

# Pemodelan Data

Modul Praktikum  
Penambangan Data 2025

**MODUL 2**



Arranged By  
**DASPRO Laboratory Team**

# Asisten Praktikum



**AANG**

Arya Anggadipa Manova



**AAAA**

Alvaro Cleosanda



**JHON**

I Gede Made Ari Ananta



**KAYE**

Kaisa Izzatunniswah



**KYLE**

Kayla Zhafira Ardinov



**KYRA**

Kirana Adira Syifa



**LALA**

Lailatul Hadhari



**MALE**

Matthew Alexander H Sitorus



**MIIZ**

Mochamad Irdan Iqbal Z



**NICE**

Nisa Trinanda Utami



**PDLY**

Muhammad Fadli M



**QIRA**

Ariq Naufal Wahyono



**QLAB**

Balqis Eka Nurfadisyah



**RHAN**

Sarah Luki Raihani

# Asisten Praktikum



**RIAU**

Alfan Gunawan Akhmad



**RIDH**

Muhammad Ridho Irhamna



**SATH**

Muhammad Salik Arethe



**SRSA**

Shalvia Retno Salsabil



**TSAA**

Maitsa Luthfiyyah Durrotunnisa



**WANG**

Ikhwan Amiruddin



**XENA**

Goesniawan Xena Adikara



**ZURU**

Daffa Muhammad Zulfikar

### DAFTAR ISI

<b>PENDAHULUAN.....</b>	<b>6</b>
<b>SPLIT DATA.....</b>	<b>11</b>
<b>EVALUASI MODEL.....</b>	<b>13</b>
A. Evaluasi Model Regresi.....	13
1. Mean Squared Error (MSE).....	13
2. R-Squared atau R2 Score.....	13
B. Evaluasi Model Klasifikasi.....	14
1. Accuracy.....	14
2. Confusion Matrix.....	15
3. Classification Report.....	17
C. Evaluasi Model Clustering.....	18
1. Metriks Evaluasi Internal.....	19
2. Metrik Evaluasi Eksternal.....	21
<b>OVERFITTING &amp; UNDERFITTING.....</b>	<b>22</b>
A. Overfitting.....	22
B. Underfitting.....	23
<b>TUNING MODEL OPTIMATION.....</b>	<b>24</b>
A. Linear Regression Hyperparameters.....	24
B. Lasso Regression.....	25
C. Logistik Regression Hyperparameters.....	25
D. K-Means Clustering.....	27
E. DBSCAN (Density-Based Spatial Clustering of Applications with Noise).....	28
F. Hierarchical Clustering.....	29
<b>K-FOLD CROSS VALIDATION.....</b>	<b>31</b>
A. Definisi K-Fold Cross Validation.....	31
B. Implementasi K-Fold.....	32

<b>GRIDSEARCHCV.....</b>	<b>34</b>
A. Definisi GridSearchCV.....	34
B. Implementasi GridSearchCV pada Regression.....	35
C. Implementasi GridSearchCV pada Clustering.....	37
<b>REGRESI.....</b>	<b>42</b>
A. Analisis Statistik (Statistical Analysis).....	42
B. Linear Regression.....	44
C. Logistic Regression.....	48
D. Lasso Regression.....	51
<b>CLUSTERING.....</b>	<b>55</b>
A. Hierarchical Clustering.....	55
a. AGNES (Agglomerative Nesting).....	55
b. DIANA (Divisive Analysis).....	62
B. Partitional Clustering.....	63
a. K-Means.....	63
C. Density-Based Clustering.....	71
a. DBSCAN (Density-Based Spatial Clustering of Application with Noise).....	71
<b>REFERENSI.....</b>	<b>80</b>



### PENDAHULUAN

#### A. Machine Learning

Pemodelan data adalah suatu model atau pola yang menangkap keteraturan dalam data. Tahap pemodelan adalah tempat utama di mana teknik data mining diterapkan pada data. Penting untuk memiliki pemahaman tentang ide-ide dasar data mining, termasuk jenis-jenis teknik dan algoritma yang ada.

### The Machine Learning Process

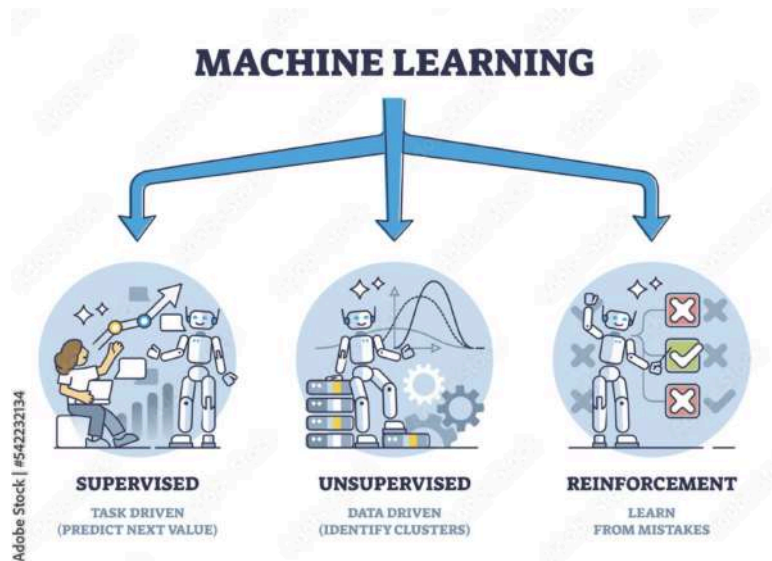


Pada praktikum kali ini kita akan memahami lebih dalam tentang apa itu machine learning. Machine learning adalah cabang dari kecerdasan buatan yang memungkinkan algoritma untuk menemukan pola tersembunyi dalam dataset, sehingga dapat membuat prediksi pada data baru yang serupa tanpa perlu diprogram secara eksplisit untuk setiap tugas. Machine learning tradisional menggabungkan data dengan alat statistik untuk memprediksi hasil, menghasilkan wawasan yang dapat ditindaklanjuti. Teknologi ini memiliki aplikasi di berbagai bidang, seperti pengenalan gambar dan suara, pemrosesan bahasa alami, sistem rekomendasi, deteksi penipuan, optimisasi portofolio, dan otomatisasi tugas.

Sebagai contoh:

Sistem rekomendasi menggunakan data historis untuk memberikan saran yang dipersonalisasi. Netflix, misalnya, menggunakan collaborative dan content-based filtering untuk merekomendasikan film dan acara TV berdasarkan riwayat tontonan pengguna, peringkat, dan preferensi genre.

### B. Pendekatan dalam Machine Learning



Berdasarkan pendekatannya, Machine learning dibagi menjadi 3 bagian besar diantaranya adalah sebagai berikut:

#### 1. Supervised learning

Supervised learning adalah jenis machine learning di mana algoritma dilatih menggunakan dataset yang telah diberi label. Model belajar untuk memetakan fitur input ke output target berdasarkan data pelatihan yang sudah dilabeli. Tujuan utama supervised learning adalah membuat model yang mampu menggeneralisasi dari data pelatihan sehingga dapat **memprediksi hasil pada data baru yang belum pernah dilihat sebelumnya**. Supervised learning dibagi menjadi dua bagian, yaitu:

- Regresi:** Model mempelajari untuk memprediksi nilai kontinu berdasarkan fitur input. Contohnya meliputi prediksi harga rumah atau harga saham. Algoritma yang sering digunakan mencakup **Linear Regression, Logistic Regression, Polynomial Regression, Support Vector Regression, dan lainnya**.
- Klasifikasi:** Model mempelajari untuk mengelompokkan data ke dalam kategori tertentu. Algoritma ini menghasilkan output berupa nilai diskrit, seperti mengklasifikasikan email sebagai spam atau bukan. Algoritma yang umum digunakan termasuk **Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), dan Support Vector Machine (SVM)**.

### 2. Unsupervised Learning

Unsupervised learning adalah jenis machine learning di mana algoritma mempelajari pola dalam data tanpa bantuan label atau output target. Tujuan dari metode ini adalah menemukan struktur atau distribusi tersembunyi dalam data. Unsupervised Learning dibagi menjadi dua bagian, yaitu:

- a. **Clustering:** Algoritma mengelompokkan data menjadi kelompok-kelompok berdasarkan karakteristik yang serupa. Algoritma populer dalam kategori ini termasuk **K-Means, Hierarchical Clustering, dan DBSCAN**.
- b. **Dimensionality Reduction:** Algoritma ini mengurangi jumlah variabel input dalam dataset sambil tetap mempertahankan informasi penting. Contohnya meliputi **Principal Component Analysis (PCA) dan t-SNE**.

### 3. Reinforcement Learning

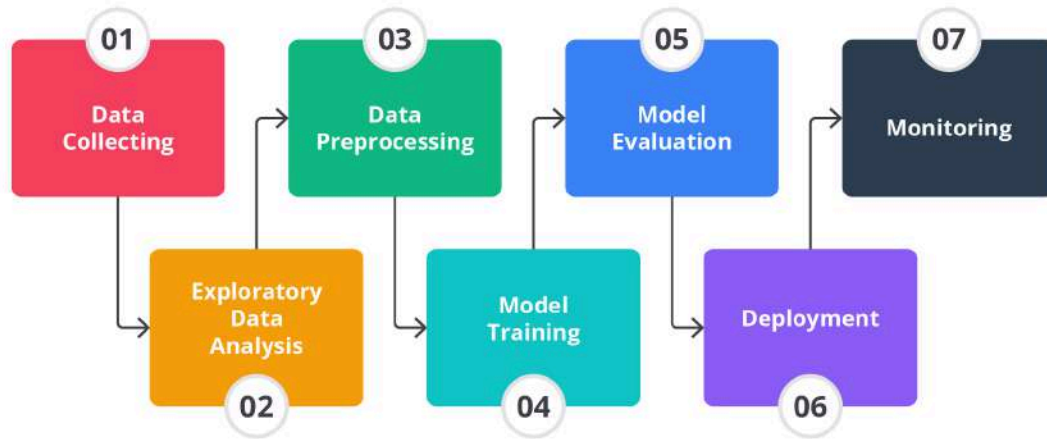
Reinforcement learning adalah jenis machine learning di mana agen belajar berinteraksi dengan lingkungan melalui tindakan dan menerima umpan balik dalam bentuk reward atau penalti. Tujuan reinforcement learning adalah mempelajari kebijakan (policy) yang memetakan keadaan ke tindakan yang memaksimalkan reward kumulatif. Reinforcement learning dibagi menjadi dua bagian, yaitu;

- a. **Model-based Reinforcement Learning:** Agen mempelajari model dari lingkungan yang mencakup probabilitas transisi antar keadaan dan reward yang terkait dengan setiap pasangan keadaan-tindakan. Algoritma seperti Value Iteration dan Policy Iteration digunakan dalam kategori ini.
- b. **Model-free Reinforcement Learning:** Agen mempelajari kebijakan langsung dari pengalaman tanpa membangun model eksplisit dari lingkungan. Algoritma populer meliputi Q-Learning, SARSA, dan Deep Reinforcement Learning.

Pada praktikum kali ini kita akan mengenal lebih dalam terkait Regression Learning dan Clustering learning dengan fokus model Logistic Regression dan K-Means.



### C. Alur Sederhana Implementasi Machine Learning



#### 1. Data Collecting

Mengumpulkan data relevan sebagai dasar pelatihan model. Data bisa didapat dari berbagai sumber, seperti database atau API.

#### 2. Exploratory Data Analysis

Menganalisis data untuk memahami karakteristiknya, mencari pola, dan mendeteksi anomali. Tahap ini memberi wawasan awal.

#### 3. Data Selection

Pemilihan fitur adalah tahap dalam pipeline machine learning di mana fitur-fitur yang paling relevan dipilih berdasarkan tujuan bisnis dan karakteristik data yang telah dianalisis. Hal ini bisa mempertimbangkan

- 1) Analisis Hubungan Fitur dengan Target (Analisis Korelasi / Chi-Square)
- 2) Evaluasi Berdasarkan Tujuan Bisnis

#### 4. Data Preprocessing

Mengolah data agar siap digunakan, termasuk menangani nilai hilang, normalisasi, encoding, dan membagi dataset menjadi data latih, uji, dan validasi.

#### 5. Model Training

Melatih model menggunakan data latih untuk menemukan hubungan antara fitur dan target.

### 6. Model Evaluation

Mengevaluasi performa model dengan data uji atau validasi untuk memastikan model bisa memprediksi data baru dengan baik.

### 7. Deployment

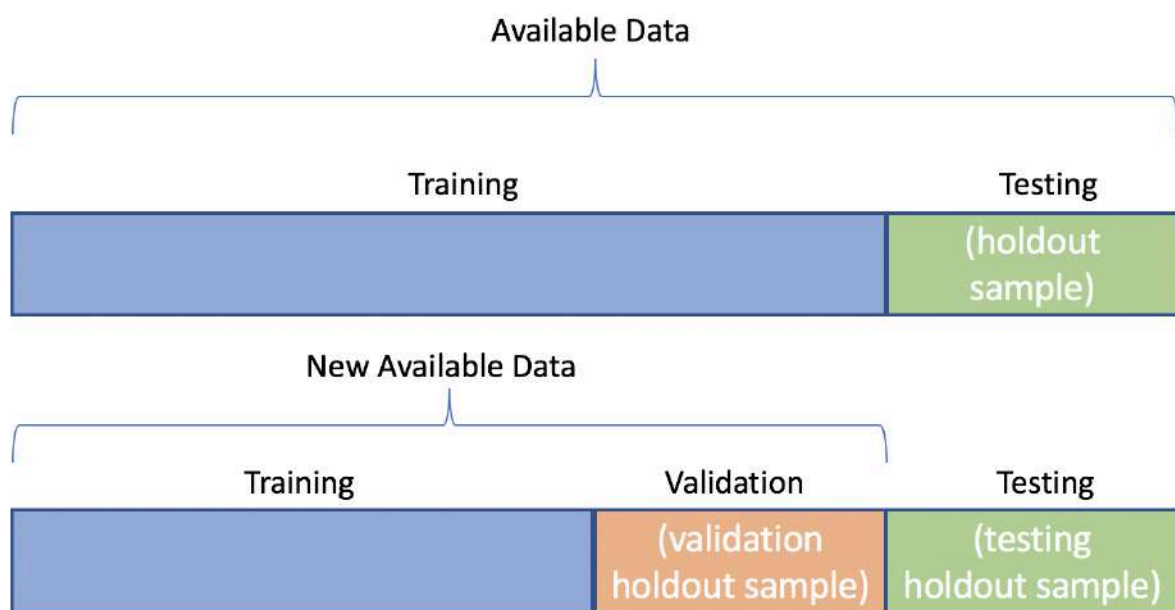
Menerapkan model ke lingkungan produksi sehingga bisa digunakan untuk prediksi oleh sistem atau pengguna.

### 8. Monitoring

Memantau kinerja model dan melakukan penyesuaian jika diperlukan agar performanya tetap optimal.

### SPLIT DATA

Sebelum melatih model, penting untuk membagi dataset menjadi beberapa bagian guna mengevaluasi kinerja model secara objektif. Tujuannya adalah memastikan model tidak hanya efektif pada data pelatihan, tetapi juga dapat memprediksi dengan baik pada data yang belum pernah dilihat sebelumnya. Tidak ada aturan khusus dalam membagi dataset namun umumnya perbandingan yang dilakukan adalah 70-80% untuk data latih, 10-20% untuk data validasi (jika digunakan), dan 10-20% untuk data uji.



#### a. Training Set (Data Latih)

Sebagian besar dataset digunakan untuk melatih model, di mana model "belajar" untuk mengidentifikasi pola dan hubungan antara variabel input dan output. Biasanya, 70-80% dari dataset dialokasikan untuk data latih.

#### b. Test Set (Data Uji)

Sekitar 20-30% dari dataset digunakan sebagai data uji. Setelah pelatihan selesai, data uji digunakan untuk mengevaluasi kemampuan model dalam membuat prediksi yang akurat pada data yang belum pernah dilihat sebelumnya. Evaluasi ini menunjukkan sejauh mana model akan bekerja di kondisi nyata.

### c. Validation Set (Opsional)

Sekitar 20-30% dari dataset digunakan sebagai data uji. Setelah pelatihan selesai, data uji digunakan untuk mengevaluasi kemampuan model dalam membuat prediksi yang akurat pada data yang belum pernah dilihat sebelumnya. Evaluasi ini menunjukkan sejauh mana model akan bekerja di kondisi nyata. Berikut adalah contoh penerapan Split Data. Jika X dan Y sudah ada, X adalah Fitur dan Y adalah target.

#### a. Split data (Train & Test) 80%-20%

```
1 from sklearn.model_selection import train_test_split
2
3 # membagi data menjadi data latih dan data uji (80% train dan 20% test)
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

#### b. Split data (Train, Test & Validation) 72 % Train, 8% Validation, 20% Test

```
1 from sklearn.model_selection import train_test_split
2
3 # membagi data menjadi data latih dan data uji (80% latih, 20% uji)
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
5
6 # membagi data latih menjadi data latih dan data validation (90% latih, 10% validasi)
7 X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=42)
```

### EVALUASI MODEL

#### A. Evaluasi Model Regresi

##### 1. Mean Squared Error (MSE)

Mean squared error merupakan metrik evaluasi yang digunakan untuk mengukur selisih kuadrat rata-rata antara nilai prediksi dan nilai target aktual dalam suatu kumpulan data. MSE menghitung selisih antara nilai prediksi model dan nilai sebenarnya dari data untuk melihat kesalahan, kemudian mengkuadratkan selisih tersebut agar tidak ada selisih yang bernilai negatif. Kemudian, selisih kuadrat dijumlahkan dan diambil rata-rata dari semua sampel data. Contoh implementasi MSE adalah sebagai berikut.

