompok (cluster), kemudian diadopsi dan disesuaikan untuk membentuk variabel target kategorikal dengan dua level (misalnya, Cluster 0 dan Cluster 1). Modifikasi ini dilakukan untuk memenuhi prasyarat analisis diskriminan yang memerlukan variabel dependen kategorikal yang telah terdefinisi dengan jelas. Proses pembentukan dua cluster target ini didokumentasikan dalam notebook Colab terpisah (<https://github.com/LatiefDataVisionary/multivariate-analysis-college-task/blob/main/src/KMeans_w11_for_LinearDiscriminantAnalysis_w12.ipynb>).

**Variabel yang Digunakan dalam Analisis Diskriminan**

Dari keseluruhan variabel yang tersedia, analisis diskriminan ini secara spesifik menggunakan dua variabel numerik sebagai variabel independen (prediktor) dan satu variabel kategorikal sebagai variabel dependen (target):

1. **Variabel Independen (Fitur Prediktor):**
   * **BloodPressure (Tekanan Darah Diastolik)**

Variabel ini mengukur tekanan darah diastolik pasien dalam satuan milimeter air raksa (mmHg). Tekanan darah merupakan indikator penting kesehatan kardiovaskular, dan variasi abnormal seringkali dikaitkan dengan risiko gangguan metabolik, termasuk predisposisi diabetes. Dalam dataset yang digunakan, nilai tekanan darah diastolik berkisar antara 0 hingga 122 mmHg (perlu dicatat bahwa nilai 0 mungkin mengindikasikan data hilang atau kesalahan pengukuran yang memerlukan perhatian khusus pada analisis data asli, namun untuk analisis ini digunakan sebagaimana adanya dalam dataset termodifikasi).

* + **Age (Usia)**

Variabel ini merepresentasikan usia pasien dalam satuan tahun. Usia adalah faktor demografis signifikan yang secara luas diakui mempengaruhi prevalensi berbagai kondisi kesehatan, termasuk diabetes tipe 2, di mana risiko umumnya meningkat seiring bertambahnya usia. Rentang usia pasien dalam dataset ini adalah dari 21 hingga 81 tahun.

1. **Variabel Dependen (Target Klasifikasi):**
   * **Cluster**

Variabel kategorikal ini merupakan hasil modifikasi dari analisis klastering sebelumnya dan berfungsi sebagai label kelompok yang ingin diprediksi atau dibedakan oleh model diskriminan. Variabel ini memiliki dua kategori, yaitu Cluster 0 dan Cluster 1, yang merepresentasikan dua profil pasien berbeda berdasarkan karakteristik BloodPressure dan Age.

**Relevansi dan Signifikansi Pemilihan Variabel**

Pemilihan BloodPressure dan Age sebagai variabel prediktor didasarkan pada relevansi klinis dan kemampuan potensialnya dalam membedakan profil risiko kesehatan antar individu. Kombinasi kedua variabel ini diharapkan dapat membantu mengidentifikasi dan membedakan kelompok pasien dengan karakteristik yang serupa, misalnya, antara individu yang lebih muda dengan tekanan darah yang relatif lebih tinggi dibandingkan dengan individu yang lebih tua dengan tekanan darah yang mungkin lebih stabil atau terkontrol, atau sebaliknya.

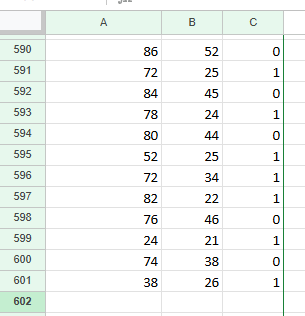
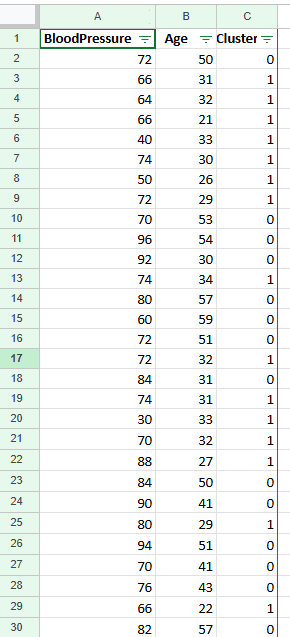
Tujuan analisis diskriminan ini adalah untuk membangun sebuah fungsi matematis yang dapat secara efektif membedakan individu ke dalam salah satu dari dua kategori Cluster tersebut, berdasarkan nilai BloodPressure dan Age mereka. Hasil analisis ini diharapkan dapat memberikan pemahaman lebih lanjut mengenai faktor-faktor yang paling berkontribusi dalam pemisahan antar kelompok dan menguji kemampuan klasifikasi dari variabel yang dipilih.

# 1. Analisis Diskriminan dengan Perhitungan Manual (Excel)

Penjelasan langkah demi langkah perhitungan LDA menggunakan Excel, termasuk rumus, matriks varians-kovarians, serta interpretasi hasil klasifikasi secara manual.

Link Spreadsheet Pengerjaan:  <https://docs.google.com/spreadsheets/d/151yl1aG28wJ04QlZqihvnpe0fLs9M78c/edit?usp=sharing&ouid=114513767125494582973&rtpof=true&sd=true>

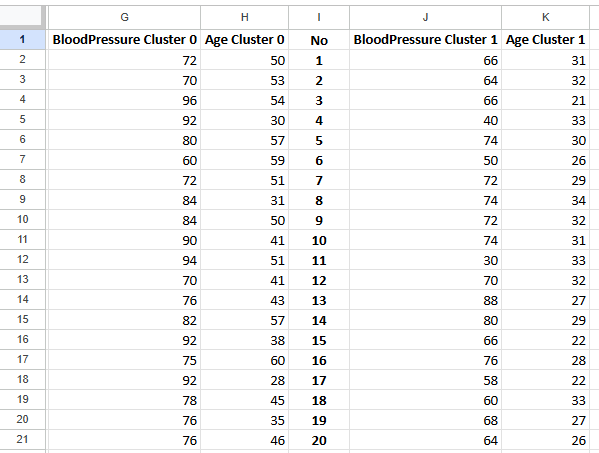
* 1. **Sheet “Data\_Raw”**
* **Isi**: Data mentah pengukuran tekanan darah (BloodPressure) dan usia (Age) pasien, beserta label Cluster (0 atau 1).
* **Struktur Kolom**:
  + Kolom A-C: Data asli (BloodPressure, Age, Cluster).
  + Kolom G-I dan J-K: Salinan data kolom A-C (kemungkinan duplikat atau untuk referensi perhitungan).
* **Total Data**: 600 baris.
* **Fungsi**: Menyimpan data dasar untuk analisis klaster.

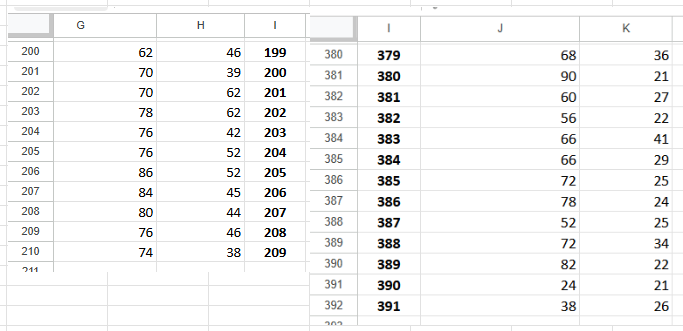


## Sheet “Perhitungan\_Awal”

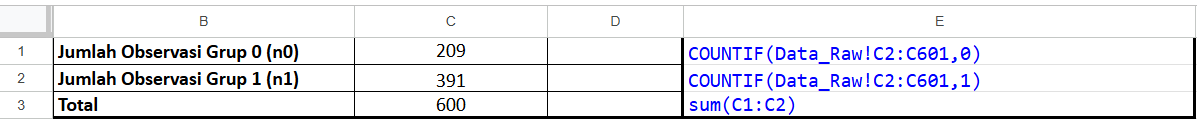
Sheet ini menghitung Linear Discriminant Analysis (LDA) untuk memisahkan dua klaster. Tahapannya:

### 1.2.1. Identifikasi Data Kelompok

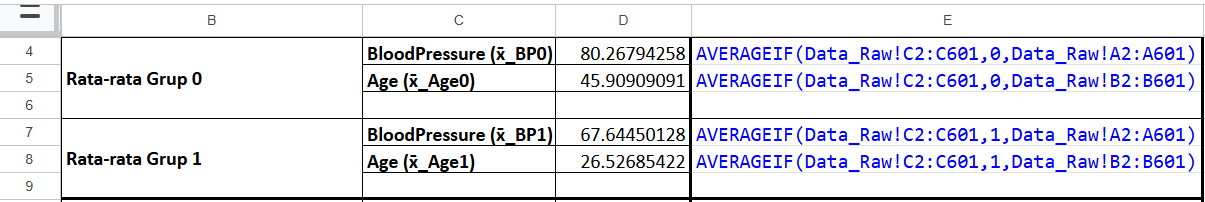




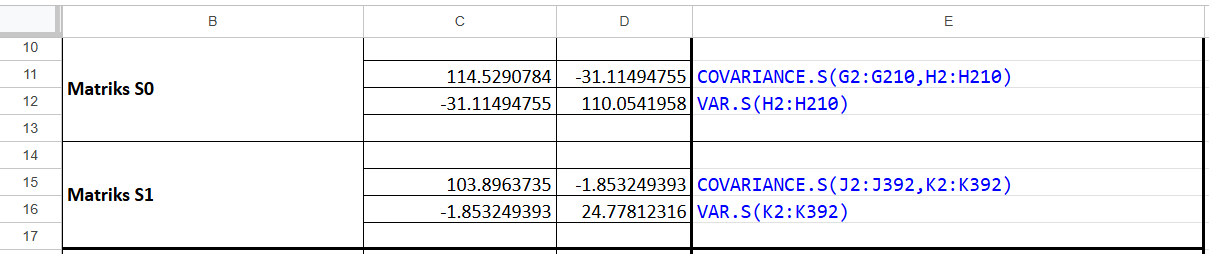
### 1.2.2. Hitung Jumlah Observasi SI (n₁ dan n₂)



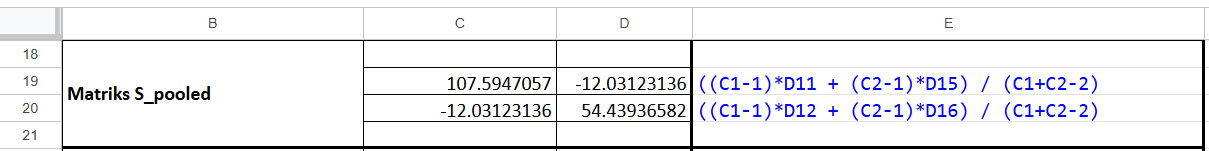
### 1.2.3. Hitung Vektor Rata-Rata (Mean Vector) untuk Setiap Grup



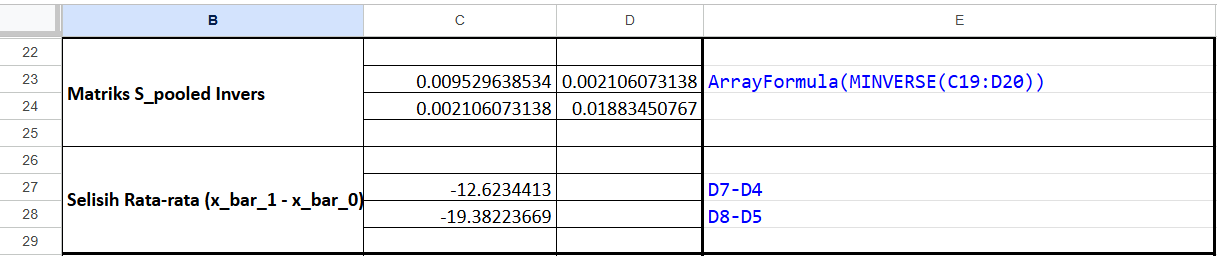
### 1.2.4. Hitung Matriks Varians-Kovarians untuk Setiap Grup (S₀ dan S₁)



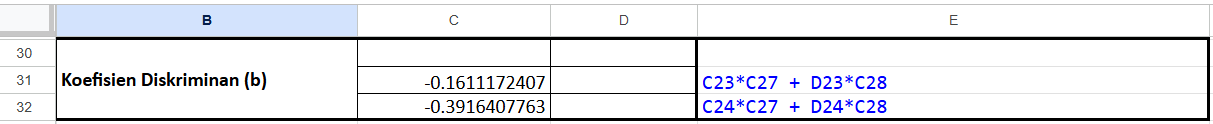
### 1.2.5. Hitung Matriks Varians-Kovarians Gabungan (S\_pooled)



### 1.2.6. Hitung Invers dari Matriks S\_pooled (S\_pooled⁻¹)



### 1.2.7. Hitung Vektor Koefisien Diskriminan (b)



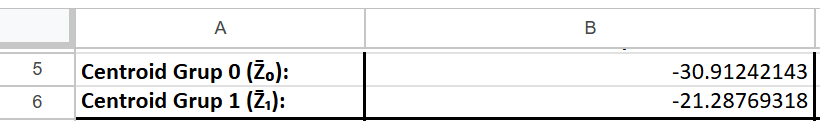
## Sheet “Model\_Diskriminan”

### 1.3.1. Fungsi Diskriminan (BENTUK Z = b₁X₁ + b₂X₂)

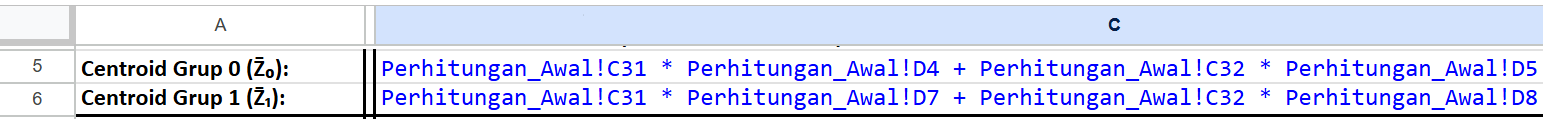
***Z = -0.1611 \* BloodPressure + -0.3916 \* Age***

CONCATENATE("Z = ",ROUND(Perhitungan\_Awal!C31,4)," \* BloodPressure + ",ROUND(Perhitungan\_Awal!C32,4)," \* Age")

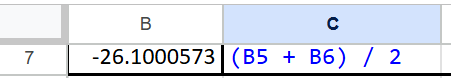
### 1.3.2. Hitung Rata-Rata Skor Diskriminan Per Group (CENTROIDS Z̄₀ dan Z̄₁)



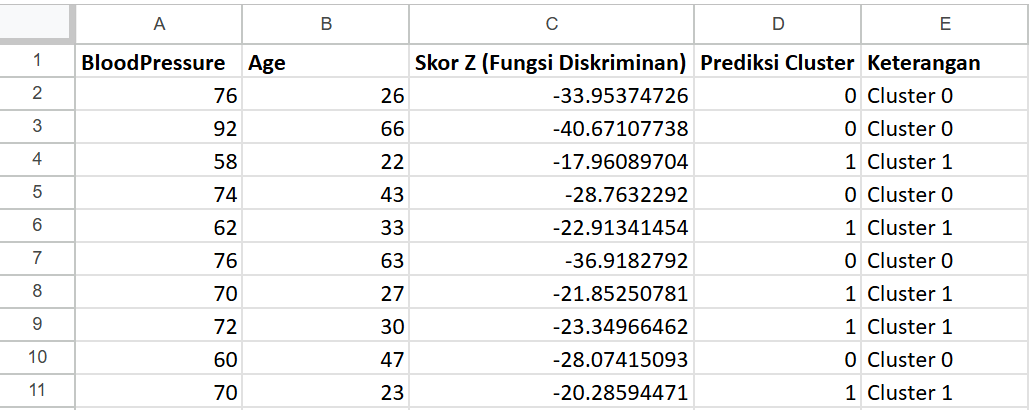
Rumus:



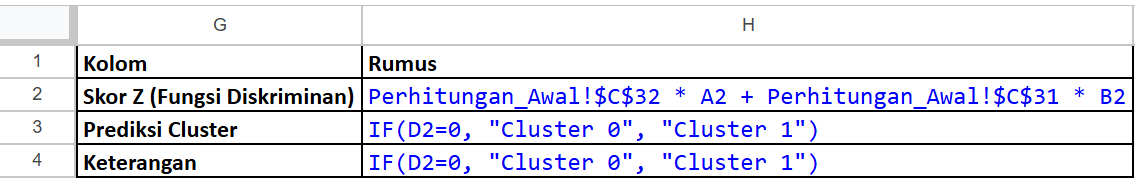
### 1.3.3. Hitung CutOff Score (Z\_c)



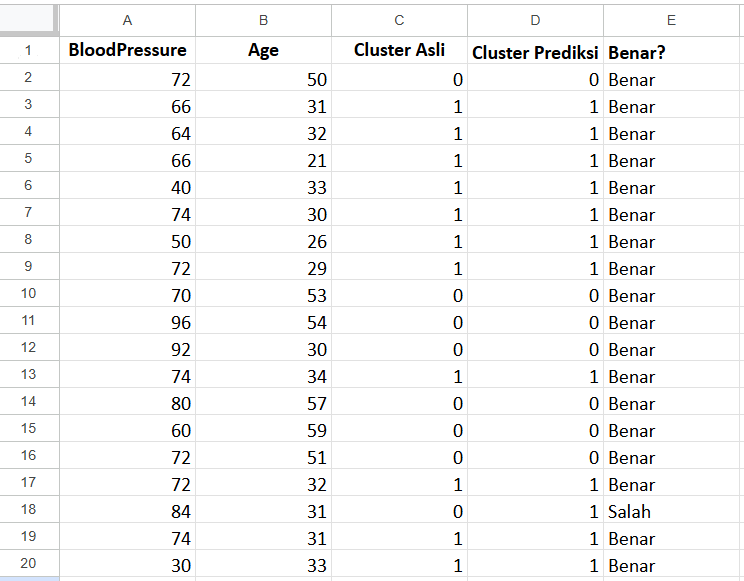
## Sheet “Simulasi\_Prediksi”

