

**Tugas Besar 2**  
**IF3070 Dasar Intelelegensi Artifisial**  
**Semester I 2025/2026**

*Machine Learning - Fraud Detection*

**Oleh Tim AbyuDAIya Ganbatte,**

Wisa Ahmaduta Dinutama	18223003
Persada Ramiiza Abyudaya	18223033
Inggried Amelia Deswanyt	18223035
Wijaksara Aptaluhung	18223088



**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2025**

## **DAFTAR ISI**

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I</b>	
<b>PENDAHULUAN &amp; SPESIFIKASI.....</b>	<b>4</b>
1. Pendahuluan.....	4
2. Tujuan.....	6
3. Spesifikasi.....	6
3.1. Algoritma Wajib.....	6
3.2. Algoritma Pilihan.....	6
<b>BAB II</b>	
<b>ANALISIS DATA EKSPLORASI (EDA).....</b>	<b>8</b>
<b>DATA UNDERSTANDING.....</b>	<b>8</b>
1. Pendahuluan.....	8
2. Part 1: Data Structure & Initial Characteristics.....	8
2.1. Data Size.....	8
2.2. Data Type & Information.....	8
2.2.1. Informasi Data.....	8
2.2.2. Categorical Features.....	9
2.2.3. Numerical Features.....	9
2.3. Statistik Deskriptif.....	9
3. Part 2: Data Integrity Evaluation.....	9
<b>BAB III</b>	
<b>DATA SPLITTING.....</b>	<b>10</b>
1. Dataset Splitting Methodology.....	10
1.1. Split Training Set.....	10
1.2. Validation Set.....	10
1.3. Testing Set.....	10
<b>BAB IV</b>	

**DATA CLEANING & PREPROCESSING..... 11**

1. Data Cleaning.....	11
1.1. Handling Missing Data.....	11
1.2. Dealing with Outliers.....	11
1.3. Data Validation.....	11
1.4. Removing Duplicates.....	11
1.5. Feature Engineering.....	11

**BAB V**

**DESKRIPSI IMPLEMENTASI..... 12**

1. Algoritma Decision Tree Learning (DTL) from Scratch.....	12
1.1. Penjelasan Teori.....	12
1.2. Implementasi Algoritma (Tanpa Library).....	13
1.3. Langkah Implementasi.....	13
1.4. Hasil Implementasi.....	14
2. Algoritma Logistic Regression from Scratch.....	14
2.1. Penjelasan Teori.....	14
2.2. Implementasi Algoritma (Tanpa Library).....	14
2.3. Langkah Implementasi.....	14
2.4. Hasil Implementasi.....	14
3. Algoritma K-Nearest Neighbor (KNN) from Scratch.....	14
3.1. Penjelasan Teori.....	14
3.2. Implementasi Algoritma (Tanpa Library).....	14
3.3. Langkah Implementasi.....	15
3.4. Hasil Implementasi.....	15

**BAB VI**

**KESIMPULAN & SARAN..... 16**

1. Kesimpulan.....	16
2. Saran.....	16

## BAB I

### PENDAHULUAN & SPESIFIKASI

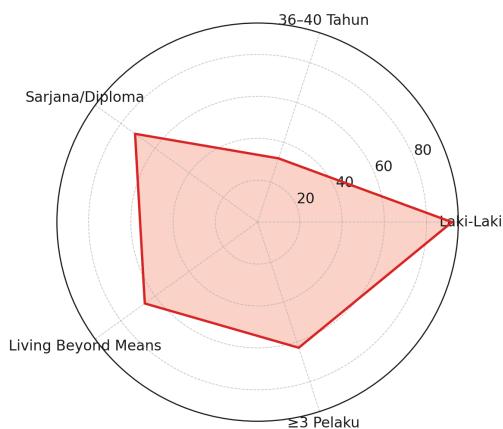
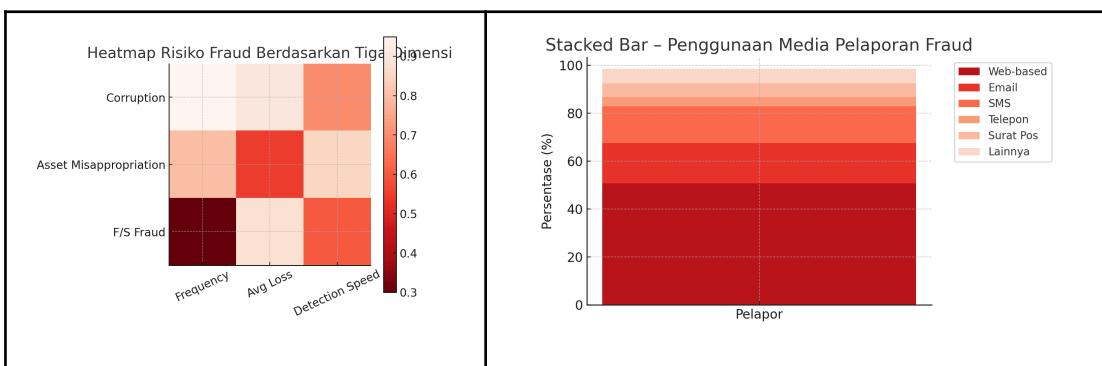
#### 1. Pendahuluan

*Fraud* terus menjadi ancaman bagi integritas sektor publik maupun swasta di Indonesia. Upaya pencegahan dan pemberantasan *fraud* tidak bisa hanya mengandalkan intuisi atau tindakan sporadis. Survei Fraud Indonesia 2025, yang diselenggarakan oleh ACFE Indonesia Chapter mengadopsi pendekatan dari Report to the Nations (RTTN) yang diterbitkan oleh ACFE Global, dan bertujuan untuk memetakan tren *fraud* di Indonesia. Berdasarkan hasil survei, bentuk kecurangan yang paling sering terjadi adalah korupsi sebesar 47,6%, diikuti oleh penyalahgunaan aset sebesar 40,2% serta kecurangan laporan keuangan sebesar 12,2%. Sebanyak 38,13% responden menyatakan bahwa pelaku melakukan lebih dari satu skema *fraud* secara bersamaan. Berbeda dengan tren global dalam Report to the Nations, di mana *asset misappropriation* lebih dominan, hasil ini menunjukkan bahwa korupsi masih menjadi tantangan terbesar di Indonesia (ACFE Indonesia Chapter, 2025).

Terdapat *financial statement fraud* yang berdiri sendiri hanya terjadi pada 1,44% kasus. Pada kategori *asset misappropriation*, sub-skema dengan risiko tertinggi adalah *expense reimbursement* (39,33%), disusul *skimming* (19,90%) dan *billing* (14,39%). *Heatmap* menunjukkan bahwa korupsi memiliki risiko tertinggi dengan frekuensi 47,6%, kerugian rata-rata hingga kategori Rp1–50 miliar (26%), dan waktu deteksi 31,18–35,01%, dibandingkan *asset misappropriation* (40,2%; rugi terbesar 19%; deteksi 41,73%) serta *financial statement fraud* (12,2%; rugi 23,74%; deteksi 27,78–33,57%). *Stacked bar chart* menunjukkan bahwa sistem pelaporan berbasis *web* digunakan pada 50,8% kasus, disusul *email* 16,8%, SMS 15,3%, telepon 3,8%, *surat pos* 5,8%, dan media lainnya 6%.

**Tabel 1.1. Insight Survei Fraud**

Korelasi Risiko <i>Fraud</i>	Penggunaan Media Pelaporan <i>Fraud</i>
------------------------------	---



mengaku khawatir terhadap publisitas yang buruk, 22% menilai proses hukum memakan biaya besar, dan sisa 13% berpendapat bahwa sanksi internal sudah memadai. Pada akhirnya, tingkat pemulihan kerugian akibat *fraud* sebagian besar hanya parsial (70%), sementara institusi yang mampu memulihkan kerugian secara penuh berjumlah 19%, dan 11% tidak dapat memulihkan kerugian sama sekali. Kondisi ini menunjukkan bahwa penanganan *fraud* yang terlambat atau tidak tuntas berdampak langsung pada rendahnya tingkat pemulihan kerugian.

Hal ini menandakan bahwa *fraud detection* bersifat sangat krusial karena penanganan *fraud* yang terlambat atau tidak tuntas berdampak langsung pada rendahnya tingkat pemulihan kerugian. Sistem deteksi yang masih menggunakan teknik konvensional tidak cukup cepat ataupun konsisten untuk mengidentifikasi pola kecurangan yang semakin kompleks dan melibatkan banyak pelaku. Oleh karena itu dibutuhkan sistem *fraud detection* yang lebih akurat, salah satunya dengan mengimplementasikan *machine learning* atau pembelajaran mesin. Dengan deteksi yang lebih cepat dan presisi yang lebih tinggi, organisasi memiliki peluang lebih besar untuk mencegah kerugian membesar.

## 2. Tujuan

Tugas Besar 2 pada kuliah IF3070 Dasar Inteligensi Buatan bertujuan untuk memberikan pengalaman langsung kepada peserta kuliah dalam menerapkan algoritma pembelajaran mesin pada permasalahan nyata. Dalam tugas besar ini, permasalahan tersebut merupakan permasalahan *fraud* yang kini menjadi ancaman bagi masyarakat.

## 3. Spesifikasi

Dalam tugas besar ini, tugas utama dari tugas besar ini adalah pembelajaran mesin merupakan salah satu cabang dari kecerdasan buatan yang memungkinkan sistem untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit. *Dataset* yang digunakan dalam tugas besar ini adalah kumpulan data tentang mahasiswa yang terdaftar dalam berbagai program sarjana yang ditawarkan oleh perguruan tinggi.

### 3.1. Algoritma Wajib

Dalam tugas besar ini, terdapat algoritma pembelajaran mesin yang wajib diterapkan, ialah *Decision Tree Learning (DTL) from scratch*. Terdapat dua algoritma *Decision Tree Learning (DTL)* yang dapat dipilih antara lain,

1. C4.5
2. Classification and Regression Tree (CART)

Algoritma *Decision Tree Learning (DTL)* yang diimplementasikan seminimal mungkin harus dapat menangani tipe data numerik dan *categorical* serta dapat menangani *null values*.

### 3.2. Algoritma Pilihan

Adapun dua algoritma pilihan yang dapat diimplementasikan, yaitu *Logistic Regression* atau *K-Nearest Neighbour (KNN) from scratch*.

1. Terapkan algoritma *Logistic Regression from scratch* yang sama dengan algoritma yang telah diajarkan di kelas. Algoritma optimasi model selain yang diajarkan di kelas akan dihitung sebagai bonus.
2. Implementasikan *K-Nearest Neighbour (KNN) from scratch*. Algoritma optimasi model selain yang diajarkan di kelas akan dihitung sebagai bonus.

Implementasi *Decision Tree Learning (DTL)* dan *Logistic Regression* atau *K-Nearest Neighbour (KNN)* menggunakan *scikit-learn*. Bandingkan hasil dari algoritma *from scratch* dan algoritma *scikit-learn*. Untuk ID3 di *scikit-learn*, gunakan *DecisionTreeClassifier* dengan parameter *criterion='entropy'*. Untuk C4.5 di *scikit-learn*, gunakan *DecisionTreeClassifier* dengan parameter *criterion='entropy'* dan tambahkan *ccp\_alpha*. Untuk CART di *scikit-learn*, gunakan *DecisionTreeClassifier* dengan parameter *criterion='gini'*. Model yang diimplementasi harus dapat di-*save* dan di-*load*. Sementara itu, implementasinya dibebaskan, seperti .txt, .pkl, dan lain-lain. Implementasi *Decision Tree Learning (DTL)*, *Logistic Regression*, dan *K-Nearest Neighbour (KNN)* *from scratch* bisa dalam bentuk kelas-kelas (class KNN, dst.) yang nantinya akan di-*import* ke *notebook* penggerjaan. Untuk implementasi *from scratch*, *library* yang boleh digunakan adalah untuk perhitungan matematika saja seperti numpy dan sejenisnya.

## **BAB II**

### **ANALISIS DATA EKSPLORASI (EDA)**

#### ***DATA UNDERSTANDING***

#### **1. Pendahuluan**

*Exploratory Data Analysis* atau Analisis Eksplorasi Data (EDA) adalah langkah penting dalam proses analisis data yang melibatkan pemeriksaan dan visualisasi kumpulan data untuk menemukan pola, tren, anomali, dan berbagai wawasan. EDA merupakan tahap awal sebelum menerapkan teknik statistik dan *machine learning* atau pembelajaran mesin yang lebih lanjut. EDA digunakan untuk mendapatkan pemahaman mendalam mengenai data yang sedang dikerjakan, sehingga memungkinkan untuk membuat keputusan yang lebih terinformasi serta merumuskan hipotesis untuk analisis berikutnya.

#### **2. Part 1: Data Structure & Initial Characteristics**

##### **2.1. Data Size**

Dengan menggunakan `df.shape`, didapatkan bahwa *dataset train.csv* ini memiliki ukuran sebesar 100.000 *instances* dengan total 31 fitur.

##### **2.2. Data Type & Information**

###### **2.2.1. Informasi Data**

Berdasarkan *output* berikut yang menggunakan `df.info()`, didapatkan bahwa *train.csv* memiliki *DataFrame pandas* yang berisi 100.000 baris data dengan 31 kolom. Rentang indeks dimulai dari 0 hingga 99.999, yang menunjukkan bahwa tidak ada baris yang hilang. Setiap kolom dideskripsikan berdasarkan jumlah nilai non-null serta tipe datanya. Dari 31 fitur yang tersedia, terdapat rincian sebagai berikut.

**Tabel 2.1.** Tipe Data Fitur

No.	Tipe Kolom	Jumlah
-----	------------	--------

1.	<i>Integer</i>	15
2.	<i>Float</i>	9
3.	<i>Object</i>	7

Hampir seluruh kolom memiliki nilai lengkap (100.000 non-null), kecuali dua kolom yaitu *transaction\_amount* dan *avg\_transaction\_amount* yang masing-masing memiliki 97.581 nilai non-null. Terdapat *label fraud (is\_fraud)* sebagai target analisis. Berikut merupakan tipe data dari setiap fitur dalam DataFrame.

**Tabel 2.2.** Tipe Data Fitur

<i>int64</i>	<i>float64</i>	<i>object</i>
ID	transaction_amount	transaction_id
user_id	transaction_duration	gender
age	avg_transaction_amount	country
time_of_day	std_transaction_amount	device_type
day_of_week	ip_risk_score	device_os
num_prev_transactions	device_trust_score	merchant_category
transactions_last_24h	merchant_risk	transaction_type
transactions_last_1h	country_risk	
failed_login_attempts	distance_from_home	

account_age_days	
has_chargeback_history	
shared_ip_users	
shared_device_users	
is_new_country	
is_fraud	

### 2.2.2. *Categorical Features*

Dalam dataset *train.csv*, terdapat beberapa kolom numerik sebenarnya bersifat kategorikal karena hanya memiliki nilai biner (0/1/NaN), yaitu *has\_chargeback\_history*, *is\_new\_country*, dan *is\_fraud*. Setelah kolom biner digabung dengan kolom kategorikal lain, jumlah nilai unik tiap fitur kategorikal dicek (termasuk 0/1/NaN yang terdeteksi).

