

# ***CLUSTERING***

DETEKSI TRANSAKSI ANOMALI  
MENGUNAKAN ALGORITMA DBSCAN  
PADA DATASET “BANK CUSTOMER SEGMENTATION”

## **LAPORAN**

**Oleh:**

**Tim GCD**

**IF-A PAGI**

- **211110347 - CINDY SINTIYA**
- **211110948 - GRACE HELENA HUTAGAOL**
- **211111930 - DAVID BATE'E**



UNIVERSITAS  
**MIKROSKIL**

**PROGRAM STUDI S-1**  
**FAKULTAS INFORMATIKA**  
**TEKNIK INFORMATIKA**  
**UNIVERSITAS MIKROSKIL**  
**2024/2025**

# Judul

Deteksi Transaksi Anomali Menggunakan Algoritma DBSCAN pada Dataset “*Bank Customer Segmentation*”

## Latar Belakang

Algoritma DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) merupakan salah satu algoritma pembelajaran mesin yang mengelompokkan (*clustering*) data dalam grup (*cluster*) berdasarkan kepadatan (*density*) dan jarak antar titik-titik data.

Pada awalnya, kami ingin menggunakan algoritma ini untuk membentuk *cluster* berdasarkan transaksi nasabah sesuai dataset yang dipilih. Namun, *cluster* yang terbentuk cenderung kecil (hanya -1, 0, 1 dengan fokus data di cluster 0 sebanyak >90%) dan tidak sesuai.

Setelah menganalisis lebih lanjut, kami memutuskan untuk tetap menggunakan data yang sama dan melihat dari pola *clustering* yang terbentuk mengarah ke pemisahan data berdasarkan *outliers/ noise*.

## Tujuan

1. Mendeteksi aktivitas transaksi yang tidak wajar dan mencurigakan (*unusual/ suspicious transaction*) dari para nasabah
2. Mendeteksi kemungkinan pencucian uang dari nasabah dengan saldo rekening banyak namun aktivitas transaksi sangat kecil
3. Mendeteksi kemungkinan rekening transaksi bodong atau penyalahgunaan rekening
4. Membagi nasabah dalam kelompok tertentu untuk penawaran promosi

## Sumber Data

[kaggle.com/datasets/shivamb/bank-customer-segmentation](https://www.kaggle.com/datasets/shivamb/bank-customer-segmentation)

## Tentang Dataset

Dataset yang digunakan diperoleh dari [kaggle](https://www.kaggle.com/datasets/shivamb/bank-customer-segmentation) dan berisi lebih dari 1 juta transaksi dari 800 ribu nasabah sebuah bank di India.

## Fitur/ Kolom Data

- |                         |                         |
|-------------------------|-------------------------|
| 1. <i>TransactionID</i> | : id unik transaksi     |
| 2. <i>CustomerID</i>    | : id unik nasabah       |
| 3. <i>CustomerDOB</i>   | : tanggal lahir nasabah |
| 4. <i>CustGender</i>    | : jenis kelamin         |

5. *CustLocation* : lokasi nasabah
6. *CustAccountBalance* : jumlah saldo nasabah
7. *TransactionDate* : tanggal transaksi dilakukan
8. *TransactionTime* : waktu transaksi dilakukan
9. *TransactionAmount (INR)* : biaya transaksi (dalam India Rupee)

## Ukuran Dataset

1.048.567 baris data, 9 kolom fitur

## Preview Dataset

```
[7]: raw_data = pd.read_csv("bank_transactions.csv")
raw_data.sample(frac = 1).head()
```

	TransactionID	CustomerID	CustomerDOB	CustGender	CustLocation	CustAccountBalance	TransactionDate	TransactionTime	TransactionAmount (INR)
313104	T313105	C9018770	22/10/91	F	GURGAON	161.23	10/8/16	163943	3602.0
617525	T617526	C8816635	12/6/96	M	HOSHIARPUR	20643.59	26/8/16	134531	810.0
4801	T4802	C3439934	19/10/65	M	NEW DELHI	3293.25	22/9/16	145327	557.0
136431	T136432	C6216950	1/1/1800	M	NEW DELHI	446739.29	5/8/16	122615	1100.0
573848	T573849	C4731579	10/8/89	M	BURDWAN	9019.95	22/8/16	112342	150.0

## Preprocessing

### 1. Menghapus baris data yang berisi nilai "NaN"

Tahapan awal yang dilakukan adalah membersihkan dataset dari data-data yang tidak lengkap (NaN). Karena data yang hilang berada pada detail informasi terkait nasabah yang akan dipakai sebagai fitur yang penting, maka baris data yang bersifat NaN akan dibuang saja.