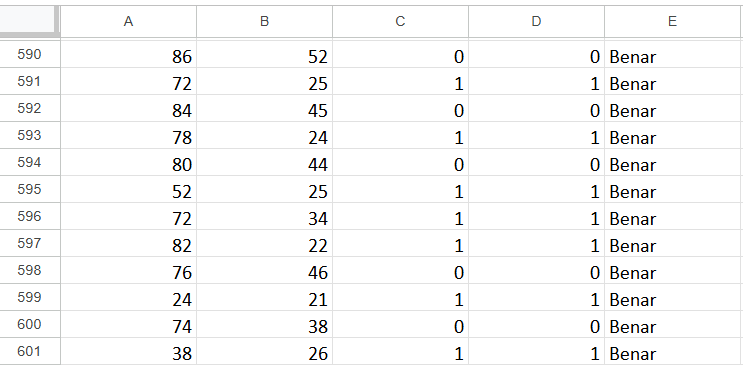


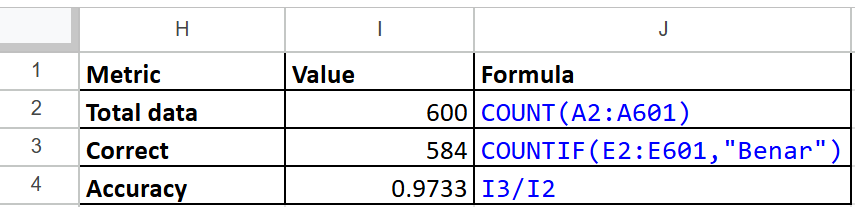
Rumus:

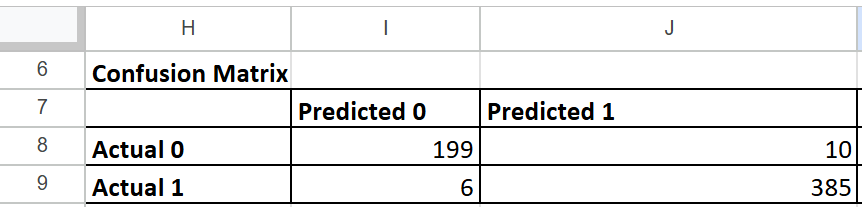


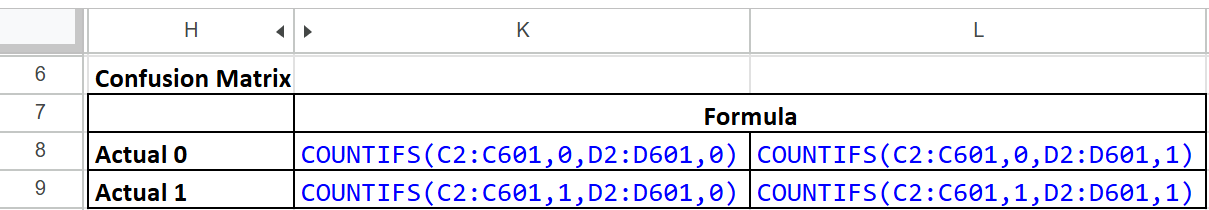
## Sheet “Evaluasi”

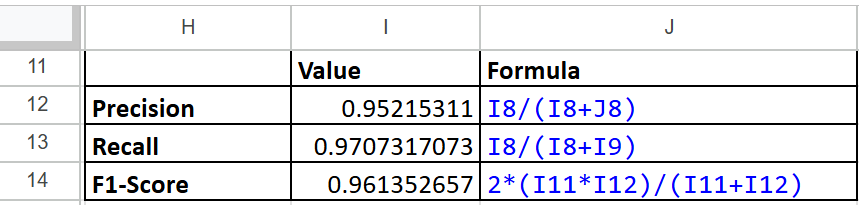












## 1.6. Kesimpulan

Berdasarkan perhitungan manual Analisis Diskriminan Linier (LDA) yang telah dilakukan menggunakan Microsoft Excel untuk klasterisasi golongan berdasarkan variabel BloodPressure dan Age pada 600 data pasien, dapat ditarik beberapa kesimpulan penting.

Pertama, berhasil dibentuk sebuah fungsi diskriminan linier yang bertujuan untuk membedakan antara dua kelompok (Cluster 0 dan Cluster 1). Fungsi diskriminan yang dihasilkan adalah:  
**Z = -0.1611 \* BloodPressure - 0.3916 \* Age**

Fungsi ini menunjukkan bagaimana kombinasi linear dari BloodPressure dan Age dapat digunakan untuk menghasilkan skor diskriminan (Skor Z) bagi setiap individu. Koefisien negatif untuk kedua variabel mengindikasikan bahwa peningkatan nilai BloodPressure maupun Age cenderung menurunkan Skor Z, yang selanjutnya mempengaruhi klasifikasi ke dalam salah satu cluster.

Kedua, dari Skor Z tersebut, telah dihitung centroid (rata-rata Skor Z) untuk masing-masing kelompok. Centroid untuk Grup 0 (Ž<sub>0</sub>) adalah **-30.9124** dan untuk Grup 1 (Ž<sub>1</sub>) adalah **-21.2877**. Berdasarkan kedua centroid ini, berhasil ditentukan sebuah nilai *cutoff score* (Z\_c) sebesar **-26.1001**. Nilai *cutoff* ini berfungsi sebagai batas keputusan: individu dengan Skor Z di atas nilai ini akan diklasifikasikan ke dalam Grup 1, sedangkan individu dengan Skor Z di bawah atau sama dengan nilai ini akan diklasifikasikan ke dalam Grup 0 (dengan asumsi Grup 1 memiliki centroid Skor Z yang lebih tinggi, yang sesuai dengan nilai centroid yang diperoleh).

Ketiga, evaluasi terhadap model diskriminan manual ini menunjukkan tingkat performa yang cukup baik. Dari total 600 data yang dianalisis, model berhasil mengklasifikasikan **584** data dengan benar, menghasilkan akurasi keseluruhan sebesar **97.33%**. Rincian lebih lanjut dari confusion matrix menunjukkan bahwa untuk Kelas Aktual 0, model memprediksi 199 benar (True Negative) dan 10 salah (False Positive). Untuk Kelas Aktual 1, model memprediksi 385 benar (True Positive) dan 6 salah (False Negative).

Berdasarkan metrik evaluasi tersebut, diperoleh nilai **Precision sebesar 0.9522** (kemampuan model untuk tidak salah mengklasifikasikan individu sebagai Grup 1 padahal sebenarnya Grup 0) dan **Recall sebesar 0.9707** (kemampuan model untuk menemukan semua individu yang sebenarnya termasuk Grup 1). Nilai **F1-Score**, yang merupakan rata-rata harmonik dari Precision dan Recall, mencapai **0.9614**, yang mengindikasikan keseimbangan yang baik antara kedua metrik tersebut.

Secara keseluruhan, perhitungan manual LDA menggunakan Excel telah berhasil menghasilkan model diskriminan yang mampu membedakan dua klaster berdasarkan tekanan darah dan usia dengan tingkat akurasi dan performa yang tinggi pada dataset yang digunakan. Meskipun proses manual memerlukan ketelitian dan pemahaman langkah demi langkah, hasil yang diperoleh memberikan dasar yang kuat untuk memahami mekanisme kerja analisis diskriminan.

# 2. Analisis Diskriminan dengan Perhitungan Manual (Python)

Implementasi kode Python untuk menghitung LDA tanpa library, mencakup pembuatan matriks gabungan, invers matriks, dan derivasi koefisien diskriminan secara manual.

Link Notebook Pengerjaan: <https://github.com/LatiefDataVisionary/multivariate-analysis-college-task/blob/main/src/DiscriminantAnalysis_week12_manually.ipynb>

## 2.1. Library Importation and Data Loading

Pada bagian awal ini, dilakukan impor terhadap pustaka-pustaka Python yang dibutuhkan dan pemuatan dataset.

1. **Impor Pustaka**:

* **Pandas**: Digunakan untuk manipulasi data tabular, khususnya untuk membaca file CSV dan bekerja dengan struktur DataFrame.
* **NumPy**: Merupakan pustaka fundamental untuk komputasi numerik, menyediakan dukungan untuk operasi array dan matriks yang efisien, yang akan sangat berguna dalam perhitungan manual LDA.

1. **Pemuatan Dataset**:

* Dataset dimuat dari URL yang disediakan (https://raw.githubusercontent.com/LatiefDataVisionary/multivariate-analysis-college-task/refs/heads/main/datasets/dataset\_LDA\_2clusters.csv) menggunakan fungsi pd.read\_csv() dari Pandas.
* Hasil pemuatan data disimpan dalam sebuah DataFrame Pandas yang dinamakan df.

1. **Inspeksi Awal Data**:

* df.head(): Menampilkan lima baris pertama dari DataFrame untuk mendapatkan gambaran awal mengenai struktur dan isi data.
* df.info(): Memberikan ringkasan informasi tentang DataFrame, termasuk jumlah entri, tipe data setiap kolom, dan jumlah nilai non-null. Ini berguna untuk memeriksa apakah ada data yang hilang atau tipe data yang tidak sesuai.
* df.shape: Menampilkan dimensi dari DataFrame (jumlah baris, jumlah kolom).
* df.describe(): Menampilkan statistik deskriptif dasar untuk kolom-kolom numerik dalam DataFrame, seperti mean, standar deviasi, nilai minimum, maksimum, dan kuartil.

Langkah-langkah ini penting untuk memastikan bahwa data telah dimuat dengan benar dan untuk mendapatkan pemahaman awal tentang dataset sebelum melanjutkan ke tahap analisis dan perhitungan selanjutnya.

#### 2.1.1. Import library yang diperlukan

##### import pandas as pd

##### import numpy as np

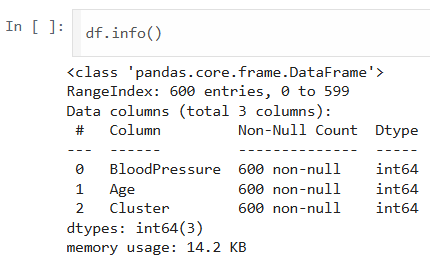
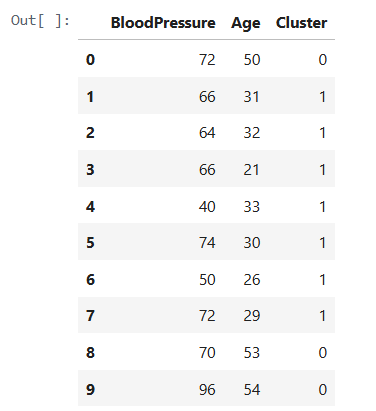
##### import matplotlib.pyplot as plt

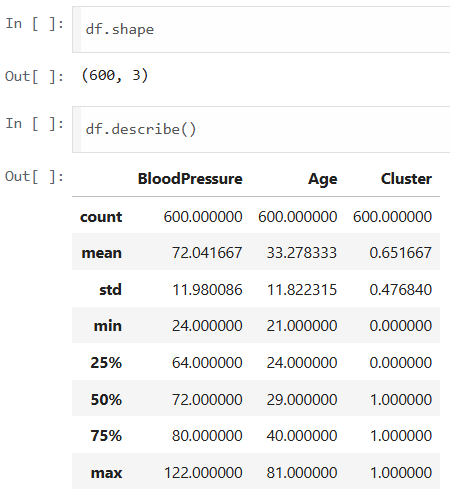
#### 2.1.2. Pemuatan dataset dari URL

data\_url = 'https://raw.githubusercontent.com/LatiefDataVisionary/multivariate-analysis-college-task/refs/heads/main/datasets/dataset\_LDA\_2clusters.csv'

df = pd.read\_csv(data\_url)

#### 2.1.3. Informasi dataset





##### # Menghitung jumlah kemunculan setiap nilai di kolom 'Cluster'

##### counts = df['Cluster'].value\_counts()

##### # Menghitung persentase kemunculan setiap nilai di kolom 'Cluster'

##### percentages = df['Cluster'].value\_counts(normalize=True) \* 100

##### # Menggabungkan kedua hasil ke dalam satu DataFrame

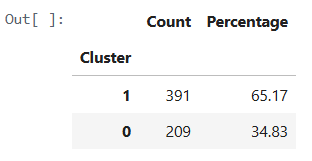
##### result = pd.DataFrame({'Count': counts, 'Percentage': percentages})

##### # Membulatkan kolom 'Percentage' hingga 2 angka di belakang koma

##### result['Percentage'] = result['Percentage'].round(2)

##### # Menampilkan hasilnya yang sudah dibulatkan

##### result



## 2.2. Data Preparation for Manual LDA

Sebelum memulai perhitungan inti LDA secara manual, data perlu disiapkan. Langkah-langkah utamanya adalah:

1. **Mengidentifikasi Variabel Fitur (Features) dan Target (Cluster)**:

* Variabel fitur adalah kolom-kolom yang akan digunakan sebagai prediktor. Berdasarkan dataset, fitur yang digunakan adalah BloodPressure dan Age. Variabel-variabel ini akan diekstrak ke dalam sebuah array NumPy atau DataFrame Pandas (features\_X).
* Variabel target adalah kolom yang menunjukkan keanggotaan kelas. Dalam dataset ini, kolom Cluster (dengan nilai 0 dan 1) adalah variabel target (target\_y).

1. **Memisahkan Data Berdasarkan Kelas**:

* Untuk perhitungan matriks sebaran dalam kelas (within-class scatter matrix) dan vektor rata-rata per kelas, data perlu dipisahkan menjadi subset berdasarkan nilai pada kolom target (Cluster).
* Akan dibuat dua subset data:
  + df\_class0: Berisi semua observasi di mana Cluster adalah 0.
  + df\_class1: Berisi semua observasi di mana Cluster adalah 1.
* Dari subset ini, akan diekstrak nilai fitur untuk masing-masing kelas (misalnya, features\_X\_class0 dan features\_X\_class1).

Persiapan ini akan mempermudah akses ke data yang relevan untuk setiap kelas selama perhitungan matematis LDA berikutnya.

### 2.2.1. Mengidentifikasi Variabel Fitur dan Target

##### feature\_column\_names = ['BloodPressure', 'Age']

##### target\_column\_name = 'Cluster'

##### *# Ekstrak fitur ke dalam array NumPy untuk kemudahan perhitungan matriks*

##### *# Kita akan menggunakan .values untuk mendapatkan array NumPy*

##### features\_X = df[feature\_column\_names].values

##### target\_y = df[target\_column\_name].values

##### print(f"Shape of features\_X: {features\_X.shape}")

##### print(f"Shape of target\_y: {target\_y.shape}")

##### print(f"First 5 rows of features\_X:\n{features\_X[:5]}")

##### print(f"First 5 values of target\_y:\n{target\_y[:5]}")

##### **Shape of features\_X: (600, 2)**

##### **Shape of target\_y: (600,)**

##### **First 5 rows of features\_X:**

##### **[[72 50]**

##### **[66 31]**

##### **[64 32]**

##### **[66 21]**

**[40 33]]**

##### **First 5 values of target\_y:**

##### **[0 1 1 1 1]**

### 2.2.2. Memisahkan Data Berdasarkan Kelas

##### # Dapatkan DataFrame dan array fitur untuk masing-masing kelas

##### # Kelas 0

df\_class0 = df[df[target\_column\_name] == 0]

##### features\_X\_class0 = df\_class0[feature\_column\_names].values

##### # Kelas 1

##### df\_class1 = df[df[target\_column\_name] == 1]

##### features\_X\_class1 = df\_class1[feature\_column\_names].values

##### print(f"Shape of features\_X\_class0 (Cluster 0): {features\_X\_class0.shape}")

##### print(f"First 5 rows of features\_X\_class0:\n{features\_X\_class0[:5]}")

##### print(f"Number of samples in Class 0: {len(df\_class0)}")

##### print("\n" + "="\*50 + "\n")

##### print(f"Shape of features\_X\_class1 (Cluster 1): {features\_X\_class1.shape}")

##### print(f"First 5 rows of features\_X\_class1:\n{features\_X\_class1[:5]}")

##### print(f"Number of samples in Class 1: {len(df\_class1)}")

##### print("\n" + "="\*50 + "\n")

##### *# Verifikasi bahwa total sampel dari kedua kelas sama dengan total sampel di df awal*

##### *# Ini juga mengasumsikan tidak ada NaN yang dihapus atau kelas lain yang ada.*

##### *# Total baris adalah 600, dan kita memuat 600 baris.*

##### if len(df\_class0) + len(df\_class1) == len(df):

##### print("Data separation by class seems correct. Total samples match.")

##### else:

##### print("Warning: Mismatch in total samples after class separation. Check for other classes or NaN issues if any were handled.")

##### **Shape of features\_X\_class0 (Cluster 0): (209, 2)**

##### **First 5 rows of features\_X\_class0:**

##### **[[72 50]**

##### **[70 53]**

##### **[96 54]**

##### **[92 30]**

##### **[80 57]]**

##### **Number of samples in Class 0: 209**

##### **==================================================**

##### **Shape of features\_X\_class1 (Cluster 1): (391, 2)**

##### **First 5 rows of features\_X\_class1:**

##### **[[66 31]**

##### **[64 32]**

##### **[66 21]**

##### **[40 33]**

##### **[74 30]]**

##### **Number of samples in Class 1: 391**

## Step 1: Calculate Mean Vectors for Each Class (Hitung Vektor Rata-rata untuk Setiap Kelas)

Langkah pertama dalam perhitungan manual LDA adalah menghitung vektor rata-rata (mean vector) untuk setiap kelas. Vektor rata-rata ini merepresentasikan titik pusat (centroid) dari data untuk masing-masing kelas dalam ruang fitur.

1. **Definisi Vektor Rata-rata Kelas**:

* Untuk sebuah kelas k dengan N\_k sampel dan d fitur, vektor rata-rata **m**k adalah sebuah vektor berdimensi d di mana setiap elemennya adalah rata-rata dari fitur yang bersangkutan untuk semua sampel dalam kelas k.
* Secara matematis, untuk fitur ke-j pada kelas k: (mk)j = (1/Nk) \* Σi=1N\_k (xi)j di mana (xi)j adalah nilai fitur ke-j dari sampel ke-i dalam kelas k.

1. **Perhitungan Menggunakan NumPy**:
   * Menggunakan data fitur yang telah dipisahkan per kelas (features\_X\_class0 dan features\_X\_class1), fungsi np.mean() dari NumPy dapat digunakan.
   * axis=0 ditentukan dalam np.mean() untuk menghitung rata-rata sepanjang kolom (yaitu, untuk setiap fitur secara terpisah).
   * Hasilnya akan berupa dua vektor:
     + mean\_vector\_class0: Vektor rata-rata untuk Kelas 0.
     + mean\_vector\_class1: Vektor rata-rata untuk Kelas 1.
   * Juga, akan dihitung vektor rata-rata keseluruhan (overall mean vector) dari semua data fitur (features\_X), yang akan berguna nanti saat menghitung matriks sebaran antar kelas (between-class scatter matrix).

Vektor rata-rata ini adalah komponen penting untuk menghitung matriks sebaran (scatter matrices) pada langkah berikutnya.