```
1 from sklearn.metrics import mean_squared_error
2
3 y_test = [3, -0.5, 2, 7]
4 y_pred = [2.5, 0.0, 2, 8]
5
6 mse = mean_squared_error(y_test, y_pred)
7 print(f"Mean Squared Error: {mse}")
```

Parameter dalam metrik evaluasi mean squared error adalah sebagai berikut.

Parameters	Descriptions
y_test	Menunjukkan label kebenaran dasar (ground truth)
y_pred	Menunjukkan label yang diprediksi, seperti yang dihasilkan dari hasil klasifikasi

##### 2. R-Squared atau R<sup>2</sup> Score

R-Squared yang disebut juga koefisien determinasi merupakan metrik evaluasi yang digunakan untuk mengukur seberapa baik data yang diprediksi sesuai dengan data sebenarnya dari sekumpulan data. Rentang skor R<sup>2</sup> berkisar antara 0 hingga 1 dengan skor yang mendekati 1 menunjukkan model yang lebih baik. Tujuan dari evaluasi ini adalah untuk mengetahui seberapa besar variasi yang dilihat dari varians



dalam data yang dapat dijelaskan oleh model. Contoh implementasi R-Squared adalah sebagai berikut.

```
1 from sklearn.metrics import r2_score
2
3 y_test = [3, -0.5, 2, 7]
4 y_pred = [2.5, 0.0, 2, 8]
5
6 r2 = r2_score(y_test, y_pred)
7 print(f"R-squared Score: {r2}")
```

Parameter dalam metrik evaluasi R-Squared adalah sebagai berikut.

Parameters	Descriptions
y_test	Menunjukkan label kebenaran dasar (ground truth)
y_pred	Menunjukkan label yang diprediksi, seperti yang dihasilkan dari hasil klasifikasi

## B. Evaluasi Model Klasifikasi

### 1. Accuracy

Accuracy adalah metrik yang mengukur seberapa sering model machine learning memprediksi hasil dengan benar. Akurasi dapat dihitung dengan membagi jumlah prediksi yang benar dengan jumlah total prediksi. Contoh implementasi untuk menghitung skor akurasi adalah sebagai berikut.

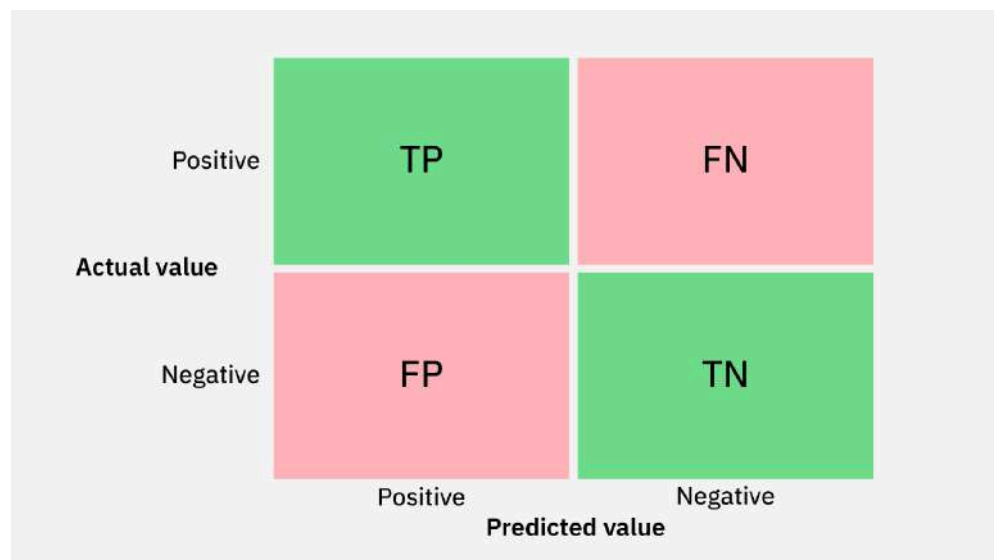
```
1 from sklearn.metrics import accuracy_score
2
3 y_test = [0, 1, 1, 0]
4 y_pred = [1, 0, 1, 0]
5
6 accuracy = accuracy_score(y_test, y_pred)
7 print(f"Accuracy: {accuracy}")
```

Parameter dalam evaluasi akurasi adalah sebagai berikut.

Parameters	Descriptions
y_test	Menunjukkan label kebenaran dasar (ground truth)
y_pred	Menunjukkan label yang diprediksi, seperti yang dihasilkan dari hasil klasifikasi
Normalize (bool)	<b>(Default=True)</b> Menentukan hasil yang diberikan dalam bentuk proporsi (persentase akurasi) atau jumlah prediksi yang benar. Jika <b>True</b> , hasil <b>skor berupa persentase antara nilai 0 hingga 1</b> , misal hasilnya 0.9 maka 90% prediksi benar. Jika <b>False</b> , hasil <b>skor berupa total angka tanpa dibagi dengan jumlah data</b> , misal model memprediksi 90 dari 100 data benar maka skornya akan 90.

## 2. Confusion Matrix

Confusion matrix atau matriks kesalahan adalah metode visualisasi untuk hasil algoritma klasifikasi berbentuk tabel. Confusion matrix akan menampilkan sebuah tabel yang merinci dari jumlah contoh ground truth dari kelas tertentu terhadap jumlah contoh kelas yang diprediksi. Matriks ini digunakan untuk mengukur kinerja model klasifikasi dalam machine learning. Gambaran kuadran dari tabel confusion matrix adalah sebagai berikut.



- True Positive (TP)  
Merupakan jumlah data yang sebenarnya positif dan berhasil diprediksi sebagai positif oleh model.
- True Negative (TN)  
Merupakan jumlah data yang sebenarnya negatif dan berhasil diprediksi sebagai negatif oleh model.
- False Positive (FP) (Type I Error)  
Terjadi ketika data sebenarnya negatif, tetapi model memprediksinya sebagai positif.
- False Negative (FN) (Type II Error)  
Terjadi ketika data sebenarnya positif, namun model memprediksinya sebagai negatif.

Contoh implementasi Confusion Matrix adalah sebagai berikut.

```
1 from sklearn.metrics import confusion_matrix
2
3 y_test = [1, 0, 1, 1, 0, 1, 0, 1]
4 y_pred = [0, 0, 1, 1, 0, 1, 1, 0]
5
6 conf_matrix = confusion_matrix(y_test, y_pred)
7 print("Confusion Matrix:")
```

Parameter dalam metrik evaluasi confusion matrix adalah sebagai berikut.

Parameters	Descriptions
y_test	Menunjukkan label kebenaran dasar (ground truth)
y_pred	Menunjukkan label yang diprediksi, seperti yang dihasilkan dari hasil klasifikasi

### 3. Classification Report

Classification report merupakan laporan evaluasi yang merangkum performa model fikasi untuk setiap kelas dari hasil klasifikasi. Beberapa metrik evaluasi yang ditampilkan pada classification report adalah precision, recall, F1-score, dan support.

**Precision** merupakan persentase untuk mengukur kemampuan pengklasifikasian untuk tidak memberi label positif pada sampel yang negatif. **Recall** merupakan persentase untuk mengukur kemampuan pengklasifikasian untuk menemukan semua sampel negatif. **F1-score** merupakan rata-rata harmonis dari precision dan recall dengan memberikan keseimbangan antara keduanya. Berikut adalah beberapa metrik evaluasi penting beserta rumusnya:

a. Precision (Presisi)

Menunjukkan seberapa banyak prediksi positif yang benar-benar positif.

$$Precision = \frac{TP}{TP + FP}$$

b. Recall (Sensitivity / True Positive Rate)

Menunjukkan seberapa baik model mengenali data positif.

$$Recall = \frac{TP}{TP + FN}$$

c. F1-Score

Merupakan rata-rata harmonik antara Precision dan Recall, berguna saat data tidak seimbang.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

d. Accuracy (Akurasi)

Mengukur proporsi prediksi yang benar terhadap seluruh data.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Contoh implementasi untuk menghasilkan classification report adalah sebagai berikut.

```
1 from sklearn.metrics import classification_report
2
3 y_test = [0, 1, 1, 0]
4 y_pred = [1, 0, 1, 0]
5
6 target_names = ['Class 0', 'Class 1']
7 class_report = classification_report(y_test, y_pred, target_names=target_names)
8 print("Classification Report:")
```

Parameter dalam classification report adalah sebagai berikut.

Parameters	Descriptions
y_test	Menunjukkan label kebenaran dasar (ground truth)
y_pred	Menunjukkan label yang diprediksi, seperti yang dihasilkan dari hasil klasifikasi
labels	Label prediksi untuk setiap sampel data
target_names	Nama dari label untuk ditampilkan secara berurutan pada classification report dan bersifat <b>opsional</b>

### C. Evaluasi Model Clustering

Hasil pemodelan data menggunakan proses clustering dapat dinilai untuk mengukur kualitas pengelompokan yang telah dibuat. Pengukuran dapat dilakukan menggunakan beberapa metode yang dibagi berdasarkan ada atau tidaknya ground truth (kebenaran dasar). Ground truth mengacu pada representasi akurat dari realitas yang mendasari atau keadaan sebenarnya dari data yang ingin dipelajari oleh sebuah model. Jika ground truth tersedia, maka dapat digunakan metode evaluasi eksternal yang membandingkan pengelompokan dengan kebenaran dan ukuran kelompok. Jika ground truth tidak tersedia, kita dapat menggunakan metode evaluasi internal yang mengevaluasi kualitas pengelompokan dengan mempertimbangkan seberapa baik pengelompokan dipisahkan. Ground truth dapat dianggap sebagai pengawasan dalam bentuk “label cluster”. Oleh



karena itu, metode evaluasi eksternal juga dikenal sebagai metode yang diawasi, sedangkan metode evaluasi internal adalah metode yang tidak diawasi.

### 1. Metriks Evaluasi Internal

Evaluasi internal merupakan proses pengukuran kualitas model clustering berdasarkan data yang ada dalam cluster tanpa memerlukan informasi label atau ground truth. Beberapa metrik evaluasi internal yang dapat digunakan untuk mengukur model clustering adalah sebagai berikut.

#### a. Silhouette Coefficient

Metrik evaluasi Silhouette Coefficient digunakan untuk mengevaluasi kualitas pengelompokan dengan mengukur seberapa baik setiap data dalam suatu kelompok dibandingkan dengan kelompok lainnya. Plot silhouette menampilkan ukuran seberapa dekat setiap titik dalam satu cluster dengan titik-titik di cluster tetangga. Skor Silhouette Coefficient berkisar antara -1 hingga 1, di mana nilai yang lebih tinggi mengindikasikan pengelompokan yang lebih baik. Jika skor Silhouette memiliki nilai mendekati +1, maka sampel jauh dari cluster tetangga. Nilai 0 menunjukkan bahwa sampel berada pada atau sangat dekat dengan batas keputusan antara dua cluster tetangga dan nilai negatif menunjukkan bahwa sampel tersebut mungkin telah dimasukkan ke cluster yang salah. Contoh implementasi untuk menghitung skor Silhouette Coefficient adalah sebagai berikut.

```
1 silhouette = silhouette_score(X_scaled, labels)
2 print(f"Silhouette Score: {silhouette}")
```

Parameter dalam evaluasi akurasi adalah sebagai berikut.

Parameters	Descriptions
X	Pasangan antara sampel data atau fitur data dan biasanya berbentuk array

labels	Label prediksi untuk setiap sampel data
--------	---

### b. Davies-Bouldin Score

Metrik evaluasi Davies-Bouldin Index digunakan untuk membandingkan kemiripan setiap kluster dengan kluster yang paling mirip berikutnya dalam dataset, yang dirata-ratakan dari semua k kluster. Metrik ini dihitung dengan menggunakan jarak rata-rata dari semua titik ke pusat di dalam setiap cluster dan jarak antara pusat untuk pasangan cluster. Nilai Davies-Bouldin yang sempurna adalah 0 yang berarti cluster-cluster terpisah dengan baik, terdefinisi dengan jelas dan padat. Tidak ada batas atas untuk metrik Davies-Bouldin, tetapi skor yang lebih tinggi berarti cluster tidak terdefinisi dengan baik dan tumpang tindih satu sama lain. Contoh implementasi untuk menghitung davies-bouldin score:

```

1 from sklearn.metrics import davies_bouldin_score
2
3 X = [[0, 1], [1,1], [3,4]]
4 labels = [0, 0, 1]
5
6 db_score = davies_bouldin_score(X, labels)
7 print(f"Davies-Bouldin Score: {db_score}")

```

Parameter dalam metrik evaluasi davies-bouldin score adalah sebagai berikut.

Parameters	Descriptions
X	Pasangan antara sampel data atau fitur data dan biasanya berbentuk array
labels	Label prediksi untuk setiap sampel data

### 2. Metrik Evaluasi Eksternal

#### a. Adjust Rand Index (ARI)

Adjusted Rand Index (ARI) adalah metrik evaluasi yang digunakan untuk mengukur kemiripan antara dua partisi dari kumpulan data dengan membandingkan penugasan atau letak titik data ke dalam cluster. Nilai ARI berkisar antara 0 hingga 1, di mana nilai 1 menunjukkan kecocokan sempurna antara partisi dan nilai yang mendekati 0 menunjukkan penugasan titik data secara acak ke dalam cluster. Contoh implementasi adjust rand index adalah sebagai berikut.

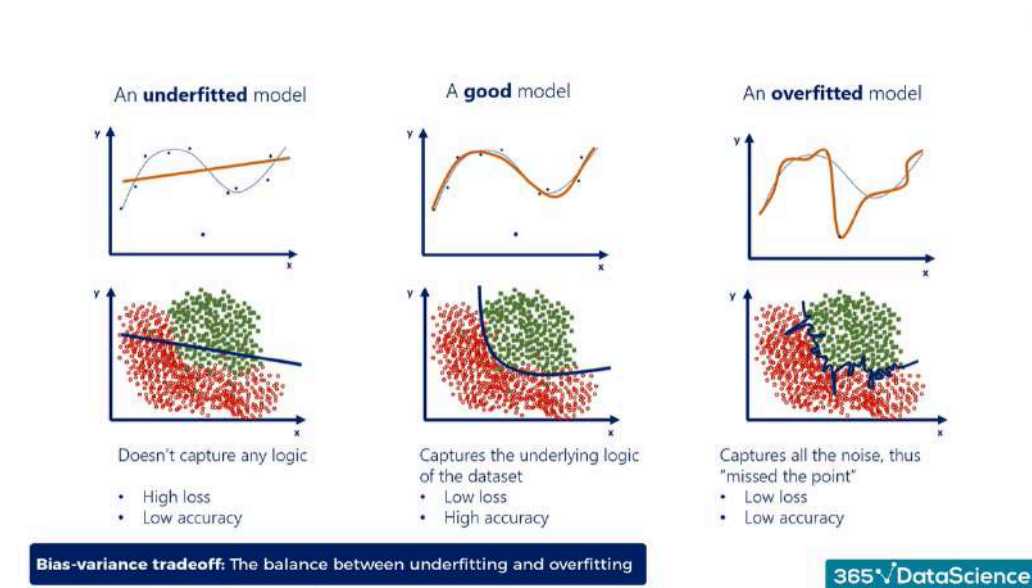
```
1 from sklearn import metrics
2
3 labels_true = [0, 0, 0, 1, 1, 1]
4 labels_pred = [0, 0, 1, 1, 2, 2]
5 rand_index = metrics.rand_score(labels_true, labels_pred)
6 print(f"Rand Index: {rand_index}")
7
8 adjust_rand = metrics.adjusted_rand_score(labels_true, labels_pred)
9 print(f"Adjusted Rand Index: {adjust_rand}")
```

Parameter dalam metrik evaluasi adjust rand index adalah sebagai berikut.

Parameters	Descriptions
labels_true	Label dari ground truth yang akan digunakan sebagai referensi
labels_pred	Label cluster yang akan dievaluasi

### OVERFITTING & UNDERFITTING

Dalam machine learning, tujuan utama adalah membuat model yang dapat memprediksi atau mengklasifikasikan data baru dengan akurat. Tantangannya adalah agar model tidak terlalu terikat pada data yang ada (overfitting) atau tidak cukup belajar dari data (underfitting). Kedua masalah ini penting untuk dipahami agar model yang dibangun dapat bekerja dengan baik pada data baru dan dapat diandalkan.



#### A. Overfitting

Overfitting adalah keadaan yang terjadi ketika model Machine Learning mempelajari data terlalu mendetail, termasuk menangkap noise yang seharusnya diabaikan. Akibatnya, model sulit melakukan generalisasi dan hanya cocok pada data pelatihan, menghasilkan akurasi tinggi saat training namun rendah pada testing. Kondisi ini mirip dengan siswa yang hanya menghafal contoh soal tanpa memahami konsep, sehingga kesulitan saat menghadapi soal baru. Overfitting disebabkan oleh kurangnya variasi data atau model yang terlalu kompleks. Untuk mengatasinya, kita bisa menambah variasi data atau mengurangi kerumitan model. Dalam mengatasi Overfitting akan dibahas dalam Tuning Optimisation.

### **B. Underfitting**

Underfitting adalah kondisi di mana model Machine Learning gagal mempelajari pola data dengan baik. Akibatnya, model tidak dapat menangkap hubungan antar variabel atau melakukan prediksi secara akurat. Pada model underfitting, akurasi rendah terjadi baik pada training maupun testing karena model tidak mampu menangkap tren data yang ada. Underfitting biasanya disebabkan oleh model yang terlalu sederhana, sehingga cenderung mengabaikan data dan membuat asumsi yang salah.

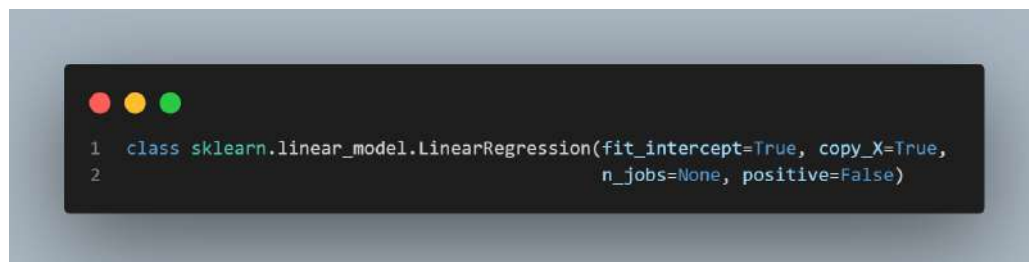


### TUNING MODEL OPTIMATION

Tuning Parameters atau Hyperparameter Tuning adalah proses mengatur hyperparameter dalam model Machine Learning untuk mencapai performa terbaik pada dataset. Hyperparameter adalah parameter yang harus diatur sebelum training dan tidak dipelajari langsung dari data, seperti jumlah pohon dalam Random Forest, jumlah tetangga pada K-Nearest Neighbors, dan learning rate dalam Neural Network. Tuning ini diperlukan untuk memastikan model bekerja secara optimal, menghindari overfitting atau underfitting, dan memaksimalkan akurasi atau metrik kinerja lainnya. Berikut contoh hyperparameters yang tersedia dari Regression dan Clustering.

#### A. Linear Regression Hyperparameters

Teknik regresi untuk memodelkan hubungan linier antara variabel input dan output dengan mencari garis terbaik yang meminimalkan error.



Hyperparameters	Description
fit_intercept	<b>(Default: True)</b> Menentukan apakah model harus menghitung nilai intercept. Jika False, model tidak akan menggunakan intercept (artinya data diharapkan sudah terpusat di sekitar nol).
copy_X	<b>(Default: True)</b> Jika True, maka salinan X akan dibuat untuk keperluan komputasi; jika False, X dapat dimodifikasi langsung, yang dapat menghemat memori pada dataset besar.
n_jobs	<b>(Default: None)</b> Jumlah prosesor yang digunakan untuk komputasi paralel. None berarti satu prosesor, sementara -1 akan menggunakan semua prosesor yang tersedia. Ini biasanya meningkatkan kecepatan pada masalah besar.
positive	<b>(Default: False)</b> Jika True, memaksa semua koefisien model positif. Opsi ini hanya didukung untuk array yang padat. Ini dapat berguna dalam beberapa kasus ketika semua fitur harus memberikan pengaruh positif.

### B. Lasso Regression

Regresi linier dengan regularisasi L1 yang mengurangi kompleksitas model dengan menyusutkan beberapa koefisien hingga nol.