**Tabel 2.3.** Number of Unique Values for Categorical Features

No.	Feature	Unique Value
1	has_chargeback_history	2
2	is_new_country	2
3	is_fraud	2
4	transaction_id	100000
5	gender	2
6	country	10
7	device_type	3

8	device_os	5
9	merchant_category	9
10	transaction_type	4

### 2.2.3. Numerical Features

Berikut ini merupakan kolom numerik sebagaimana yang tertera pada tabel berikut ini beserta *unique values*-nya.

**Tabel 2.4.** Number of Unique Values for Numerical Features

No.	Feature	Unique Value
1	user_id	28918
2	age	62
3	transaction_amount	28920
4	time_of_day	24
5	day_of_week	7
6	transaction_duration	2448
7	num_prev_transactions	49
8	avg_transaction_amount	14709
9	std_transaction_amount	7501
10	transactions_last_24h	45
11	transactions_last_1h	16
12	failed_login_attempts	6
13	ip_risk_score	1001

14	device_trust_score	1001
15	account_age_days	2999
16	shared_ip_users	12
17	shared_device_users	11
18	merchant_risk	8
19	country_risk	7
20	distance_from_home	30865

### 2.3. Statistik Deskriptif

Statistika deskriptif adalah metode-metode yang berkaitan dengan pengumpulan dan penyajian suatu himpunan data sehingga memberikan informasi yang berguna. Berikut merupakan statistik deskriptif dari *train.csv* secara lengkap.

**Tabel 2.5.** Tabel Statistik Deskriptif Lengkap

stat	ID	user_id	age	transaction_amount	time_of_day	day_of_week	transaction_duration	num_prev_transactions
count	100000	100000	100000	97581	100000	100000	100000	100000
mean	49999.5	15040.17318	48.48415	126.855288	12.77016	2.7807	3.999587	29.99208
std	28867.6578	8655.074007	17.892168	376.035544	5.748626	2.064754	4.012831	5.48621
min	0	1	18	0.01	0	0	0	8
25%	24999.75	7549	33	26.91	8	1	1.15	26
50%	49999.5	15093	48	64.46	12	3	2.76	30
75%	74999.25	22541	64	131.06	18	5	5.54	34
max	99999	29999	79	20601.13335	23	6	52.07	59
stat	avg_transaction_amount	std_transaction_amount	transactions_last_24h	transactions_last_1h	failed_login_attempts	ip_risk_score	device_trust_score	account_age_days
count	97581	100000	100000	100000	100000	97581	100000	
mean	69.969722	42.541209	2.04015	0.30923	0.29811	0.500395	0.49948	1497.32554
std	29.96153	21.652791	3.967414	1.46423	0.546208	0.288944	0.288812	866.004305
min	-52.89	5	0	0	0	0	0	1
25%	49.78	23.86	1	0	0	0.25	0.249	745
50%	69.99	42.48	1	0	0	0.501	0.499	1496
75%	90.25	61.31	2	0	1	0.751	0.75	2244
max	195.79	80	44	15	5	1	1	2999
stat	has_chargeback_history	shared_ip_users	shared_device_users	merchant_risk	country_risk	distance_from_home	is_new_country	is_fraud
count	100000	100000	100000	100000	100000	100000	100000	100000
mean	0.60627	2.00077	1.4998	0.19298	0.071324	117.411584	0.04958	0.14128
std	0.237988	1.414676	1.225885	0.134019	0.033203	212.016312	0.217077	0.348312
min	0	0	0	0.05	0.02	0	0	0
25%	0	1	1	0.08	0.04	29.62	0	0
50%	0	2	1	0.12	0.05	71.81	0	0
75%	0	3	2	0.3	0.1	144.44	0	0
max	1	11	11	0.55	0.13	6388.517628	1	1

Berikut merupakan rangkuman utama dari statistik-statistik yang tertera pada tabel statistik deskriptif secara lengkap.

**Tabel 2.6.** Tabel *Summary* Statistik Deskriptif

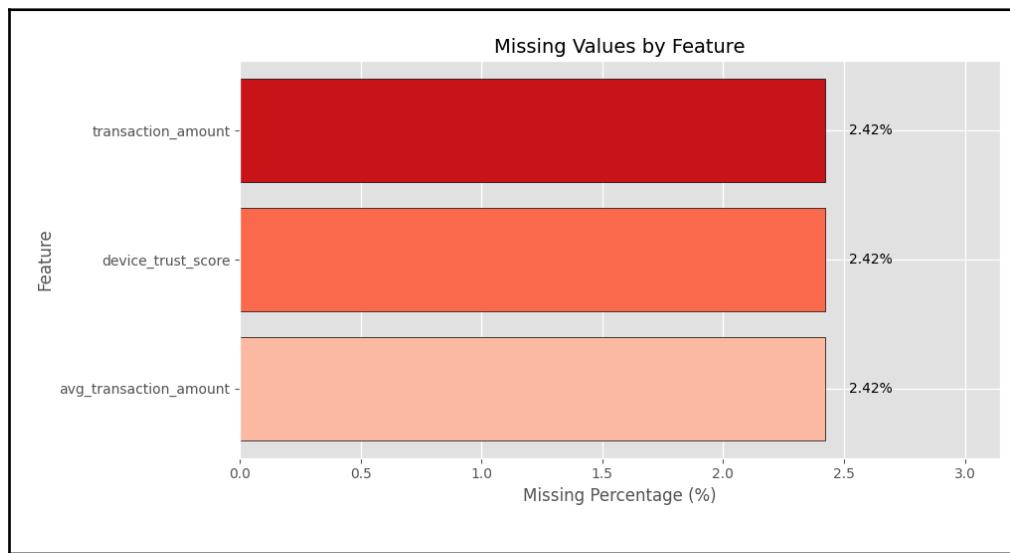
Feature	min	max	mean	median	std
---------	-----	-----	------	--------	-----

1	1	29999	15040.17318	15093	8655.074007
2	18	79	48.48415	48	17.892168
3	0.01	20601.13335	126.855288	64.46	376.035544
4	0	23	12.77016	12	5.748626
5	0	6	2.7807	3	2.064754
6	0	52.07	3.999587	2.76	4.012831
7	8	59	29.99208	30	5.48621
8	-52.89	195.79	69.969722	69.99	29.96153
9	5	80	42.541209	42.48	21.652791
10	0	44	2.04015	1	3.967414
11	0	15	0.30923	0	1.46423
12	0	5	0.29811	0	0.546208
13	0	1	0.500395	0.501	0.288944
14	0	1	0.49948	0.499	0.288812
15	1	2999	1497.32554	1496	866.004305
16	0	11	2.00077	2	1.414676
17	0	11	1.4998	1	1.225885
18	0.05	0.55	0.19298	0.12	0.134019
19	0.02	0.13	0.071324	0.05	0.033203
20	0	6388.517628	117.411584	71.81	212.016312

### 3. Part 2: Data Integrity Evaluation

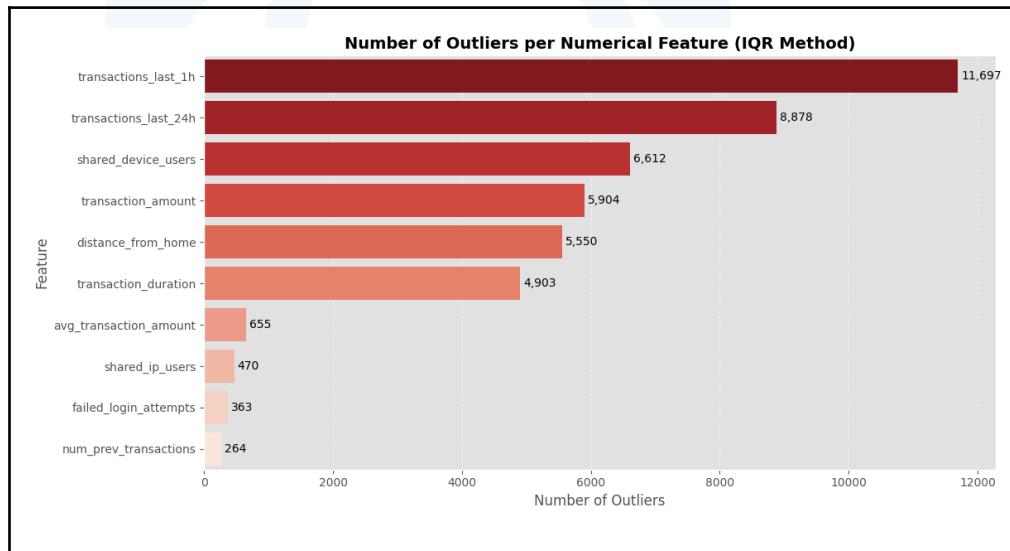
#### 3.1. Missing Values

Terdapat 3 fitur dengan *missing values* yang berarti data dari 3 kolom ini tidak lengkap. Terdapat 7.257 sel atau entri yang kosong (*missing entries*) di ketiga kolom tersebut. Sehingga, jika dijumlahkan dari seluruh baris untuk ketiga kolom tersebut, ada 7.257 data yang tidak tersedia.



Gambar 2.1. Missing Values by Feature

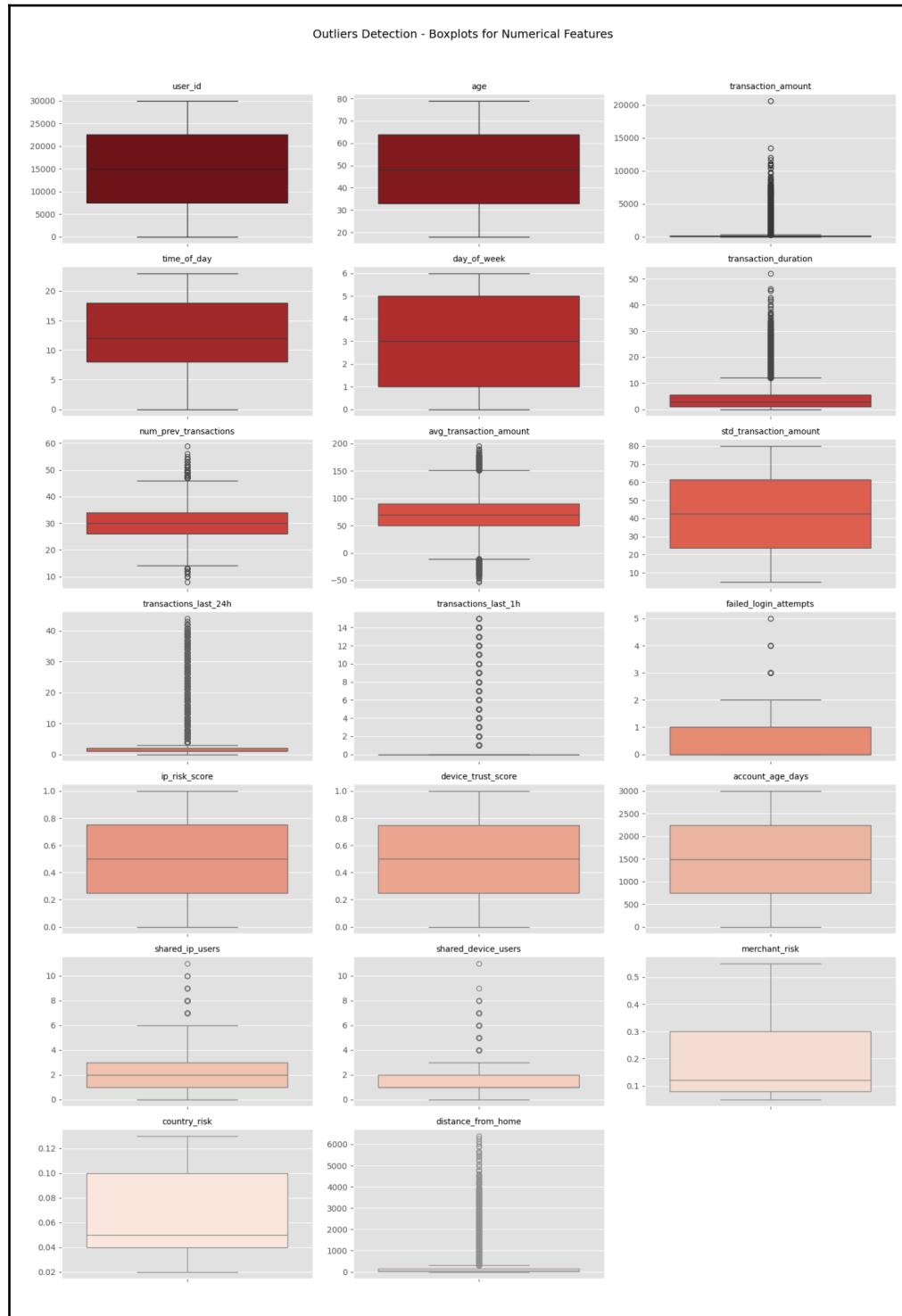
### 3.2. Outliers



Gambar 2.2. Number of Outliers per Numerical Feature

Terdapat 10 fitur dengan *outliers* dan *total outlier entries* sejumlah 45.296. Berdasarkan hasil *plot* untuk *outliers*, didapatkan bahwa hampir semua fitur numerik yang berkaitan dengan aktivitas transaksi menunjukkan *outlier* besar. Didapatkan juga bahwa aktivitas transaksi memiliki *outliers* yang mencolok. Banyak *user* yang memiliki jumlah transaksi sangat tinggi (*last\_24h*, *last\_1h*) dan *num\_prev\_transactions* juga memiliki *outlier* skala besar. Adapun *shared\_ip\_users* dan *shared\_device\_users* memiliki *outlier*

besar yang dapat menjadi indikasi akun *fraud*. Berikut merupakan visualisasi *boxplot* dari fitur numerikal.



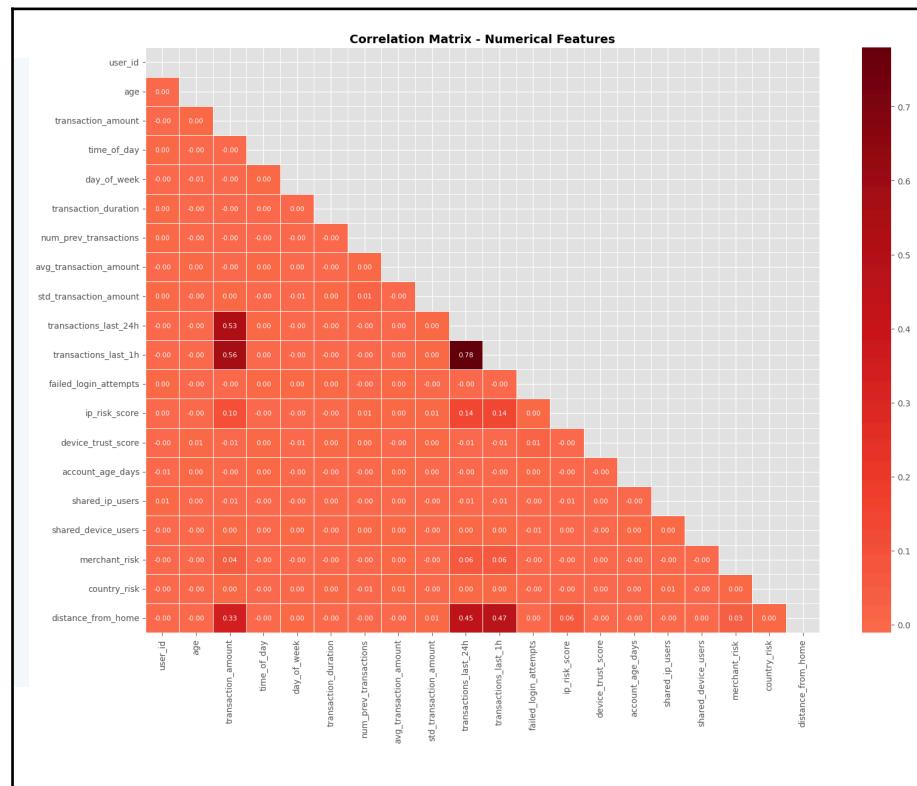
**Gambar 2.3. Outliers Detectors Boxplot (Numerical Features)**

#### 4. Part 3: Feature Exploration & Relationship Analysis

##### 4.1. Korelasi Fitur

###### 4.1.1. Numerical Features Correlation

Terdapat 2 jenis korelasi, yaitu korelasi positif dan korelasi negatif dari fitur-fitur numerikal. Angka korelasi positif berarti ketika satu variabel naik, maka variabel lain akan cenderung ikut naik juga. Sementara itu, angka korelasi negatif berarti ketika satu variabel naik, maka variabel lain akan cenderung turun. Berikut merupakan *heatmap* yang sudah di-mask agar lebih jelas dan akurat.