### 2.1.a. Menghitung vektor rata-rata untuk Kelas 0

##### # features\_X\_class0 adalah array NumPy dari fitur untuk kelas 0

##### mean\_vector\_class0 = np.mean(features\_X\_class0, axis=0)

##### print(f"Mean Vector for Class 0 (m₀) (Rata-rata Vektor Kelas 0): \n{mean\_vector\_class0}")

##### print(f" - Rata-rata BloodPressure (Kelas 0): {mean\_vector\_class0[0]:.4f}")

##### print(f" - Rata-rata Age (Kelas 0): {mean\_vector\_class0[1]:.4f}")

##### 

##### **Mean Vector for Class 0 (m₀) (Rata-rata Vektor Kelas 0):**

##### **[80.26794258 45.90909091]**

##### **- Rata-rata BloodPressure (Kelas 0): 80.2679**

##### **- Rata-rata Age (Kelas 0): 45.9091**

### 2.1.b. Menghitung vektor rata-rata untuk Kelas 1

##### # features\_X\_class1 adalah array NumPy dari fitur untuk kelas 1

##### mean\_vector\_class1 = np.mean(features\_X\_class1, axis=0)

##### print(f"Mean Vector for Class 1 (m₁) (Rata-rata Vektor Kelas 1): \n{mean\_vector\_class1}")

##### print(f" - Rata-rata BloodPressure (Kelas 1): {mean\_vector\_class1[0]:.4f}")

##### print(f" - Rata-rata Age (Kelas 1): {mean\_vector\_class1[1]:.4f}")

##### 

##### **Mean Vector for Class 1 (m₁) (Rata-rata Vektor Kelas 1):**

##### **[67.64450128 26.52685422]**

##### **- Rata-rata BloodPressure (Kelas 1): 67.6445**

##### **- Rata-rata Age (Kelas 1): 26.5269**

### 2.1.c. Menghitung vektor rata-rata keseluruhan (overall mean) untuk semua data

##### # features\_X adalah array NumPy dari semua fitur

##### overall\_mean\_vector = np.mean(features\_X, axis=0)

##### print(f"Overall Mean Vector (m) (Rata-rata Vektor Keseluruhan): \n{overall\_mean\_vector}")

##### print(f" - Rata-rata BloodPressure (Keseluruhan): {overall\_mean\_vector[0]:.4f}")

##### print(f" - Rata-rata Age (Keseluruhan): {overall\_mean\_vector[1]:.4f}")

##### 

##### **Overall Mean Vector (m) (Rata-rata Vektor Keseluruhan):**

##### **[72.04166667 33.27833333]**

##### **- Rata-rata BloodPressure (Keseluruhan): 72.0417**

##### **- Rata-rata Age (Keseluruhan): 33.2783**

## Step 2: Calculate Scatter Matrices (Hitung Matriks Sebaran)

Matriks sebaran (scatter matrices) mengukur dispersi data. Ada dua jenis matriks sebaran utama yang perlu dihitung untuk LDA:

1. **Within-Class Scatter Matrix (SW) (Matriks Sebaran Dalam Kelas)**:

* SW mengukur sebaran (varians) observasi di sekitar rata-rata kelasnya masing-masing. Ini pada dasarnya adalah jumlah dari matriks kovarians individual untuk setiap kelas.
* Tujuannya adalah untuk meminimalkan sebaran ini pada sumbu diskriminan.
* Untuk setiap kelas k, matriks sebaran Sk dihitung sebagai: Sk = Σx ∈ Kelas k (x - mk)(x - mk)T di mana x adalah vektor fitur sampel, dan m\_k adalah vektor rata-rata kelas k.
* Kemudian, SW adalah jumlah dari semua Sk: SW = Σk Sk = S0 + S1 (untuk kasus dua kelas).
* Dalam praktiknya, seringkali lebih mudah menghitung matriks kovarians untuk setiap kelas (dibagi dengan Nk-1 atau Nk) dan kemudian menjumlahkannya (setelah dikalikan dengan Nk-1 atau Nk, jika Sk yang digunakan adalah matriks kovarians yang telah dinormalisasi). Alternatif lain, jika Sk adalah *sum of squared deviations*, maka langsung dijumlahkan. Untuk konsistensi dengan banyak teks, Sk sering didefinisikan sebagai (Nk-1) \* Covk atau Σ(x-mk)(x-mk)T.

1. **Between-Class Scatter Matrix (SB) (Matriks Sebaran Antar Kelas)**:

* SB mengukur sebaran (varians) antar rata-rata kelas relatif terhadap rata-rata keseluruhan data.
* Tujuannya adalah untuk memaksimalkan sebaran ini pada sumbu diskriminan, sehingga kelas-kelas menjadi sejauh mungkin terpisah.
* Untuk kasus dua kelas (Kelas 0 dan Kelas 1), SB dapat dihitung sebagai: SB = N0(m0 - m)(m0 - m)T + N1(m1 - m)(m1 - m)T di mana N0 dan N1 adalah jumlah sampel di Kelas 0 dan Kelas 1, m0 dan m1 adalah vektor rata-rata Kelas 0 dan Kelas 1, dan m adalah vektor rata-rata keseluruhan.
* Alternatif untuk dua kelas yang sering digunakan (yang akan kita gunakan karena kesederhanaannya dan menghasilkan arah eigenvector yang sama) adalah: SB = (m0 - m1)(m0 - m1)T Perbedaan utamanya adalah pada penskalaan eigenvalue, namun arah eigenvector (yang paling penting untuk sumbu diskriminan) tetap sama. Kita akan menggunakan pendekatan ini untuk SB karena kesederhanaan komputasi manualnya, meskipun definisi pertama lebih umum untuk multikelas.

Perhitungan matriks-matriks ini menggunakan operasi vektor dan matriks dari NumPy.

### 2.2.a. Within-Class Scatter Matrix (S\_W) (Matriks Sebaran Dalam Kelas)

#### 2.2.a.1. Matriks sebaran untuk Kelas 0 (S\_0)

##### **# features\_X\_class0 dan mean\_vector\_class0 sudah dihitung sebelumnya**

##### **scatter\_matrix\_class0 = np.zeros((features\_X.shape[1], features\_X.shape[1])) # Inisialisasi matriks nol d x d (d=jumlah fitur)**

##### **for row\_idx in range(features\_X\_class0.shape[0]):**

##### **row\_vector = features\_X\_class0[row\_idx, :].reshape(features\_X.shape[1], 1) # Vektor baris (d x 1)**

##### **mean\_vec\_c0 = mean\_vector\_class0.reshape(features\_X.shape[1], 1) # Vektor rata-rata kelas 0 (d x 1)**

##### **scatter\_matrix\_class0 += (row\_vector - mean\_vec\_c0).dot((row\_vector - mean\_vec\_c0).T)**

##### **print(f"Scatter Matrix for Class 0 (S₀) (Matriks Sebaran Kelas 0):\n{scatter\_matrix\_class0}")**

##### 

##### **Scatter Matrix for Class 0 (S₀) (Matriks Sebaran Kelas 0):**

##### **[[23746.99521531 -6471.90909091]**

##### **[-6471.90909091 22891.27272727]]**

#### 2.2.a.2. Matriks sebaran untuk Kelas 1 (S\_1)

##### # features\_X\_class1 dan mean\_vector\_class1 sudah dihitung sebelumnya

##### scatter\_matrix\_class1 = np.zeros((features\_X.shape[1], features\_X.shape[1])) # Inisialisasi matriks nol d x d

##### for row\_idx in range(features\_X\_class1.shape[0]):

##### row\_vector = features\_X\_class1[row\_idx, :].reshape(features\_X.shape[1], 1) # Vektor baris (d x 1)

##### mean\_vec\_c1 = mean\_vector\_class1.reshape(features\_X.shape[1], 1) # Vektor rata-rata kelas 1 (d x 1)

##### scatter\_matrix\_class1 += (row\_vector - mean\_vec\_c1).dot((row\_vector - mean\_vec\_c1).T)

##### print(f"Scatter Matrix for Class 1 (S₁) (Matriks Sebaran Kelas 1):\n{scatter\_matrix\_class1}")

##### **Scatter Matrix for Class 1 (S₁) (Matriks Sebaran Kelas 1):**

##### **[[40519.58567775 -722.76726343]**

##### **[ -722.76726343 9663.46803069]]**

#### 2.a.3. Within-class scatter matrix S\_W

##### S\_W = scatter\_matrix\_class0 + scatter\_matrix\_class1

##### print(f"Within-Class Scatter Matrix (S\_W) (Matriks Sebaran Dalam Kelas Gabungan):\n{S\_W}")

##### 

##### **Within-Class Scatter Matrix (S\_W) (Matriks Sebaran Dalam Kelas Gabungan):**

##### **[[*64266*.58089306 -7194.67635434]**

##### **[-7194.67635434 32554.74075796]]**

### 2.2.b. Between-Class Scatter Matrix (S\_B) (Matriks Sebaran Antar Kelas)

##### # mean\_vector\_class0 dan mean\_vector\_class1 sudah dihitung sebelumnya

##### mean\_diff\_vector = (mean\_vector\_class0 - mean\_vector\_class1).reshape(features\_X.shape[1], 1) # (d x 1)

##### S\_B = mean\_diff\_vector.dot(mean\_diff\_vector.T)

##### # Alternatif definisi S\_B (lebih umum untuk >2 kelas, dan juga valid untuk 2 kelas):

##### # N0 = features\_X\_class0.shape[0]

##### # N1 = features\_X\_class1.shape[0]

##### # overall\_mean\_vec\_reshaped = overall\_mean\_vector.reshape(features\_X.shape[1], 1)

##### # term0 = N0 \* (mean\_vector\_class0.reshape(features\_X.shape[1],1) - overall\_mean\_vec\_reshaped).dot((mean\_vector\_class0.reshape(features\_X.shape[1],1) - overall\_mean\_vec\_reshaped).T)

##### # term1 = N1 \* (mean\_vector\_class1.reshape(features\_X.shape[1],1) - overall\_mean\_vec\_reshaped).dot((mean\_vector\_class1.reshape(features\_X.shape[1],1) - overall\_mean\_vec\_reshaped).T)

##### # S\_B\_alternative = term0 + term1

##### # print(f"Alternative Between-Class Scatter Matrix (S\_B\_alt):\n{S\_B\_alternative}")

##### # Untuk LDA 2 kelas, arah eigenvector akan sama meskipun S\_B berbeda skala. Kita gunakan S\_B yang lebih sederhana.

##### print(f"Mean Difference Vector (m₀-m₁) (Vektor Perbedaan Rata-rata):\n{mean\_diff\_vector.T}") # Transpose untuk tampilan lebih baik

##### print(f"Between-Class Scatter Matrix (S\_B) (Matriks Sebaran Antar Kelas):\n{S\_B}")

##### 

##### **Mean Difference Vector (m₀-m₁) (Vektor Perbedaan Rata-rata):**

##### **[[12.6234413 19.38223669]]**

##### **Between-Class Scatter Matrix (S\_B) (Matriks Sebaran Antar Kelas):**

##### **[[159.35127038 244.6705272 ]**

##### **[244.6705272 375.67109907]]**

## Step 3: Solve the Generalized Eigenvalue Problem (Selesaikan Masalah Eigenvalue Umum)

Tujuan dari LDA adalah untuk menemukan sebuah proyeksi (sumbu diskriminan) yang memaksimalkan separabilitas antar kelas. Ini secara matematis dapat dirumuskan sebagai pencarian vektor w yang memaksimalkan rasio J(w):

J(w) = (wTSBw) / (wTSWw)

Solusi untuk w dapat ditemukan dengan menyelesaikan masalah eigenvalue umum: SBw = λSWw

Jika SW bersifat non-singular (dapat diinvers), ini dapat diubah menjadi masalah eigenvalue standar: SW-1SBw = λw

Di sini:

* w adalah eigenvector.
* λ adalah eigenvalue.

Langkah-langkahnya adalah:

1. **Hitung Invers dari Within-Class Scatter Matrix (SW-1)**:

* Menggunakan fungsi np.linalg.inv() dari NumPy untuk menghitung matriks invers dari SW.
* Penting untuk memastikan SW tidak singular (determinannya tidak nol). Jika singular, ini bisa menandakan adanya kolinearitas sempurna antar fitur atau jumlah sampel yang lebih sedikit dari jumlah fitur, yang mungkin memerlukan teknik regularisasi atau pemilihan fitur.

1. **Hitung Matriks Transformasi (SW-1SB)**:
   * Melakukan perkalian matriks antara SW-1 dan SB.
2. **Hitung Eigenvalue dan Eigenvector**:
   * Menggunakan fungsi np.linalg.eig() dari NumPy pada matriks SW-1SB. Fungsi ini mengembalikan dua array: satu berisi eigenvalue dan satu lagi berisi eigenvector yang bersesuaian (sebagai kolom-kolom matriks).
3. **Pilih Eigenvector yang Relevan**:

* Untuk LDA dengan C kelas, akan ada paling banyak C-1 eigenvector yang tidak nol (atau eigenvalue yang tidak nol jika menggunakan definisi SB yang bergantung pada rata-rata keseluruhan). Dalam kasus klasifikasi biner (C=2), akan ada satu eigenvector yang paling signifikan yang sesuai dengan eigenvalue terbesar. Eigenvector ini, w, mendefinisikan arah sumbu diskriminan linier.
* Eigenvector akan diurutkan berdasarkan eigenvalue-nya dari yang terbesar ke yang terkecil. Kita akan memilih eigenvector yang memiliki eigenvalue terbesar.

Eigenvector w yang dipilih ini adalah vektor koefisien yang akan digunakan untuk memproyeksikan data ke sumbu diskriminan.

##### # Contoh regularisasi (jika diperlukan dan S\_W\_inv tidak dapat dihitung):

##### # epsilon = 1e-6 # Nilai regularisasi kecil

##### # S\_W\_reg = S\_W + np.eye(S\_W.shape[0]) \* epsilon

##### # S\_W\_inv = np.linalg.inv(S\_W\_reg)

##### # print(f"\nInverse of Regularized Within-Class Scatter Matrix (S\_W\_reg⁻¹):\n{S\_W\_inv}")

##### S\_W\_inv = None # Atau tangani error sesuai kebutuhan

##### # 1. Hitung Invers dari Within-Class Scatter Matrix (S\_W⁻¹)

##### # S\_W sudah dihitung pada langkah sebelumnya

##### try:

##### S\_W\_inv = np.linalg.inv(S\_W)

##### print(f"Inverse of Within-Class Scatter Matrix (S\_W⁻¹) (Invers Matriks Sebaran Dalam Kelas):\n{S\_W\_inv}")

##### except np.linalg.LinAlgError:

##### print("Error: Within-Class Scatter Matrix (S\_W) is singular and cannot be inverted.")

##### print("This might be due to perfect multicollinearity or n\_samples < n\_features within classes.")

##### print("Consider adding a small identity matrix (regularization) or re-evaluating features.")

##### if S\_W\_inv is not None:

##### print("\n" + "="\*50 + "\n")

##### *# 2. Hitung Matriks Transformasi (S\_W⁻¹ \* S\_B)*

##### *# S\_B sudah dihitung pada langkah sebelumnya*

##### matrix\_for\_eigen\_decomposition = S\_W\_inv.dot(S\_B)

##### print(f"Matrix for Eigen Decomposition (S\_W⁻¹ \* S\_B) (Matriks untuk Dekomposisi Eigen):\n{matrix\_for\_eigen\_decomposition}")

##### print("\n" + "="\*50 + "\n")

##### # 3. Hitung Eigenvalue dan Eigenvector

##### # eigenvalues adalah array 1D dari eigenvalue

##### # eigenvectors adalah matriks di mana setiap kolom adalah eigenvector yang bersesuaian dengan eigenvalue

##### eigenvalues, eigenvectors = np.linalg.eig(matrix\_for\_eigen\_decomposition)

##### print(f"Eigenvalues (λ) (Nilai Eigen):\n{eigenvalues}")

##### print(f"Eigenvectors (v) (Vektor Eigen) (kolom adalah vektor eigen):\n{eigenvectors}")

##### print("\n" + "="\*50 + "\n")

##### *# 4. Pilih Eigenvector yang Relevan (w)*

##### *# Eigenvector yang bersesuaian dengan eigenvalue terbesar akan menjadi sumbu diskriminan.*

##### *# Untuk kasus 2 kelas, akan ada 1 eigenvalue non-nol yang signifikan (atau C-1 jika C>2).*

##### *# Pasangkan eigenvalue dengan eigenvector-nya*

##### eigen\_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:, i]) for i in range(len(eigenvalues))]

##### *# Urutkan pasangan (eigenvalue, eigenvector) berdasarkan eigenvalue secara menurun*

##### eigen\_pairs.sort(key=lambda k: k[0], reverse=True)

##### print("Eigenvalue-Eigenvector pairs, sorted by eigenvalue (descending):")

##### for i, pair in enumerate(eigen\_pairs):

##### print(f"Pair {i+1}: Eigenvalue = {pair[0]:.4e}, Eigenvector = {pair[1].T}") *# .T untuk tampilan baris*

##### print("\n" + "="\*50 + "\n")

##### *# Pilih eigenvector (w) yang sesuai dengan eigenvalue terbesar.*

##### *# Untuk klasifikasi biner, kita hanya membutuhkan satu eigenvector.*

##### W\_vector = eigen\_pairs[0][1].reshape(features\_X.shape[1], 1) *# Ambil eigenvector pertama (nilai eigen terbesar) dan reshape*

##### print(f"Selected Discriminant Vector (w) (Vektor Diskriminan Terpilih):\n{W\_vector.T}") *# .T untuk tampilan baris*

##### print("\n" + "="\*50 + "\n")

##### print("Step 3: Solving generalized eigenvalue problem completed.")

##### else:

##### print("Step 3 could not be completed due to singular S\_W.")

##### 

##### **Inverse of Within-Class Scatter Matrix (S\_W⁻¹) (Invers Matriks Sebaran Dalam Kelas):**

##### **[[1.59549330e-05 3.52607875e-06]**

##### **[3.52607875e-06 3.14967643e-05]]**

##### **==================================================**

##### **Matrix for Eigen Decomposition (S\_W⁻¹ \* S\_B) (Matriks untuk Dekomposisi Eigen):**

##### **[[0.00340517 0.00522835]**

##### **[0.00826822 0.01269515]]**

##### **==================================================**

##### **Eigenvalues (λ) (Nilai Eigen):**

##### **[0. 0.01610032]**

##### **Eigenvectors (v) (Vektor Eigen) (kolom adalah vektor eigen):**

##### **[[-0.83794965 -0.38080787]**

##### **[ 0.54574755 -0.92465419]]**

##### **==================================================**

##### **Eigenvalue-Eigenvector pairs, sorted by eigenvalue (descending):**

##### **Pair 1: Eigenvalue = 1.6100e-02, Eigenvector = [-0.38080787 -0.92465419]**

##### **Pair 2: Eigenvalue = 0.0000e+00, Eigenvector = [-0.83794965 0.54574755]**

##### **==================================================**

##### **Selected Discriminant Vector (w) (Vektor Diskriminan Terpilih):**

##### **[[-0.38080787 -0.92465419]]**

## Step 4: Project Data onto the Discriminant Axis (LD Scores) (Proyeksikan Data ke Sumbu Diskriminan (Skor LD))

Setelah vektor diskriminan W\_vector (eigenvector yang sesuai dengan eigenvalue terbesar dari SW-1SB) ditentukan, langkah berikutnya adalah memproyeksikan data fitur asli ke sumbu baru yang didefinisikan oleh W\_vector. Proyeksi ini mengubah data asli yang mungkin berdimensi tinggi (dalam kasus ini, 2D) menjadi data berdimensi lebih rendah (dalam kasus klasifikasi biner, menjadi 1D) yang memaksimalkan pemisahan antar kelas.