Before	After
<pre>[10]: # cek apakah ada dataset yang hilang raw_data.isna().sum()</pre> <pre>[10]: TransactionID      0 CustomerID      0 CustomerDOB    3397 CustGender     1100 CustLocation    151 CustAccountBalance  2369 TransactionDate      0 TransactionTime      0 TransactionAmount (INR)  0 dtype: int64</pre>	<pre>[12]: # Drop row(s) with nan value preprocessed_data = raw_data.dropna() display(preprocessed_data.isna().sum())</pre> <pre>TransactionID      0 CustomerID      0 CustomerDOB      0 CustGender       0 CustLocation      0 CustAccountBalance  0 TransactionDate    0 TransactionTime    0 TransactionAmount (INR)  0 dtype: int64</pre>
<pre>[8]: raw_data.shape</pre> <pre>[8]: (1048567, 9)</pre>	<pre>[13]: preprocessed_data.shape</pre> <pre>[13]: (1041614, 9)</pre>

### 2. Membuang baris data *invalid*

Pengecekan nilai unik dari masing-masing kolom sudah dilakukan dan ditemukan bahwa terdapat beberapa baris data yang nilainya tidak sesuai

(*invalid*), salah satunya terdapat nasabah dengan tahun lahir 1800 dan transaksi terakhir di tahun 2016, maka data tersebut juga akan dihapus dan menyisakan 985 ribuan data.

```
[15]: for data in preprocessed_data["CustomerDOB"].unique():
      date, month, year = data.split("/")
      if int(year) > 100: # remove more than two digit year
          print(data)
          preprocessed_data = preprocessed_data[preprocessed_data["CustomerDOB"] != data]

1/1/1800

[16]: preprocessed_data.drop(preprocessed_data[preprocessed_data['CustGender']=='T'].index, inplace=True)
preprocessed_data.shape

[16]: (985322, 9)
```

### 3. Format tipe data dan *encoding* fitur

```
Nama kolom: CustomerID (884265, object)
['C1010011' 'C1010012' 'C1010014' ... 'C9099919' 'C9099941' 'C9099956']
Nama kolom: CustomerDOB (17255, object)
['10/1/94' '4/4/57' '26/11/96' ... '18/7/65' '15/5/42' '24/10/44']
Nama kolom: CustGender (4, object)
['F' 'M' nan 'T']
Nama kolom: CustLocation (9356, object)
['JAMSHEDPUR' 'JHAJJAR' 'MUMBAI' ... 'KARANJIA'
 'NR HERITAGE FRESH HYDERABAD' 'IMPERIA THANE WEST']
Nama kolom: CustAccountBalance (161329, float64)
[0.00000000e+00 1.00000000e-02 3.00000000e-02 ... 6.97993296e+07
 8.22446299e+07 1.15035495e+08]
Nama kolom: TransactionDate (55, object)
['1/8/16' '1/9/16' '10/8/16' '10/9/16' '11/8/16' '11/9/16' '12/8/16'
 '12/9/16' '13/8/16' '13/9/16' '14/8/16' '14/9/16' '15/8/16' '15/9/16'
 '16/10/16' '16/8/16' '17/8/16' '18/8/16' '18/9/16' '19/8/16' '2/8/16'
 '2/9/16' '20/8/16' '21/10/16' '21/8/16' '22/8/16' '22/9/16' '23/8/16'
 '23/9/16' '24/8/16' '25/8/16' '25/9/16' '26/8/16' '26/9/16' '27/8/16'
 '27/9/16' '28/8/16' '29/8/16' '3/8/16' '3/9/16' '30/8/16' '30/9/16'
 '31/8/16' '4/8/16' '4/9/16' '5/8/16' '5/9/16' '6/8/16' '6/9/16' '7/8/16'
 '7/9/16' '8/8/16' '8/9/16' '9/8/16' '9/9/16']
Nama kolom: TransactionTime (81918, int64)
[ 0 1 2 ... 235957 235958 235959]
Nama kolom: TransactionAmount (INR) (93024, float64)
[0.00000000e+00 1.00000000e-02 2.00000000e-02 ... 9.91132220e+05
 1.38000288e+06 1.56003499e+06]
```

Kolom-kolom yang berisi nilai tanggal dan waktu masih bertipe '*object*' sehingga perlu dilakukan konversi ke format tanggal yang sesuai.

Untuk kolom jenis kelamin, perlu dilakukan *feature encoding* atau konversi ke nilai numerik agar dapat diproses oleh model.

```
[19]: # convert type of columns TransactionDate, CustomerDOB from string to datetime
preprocessed_data['TransactionDate'] = pd.to_datetime(preprocessed_data['TransactionDate'], format='%d/%m/%y', errors='coerce')
preprocessed_data['CustomerDOB'] = pd.to_datetime(preprocessed_data['CustomerDOB'], format='%d/%m/%y', errors='coerce')

# encode 'CustGender' to numeric format (F = 0, M = 1)
preprocessed_data['CustGender'] = preprocessed_data['CustGender'].map({'F': 0, 'M': 1})

# calculate 'CustomerAge' based on 'CustomerDOB'
preprocessed_data['CustomerAge'] = preprocessed_data['TransactionDate'].dt.year - preprocessed_data['CustomerDOB'].dt.year
preprocessed_data = preprocessed_data[preprocessed_data['CustomerAge'] > 0]

np.array(sorted(preprocessed_data['CustomerAge'].unique()))

[19]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
        35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47], dtype=int32)
```

Setelah format kolom sudah sesuai, kita dapat menghitung usia nasabah saat transaksi berdasarkan tanggal lahir dan tanggal transaksinya. Data kemudian di-filter lagi agar hanya tersisa nasabah dengan usia diatas 0 tahun (sisanya dianggap *invalid* untuk usia -1 dan dibawahnya).