**Gambar 2.4. Heatmap Correlation Matrix**

Jumlah transaksi yang terjadi dalam 1 jam terakhir sangat berkaitan dengan jumlah transaksi 24 jam terakhir. Hal ini dikarenakan transaksi dalam 1 jam adalah bagian dari transaksi dalam 24 jam. Kemudian, nominal transaksi juga berkorelasi dengan jumlah transaksi di mana semakin sering bertransaksi, semakin besar total nominal. Jarak dari rumah juga berkaitan dengan jumlah transaksi, di mana

pembelian jauh dari lokasi rumah sering terjadi saat aktivitas transaksi meningkat. Hal ini dapat menjadi indikasi terjadinya *fraud*.

**Tabel 2.7.** Korelasi Positif (10 Besar)

Variabel 1	Variabel 2	Korelasi
transactions_last_24h	transactions_last_1h	0.779558
transactions_last_1h	transactions_last_24h	0.779558
transaction_amount	transactions_last_1h	0.563724
transactions_last_1h	transaction_amount	0.563724
transactions_last_24h	transaction_amount	0.526270
transaction_amount	transactions_last_24h	0.526270
distance_from_home	transactions_last_1h	0.468482
transactions_last_1h	distance_from_home	0.468482
distance_from_home	transactions_last_24h	0.446735
transactions_last_24h	distance_from_home	0.446735

Sementara itu, angka korelasi negatif yang muncul sangat kecil dengan nominal mendekati 0. Hal ini berarti bahwa hampir tidak ada hubungan antara kedua pasangan variabel tersebut. Adapun beberapa skor, seperti skor kepercayaan *device* yang sedikit menurun saat jumlah transaksi meningkat. Kemudian, jumlah pengguna perangkat bersama dan *failed login* juga tidak punya hubungan yang signifikan.

**Tabel 2.8.** Korelasi Negatif (10 Besar)

Variabel 1	Variabel 2	Korelasi
transactions_last_1h	device_trust_score	-0.007086
device_trust_score	transactions_last_1h	-0.007086

shared_ip_users	transactions_last_1h	-0.008645
transactions_last_1h	shared_ip_users	-0.008645
transactions_last_24h	device_trust_score	-0.008680
device_trust_score	transactions_last_24h	-0.008680
account_age_days	user_id	-0.008871
user_id	account_age_days	-0.008871
failed_login_attempts	shared_device_users	-0.011030
shared_device_users	failed_login_attempts	-0.011030

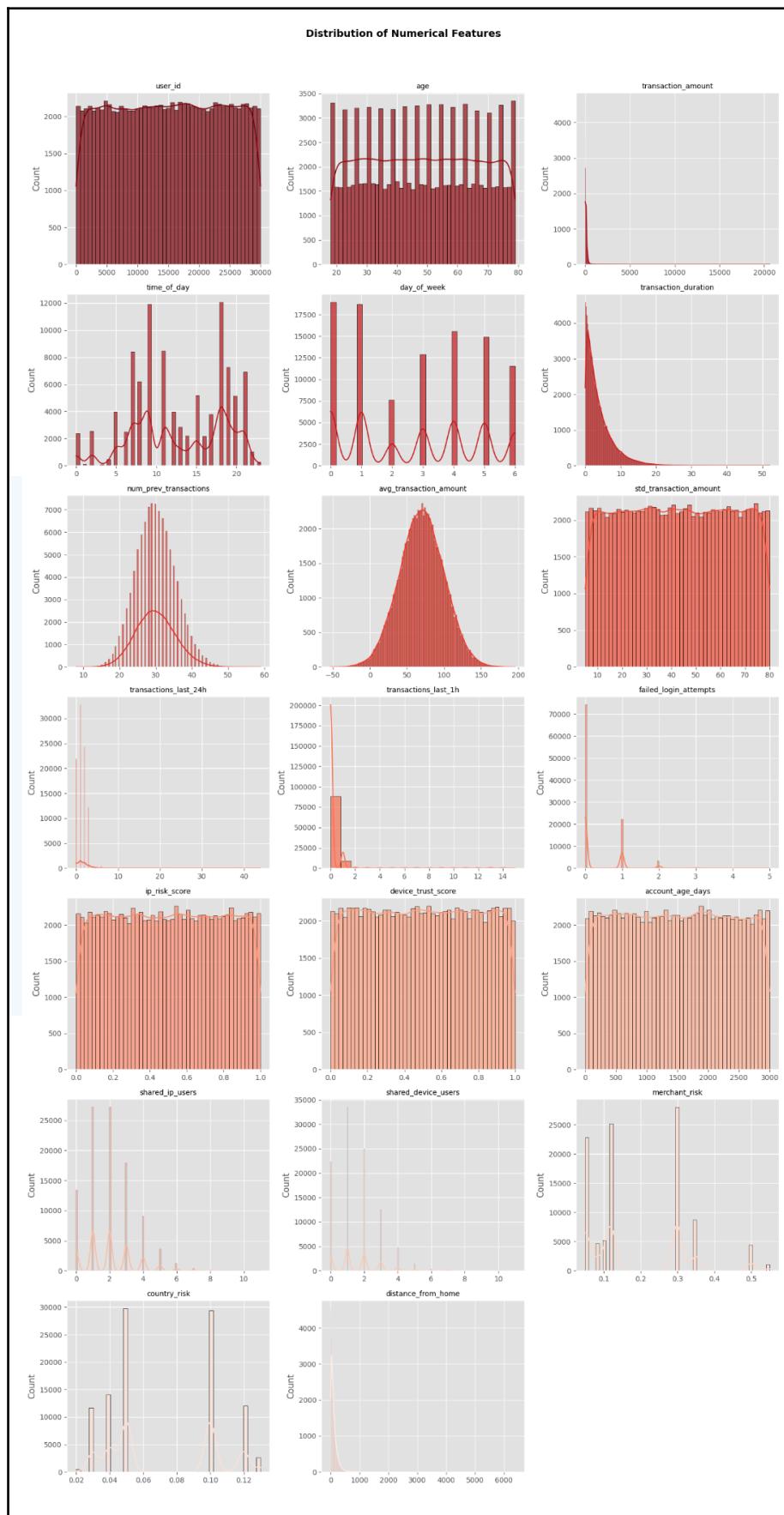
## 4.2. Distribusi Fitur

### 4.2.1. Numerical Features Distribution

Berdasarkan distribusi fitur, didapatkan bahwa banyak fitur yang memiliki distribusi tidak merata atau *skewed*. Beberapa variabel menunjukkan pola *right-skewed* (banyak nilai kecil, sedikit nilai besar). Pola ini muncul pada fitur yang berkaitan dengan transaksi, seperti *transaction\_amount* dan *transaction\_duration*. Namun, ada beberapa yang terdistribusi cukup simetris dan terlihat seperti *bell curve* pada fitur, seperti *num\_prev\_transactions* dan *avg\_transaction\_amount*. Adapun fitur risiko yang memiliki distribusi *clustered*, seperti *country\_risk* dan *merchant\_risk*. Hal ini menunjukkan adanya beberapa *cluster* atau grup nilai. Fitur dengan pola *cluster* memiliki kemungkinan untuk dijadikan *engineered features*.

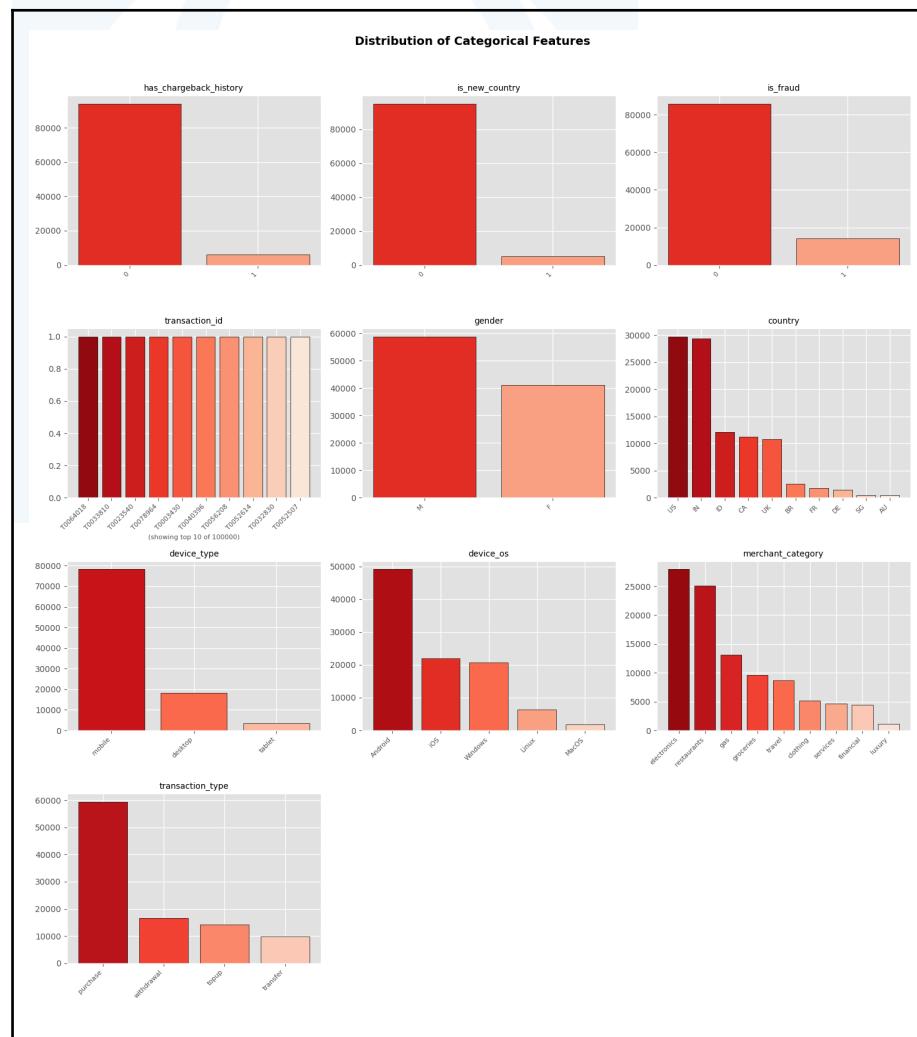
## Tugas Besar 2

### IF3070 Dasar Intelelegensi Artifisial



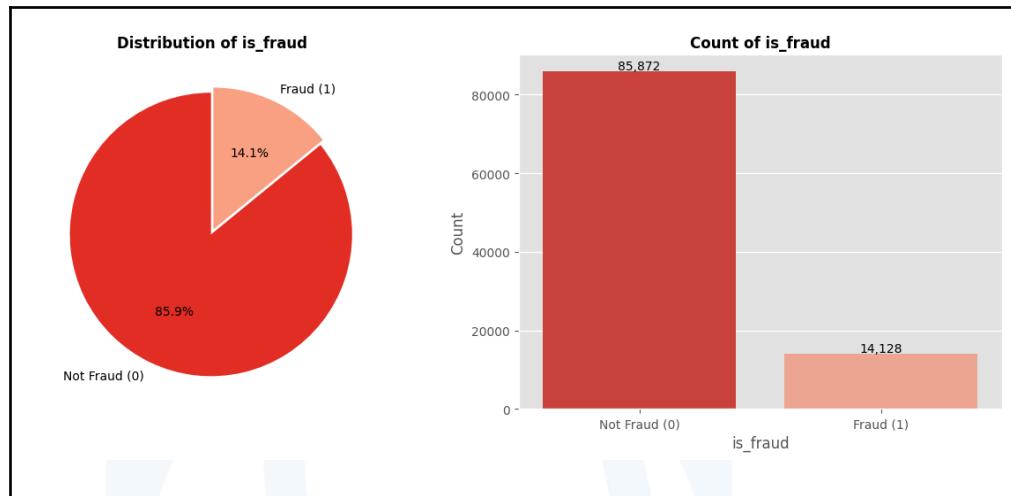
**Gambar 2.5. Distribution of Numerical Features****4.2.2. Categorical Features Distribution**

Berdasarkan distribusi fitur kategorikal, mayoritas *user* tidak memiliki riwayat *chargeback* yang ada pada kolom *has\_chargeback\_history*. Hal ini menunjukkan bahwa sebagian besar *user* memiliki rekam jejak yang bersih. Sebagian besar transaksi berasal dari negara US, menggunakan perangkat *mobile* (terutama Android), dan transaksi yang paling umum adalah *purchase*. Kategori *merchant* seperti *electronics* dan *entertainment* muncul paling sering, yang biasanya merupakan kategori bernilai tinggi dan rawan *fraud*. Secara keseluruhan, pola distribusi ini menegaskan bahwa kasus *fraud* jarang tetapi dapat dikenali melalui ciri-ciri spesifik

**Gambar 2.6. Distribution of Categorical Features**

### 4.3. Distribusi Variabel Target

Berdasarkan distribusi variabel target, 85.9% transaksi adalah *not fraud* dan hanya 14.1% yang merupakan *fraud*. Hal ini berarti bahwa transaksi *fraud* jumlahnya jauh lebih sedikit dibanding transaksi normal,