```
1 class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False,
2                                   copy_X=True, max_iter=1000, tol=0.0001, warm_start=False,
3                                   positive=False, random_state=None, selection='cyclic')[source]
```

Hyperparameters	Description
alpha	<b>(Default: 1.0)</b> Mengontrol kekuatan regularisasi dalam model. Semakin tinggi nilai alpha, semakin kuat efek regularisasi untuk mencegah overfitting. Nilai ini harus lebih besar atau sama dengan 0.
fit_intercept	<b>(Default: True)</b> Menentukan apakah model harus menghitung nilai intercept. Jika True, model akan menghitung intercept; jika False, model akan mengasumsikan data sudah terpusat di sekitar nol.
max_iter	<b>(Default: 1000)</b> Menentukan jumlah maksimum iterasi yang akan digunakan oleh algoritma optimasi. Jika iterasi mencapai batas ini sebelum model selesai konvergen, proses akan berhenti.
tol	<b>(Default: 0.0001)</b> Toleransi untuk menentukan kapan optimasi harus berhenti. Jika perbaikan dalam iterasi berada di bawah nilai toleransi ini, algoritma akan berhenti.
positive	<b>(Default: False):</b> Jika True, membatasi semua koefisien model agar bernilai positif, berguna untuk konteks dimana interpretasi koefisien negatif tidak diinginkan.
selection	<b>(Default: 'cyclic')</b> Mengontrol cara pemilihan fitur selama iterasi. Opsi 'cyclic' memperbarui fitur secara berurutan, sedangkan opsi 'random' memilih fitur secara acak di setiap iterasi.

### C. Logistik Regression Hyperparameters

Model klasifikasi untuk memprediksi probabilitas kelas biner atau ganda berdasarkan hubungan logistik antara variabel input dan output.

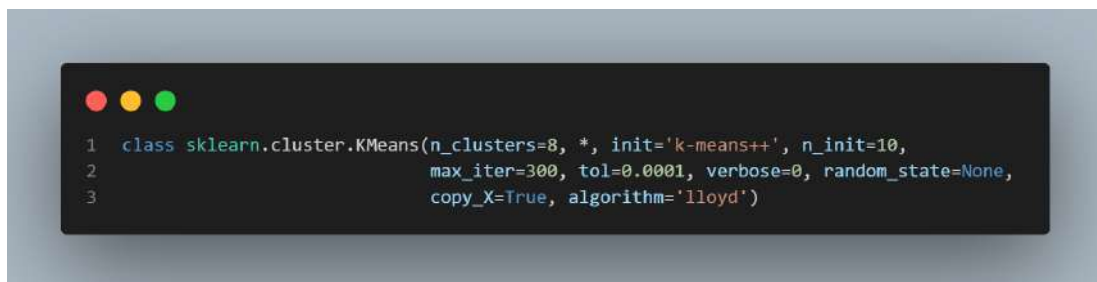
```
1 class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001,
2                                               C=1.0, fit_intercept=True, intercept_scaling=1,
3                                               class_weight=None, random_state=None, solver='lbfgs',
4                                               max_iter=100, multi_class='deprecated', verbose=0,
5                                               warm_start=False, n_jobs=None, l1_ratio=None)
```

Hyperparameters	Description
penalty	<p><b>(Default: "l2")</b> Menentukan jenis regularisasi yang digunakan dalam model.</p> <p>"l2": Regularisasi berdasarkan norma kuadrat (default), berguna untuk mengurangi overfitting.</p> <p>"l1": Regularisasi berdasarkan norma absolut, menghasilkan koefisien sparse (beberapa koefisien menjadi nol).</p> <p>"elasticnet": Kombinasi dari l1 dan l2, hanya digunakan dengan solver saga.</p>
dual	<p><b>(Default: False)</b> Menentukan apakah menggunakan formulasi dual untuk optimasi, dengan nilai True: Menggunakan formulasi dual, lebih efisien untuk data dengan lebih banyak fitur daripada sampel, dan nilai False: Menggunakan formulasi primal, lebih cocok untuk data dengan lebih banyak sampel daripada fitur.</p>
tol	<p><b>(Default: 0.0001)</b> Toleransi untuk menentukan kapan optimasi harus berhenti. Jika perbaikan dalam iterasi berada di bawah nilai toleransi ini, algoritma akan berhenti.</p>
c	<p><b>(Default: 1.0)</b> Kebalikan dari kekuatan regularisasi, semakin kecil nilai C, semakin kuat regularisasi. Jika Nilai lebih kecil dari C berarti regularisasi yang lebih kuat. Jika lebih besar memungkinkan model lebih bebas, berisiko overfitting.</p>
fit_intercept	<p><b>(Default: True)</b> Menentukan apakah akan menambahkan intercept (konstanta) dalam model, dengan value True: Menambahkan intercept (konstanta), dan False: Tidak menambahkan intercept, model hanya menggunakan fitur.</p>
solver	<p><b>(Default: "lbfgs")</b> Menentukan algoritma yang digunakan untuk optimasi.</p> <ul style="list-style-type: none"> <li>"lbfgs": Metode optimasi berbasis kuadrat, cocok untuk data besar dengan regularisasi l2.</li> <li>"liblinear": Cocok untuk data kecil, mendukung regularisasi l1 dan l2.</li> <li>"newton-cg": Solver berbasis metode Newton, lebih efisien</li> </ul>

	<p>untuk data besar.</p> <ul style="list-style-type: none"> <li>• "sag": Metode Stochastic Average Gradient, cocok untuk data besar.</li> <li>• "saga": Versi lebih cepat dari sag, mendukung regularisasi l1, l2, dan elasticnet</li> </ul>
max_iter	<b>(Default: 100)</b> Menentukan jumlah iterasi maksimum untuk konvergensi. Nilai lebih tinggi memberi lebih banyak kesempatan untuk mencapai konvergensi, namun meningkatkan waktu komputasi.
multi_class	<b>(Default: "auto")</b> Menentukan pendekatan klasifikasi untuk multi-kelas. <ul style="list-style-type: none"> <li>• "auto": Secara otomatis memilih ovr (one-vs-rest) untuk dua kelas dan multinomial untuk lebih dari dua kelas.</li> <li>• "ovr": One-vs-rest, membuat model untuk setiap kelas.</li> <li>• "multinomial": Mempertimbangkan semua kelas sekaligus, lebih akurat untuk multi-kelas.</li> </ul>
warm_start	<b>(Default: False)</b> Jika True, solusi dari panggilan fit sebelumnya digunakan sebagai inisialisasi.
n_jobs	<b>(Default: None)</b> Menentukan jumlah prosesor untuk klasifikasi paralel saat multi_class='ovr'

### D. K-Means Clustering

Algoritma clustering yang membagi data ke dalam sejumlah cluster berdasarkan kedekatan centroid.

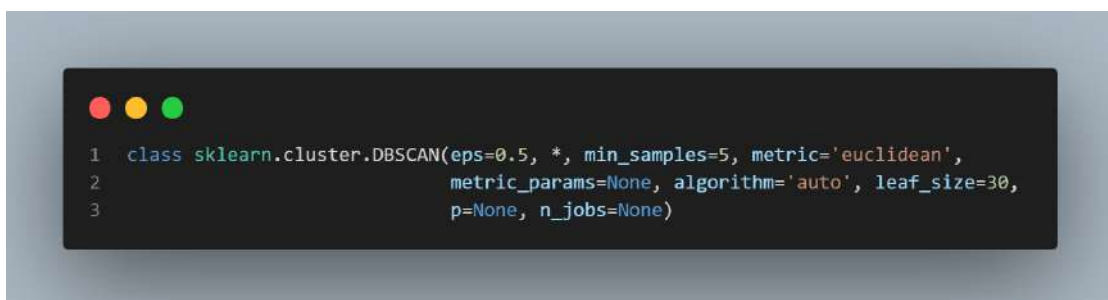


Hyperparameters	Description
n_clusters	<b>(Default : 8)</b> Jumlah cluster yang akan dibentuk dan jumlah centroid yang dihasilkan.
init	<b>(Default : 'k-means++')</b> Metode untuk inisialisasi centroid awal. 'k-means++' menggunakan distribusi probabilitas untuk mempercepat konvergensi, sedangkan 'random' memilih observasi acak dari data.
n_init	<b>(Default : 'auto')</b> Jumlah kali algoritma K-Means dijalankan

	dengan centroid seed berbeda. Hasil akhir adalah keluaran terbaik dari beberapa run ini dalam hal inertia. Ketika <code>n_init='auto'</code> , jumlah run bergantung pada <code>init</code> : 10 kali jika <code>init='random'</code> atau <code>init callable</code> , 1 kali jika <code>init='k-means++'</code> atau <code>init</code> berupa array.
<code>max_iter</code>	<b>(Default : 300)</b> Jumlah iterasi maksimum algoritma K-Means untuk satu run.
<code>tol</code>	<b>(Default : 0.0001)</b> Toleransi relatif terhadap norma Frobenius dari perbedaan pusat cluster dalam dua iterasi berturut-turut untuk menentukan konvergensi.
<code>verbose</code>	<b>(Default : 0)</b> Mode verbosity. Mengontrol tingkat informasi yang ditampilkan selama proses fitting.
<code>random_state</code>	<b>(Default : none)</b> Menentukan pengacakan untuk inisialisasi centroid. Menggunakan nilai <code>int</code> agar hasil acak menjadi deterministik.
<code>copy_x</code>	<b>(Default : True)</b> Jika <code>True</code> , data asli tidak diubah. Jika <code>False</code> , data asli dapat dimodifikasi untuk perhitungan jarak, dan dikembalikan ke bentuk semula setelah fungsi selesai dijalankan, tetapi perbedaan numerik kecil mungkin terjadi.
<code>algorithm</code>	<b>(Default : 'lloyd')</b> Algoritma K-Means yang digunakan. 'lloyd' adalah algoritma gaya EM klasik, sedangkan 'elkan' lebih efisien pada beberapa dataset dengan cluster yang jelas, tetapi memerlukan lebih banyak memori.

### E. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Algoritma clustering yang mengelompokkan data berdasarkan kepadatan dan dapat menangani bentuk cluster yang kompleks serta mengabaikan outlier.



Hyperparameters	Description
<code>eps</code>	<b>(Default : 0.5)</b> Jarak maksimum antara dua sampel agar dapat dianggap dalam satu lingkungan. Nilai ini sangat penting untuk disesuaikan dengan dataset dan fungsi jarak yang dipilih.
<code>min_samples</code>	<b>(Default: 5)</b> Jumlah sampel minimum dalam lingkungan untuk



	suatu titik dianggap sebagai titik inti. Semakin tinggi nilai <code>min_samples</code> , semakin padat cluster yang akan ditemukan.
<code>metric</code>	<b>(Default: 'euclidean')</b> Fungsi jarak untuk mengukur jarak antar-titik. Dapat berupa string atau callable. Jika "precomputed" digunakan, X harus berupa matriks jarak berbentuk kotak.
<code>algorithm</code>	<b>(Default: 'auto')</b> Algoritma untuk menghitung jarak titik. Pilihan termasuk 'ball_tree', 'kd_tree', dan 'brute'.
<code>leaf_size</code>	<b>(Default: 30)</b> Ukuran leaf yang diberikan untuk BallTree atau cKDTree.
<code>n_jobs</code>	<b>(Default: none)</b> Jumlah pekerjaan paralel. -1 berarti menggunakan semua prosesor.
<code>p</code>	<b>(Default: none)</b> Kekuatan metrik Minkowski yang akan digunakan untuk menghitung jarak antar titik. Jika Tidak Ada, maka $p=2$ (setara dengan jarak Euclidean).

### F. Hierarchical Clustering

Metode clustering yang membangun hierarki cluster secara bertahap, menggabungkan atau memisahkan data untuk membentuk dendrogram.

```

1 class sklearn.cluster.AgglomerativeClustering(n_clusters=2, *, metric='euclidean',
2                                               memory=None, connectivity=None,
3                                               compute_full_tree='auto', linkage='ward',
4                                               distance_threshold=None, compute_distances=False)

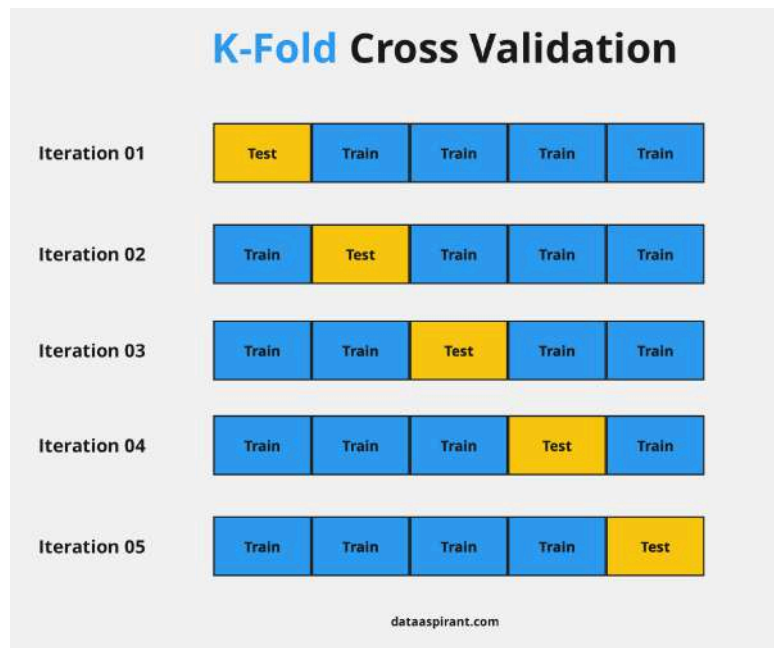
```

Hyperparameters	Description
<code>n_clusters</code>	<b>(Default: 2)</b> Menentukan jumlah cluster yang ingin ditemukan. Jika <code>distance_threshold</code> ditentukan, maka <code>n_clusters</code> harus bernilai None.
<code>metric</code>	<b>(Default = "euclidean")</b> Menghitung linkage dengan menggunakan metrik jarak tertentu, seperti "euclidean", "l1", "l2", "manhattan", "cosine", atau "precomputed".
<code>memory</code>	<b>(Default = None)</b> Menyimpan hasil komputasi sementara untuk mempercepat eksekusi. Jika diisi dengan path, hasil akan disimpan pada direktori tersebut.
<code>connectivity</code>	<b>(Default = None)</b> Matriks konektivitas untuk menentukan struktur tetangga antar data dalam algoritma clustering.
<code>compute_full_tree</code>	<b>(Default = 'auto')</b> Menghentikan pembangunan tree pada

	n_clusters untuk efisiensi waktu. Berguna jika connectivity disediakan. Value yang dapat diisi adalah nilai bool.
linkage	<b>(Default = 'ward')</b> Menentukan kriteria linkage antara cluster: <ul style="list-style-type: none"><li>• 'ward' meminimalkan variansi antar cluster.</li><li>• 'average' menghitung rata-rata jarak.</li><li>• 'complete' menggunakan jarak maksimum.</li><li>• 'single' menggunakan jarak minimum.</li></ul>
distance_treshold	<b>(Default: None)</b> Menentukan jarak linkage minimal untuk menghentikan penggabungan cluster. Jika diisi, n_clusters harus None, Value yang dapat diisi adalah float
compute_distances	<b>(Default: False)</b> Menghitung jarak antar cluster meskipun distance_threshold tidak diatur. Digunakan untuk visualisasi dendrogram. Value yang dapat diisi adalah nilai bool

### K-FOLD CROSS VALIDATION

#### A. Definisi K-Fold Cross Validation



*K-Fold Cross Validation* adalah teknik validasi yang membagi dataset menjadi beberapa subset atau fold yang sama besar. Model dilatih dan dievaluasi  $k$  kali, masing-masing menggunakan satu *fold* sebagai data validasi dan sisanya sebagai data latih. Skor akhir adalah rata-rata dari semua evaluasi pada setiap *fold*. Teknik ini penting untuk mendapatkan evaluasi model yang lebih andal, mengurangi bias dan variance, serta memastikan performa model yang konsisten.

**Mengapa K-Fold Penting:** K-Fold membantu kita menghindari bias yang bisa terjadi bila model hanya diuji sekali pada satu subset data. **K-Fold** juga membantu dalam mengatasi permasalahan overfitting, selain itu, K-Fold memberikan gambaran performa model yang lebih stabil di seluruh dataset dan sangat berguna saat dataset terbatas.

**Rasio Pembagian K-Folds,** data dibagi menjadi  $K$  bagian (folds). Pada setiap iterasi, 1 fold digunakan untuk pengujian (testing), dan  $K-1$  fold digunakan untuk pelatihan (training). Rasio training dan testing tergantung pada nilai  $K$ :

- $K = 4$ : 75% training, 25% testing (3 fold untuk training, 1 fold untuk testing).
- $K = 5$ : 80% training, 20% testing (4 fold untuk training, 1 fold untuk testing).
- $K = 10$ : 90% training, 10% testing (9 fold untuk training, 1 fold untuk testing).

*hyperparameter yang ada dalam K-Fold.*

<i>Hyperparameter</i>	<i>Description</i>
<b>n_splits</b> (int, default = 5)	<b>Jumlah fold (lipatan)</b> yang digunakan dalam K-Fold cross-validation. Harus lebih besar atau sama dengan 2.
<b>shuffle</b> (bool, default=False)	Menentukan apakah data akan diacak sebelum dibagi menjadi batch (fold). <b>Jika True, data akan diacak terlebih dahulu sebelum pembagian.</b>
<b>random_state</b> (int, default=None)	Ketika shuffle=True, random_state mengontrol urutan indeks data, yang menentukan acak atau tidaknya pembagian fold. <b>Pengaturan nilai random_state dengan integer akan memberikan hasil yang konsisten pada pemanggilan yang berbeda.</b>

### B. Implementasi K-Fold

#### a. Import Library