1. **Rumus Proyeksi**:

* Untuk setiap sampel data x (vektor fitur d x 1), skor diskriminan linier (LD Score), z, dihitung sebagai: z = W\_vectorTx
* Ini adalah perkalian dot antara transpose dari vektor diskriminan (1 x d) dan vektor fitur sampel (d x 1), yang menghasilkan skalar (nilai 1D).

1. **Penerapan pada Seluruh Dataset**:

* Operasi proyeksi ini diterapkan pada seluruh dataset fitur (features\_X). Jika features\_X adalah matriks N x d (N sampel, d fitur) dan W\_vector adalah d x 1, maka features\_X.dot(W\_vector) akan menghasilkan matriks N x 1, yang berisi skor LD untuk setiap sampel.
* Skor LD ini dapat disimpan dalam kolom baru di DataFrame asli atau sebagai array NumPy terpisah.
* Skor LD ini akan digunakan untuk menentukan batas keputusan (threshold) dan untuk melakukan klasifikasi.

Setelah langkah ini, setiap observasi data akan memiliki satu skor numerik yang merepresentasikan posisinya pada sumbu diskriminan linier. Idealnya, skor LD untuk kelas yang berbeda akan terkelompok secara terpisah pada sumbu ini.

##### # Untuk kemudahan analisis dan visualisasi, tambahkan ld\_scores ke DataFrame df

##### df['LD\_Score\_Manual'] = ld\_scores.flatten() # flatten() untuk mengubah array (N,1) menjadi (N,)

##### print(f"Shape of LD Scores array (Skor LD): {ld\_scores.shape}")

##### print(f"First 10 LD Scores (Skor LD pertama):\n{ld\_scores[:10].flatten()}") # Tampilkan 10 skor pertama

##### print("\n" + "="\*50 + "\n")

##### print("DataFrame with LD\_Score\_Manual added:")

##### print(df.head())

##### print("\n" + "="\*50 + "\n")

##### # Visualisasi distribusi LD Scores berdasarkan kelas (opsional, tapi sangat membantu)

##### # Ini mirip dengan yang dilakukan pada evaluasi model scikit-learn,

##### # tapi kali ini menggunakan skor yang dihitung manual.

##### import matplotlib.pyplot as plt

##### import seaborn as sns

##### plt.figure(figsize=(10, 6))

##### sns.histplot(data=df, x='LD\_Score\_Manual', hue=target\_column\_name, kde=True, palette='viridis', stat="density", common\_norm=False)

##### plt.title('Distribution of Manually Calculated LD Scores by Cluster (Distribusi Skor LD Manual Berdasarkan Cluster)')

##### plt.xlabel('LD Score (Manual) (Skor LD (Manual))')

##### plt.ylabel('Density (Kepadatan)')

##### plt.legend(title=target\_column\_name, labels=[f'Cluster {val}' for val in sorted(df[target\_column\_name].unique())])

##### plt.show()

##### else:

##### print("W\_vector is not defined. Please ensure Step 3 was completed successfully.")

##### **Shape of LD Scores array (Skor LD): (600, 1)**

##### **First 10 LD Scores (Skor LD pertama):**

##### **[-73.65087604 -53.79759926 -53.96063771 -44.55105739 -45.745903**

##### **-55.91940804 -43.0814024 -54.23313811 -75.66322286 -86.48888169]**

##### **==================================================**

##### **DataFrame with LD\_Score\_Manual added:**

##### **BloodPressure Age Cluster LD\_Score\_Manual**

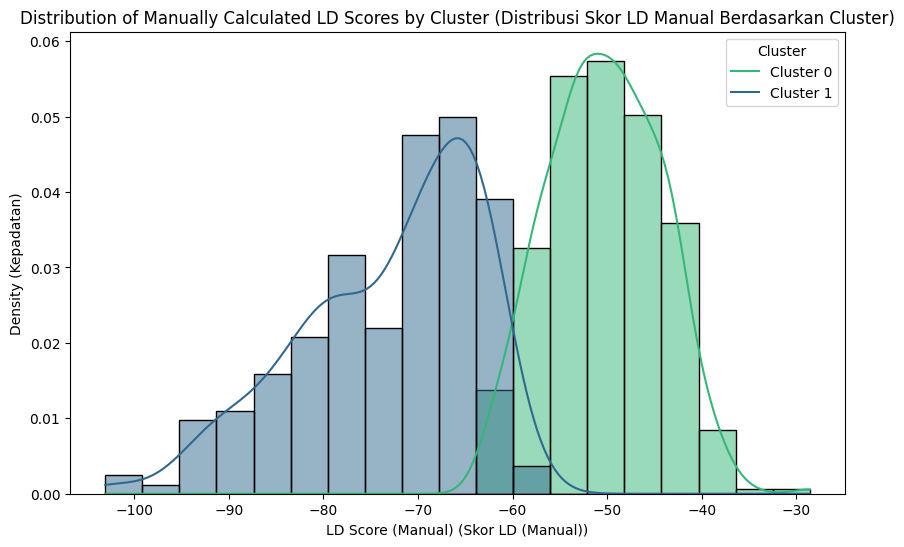
##### **0 72 50 0 -73.650876**

##### **1 66 31 1 -53.797599**

##### **2 64 32 1 -53.960638**

##### **3 66 21 1 -44.551057**

##### **4 40 33 1 -45.745903**



## Step 5: Determine a Threshold for Classification (Tentukan Ambang Batas untuk Klasifikasi)

Setelah mendapatkan skor diskriminan linier (LD Scores) untuk setiap observasi, sebuah nilai ambang batas (threshold atau cut-off value) perlu ditentukan. Threshold ini akan digunakan untuk mengklasifikasikan observasi ke dalam salah satu dari dua kelas berdasarkan skor LD-nya.

1. **Menghitung Rata-rata Skor LD (Centroid) per Kelas pada Sumbu Diskriminan**:
   * Pertama, skor LD yang telah dihitung pada langkah sebelumnya (df['LD\_Score\_Manual']) dipisahkan berdasarkan kelas aslinya (Cluster 0 dan Cluster 1).
   * Kemudian, rata-rata dari skor LD dihitung untuk setiap kelas. Rata-rata ini adalah proyeksi dari vektor rata-rata kelas (m0 dan m1) ke sumbu diskriminan W\_vector.
     + ld\_mean\_class0: Rata-rata skor LD untuk Kelas 0.
     + ld\_mean\_class1: Rata-rata skor LD untuk Kelas 1.
   * Titik-titik ini dapat dianggap sebagai pusat (centroid) dari masing-masing kelas pada sumbu diskriminan 1D.
2. **Menentukan Threshold**:

* Cara paling sederhana dan umum untuk menentukan threshold (jika tidak ada informasi prior probability yang berbeda atau biaya misklasifikasi yang tidak sama) adalah dengan mengambil titik tengah antara dua centroid kelas pada sumbu diskriminan.
* Threshold = (ld\_mean\_class0 + ld\_mean\_class1) / 2
* Observasi dengan skor LD di satu sisi threshold akan diklasifikasikan sebagai Kelas 0, dan yang di sisi lain sebagai Kelas 1. Keputusan sisi mana yang menjadi Kelas 0 atau Kelas 1 akan bergantung pada nilai relatif ld\_mean\_class0 dan ld\_mean\_class1 serta bagaimana W\_vector diorientasikan (misalnya, apakah nilai LD yang lebih tinggi cenderung ke Kelas 1 atau Kelas 0).

Threshold ini adalah nilai kunci yang akan digunakan pada langkah klasifikasi berikutnya.

##### # Pastikan kolom 'LD\_Score\_Manual' ada di DataFrame df dari langkah sebelumnya

##### if 'LD\_Score\_Manual' in df.columns:

##### # 1. Menghitung Rata-rata Skor LD (Centroid) per Kelas

##### # df\_class0 dan df\_class1 sudah dibuat pada langkah persiapan data,

##### # kita bisa menggunakannya atau memfilter df yang sudah memiliki LD\_Score\_Manual

##### # Ambil skor LD untuk masing-masing kelas dari DataFrame 'df' yang sudah diupdate

##### ld\_scores\_class0 = df[df[target\_column\_name] == 0]['LD\_Score\_Manual']

##### ld\_scores\_class1 = df[df[target\_column\_name] == 1]['LD\_Score\_Manual']

##### # Hitung rata-rata skor LD untuk setiap kelas

##### ld\_mean\_class0 = np.mean(ld\_scores\_class0)

##### ld\_mean\_class1 = np.mean(ld\_scores\_class1)

##### print(f"Mean LD Score for Class 0 (Centroid Kelas 0 pada Sumbu LD): {ld\_mean\_class0:.4f}")

##### print(f"Mean LD Score for Class 1 (Centroid Kelas 1 pada Sumbu LD): {ld\_mean\_class1:.4f}")

##### print("\n" + "="\*50 + "\n")

##### # 2. Menentukan Threshold

##### # Threshold adalah titik tengah antara dua centroid kelas

##### classification\_threshold = (ld\_mean\_class0 + ld\_mean\_class1) / 2

##### print(f"Calculated Classification Threshold (Ambang Batas Klasifikasi Terhitung): {classification\_threshold:.4f}")

##### print("\n" + "="\*50 + "\n")

##### # Menampilkan kembali visualisasi distribusi LD Scores dengan threshold

##### plt.figure(figsize=(10, 6))

##### sns.histplot(data=df, x='LD\_Score\_Manual', hue=target\_column\_name, kde=True, palette='viridis', stat="density", common\_norm=False)

##### plt.title('Distribution of LD Scores with Classification Threshold (Distribusi Skor LD dengan Ambang Batas)')

##### plt.xlabel('LD Score (Manual) (Skor LD (Manual))')

##### plt.ylabel('Density (Kepadatan)')

##### plt.axvline(classification\_threshold, color='red', linestyle='--', lw=2, label=f'Threshold ({classification\_threshold:.2f})')

##### plt.axvline(ld\_mean\_class0, color='blue', linestyle=':', lw=2, label=f'Mean C0 ({ld\_mean\_class0:.2f})')

##### plt.axvline(ld\_mean\_class1, color='orange', linestyle=':', lw=2, label=f'Mean C1 ({ld\_mean\_class1:.2f})')

##### plt.legend(title=target\_column\_name, labels=[f'Cluster {val}' for val in sorted(df[target\_column\_name].unique())] + ['Threshold', 'Mean C0', 'Mean C1'])

##### plt.show()

##### else:

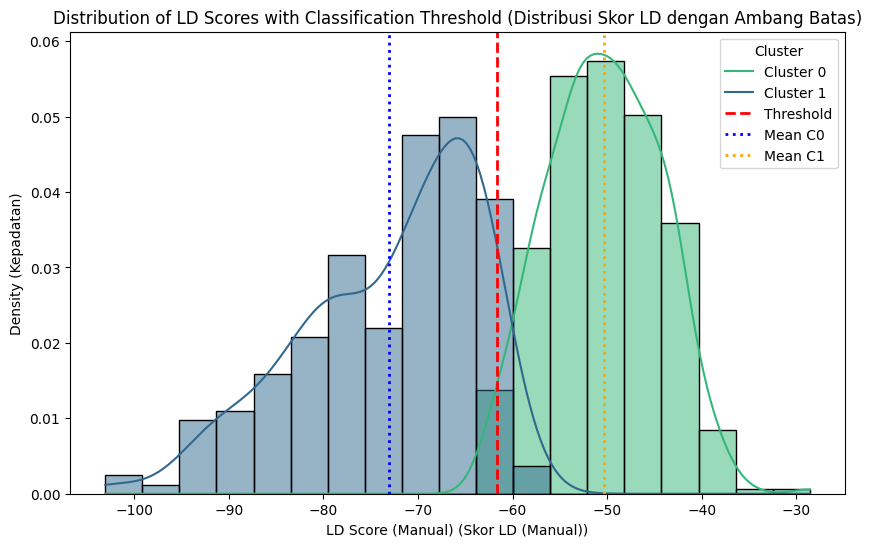
##### print("'LD\_Score\_Manual' column not found in DataFrame. Please ensure Step 4 was completed successfully.")

##### **Mean LD Score for Class 0 (Centroid Kelas 0 pada Sumbu LD): -73.0167**

##### **Mean LD Score for Class 1 (Centroid Kelas 1 pada Sumbu LD): -50.2877**

##### **==================================================**

##### **Calculated Classification Threshold (Ambang Batas Klasifikasi Terhitung): -61.6522**



## Step 6: Formulate the Discriminant Function and Classify Data (Rumuskan Fungsi Diskriminan dan Klasifikasikan Data)

Dengan vektor diskriminan W\_vector dan threshold klasifikasi yang telah ditentukan, kita sekarang dapat merumuskan fungsi diskriminan dan menggunakannya untuk mengklasifikasikan data.

1. **Merumuskan Fungsi Diskriminan Linier (Bentuk Skor LD)**:

* Fungsi diskriminan yang menghasilkan skor LD (z) untuk suatu sampel x = [BloodPressure, Age]T diberikan oleh: z = W\_vectorTx = w₁\*BloodPressure + w₂\*Age Di mana w₁ dan w₂ adalah komponen dari W\_vector.
* Penting diingat bahwa ini adalah skor mentah. Klasifikasi terjadi dengan membandingkan skor ini dengan classification\_threshold.

1. **Aturan Klasifikasi**:
   * Sebuah sampel akan diklasifikasikan ke salah satu kelas berdasarkan perbandingan skor LD-nya (df['LD\_Score\_Manual']) dengan classification\_threshold.
   * Perlu ditentukan kelas mana yang diasosiasikan dengan skor LD yang lebih tinggi dan mana yang lebih rendah. Ini tergantung pada orientasi W\_vector dan nilai ld\_mean\_class0 serta ld\_mean\_class1.
   * Misalnya, jika ld\_mean\_class1 > ld\_mean\_class0:
     + Jika LD\_Score > classification\_threshold, maka prediksi Kelas 1.
     + Jika LD\_Score <= classification\_threshold, maka prediksi Kelas 0.
   * Sebaliknya, jika ld\_mean\_class0 > ld\_mean\_class1:
     + Jika LD\_Score > classification\_threshold, maka prediksi Kelas 0.
     + Jika LD\_Score <= classification\_threshold, maka prediksi Kelas 1.
   * Aturan ini akan diterapkan untuk membuat kolom prediksi baru di DataFrame.
2. **Menghitung Akurasi Klasifikasi Manual (Opsional)**:

* Setelah prediksi dibuat untuk seluruh dataset berdasarkan aturan manual, akurasi klasifikasi dapat dihitung dengan membandingkan prediksi manual dengan label kelas asli (target\_y).
* Akurasi = (Jumlah Prediksi Benar) / (Total Jumlah Sampel)

Merumuskan fungsi secara eksplisit membantu pemahaman, dan klasifikasi ulang memberikan ukuran seberapa baik metode manual ini bekerja pada data yang sama yang digunakan untuk membangunnya (ini adalah akurasi pada "data pelatihan" untuk metode manual ini, bukan evaluasi generalisasi yang sebenarnya).

##### # Pastikan variabel-variabel dari langkah sebelumnya tersedia:

##### # W\_vector, classification\_threshold, df (dengan 'LD\_Score\_Manual'),

##### # target\_column\_name, ld\_mean\_class0, ld\_mean\_class1

##### if 'W\_vector' in globals() and W\_vector is not None and \

##### 'classification\_threshold' in globals() and \

##### 'df' in globals() and 'LD\_Score\_Manual' in df.columns and \

##### 'ld\_mean\_class0' in globals() and 'ld\_mean\_class1' in globals():

##### w1 = W\_vector[0, 0] # Koefisien untuk BloodPressure

##### w2 = W\_vector[1, 0] # Koefisien untuk Age

##### # 1. Merumuskan Fungsi Diskriminan Linier (Bentuk Skor LD)

##### print("Linear Discriminant Function (LD Score Form) (Fungsi Diskriminan Linier (Bentuk Skor LD)):")

##### print(f"LD\_Score = ({w1:.4f} \* BloodPressure) + ({w2:.4f} \* Age)")

##### print("Catatan: 'BloodPressure' dan 'Age' adalah nilai fitur asli, bukan yang diskalakan, karena W\_vector berasal dari S\_W dan S\_B data asli.")