#### 4. Membuat ringkasan transaksi per nasabah

Pengelompokkan data dilakukan untuk menggabungkan ringkasan transaksi berdasarkan ‘*CustomerID*’ (id unik nasabah), mulai dari jumlah transaksi yang dilakukan, total nominal transaksi, saldo awal, tanggal transaksi terakhir dilakukan, dan jangka waktu antar transaksi.

```
[48]: # Calculate transaction frequency, total, and average per customer
transaction_summary = preprocessed_data.groupby('CustomerID').agg({
    # 'TransactionAmount (INR)': ['count', 'sum', 'mean'],
    'TransactionAmount (INR)': ['count', 'sum'],
    'TransactionDate': ['min', 'max'],
}).reset_index()

# Flatten the MultiIndex columns
transaction_summary.columns = [
    'CustomerID',
    'TransactionCount',
    'TotalAmount',
    # 'AverageAmount',
    'FirstTransactionDate',
    'LastTransactionDate',
]

# No need to calculate Age cause it's already in the dataset
transaction_summary['Age'] = preprocessed_data.groupby('CustomerID')['CustomerAge'].first().values

# Add customer gender
transaction_summary['Gender'] = preprocessed_data.groupby('CustomerID')['CustGender'].first().values

# Add customer account balance
transaction_summary['AccBalance'] = preprocessed_data.groupby('CustomerID')['CustAccountBalance'].first().values

# Add recency (days since last transaction)
transaction_summary['Recency'] = transaction_summary['LastTransactionDate'].apply(
    lambda date: (datetime.now() - pd.to_datetime(date)).days
)

# Group by customer and calculate the date range
transaction_summary['DayRange'] = (transaction_summary['LastTransactionDate'] - transaction_summary['FirstTransactionDate']).dt.days
```

Tampilan ringkasan data dari masing-masing nasabah sebagai berikut:

```
[18]: # Check the results
transaction_summary.head(10)
```

	CustomerID	TransactionCount	TotalAmount	FirstTransactionDate	LastTransactionDate	Age	Gender	AccBalance	Recency	DayRange
0	C1010011	2	5106.0	2016-08-09	2016-09-26	24	0	32500.73	3003	48
1	C1010012	1	1499.0	2016-08-14	2016-08-14	22	1	24204.49	3046	0
2	C1010014	2	1455.0	2016-08-01	2016-08-07	24	0	38377.14	3053	6
3	C1010018	1	30.0	2016-09-15	2016-09-15	26	0	496.18	3014	0
4	C1010028	1	557.0	2016-08-29	2016-08-29	28	0	296828.37	3031	0
5	C1010031	2	1864.0	2016-08-03	2016-08-04	32	1	1754.10	3056	1
6	C1010035	2	750.0	2016-08-01	2016-08-27	24	1	7284.42	3033	26
7	C1010036	1	208.0	2016-08-26	2016-08-26	20	1	355430.17	3034	0
8	C1010037	1	19680.0	2016-08-09	2016-08-09	35	1	95859.17	3051	0
9	C1010038	1	100.0	2016-09-07	2016-09-07	24	0	1290.76	3022	0

## 5. Pemilihan fitur dan subset dataset

Beberapa percobaan dilakukan untuk memilih fitur yang paling relevan untuk mendeteksi transaksi mencurigakan (*anomalous/fraudulent/outlier*).

```
[19]: # Select features for clustering
columns = [
    'Age',
    # 'Gender',
    # 'DayRange',
    'AccBalance',
    # 'TransactionCount',
    'TotalAmount',
    # 'AverageAmount',
    # 'Recency',
]
features = transaction_summary[columns]
features.sample(frac = 1).head()
```

```
[19]:
```

	Age	AccBalance	TotalAmount
435550	40	86685.34	400.00
139609	21	3972.83	4200.00
592661	32	8897.50	201.73
465142	22	11719.55	90.00
534332	23	13232.97	400.00

Karena keterbatasan sumber daya, hanya 3% data (sekitar 23 ribuan data) yang akan digunakan untuk diproses oleh model.

```
[20]: # Process only 3% of data cause Lack of resource
features = features.sample(frac = 0.03, random_state = 1000)
features.head()
```

```
[22]: data_length = len(scaled_features)
data_length
```

```
[22]: 23988
```

## 6. Normalisasi (atau Standarisasi)

Normalisasi data dilakukan menggunakan salah satu *library* dari sklearn, yaitu `StandardScaler`, yang akan mentransformasi data dengan pengurangan dengan nilai rata-rata kemudian dibagi dengan standar deviasi dari masing-masing fitur secara independen.