```
1 from sklearn.datasets import make_classification # untuk dataset
2 from sklearn.model_selection import KFold, cross_val_score, train_test_split # untuk K-Fold dan train_test_split
3 from sklearn.linear_model import LogisticRegression # untuk model
4 from sklearn.metrics import accuracy_score
```

#### b. Load Dataset & Model (dataset contoh)

```
1 # buat dataset contoh dan inisiasi
2 X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
3 model = LogisticRegression(max_iter=200)
```

#### c. Akurasi tanpa K-Fold

```
1 # Tanpa K-Fold
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # membagi data menjadi training dan testing
3 model.fit(X_train, y_train)
4
5 y_pred = model.predict(X_test)
6 accuracy_no_fold = accuracy_score(y_test, y_pred)
```

### d. K-Fold Implementation

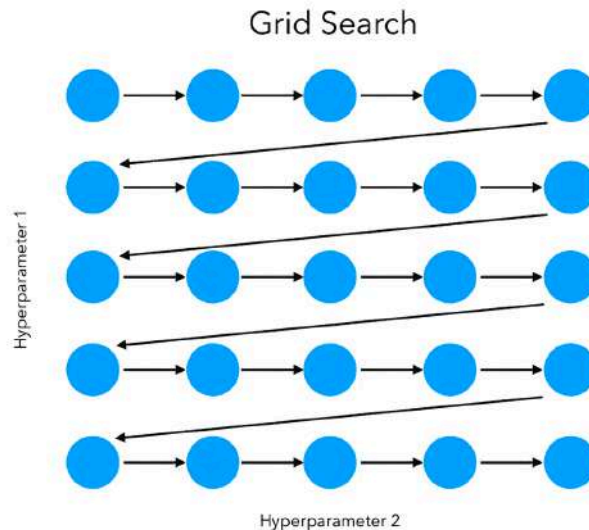
```
1 # pakai K-Folds
2 k = 5 # K-Fold dengan 5 fold
3 kf = KFold(n_splits=10, shuffle=True, random_state=42)
4 accuracy_with_fold = cross_val_score(model, X, y, cv=kf) # hitung akurasi K-Folds
5
6 print("Akurasi tanpa K-Fold:", accuracy_no_fold)
7 print("Akurasi tiap fold:", accuracy_with_fold)
8 print("Akurasi rata-rata dengan K-Fold:", accuracy_with_fold.mean())
```

### e. Perbandingan Hasil

```
Akurasi tanpa K-Fold: 0.855
Akurasi tiap fold: [0.87 0.86 0.83 0.89 0.88 0.86 0.85 0.87 0.9 0.86]
Akurasi rata-rata dengan K-Fold: 0.867
```

### GRIDSEARCHCV

#### A. Definisi GridSearchCV



GridSearchCV adalah teknik dalam machine learning untuk mencari kombinasi terbaik dari hyperparameter model secara otomatis. GridSearchCV mencoba berbagai kombinasi dari hyperparameter yang diberikan dalam grid pencarian (daftar nilai-nilai parameter) dan melakukan cross-validation untuk memilih kombinasi terbaik yang menghasilkan performa model yang optimal. Dengan cara ini, GridSearchCV membantu menemukan konfigurasi parameter yang menghasilkan akurasi atau performa terbaik pada model. Berikut adalah **parameter dalam GridSearchCV**:

<i>Hyperparameter</i>	<i>Description</i>
<b>Estimator</b> (Required)	Model yang ingin dipakai, misalnya <code>svm.SVC()</code> untuk SVM.
<b>param_grid</b> (Required)	Dictionary yang berisi nama parameter sebagai key dan list nilai parameter sebagai value.
<b>Scoring</b>	Cara untuk mengevaluasi performa model. Bisa berupa string (misalnya <b>'accuracy'</b> , <b>'roc_auc'</b> ), callable function, atau dict untuk beberapa metrik.
<b>n_jobs</b>	Jumlah pekerjaan yang dijalankan secara



	paralel. Nilai -1 berarti menggunakan semua core yang tersedia.
<b>refit</b>	Menentukan apakah model harus di-fit ulang menggunakan parameter terbaik setelah pencarian. Biasanya di-set ke <b>True</b>
<b>cv</b>	Strategi cross-validation. Bisa berupa integer (jumlah fold), objek pembagi cross-validation, atau None ( <b>default 5-fold</b> ).
<b>verbose</b>	Mengontrol tingkat verbositas. Nilai yang lebih tinggi akan memberikan lebih banyak informasi selama proses pencarian. <b>value 0-2</b> .
<b>pre_dispatch</b>	Mengontrol jumlah pekerjaan yang dikerjakan secara paralel. Ini berguna untuk mencegah penggunaan memori berlebihan.
<b>error_score</b>	Nilai yang akan diberikan jika ada error saat fitting model. Bisa berupa nilai numerik atau 'raise' untuk melemparkan error.
<b>return_train_score</b>	Jika True, akan mengembalikan skor untuk data pelatihan dalam cv_results_. Defaultnya adalah False.

### B. Implementasi GridSearchCV pada Regression

#### a. Import Library

```
1 from sklearn.datasets import make_classification # untuk dataset
2 from sklearn.model_selection import GridSearchCV, train_test_split # untuk GridSearch dan train_test_split
3 from sklearn.linear_model import LogisticRegression # untuk model
4 from sklearn.metrics import accuracy_score # untuk akurasi
```

#### b. Load Dataset

```
1 # membuat data dummy untuk contoh
2 X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=42)
```

### c. Split Data dan Load Model

```
1 # membagi data menjadi training dan testing 0.8-0.2 dan Load Model
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
3 model = LogisticRegression()
```

### d. Param-Grid

```
1 # mendefinisikan parameter yang akan di-tune
2 param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], # parameter regulasi
3               'penalty': ['l1', 'l2'], # jenis regulasi
4               'solver': ['liblinear', 'saga'], # algoritma optimasi
5               'max_iter': [100, 200, 300], # jumlah iterasi maksimum
6               'tol': [1e-3, 1e-4, 1e-5], # toleransi untuk kriteria penghentian
7               'class_weight': [None, 'balanced'], # bobot kelas
8               'warm_start': [True, False], # inisiasi solusi sebelumnya
9               'random_state': [42, None] # random seed
10            }
```

### e. Grid-Search

```
1 # membuat objek GridSearchCV
2 grid_search = GridSearchCV(
3     estimator=model, # model yang digunakan
4     param_grid=param_grid, # parameter yang akan di-tune
5     cv=5, # jumlah fold dalam cross-validation
6     scoring='accuracy', # metrik evaluasi
7     n_jobs=-1 # menggunakan semua core CPU
8 )
```

### f. Fit Model

```
1 # melakukan fitting model dengan GridSearchCV
2 grid_search.fit(X_train, y_train)
3 y_pred = grid_search.predict(X_test)
```

### g. Hasil

```
1 # menampilkan parameter terbaik
2 print("Parameter terbaik:", grid_search.best_params_)
3 print("Skor akurasi terbaik:", grid_search.best_score_)
4 print("Akurasi pada prediksi:", accuracy_score(y_test, y_pred))
```

- Parameter terbaik: {'C': 0.01, 'class\_weight': None, 'max\_iter': 100, 'penalty': 'l2', 'random\_state': 42, 'solver': 'saga', 'tol': 0.0001, 'warm\_start': True}
- Skor akurasi terbaik: 0.80625
- Akurasi pada prediksi: 0.83

## C. Implementasi GridSearchCV pada Clustering

### a. Import Library

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import make_blobs
4 from sklearn.cluster import KMeans
5 from sklearn.model_selection import GridSearchCV
6 from sklearn.metrics import silhouette_score
```

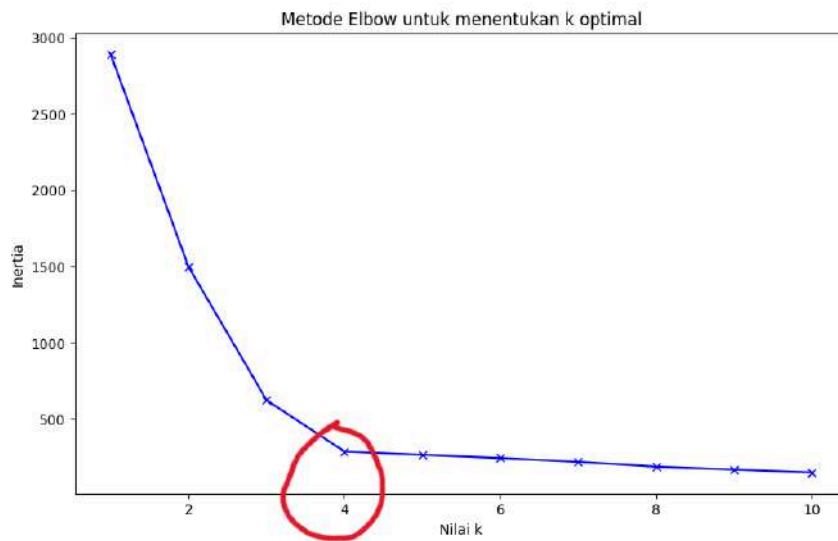
### b. Load Dataset

```
1 # membuat dataset contoh
2 X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.7, random_state=0)
```

### c. Implementasi Elbow Methods

```
1 # implementasi Elbow method
2 inertias = []
3 K = range(1, 11)
4
5 for k in K:
6     k_means = KMeans(n_clusters=k, random_state=0)
7     k_means.fit(X)
8     inertias.append(k_means.inertia_)
```

```
1 # visualisai Elbow Method
2 plt.figure(figsize=(10, 6))
3 plt.plot(K, inertias, 'bx-')
4 plt.xlabel('Nilai k')
5 plt.ylabel('Inertia')
6 plt.title('Metode Elbow untuk menentukan k optimal')
7 plt.show()
```



### d. Inisiasi Model

```
1 # inisiasi model KMeans
2 k_means = KMeans(random_state=42)
```

### e. Param-Grid & Grid Search

```
1 # tentukan parameter grid
2 param_grid = {
3     'n_clusters': range(3, 8), # nilai yang dicoba untuk jumlah cluster
4     'init': ['k-means++', 'random'],
5     'n_init': [10, 20, 30],
6     'max_iter': [100, 200, 300, 400, 500]
7 }
8
9 # GridSearchCV untuk KMeans menggunakan scoring dengan inertia sebagai acuan
10 grid_search = GridSearchCV(estimator=k_means, param_grid=param_grid, refit=False, cv=10)
11 grid_search.fit(X)
```

### f.

### g. Get Best Result

```
1 best_score = -1
2 best_params = None
3
4 for params in grid_search.cv_results_['params']:
5     model = KMeans(**params)
6     cluster_labels = model.fit_predict(X)
7     score = silhouette_score(X, cluster_labels)
8     if score > best_score:
9         best_score = score
10        best_params = params
11
12 best_k = best_params['n_clusters']
```

### h. Hasil

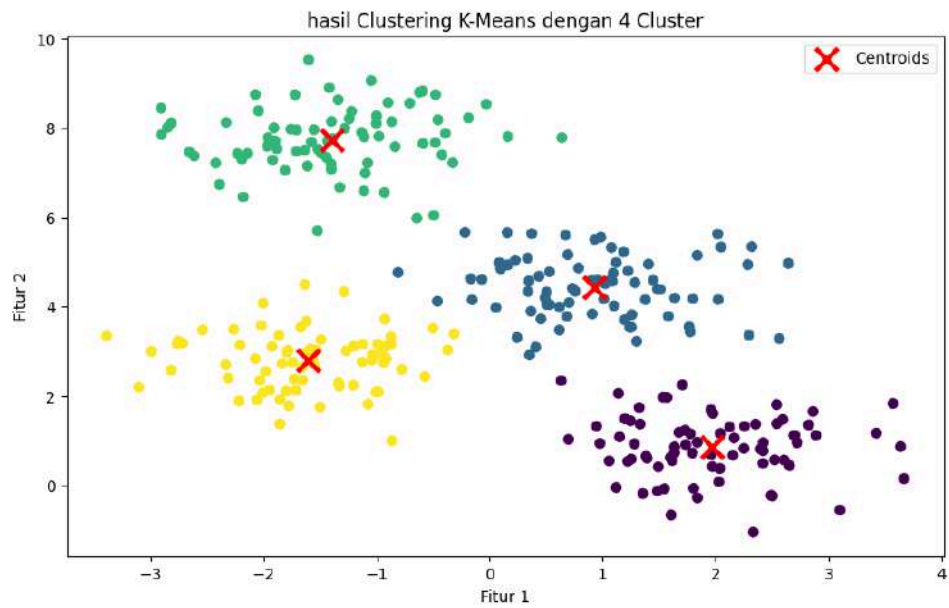
```
1 print("Best parameters found: ", best_params)
2 print("Best silhouette score: ", best_score)
```

```
Best parameters found: {'init': 'k-means++', 'max_iter': 100, 'n_clusters': 4, 'n_init': 10}
Best silhouette score: 0.6290184147089205
```

### i. Visualisasi

```
1 # visualisasi hasil clustering menggunakan best_k dari silhouette score
2 kmeans_final = KMeans(n_clusters=best_k, random_state=42)
3 cluster_labels = kmeans_final.fit_predict(X)
4
5 # membuat visualisasi
6 plt.figure(figsize=(10,6))
7 plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis')
8 plt.scatter(kmeans_final.cluster_centers[:, 0], kmeans_final.cluster_centers[:, 1], marker='x', s=200,
9             linewidths=3, color='r', label='Centroids')
10 plt.title(f"hasil Clustering K-Means dengan {best_k} Cluster")
11 plt.xlabel('Fitur 1')
12 plt.ylabel('Fitur 2')
13 plt.legend()
14 plt.show()
```





### REGRESI

Regresi adalah pendekatan statistik yang digunakan untuk menganalisis hubungan antara variabel dependen (variabel target) dan satu atau lebih variabel independen (variabel prediktor). Tujuannya adalah untuk menentukan fungsi yang paling sesuai yang mencirikan hubungan antara variabel-variabel ini. Dalam konteks Machine Learning, Regresi adalah teknik pembelajaran mesin terbimbing, yang digunakan untuk memprediksi nilai variabel dependen untuk data baru yang tak terlihat. Teknik ini memodelkan hubungan antara fitur input dan variabel target, yang memungkinkan estimasi atau prediksi nilai numerik. Untuk memulai implementasi model pada regresi, kita akan menggunakan sebuah dataset yang berisi informasi kesehatan dari sejumlah pasien. Dataset ini mencakup berbagai variabel kesehatan dan gaya hidup dari pasien.

Penjelasan variable pada :

- PatientID: Pengidentifikasi unik untuk setiap pasien.
- BMI: Indeks Massa Tubuh, dihitung dari tinggi dan berat badan.
- HadArthritis: Indikator apakah pasien menderita arthritis.
- HadDiabetes: Indikator apakah pasien menderita diabetes.
- BloodSugarLevel: Kadar gula darah pasien.

#### A. Analisis Statistik (Statistical Analysis)

Analisis statistik digunakan untuk memahami hubungan antara variabel dalam dataset dan mengukur kekuatan serta signifikansi hubungan tersebut. Analisis Statistik penting untuk mengevaluasi model dan menentukan fitur yang berpengaruh signifikan terhadap prediksi. Analisis Statistik digunakan setelah model dilatih.