##### print("\n" + "="\*50 + "\n")

##### # 2. Aturan Klasifikasi dan Membuat Prediksi Manual

##### # Tentukan kelas mana yang memiliki rata-rata LD Score lebih tinggi

##### # Ini akan menjadi kelas "positif" jika skor LD di atas threshold

##### predicted\_class\_manual = []

##### # Menentukan kelas mana yang memiliki nilai LD score lebih tinggi secara rata-rata

##### # Ini membantu menentukan logika > atau <= threshold

##### class\_for\_higher\_score = 1 if ld\_mean\_class1 > ld\_mean\_class0 else 0

##### class\_for\_lower\_score = 0 if ld\_mean\_class1 > ld\_mean\_class0 else 1

##### print(f"Mean LD Score Class 0: {ld\_mean\_class0:.4f}")

##### print(f"Mean LD Score Class 1: {ld\_mean\_class1:.4f}")

##### print(f"Classification Threshold: {classification\_threshold:.4f}")

##### print(f"Class associated with scores > threshold will be: Cluster {class\_for\_higher\_score}")

##### print(f"Class associated with scores <= threshold will be: Cluster {class\_for\_lower\_score}")

##### print("\n" + "="\*50 + "\n")

##### for score in df['LD\_Score\_Manual']:

##### if score > classification\_threshold:

##### predicted\_class\_manual.append(class\_for\_higher\_score)

##### else:

##### predicted\_class\_manual.append(class\_for\_lower\_score)

##### df['Predicted\_Cluster\_Manual'] = predicted\_class\_manual

##### print("DataFrame with Manual Predictions (DataFrame dengan Prediksi Manual):")

##### print(df[[target\_column\_name, 'LD\_Score\_Manual', 'Predicted\_Cluster\_Manual']].head(10))

##### print("\n" + "="\*50 + "\n")

##### *# 3. Menghitung Akurasi Klasifikasi Manual*

##### actual\_labels = df[target\_column\_name].values

##### manual\_predictions = df['Predicted\_Cluster\_Manual'].values

correct\_predictions = np.sum(actual\_labels == manual\_predictions)

##### total\_samples = len(actual\_labels)

##### manual\_accuracy = correct\_predictions / total\_samples

##### print(f"Accuracy of Manual LDA Classification (Akurasi Klasifikasi LDA Manual): {manual\_accuracy:.4f}")

##### print(f"Accuracy in percentage: {manual\_accuracy\*100:.2f}%")

##### print(f"Number of correct predictions: {correct\_predictions} out of {total\_samples} samples.")

##### print("\n" + "="\*50 + "\n")

##### *# (Opsional) Menampilkan Confusion Matrix untuk prediksi manual*

##### from sklearn.metrics import confusion\_matrix as sk\_confusion\_matrix

##### import seaborn as sns

##### import matplotlib.pyplot as plt

##### conf\_matrix\_manual = sk\_confusion\_matrix(actual\_labels, manual\_predictions)

##### print("Manual Classification Confusion Matrix (Matriks Konfusi Klasifikasi Manual):")

##### print(conf\_matrix\_manual)

##### plt.figure(figsize=(8, 6))

##### sns.heatmap(conf\_matrix\_manual, annot=True, fmt='d', cmap='Greens',

##### xticklabels=sorted(df[target\_column\_name].unique()),

##### yticklabels=sorted(df[target\_column\_name].unique()))

##### plt.xlabel('Predicted Label (Manual)')

##### plt.ylabel('True Label')

##### plt.title('Confusion Matrix for Manual LDA Classification (Matriks Konfusi LDA Manual)')

##### plt.show()

##### print("\n" + "="\*50 + "\n")

##### *# Merumuskan Fungsi Diskriminan dalam bentuk D = w1\*X1 + w2\*X2 - C (threshold diserap jadi konstanta)*

##### *# D > 0 -> class\_for\_higher\_score ; D <= 0 -> class\_for\_lower\_score*

##### *# Ini ekuivalen dengan LD\_Score - threshold > 0*

##### constant\_c = classification\_threshold

##### print("Alternative Discriminant Function (D) (Fungsi Diskriminan Alternatif D):")

##### print(f"D = ({w1:.4f} \* BloodPressure) + ({w2:.4f} \* Age) - ({constant\_c:.4f})")

##### print(f"Jika D > 0, prediksi Cluster {class\_for\_higher\_score}.")

##### print(f"Jika D <= 0, prediksi Cluster {class\_for\_lower\_score}.")

##### print("\n" + "="\*50 + "\n")

##### print("Step 6: Formulation of discriminant function and classification completed.")

##### else:

##### print("Required variables (W\_vector, classification\_threshold, df with 'LD\_Score\_Manual', ld\_mean\_class0, ld\_mean\_class1) not found.")

##### print("Please ensure Steps 1-5 were completed successfully.")

**Linear Discriminant Function (LD Score Form) (Fungsi Diskriminan Linier (Bentuk Skor LD)):**

**LD\_Score = (-0.3808 \* BloodPressure) + (-0.9247 \* Age)**

**Catatan: 'BloodPressure' dan 'Age' adalah nilai fitur asli, bukan yang diskalakan, karena W\_vector berasal dari S\_W dan S\_B data asli.**

**==================================================**

**Mean LD Score Class 0: -73.0167**

**Mean LD Score Class 1: -50.2877**

**Classification Threshold: -61.6522**

**Class associated with scores > threshold will be: Cluster 1**

**Class associated with scores <= threshold will be: Cluster 0**

**==================================================**

**DataFrame with Manual Predictions (DataFrame dengan Prediksi Manual):**

**Cluster LD\_Score\_Manual Predicted\_Cluster\_Manual**

**0 0 -73.650876 0**

**1 1 -53.797599 1**

**2 1 -53.960638 1**

**3 1 -44.551057 1**

**4 1 -45.745903 1**

**5 1 -55.919408 1**

**6 1 -43.081402 1**

**7 1 -54.233138 1**

**8 0 -75.663223 0**

**9 0 -86.488882 0**

**Accuracy of Manual LDA Classification (Akurasi Klasifikasi LDA Manual): 0.9733**

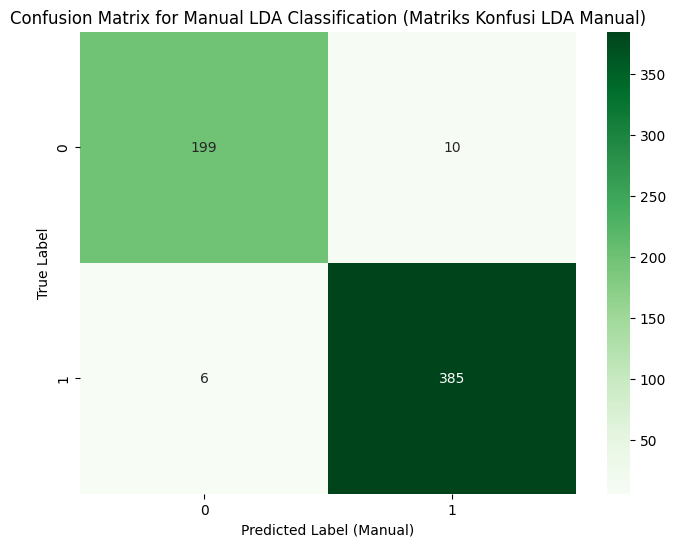
**Accuracy in percentage: 97.33%**

**Number of correct predictions: 584 out of 600 samples.**

**Manual Classification Confusion Matrix (Matriks Konfusi Klasifikasi Manual):**

**[[199 10]**

**[ 6 385]]**



**Alternative Discriminant Function (D) (Fungsi Diskriminan Alternatif D):**

**D = (-0.3808 \* BloodPressure) + (-0.9247 \* Age) - (-61.6522)**

**Jika D > 0, prediksi Cluster 1.**

**Jika D <= 0, prediksi Cluster 0.**

## Step 7: Reviewing the Formulated Manual Discriminant Function (Meninjau Rumus Fungsi Diskriminan Manual yang Telah Dirumuskan)

Sebelum melanjutkan untuk membuat prediksi pada data baru, penting untuk meninjau kembali bentuk eksplisit dari fungsi diskriminan linier yang telah berhasil dibangun melalui perhitungan manual. Fungsi ini akan menjadi dasar untuk semua klasifikasi manual selanjutnya.

Ada dua cara umum untuk menyajikan fungsi diskriminan:

1. **Bentuk Skor LD dengan Threshold Terpisah**:
   * Hitung Skor Diskriminan Linier (LD Score): LD\_Score = w₁ \* BloodPressure + w₂ \* Age
   * Kemudian, bandingkan LD\_Score dengan classification\_threshold:
     + Jika LD\_Score > classification\_threshold, prediksi class\_for\_higher\_score.
     + Jika LD\_Score <= classification\_threshold, prediksi class\_for\_lower\_score.
2. **Bentuk Fungsi Diskriminan D (Threshold Terintegrasi)**:
   * Definisikan fungsi D sebagai: D = w₁ \* BloodPressure + w₂ \* Age - classification\_threshold atau dapat ditulis juga sebagai: D = LD\_Score - classification\_threshold
   * Kemudian, aturan klasifikasi menjadi:
     + Jika D > 0, prediksi class\_for\_higher\_score.
     + Jika D <= 0, prediksi class\_for\_lower\_score.

Kedua bentuk ini ekuivalen dan menghasilkan keputusan klasifikasi yang sama. w₁ dan w₂ adalah koefisien (komponen dari W\_vector) yang ditemukan pada Step 3, dan classification\_threshold ditentukan pada Step 5. Variabel class\_for\_higher\_score dan class\_for\_lower\_score ditentukan pada Step 6.

Mengingat kembali rumus ini memastikan bahwa kita menggunakan dasar yang benar untuk prediksi pada data baru.

##### # Pastikan variabel dari langkah-langkah sebelumnya tersedia:

##### # W\_vector, classification\_threshold, class\_for\_higher\_score, class\_for\_lower\_score

##### if 'W\_vector' in globals() and W\_vector is not None and \

##### 'classification\_threshold' in globals() and \

##### 'class\_for\_higher\_score' in globals() and 'class\_for\_lower\_score' in globals():

##### w1\_manual\_review = W\_vector[0, 0]

##### w2\_manual\_review = W\_vector[1, 0]

##### threshold\_review = classification\_threshold

##### class\_high\_review = class\_for\_higher\_score

##### class\_low\_review = class\_for\_lower\_score

##### print("--- Review of Manual Discriminant Function ---")

##### print("(Tinjauan Rumus Fungsi Diskriminan Manual)\n")

print("Bentuk 1: Skor LD dengan Threshold Terpisah")

print("-------------------------------------------")

print(f"1. Hitung Skor LD (LD\_Score):")

print(f" LD\_Score = ({w1\_manual\_review:.4f} \* BloodPressure) + ({w2\_manual\_review:.4f} \* Age)")

print(f"\n2. Bandingkan LD\_Score dengan Classification Threshold = {threshold\_review:.4f}:")

##### print(f" - Jika LD\_Score > {threshold\_review:.4f}, maka prediksi Cluster {class\_high\_review}")

##### print(f" - Jika LD\_Score <= {threshold\_review:.4f}, maka prediksi Cluster {class\_low\_review}")

##### print("\n" + "="\*60 + "\n")

##### print("Bentuk 2: Fungsi Diskriminan D (Threshold Terintegrasi)")

##### print("-------------------------------------------------------")

##### print(f"1. Hitung Fungsi Diskriminan D:")

##### *# D = LD\_Score - threshold*

##### print(f" D = ({w1\_manual\_review:.4f} \* BloodPressure) + ({w2\_manual\_review:.4f} \* Age) - ({threshold\_review:.4f})")

##### print(f"\n2. Bandingkan D dengan 0:")

##### print(f" - Jika D > 0, maka prediksi Cluster {class\_high\_review}")

##### print(f" - Jika D <= 0, maka prediksi Cluster {class\_low\_review}")

##### print("\n" + "="\*60 + "\n")

##### print("Step 6.1: Review of manual discriminant function completed.")

##### else:

##### print("Required variables (W\_vector, classification\_threshold, etc.) not found.")

##### print("Please ensure previous steps (Manual LDA up to Step 6) were completed successfully.")

##### **--- Review of Manual Discriminant Function ---**

##### **(Tinjauan Rumus Fungsi Diskriminan Manual)**

##### **Bentuk 1: Skor LD dengan Threshold Terpisah**

##### **-------------------------------------------**

##### **1. Hitung Skor LD (LD\_Score):**

##### **LD\_Score = (-0.3808 \* BloodPressure) + (-0.9247 \* Age)**

##### **2. Bandingkan LD\_Score dengan Classification Threshold = -61.6522:**

##### **- Jika LD\_Score > -61.6522, maka prediksi Cluster 1**

##### **- Jika LD\_Score <= -61.6522, maka prediksi Cluster 0**

##### **============================================================**

##### **Bentuk 2: Fungsi Diskriminan D (Threshold Terintegrasi)**

##### **-------------------------------------------------------**

##### **1. Hitung Fungsi Diskriminan D:**

##### **D = (-0.3808 \* BloodPressure) + (-0.9247 \* Age) - (-61.6522)**

##### **2. Bandingkan D dengan 0:**

##### **- Jika D > 0, maka prediksi Cluster 1**

##### **- Jika D <= 0, maka prediksi Cluster 0**

## Step 8: Making Predictions on New Data (Manual) (Membuat Prediksi pada Data Baru (Manual))

Setelah fungsi diskriminan manual dan threshold klasifikasi ditentukan, model ini dapat digunakan untuk membuat prediksi pada data baru (unseen data).

1. **Menyiapkan Data Baru**:
   * Data baru harus memiliki struktur kolom fitur yang sama dengan data yang digunakan untuk membangun fungsi diskriminan (yaitu, BloodPressure dan Age).
   * Data baru akan dibuat sebagai DataFrame Pandas.
2. **Menghitung Skor LD untuk Data Baru**:
   * Untuk setiap sampel dalam data baru, skor LD (z\_new) dihitung menggunakan fungsi diskriminan yang telah dirumuskan pada Langkah 6: z\_new = w₁ \* BloodPressure\_new + w₂ \* Age\_new di mana w₁ dan w₂ adalah komponen dari W\_vector.
3. **Menerapkan Aturan Klasifikasi**:
   * Skor LD (z\_new) dari setiap sampel data baru dibandingkan dengan classification\_threshold yang telah ditentukan pada Langkah 5.
   * Aturan klasifikasi yang sama (berdasarkan class\_for\_higher\_score dan class\_for\_lower\_score yang ditentukan pada Langkah 6) diterapkan:
     + Jika z\_new > classification\_threshold, maka prediksi class\_for\_higher\_score.
     + Jika z\_new <= classification\_threshold, maka prediksi class\_for\_lower\_score.
4. **Menampilkan Prediksi**:
   * Prediksi kelas untuk data baru akan ditampilkan.

Langkah ini menunjukkan bagaimana model diskriminan manual dapat digunakan untuk tujuan prediktif pada observasi baru.

##### # Pastikan variabel dari langkah sebelumnya tersedia:

##### # W\_vector (untuk w1, w2), classification\_threshold,

##### # class\_for\_higher\_score, class\_for\_lower\_score

##### if 'W\_vector' in globals() and W\_vector is not None and \

##### 'classification\_threshold' in globals() and \

##### 'class\_for\_higher\_score' in globals() and 'class\_for\_lower\_score' in globals():

##### # 1. Menyiapkan Data Baru

##### new\_data\_to\_predict\_dict = {

##### 'BloodPressure': [76, 92, 58, 74, 62, 76, 70, 72, 60, 70],

##### 'Age': [26, 66, 22, 43, 33, 63, 27, 30, 47, 23]

##### }

##### df\_new\_data = pd.DataFrame(new\_data\_to\_predict\_dict)

##### print("New data for manual prediction (Data baru untuk prediksi manual):")

##### print(df\_new\_data)

##### print("\n" + "="\*50 + "\n")

##### # Ekstrak koefisien w1 dan w2 dari W\_vector

##### w1\_manual = W\_vector[0, 0]

##### w2\_manual = W\_vector[1, 0]

##### # 2. Menghitung Skor LD untuk Data Baru

##### ld\_scores\_new\_data = []

##### for index, row in df\_new\_data.iterrows():

##### bp\_new = row['BloodPressure']

##### age\_new = row['Age']

##### ld\_score\_sample = (w1\_manual \* bp\_new) + (w2\_manual \* age\_new)

##### ld\_scores\_new\_data.append(ld\_score\_sample)

##### df\_new\_data['LD\_Score\_Manual'] = ld\_scores\_new\_data

##### print("New data with calculated LD Scores (Data baru dengan Skor LD terhitung):")

##### print(df\_new\_data[['BloodPressure', 'Age', 'LD\_Score\_Manual']])

##### print("\n" + "="\*50 + "\n")

##### # 3. Menerapkan Aturan Klasifikasi

##### predicted\_classes\_new\_data = []

##### for score in df\_new\_data['LD\_Score\_Manual']:

##### if score > classification\_threshold:

##### predicted\_classes\_new\_data.append(class\_for\_higher\_score)

##### else:

##### predicted\_classes\_new\_data.append(class\_for\_lower\_score)

##### df\_new\_data['Predicted\_Cluster\_Manual'] = predicted\_classes\_new\_data

*# 4. Menampilkan Prediksi*

##### print("Predictions for new data using manual LDA (Prediksi untuk data baru menggunakan LDA manual):")

##### print(df\_new\_data[['BloodPressure', 'Age', 'LD\_Score\_Manual', 'Predicted\_Cluster\_Manual']])

##### print("\n" + "="\*50 + "\n")

##### print("Detailed predictions for each new sample (Prediksi detail untuk setiap sampel baru):")

##### for index, row in df\_new\_data.iterrows():

##### print(f"Sample {index + 1} (BP: {row['BloodPressure']}, Age: {row['Age']}):")

##### print(f" - Calculated LD Score (Manual): {row['LD\_Score\_Manual']:.4f}")

##### print(f" - Predicted Cluster (Manual): {row['Predicted\_Cluster\_Manual']}")

##### *# Untuk perbandingan, kita bisa juga mencetak nilai D = LD\_Score - threshold*

##### d\_value = row['LD\_Score\_Manual'] - classification\_threshold

##### print(f" - Value of D (LD\_Score - Threshold): {d\_value:.4f} (Positive suggests Cluster {class\_for\_higher\_score}, Negative/Zero suggests Cluster {class\_for\_lower\_score})")

##### print("-" \* 30)

##### print("\nStep 7: Manual prediction on new data completed.")

##### else:

##### print("Required variables for manual prediction are not defined.")

##### print("Please ensure Steps 1-6 (Manual LDA) were completed successfully.")

##### 

**==================================================**

##### **New data with calculated LD Scores (Data baru dengan Skor LD terhitung):**

##### **BloodPressure Age LD\_Score\_Manual**

##### **0 76 26 -52.982407**

##### **1 92 66 -96.061500**

##### **2 58 22 -42.429249**

##### **3 74 43 -67.939912**

##### **4 62 33 -54.123676**

##### **5 76 63 -87.194612**

##### **6 70 27 -51.622214**

##### **7 72 30 -55.157792**

##### **8 60 47 -66.307219**

##### **9 70 23 -47.923597**

##### **==================================================**

##### **Predictions for new data using manual LDA (Prediksi untuk data baru menggunakan LDA manual):**

##### **BloodPressure Age LD\_Score\_Manual Predicted\_Cluster\_Manual**

##### **0 76 26 -52.982407 1**

##### **1 92 66 -96.061500 0**

##### **2 58 22 -42.429249 1**

##### **3 74 43 -67.939912 0**

##### **4 62 33 -54.123676 1**

##### **5 76 63 -87.194612 0**

##### **6 70 27 -51.622214 1**

##### **7 72 30 -55.157792 1**

##### **8 60 47 -66.307219 0**

##### **9 70 23 -47.923597 1**

## 2.3. Kesimpulan

Analisis Diskriminan Linier (LDA) secara manual telah berhasil diimplementasikan menggunakan Python untuk mengklasifikasikan data berdasarkan variabel BloodPressure dan Age ke dalam dua cluster. Perhitungan ini melibatkan serangkaian langkah matematis untuk menurunkan fungsi diskriminan yang optimal dalam memisahkan kedua kelompok.

Dari perhitungan manual, diperoleh fungsi diskriminan linier dalam bentuk Skor LD (Linear Discriminant Score) sebagai berikut:  
**LD\_Score = (-0.3808 \* BloodPressure) + (-0.9247 \* Age)**

Koefisien negatif untuk kedua variabel, BloodPressure dan Age, menunjukkan bahwa peningkatan nilai pada salah satu atau kedua variabel ini cenderung menghasilkan skor LD yang lebih rendah. Skor LD ini kemudian dibandingkan dengan *classification threshold* (ambang batas klasifikasi) yang telah dihitung.