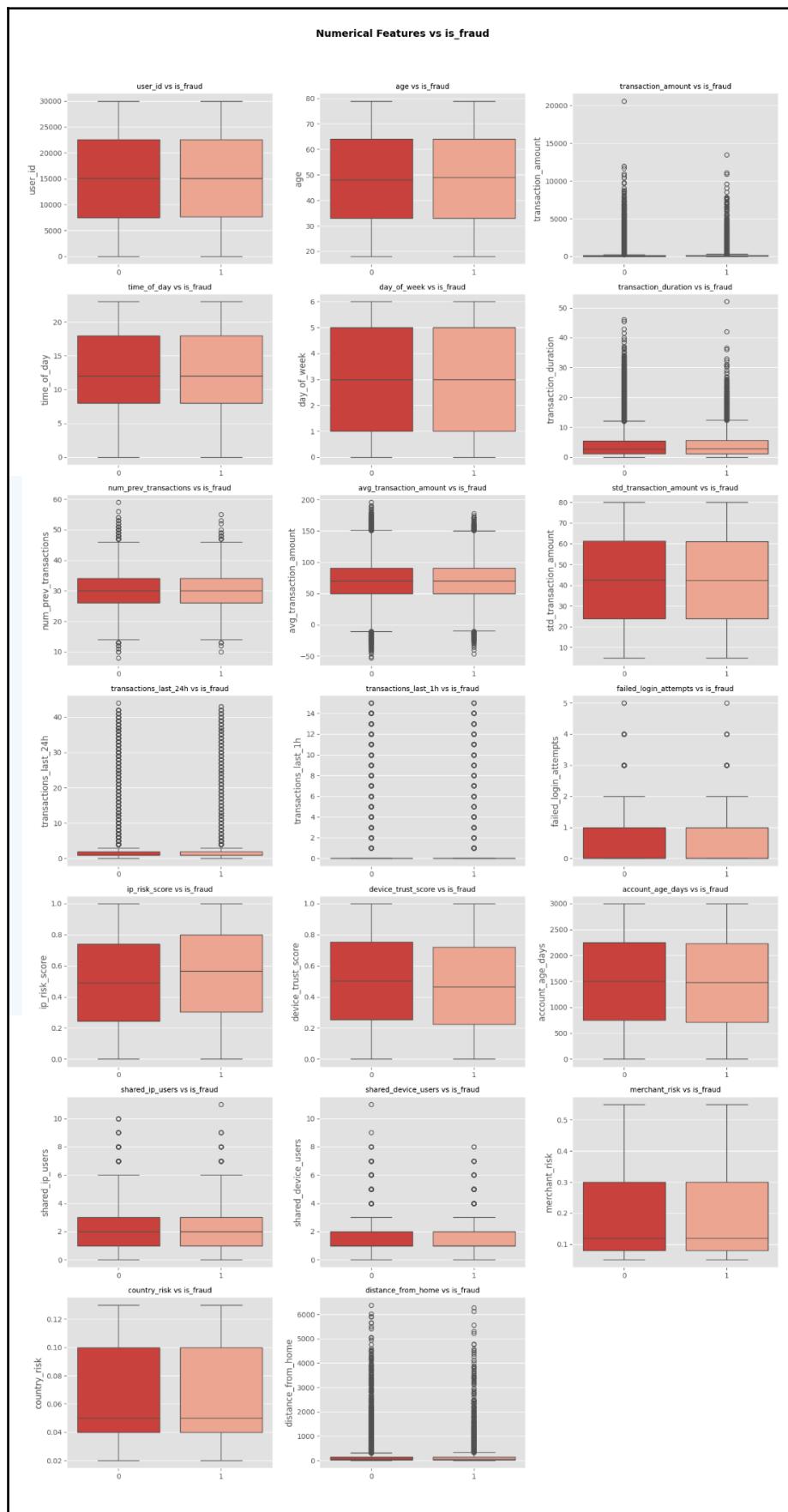


Gambar 2.7. Target Variable Distribution

### 4.4. Perbandingan Fitur dan Target

#### 4.4.1. Perbandingan Fitur Numerikal dan Target

Secara umum, nilai pada beberapa kolom seperti *transaction\_amount*, *transaction\_duration*, *num\_prev\_transactions*, *failed\_login\_attempts*, *ip\_risk\_score*, *shared\_ip\_users*, *shared\_device\_users*, *merchant\_risk*, *country\_risk*, dan terutama *distance\_from\_home* cenderung lebih tinggi pada transaksi *fraud*. Hal ini menggambarkan bahwa *fraud* lebih sering terjadi pada transaksi yang nilainya besar, berlangsung lebih lama, dilakukan dari lokasi yang jauh. *Fraud* juga menggunakan perangkat atau IP yang tidak biasa, atau melibatkan risiko tinggi dari sisi *merchant* maupun negara. Di sisi lain, kolom seperti *age*, *time\_of\_day*, *day\_of\_week*, dan *account\_age\_days* tidak menunjukkan perbedaan berarti antara *fraud* dan *non fraud*.

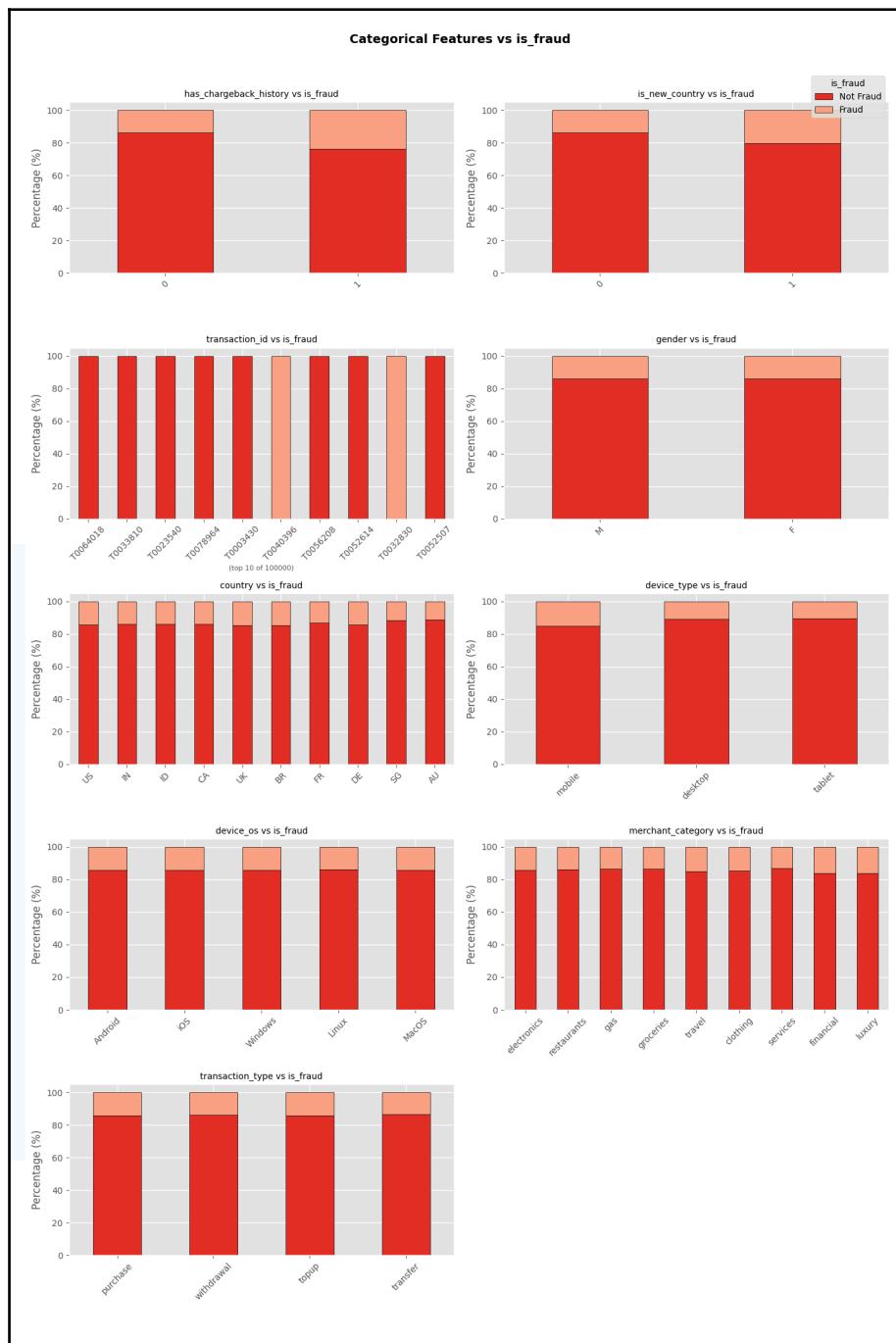


**Gambar 2.8. Numerical Features vs Target**

**4.4.2. Perbandingan Fitur Kategorikal dan Target**

Secara singkat, pola yang paling terlihat dari fitur kategorikal adalah bahwa transaksi *fraud* lebih sering terjadi pada *user* yang memiliki *has\_chargeback\_history* = 1 dan pada transaksi yang dilakukan di *is\_new\_country* = 1. Sementara itu, kolom lain seperti *gender*, *country*, *device\_type*, *device\_os*, *merchant\_category*, dan *transaction\_type* menunjukkan persentase *fraud* yang hampir sama di setiap kategori, sehingga tidak terlalu membantu membedakan mana transaksi yang berisiko.



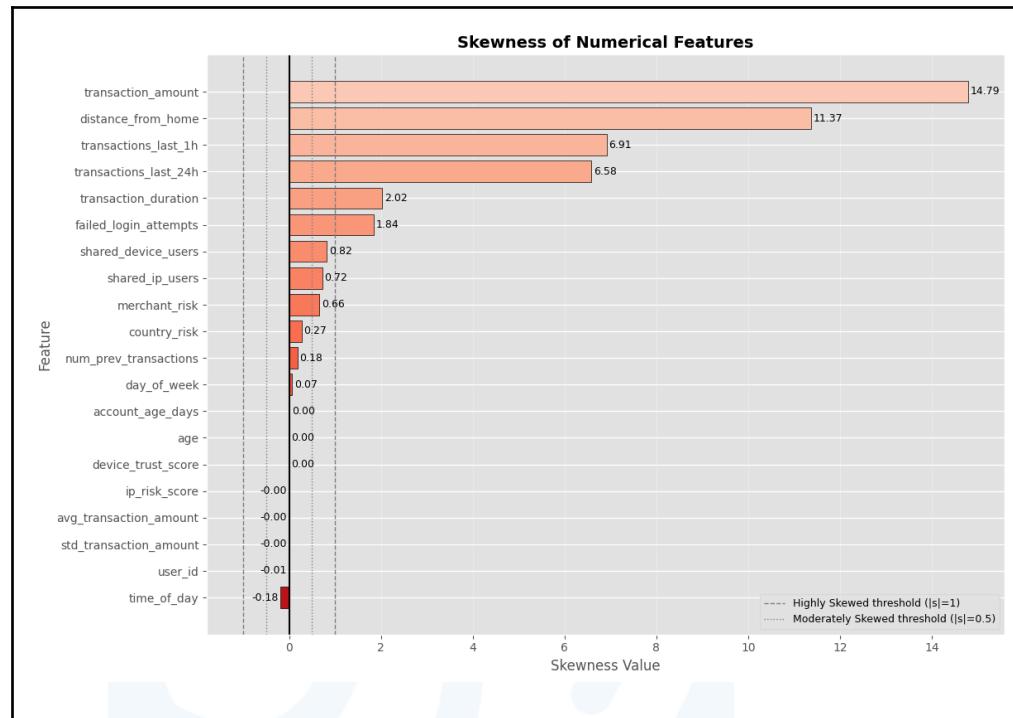


Gambar 2.9. Categorical Features vs Target

#### 4.5. Skewness Analysis

Grafik skewness berikut ini menunjukkan bahwa beberapa fitur numerik seperti *transaction\_amount*, *distance\_from\_home*, *transactions\_last\_1h*, dan *transactions\_last\_24h* memiliki skewness yang sangat tinggi. Hal ini berarti sebagian besar nilainya kecil sementara sebagian

kecil lainnya sangat besar. Fitur seperti *transaction\_duration*, *failed\_login\_attempts*, *shared\_device\_users*, dan *shared\_ip\_users* juga cukup skewed meski tidak separah fitur utama. Sementara itu, fitur lain seperti *country\_risk*, *merchant\_risk*, *num\_prev\_transactions*, *age*, dan *account\_age\_days* cenderung lebih seimbang.



Gambar 2.10. Skewness Analysis

## BAB III

### **DATA SPLITTING**

#### **1. Dataset Splitting Methodology**

Pembagian data menjadi *training set* dan *validation set* digunakan sebagai langkah awal untuk mengecek performa model yang akan dilatih. Pembagian ini dilakukan sebelum proses preprocessing untuk menghindari terjadinya *data leakage* antar *set*. Data *splitting* memiliki peran krusial dalam pengembangan model machine karena sebagai sebuah cara untuk mencegah *overfitting* di mana model menjadi terlalu baik dalam mempelajari pola pada data *training* tetapi gagal untuk generalisasi pada data baru. Selain itu data *splitting* dapat memberikan estimasi yang tidak bias tentang performa model pada data yang belum pernah dilihat sebelumnya serta memungkinkan penyetelan *hyperparameter* secara efektif tanpa menggunakan data test. Terakhir, memastikan evaluasi model dilakukan pada dataset yang sama sekali terpisah dari proses *training*.

Metode pemisahan antara *training set* dan *validation set* yang digunakan adalah stratified k-fold validation, yaitu teknik validasi berulang dengan membagi data ke dalam beberapa fold yang mempertahankan proporsi kelas. Pendekatan ini bertujuan untuk meningkatkan kestabilan hasil validasi serta memastikan bahwa nilai evaluasi yang diperoleh tidak semata-mata merupakan kebetulan statistik.

**Tabel 3.1. Data Splitting**

```
from sklearn.model_selection import StratifiedKFold  
  
skf = StratifiedKFold(n_splits=5, shuffle=True,  
random_state=42)  
print("Using 5-Fold Stratified Cross-Validation for  
model evaluation")
```

#### **1.1. Split Training Set**

*Split Training Set* adalah subset data yang digunakan untuk melatih atau membangun model *machine learning*. Dalam proses *training*, model mempelajari pola, fitur, dan hubungan yang terdapat dalam data untuk mengoptimalkan parameter internalnya. *Training set* harus memiliki ukuran

yang cukup besar untuk memastikan model dapat menangkap variabilitas dan kompleksitas dalam data secara memadai. Ukuran yang terlalu kecil akan menyebabkan *high bias*, sedangkan ukuran yang terlalu besar tanpa diimbangi dengan data *validation* dan *test* yang cukup dapat menyebabkan model sulit untuk dievaluasi dengan *proper*.

### 1.2. *Validation Set*

*Validation Set* adalah subset data independen yang digunakan untuk menyetel *hyperparameter* model dan memantau performa selama proses *training* berlangsung. *Validation set* tidak digunakan untuk melatih model, melainkan untuk melakukan evaluasi dan *monitoring*. Data dalam *validation set* tidak boleh digunakan untuk *training* model, agar estimasi perform yang diberikan tetap tidak bias dan mencerminkan performa model pada data yang benar-benar baru.

### 1.3. *Testing Set*

*Testing Set* adalah subset data yang sepenuhnya independen dan tidak pernah digunakan selama proses *training* maupun *validation*. *Test set* hanya digunakan sekali di akhir untuk evaluasi final model. Karena *test* sama sekali tidak terlihat oleh model selama *development* dan *tuning hyperparameter*, hasil evaluasi pada *test set* memberikan estimasi yang objektif dan akurat tentang bagaimana model akan berkinerja pada data dunia nyata yang benar-benar baru dan belum pernah dilihat sebelumnya. *Test set* umumnya mengambil porsi 10-20% dari total dataset. Namun, karena masih dalam konteks kompetisi *kaggle*, data test sudah disediakan oleh asisten dan tidak perlu membagi dari data training yang diberikan.

## BAB IV

### ***PREPROCESSING***

*Preprocessing* merupakan tahapan selanjutnya dari pemahaman gambaran umum data. Data mentah jarang sekali dapat langsung digunakan untuk proses *training*. Sehingga, data perlu pembersihan dan penyesuaian agar data tersebut dapat dibaca dan dipahami oleh model *machine learning*.