- Koefisien menunjukkan seberapa besar perubahan pada variabel target jika fitur berubah. Jika koefisien positif, artinya peningkatan pada fitur akan meningkatkan prediksi target. Sebaliknya, koefisien negatif berarti peningkatan pada fitur akan menurunkan prediksi target. Semakin besar nilai koefisien, semakin kuat pengaruh fitur terhadap target.

- Intercept adalah nilai prediksi model ketika semua fitur bernilai nol. Untuk membaca intercept, jika semua fitur tidak ada pengaruhnya (nilai nol), maka prediksi model akan bernilai sebesar intercept. Ini adalah titik dasar sebelum fitur mempengaruhi prediksi.
- P-value digunakan untuk mengukur apakah hubungan antara fitur dan target signifikan. Jika p-value lebih kecil dari 0.05 (nilai signifikansi umum), maka fitur dianggap berpengaruh signifikan terhadap model. Sebaliknya, jika p-value lebih besar dari 0.05, fitur tersebut dianggap tidak signifikan, yang berarti tidak ada bukti kuat bahwa fitur itu berpengaruh terhadap target.

### Implementasi Analisis Statistik

Pada implementasi Analisis Statistik menggunakan data dummy agar mempermudah pemahaman terhadap data.

```
1 import numpy as np
2 from sklearn.linear_model import LinearRegression
3 import statsmodels.api as sm
4
5 # Data dummy dengan beberapa fitur jam belajar, jam tidur, dan waktu ngepush ML
6 X= np.array([[2, 5, 4.8],
7             [3.1, 6.2, 3],
8             [4, 7, 3],
9             [5.5, 8.9, 2.8],
10            [6, 9, 1.6]])
11
12 # Jika menggunakan Logistic Regression untuk y gunakan value binary (1-0)
13 y = np.array([2, 3, 4, 5, 6])
14
15 # Inisialisasi dan latih model regresi linier
16 model = LinearRegression()
17 model.fit(X, y)
18
19 # Mengambil nilai intercept dan koefisien
20 intercept = model.intercept_
21 coefficients = model.coef_
22
23 # Mengambil nilai p-value dari model
24 X_const = sm.add_constant(X)
25 model_sm = sm.OLS(y, X_const)
26 results = model_sm.fit() # bisa memakai results.summary() untuk menampilkan ringkasan hasil
27
28 print("Intercept:", intercept)
29 print("Koefisien:", coefficients)
30 print("P-Values:", results.pvalues)
```

Output:

```
Intercept: 3.7457234230920187  
Koefisien: [ 2.1825283 -1.2006977 -0.02259955]  
P-Values: [0.00593635 0.0046289 0.00731129 0.11582821]
```

### B. Linear Regression

Regresi linear memperkirakan hubungan antara variabel target kontinu dan satu atau lebih variabel independen dengan asumsi adanya hubungan linear di antara mereka. Model ini digunakan untuk memprediksi nilai kontinu, seperti tinggi, berat, BMI, atau kadar gula darah pasien, berdasarkan variabel yang memengaruhinya. Regresi linear menghasilkan persamaan yang memungkinkan prediksi nilai target pada data baru, tanpa batasan pada rentang prediksi. Sebaliknya, regresi logistik digunakan untuk klasifikasi dengan hasil dalam bentuk probabilitas antara 0 dan 1.

**Rumus :  $Y = a + bX$**

**Y:** variabel dependen atau variabel yang ingin diprediksi

**X:** variabel independen atau variabel yang digunakan untuk memprediksi Y

**a:** intercept atau nilai konstan ketika  $X = 0$

**b:** slope atau kemiringan garis regresi

### Kenapa Menggunakan Linear Regression

Regresi linear cocok untuk masalah prediksi dengan hasil kontinu, di mana kita ingin mengetahui hubungan linear antara variabel target dan prediktor. Model ini mudah diinterpretasikan, karena menghasilkan persamaan linear yang menggambarkan hubungan antara target dan prediktor. Regresi linear juga efisien secara komputasi, sehingga ideal untuk digunakan pada dataset dengan ukuran sedang hingga besar. Model ini sering digunakan sebagai langkah awal dalam pemodelan data kontinu karena kesederhanaan dan kejelasannya.

### Implementasi Code

**Import Library yang akan digunakan:** Langkah pertama adalah mengimport library yang dibutuhkan untuk mengimplementasikan model regresi linear. Library ini mencakup algoritma regresi linear dari **scikit-learn** dan fungsi-fungsi evaluasi yang diperlukan.

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.metrics import mean_squared_error
3 from sklearn.model_selection import train_test_split
4 import math
5 import matplotlib.pyplot as plt
```

- **LinearRegression:** Untuk membuat dan melatih model regresi linear.
- **mean\_squared\_error:** Untuk mengevaluasi kinerja model dengan menghitung rata-rata kuadrat kesalahan (MSE).
- **train\_test\_split:** Untuk membagi dataset menjadi data latih dan data uji.
- **matplotlib.pyplot:** Untuk memvisualisasikan hasil prediksi dengan grafik scatterplot.

**Persiapan Data:** Pisahkan variabel independen (fitur) dan variabel dependen (label). Variabel independen adalah semua kolom yang akan digunakan untuk memprediksi kadar gula darah, sementara variabel dependen adalah kolom BloodSugarLevel yang menjadi target prediksi.

```
1 X = df_num.drop(['BloodSugarLevel', 'PatientID'], axis=1)
2 y = df_num['BloodSugarLevel']
```

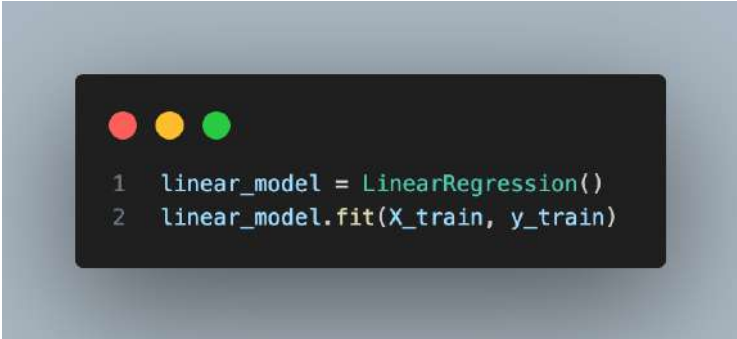
**Pembagian Data:** Bagi data menjadi data latih dan data uji untuk melatih model dan mengukur akurasi model pada data yang tidak terlihat sebelumnya.



```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **test\_size=0.2:** Menentukan bahwa 20% dari data akan digunakan sebagai data uji, sementara 80% sebagai data latih.
- **random\_state=12:** Menetapkan nilai acak agar hasil pembagian data tetap konsisten setiap kali kode dijalankan.

**Pelatihan Model:** Latih model regresi linear menggunakan data train. Model akan menghitung koefisien yang sesuai untuk setiap fitur agar dapat memprediksi kadar gula darah sebaik mungkin berdasarkan data train



```
1 linear_model = LinearRegression()
2 linear_model.fit(X_train, y_train)
```

- **linear\_model.fit():** Melatih model untuk menemukan hubungan antara variabel independen dan variabel dependen dalam data latih.

**Prediksi:** Setelah model dilatih, gunakan model untuk memprediksi kadar gula darah pada data uji.





```
1 y_pred = linear_model.predict(X_test)
```

- **y\_pred**: Berisi prediksi kadar gula darah untuk data uji ( $X_{\text{test}}$ ) berdasarkan model yang sudah dilatih.

**Intercept dan Koefisien** : Setelah melatih model, kita dapat memperoleh intercept (titik potong garis regresi dengan sumbu Y) dan koefisien (pengaruh setiap variabel input terhadap hasil prediksi).



```
1 intercept = linear_model.intercept_[0] # Mendapatkan nilai intercept dari model
2 coefficients = linear_model.coef_[0]   # Mendapatkan koefisien untuk setiap fitur
3
4 print("Intercept:", intercept) # Menampilkan nilai intercept
5 print("Koefisien:", coefficients) # Menampilkan koefisien setiap fitur
```

- **Intercept ()**: Konstanta dalam model, menunjukkan prediksi ketika semua fitur bernilai nol.
- **Koefisien ()**: Pengaruh masing-masing fitur terhadap prediksi. Koefisien positif meningkatkan prediksi, sedangkan negatif menurunkannya.

**p-value** : digunakan untuk menguji signifikansi koefisien.



```
1 X_train_const = sm.add_constant(X_train) # Menambahkan konstanta (intercept) pada fitur input
2 ols_model = sm.OLS(y_train, X_train_const) # Membuat model regresi linier (OLS)
3 result = ols_model.fit() # Melatih model dengan data latih
```

- **p-value**: Menunjukkan signifikansi koefisien. Jika  $p\text{-value} < 0.05$ , koefisien signifikan; jika  $p\text{-value} > 0.05$ , koefisien tidak signifikan.

### C. Logistic Regression

#### Apa itu Logistic Regression

Regresi logistik memperkirakan probabilitas terjadinya suatu peristiwa berdasarkan variabel independen dan digunakan untuk prediksi hasil biner, seperti "Ya/Tidak" atau "1/0." Berbeda dengan regresi linear yang dirancang untuk prediksi nilai kontinu, regresi logistik menggunakan fungsi sigmoid untuk membatasi prediksi dalam rentang 0 hingga 1, memastikan hasil probabilitas yang valid bagi data biner.

**Formula :**  $\ln(p/1-p) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n$

- $\ln(p/1-p)$ : logit dari variabel dependen (yaitu probabilitas sukses dibagi dengan probabilitas gagal)
- $\beta_0$ : konstanta (intercept) dari model
- $\beta_1, \beta_2, \dots, \beta_n$ : koefisien regresi dari masing-masing variabel independen ( $X_1, X_2, \dots, X_n$ )
- $X_1, X_2, \dots, X_n$ : variabel independen yang digunakan dalam model

#### Kenapa Menggunakan Logistic Regression

Regresi logistik cocok untuk masalah klasifikasi biner (ya atau tidak, 1 atau 0) atau kategorik (besar sedang kecil, sering jarang tidak pernah). Model ini dapat diinterpretasikan, dan probabilitas output memberikan wawasan mengenai kemungkinan keanggotaan kelas. Regresi logistik efisien secara komputasi dan tidak mengharuskan variabel target memiliki hubungan linear dengan prediktor. Regresi ini sering menjadi pilihan pertama untuk tugas klasifikasi karena kesederhanaan dan interpretabilitasnya.

#### Implementasi Code

**Import Library yang akan digunakan:** Import library yang dibutuhkan untuk melakukan regresi logistik

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

- **LogisticRegression**: Kelas dari scikit-learn untuk membuat model regresi logistik.
- **train\_test\_split**: Untuk membagi dataset menjadi data latih dan data uji.
- **accuracy\_score**: Metrik untuk menghitung akurasi model.
- **Confusion\_matrix**: Metrik untuk menghasilkan confusion matrix yang berguna dalam evaluasi model klasifikasi.
- **Classification\_report** metrik untuk menghasilkan laporan yang mencakup precision, recall, dan F1-score.

**Persiapan Data:** Pisahkan variabel independen (fitur) dan variabel dependen (label).

Dalam contoh ini:

**Variabel independen:** semua variable selain HadArthritis dan PatientID.


**Variabel dependen:** HadArthritis.

```
1 X = df_num.drop(['HadArthritis', 'PatientID'], axis=1)
2 y = df_num['HadArthritis']
```

**Pembagian Data:** Bagi data menjadi data latih dan data uji untuk melatih model dan mengukur akurasi model.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```


**Pelatihan Model:** Gunakan data latih untuk melatih model regresi logistik. Model akan menyesuaikan koefisien dari setiap variabel independen untuk menemukan hubungan terbaik antara variabel input dan hasil.



```
1 log_regressor = LogisticRegression()  
2 log_regressor.fit(X_train, y_train)
```

- **Log\_regressor.fit():** Melatih model dengan data latih (**X\_train** sebagai fitur dan **y\_train** sebagai label).

**Prediksi:** Gunakan model yang sudah dilatih untuk memprediksi probabilitas pada data uji.



```
1 y_pred_log = log_regressor.predict(X_test)
```

- **predict(X\_test):** Menghasilkan prediksi berdasarkan data uji (**X\_test**). Hasil prediksi disimpan dalam **y\_pred\_log**.

**Intercept dan Koefisien :** Setelah melatih model, kita dapat memperoleh intercept (titik potong garis regresi dengan sumbu Y) dan koefisien (pengaruh setiap variabel input terhadap hasil prediksi).

```
1 intercept = log_regressor.intercept_[0] # Mendapatkan nilai intercept dari model
2 coefficients = log_regressor.coef_[0] # Mendapatkan koefisien untuk setiap fitur
3
4 print("Intercept:", intercept) # Menampilkan nilai intercept
5 print("Koefisien:", coefficients) # Menampilkan koefisien setiap fitur
```

- **Intercept ()**: Konstanta dalam model, menunjukkan prediksi ketika semua fitur bernilai nol.
- **Koefisien ()**: Pengaruh masing-masing fitur terhadap prediksi. Koefisien positif meningkatkan prediksi, sedangkan negatif menurunkannya.

**p-value** : digunakan untuk menguji signifikansi koefisien.

```
1 import statsmodels.api as sm # Mengimpor modul statsmodels untuk analisis statistik
2
3 X_train_const = sm.add_constant(X_train) # Menambahkan konstanta (intercept) pada data fitur (X_train)
4 logit_model = sm.Logit(y_train, X_train_const) # Membuat model regresi logistik (Logit)
5 result = logit_model.fit(dis=0) # Melatih model dengan data latih tanpa menampilkan proses fitting
6
7 print("p-value:", result.pvalues) # Menampilkan p-value untuk menguji signifikansi koefisien model
```

- **p-value**: Menunjukkan signifikansi koefisien. Jika  $p\text{-value} < 0.05$ , koefisien signifikan; jika  $> 0.05$ , koefisien tidak signifikan.

### D. Lasso Regression

#### Apa Itu Lasso Regression

Regresi Lasso (Least Absolute Shrinkage and Selection Operator) adalah metode regresi linear yang mengintegrasikan regularisasi untuk mengurangi kompleksitas model. Dengan menambahkan penalti pada fungsi biaya, Lasso mendorong beberapa koefisien prediktor menjadi nol, yang berfungsi sebagai teknik seleksi fitur otomatis. Ini memastikan hanya variabel yang paling relevan yang dipertahankan dalam model. Berbeda dengan regresi linear tradisional, yang mempertimbangkan semua variabel

independen, Lasso efektif dalam menangani data dengan banyak fitur dan variabel yang mungkin tidak relevan. Dengan menyusutkan koefisien-koefisien yang kurang penting, Lasso membantu menghindari overfitting dan menghasilkan model yang lebih sederhana dan lebih mudah diinterpretasi.

**Formula:**

$$\sum_{i=1}^n \left( y_i - \left( \beta_0 + \sum_{j=1}^p \beta_j x_{ij} \right) \right)^2 + \alpha \sum_{j=1}^p |\beta_j|$$

- $n$  mewakili jumlah pengamatan.
- $p$  menyatakan jumlah variabel yang tersedia dalam dataset.
- $X_{ij}$  mewakili nilai variabel ke- $j$  untuk pengamatan ke- $i$ , di mana  $i = 1, 2, \dots, n$  dan  $j = 1, 2, \dots, p$

**$\alpha$  (alfa) dapat mengambil berbagai nilai:**

- $\alpha = 0$ : Koefisien sama dengan regresi linear kuadrat terkecil
- $\alpha = \infty$ : Semua koefisien adalah nol
- $0 < \alpha < \infty$ : Koefisien berada di antara 0 dan regresi linear kuadrat terkecil:

### **Kenapa Menggunakan Lasso Regression**

Regresi Lasso cocok digunakan saat kita ingin melakukan seleksi fitur otomatis sebagai bagian dari pemodelan. Dengan menyusutkan nilai-nilai koefisien fitur yang kurang penting menjadi nol, regresi Lasso membantu menciptakan model yang lebih sederhana dan mudah diinterpretasikan, sekaligus mencegah overfitting. Model ini efisien secara komputasi, terutama ketika bekerja dengan data berdimensi tinggi dan fitur yang saling terkait. Regresi Lasso adalah pilihan tepat untuk kasus dimana kita membutuhkan model yang lebih sederhana dengan kinerja optimal.

### **Implementasi Code**

**Import Library yang akan digunakan:** Langkah pertama adalah mengimpor pustaka yang dibutuhkan untuk melakukan regresi Lasso dan mengevaluasi kinerjanya.



```
1 from sklearn.linear_model import Lasso
2 from sklearn.metrics import mean_squared_error
3 from sklearn.model_selection import train_test_split
4 import math
5 import matplotlib.pyplot as plt
```

- **Lasso:** Algoritma regresi Lasso dari scikit-learn yang akan kita gunakan untuk model regresi.
- **Mean\_squared\_error** Untuk menghitung rata-rata kesalahan kuadrat yang akan digunakan untuk mengevaluasi model.
- **Train\_test\_split:** Untuk membagi data menjadi data latih dan data uji.