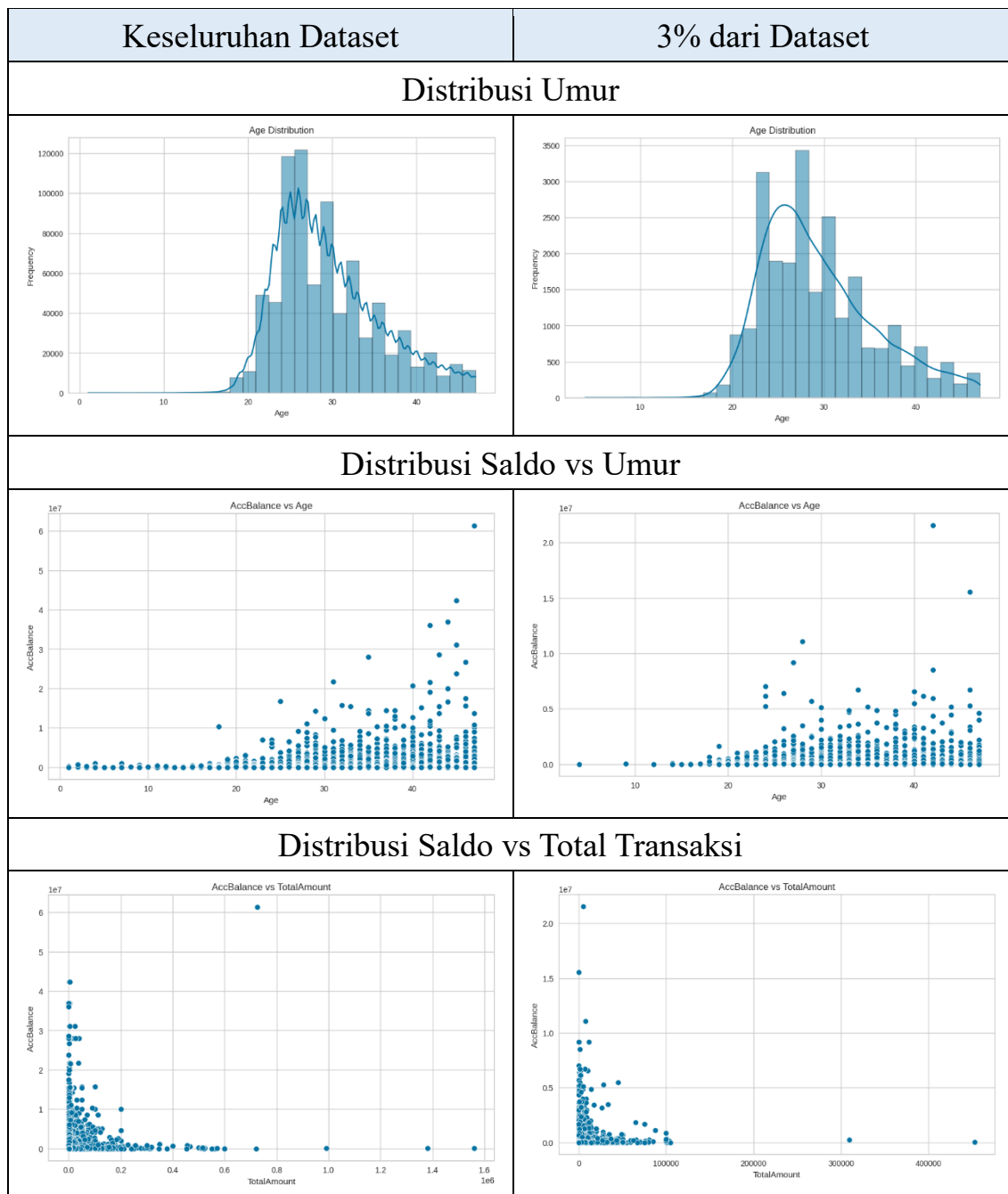
Metode ini cocok untuk fitur data dengan jangkauan terlalu jauh (misalnya '*TransactionCount*' dalam *range* 1-4 dengan '*TransactionAmount*' dengan nilai diatas 50.000), sehingga fitur dengan nilai yang terlalu besar tidak mendominasi fitur lainnya.

```
[20]: # Scale features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

scaled_features[:5]
```

```
[20]: array([[ -0.90450846, -1.61764198,  0.07265365,  1.98048597,  0.55887532],
        [-1.23043586,  0.61818376, -0.14032966, -0.39000265, -0.01252418],
        [-0.90450846, -1.61764198,  0.16512084,  1.98048597, -0.01949439],
        [-0.57858106, -1.61764198, -0.19294148, -0.39000265, -0.24523442],
        [-0.25265366, -1.61764198,  0.46465811, -0.39000265, -0.16175022]])
```

# Visualisasi Distribusi Subset Data



Dari grafik yang ditampilkan, subset data masih memiliki distribusi yang cukup mirip dengan dataset aslinya. Diharapkan dengan 3% subset data yang akan digunakan sudah dapat merepresentasikan keseluruhan dataset yang ada.