#### 1. ***Data Cleaning***

*Data cleaning* adalah proses identifikasi dan perbaikan kesalahan dalam *dataset*, termasuk menangani nilai kosong, data duplikat, atau inkonsistensi format. Tujuan utama *data cleaning* adalah memastikan *dataset* yang akan digunakan memiliki kualitas tinggi untuk analisis dan model prediktif. Data yang sudah melewati tahapan *data cleaning* akan harus di-*preprocess*, sebelum akhirnya digunakan dalam *training machine learning*.

**Tabel 4.1. *Data Cleaning***

Kode	Deskripsi
<pre>test_ids = test['ID'].values train_ids = train['ID'].values  print(f"Stored {len(train_ids)} training IDs") print(f"Stored {len(test_ids)} test IDs")</pre>	Menyimpan nilai ID dari <i>dataset train</i> dan <i>test</i> ke dalam variabel <i>train_ids</i> dan <i>test_ids</i> dalam bentuk <i>array</i> , lalu menampilkan jumlah ID yang tersimpan pada masing-masing <i>dataset</i> ( <i>training</i> dan <i>test</i> ) ke layar.
<pre>y_train = train['is_fraud'].values</pre>	Memisahkan variabel target <i>is_fraud</i> dari <i>dataset train</i> dan menyimpannya ke dalam variabel

<pre>print(f"Target variable shape: {y_train.shape}") print(f"Target variable type: {y_train.dtype}")</pre>	<p>y_train dalam bentuk array, kemudian menampilkan bentuk (<i>shape</i>) dan tipe data (<i>dtype</i>) dari variabel target tersebut.</p>
<pre>cols_to_drop = ['ID', 'transaction_id', 'user_id']  train_features = train.drop(columns=['is_fraud'] + cols_to_drop, errors='ignore')  test_features = test.drop(columns=cols_to_drop, errors='ignore')  print(f"Training features shape: {train_features.shape}") print(f"Test features shape: {test_features.shape}")</pre>	<p>Menghapus kolom yang tidak diperlukan (seperti <i>ID</i>, <i>transaction_id</i>, <i>user_id</i>, dan <i>is_fraud</i> pada <i>data train</i>), kemudian menyimpan fitur hasilnya ke dalam <i>train_features</i> dan <i>test_features</i>, serta menampilkan ukuran (<i>shape</i>) fitur <i>training</i> dan <i>test</i>.</p>
<pre>numerical_cols = train_features.select_dtypes(include=[np.number]).columns.tolist()  categorical_cols = train_features.select_dtypes(include=['object', 'category']).columns.tolist() ()</pre>	<p>Mengelompokkan kolom pada <i>train_features</i> menjadi kolom numerik dan kolom kategorikal berdasarkan tipe datanya, lalu menampilkan jumlah serta daftar nama kolom untuk masing-masing kelompok tersebut.</p>

```
print(f"Numerical columns\n({len(numerical_cols)}):")\nprint(numerical_cols)\nprint(f"\nCategorical\ncolumns\n({len(categorical_cols)}):\n")\nprint(categorical_cols)
```

### 1.1. Handling Missing Data

*Missing values* atau nilai kosong pada data *train.csv* dan *test.csv* ditangani dengan menggunakan metode imputasi, menggunakan KNN *Imputer* dari *sklearn* untuk data numerik dan mode (nilai paling sering muncul) untuk data kategorikal. Pertama, jumlah *missing values* pada setiap kolom di *dataset* akan ditampilkan agar dapat diketahui seberapa banyak data yang hilang. Kemudian, diimplementasikan metode imputasi melalui variabel *USE\_KNN\_IMPUTATION*. Jika bernilai *True*, maka digunakan KNN, sedangkan jika *False*, digunakan imputasi sederhana *mean* atau *mode*. Terdapat dua tahapan proses yang dijalankan jika KNN *imputation* digunakan sebagai berikut:

1. Kolom kategorikal diisi terlebih dahulu menggunakan mode, karena KNN hanya dapat bekerja pada data numerik.
2. Kolom numerik diimputasi menggunakan KNN *Imputer*, di mana setiap nilai kosong diisi berdasarkan rata-rata berbobot dari k tetangga terdekat. Hal ini ditentukan oleh *n\_neighbors* dengan jarak *euclidean* sehingga dapat menangani nilai NaN.

Model KNN dilatih menggunakan data numerik dari *training set* yang kemudian akan diaplikasikan juga ke *test set* agar konsisten dan tidak terjadi *data leakage*. Jika, KNN *Imputation* tidak digunakan, maka kolom numerik diisi dengan *mean* atau rata-rata dan kolom kategorikal diisi dengan *mode*.

**Tabel 4.2. Imputation**

```
import time

train_filled = train_features.copy()
test_filled = test_features.copy()
imputation_values = {}

if USE_KNN_IMPUTATION:
    start_time = time.time()

    print("\nKategorikal:")
    cat_modes = {}
    for col in categorical_cols:
        if col in train_filled.columns:
            mode_val = train_filled[col].mode()
            if len(mode_val) > 0:
                cat_modes[col] = mode_val[0]
                if
train_filled[col].isnull().any():
                train_filled[col] =
train_filled[col].fillna(cat_modes[col])
                print(f"  Imputed train {col}
with mode: {cat_modes[col]}")
                if test_filled[col].isnull().any():
                    test_filled[col] =
test_filled[col].fillna(cat_modes[col])
                    print(f"  Imputed test {col}
with mode: {cat_modes[col]}")

    print("\nNumerikal:")
    num_cols_exist = [c for c in numerical_cols if
c in train_filled.columns]

    if len(num_cols_exist) > 0:
        train_numerical =
train_filled[num_cols_exist].values
        test_numerical =
test_filled[num_cols_exist].values

        knn_imputer = SklearnKNNImputer(
            n_neighbors=KNN_NEIGHBORS,
            weights='distance',
            metric='nan_euclidean'
        )

        train_numerical_imputed =
knn_imputer.fit_transform(train_numerical)

        test_numerical_imputed =
```

```
knn_imputer.transform(test_numerical)

        train_filled[num_cols_exist] =
train_numerical_imputed
        test_filled[num_cols_exist] =
test_numerical_imputed

    elapsed = time.time() - start_time
    print(f"Imputation time: {elapsed:.2f} seconds")

    imputation_values = {
        'method': 'knn_sklearn',
        'n_neighbors': KNN_NEIGHBORS,
        'categorical_modes': cat_modes
    }

else:
    print("Mean/Mode")
    for col in numerical_cols:
        if col in train_filled.columns and
train_filled[col].isnull().any():
            mean_value = train_filled[col].mean()
            train_filled[col] =
train_filled[col].fillna(mean_value)
            test_filled[col] =
test_filled[col].fillna(mean_value)
            imputation_values[col] = {'type':
'mean', 'value': mean_value}
            print(f"Imputed {col} with mean:
{mean_value:.4f}")

        for col in categorical_cols:
            if col in train_filled.columns and
train_filled[col].isnull().any():
                mode_value = train_filled[col].mode()
                if len(mode_value) > 0:
                    mode_value = mode_value[0]
                    train_filled[col] =
train_filled[col].fillna(mode_value)
                    test_filled[col] =
test_filled[col].fillna(mode_value)
                    imputation_values[col] = {'type':
'mode', 'value': mode_value}
                    print(f"Imputed {col} with mode:
{mode_value}")
```

```
print("Verification - Missing values after  
imputation:")  
print(f" Train:  
{train_filled.isnull().sum().sum() }")  
print(f" Test:  
{test_filled.isnull().sum().sum() }")
```

## 2. Feature Engineering

### 2.1. Ratio Features

*Ratio Features* menangkap hubungan proporsional antara kedua variabel numerik yang *logically related*, seperti amount\_vs\_avg\_ratio yang menghitung rasio nilai transaksi saat ini dan rata-rata transaksi historis pengguna yang mengindikasikan transaksi anomali kalau nilainya terlalu tinggi.

**Tabel 4.3. Feature Engineering**

```
ratio_pairs = [  
    ('transaction_amount',  
     'avg_transaction_amount', 'amount_vs_avg_ratio'),  
    ('transaction_amount',  
     'std_transaction_amount', 'amount_vs_std_ratio'),  
    ('transactions_last_1h',  
     'transactions_last_24h', 'hourly_vs_daily_ratio'),  
    ('failed_login_attempts',  
     'num_prev_transactions', 'failed_vs_total_ratio'),  
    ('shared_ip_users', 'shared_device_users',  
     'ip_vs_device_shared_ratio'),  
]  
  
for num, denom, name in ratio_pairs:  
    if num in train_filled.columns and denom in  
        train_filled.columns:  
        denom_safe = train_filled[denom].replace(0,  
        1e-10)  
        train_filled[name] = train_filled[num] /  
        denom_safe  
        denom_safe = test_filled[denom].replace(0,  
        1e-10)  
        test_filled[name] = test_filled[num] /  
        denom_safe  
  
    print(f"Created ratio feature: {name}")
```

```
print(f"\nTotal features after ratio creation:  
{train_filled.shape[1]}")
```

## 2.2. Log-Transformed Features

*Log-Transformed Features* memanfaatkan logaritmik yang diterapkan pada seluruh kolom numerik non-negatif. Hal ini diterapkan agar mengurangi *skewness* dari distribusi data yang tidak rata, seperti *transaction\_amount*. Dengan ini, model *machine learning* bisa lebih mudah mempelajari pola data.

**Tabel 4.4.** Log-Transformed Features

```
log_candidates =  
train.select_dtypes(include=[np.number]).columns  
for col in log_candidates:  
    if col in train_filled.columns:  
        min_val = train_filled[col].min()  
        if min_val >= 0:  
            train_filled[f'{col}_log'] =  
np.log1p(train_filled[col])  
            test_filled[f'{col}_log'] =  
np.log1p(test_filled[col])  
            print(f"Created log feature:  
{col}_log")  
  
print(f"\nTotal features after log transformation:  
{train_filled.shape[1]}")
```

## 2.3. Interaction Features

*Interaction Features* menggabungkan dua atau lebih variabel untuk mendapat pola kombinatorik yang muncul hanya bila ditangkap secara kolektif. Contohnya seperti fitur *risk\_interaction* yang menghitung interaksi antara *IP risk score* dengan *inverse device trust score* untuk mengidentifikasi kombinasi yang berisiko tinggi.

**Tabel 4.5.** Interaction Features

```
if 'ip_risk_score' in train_filled.columns and  
'device_trust_score' in train_filled.columns:  
    train_filled['risk_interaction'] =  
train_filled['ip_risk_score'] * (1 -  
train_filled['device_trust_score'] / 100)
```

```
    test_filled['risk_interaction'] =
test_filled['ip_risk_score'] * (1 -
test_filled['device_trust_score'] / 100)
    print("Created: risk_interaction")

if 'merchant_risk' in train_filled.columns and
'country_risk' in train_filled.columns:
    train_filled['merchant_country_risk'] =
train_filled['merchant_risk'] *
train_filled['country_risk']
    test_filled['merchant_country_risk'] =
test_filled['merchant_risk'] *
test_filled['country_risk']
    print("Created: merchant_country_risk")

if 'transaction_amount' in train_filled.columns and
'avg_transaction_amount' in train_filled.columns:
    std_col = 'std_transaction_amount'
    if std_col in train_filled.columns and
train_filled[std_col].mean() > 0:
        train_filled['amount_zscore'] =
(train_filled['transaction_amount'] -
train_filled['avg_transaction_amount']) /
(train_filled[std_col] + 1e-6)
        test_filled['amount_zscore'] =
(test_filled['transaction_amount'] -
test_filled['avg_transaction_amount']) /
(test_filled[std_col] + 1e-6)
        print("Created: amount_zscore")

if 'transactions_last_24h' in train_filled.columns and
'transactions_last_1h' in train_filled.columns:
    train_filled['hourly_concentration'] =
train_filled['transactions_last_1h'] /
(train_filled['transactions_last_24h'] + 1)
    test_filled['hourly_concentration'] =
test_filled['transactions_last_1h'] /
(test_filled['transactions_last_24h'] + 1)
    print("Created: hourly_concentration")

if 'account_age_days' in train_filled.columns and
'num_prev_transactions' in train_filled.columns:
    train_filled['tx_per_day_age'] =
train_filled['num_prev_transactions'] /
(train_filled['account_age_days'] + 1)
    test_filled['tx_per_day_age'] =
test_filled['num_prev_transactions'] /
(test_filled['account_age_days'] + 1)
```

```
print("Created: tx_per_day_age")

if 'is_new_country' in train_filled.columns and
'distance_from_home' in train_filled.columns:
    train_filled['new_location_distance'] =
train_filled['is_new_country'] *
train_filled['distance_from_home']
    test_filled['new_location_distance'] =
test_filled['is_new_country'] *
test_filled['distance_from_home']
    print("Created: new_location_distance")

if 'failed_login_attempts' in train_filled.columns and
'transaction_amount' in train_filled.columns:
    train_filled['failed_login_amount'] =
train_filled['failed_login_attempts'] *
train_filled['transaction_amount']
    test_filled['failed_login_amount'] =
test_filled['failed_login_attempts'] *
test_filled['transaction_amount']
    print("Created: failed_login_amount")

if 'shared_ip_users' in train_filled.columns and
'shared_device_users' in train_filled.columns:
    train_filled['total_shared_users'] =
train_filled['shared_ip_users'] +
train_filled['shared_device_users']
    test_filled['total_shared_users'] =
test_filled['shared_ip_users'] +
test_filled['shared_device_users']

    train_filled['shared_resource_product'] =
train_filled['shared_ip_users'] *
train_filled['shared_device_users']
    test_filled['shared_resource_product'] =
test_filled['shared_ip_users'] *
test_filled['shared_device_users']
    print("Created: total_shared_users,
shared_resource_product")

if 'time_of_day' in train_filled.columns:
    train_filled['is_night_time'] =
((train_filled['time_of_day'] >= 0) &
(train_filled['time_of_day'] <= 6) |
(train_filled['time_of_day'] >= 22)).astype(float)
    test_filled['is_night_time'] =
((test_filled['time_of_day'] >= 0) &
```

```
(test_filled['time_of_day'] <= 6) |  
  
(test_filled['time_of_day'] >= 22)).astype(float)  
    print("Created: is_night_time")  
  
if 'day_of_week' in train_filled.columns:  
    train_filled['is_weekend'] =  
(train_filled['day_of_week'] >= 5).astype(float)  
    test_filled['is_weekend'] =  
(test_filled['day_of_week'] >= 5).astype(float)  
    print("Created: is_weekend")  
  
if 'has_chargeback_history' in train_filled.columns  
and 'transaction_amount' in train_filled.columns:  
    train_filled['chargeback_high_amount'] =  
train_filled['has_chargeback_history'] *  
(train_filled['transaction_amount'] >  
train_filled['transaction_amount'].median()).astype  
(float)  
    test_filled['chargeback_high_amount'] =  
test_filled['has_chargeback_history'] *  
(test_filled['transaction_amount'] >  
train_filled['transaction_amount'].median()).astype  
(float)  
    print("Created: chargeback_high_amount")  
  
if 'ip_risk_score' in train_filled.columns and  
'transaction_amount' in train_filled.columns:  
    train_filled['high_risk_high_amount'] =  
train_filled['ip_risk_score'] *  
train_filled['transaction_amount']  
    test_filled['high_risk_high_amount'] =  
test_filled['ip_risk_score'] *  
test_filled['transaction_amount']  
    print("Created: high_risk_high_amount")  
  
if 'failed_login_attempts' in train_filled.columns  
and 'ip_risk_score' in train_filled.columns:  
    train_filled['failed_login_risk'] =  
train_filled['failed_login_attempts'] *  
train_filled['ip_risk_score']  
    test_filled['failed_login_risk'] =  
test_filled['failed_login_attempts'] *  
test_filled['ip_risk_score']  
    print("Created: failed_login_risk")  
  
if 'is_new_country' in train_filled.columns and  
'transaction_amount' in train_filled.columns:
```

```
train_filled['new_country_amount'] =
train_filled['is_new_country'] *
train_filled['transaction_amount']
test_filled['new_country_amount'] =
test_filled['is_new_country'] *
test_filled['transaction_amount']
print("Created: new_country_amount")

if 'ip_risk_score' in train_filled.columns:
    train_filled['ip_risk_squared'] =
train_filled['ip_risk_score'] ** 2
    test_filled['ip_risk_squared'] =
test_filled['ip_risk_score'] ** 2
    print("Created: ip_risk_squared")

if 'merchant_risk' in train_filled.columns:
    train_filled['merchant_risk_squared'] =
train_filled['merchant_risk'] ** 2
    test_filled['merchant_risk_squared'] =
test_filled['merchant_risk'] ** 2
    print("Created: merchant_risk_squared")

if 'country_risk' in train_filled.columns:
    train_filled['country_risk_squared'] =
train_filled['country_risk'] ** 2
    test_filled['country_risk_squared'] =
test_filled['country_risk'] ** 2
    print("Created: country_risk_squared")

risk_cols = []
if 'ip_risk_score' in train_filled.columns:
    risk_cols.append('ip_risk_score')
if 'merchant_risk' in train_filled.columns:
    risk_cols.append('merchant_risk')
if 'country_risk' in train_filled.columns:
    risk_cols.append('country_risk')
if len(risk_cols) > 0:
    train_filled['combined_risk'] =
train_filled[risk_cols].mean(axis=1)
    test_filled['combined_risk'] =
test_filled[risk_cols].mean(axis=1)
    print("Created: combined_risk")

print(f"\nTotal features after interaction
creation: {train_filled.shape[1]}")
```