**Persiapan Data:** Pisahkan variabel independen (fitur) dan variabel dependen (label).

Dalam contoh ini:

**Variabel independen:** semua variable selain BloodSugarLevel dan PatientID.


**Variabel dependen:** BloodSugarLevel.

```
1 X = df_num.drop(['BloodSugarLevel', 'PatientID'], axis=1)
2 y = df_num['BloodSugarLevel']
```

**Pembagian Data:** Bagi data menjadi data latih dan data uji untuk melatih model dan mengukur akurasi model.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

**Pelatihan Model:** Latih model menggunakan data train. Regresi Lasso menambahkan regularisasi L1 yang dapat mengurangi beberapa koefisien variabel menjadi nol, yang secara efektif mengurangi kompleksitas model.



```
1 lasso_model = Lasso(alpha=0.1)
2 lasso_model.fit(X_train, y_train)
```

**Prediksi:** Gunakan model yang sudah dilatih untuk memprediksi probabilitas pada data uji.



```
1 y_pred_lasso = lasso_model.predict(X_test)
```

### CLUSTERING

Clustering adalah proses mengelompokkan data ke dalam beberapa grup (cluster) berdasarkan kemiripan antar objek dalam grup tersebut. Tujuannya adalah agar objek dalam satu cluster memiliki kesamaan yang tinggi, sementara objek di cluster lain sangat berbeda. Proses ini sering melibatkan pengukuran jarak antara data dalam dataset. Metode clustering yang baik akan menghasilkan cluster dengan anggota yang serupa (intra-class similarity tinggi) dan membedakan cluster satu dengan lainnya (inter-class similarity rendah). Kualitas cluster ditentukan oleh seberapa baik metode tersebut menangkap pola dalam data menggunakan ukuran kemiripan, yang sering kali diukur dengan jarak antar objek. Tujuan melakukan clustering atau pengelompokkan dibedakan menjadi dua persepsi, yaitu clustering untuk pemahaman dan clustering untuk penggunaan. Proses clustering untuk tujuan pemahaman hanya sebagai proses awal sebelum melakukan summarization (rata-rata, standar deviasi), serta pelabelan kelas untuk setiap kelompok sehingga dapat digunakan sebagai data training dalam klasifikasi supervised. Sedangkan proses clustering untuk tujuan penggunaan biasanya untuk mencari prototipe kelompok yang paling merepresentasikan data serta memberikan abstraksi dari setiap objek data dalam suatu dataset.

#### A. Hierarchical Clustering

Hierarchical clustering pengelompokkan data dilakukan dengan **membuat suatu bagan hirarki (dendogram) untuk menunjukkan kemiripan antar data**. Dalam konteks ini, suatu cluster kecil dapat menjadi bagian dari cluster yang lebih besar sehingga menciptakan struktur yang berlapis. Beberapa **algoritma** yang termasuk ke dalam metode ini adalah **DIANA, AGNES, BIRCH, ROCK, dan CAMELEON**. Metode ini dikelompokkan menjadi dua pendekatan, yaitu agglomerative (penggabungan) dan divisive (pemisahan).

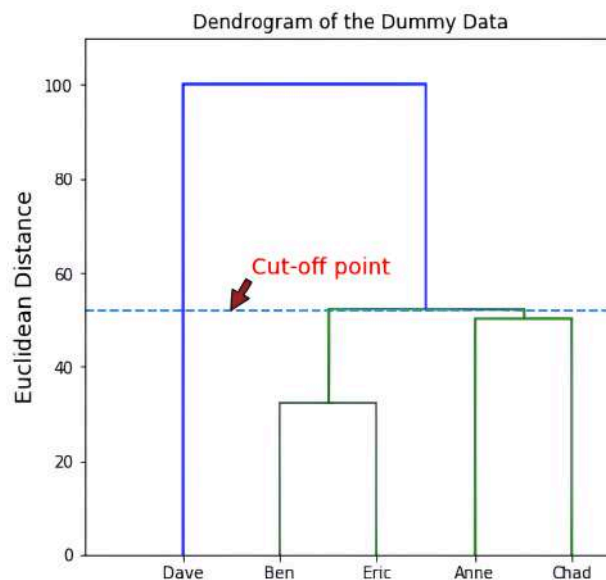
##### a. AGNES (Agglomerative Nesting)

Metode agglomerative merupakan pendekatan dari metode hierarchical clustering yang menggunakan strategi penggabungan dari bawah ke atas. Metode ini dimulai dengan menggabungkan setiap objek membentuk cluster sendiri dan secara iteratif menggabungkan cluster-cluster tersebut ke dalam cluster yang lebih besar hingga

semua objek data berada dalam satu cluster. Dengan demikian, metode agglomerative membutuhkan paling banyak  $n$  iterasi tergantung dari jumlah objek datanya.

### Dendrogram Complete Linkage Clustering

Complete linkage clustering adalah salah satu metode dalam hierarchical clustering yang mengukur jarak antara dua cluster dengan cara mencari jarak maksimum antara setiap titik data dalam cluster pertama dan titik data dalam cluster kedua. Metode ini merupakan pendekatan agglomerative, di mana setiap elemen awalnya berada dalam cluster terpisah dan kemudian digabungkan berdasarkan jarak antar cluster tersebut.



Metode terbaik untuk menentukan jumlah klaster adalah dengan mengamati dendrogram dan memilih nilai tertentu sebagai titik cut-off (secara manual). Jumlah perpotongan antara garis vertikal dan garis horizontal yang dibuat pada dendrogram akan menunjukkan jumlah klaster. Pada gambar di atas dengan memilih titik cut-off di 52, kita akan memperoleh 3 klaster yang berbeda, yaitu (Dave), (Ben, Eric), dan (Anne, Chad).

### Variasi dalam Penggabungan Cluster

Hierarki agglomerative clustering adalah metode bottom-up yang bertujuan menggabungkan  $n$  cluster hingga terbentuk satu cluster tunggal. Teknik-teknik pengelompokan antar cluster yang umum digunakan adalah sebagai berikut:

### 1. Single Linkage

Single Linkage adalah metode analisis hierarki yang mengelompokkan objek berdasarkan jarak terpendek antar objek terlebih dahulu. Jarak pada metode ini dapat dihitung menggunakan rumus berikut:

$$d_{w(i,j)} = \max\{d_{wi}, d_{wj}\}$$

di mana  $d_{w(i,j)}$  adalah ukuran kesamaan antara kelompok ke-w dengan kelompok (i,j), yaitu hasil penggabungan antara kelompok ke-i dan kelompok ke-j.

### 2. Complete Linkage

Complete Linkage mengelompokkan berdasarkan jarak terjauh antar objek dalam setiap cluster. Jarak antar cluster diperoleh dari perhitungan objek dengan jarak paling jauh. Rumus untuk menentukan jarak tersebut adalah sebagai berikut:

$$d_{w(i,j)} = \min\{d_{wi}, d_{wj}\}$$

di mana  $d_{w(i,j)}$  adalah ukuran kesamaan antara kelompok ke-w dengan kelompok (i,j), yaitu hasil penggabungan antara kelompok ke-i dan kelompok ke-j.

### 3. Average Linkage

Average Linkage menggunakan rata-rata perbedaan jarak individu di satu kelompok dengan seluruh individu di kelompok lain. Jarak pada metode ini dapat dihitung menggunakan rumus berikut:

$$d_{w(i,j)} = \frac{\sum_q \sum_r d_{qr}}{n_{(ij)} n_w}$$

di mana  $d_{w(i,j)}$  menunjukkan ukuran kesamaan antara kelompok ke-ww dan kelompok (i,j), yang merupakan hasil penggabungan kelompok ke-i dan ke-j (disimbolkan sebagai kelompok ij).  $n_{(ij)}$  dan  $n_w$  adalah jumlah pengamatan dalam kelompok ij dan w, sedangkan  $d_{qr}$  adalah jarak antara pengamatan ke-q dalam kelompok ij dan pengamatan ke-r dalam kelompok w.

#### A. Study Case

**Study Case: Identifikasi Risiko Diabetes dan Obesitas Berdasarkan Profil BMI dan Gula Darah di Klinik Kesehatan.**

### Deskripsi:

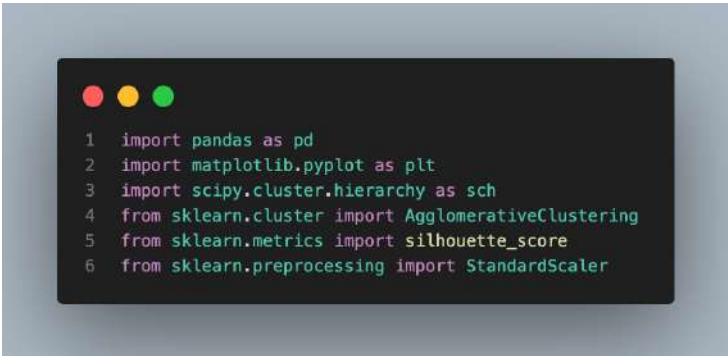
Pada studi kasus ini, variabel yang digunakan adalah BMI dan Blood Sugar Level. BMI adalah pengukuran yang menunjukkan apakah seseorang memiliki berat badan yang sesuai dengan tinggi badannya, sedangkan Blood Sugar Level mengukur kadar glukosa dalam darah yang dapat mencerminkan risiko kesehatan terkait metabolisme.

### Tujuan:

Analisis ini adalah untuk mengidentifikasi individu berdasarkan tingkat risiko kesehatan mereka menggunakan klusterisasi BMI dan gula darah. Sehingga dapat berfungsi sebagai alat pemantauan jangka panjang untuk membantu manajemen kesehatan berkelanjutan dengan pendekatan yang lebih proaktif.


## B. Penjelasan

### a. Import Libraries



```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import scipy.cluster.hierarchy as sch
4 from sklearn.cluster import AgglomerativeClustering
5 from sklearn.metrics import silhouette_score
6 from sklearn.preprocessing import StandardScaler
```

### b. Load Data



```
1 data_path = 'data_modul_2.csv'
2 data = pd.read_csv(data_path)
3 X = data[['BMI', 'BloodSugarLevel']]
```

### Deskripsi:

`data_path` menyimpan path file dataset. Setelah itu, `pd.read_csv (data_path)` membaca data dari file CSV dan `X` memilih kolom BMI dan BloodSugarLevel dari data untuk digunakan dalam clustering.



### c. Standarisasi Data

```
1 scaler = StandardScaler()  
2 X_scaled = scaler.fit_transform(X)
```

#### Deskripsi:

Data pada X dinormalisasi dengan StandardScaler agar tiap fitur memiliki skala yang sama (mean=0 dan std=1), penting jika fitur memiliki skala yang berbeda. Hasilnya disimpan pada X\_scaled.

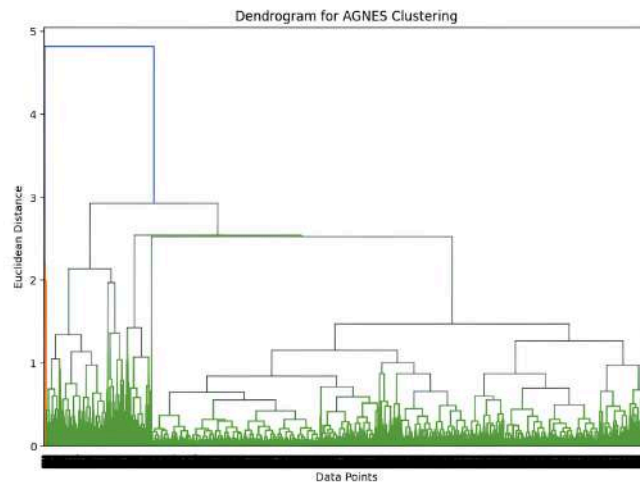
### d. Dendrogram untuk AGNES

```
1 plt.figure(figsize=(10, 7))  
2 dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))  
3 plt.title('Dendrogram for AGNES Clustering')  
4 plt.xlabel('Data Points')  
5 plt.ylabel('Euclidean Distance')  
6 plt.show()
```

#### Deskripsi:

Fungsi `sch.dendrogram (sch.linkage (X_scaled, method='average'))` menghasilkan dendrogram dengan metode 'average' untuk menghitung jarak antar kelompok. Dendrogram ini menunjukkan proses pengelompokan data dari level individu hingga terbentuknya satu cluster besar, membantu dalam menentukan jumlah cluster optimal berdasarkan jarak antar node.

## Output:



### e. Melakukan Clustering AGNES dan Evaluasi dengan Silhouette Score

```
1 n_clusters = 3 # Menetapkan cluster berdasarkan dendrogram
2 model = AgglomerativeClustering(n_clusters=n_clusters, linkage='average') # Variasi gabung cluster 'single', 'complete', 'average', 'ward'
3 cluster_labels = model.fit_predict(X_scaled)
```

#### Deskripsi:

Langkah selanjutnya adalah memulai model AGNES dengan jumlah cluster `n_clusters` (misalnya 3) dan metode linkage 'average'. Model `AgglomerativeClustering` ini akan mengelompokkan data pada `X_scaled` dan menyimpan label cluster di `cluster_labels`, yang menunjukkan keanggotaan tiap data.

### f. Menghitung Silhouette Score

```
1 score = silhouette_score(X_scaled, cluster_labels)
2 print(f'Silhouette Score for {n_clusters} clusters: {score}')
```

#### Deskripsi:

Kualitas clustering dievaluasi dengan `silhouette_score`, yang mengukur seberapa baik tiap data berada dalam clusternya dibandingkan dengan cluster lain. Skor yang lebih tinggi menunjukkan kualitas cluster yang lebih

baik. Hasilnya dicetak untuk memberi wawasan apakah jumlah cluster sudah optimal atau perlu disesuaikan.

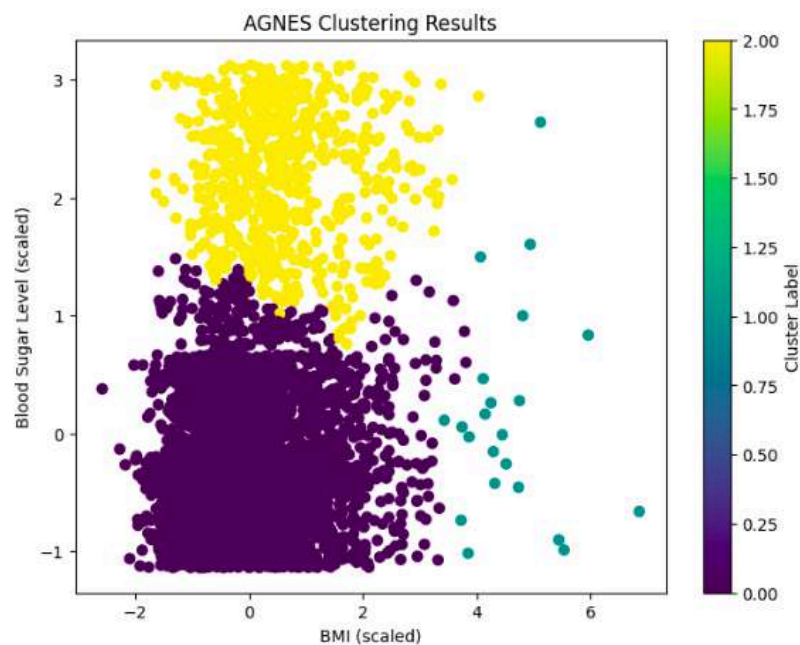
g. Memvisualisasikan Hasil Clustering

```
1 plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=cluster_labels, cmap='viridis')
2 plt.title('AGNES Clustering Results')
3 plt.xlabel('BMI (scaled)')
4 plt.ylabel('Blood Sugar Level (scaled)')
5 plt.colorbar(label='Cluster Label')
6 plt.show()
```

**Deskripsi:**

Langkah terakhir adalah memvisualisasikan hasil clustering dalam scatter plot. Dengan ukuran plot yang diatur, data diplot berdasarkan 'BMI' (sumbu x) dan 'BloodSugarLevel' (sumbu y), dan tiap cluster ditandai warna berbeda (c=cluster\_labels). 'plt.colorbar' ditambahkan untuk menunjukkan label cluster, memberikan gambaran distribusi akhir data dalam cluster.

**Output:**



### Deskripsi:

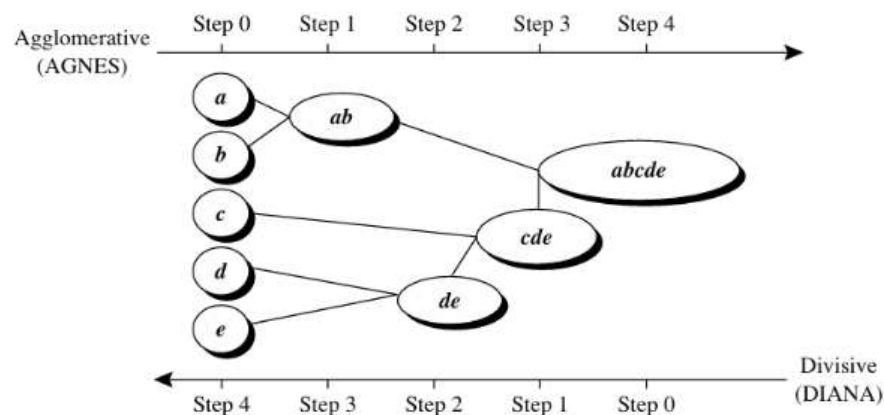
Hasil klusterisasi AGNES pada plot menunjukkan tiga kelompok berdasarkan BMI dan Blood Sugar Level:

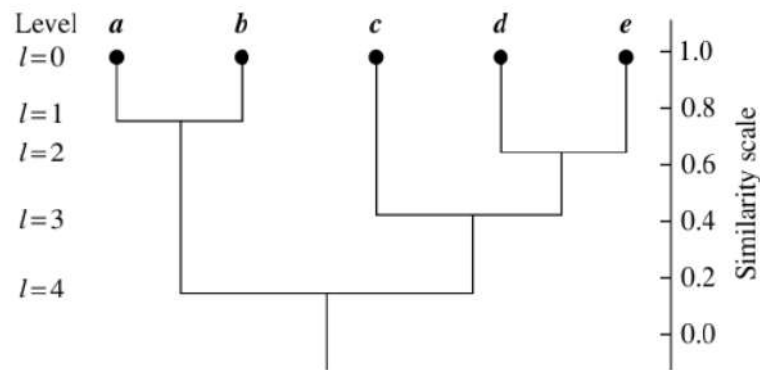
- Kluster Ungu (Label 0): Mencakup individu dengan BMI dan gula darah rata-rata lebih rendah.
- Kluster Kuning (Label 1): Berisi individu dengan gula darah lebih tinggi, menunjukkan potensi risiko kesehatan, seperti diabetes.
- Kluster Hijau-Biru (Label 2): Mengelompokkan individu dengan BMI tinggi dan variasi gula darah, yang bisa terkait dengan risiko kesehatan kronis, seperti diabetes tipe 2.