## Modelling

Data yang sudah bersih kemudian diteruskan ke model DBSCAN. Namun, karena keterbatasan sumber daya, model yang sudah dibangun dan akan dijalankan menggunakan server yang disediakan Mikroskil hanya mampu

memproses kurang dari 100.000 data, sehingga kami hanya memilih 3% data pertama (sekitar 20 ribu data) untuk di-*training* oleh model. Hal ini mungkin saja mempengaruhi hasil/ jumlah *cluster* yang terbentuk karena percobaan menggunakan 2% hingga 10% data secara acak memberikan perbedaan hasil *cluster* yang cukup signifikan.

## Pemilihan hyperparameter terbaik

*Hyperparameter* yang terdiri dari *eps* (*epsilon*; jarak maksimum 2 poin data agar dikatakan sebagai tetangga) dan *min\_samples* (jumlah data minimum untuk dapat membentuk *core point/ cluster*).

```
eps_range = np.arange(0.5, 3.5, 0.5) # Example range for eps
min_samples_range = np.arange(10, 30, 5) # Example range for min_samples

results = []

for eps in eps_range:
    for min_samples in min_samples_range:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        dbscan.fit(scaled_features)

        # Label cluster
        labels_dbscan = dbscan.labels_

        # menghitung jumlah elemen dari tiap cluster
        print(f'{eps}, {min_samples}; Done, Next", end=" --> ")
        if len(np.unique(labels_dbscan)) > 1: # Check if more than one cluster is found
            # Remove the 'ignore_index' argument as it's not supported by silhouette_score
            score = silhouette_score(scaled_features, labels_dbscan)
            results.append([eps, min_samples, score, np.unique(labels_dbscan, return_counts=True)])
        print("Finish.")

result_df = pd.DataFrame(results, columns=['eps', 'min_samples', 'silhouette_score', 'clusters'])
display(result_df)
```

0.5, 10; Done, Next --> 0.5, 15; Done, Next --> 0.5, 20; Done, Next --> 0.5, 25; Done, Next --> 1.0, 10; Done, Next --> 1.0, 15; Done, Next --> 1.0, 20; Done, Next --> 1.0, 25; Done, Next --> 1.5, 10; Done, Next --> 1.5, 15; Done, Next --> 1.5, 20; Done, Next --> 1.5, 25; Done, Next --> 2.0, 10; Done, Next --> 2.0, 15; Done, Next --> 2.0, 20; Done, Next --> 2.0, 25; Done, Next --> 2.5, 10; Done, Next --> 2.5, 15; Done, Next --> 2.5, 20; Done, Next --> 2.5, 25; Done, Next --> 3.0, 10; Done, Next --> 3.0, 15; Done, Next --> 3.0, 20; Done, Next --> 3.0, 25; Done, Next --> Finish.

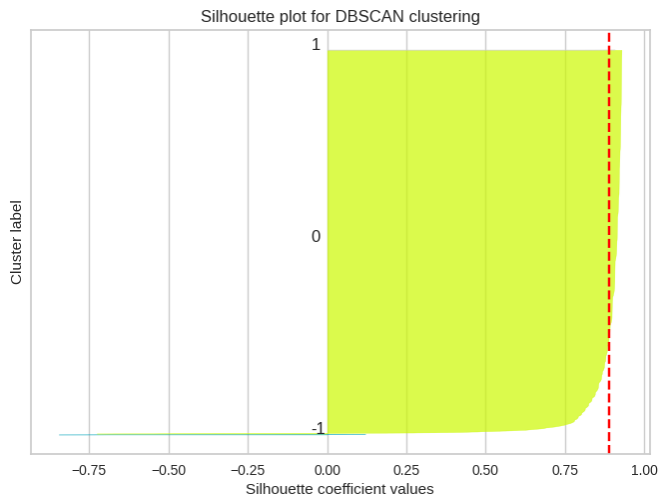
Pencarian *hyperparameter* terbaik untuk model DBScan dilakukan secara manual dengan *looping for*, kemudian dilakukan pemetaan hasil pada grafik garis untuk melihat nilai *silhouette* terbaik dari kombinasi *hyperparameter* yang sudah dibuat.