Berdasarkan proyeksi data ke sumbu diskriminan, rata-rata Skor LD untuk Cluster 0 (Mean LD Score Class 0) adalah **-73.0167**, dan untuk Cluster 1 (Mean LD Score Class 1) adalah **-50.2877**. Dengan kedua nilai centroid ini, *classification threshold* ditetapkan pada **-61.6522**. Aturan klasifikasinya adalah:

* Jika LD\_Score > -61.6522, maka individu diprediksi sebagai Cluster 1.
* Jika LD\_Score <= -61.6522, maka individu diprediksi sebagai Cluster 0.

Fungsi diskriminan ini juga dapat diekspresikan dalam bentuk alternatif D, di mana threshold telah diintegrasikan:  
**D = (-0.3808 \* BloodPressure) + (-0.9247 \* Age) - (-61.6522)**  
Dengan aturan: Jika D > 0, prediksi Cluster 1; jika D <= 0, prediksi Cluster 0.

Evaluasi terhadap model diskriminan yang dibangun secara manual ini pada 600 sampel data menunjukkan tingkat akurasi yang tinggi. Model berhasil mengklasifikasikan **584 dari 600 sampel dengan benar**, menghasilkan **akurasi keseluruhan sebesar 97.33%**. Performa ini menunjukkan bahwa pendekatan manual, meskipun lebih kompleks dalam perhitungannya, mampu menghasilkan fungsi diskriminan yang efektif untuk membedakan antar cluster pada dataset yang digunakan.

Pengujian fungsi diskriminan manual pada 10 data baru juga berhasil menghasilkan skor LD dan prediksi cluster untuk setiap observasi baru, menunjukkan aplikabilitas fungsi yang telah diturunkan.

Secara keseluruhan, implementasi manual Analisis Diskriminan Linier menggunakan Python telah berhasil menghasilkan sebuah model klasifikasi yang efektif dengan performa akurasi yang sangat baik, serta memberikan pemahaman mendalam mengenai mekanisme matematis di balik LDA.

# 3. Analisis Diskriminan Menggunakan Library Scikit-Learn (Python)

Penerapan LDA dengan memanfaatkan fungsi LinearDiscriminantAnalysis dari Scikit-Learn.

Link Notebook Pengerjaan: <https://github.com/LatiefDataVisionary/multivariate-analysis-college-task/blob/main/src/DiscriminantAnalysis_week12.ipynb>

## 3.1. Import Libraries

##### import pandas as pd

##### import numpy as np

##### from sklearn.model\_selection import train\_test\_split

##### from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

##### from sklearn.preprocessing import StandardScaler

##### from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

##### import matplotlib.pyplot as plt

##### import seaborn as sns

##### import kagglehub

##### import os

Pada bagian ini, dilakukan impor terhadap pustaka-pustaka (libraries) Python yang esensial untuk analisis data, manipulasi data, pemodelan machine learning, dan visualisasi.

* **Pandas**: Digunakan untuk manipulasi dan analisis data, khususnya untuk bekerja dengan struktur data DataFrame yang tabular.
* **NumPy**: Merupakan pustaka fundamental untuk komputasi numerik di Python, menyediakan dukungan untuk array dan matriks multidimensi, beserta kumpulan fungsi matematika tingkat tinggi untuk beroperasi pada array tersebut.
* **Scikit-learn (sklearn)**: Pustaka machine learning yang komprehensif. Dari sini, akan diimpor beberapa modul spesifik:
  + train\_test\_split: Untuk membagi dataset menjadi set pelatihan dan pengujian.
  + LinearDiscriminantAnalysis: Implementasi dari model Analisis Diskriminan Linier.
  + StandardScaler: Untuk melakukan standarisasi fitur (opsional, tetapi seringkali bermanfaat).
  + accuracy\_score, confusion\_matrix, classification\_report: Metrik untuk mengevaluasi performa model klasifikasi.
* **Matplotlib.pyplot** dan **Seaborn**: Digunakan untuk membuat visualisasi data statis, animasi, dan interaktif. (Meskipun mungkin tidak selalu digunakan secara ekstensif dalam LDA dasar, ada baiknya untuk diimpor jika diperlukan eksplorasi atau visualisasi hasil).
* **KaggleHub** dan **OS**: Digunakan spesifik untuk mengunduh dan mengakses dataset dari Kaggle, sesuai dengan instruksi pemuatan data yang diberikan.

## 3.2. Load Dataset

##### path = 'https://raw.githubusercontent.com/LatiefDataVisionary/multivariate-analysis-college-task/refs/heads/main/datasets/dataset\_LDA\_2clusters.csv'

|  | **BloodPressure** | **Age** | **Cluster** |
| --- | --- | --- | --- |
| **0** | **72** | **50** | **0** |
| **1** | **66** | **31** | **1** |
| **2** | **64** | **32** | **1** |
| **3** | **66** | **21** | **1** |
| **4** | **40** | **33** | **1** |

##### df.info()

##### **<class 'pandas.core.frame.DataFrame'>**

##### **RangeIndex: 600 entries, 0 to 599**

##### **Data columns (total 3 columns):**

##### **# Column Non-Null Count Dtype**

##### **--- ------ -------------- -----**

##### **0 BloodPressure 600 non-null int64**

##### **1 Age 600 non-null int64**

##### **2 Cluster 600 non-null int64**

##### **dtypes: int64(3)**

##### **memory usage: 14.2 KB**

##### df.describe()

|  | **BloodPressure** | **Age** | **Cluster** |
| --- | --- | --- | --- |
| **count** | **600.000000** | **600.000000** | **600.000000** |
| **mean** | **72.041667** | **33.278333** | **0.651667** |
| **std** | **11.980086** | **11.822315** | **0.476840** |
| **min** | **24.000000** | **21.000000** | **0.000000** |
| **25%** | **64.000000** | **24.000000** | **0.000000** |
| **50%** | **72.000000** | **29.000000** | **1.000000** |
| **75%** | **80.000000** | **40.000000** | **1.000000** |
| **max** | **122.000000** | **81.000000** | **1.000000** |

##### **df.shape**

##### **(600, 3)**

## 3.3. Data Preprocessing and Preparation

Tahap pra-pemrosesan data adalah langkah krusial sebelum membangun model machine learning. Tujuannya adalah untuk menyiapkan data dalam format yang sesuai untuk algoritma LDA.

1. **Memisahkan Fitur (Variabel Independen) dan Target (Variabel Dependen)**:

* Variabel independen (fitur) adalah kolom-kolom yang akan digunakan untuk membuat prediksi. Dalam kasus ini, fitur adalah BloodPressure dan Age. Variabel-variabel ini akan disimpan dalam DataFrame X.
* Variabel dependen (target) adalah kolom yang ingin diprediksi. Dalam kasus ini, target adalah kolom Cluster. Variabel ini akan disimpan dalam Series y.

1. **Membagi Data menjadi Set Pelatihan dan Pengujian (train\_test\_split)**:

* Dataset akan dibagi menjadi dua subset: set pelatihan (X\_train, y\_train) dan set pengujian (X\_test, y\_test).
* Model akan dilatih menggunakan set pelatihan.
* Performa model akan dievaluasi menggunakan set pengujian, yang merupakan data yang belum pernah dilihat oleh model selama proses pelatihan.
* Parameter test\_size=0.3 menunjukkan bahwa 30% dari data akan dialokasikan untuk set pengujian, dan sisanya (70%) untuk set pelatihan.
* Parameter random\_state=42 digunakan untuk memastikan bahwa pembagian data bersifat reproduktif. Artinya, jika kode dijalankan kembali dengan random\_state yang sama, pembagian datanya akan selalu sama. Ini penting untuk konsistensi hasil.

1. **Penskalaan Fitur (StandardScaler) (Opsional namun Direkomendasikan)**:

* LDA sensitif terhadap skala fitur, karena ia mencoba untuk memaksimalkan jarak antar kelas. Fitur dengan skala yang lebih besar mungkin mendominasi perhitungan jarak.
* StandardScaler dari scikit-learn digunakan untuk melakukan standarisasi fitur dengan menghilangkan mean dan menskalakan ke varians unit.
* Prosesnya adalah:
  + Menginisialisasi objek StandardScaler.
  + Melatih (fit) scaler **hanya** pada data fitur pelatihan (X\_train). Ini penting untuk menghindari kebocoran data (data leakage) dari set pengujian ke proses pelatihan scaler.
  + Mengaplikasikan (transform) penskalaan yang telah dipelajari ke data fitur pelatihan (X\_train) dan data fitur pengujian (X\_test).
* Setelah penskalaan, semua fitur akan memiliki mean sekitar 0 dan standar deviasi sekitar 1.

##### # Memastikan tidak ada nilai NaN di kolom yang akan digunakan

##### # Jika ada, perlu strategi penanganan (misal, imputasi atau penghapusan baris)

##### print("Missing values before any processing:")

##### print(df.isnull().sum())

##### 

##### **Missing values before any processing:**

##### **BloodPressure 0**

##### **Age 0**

##### **Cluster 0**

##### **dtype: int64**

### 3.3.1. Memisahkan Fitur (X) dan Target (y)

##### feature\_columns = ['BloodPressure', 'Age']

##### target\_column = 'Cluster'

##### X = df[feature\_columns]

##### y = df[target\_column]

##### print("\nShape of features (X):", X.shape)

##### print("Shape of target (y):", y.shape)

##### print("\nFirst 5 rows of features (X):")

##### print(X.head(10))

##### print("\nFirst 5 values of target (y):")

##### print(y.head(10))

##### 

##### Shape of features (X): (600, 2)

##### Shape of target (y): (600,)

### 3.3.2. Membagi Data menjadi Set Pelatihan dan Pengujian

##### # Umumnya menggunakan 70-80% untuk pelatihan dan 20-30% untuk pengujian

##### X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)

##### # stratify=y digunakan untuk memastikan proporsi kelas target sama di train dan test set, penting untuk klasifikasi

##### print("Shape of X\_train:", X\_train.shape)

##### print("Shape of X\_test:", X\_test.shape)

##### print("Shape of y\_train:", y\_train.shape)

##### print("Shape of y\_test:", y\_test.shape)

##### 

##### **Shape of X\_train: (420, 2)**

##### **Shape of X\_test: (180, 2)**

##### **Shape of y\_train: (420,)**

##### **Shape of y\_test: (180,)**

##### print("\nProportion of target variable in original dataset:")

##### print(y.value\_counts(normalize=True))

##### print("\nProportion of target variable in training set:")

##### print(y\_train.value\_counts(normalize=True))

##### print("\nProportion of target variable in test set:")

##### print(y\_test.value\_counts(normalize=True))

##### 

##### **Proportion of target variable in original dataset:**

##### **Cluster**

##### **1 0.651667**

##### **0 0.348333**

##### **Name: proportion, dtype: float64**

##### **Proportion of target variable in training set:**

##### **Cluster**

##### **1 0.652381**

##### **0 0.347619**

##### **Name: proportion, dtype: float64**

##### **Proportion of target variable in test set:**

##### **Cluster**

##### **1 0.65**

##### **0 0.35**

##### **Name: proportion, dtype: float64**

### 3.3.3. Penskalaan Fitur (Feature Scaling)

##### # Inisialisasi StandardScaler

##### scaler = StandardScaler()

##### # Fit scaler pada data pelatihan dan transformasikan data pelatihan

##### X\_train\_scaled = scaler.fit\_transform(X\_train)

##### # Transformasikan data pengujian menggunakan scaler yang sudah di-fit pada data pelatihan

##### X\_test\_scaled = scaler.transform(X\_test)

##### # Mengubah hasil scaling kembali menjadi DataFrame Pandas (opsional, untuk kemudahan inspeksi)

##### X\_train\_scaled\_df = pd.DataFrame(X\_train\_scaled, columns=feature\_columns)

##### X\_test\_scaled\_df = pd.DataFrame(X\_test\_scaled, columns=feature\_columns)

##### print("First 5 rows of scaled training features (X\_train\_scaled\_df):")

##### print(X\_train\_scaled\_df.head())

##### print("\nDescriptive statistics of scaled training features:")

##### print(X\_train\_scaled\_df.describe().round(2)) *# .round(2) untuk membulatkan agar mudah dibaca*

##### print("\nFirst 5 rows of scaled test features (X\_test\_scaled\_df):")

##### print(X\_test\_scaled\_df.head())

##### **First 5 rows of scaled training features (X\_train\_scaled\_df):**

##### **BloodPressure Age**

##### **0 -0.002763 -0.774024**

##### **1 -0.334348 -0.689673**

##### **2 0.328822 -0.605321**

##### **3 -0.168556 -0.689673**

##### **4 0.163029 0.659949**

##### **Descriptive statistics of scaled training features:**

##### **BloodPressure Age**

##### **count 420.00 420.00**

##### **mean 0.00 -0.00**

##### **std 1.00 1.00**

##### **min -3.98 -1.03**

##### **25% -0.67 -0.77**

##### **50% -0.00 -0.35**

##### **75% 0.49 0.58**

##### **max 4.14 4.03**

##### **First 5 rows of scaled test features (X\_test\_scaled\_df):**

##### **BloodPressure Age**

##### **0 -0.665933 -1.027078**

##### **1 -0.831725 0.153841**

##### **2 -0.168556 0.322544**

##### **3 -0.334348 -0.267916**

##### **4 0.494614 1.081706**

Setelah langkah-langkah ini, data X\_train\_scaled, X\_test\_scaled, y\_train, dan y\_test siap digunakan untuk membangun dan mengevaluasi model LDA.

## 3.4. Model Building: Linear Discriminant Analysis (LDA)

Setelah data disiapkan dan diproses, langkah selanjutnya adalah membangun model Analisis Diskriminan Linier (LDA).

1. **Menginisialisasi Model LDA**:

* Sebuah instance dari kelas LinearDiscriminantAnalysis dari sklearn.discriminant\_analysis dibuat.
* Untuk kasus klasifikasi biner (dua kelompok/cluster seperti dalam dataset ini), LDA secara default akan menghasilkan satu komponen diskriminan (n\_components = min(n\_features, n\_classes - 1)). Jika ada 2 fitur dan 2 kelas, maka akan ada min(2, 2-1) = 1 komponen diskriminan. Parameter n\_components dapat diatur secara eksplisit jika LDA juga ingin digunakan untuk reduksi dimensi menjadi jumlah komponen yang lebih sedikit dari defaultnya, namun untuk klasifikasi saja, pengaturan default biasanya sudah cukup.

1. **Melatih Model LDA**:

* Model LDA yang telah diinisialisasi kemudian dilatih (di-"fit") menggunakan data fitur pelatihan yang telah diskalakan (X\_train\_scaled) dan variabel target pelatihan (y\_train).
* Selama proses fit, model LDA akan mempelajari parameter-parameter (koefisien dan intercept) yang paling baik untuk memisahkan antar kelas berdasarkan data pelatihan. Ini melibatkan perhitungan rata-rata kelas, matriks kovarians dalam kelas, dan matriks kovarians antar kelas untuk menemukan proyeksi yang memaksimalkan separabilitas kelas.

### 3.4.1. Menginisialisasi model LDA

Untuk klasifikasi, n\_components tidak perlu diatur secara eksplisit jika jumlahnya kurang dari n\_classes-1. Namun, jika ingin menggunakan LDA untuk reduksi dimensi, parameter n\_components bisa diatur. Dalam kasus ini, dengan 2 kelas, akan ada n\_classes - 1 = 1 komponen diskriminan.

##### lda\_model = LinearDiscriminantAnalysis(n\_components=1) *# Bisa juga: LinearDiscriminantAnalysis()*

### 3.4.2. Melatih model LDA menggunakan data pelatihan yang sudah diskalakan

##### lda\_model.fit(X\_train\_scaled, y\_train)

##### print(f"Number of discriminant components learned: {lda\_model.n\_components}")

##### 

##### **Number of discriminant components learned: 1**

## 3.5. Model Evaluation

Setelah model LDA berhasil dilatih, langkah selanjutnya adalah mengevaluasi performanya menggunakan data pengujian. Evaluasi ini memberikan gambaran seberapa baik model dapat menggeneralisasi kemampuannya pada data baru.

1. **Membuat Prediksi pada Data Pengujian**:

* Metode predict() dari model LDA yang telah dilatih (lda\_model) digunakan untuk membuat prediksi kelas pada data fitur pengujian yang telah diskalakan (X\_test\_scaled).
* Hasil prediksi ini (y\_pred) akan dibandingkan dengan nilai target sebenarnya dari data pengujian (y\_test).

1. **Menghitung Akurasi (accuracy\_score)**:

* Akurasi adalah metrik paling sederhana yang mengukur proporsi prediksi yang benar dari total jumlah prediksi.
* accuracy\_score(y\_test, y\_pred) menghitung nilai akurasi.

1. **Menampilkan Confusion Matrix (confusion\_matrix)**:

* Confusion matrix adalah tabel yang merangkum performa model klasifikasi. Untuk klasifikasi biner, tabelnya berukuran 2x2:
  + **True Positives (TP)**: Jumlah observasi positif yang diprediksi dengan benar sebagai positif.
  + **True Negatives (TN)**: Jumlah observasi negatif yang diprediksi dengan benar sebagai negatif.
  + **False Positives (FP) / Type I Error**: Jumlah observasi negatif yang salah diprediksi sebagai positif.
  + **False Negatives (FN) / Type II Error**: Jumlah observasi positif yang salah diprediksi sebagai negatif.
* confusion\_matrix(y\_test, y\_pred) menghasilkan matriks ini.
* Visualisasi confusion matrix menggunakan seaborn.heatmap dapat membuatnya lebih mudah dibaca dan diinterpretasikan.

1. **Menampilkan Laporan Klasifikasi (classification\_report)**:
   * Laporan klasifikasi memberikan metrik evaluasi yang lebih detail per kelas, termasuk:
     + **Precision**: Kemampuan model untuk tidak memberi label positif pada sampel yang sebenarnya negatif (TP / (TP + FP)).
     + **Recall (Sensitivity)**: Kemampuan model untuk menemukan semua sampel positif (TP / (TP + FN)).
     + **F1-score**: Rata-rata harmonik dari precision dan recall (2 \* (Precision \* Recall) / (Precision + Recall)). Ini berguna ketika ada ketidakseimbangan kelas.
     + **Support**: Jumlah aktual kemunculan setiap kelas dalam y\_test.
   * classification\_report(y\_test, y\_pred) menghasilkan laporan ini.

Metrik-metrik ini secara kolektif memberikan pemahaman yang komprehensif tentang seberapa baik model LDA melakukan tugas klasifikasi pada data pengujian.