### b. DIANA (Divisive Analysis)

Metode divisive merupakan pendekatan dari metode hierarchical clustering yang menggunakan strategi pemisahan dari atas ke bawah. Metode ini dimulai dengan menempatkan semua objek data dalam satu cluster yang merupakan akar hierarki. Kemudian membagi akar cluster tersebut ke beberapa cluster yang lebih kecil dan secara rekursif akan mempartisi atau memisahkan cluster-cluster tersebut menjadi cluster yang lebih kecil lagi. Proses pemisahan berlanjut hingga cluster terendah yang hanya berisi satu objek data saja atau objek-objek di dalam sebuah cluster yang mirip satu sama lain.

Hasil clustering dengan metode ini digambarkan dalam bentuk dendogram. Root pada dendogram merupakan keseluruhan dataset dan setiap cabang merupakan data point sesuai dengan level-level dari hasil clustering.





Gambar di atas menunjukkan dendrogram untuk lima data, yaitu a, b, c, , dan e, di mana  $l=0$  menunjukkan kelima objek sebagai cluster tunggal pada level 0. Pada  $l=1$ , objek a dan b dikelompokkan bersama untuk membentuk cluster pertama, dan mereka tetap bersama di semua level berikutnya. Kemudian dapat menggunakan sumbu vertikal untuk menunjukkan skala kemiripan antar cluster. Sebagai contoh, ketika kemiripan dua kelompok objek, a, b dan c, d, e, adalah sekitar 0.16, mereka digabungkan bersama untuk membentuk satu cluster.

### B. Partitional Clustering

Partitional clustering merupakan pengelompokan data dilakukan dengan mengelompokkan data point ke dalam k cluster tanpa struktur hirarki. Metode ini membagi dataset ke dalam sejumlah kelompok yang tidak saling tumpang tindih antara satu cluster dengan cluster lainnya. Dengan demikian, setiap data hanya termasuk ke dalam satu cluster. Beberapa algoritma yang termasuk ke dalam metode ini adalah K-Means, K-Medoids, dan CLARANS.

#### a. K-Means

Menurut Amalina et al. (2019), K-Means Clustering adalah metode pengelompokan data non-hirarki yang membagi data ke dalam beberapa klaster. Teknik ini mengelompokkan data dengan karakteristik serupa ke dalam satu klaster dan memisahkan data dengan karakteristik berbeda ke klaster lain. Dalam K-Means Clustering, "K" adalah konstanta yang menunjukkan jumlah klaster yang diinginkan, sedangkan "Means" merujuk pada nilai rata-rata dari setiap kelompok data yang diidentifikasi sebagai klaster.

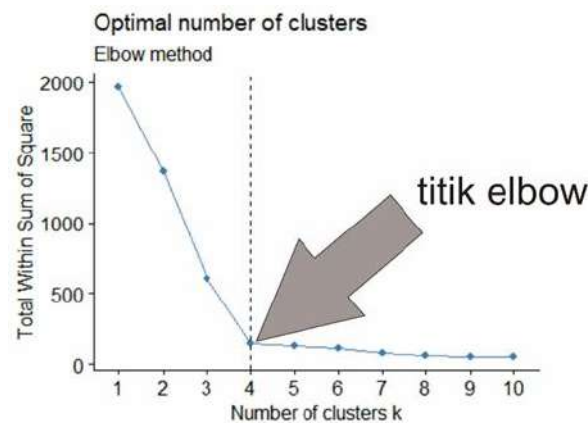
Metode K-Means mengelompokkan data ke dalam kluster dengan tujuan meminimalkan jarak kuadrat antara titik data dan pusat kluster. Setiap kluster berisi titik data yang serupa, sedangkan antar kluster memiliki perbedaan. Namun, algoritma ini sangat bergantung pada inisialisasi kluster, yang dapat mempengaruhi hasil akhir.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Observation A = (X1, Y1)

Observation B = (X2, Y2)

### Elbow Method



Metode Elbow adalah teknik yang digunakan untuk menentukan jumlah cluster yang optimal dengan mengamati persentase perbandingan antar jumlah cluster yang membentuk pola mirip siku pada suatu titik tertentu. Jika perbedaan antara cluster pertama dan kedua menghasilkan sudut pada grafik atau menunjukkan penurunan yang paling signifikan, maka jumlah cluster pada titik tersebut dianggap optimal. Perbandingan ini dilakukan dengan menghitung Sum of Square Error (SSE) dari tiap cluster, karena semakin besar jumlah cluster K, semakin kecil nilai SSE yang dihasilkan.



### A. Study Case

#### Study Case: Analisis Risiko Kesehatan Berdasarkan BMI dan Tingkat Gula Darah

##### Deskripsi:

Studi ini dapat digunakan dalam analisis risiko kesehatan untuk mengidentifikasi kelompok-kelompok populasi berdasarkan BMI dan tingkat gula darah. Dengan demikian, dapat membantu dalam mendiagnosis atau mengklasifikasikan tingkat risiko kesehatan terkait obesitas dan diabetes.

##### Tujuan:

Mengidentifikasi pola atau kluster dalam data yang menunjukkan perbedaan karakteristik antara kelompok dengan 'BMI' dan 'BloodSugarLevel' tertentu.

### B. Penjelasan

#### a. Import Libraries

```
1 from sklearn.preprocessing import StandardScaler, OneHotEncoder
2 from sklearn.compose import ColumnTransformer
3 from sklearn.pipeline import Pipeline
4 from sklearn.cluster import KMeans
5 from sklearn.metrics import silhouette_score
6 from sklearn.decomposition import PCA
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9 import pandas as pd
10 import numpy as np
```

#### b. Read Data

```
1 data_path = 'data_modul_2.csv'
2 data = pd.read_csv(data_path)
```

##### Deskripsi:

Membaca data dari file CSV data\_modul\_2.csv ke dalam DataFrame data.

### c. Mendefinisikan Fitur untuk Klasterisasi

```
1 X = data[['BMI', 'BloodSugarLevel']]
```

#### Deskripsi:

Di sini kita memilih kolom BMI dan BloodSugarLevel sebagai fitur yang akan digunakan untuk klasterisasi.

### d. Standarisasi Data

```
1 scaler = StandardScaler()  
2 X_scaled = scaler.fit_transform(X)
```

#### Deskripsi:

Standarisasi data menggunakan StandardScaler, agar fitur BMI dan BloodSugarLevel memiliki rata-rata 0 dan standar deviasi 1.

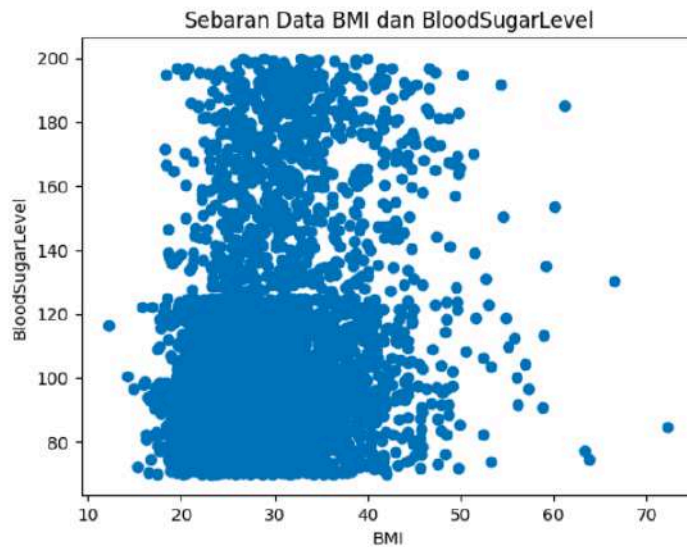
### e. Visualisasi Sebaran Data Awal

```
1 plt.scatter(data=X['BMI'], y=data['BloodSugarLevel'])  
2 plt.xlabel('BMI')  
3 plt.ylabel('BloodSugarLevel')  
4 plt.title('Sebaran Data BMI dan BloodSugarLevel')  
5 plt.show()
```

#### Deskripsi:

Membuat scatter plot untuk melihat sebaran awal data berdasarkan BMI dan BloodSugarLevel.

### Output:



Scatter plot di atas menunjukkan sebaran data berdasarkan dua variabel, yaitu BMI (Body Mass Index) pada sumbu x dan Blood Sugar Level (Tingkat Gula Darah) pada sumbu y. Secara keseluruhan, plot ini memberikan gambaran mengenai sebaran individu dalam dataset berdasarkan BMI dan tingkat gula darah, dengan mayoritas data terkonsentrasi pada BMI sedang dan tingkat gula darah normal atau sedikit di atas normal.

### f. Elbow Method untuk Menentukan Jumlah Kluster

```
1 wcss = []
2 for i in range(1, 11): # Rentang nilai ini dapat diatur sesuai kebutuhan
3     kmeans = KMeans(n_clusters=i, random_state=42)
4     kmeans.fit(X_scaled)
5     wcss.append(kmeans.inertia_)
```

### Deskripsi:

Bagian ini menggunakan Metode Elbow untuk menentukan jumlah kluster optimal. WCSS (Within-Cluster Sum of Squares) dihitung untuk berbagai nilai K. Nilai K yang optimal terlihat ketika grafik WCSS mulai rata.

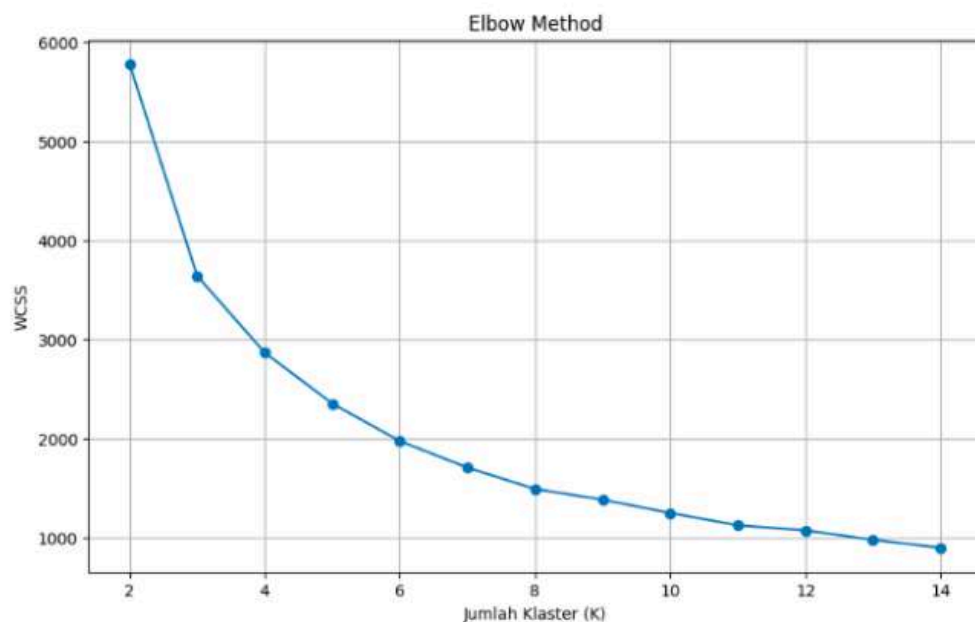
### g. Plot Hasil Metode Elbow



#### Deskripsi:

Grafik Elbow Method untuk memvisualisasikan penurunan WCSS dengan bertambahnya jumlah klaster. Titik elbow akan menunjukkan jumlah klaster yang optimal.

#### Output:



Gambar di atas menunjukkan Elbow Method untuk menentukan jumlah klaster optimal berdasarkan Within-Cluster Sum of Squares (WCSS). Pada grafik ini, terlihat bahwa penurunan WCSS yang signifikan terjadi antara jumlah klaster

2 dan 4. Setelah kluster ke-3 atau ke-4, penurunan WCSS mulai melandai, menunjukkan bahwa penambahan kluster tidak lagi memberikan peningkatan yang signifikan pada pemisahan data.

h. Melakukan K-Means dengan K Optimal

```
1 optimal_k = 3 # Sesuaikan dengan hasil plot elbow method
2 kmeans = KMeans(n_clusters=optimal_k, random_state=42)
3 labels = kmeans.fit_predict(X_scaled)
```

**Deskripsi:**

Menentukan K optimal (contohnya 3) berdasarkan hasil dari Metode Elbow. K-Means diterapkan pada data yang sudah diskalakan, dan label kluster disimpan di variabel labels.

i. Menghitung Silhouette Score

```
1 silhouette_avg = silhouette_score(X_scaled, labels)
2 print(f'Silhouette Score untuk {optimal_k} cluster: {silhouette_avg}')
```

**Deskripsi:**

Menghitung skor Silhouette untuk mengevaluasi kualitas kluster. Skor ini mengukur seberapa baik data dalam kluster tersebut dan seberapa jauh kluster lainnya. Nilai mendekati 1 menunjukkan kluster yang baik.

j. Menyimpan Label Kluster di DataFrame

```
1 data['Cluster'] = labels
```

### Deskripsi:

Menyimpan hasil klasterisasi (labels) ke dalam kolom Cluster pada DataFrame data.

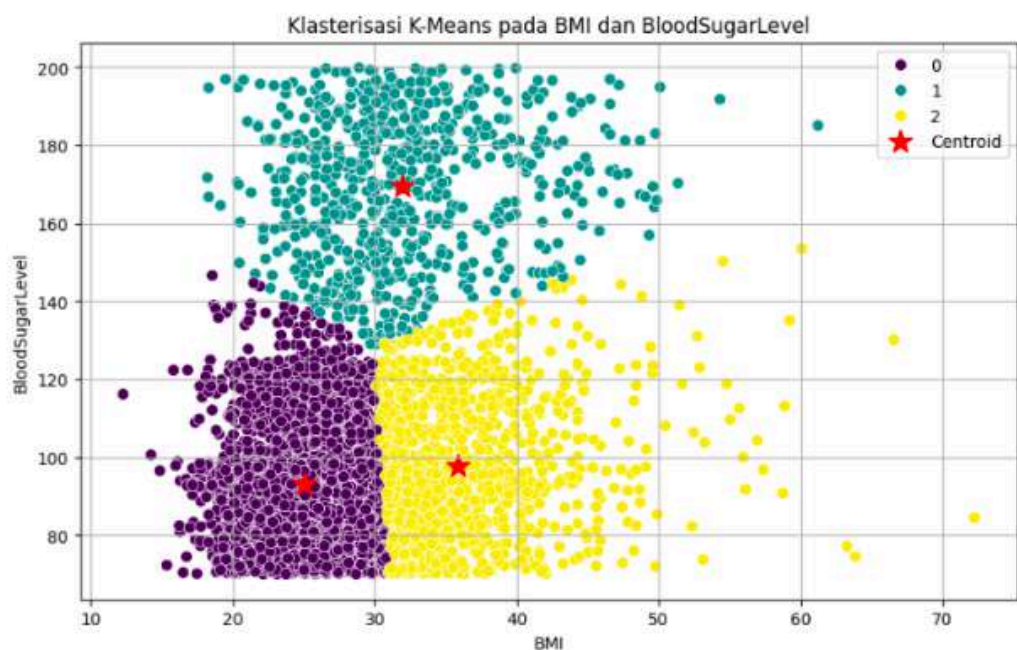
- k. Visualisasi Hasil Klasterisasi, Menghitung dan Menampilkan Centroid Cluster

```
1 plt.figure(figsize=(10, 5))
2
3 sns.scatterplot(data=data, x='BMI', y='BloodSugarLevel', hue='Cluster', data=data, palette='viridis', s=5)
4 plt.title('Klasterisasi K-Means pada BMI dan BloodSugarLevel')
5 plt.xlabel('BMI')
6 plt.ylabel('BloodSugarLevel')
7 plt.legend(title='Klaster')
8 plt.grid(True)
9
10 centroids = scaler.inverse_transform(kmeans.cluster_centers_)
11
12 plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='*', label='Centroid')
13
14 plt.legend()
15 plt.show()
```

### Deskripsi:

Membuat scatter plot dengan seaborn untuk memvisualisasikan hasil klasterisasi, dimana setiap klaster memiliki warna berbeda. Menghitung centroid setiap klaster (titik tengah klaster) dan menampilkan centroid tersebut pada plot. `inverse_transform` digunakan agar centroid kembali dalam skala asli data sebelum standarisasi.