	eps	min_samples	silhouette_score	clusters
0	0.5	10	0.695793	([-1, 0, 1, 2, 3], [316, 23623, 31, 8, 10])
1	0.5	15	0.803552	([-1, 0], [425, 23563])
2	0.5	20	0.789556	([-1, 0], [505, 23483])
3	0.5	25	0.780932	([-1, 0], [561, 23427])
4	1.0	10	0.889953	([-1, 0], [126, 23862])
5	1.0	15	0.881662	([-1, 0], [150, 23838])
6	1.0	20	0.876581	([-1, 0], [164, 23824])
7	1.0	25	0.870994	([-1, 0], [181, 23807])
8	1.5	10	0.888455	([-1, 0, 1], [57, 23918, 13])
9	1.5	15	0.912663	([-1, 0], [78, 23910])
10	1.5	20	0.902984	([-1, 0], [99, 23889])
11	1.5	25	0.896720	([-1, 0], [116, 23872])
12	2.0	10	0.930488	([-1, 0], [41, 23947])
13	2.0	15	0.888391	([-1, 0, 1], [49, 23924, 15])
14	2.0	20	0.918941	([-1, 0], [67, 23921])
15	2.0	25	0.917989	([-1, 0], [69, 23919])
16	2.5	10	0.916101	([-1, 0, 1], [17, 23961, 10])
17	2.5	15	0.930769	([-1, 0], [41, 23947])
18	2.5	20	0.924035	([-1, 0], [56, 23932])
19	2.5	25	0.922682	([-1, 0], [59, 23929])
20	3.0	10	0.916025	([-1, 0, 1], [14, 23966, 8])
21	3.0	15	0.934758	([-1, 0], [33, 23955])
22	3.0	20	0.932207	([-1, 0], [39, 23949])
23	3.0	25	0.927700	([-1, 0], [48, 23940])



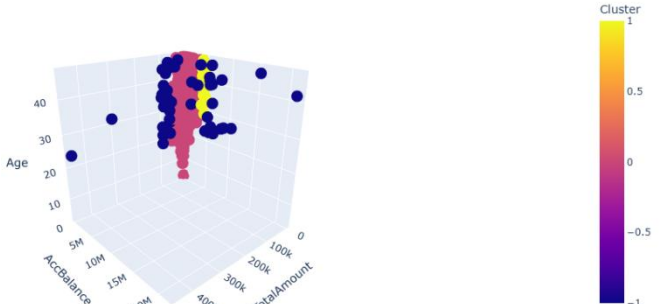
## Evaluasi Model

Kami memilih kombinasi  $eps=2.0$  dan  $min\_samples=15$  dengan nilai *silhouette score* sebesar 0.888 untuk digunakan pada model karena menghasilkan *cluster* yang lebih seimbang.

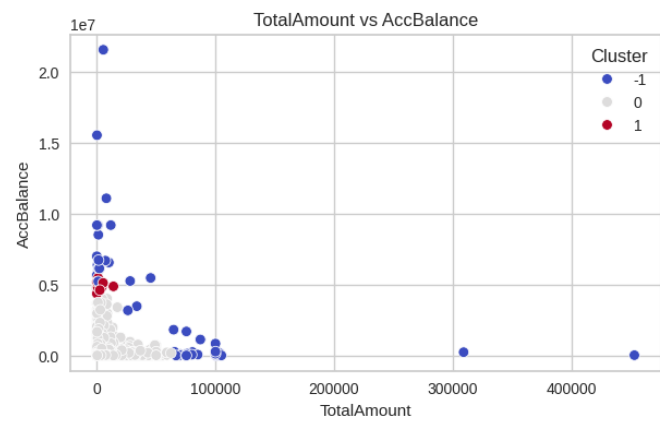
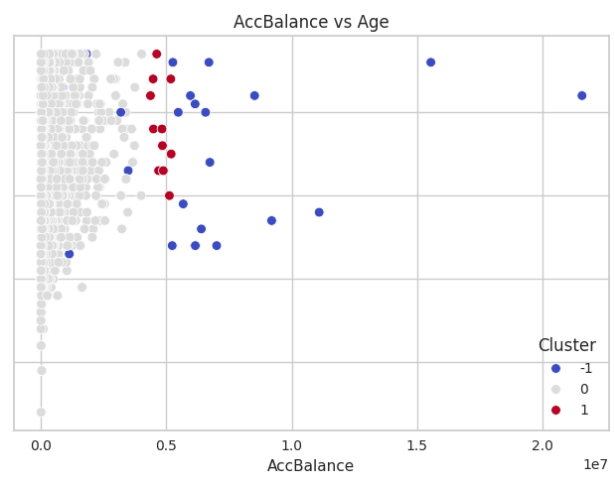
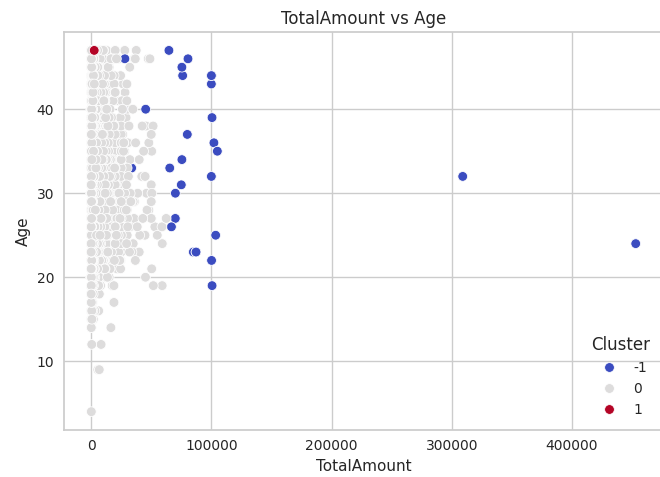
<i>Silhouette Score</i>	<pre>[30]: display(np.unique(labels_dbscan, return_counts=True)) silhouette_score(scaled_features, labels_dbscan)  (array([-1,  0,  1]), array([ 49, 23924,  15])) [30]: np.float64(0.8883913099110776)  Cluster -1: Min=-0.85, Avg=-0.31, Max=0.12 Cluster 0: Min=-0.73, Avg=0.89, Max=0.93 Cluster 1: Min=0.81, Avg=0.87, Max=0.91</pre>
<i>Silhouette Plot</i>	