### 3.5.1. Membuat prediksi pada data pengujian

##### # X\_test\_scaled didapatkan dari langkah pra-pemrosesan sebelumnya

##### y\_pred = lda\_model.predict(X\_test\_scaled)

### 3.5.2. Menghitung Akurasi

##### accuracy = accuracy\_score(y\_test, y\_pred)

##### print(f"Accuracy of the LDA model: {accuracy:.4f}")

##### print(f"Accuracy in percentage: {accuracy\*100:.2f}%")

##### 

##### **Accuracy of the LDA model: 0.9833**

##### **Accuracy in percentage: 98.33%**

### 3.5.3. Menampilkan Confusion Matrix

##### conf\_matrix = confusion\_matrix(y\_test, y\_pred)

##### print("Confusion Matrix:")

##### print(conf\_matrix)

##### 

##### **Confusion Matrix:**

##### **[[ 60 3]**

##### **[ 0 117]]**

##### Visualisasi Confusion Matrix

##### plt.figure(figsize=(8, 6))

##### sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

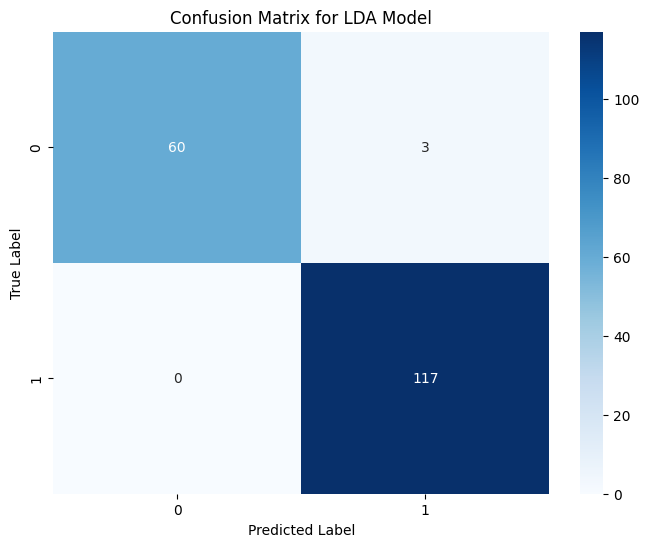
##### xticklabels=lda\_model.classes\_, yticklabels=lda\_model.classes\_)

##### plt.xlabel('Predicted Label')

##### plt.ylabel('True Label')

##### plt.title('Confusion Matrix for LDA Model')

##### plt.show()



### 3.5.4. Menampilkan Laporan Klasifikasi

##### class\_report = classification\_report(y\_test, y\_pred, target\_names=[f'Cluster {cls}' for cls in lda\_model.classes\_])

##### print("Classification Report:")

##### print(class\_report)

##### **Classification Report:**

##### **precision recall f1-score support**

##### **Cluster 0 1.00 0.95 0.98 63**

##### **Cluster 1 0.97 1.00 0.99 117**

##### **accuracy 0.98 180**

##### **macro avg 0.99 0.98 0.98 180**

##### **weighted avg 0.98 0.98 0.98 180**

## 3.6. Extracting Discriminant Function Coefficients

Model Analisis Diskriminan Linier (LDA) yang telah dilatih menghasilkan sebuah fungsi (atau beberapa fungsi, tergantung jumlah kelas dan fitur) yang digunakan untuk memproyeksikan data ke ruang berdimensi lebih rendah yang memaksimalkan separabilitas antar kelas. Untuk kasus klasifikasi biner dengan dua fitur, akan ada satu fungsi diskriminan linier.

Fungsi diskriminan ini memiliki bentuk: LD = coef₁ \* Feature₁\_scaled + coef₂ \* Feature₂\_scaled + ... + intercept

Atau, jika hanya ada satu fungsi diskriminan (kasus biner), nilai batas keputusan (threshold) seringkali diasumsikan 0 pada sumbu diskriminan ini. Skor diskriminan (LD\_score) untuk setiap observasi dapat dihitung, dan kemudian diklasifikasikan berdasarkan apakah skor tersebut di atas atau di bawah threshold.

1. **Mengekstrak Koefisien (lda\_model.coef\_)**:

* Atribut coef\_ pada objek model LDA yang telah dilatih berisi bobot (koefisien) untuk setiap fitur.
* Untuk klasifikasi biner, lda\_model.coef\_ akan menghasilkan array 2D dengan satu baris (karena hanya ada satu fungsi diskriminan) dan sejumlah kolom yang sama dengan jumlah fitur. Setiap nilai dalam baris tersebut adalah koefisien untuk fitur yang bersesuaian.
* Koefisien ini menunjukkan kontribusi relatif dari setiap fitur (yang telah diskalakan) terhadap skor fungsi diskriminan. Tanda (positif atau negatif) dari koefisien menunjukkan arah hubungan fitur dengan skor diskriminan.

1. **Mengekstrak Intercept (lda\_model.intercept\_)**:

* Atribut intercept\_ pada objek model LDA berisi nilai konstanta (bias) dari fungsi diskriminan.
* Untuk klasifikasi biner, ini adalah array 1D dengan satu elemen.

1. **Menyajikan Fungsi Diskriminan**:

* Dengan koefisien dan intercept yang telah diekstrak, fungsi diskriminan linier dapat dituliskan secara eksplisit.
* Penting untuk diingat bahwa koefisien ini berlaku untuk fitur yang **telah diskalakan** (jika penskalaan dilakukan sebelumnya, seperti StandardScaler).

Memahami koefisien ini dapat memberikan wawasan tentang fitur mana yang paling penting dalam membedakan antar kelas menurut model LDA.

### 3.6.1. Mengekstrak Koefisien

##### # lda\_model.coef\_ adalah array 2D, untuk klasifikasi biner, ini akan menjadi (1, n\_features)

##### coefficients = lda\_model.coef\_[0] # Ambil baris pertama karena hanya ada 1 fungsi diskriminan

##### 

##### Membuat Series Pandas untuk menampilkan koefisien dengan nama fitur

##### coeffs\_df = pd.DataFrame(coefficients, index=feature\_columns, columns=['Coefficient'])

##### print("Discriminant Function Coefficients (for scaled features):")

##### print(coeffs\_df)

##### 

##### **Discriminant Function Coefficients (for scaled features):**

##### **Coefficient**

##### **BloodPressure -1.939921**

##### **Age -4.606589**

### 3.6.2. Mengekstrak Intercept

##### intercept = lda\_model.intercept\_[0] # Intercept juga array, ambil elemen pertamanya

##### print(f"Intercept of the Discriminant Function: {intercept:.4f}")

##### 

##### **Intercept of the Discriminant Function: 2.0805**

### 3.6.3. Menyajikan Fungsi Diskriminan Linier

##### # Fungsi: LD\_score = (Coefficient\_BP \* Scaled\_BloodPressure) + (Coefficient\_Age \* Scaled\_Age) + Intercept

##### # Klasifikasi: Jika LD\_score > 0, prediksi kelas 1 (atau kelas dengan label lebih tinggi), jika < 0, prediksi kelas 0 (atau kelas dengan label lebih rendah).

##### # (Catatan: Scikit-learn menangani ambang batasnya secara internal berdasarkan nilai output dari lda\_model.decision\_function())

##### function\_string = f"LD\_Score = ({coefficients[0]:.4f} \* Scaled\_BloodPressure) + ({coefficients[1]:.4f} \* Scaled\_Age) + ({intercept:.4f})"

##### print("Linear Discriminant Function:")

##### print(function\_string)

**Linear Discriminant Function:**

**LD\_Score = (-1.9399 \* Scaled\_BloodPressure) + (-4.6066 \* Scaled\_Age) + (2.0805)**

##### print("\nPenjelasan Fungsi:")

##### print("Klasifikasi umumnya dilakukan berdasarkan tanda dari LD\_Score:")

##### print(f" - Jika LD\_Score > 0, observasi cenderung diklasifikasikan sebagai Cluster {lda\_model.classes\_[1]}.") *# Asumsi kelas 1 adalah kelas "positif"*

##### print(f" - Jika LD\_Score < 0, observasi cenderung diklasifikasikan sebagai Cluster {lda\_model.classes\_[0]}.") *# Asumsi kelas 0 adalah kelas "negatif"*

##### print("(Ambang batas sebenarnya yang digunakan oleh `predict` bisa sedikit berbeda dari 0 ketat, namun ini adalah interpretasi umumnya).")

##### *# Verifikasi manual dengan decision\_function untuk satu sampel*

##### *# Ambil satu sampel dari X\_test\_scaled*

##### sample\_scaled = X\_test\_scaled[0]

##### manual\_ld\_score = np.dot(sample\_scaled, coefficients) + intercept

##### sklearn\_decision\_value = lda\_model.decision\_function(sample\_scaled.reshape(1, -1))[0]

##### predicted\_class\_for\_sample = lda\_model.predict(sample\_scaled.reshape(1, -1))[0]

##### print(f"\nContoh perhitungan untuk sampel pertama dari X\_test\_scaled:")

##### print(f" Scaled\_BloodPressure: {sample\_scaled[0]:.4f}, Scaled\_Age: {sample\_scaled[1]:.4f}")

##### print(f" Manual LD\_Score calculated: ({coefficients[0]:.4f} \* {sample\_scaled[0]:.4f}) + ({coefficients[1]:.4f} \* {sample\_scaled[1]:.4f}) + ({intercept:.4f}) = {manual\_ld\_score:.4f}")

##### print(f" Scikit-learn decision\_function value: {sklearn\_decision\_value:.4f}")

##### print(f" Predicted class by lda\_model.predict(): Cluster {predicted\_class\_for\_sample}")

##### print(f" Actual class for this sample (from y\_test): Cluster {y\_test.iloc[0]}")

##### 

**Penjelasan Fungsi:**

**Klasifikasi umumnya dilakukan berdasarkan tanda dari LD\_Score:**

**- Jika LD\_Score > 0, observasi cenderung diklasifikasikan sebagai Cluster 1.**

**- Jika LD\_Score < 0, observasi cenderung diklasifikasikan sebagai Cluster 0.**

**(Ambang batas sebenarnya yang digunakan oleh `predict` bisa sedikit berbeda dari 0 ketat, namun ini adalah interpretasi umumnya).**

**Contoh perhitungan untuk sampel pertama dari X\_test\_scaled:**

**Scaled\_BloodPressure: -0.6659, Scaled\_Age: -1.0271**

**Manual LD\_Score calculated: (-1.9399 \* -0.6659) + (-4.6066 \* -1.0271) + (2.0805) = 8.1037**

**Scikit-learn decision\_function value: 8.1037**

**Predicted class by lda\_model.predict(): Cluster 1**

**Actual class for this sample (from y\_test): Cluster 1**

## 3.7. Visualization

Visualisasi dapat memberikan pemahaman intuitif tentang bagaimana model LDA bekerja dan bagaimana data dipisahkan.

1. **Distribusi Fitur per Kelas**:
   * Membuat histogram atau density plot untuk setiap fitur (BloodPressure dan Age), yang diwarnai berdasarkan kelas (Cluster). Ini membantu untuk melihat apakah ada perbedaan distribusi yang jelas antara dua kelas untuk masing-masing fitur.
2. **Scatter Plot Fitur Asli dengan Batas Keputusan LDA**:

* Karena kita memiliki dua fitur, data asli (sebelum atau sesudah penskalaan) dapat divisualisasikan dalam scatter plot 2D.
* Batas keputusan (decision boundary) yang dipelajari oleh LDA dapat digambarkan sebagai garis pada plot ini. Garis ini memisahkan area di mana model akan memprediksi Kelas 0 dari area di mana ia akan memprediksi Kelas 1.
* Untuk mendapatkan garis batas keputusan, kita bisa menggunakan koefisien dan intercept dari model LDA. Batas keputusan adalah tempat di mana fungsi diskriminan (atau decision\_function) sama dengan nol. coef₁ \* x₁\_scaled + coef₂ \* x₂\_scaled + intercept = 0 Jika kita menggunakan data yang diskalakan, x₁\_scaled dan x₂\_scaled adalah variabelnya. Kita bisa mengekspresikan salah satu variabel dalam bentuk yang lain untuk menggambar garisnya. Misal, x₂\_scaled = (-coef₁ \* x₁\_scaled - intercept) / coef₂. Kemudian, nilai-nilai x₁\_scaled dan x₂\_scaled ini perlu di-transformasikan kembali ke skala fitur asli jika kita ingin menggambar batas keputusan pada plot fitur asli.

1. **Distribusi Skor Diskriminan (LD Scores)**:

* Model LDA memproyeksikan data ke satu sumbu diskriminan (karena ini adalah masalah klasifikasi biner). Kita bisa menghitung skor diskriminan (proyeksi) untuk setiap titik data pada sumbu ini.
* Membuat histogram atau density plot dari skor diskriminan ini, yang diwarnai berdasarkan kelas aktual, dapat menunjukkan seberapa baik sumbu diskriminan tersebut memisahkan kedua kelas. Idealnya, distribusi skor untuk kedua kelas akan memiliki tumpang tindih (overlap) yang minimal.
* Skor ini bisa didapatkan dari lda\_model.transform(X\_scaled).

Visualisasi ini membantu dalam interpretasi model dan identifikasi apakah asumsi LDA (seperti distribusi normal dan kesamaan kovarians) mungkin terpenuhi secara kasar.

### 3.7.1. Distribusi Fitur per Kelas

plt.figure(figsize=(14, 6))

##### plt.subplot(1, 2, 1)

##### sns.histplot(data=df, x='BloodPressure', hue='Cluster', kde=True, palette='viridis')

##### plt.title('Distribution of BloodPressure by Cluster')

##### plt.xlabel('BloodPressure')

##### plt.ylabel('Frequency')

##### plt.subplot(1, 2, 2)

##### sns.histplot(data=df, x='Age', hue='Cluster', kde=True, palette='viridis')

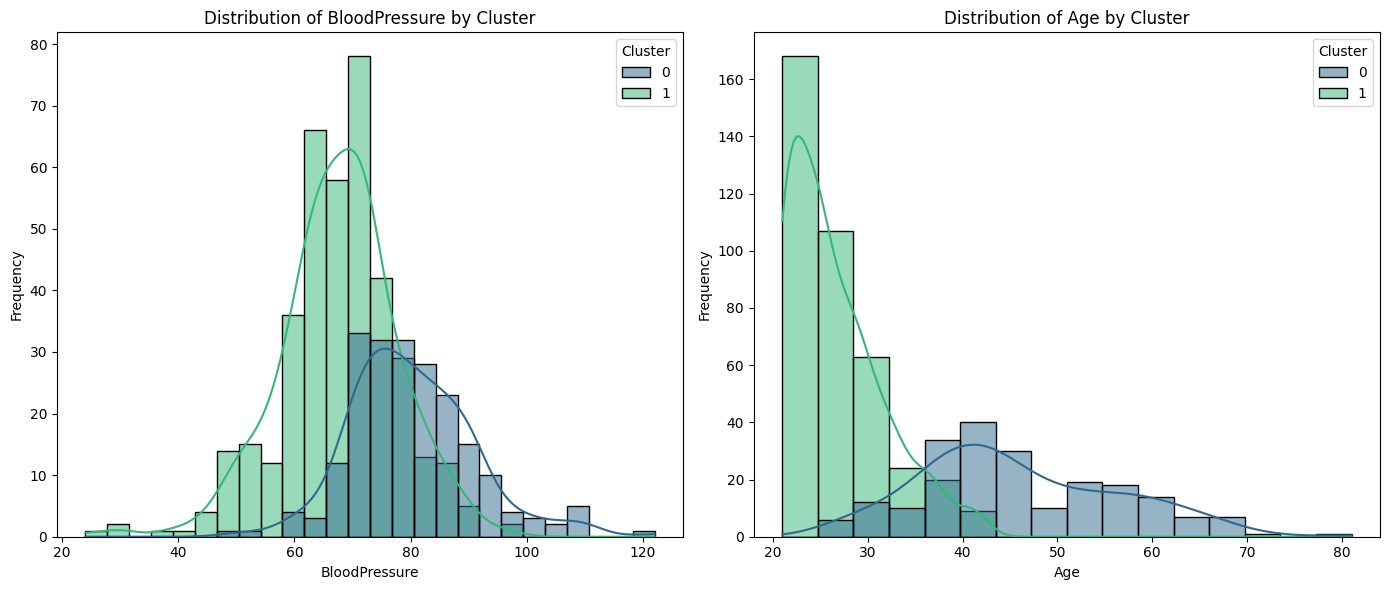
##### plt.title('Distribution of Age by Cluster')

##### plt.xlabel('Age')

##### plt.ylabel('Frequency')

##### plt.tight\_layout()

##### plt.show()



### 3.7.2. Scatter Plot Fitur (Scaled) dengan Batas Keputusan LDA

##### # Kita akan menggunakan data yang diskalakan karena model dilatih padanya.

##### # Ambil koefisien dan intercept dari model LDA

##### coef = lda\_model.coef\_[0]

##### intercept = lda\_model.intercept\_[0]

##### # Membuat meshgrid untuk plot batas keputusan

##### # Kita perlu menentukan rentang untuk x1\_scaled (BloodPressure\_scaled)

##### # dan kemudian hitung x2\_scaled (Age\_scaled) berdasarkan persamaan batas keputusan.

##### # Persamaan batas: coef[0]\*x1\_s + coef[1]\*x2\_s + intercept = 0

##### # Maka, x2\_s = (-coef[0]\*x1\_s - intercept) / coef[1]

##### # Plot data yang diskalakan dari set pelatihan (atau bisa juga keseluruhan data yang diskalakan)

##### X\_scaled\_df = pd.DataFrame(scaler.transform(X), columns=feature\_columns) # Skalakan seluruh X

##### X\_scaled\_df['Cluster'] = y.values # Tambahkan kolom cluster untuk pewarnaan

##### plt.figure(figsize=(10, 7))

##### sns.scatterplot(data=X\_scaled\_df, x='BloodPressure', y='Age', hue='Cluster', palette='viridis', s=50, alpha=0.7)

##### plt.title('LDA Decision Boundary on Scaled Features')

##### plt.xlabel('Scaled BloodPressure')

##### plt.ylabel('Scaled Age')

##### # Menyiapkan titik-titik untuk garis batas keputusan

##### ax = plt.gca()

##### xlim = ax.get\_xlim()

##### ylim = ax.get\_ylim()

##### # Buat meshgrid (alternatif: cukup plot garisnya saja)

##### xx = np.linspace(xlim[0], xlim[1], 100)

##### yy = np.linspace(ylim[0], ylim[1], 100)

XX, YY = np.meshgrid(xx, yy)