### Output:





### Deskripsi:

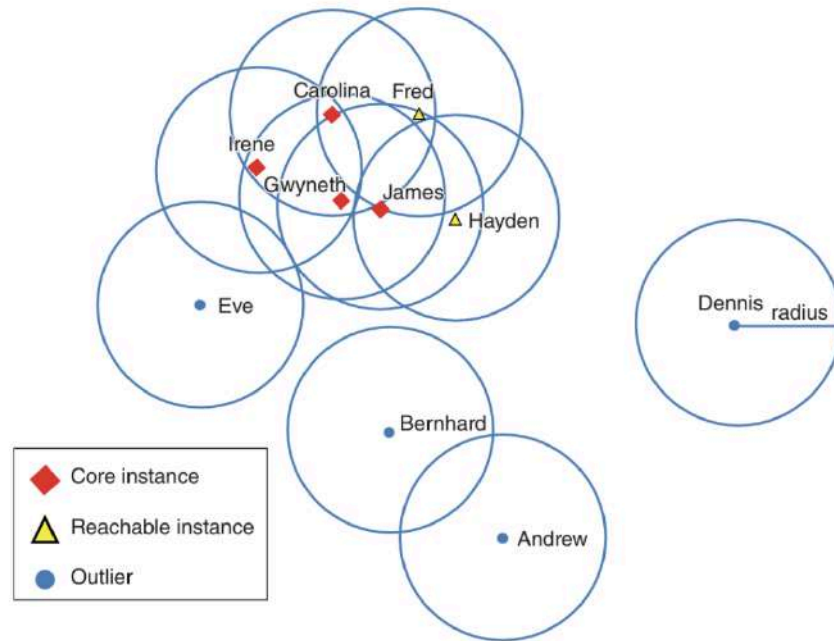
- Klaster 0: Individu dalam klaster ini mungkin berada pada risiko lebih rendah terhadap kondisi terkait obesitas atau diabetes karena memiliki BMI dan BloodSugarLevel yang lebih rendah.
- Klaster 1: Individu dalam klaster ini memiliki risiko yang mungkin lebih tinggi terhadap diabetes atau masalah gula darah, karena memiliki BloodSugarLevel yang tinggi meskipun BMI beragam.
- Klaster 2: Individu dengan BMI tinggi dan BloodSugarLevel menengah dapat mengarah ke risiko masalah kesehatan tertentu yang terkait dengan obesitas tetapi bukan masalah gula darah tinggi.

### C. Density-Based Clustering

Metode density-based clustering adalah metode pengelompokkan objek data yang berbasis kepadatan dalam ruang antar data sehingga memiliki bentuk cluster yang berubah-ubah. Sebuah cluster terbentuk di area dengan kepadatan data tinggi dan area dengan kepadatan rendah akan dianggap sebagai pembatas antar cluster atau dianggap sebagai outlier. Beberapa algoritma yang termasuk ke dalam metode ini adalah DBSCAN, OPTICS, dan DenClue.

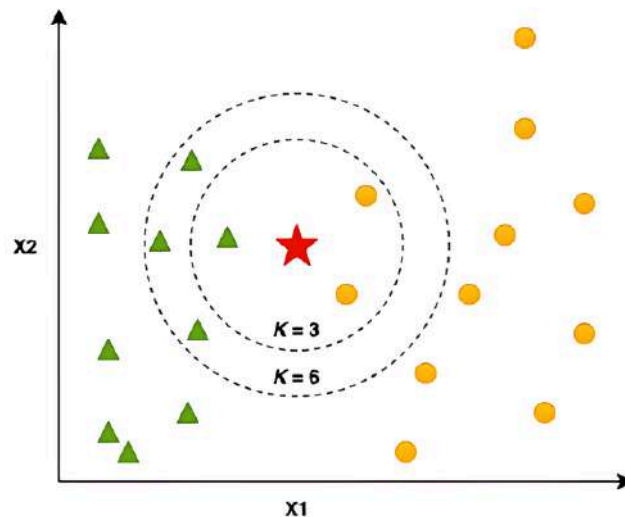
#### a. DBSCAN (Density-Based Spatial Clustering of Application with Noise)

DBSCAN merupakan metode pengelompokkan data berbasis kepadatan yang mendefinisikan objek yang membentuk area padat sebagai bagian dari cluster yang sama. Objek yang tidak termasuk ke dalam area dengan kepadatan tinggi akan dianggap sebagai noise atau outlier. Sebuah instance inti  $p$  merupakan instance yang mencapai jumlah minimum dari instance lainnya. Untuk dianggap “dapat dijangkau secara langsung”, sebuah instance  $q$  harus berada pada jarak yang lebih rendah dari  $p$  daripada ambang batas yang telah ditentukan (dilambangkan dengan  $\epsilon$ ). Jika  $p$  adalah sebuah instance inti, maka ia membentuk sebuah cluster bersama dengan semua instance yang dapat dijangkau darinya, baik secara langsung maupun tidak langsung. Setiap cluster berisi setidaknya satu instance inti. Gambar di bawah menunjukkan adanya satu cluster (Carolina, Fred, Gwyneth, Hayden, Irene dan James) dan empat outlier (Andrew, Bernhard, Dennis dan Eve).



Menilai dan mengevaluasi hasil DBSCAN tidak memiliki karakteristik khusus. Tidak ada centroid atau pusat dalam DBSCAN sehingga instance dari setiap cluster dapat dinilai. Mengatur hyper-parameter utama dari DBSCAN adalah jumlah minimum instance yang diperlukan untuk mengklasifikasikan sebuah instance sebagai inti atau core dan jarak maksimum sebuah instance untuk dapat dianggap “dapat dijangkau secara langsung”. Ukuran jarak juga merupakan hyper-parameter yang harus dipilih tergantung pada jenis skala atribut data.

### K-Nearest Neighbors



K-Nearest Neighbor Distance (kNNdist) adalah metode untuk menghitung jarak cepat antara titik dan tetangga k-terdekat dalam matriks titik. Plot kNNdist dapat digunakan untuk membantu menentukan nilai eps yang optimal untuk DBSCAN. Pada plot kNNdist, nilai eps yang tepat dapat ditemukan karena akan menunjukkan titik optimal untuk pengelompokan.

#### A. Study Case

**Study Case: Identifikasi Kelompok Risiko Kesehatan Berdasarkan BMI dan Kadar Gula Darah.**

##### **Deskripsi:**

Mengidentifikasi pola kelompok berdasarkan indeks massa tubuh (BMI) dan kadar gula darah. Penggunaan algoritma DBSCAN memungkinkan kita untuk menemukan pola atau kelompok yang alami dalam data, sekaligus mengidentifikasi outlier atau individu yang memiliki nilai yang signifikan berbeda dari populasi umum

##### **Tujuan:**

Tujuan dari analisis ini adalah untuk memahami pola kesehatan dalam populasi dengan mengkaji hubungan antara indeks massa tubuh (BMI) dan kadar gula darah. Melalui pemahaman pola umum ini, kita dapat mengidentifikasi kelompok risiko rendah, sedang, dan tinggi berdasarkan kombinasi antar variabel.

### B. Penjelasan

#### a. Import Libraries

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.cluster import DBSCAN
4 from sklearn.preprocessing import StandardScaler
5 import matplotlib.pyplot as plt
6 from sklearn.neighbors import NearestNeighbors
```

#### b. Load Dataset

```
1 data_path = 'data_modul_2.csv'
2 data = pd.read_csv(data_path)
3 X = data[['BMI', 'BloodSugarLevel']]
```

#### Deskripsi:

Data dari data\_modul\_2.csv dibaca ke dalam DataFrame data. Kemudian, kolom BMI dan BloodSugarLevel diseleksi dan disimpan dalam variabel X, yang akan digunakan untuk proses clustering.

#### c. Normalisasi Data

```
1 scaler = StandardScaler()
2 X_scaled = scaler.fit_transform(X)
```

#### Deskripsi:

Data X dinormalisasi menggunakan StandardScaler, agar setiap fitur memiliki nilai rata-rata 0 dan standar deviasi 1. Ini penting dalam DBSCAN untuk

memperlakukan setiap fitur dengan skala yang seimbang. `X_scaled` adalah data hasil normalisasi.

d. Find Nearest Neighbors

```
1 k = 4
2 neighbors = NearestNeighbors(n_neighbors=k)
3 neighbors.fit(X_scaled)
4 distances, indices = neighbors.kneighbors(X_scaled)
```

**Deskripsi:**

Tentukan jumlah tetangga terdekat ( $k$ ) yang akan digunakan untuk menentukan parameter  $\epsilon$ . Di sini,  $k$  dipilih 4. `NearestNeighbors` digunakan untuk mencari jarak antara setiap titik ke tetangga terdekat. ke-4.

e. Sort Distance

```
1 distances = np.sort(distances, axis=0)
```

**Deskripsi:**

Jarak yang ditemukan di langkah sebelumnya diurutkan berdasarkan jarak tetangga ke-4 dari setiap titik. Pengurutan ini membantu dalam menemukan elbow point pada grafik jarak, yang dapat digunakan untuk menentukan nilai  $\epsilon$ .

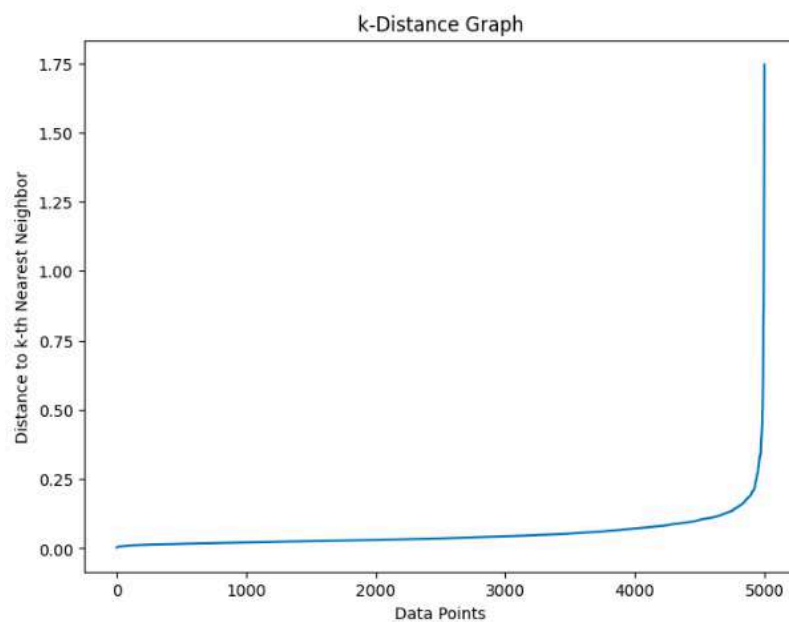
### f. Plot K-Distance Graph

```
1 plt.figure(figsize=(10, 6))
2 plt.plot(distances)
3 plt.title('k-Distance Graph')
4 plt.xlabel('Data Points')
5 plt.ylabel('Distance to k-th Nearest Neighbor')
6 plt.show()
```

#### Deskripsi:

Grafik k-distance digambar untuk melihat titik bahu (elbow point). Titik ini menunjukkan jarak optimal sebagai parameter eps untuk DBSCAN.

#### Output:



Dari grafik k-Distance, terdapat titik elbow di mana jarak mulai meningkat tajam. Memilih  $\text{eps} = 0.5$  dan  $\text{min\_samples} = 5$  memungkinkan DBSCAN untuk secara efektif membedakan kluster padat dari anomali. Nilai eps dipilih berdasarkan titik di mana kurva mulai naik tajam, menunjukkan jarak yang lebih besar ke tetangga. Titik-titik yang tidak memiliki cukup tetangga dalam radius 0.5 akan dianggap sebagai anomali.



### g. Set Parameters untuk DBSCAN

```
1 eps = 0.5 # Sesuaikan dengan hasil "elbow point" dari plot sebelumnya
2 min_samples = 5
```

#### Deskripsi:

Nilai eps (jarak maksimum untuk memasukkan titik dalam cluster) ditentukan berdasarkan elbow point dari grafik k-distance, dan min\_samples diatur ke 5.

### h. Inisialisasi dan Penyesuaian Model DBSCAN

```
1 dbscan = DBSCAN(eps=eps, min_samples=min_samples)
2 dbscan.fit(X_scaled)
```

#### Deskripsi:

Model DBSCAN diinisialisasi dengan parameter eps dan min\_samples, kemudian dipasangkan ke data yang telah dinormalisasi (X\_scaled).

### i. Tambahkan Label Cluster ke Data

```
1 labels = dbscan.labels_
2 data['Cluster'] = labels
```

### Deskripsi:

Label hasil clustering (labels) ditambahkan ke DataFrame data dalam kolom Cluster. Label ini menunjukkan cluster yang dihasilkan untuk setiap data; -1 berarti data tersebut dianggap sebagai noise.

#### j. Periksa Distribusi Cluster



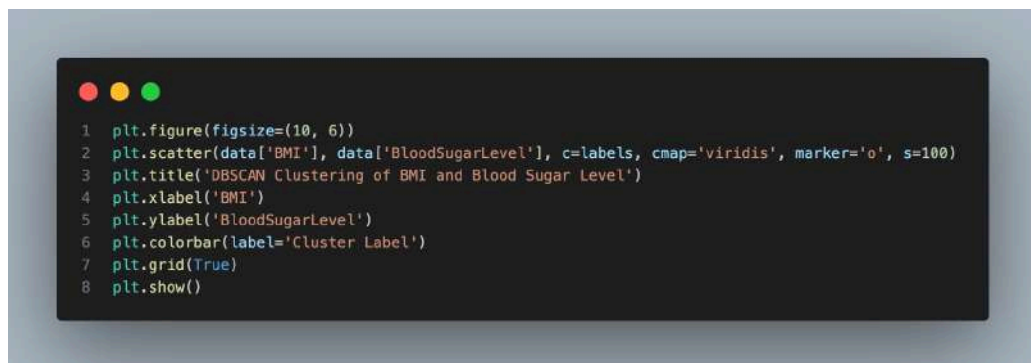
### Deskripsi:

Melihat jumlah data dalam setiap cluster, termasuk data yang dianggap noise.

### Output:

```
Cluster
0      4989
-1       11
Name: count, dtype: int64
```

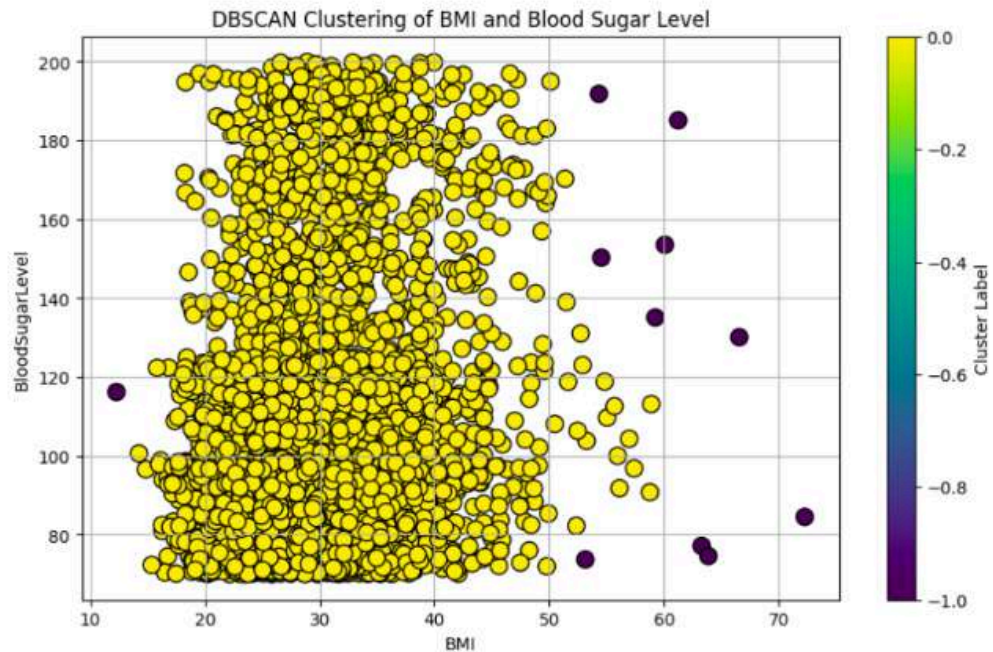
#### k. Hasil Pengelompokan Plot



### Deskripsi:

Visualisasi hasil clustering menggunakan scatter plot. Setiap titik data diberi warna sesuai dengan cluster yang dihasilkan. Data noise berwarna berbeda untuk memudahkan identifikasi.

### Output:



### Deskripsi:

Grafik hasil DBSCAN clustering pada data 'BMI' dan 'BloodSugarLevel' menunjukkan bahwa terdapat dua kelompok utama:

- Cluster 0 (Kuning) - Sebagian besar data termasuk dalam cluster ini, yang mengindikasikan bahwa sebagian besar individu memiliki pola yang serupa pada nilai BMI dan Blood Sugar Level. Ini mungkin merupakan kelompok utama dalam populasi yang dianalisis, dengan rentang BMI sekitar 15 hingga 45 dan Blood Sugar Level antara 70 hingga 200.
- Cluster -1 (Outlier, Ungu) - Beberapa titik data berada di luar cluster utama (diidentifikasi oleh DBSCAN sebagai outlier dengan label -1). Outlier ini mungkin mewakili individu dengan nilai BMI atau Blood Sugar Level yang sangat tinggi, yang secara statistik berbeda dari populasi utama.

### REFERENSI

GeeksforGeeks. (n.d.). Types of regression techniques. GeeksforGeeks. Retrieved April, 2025, from <https://www.geeksforgeeks.org/types-of-regression-techniques/>

GeeksforGeeks. (n.d.). Clustering in machine learning. GeeksforGeeks. Retrieved April, 2025, from <https://www.geeksforgeeks.org/clustering-in-machine-learning/>

Nugroho, K. S. (2020, June 28). Confusion matrix untuk evaluasi model pada unsupervised machine learning. Medium. Retrieved April, 2025, from <https://ksnugroho.medium.com/confusion-matrix-untuk-evaluasi-model-pada-unsupervised-machine-learning-bc4b1ae9ae3f>

Dicoding. (2021, June 7). Machine learning workflow: Langkah-langkah praktis untuk membangun model AI. Dicoding Blog. Retrieved April, 2025, from <https://www.dicoding.com/blog/machine-learning-workflow-langkah-langkah-praktis-untuk-membangun-model-ai/>

Esairina, N. (2021, September 28). Algoritma K-Nearest Neighbor (KNN): Penjelasan dan implementasi untuk klasifikasi kanker. Medium. Retrieved April, 2025, from <https://esairina.medium.com/algoritma-k-nearest-neighbor-knn-penjelasan-dan-implementasi-untuk-klasifikasi-kanker-ff9b7fbe0a4>

Berdata. (2020, September 17). Memahami proses agglomerative clustering. Berdata. Retrieved April, 2025, from <https://www.berdata.com/post/memahami-proses-agglomerative-clustering-1>