## 2.4. Percentile Rank Features

*Percentile Rank Features* mengonversi nilai numerik menjadi skala persentil (dalam skala 0-1). Ini akan lebih matang karena hanya mempertimbangkan urutan secara relatif. Hal ini dilakukan untuk menghindari *data leakage*.

**Tabel 4.6. Percentile Rank Features**

```
percentile_cols = ['transaction_amount',
'distance_from_home', 'ip_risk_score',
'merchant_risk', 'country_risk',
'account_age_days']

for col in percentile_cols:
    if col in train_filled.columns:
        train_filled[f'{col}_percentile'] =
train_filled[col].rank(pct=True)
        train_sorted =
np.sort(train_filled[col].dropna().values)
        test_filled[f'{col}_percentile'] =
test_filled[col].apply(
            lambda x: np.searchsorted(train_sorted,
x) / len(train_sorted) if pd.notna(x) else 0.5
        )
        print(f"  Created: {col}_percentile")
```

## 2.5. Binned Risk Categories

*Binned Risk Categories* dilakukan dengan membagi *risk scores* yang kontinu menjadi kategori diskrit dengan interval tertentu. *Binning* ini memungkinkan model menangkap hubungan non-linear antara beberapa fitur.

**Tabel 4.7. Banned Risk Categories**

```
def create_risk_bins(series, bins=[0, 0.2, 0.4,
0.6, 0.8, 1.0], labels=['very_low', 'low',
'medium', 'high', 'very_high']):
    return pd.cut(series, bins=bins, labels=labels,
include_lowest=True)

risk_bin_cols = ['ip_risk_score', 'merchant_risk',
'country_risk']
for col in risk_bin_cols:
    if col in train_filled.columns:
```

```
train_bins =  
create_risk_bins(train_filled[col])  
    test_bins =  
create_risk_bins(test_filled[col])  
  
    for label in ['very_low', 'low', 'medium',  
'high', 'very_high']:  
        train_filled[f'{col}_bin_{label}'] =  
(train_bins == label).astype(int)  
        test_filled[f'{col}_bin_{label}'] =  
(test_bins == label).astype(int)  
        print(f"  Binned: {col} into 5 risk  
levels")
```

## 2.6. Anomaly Score Features

*Anomaly Score Features* dilakukan dengan menghitung *Z-score* pada kolom yang kritis, seperti contohnya *transaction\_amount*. Untuk yang nilai observasi di luar range -2 sampai 2 akan dianggap *outlier*.

**Tabel 4.8. Anomaly Score Features**

```
anomaly_cols = ['transaction_amount',  
'distance_from_home', 'transactions_last_1h',  
                 'transactions_last_24h',  
'failed_login_attempts']  
  
for col in anomaly_cols:  
    if col in train_filled.columns:  
        col_mean = train_filled[col].mean()  
        col_std = train_filled[col].std()  
  
        if col_std > 0:  
            train_filled[f'{col}_zscore'] =  
(train_filled[col] - col_mean) / col_std  
            test_filled[f'{col}_zscore'] =  
(test_filled[col] - col_mean) / col_std  
  
            train_filled[f'{col}_is_outlier'] =  
(np.abs(train_filled[f'{col}_zscore']) >  
2).astype(int)  
            test_filled[f'{col}_is_outlier'] =  
(np.abs(test_filled[f'{col}_zscore']) >  
2).astype(int)  
  
            print(f"Created: {col}_zscore,  
{col}_is_outlier")
```

```
outlier_cols = [c for c in train_filled.columns if
c.endswith('_is_outlier')]
if len(outlier_cols) > 0:
    train_filled['total_outlier_count'] =
train_filled[outlier_cols].sum(axis=1)
    test_filled['total_outlier_count'] =
test_filled[outlier_cols].sum(axis=1)
    print(f"Created: total_outlier_count (sum of
{len(outlier_cols)} outlier flags)")
```

## 2.7. Velocity & Acceleration Features

*Velocity & Acceleration Features* mengukur kecepatan perubahan perilaku dari pengguna, seperti membandingkan aktivitas per jam dengan rata-rata aktivitas harian untuk mendeteksi adanya *sudden burst* yang mengindikasikan anomali.

**Tabel 4.9. Velocity & Acceleration Features**

```
if 'transactions_last_1h' in train_filled.columns
and 'transactions_last_24h' in
train_filled.columns:
    train_filled['tx_burst_ratio'] =
train_filled['transactions_last_1h'] /
(train_filled['transactions_last_24h'] / 24 + 0.1)
    test_filled['tx_burst_ratio'] =
test_filled['transactions_last_1h'] /
(test_filled['transactions_last_24h'] / 24 + 0.1)

    train_filled['is_tx_burst'] =
(train_filled['tx_burst_ratio'] > 3).astype(int)
    test_filled['is_tx_burst'] =
(test_filled['tx_burst_ratio'] > 3).astype(int)

if 'transaction_amount' in train_filled.columns and
'avg_transaction_amount' in train_filled.columns:
    train_filled['amount_deviation_ratio'] =
train_filled['transaction_amount'] /
(train_filled['avg_transaction_amount'] + 1)
    test_filled['amount_deviation_ratio'] =
test_filled['transaction_amount'] /
(test_filled['avg_transaction_amount'] + 1)

    train_filled['is_unusual_amount'] =
(train_filled['amount_deviation_ratio'] >
```

```
3).astype(int)
    test_filled['is_unusual_amount'] =
(test_filled['amount_deviation_ratio'] >
3).astype(int)
```

## 2.8. Composite Risk Scores

*Composite Risk Scores* menggabungkan *multiple risk indicators* menjadi satu metrik, seperti mengalikan *combined risk* dengan *normalized transaction amount* untuk potensi dampak kerugian transaksi risikan.

**Tabel 4.10. Composite Risk Scores**

```
risk_flags = []

if 'is_new_country' in train_filled.columns:
    risk_flags.append('is_new_country')
if 'has_chargeback_history' in
train_filled.columns:
    risk_flags.append('has_chargeback_history')
if 'failed_login_attempts' in train_filled.columns:
    train_filled['has_failed_logins'] =
(train_filled['failed_login_attempts'] >
0).astype(int)
    test_filled['has_failed_logins'] =
(test_filled['failed_login_attempts'] >
0).astype(int)
    risk_flags.append('has_failed_logins')
if 'is_tx_burst' in train_filled.columns:
    risk_flags.append('is_tx_burst')
if 'is_unusual_amount' in train_filled.columns:
    risk_flags.append('is_unusual_amount')
if 'is_night_time' in train_filled.columns:
    risk_flags.append('is_night_time')

if len(risk_flags) > 0:
    train_filled['risk_flag_count'] =
train_filled[risk_flags].sum(axis=1)
    test_filled['risk_flag_count'] =
test_filled[risk_flags].sum(axis=1)
    print(f" Created: risk_flag_count (sum of
{len(risk_flags)} flags)")

if 'combined_risk' in train_filled.columns and
'transaction_amount' in train_filled.columns:
    amount_max =
train_filled['transaction_amount'].max()
```

```
train_filled['weighted_risk_impact'] =  
train_filled['combined_risk'] *  
(train_filled['transaction_amount'] / amount_max)  
test_filled['weighted_risk_impact'] =  
test_filled['combined_risk'] *  
(test_filled['transaction_amount'] / amount_max)
```

## 2.9. Device & Session Risk Features

*Device & Session Risk Features* menganalisis karakteristik *device* dengan sesi pengguna, seperti identifikasi penggunaan perangkat yang digunakan bersama oleh *users* yang banyak, yang bisa mengindikasikan *fraud*.

**Tabel 4.11. Device & Session Risk Features**

```
if 'shared_ip_users' in train_filled.columns and  
'shared_device_users' in train_filled.columns:  
    train_filled['max_shared_users'] =  
train_filled[['shared_ip_users',  
'shared_device_users']].max(axis=1)  
    test_filled['max_shared_users'] =  
test_filled[['shared_ip_users',  
'shared_device_users']].max(axis=1)  
  
    train_filled['is_shared_resource'] =  
((train_filled['shared_ip_users'] > 1) |  
(train_filled['shared_device_users'] >  
1)).astype(int)  
    test_filled['is_shared_resource'] =  
((test_filled['shared_ip_users'] > 1) |  
(test_filled['shared_device_users'] >  
1)).astype(int)  
if 'device_trust_score' in train_filled.columns and  
'ip_risk_score' in train_filled.columns:  
    train_filled['trust_risk_discrepancy'] =  
train_filled['ip_risk_score'] - (1 -  
train_filled['device_trust_score'])/100  
    test_filled['trust_risk_discrepancy'] =  
test_filled['ip_risk_score'] - (1 -  
test_filled['device_trust_score'])/100
```

## 2.10. Time Pattern Features

*Time Pattern Features* dilakukan dengan *encoding* yang siklis menggunakan fungsi sin dan cos untuk menangkap sifat siklis dari waktu, seperti hari Minggu yang dekat dengan hari Senin. Dibuat *buckets* yang

membagi waktu menjadi kategori untuk menangkap pola *fraud* yang berbeda-beda tiap periode waktu. Ini diterapkan pada fitur yang berkaitan dengan waktu dan hari.

**Tabel 4.12. Time Pattern Features**

```
if 'time_of_day' in train_filled.columns:
    train_filled['time_sin'] = np.sin(2 * np.pi *
train_filled['time_of_day'] / 24)
    train_filled['time_cos'] = np.cos(2 * np.pi *
train_filled['time_of_day'] / 24)
    test_filled['time_sin'] = np.sin(2 * np.pi *
test_filled['time_of_day'] / 24)
    test_filled['time_cos'] = np.cos(2 * np.pi *
test_filled['time_of_day'] / 24)

def get_time_bucket(hour):
    if 6 <= hour < 12:
        return 'morning'
    elif 12 <= hour < 18:
        return 'afternoon'
    elif 18 <= hour < 22:
        return 'evening'
    else:
        return 'night'

    for bucket in ['morning', 'afternoon',
'evening', 'night']:
        train_filled[f'time_{bucket}'] =
train_filled['time_of_day'].apply(lambda x: 1 if
get_time_bucket(x) == bucket else 0)
        test_filled[f'time_{bucket}'] =
test_filled['time_of_day'].apply(lambda x: 1 if
get_time_bucket(x) == bucket else 0)

if 'day_of_week' in train_filled.columns:
    train_filled['day_sin'] = np.sin(2 * np.pi *
train_filled['day_of_week'] / 7)
    train_filled['day_cos'] = np.cos(2 * np.pi *
train_filled['day_of_week'] / 7)
    test_filled['day_sin'] = np.sin(2 * np.pi *
test_filled['day_of_week'] / 7)
    test_filled['day_cos'] = np.cos(2 * np.pi *
test_filled['day_of_week'] / 7)
```

## 2.11. Account Age Risk Features

*Account Age Risk Features* mengindikasikan hal penting dalam pendekslan *fraud*. Akun yang baru dan di bawah 30 hari lebih rentan terhadap *fraud*. Umur akun pun dibagi kelompok-kelompok berdasarkan betapa tuanya akun tersebut untuk menangkap risiko di tiap kelompok umur.

**Tabel 4.13. Account Age Risk Features**

```

if 'account_age_days' in train_filled.columns:
    train_filled['is_new_account'] =
        (train_filled['account_age_days'] < 30).astype(int)
    test_filled['is_new_account'] =
        (test_filled['account_age_days'] < 30).astype(int)

    def get_account_age_bucket(days):
        if days < 7:
            return 'very_new'
        elif days < 30:
            return 'new'
        elif days < 90:
            return 'established'
        elif days < 365:
            return 'mature'
        else:
            return 'veteran'

        for bucket in ['very_new', 'new',
        'established', 'mature', 'veteran']:
            train_filled[f'account_{bucket}'] =
            train_filled['account_age_days'].apply(lambda x: 1
            if get_account_age_bucket(x) == bucket else 0)
            test_filled[f'account_{bucket}'] =
            test_filled['account_age_days'].apply(lambda x: 1
            if get_account_age_bucket(x) == bucket else 0)

            if 'transaction_amount' in
            train_filled.columns:
                amount_median =
                train_filled['transaction_amount'].median()
                train_filled['new_account_high_tx'] =
                (train_filled['is_new_account'] *
                (train_filled['transaction_amount'] >
                amount_median)).astype(int)
                test_filled['new_account_high_tx'] =
                (test_filled['is_new_account'] *
                (test_filled['transaction_amount'] >
                amount_median)).astype(int)

```

## 2.12. Frequency Encoding

*Frequency Encoding* menghitung frekuensi kemunculan tiap kategori pada variabel yang kategorikal, seperti contohnya `payment_method`. Kategori yang jarang muncul akan ditinjau lebih lanjut karena atribut langka mengindikasikan *fraud*.

**Tabel 4.14.** Frequency Encoding

```
freq_encode_cols = ['payment_method',
'device_type', 'browser']

for col in freq_encode_cols:
    if col in train_filled.columns:
        freq_map =
train_filled[col].value_counts(normalize=True).to_d
ict()

        train_filled[f'{col}_frequency'] =
train_filled[col].map(freq_map).fillna(0)
        test_filled[f'{col}_frequency'] =
test_filled[col].map(freq_map).fillna(0) # Unknown
= 0 (rare)

        train_filled[f'{col}_is_rare'] =
(train_filled[f'{col}_frequency'] <
0.05).astype(int)
        test_filled[f'{col}_is_rare'] =
(test_filled[f'{col}_frequency'] <
0.05).astype(int)

        print(f"  Created: {col}_frequency,
{col}_is_rare")
```

## 2.13. One-Hot Encoding

*One-Hot Encoding* melakukan konversi variabel kategorikal menjadi biner, dengan tiap nilai unik dipisah kolomnya. Ini diimplementasikan dengan `OneHotEncoder` milik `scikit-learn`. Penggunaan *encoding* ini akan membuat interpretabilitas lebih tinggi dalam algoritma *machine learning*.