Visualisasi *cluster* yang terbentuk berdasarkan 3 fitur yang dipilih, yaitu umur, saldo, dan total transaksi menunjukkan bahwa data *outliers* (dalam *cluster* -1; ditunjukkan oleh *dot* berwarna biru) cenderung terpisah terlalu jauh dari data aktivitas transaksi yang normal. Sedangkan data yang masuk ke *cluster* 1 (ditunjukkan oleh *dot* berwarna kuning) merupakan data transaksi yang dicurigai (*suspicious*) namun masih belum masuk ke data anomali.

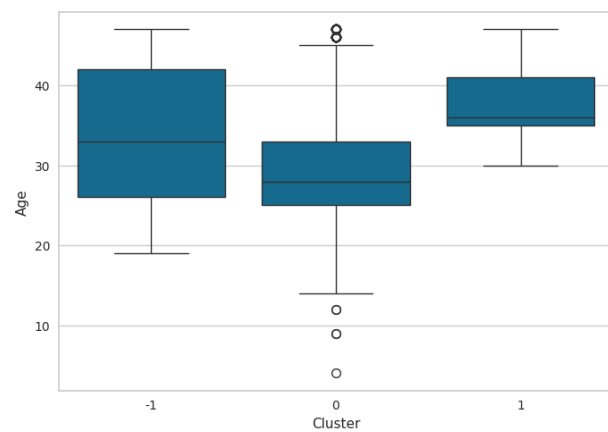
Untuk visualisasi *cluster* juga dibuat dalam 2D agar pembentukkan *cluster* lebih mudah dilihat.