##### Z = lda\_model.decision\_function(np.c\_[XX.ravel(), YY.ravel()])

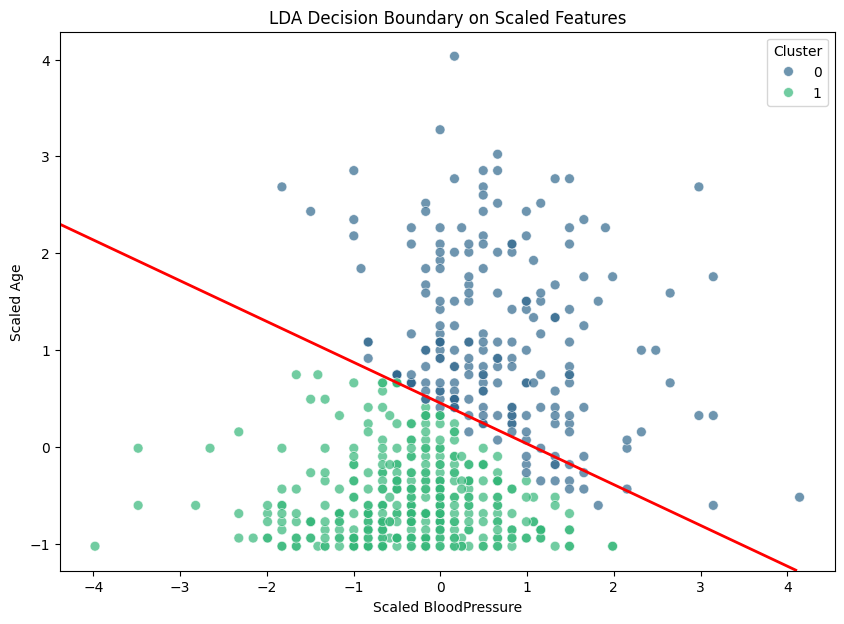
##### Z = Z.reshape(XX.shape)

##### *# Plot batas keputusan (contour di mana decision\_function == 0)*

##### plt.contour(XX, YY, Z, levels=[0], linewidths=2, colors='red')

##### plt.legend(title='Cluster')

##### plt.show()



### 3.7.3. Distribusi Skor Diskriminan (LD Scores)

##### # Transformasikan data fitur yang diskalakan (misalnya, X\_train\_scaled atau X\_test\_scaled) ke komponen diskriminan

##### # Karena n\_components=1, hasilnya akan menjadi array 1D.

##### X\_lda\_train = lda\_model.transform(X\_train\_scaled)

##### X\_lda\_test = lda\_model.transform(X\_test\_scaled) # Atau gunakan seluruh data jika diinginkan

##### # Membuat DataFrame untuk kemudahan plotting dengan Seaborn

##### df\_lda\_scores\_train = pd.DataFrame({'LD1\_Score': X\_lda\_train[:, 0], 'Cluster': y\_train})

##### df\_lda\_scores\_test = pd.DataFrame({'LD1\_Score': X\_lda\_test[:, 0], 'Cluster': y\_test})

##### plt.figure(figsize=(10, 6))

##### sns.histplot(data=df\_lda\_scores\_train, x='LD1\_Score', hue='Cluster', kde=True, palette='viridis', stat="density", common\_norm=False)

##### plt.title('Distribution of LDA Scores (Training Data) by Cluster')

##### plt.xlabel('LDA Score (LD1)')

##### plt.ylabel('Density')

##### plt.axvline(0, color='grey', linestyle='--', lw=2, label='Decision Threshold (approx.)') # Ambang batas umumnya 0

##### plt.legend()

##### plt.show()

##### plt.figure(figsize=(10, 6))

##### sns.histplot(data=df\_lda\_scores\_test, x='LD1\_Score', hue='Cluster', kde=True, palette='viridis', stat="density", common\_norm=False)

##### plt.title('Distribution of LDA Scores (Test Data) by Cluster')

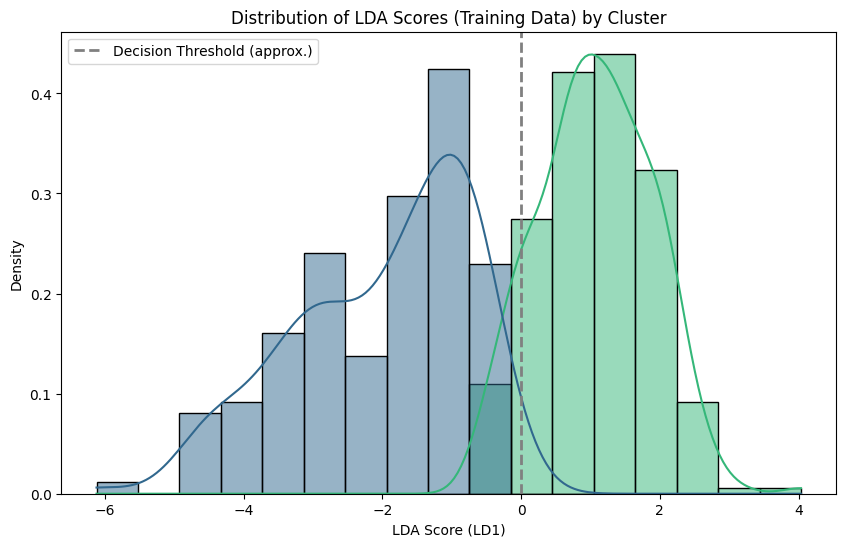
##### plt.xlabel('LDA Score (LD1)')

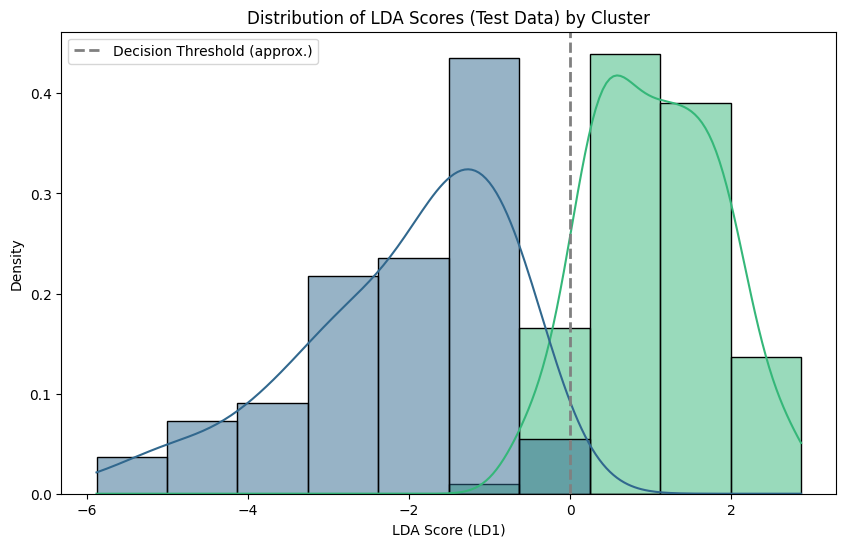
##### plt.ylabel('Density')

##### plt.axvline(0, color='grey', linestyle='--', lw=2, label='Decision Threshold (approx.)')

##### plt.legend()

##### plt.show()





## 3.8. Making Predictions on New Data

Setelah model LDA dilatih dan dievaluasi, model tersebut dapat digunakan untuk membuat prediksi pada data baru yang belum pernah dilihat sebelumnya dan tidak memiliki label kelas.

1. **Menyiapkan Data Baru**:

* Data baru harus memiliki struktur kolom fitur yang sama dengan data yang digunakan untuk melatih model (yaitu, BloodPressure dan Age dalam kasus ini).
* Data baru ini akan dibuat sebagai DataFrame Pandas.

1. **Menerapkan Penskalaan Fitur**:

* Sangat penting untuk menerapkan **transformasi penskalaan yang sama** yang digunakan pada data pelatihan. Ini berarti menggunakan objek scaler yang telah di-fit pada X\_train untuk melakukan transform pada data baru.
* Jangan melakukan fit atau fit\_transform baru pada data baru, karena ini akan menskalakannya berdasarkan statistik data baru itu sendiri, bukan berdasarkan statistik data pelatihan, yang akan menyebabkan inkonsistensi.

1. **Membuat Prediksi**:

* Metode predict() dari lda\_model digunakan pada data baru yang telah diskalakan untuk mendapatkan prediksi kelas.
* Metode predict\_proba() juga dapat digunakan jika ingin mendapatkan probabilitas keanggotaan kelas untuk setiap sampel data baru. LDA secara inheren adalah model linier dan probabilitasnya diestimasi (biasanya dengan menerapkan fungsi logistik atau softmax pada skor diskriminan).
* Metode decision\_function() dapat digunakan untuk mendapatkan skor mentah pada sumbu diskriminan.

Hasil prediksi ini akan menunjukkan bagaimana model mengklasifikasikan observasi baru berdasarkan pola yang telah dipelajarinya dari data pelatihan.

### 3.8.1. Menyiapkan Data Baru

##### new\_data\_dict = {

##### 'BloodPressure': [76, 92, 58, 74, 62, 76, 70, 72, 60, 70],

##### 'Age': [26, 66, 22, 43, 33, 63, 27, 30, 47, 23]

##### }

##### new\_df = pd.DataFrame(new\_data\_dict)

##### print("New data to predict:")

##### print(new\_df)

##### New data to predict:

##### BloodPressure Age

##### 0 76 26

##### 1 92 66

##### 2 58 22

##### 3 74 43

##### 4 62 33

##### 5 76 63

##### 6 70 27

##### 7 72 30

##### 8 60 47

##### 9 70 23

### 3.8.2. Menerapkan Penskalaan Fitur

##### # Gunakan scaler yang sudah di-fit pada X\_train

##### # scaler ini sudah diinisialisasi dan di-fit di langkah pra-pemrosesan

##### new\_df\_scaled\_array = scaler.transform(new\_df)

##### # Untuk kemudahan inspeksi, ubah menjadi DataFrame (opsional)

##### new\_df\_scaled = pd.DataFrame(new\_df\_scaled\_array, columns=feature\_columns)

##### print("New data after scaling (using the training set's scaler):")

##### print(new\_df\_scaled)

##### 

##### **New data after scaling (using the training set's scaler):**

##### **BloodPressure Age**

##### **0 0.328822 -0.605321**

##### **1 1.655161 2.768733**

##### **2 -1.163310 -0.942727**

##### **3 0.163029 0.828652**

##### **4 -0.831725 -0.014862**

##### **5 0.328822 2.515679**

##### **6 -0.168556 -0.520970**

##### **7 -0.002763 -0.267916**

##### **8 -0.997518 1.166057**

##### **9 -0.168556 -0.858375**

### 3.8.3. Membuat Prediksi

##### # Menggunakan model LDA yang telah dilatih (lda\_model)

##### new\_predictions\_class = lda\_model.predict(new\_df\_scaled\_array)

##### new\_predictions\_proba = lda\_model.predict\_proba(new\_df\_scaled\_array)

##### new\_decision\_scores = lda\_model.decision\_function(new\_df\_scaled\_array)

##### # Menambahkan hasil prediksi ke DataFrame data baru untuk kemudahan analisis

##### new\_df['Predicted\_Cluster'] = new\_predictions\_class

##### new\_df['Probability\_Cluster\_0'] = new\_predictions\_proba[:, 0] # Probabilitas untuk kelas 0

##### new\_df['Probability\_Cluster\_1'] = new\_predictions\_proba[:, 1] # Probabilitas untuk kelas 1

##### new\_df['LD\_Score'] = new\_decision\_scores

##### print("New data with predictions and probabilities:")

##### print(new\_df)

##### print("\n" + "="\*50 + "\n")

##### print("Interpretation of results:")

##### for index, row in new\_df.iterrows():

##### print(f"Sample {index + 1} (BP: {row['BloodPressure']}, Age: {row['Age']}):")

##### print(f" - Predicted Cluster: {row['Predicted\_Cluster']}")

##### print(f" - Probability for Cluster 0: {row['Probability\_Cluster\_0']:.4f}")

##### print(f" - Probability for Cluster 1: {row['Probability\_Cluster\_1']:.4f}")

##### print(f" - Raw LDA Score (Decision Function): {row['LD\_Score']:.4f}\n")

**New data with predictions and probabilities:**

**BloodPressure Age Predicted\_Cluster Probability\_Cluster\_0 \**

**0 76 26 1 0.014328**

**1 92 66 0 0.999999**

**2 58 22 1 0.000170**

**3 74 43 0 0.886251**

**4 62 33 1 0.022699**

**5 76 63 0 0.999961**

**6 70 27 1 0.008103**

**7 72 30 1 0.034889**

**8 60 47 0 0.795096**

**9 70 23 1 0.001723**

**Probability\_Cluster\_1 LD\_Score**

**0 9.856718e-01 4.231096**

**1 9.330757e-07 -13.884779**

**2 9.998301e-01 8.680003**

**3 1.137492e-01 -2.053004**

**4 9.773007e-01 3.762462**

**5 3.922847e-05 -10.146069**

**6 9.918971e-01 4.807397**

**7 9.651105e-01 3.320057**

**8 2.049038e-01 -1.355922**

**9 9.982765e-01 6.361685**

## 3.9. Kesimpulan

Analisis Diskriminan Linier (LDA) telah berhasil diterapkan untuk mengklasifikasikan golongan berdasarkan variabel BloodPressure dan Age menggunakan dataset yang terdiri dari 600 observasi. Implementasi menggunakan Python dengan pustaka Scikit-learn menunjukkan kemampuan model dalam membedakan antara dua cluster (Cluster 0 dan Cluster 1).

Model LDA yang dibangun menghasilkan fungsi diskriminan linier berikut, di mana Scaled\_BloodPressure dan Scaled\_Age merujuk pada nilai fitur yang telah melalui proses standarisasi (penskalaan):  
**LD\_Score = (-1.9399 \* Scaled\_BloodPressure) + (-4.6066 \* Scaled\_Age) + 2.0805**

Fungsi ini mengindikasikan kontribusi relatif dari masing-masing fitur (yang telah diskalakan) terhadap skor diskriminan. Koefisien negatif untuk Scaled\_BloodPressure (-1.9399) dan Scaled\_Age (-4.6066) menunjukkan bahwa peningkatan nilai pada kedua fitur tersebut (setelah penskalaan) cenderung menurunkan nilai LD\_Score. Intercept dari fungsi diskriminan adalah 2.0805. Klasifikasi dilakukan berdasarkan tanda dari LD\_Score: skor positif cenderung mengarah ke klasifikasi sebagai Cluster 1, sedangkan skor negatif cenderung mengarah ke Cluster 0.

Evaluasi performa model pada data pengujian (test set), yang terdiri dari 180 sampel, menunjukkan hasil yang sangat baik. **Akurasi keseluruhan model mencapai 98.33%**. Hal ini berarti model mampu mengklasifikasikan observasi dengan benar pada sebagian besar kasus.

Analisis lebih lanjut melalui *confusion matrix* memberikan rincian sebagai berikut:

* Untuk **Cluster 0**, dari 63 sampel aktual, 60 diprediksi dengan benar sebagai Cluster 0 (True Negative), dan 3 salah diprediksi sebagai Cluster 1 (False Positive).
* Untuk **Cluster 1**, dari 117 sampel aktual, 117 diprediksi dengan benar sebagai Cluster 1 (True Positive), dan tidak ada (0) yang salah diprediksi sebagai Cluster 0 (False Negative).

Dari *classification report*, diperoleh metrik performa per kelas:

* **Cluster 0**: Precision mencapai 1.00 (sempurna, artinya semua yang diprediksi sebagai Cluster 0 memang benar Cluster 0), Recall sebesar 0.95 (artinya 95% dari semua Cluster 0 yang sebenarnya berhasil diidentifikasi), dan F1-score sebesar 0.98.
* **Cluster 1**: Precision sebesar 0.97, Recall mencapai 1.00 (sempurna, artinya semua Cluster 1 yang sebenarnya berhasil diidentifikasi), dan F1-score sebesar 0.99.

Secara keseluruhan, metrik-metrik ini (terutama akurasi tinggi, serta nilai precision, recall, dan F1-score yang mendekati 1 untuk kedua kelas) mengindikasikan bahwa model LDA yang dibangun sangat efektif dan memiliki kemampuan generalisasi yang baik dalam membedakan antara dua cluster berdasarkan variabel BloodPressure dan Age pada dataset yang digunakan. Pengujian pada 10 data baru juga menunjukkan aplikasi fungsi diskriminan ini dalam menghasilkan prediksi kelas dan skor LD untuk observasi yang belum pernah dilihat sebelumnya.

# Kesimpulan Akhir

Tugas Analisis Diskriminan Linier (LDA) untuk klasterisasi golongan berdasarkan variabel BloodPressure dan Age telah berhasil dilaksanakan melalui tiga pendekatan berbeda: perhitungan manual menggunakan Microsoft Excel, implementasi perhitungan manual menggunakan Python, dan penggunaan pustaka Scikit-learn (LDA) di Python. Ketiga metode ini bertujuan untuk membangun fungsi diskriminan yang optimal dalam membedakan antara dua cluster (Cluster 0 dan Cluster 1) pada dataset yang terdiri dari 600 observasi.

Perbandingan hasil dari ketiga pendekatan menunjukkan konsistensi yang tinggi dalam performa klasifikasi. Baik metode manual menggunakan Excel maupun metode manual menggunakan Python menghasilkan **akurasi klasifikasi keseluruhan yang identik, yaitu sebesar 97.33%** pada dataset yang digunakan. Ini menunjukkan bahwa implementasi matematis langkah-langkah LDA secara manual, baik melalui spreadsheet maupun scripting Python, telah dilakukan dengan benar dan menghasilkan model dengan kemampuan prediktif yang sebanding.

Lebih lanjut, model LDA yang dibangun menggunakan pustaka Scikit-learn di Python, setelah melalui proses pra-pemrosesan termasuk penskalaan fitur, menunjukkan **akurasi yang sedikit lebih tinggi, yaitu sebesar 98.33%** pada data pengujian. Perbedaan minor ini mungkin disebabkan oleh optimasi internal dan penggunaan data yang diskalakan dalam implementasi Scikit-learn, yang dapat menangani variasi skala antar fitur dengan lebih efektif.

Menariknya, ketika dilakukan pengujian pada 10 sampel data baru yang sama untuk ketiga pendekatan, hasil klasifikasi menunjukkan kesamaan yang signifikan. Perbedaan prediksi hanya teramati pada satu baris data pertama pada metode Excel dibandingkan dengan kedua metode Python. Untuk sembilan baris data uji berikutnya, ketiga metode memberikan hasil klasifikasi yang konsisten. Hal ini semakin memperkuat validitas dan reliabilitas fungsi diskriminan yang telah berhasil diturunkan melalui berbagai pendekatan.

Fungsi diskriminan yang dihasilkan, meskipun memiliki nilai koefisien yang berbeda antara metode manual (yang menggunakan fitur asli) dan metode Scikit-learn (yang menggunakan fitur yang diskalakan), secara fundamental bertujuan sama: mengoptimalkan pemisahan antar cluster. Perbedaan nilai koefisien ini wajar mengingat perbedaan skala input fitur yang digunakan dalam perhitungannya.

Secara keseluruhan, pelaksanaan tugas ini melalui tiga metode berbeda tidak hanya berhasil mencapai tujuan klasterisasi dengan akurasi yang sangat baik, tetapi juga memberikan pemahaman komprehensif mengenai mekanisme Analisis Diskriminan Linier. Konsistensi hasil antara perhitungan manual dan penggunaan pustaka standar menegaskan validitas pendekatan yang digunakan, sementara perbedaan kecil dalam akurasi menyoroti potensi manfaat dari teknik pra-pemrosesan standar seperti penskalaan fitur dalam implementasi machine learning.