**Tabel 4.15.** One-Hot Encoding

```
train_encoded = train_filled.copy()
```

```
ohe = OneHotEncoder(
    handle_unknown="ignore",
    sparse_output=False
)

encoded_array =
ohe.fit_transform(train_encoded[categorical_cols])
encoded_feature_names =
ohe.get_feature_names_out(categorical_cols)

encoded_df = pd.DataFrame(
    encoded_array,
    columns=encoded_feature_names,
    index=train_encoded.index
)

encoding_info = {
    col: list(cats)
    for col, cats in zip(categorical_cols,
ohe.categories_)
}

train_encoded = pd.concat(
    [train_encoded.drop(columns=categorical_cols),
encoded_df],
    axis=1
)

print(f"Shape after encoding:
{train_encoded.shape}")

test_encoded = test_filled.copy()

encoded_test_array =
ohe.transform(test_encoded[categorical_cols])

encoded_test_df = pd.DataFrame(
    encoded_test_array,
    columns=ohe.get_feature_names_out(categorical_cols),
    index=test_encoded.index
)
encoded_test_df = pd.DataFrame(
    encoded_test_array,
    columns=ohe.get_feature_names_out(categorical_cols),
    index=test_encoded.index
)
```

```
    index=test_encoded.index
)

test_encoded = pd.concat(
    [test_encoded.drop(columns=categorical_cols),
     encoded_test_df],
    axis=1
)

print(f"Test shape after encoding:
{test_encoded.shape}")

train_cols = set(train_encoded.columns)
test_cols = set(test_encoded.columns)

for col in train_cols - test_cols:
    test_encoded[col] = 0
    print(f"Added missing column to test: {col}")

for col in test_cols - train_cols:
    train_encoded[col] = 0
    print(f"Added missing column to train: {col}")

all_cols = sorted(train_encoded.columns.tolist())
train_encoded = train_encoded[all_cols]
test_encoded = test_encoded[all_cols]

print("\nFinal shapes:")
print(f" Train: {train_encoded.shape}")
print(f" Test: {test_encoded.shape}")
```

## 2.14. Target Encoding

*Target Encoding* mnengkodekan variabel kategorikal berdasarkan rata-rata target per kategori. Dengan *encoding* ini, hubungan langsung antara kategori dan probabilitas *fraud* akan terdeteksi sehingga performal model akan meningkat.

**Tabel 4.16. Target Encoding**

```
te = TargetEncoder(
    smooth="auto",
    target_type="binary"
)

te_array = te.fit_transform(
```

```
    train_filled[categorical_cols],
    y_train
)

te_feature_names = [f"{col}_te" for col in
categorical_cols]
te_train_df = pd.DataFrame(
    te_array,
    columns=te_feature_names,
    index=train_filled.index
)

train_encoded = pd.concat(
    [train_encoded, te_train_df],
    axis=1
)

print(f"Train shape after adding target encoding:
{train_encoded.shape}")

te_test_array =
te.transform(test_filled[categorical_cols])

te_test_df = pd.DataFrame(
    te_test_array,
    columns=te_feature_names,
    index=test_filled.index
)

test_encoded = pd.concat(
    [test_encoded, te_test_df],
    axis=1
)

print(f"Test shape after adding target encoding:
{test_encoded.shape}")
```

### 3. Standardization

#### 3.1. Outlier Handling

Pertama, data hasil *encoding* dengan *train\_encoded* dan *test\_encoded* yang akan diubah menjadi *array NumPy* agar proses numerik lebih cepat. Kemudian, jumlah fitur, persentil batas bawah (*1st percentile*), dan batas atas (*99th percentile*). Untuk setiap fitur pada *data training*, akan dihitung nilai batas bawah dan atas berdasarkan persentil yang ditentukan. Nilai-nilai

tersebut akan disimpan di `clip_bounds` agar bisa digunakan kembali. Kemudian, nilai dari setiap fitur yang nilainya berada di bawah atau di atas batas tersebut akan dipotong menggunakan `np.clip`, sehingga *outlier* ekstrem tidak lagi memengaruhi distribusi data. Setelah proses pada data *training* selesai, batas *clipping* yang sama diterapkan pada data *test* untuk mencegah terjadinya *data leakage*.

**Tabel 4.17. Outlier Handling**

```
train_values = train_encoded.values.copy()
test_values = test_encoded.values.copy()

n_features = train_values.shape[1]
clip_bounds = {'lower': [], 'upper': []}

lower_percentile = 1
upper_percentile = 99

for i in range(n_features):
    lower = np.percentile(train_values[:, i],
    lower_percentile)
    upper = np.percentile(train_values[:, i],
    upper_percentile)

    clip_bounds['lower'].append(lower)
    clip_bounds['upper'].append(upper)

    train_values[:, i] = np.clip(train_values[:, i],
    lower, upper)

print(f"Clipped outliers in training data using
{lower_percentile}th and {upper_percentile}th
percentiles")
print(f"Shape: {train_values.shape}")

for i in range(test_values.shape[1]):
    test_values[:, i] = np.clip(test_values[:, i],
    clip_bounds['lower'][i], clip_bounds['upper'][i])

print(f"Shape: {test_values.shape}")
```

### 3.2. Feature Scaling

*Feature Scaling* menggunakan *Z-score standardization* yang akan membuat fitur memiliki mean 0 dan standar deviasi 1. Ini dilakukan untuk mencegah *data leakage*, kemudian diterapkan kepada seluruh data.

**Tabel 4.18. Feature Scaling**

```
means = np.mean(train_values, axis=0)
stds = np.std(train_values, axis=0)

stds[stds == 0] = 1.0

print(f"Computed scaling parameters:")
print(f"  Mean range: [{means.min():.4f}, {means.max():.4f}]")
print(f"  Std range: [{stds.min():.4f}, {stds.max():.4f}]")

X_train = (train_values - means) / stds
X_test = (test_values - means) / stds

print(f"Standardization complete!")
print(f"\nX_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")

print(f"\nX_train statistics:")
print(f"  Mean: {X_train.mean():.6f} (should be ~0)")
print(f"  Std: {X_train.std():.6f} (should be ~1)")
```

## **BAB V**

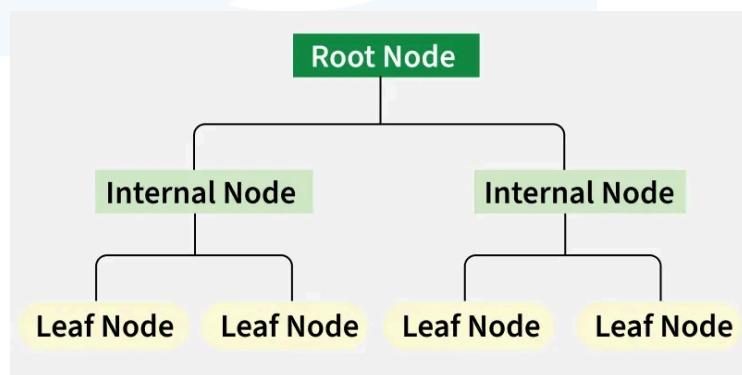
### **DESKRIPSI IMPLEMENTASI**

## 1. Algoritma *Decision Tree Learning (DTL)*

*Decision Tree Learning* adalah salah satu algoritma dalam *machine learning* yang memiliki *use case* yang luas dan mudah untuk diinterpretasikan. Algoritma ini dapat membantu dalam membuat keputusan dengan memetakan berbagai atribut dan melakukan konsiderasi terhadap hasil *output* yang mungkin.

### **1.1. Penjelasan Teori**

*Decision Tree* membantu membuat suatu keputusan dengan menunjukkan berbagai pilihan berbeda dan bagaimana pilihan tersebut saling berhubungan. Algoritma ini memiliki struktur seperti pohon yang dimulai dari akar atau *root node*, yang merupakan representasi dari seluruh dataset. Dari akar ini, pohon akan terus tumbuh berdasarkan *features* yang ada dalam *dataset*. Terdapat beberapa komponen dari pohon ini: *root node* sebagai *starting point*, *branches* sebagai garis yang menghubungkan antar *node* yang menunjukkan *flow* dari satu keputusan ke keputusan yang lain, *internal node* sebagai poin tempat dibuatnya keputusan, dan *leaf nodes* sebagai poin terakhir yang berisi keputusan final.



**Gambar 5.1 . Struktur *Decision Tree***

Bila ditinjau dari *target variables*-nya, terdapat dua tipe *Decision Tree*: *Classification Trees* yang digunakan untuk melakukan prediksi terhadap *outcome* yang kategorikal dan *Regression Trees* yang digunakan untuk melakukan prediksi terhadap *outcome* yang kontinu. Dalam kasus ini, karena

hasil *outcome* adalah *is\_fraud* yang dapat dikategorikan sebagai *true* dan *false*, **Decision Tree** yang akan digunakan adalah **Classification Trees**.

Dalam tugas besar ini, terdapat dua algoritma dalam *Decision Tree* yang dapat digunakan, yaitu C4.5 dan CART. Kami memutuskan untuk **memilih CART sebagai algoritma yang digunakan** karena algoritma ini bekerja sangat baik untuk *dataset* yang mengandung banyak data numerik, mengingat banyak dari kolom pada *dataset* merupakan data numerik yang kontinu. Algoritma CART dapat digunakan untuk *classification* dengan menggunakan *Gini Index* dan *regression* dengan menggunakan MSE. Karena data yang dihasilkan kategorikal, algoritma ini hanya akan menggunakan *Gini Index*. Algoritma CART dimulai dengan menggunakan *training set* sebagai *root node* yang akan dibagi menjadi dua *subset* dengan nilai *Gini Index* terbaik. Setelah itu, *subset* ini pun dibagi menjadi dua *sub-subset* dengan logika yang sama, secara rekursif hingga mencapai *leaves*. *Gini Index* bisa dihitung dengan formula berikut.

$$Gini = 1 - \sum_{i=1}^c (p_i)^2$$

$p_i$  = probabilitas objek diklasifikasikan menjadi suatu kelas lain

$c$  = jumlah kelas

## 1.2. Implementasi Algoritma

Berikut merupakan implementasi dari algoritma *Decision Tree Learning (DTL)* yang diterapkan dalam tugas besar ini. Sayangnya, *Decision Tree Learning (DTL)* masih kurang kuat dibanding algoritma *Logistic Regression*.

**Tabel 5.1.** Implementasi Algoritma *Decision Tree Learning (DTL)*

Fungsi	Deskripsi
<pre> <b>def __init__(self,</b>     max_depth=100,     min_samples=2):      self.max_depth =         max_depth      self.min_samples =         min_samples         self.tree =         None         self._y_dtype         = None         self._num_all_samples         = None         self._node_count = 0 </pre>	<p>Fungsi <code>__init__</code> merupakan konstruktor (fungsi yang dijalankan saat objek dibuat) dari kelas DTL. Konstruktor ini menyimpan beberapa variabel yang akan digunakan di fungsi lainnya dalam kelas DTL. Berikut merupakan beberapa variabel yang diinisiasi:</p> <ol style="list-style-type: none"> <li>1. <code>max_depth</code> merupakan batas maksimal kedalaman pohon, dalam kasus ini digunakan <i>depth</i> 10.</li> <li>2. <code>min_samples</code> merupakan jumlah minimal <i>sample</i> di <i>node</i> untuk <i>di-split</i>, dalam kasus ini digunakan <code>min_samples</code> 100.</li> <li>3. <code>tree</code> untuk menyimpan struktur <i>decision tree</i>. Di awal <code>None</code> karena belum dilatih.</li> <li>4. <code>_y_dtype</code> untuk menyimpan tipe data dari label (<i>y</i>)</li> <li>5. <code>_num_all_samples</code> untuk menyimpan jumlah total sampel dalam <i>dataset</i>, yang akan diisi ketika dimulai <i>training</i></li> <li>6. <code>_node_count</code> untuk menghitung jumlah <i>node</i> yang dibuat selama pohon dibangun, diawali dengan nilai 0 yang akan bertambah selama <i>tree</i> dibangun</li> </ol>

Fungsi	Deskripsi
<pre><b>def</b> <b>_set_df_type</b>(self, X, y, dtype):     X =     X.astype(dtype)     self._y_dtype = y.dtype     return X, y</pre>	Fungsi <b>_set_df_type</b> merupakan fungsi internal yang mengubah tipe data dari <i>X</i> (fitur) menjadi suatu tipe tertentu, bergantung pada parameter <i>dtype</i> dan menyimpan tipe data <i>y</i> . Ini akan mengembalikan nilai <i>X</i> dan <i>y</i> yang sudah diubah tipenya.
<pre><b>def</b> <b>_purity</b>(y):     unique_classes =     np.unique(y)     return     unique_classes.size == 1</pre>	Fungsi <b>def _purity</b> merupakan fungsi yang memeriksa apakah semua label di set <i>y</i> itu sama, atau memiliki hanya satu kelas unik. Dengan ini, bisa ditentukan apakah ini masih termasuk <i>node</i> atau sudah menjadi <i>leaf</i> . Akan dikembalikan <i>true</i> bila hanya satu kelas, dan <i>false</i> bila tidak.
<pre><b>def</b> <b>_is_leaf_node</b>(node):     return not     isinstance(node, dict)</pre>	Fungsi <b>_is_leaf_node</b> merupakan fungsi yang memeriksa apakah suatu <i>node</i> adalah <i>leaf</i> . DI sini, <i>leaf</i> direpresentasikan dalam <i>string</i> dan jika iya, akan <i>return true</i> . Jika masih dalam <i>dict</i> , dia bukan termasuk <i>leaf</i> sehingga <i>return false</i> .
<pre><b>def</b> <b>_leaf_node</b>(self, y):     class_index = 0     return     y.mode()[class_index]</pre>	Fungsi <b>_leaf_node</b> merupakan fungsi yang membuat nilai <i>leaf node</i> , yaitu modus dari kelas di <i>y</i> . Ini digunakan saat <i>node</i> tidak bisa di-split lagi, dengan <i>output</i> berupa nilai modusnya.