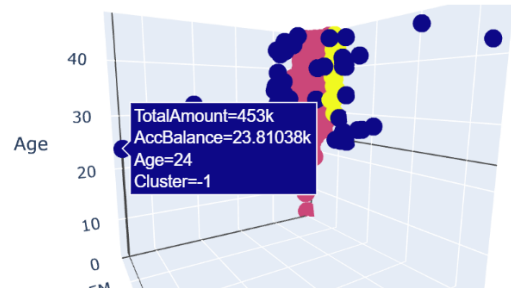
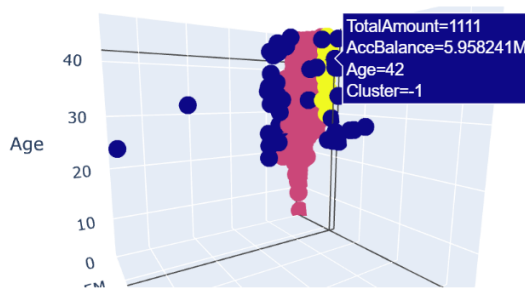
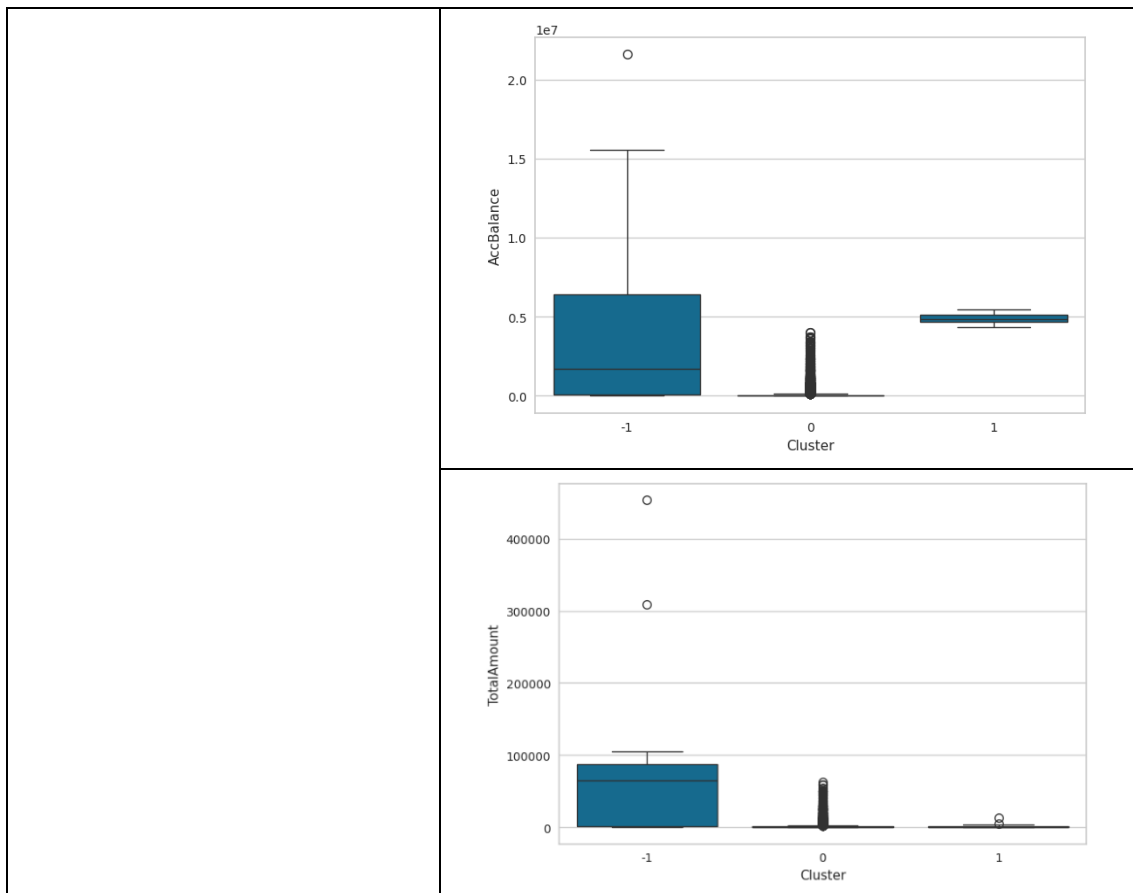
Visualisasi <i>Cluster</i> (3D)	
---------------------------------	--

## Visualisasi *Cluster* (2D)



## Distribusi data tiap *Cluster*





Contoh data transaksi nasabah yang dianggap anomali ditunjukkan pada gambar diatas. Pada gambar sebelah kiri, nasabah berumur 42 tahun dengan saldo 5M hanya bertransaksi sebanyak 1.100 INR dalam waktu 2 bulan, dan juga pada gambar sebelah kanan, nasabah berumur 24 tahun dengan saldo 23k bertransaksi sebesar 453k INR dan ini bukan merupakan angka yang wajar sehingga masuk ke dalam *cluster* -1 sebagai data *outlier* atau *fraudulent*.

## Tantangan yang Dihadapi

### 1. Keterbatasan Sumber Daya

Ketika proses *training* dilakukan dengan seluruh data yang sudah di-*preprocessing* (sekitar 700 ribuan data), server *crash* seketika. Setelah dicari tahu, ternyata batasan jumlah data yang dapat diproses oleh model DBSCAN hanyalah 20.000 data, sehingga jumlah *cluster* yang terbentuk juga akan bergantung dari data yang dimasukkan.

### 2. Mencari Jumlah *Cluster* Terbaik

Proses pencarian *hyperparameter* terbaik agar mendapatkan nilai *silhouette score* terbaik namun juga tetap mempertahankan jumlah *cluster* yang tidak terlalu banyak dan juga tidak terlalu sedikit, membutuhkan waktu yang cukup lama karena di-*looping* secara manual. Setelah nilai *silhouette score* didapatkan, ternyata nilai terbaik ada pada pembagian *cluster outliers* dari *cluster* data normal saja. Sehingga diputuskan untuk mengganti judul dan tujuan proyek dari “Segmentasi Nasabah Berdasarkan Transaksi” menjadi “Deteksi Aktivitas Transaksi Anomali”.

## Kesimpulan

Algoritma DBSCAN cocok digunakan untuk memisahkan data *outliers* berupa data aktivitas transaksi anomali pada 3% subset dataset yang digunakan. Dengan *cluster* yang terbentuk, diharapkan dapat membantu bank untuk menemukan rekening nasabah yang dicurigai melakukan aktivitas transaksi tidak wajar.

## Link PPT

[Akses slide presentasi GCD - Clustering with DBSCAN disini](#)

## Link Notebook

[Akses notebook GCD - Clustering with DBSCAN disini](#)