Fungsi	Deskripsi
<pre> <b>def _split_df(self,</b>     X, y, feature,     threshold):     feature_values =         X[feature]         left_indexes     = X[feature_values &lt;=         threshold].index         right_indexes     = X[feature_values &gt;         threshold].index         sizes =         np.array([left_indexe         s.size,         right_indexes.size])         return     self._leaf_node(y) if     any(sizes == 0) else     (left_indexes,     right_indexes) </pre>	<p>Fungsi <code>_split_df</code> merupakan fungsi yang membagi data <code>X</code> dan <code>y</code> berdasarkan fitur dan <code>threshold</code>. Data akan masuk kiri bila fitur di bawah <code>threshold</code> dan akan masuk kanan bila di atas <code>threshold</code>. <code>Return</code> akan berupa <i>tuple</i> indeks kiri dan kanan, atau berupa <i>leaf</i> bila ada sisi yang kosong.</p>
<pre> <b>def</b>     <u><b>_gini_impurity</b></u>(y):         _' counts_classes =     np.unique(y, return_counts=True)         squared_probabilities = </pre>	<p>Fungsi <code>_gini_impurity</code> merupakan fungsi yang mmenghitung seberapa <i>pure</i> kelas di set <code>y</code>. Semakin mendekati 0, artinya semakin murni. Hasil yang dihasilkan ada di <i>range</i> 0 sampai 1. Ini akan <code>return</code> nilai Gini yang merupakan <i>float</i> antara 0 dan 1.</p>

Fungsi	Deskripsi
<pre> np.square(counts_classes / y.size)         gini_impurity = 1 - sum(squared_probabilities)  return gini_impurity </pre>	
<pre> def _cost_function(self, left_df, right_df):     total_df_size = left_df.size + right_df.size     p_left_df = left_df.size / total_df_size     p_right_df = right_df.size / total_df_size     J_left = self._gini_impurity(left_df)     J_right = self._gini_impurity(right_df)     J = p_left_df*J_left + p_right_df*J_right     return J </pre>	<p>Fungsi <b>_cost_function</b> merupakan fungsi yang menghitung <i>cost</i> atau <i>weighted Gini</i> untuk melakukan <i>split</i>. Akan dilakukan <i>return</i> berupa nilai <i>cost</i> dari Gini, semakin kecil maka kualitas <i>split</i> makin baik.</p>

Fungsi	Deskripsi
<pre> <b>def _best_split(self,</b> X, y):     features = X.columns      min_cost_function = np.inf      best_feature, best_threshold = None, None      for feature in features:          unique_feature_values = np.unique(X[feature])          if len(unique_feature_va lues) &gt; 100:              indices = np.linspace(0, len(unique_feature_va lues)-1, 100, dtype=int)              unique_feature_values = </pre>	<p>Fungsi <code>_best_split</code> merupakan fungsi yang mencari fitur dan <i>threshold</i> terbaik untuk memberi nilai Gini terendah, dengan <i>threshold</i> merupakan nilai rata-rata dari nilai unik. Ini akan <i>return feature</i> dan <i>threshold</i> terbaik, atau <i>None</i> bila tidak ada yang bagus.</p>

Fungsi	Deskripsi
<pre>unique_feature_values [indices]          for i in range(1, len(unique_feature_va lues)):      current_value = unique_feature_values [i]      previous_value = unique_feature_values [i-1]      threshold = (current_value + previous_value) / 2      split_result = self._split_df(X, y, feature, threshold)      if isinstance(split_resu lt, tuple):          left_indexes, right_indexes =</pre>	

Fungsi	Deskripsi
<pre>split_result  left_labels, right_labels = y.loc[left_indexes], y.loc[right_indexes]  current_J = self._cost_function(left_labels, right_labels)  if current_J &lt;= min_cost_function:  min_cost_function = current_J  best_feature = feature  best_threshold = threshold  return best_feature, best_threshold</pre>	
<b>def _stopping_conditions(</b>	Fungsi <b>_stopping_conditions</b> merupakan fungsi yang memeriksa

Fungsi	Deskripsi
<pre> self, x, y, depth):         if depth &gt;= self.max_depth:             return True         if len(y) &lt; self.min_samples:             return True         if self._purity(y):             return True         return False     </pre>	<p>kondisi penghentian pembangunan pohon. Ini akan berdasarkan tiga <i>parameter</i>: kedalaman maksimum pohon, sampel yang jumlahnya kurang dari minimum, atau nilai <i>y</i> yang sudah murni. Apabila perlu <i>stop</i>, maka sudah termasuk <i>leaf</i> yang berarti <i>return true</i>. Dan apabila tidak, akan lanjut <i>split</i> dan <i>return false</i>.</p>
<pre> def _grow_tree(self, x, y, depth=0):     x, y = self._set_df_type(x, y, np.float64)      if self._stopping_conditions(x, y, depth):         return     str(self._leaf_node(y))     self._node_count += 1     if     </pre>	<p>Fungsi <code>_grow_tree</code> merupakan fungsi rekursif utama untuk membangun pohon. Pertama dilakukan set tipe data, lalu dicek <i>stopping</i>. Bila belum <i>stop</i>, dicari <i>best split</i> dan dibagi data. Setelah sudah, dibangun <i>subtree</i> kiri-kanan. Pohon akan direpresentasikan sebagai <i>dict</i> dengan <i>key</i> "feature &lt;= threshold   " dan <i>value list</i> [left, right]. Jika <i>subtree</i> sama, akan simplifikasi jadi <i>leaf</i>. Ada juga <i>print progress</i> tiap 50 <i>node</i> untuk mengetahui <i>progress</i>. Fungsi akan <i>return</i> struktur pohon yang berupa <i>dictionary</i> atau <i>string</i> apabila hanya tersisa <i>leaf</i>.</p>

Fungsi	Deskripsi
<pre>self._node_count % 50 == 0:      print(f"Progress: {self._node_count} nodes   depth: {depth}/{self.max_depth}   samples: {len(y)}")          best_feature, best_threshold = self._best_split(X, y)          if best_feature is None:             return str(self._leaf_node(y ))          decision_node = f'{best_feature} &lt;= {best_threshold}   '          split_result = self._split_df(X, y, best_feature, best_threshold)         if not</pre>	

Fungsi	Deskripsi
<pre>isinstance(split_resu lt, tuple):     return str(self._leaf_node(y ))          left_indexes, right_indexes = split_result         left_X, right_X = X.loc[left_indexes], X.loc[right_indexes]         left_labels, right_labels = y.loc[left_indexes], y.loc[right_indexes]          tree = {decision_node: []}         left_subtree = self._grow_tree(left_ X, left_labels, depth+1)         right_subtree = self._grow_tree(right _X, right_labels, depth+1)</pre>	

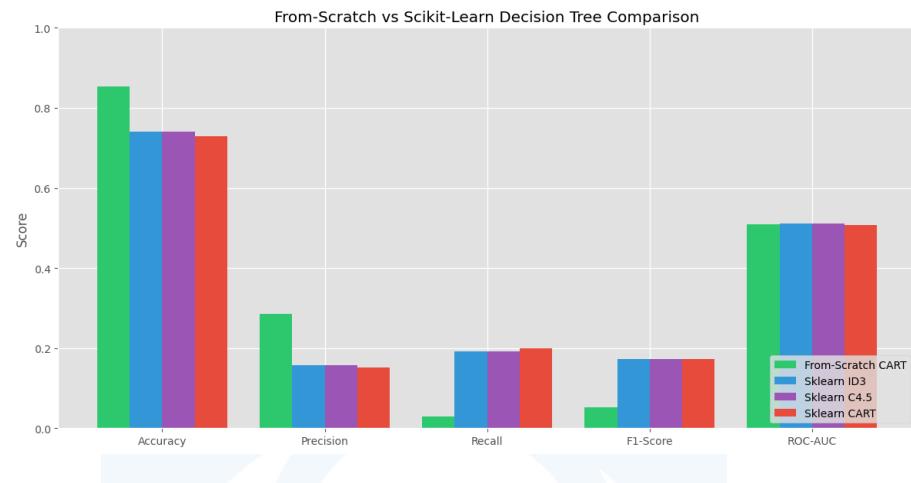
Fungsi	Deskripsi
<pre>         if left_subtree == right_subtree:     tree = left_subtree else:      tree[decision_node].e xtend([left_subtree, right_subtree])      return tree </pre>	
<pre> <b>def fit</b>(self, X, y):      self._num_all_samples = len(y)      self._node_count = 0      print(f"Parameters: max_depth={self.max_d epth}, min_samples={self.min _samples}")      self.tree = self._grow_tree(X, y)     print(f"Tree selesai! Nodes: {self._node_count}") </pre>	<p>Method <b>fit</b> merupakan <i>method</i> utama untuk menjalankan dan melatih model. Di sini dilakukan <i>inisialisasi counter</i> dan pembangunan pohon.</p>

Fungsi	Deskripsi
<pre>return self</pre>	
<pre>def     _traverse_tree(self, sample, tree):     if         self._is_leaf_node(tree):             leaf, *__ = tree.split()             return leaf      decision_node = next(iter(tree))     left_node, right_node = tree[decision_node]     feature, other = decision_node.split(' &lt;=')     threshold, *__ = other.split()     feature_value = sample[feature]      if np.float64(feature_va lue) &lt;= np.float64(threshold)</pre>	<p>Fungsi <code>_traverse_tree</code> merupakan fungsi <i>traversal</i> untuk <i>visit nodes</i> dalam memproses prediksi. Dari <i>root</i>, dicek <i>node</i> yang sesuai sampai ke <i>leaf</i>. Fungsi akan <i>return</i> hasil prediksi kelas yang berupa <i>string</i> dari <i>leaf</i>.</p>

Fungsi	Deskripsi
<pre> :     next_node = self._traverse_tree(s ample, left_node) else:     next_node = self._traverse_tree(s ample, right_node)  return next_node </pre>	
<pre> def predict(self, samples: pd.DataFrame):     results = samples.apply(self._t raverse_tree, args=(self.tree,), axis=1)  print(f"Prediction selesai!") return np.array(results.asty pe(self._y_dtype)) </pre>	<p>Method <b>predict</b> merupakan <i>method</i> untuk prediksi. Ini akan mengecek tiap data dan mendapat hasil prediksi, yang akan disimpan dalam bentuk <i>array</i> dan di-<i>return</i>.</p>

### 1.3. Hasil Implementasi

Berikut merupakan perbandingan hasil implementasi dari *Decision Tree Learning (DTL)* dengan dan tanpa pustaka.



Gambar 5.2. Perbandingan DTL

Dapat dilihat bahwa berdasarkan skor ROC-AUC, model CART yang diimplementasikan dari scratch menunjukkan kemampuan pemisahan kelas yang relatif sebanding dengan model Decision Tree Learning (DTL) lainnya. Namun, perbedaan pada nilai metrik evaluasi lain, seperti precision, recall, dan F1-score, mengindikasikan adanya ketidakseimbangan performa yang kemungkinan disebabkan oleh perbedaan dalam penentuan keputusan klasifikasi kelas, khususnya pada mekanisme konversi probabilitas menjadi label kelas. Selain itu, perbedaan performa tersebut juga dapat dipengaruhi oleh keterbatasan implementasi, di mana tidak seluruh hyperparameter yang tersedia pada DTL lain diimplementasikan dalam model ini.

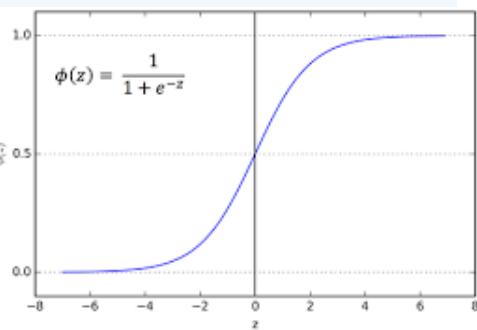
## 2. Algoritma *Logistic Regression*

*Logistic Regression* adalah salah satu algoritma *supervised machine learning* yang digunakan dalam permasalahan klasifikasi dengan melakukan prediksi probabilitas dari klasifikasi suatu *data input*.

### 2.1. Penjelasan Teori

*Logistic Regression* terdiri atas beberapa tipe, tetapi **tipe Logistic Regression yang digunakan dalam kasus ini adalah *Binomial Logistic***

**Regression** karena *dependent variable*, yaitu *is\_fraud*, hanya memiliki dua kemungkinan kategori, yaitu *true or false*. Prediksi dari probabilitas untuk kategorisasi ini menggunakan suatu fungsi yang bernama *Sigmoid Function*. Dengan fungsi ini, *output* dari model bisa dijadikan suatu angka probabilitas di antara 0 sampai 1. Selain itu, diperlukan juga suatu *threshold* untuk penentuan *labeling* dari kategori. Dalam kasus ini, apabila nilai probabilitas di atas *threshold*, dia akan dianggap *true* atau termasuk suatu aksi *fraud*. Sedangkan apabila nilai probabilitas di bawah *threshold*, dia akan dianggap *false* atau tidak termasuk aksi *fraud*. Untuk rumus dari *Sigmoid Function* adalah sebagai berikut.



**Gambar 5.3.** Sigmoid Function

$z$  = *input* dari data

*Logistic Regression* bekerja dengan melakukan transformasi dari fungsi *linear regression* pada nilai kontinu menjadi kategorikal dengan membuat fungsi yang terdiri atas berbagai variabel yang *independent* yang akan berhasil akhir menjadi *value* 0 atau 1, dikenal juga dengan nama *logistic function*. Fungsi ini termasuk *multi-linear function* yang dapat direpresentasikan dalam bentuk:

$$z = (\sum_{i=1}^n w_i x_i) + b$$

$w_i$  = *weights* atau koefisien

$x_i$  = observasi X yang ke-*i*

$b$  = bias atau *intercept*

$z$  = nilai kontinu dari regresi linear

Setelah didapat nilai  $z$ , dilakukan konversi dengan fungsi *Sigmoid* untuk

menjadi probabilitas dari angka 0 dan 1, digunakan untuk memprediksi kategorisasi kelas. Probabilitas tersebut dihitung dengan:

$$P(y = 1) = \sigma(z)$$

$$P(y = 0) = 1 - \sigma(z)$$

Terdapat beberapa metrik yang penting dalam implementasi dari *Logistic Regression*, antara lain adalah sebagai berikut.

### 1. Logistic Regression Equation and Odds

Dalam *Logistic Regression*, model memodelkan *odds* dari suatu kejadian. *Odds* di sini merujuk pada perbandingan probabilitas kejadian terhadap probabilitas tidak kejadian, dimodelkan sebagai eksponensial dari kombinasi linear *input* dan *parameter*, yang dapat dinyatakan sebagai berikut:

$$Odds = \frac{p(x)}{1-p(x)} = e^z = e^{w.X+b}$$

dengan  $z = w.X + b$  sebagai kombinasi linear fitur dan parameter model. Dengan mengambil logaritma natural dari *odds*, bisa didapat hubungan linear penuh yang disebut dengan *log-odds* atau *logit*, bentuk persamaan linear yang dioptimasi dalam *Logistic Regression* dengan detail sebagai berikut.

$$\log\left(\frac{p(x)}{1-p(x)}\right) = w.X + b$$

Apabila kedua sisi dibuat eksponensial, akan didapat hasil akhir sebagai *Final Logistic Regression*:

$$p(X; b; x) = \frac{e^{w.X+b}}{1+e^{w.X+b}} = \frac{1}{1+e^{-w.X+b}}$$

### 2. Likelihood Function for Logistic Regression

Dalam *Logistic Regression*, *weight* dan *bias* memiliki peran penting dalam menentukan hasil prediksi. Maka dari itu, perlu suatu fungsi yang dapat mencari *weights* dan *bias* yang bisa membuat model paling sesuai dengan data. Fungsi ini disebut dengan Maximum Likelihood Estimation (MLE) dengan rumus berikut.

$$\log L(w, b) = \sum_{i=1}^n [y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))]$$

### 3. Gradient of the Log-Likelihood Function

Dalam *Logistic Regression*, untuk menentukan parameter terbaik ( $w$  atau  $b$ ), kita memaksimalkan nilai *log-likelihood*. Hal ini bisa dilakukan dengan mencari gradien untuk mencari nilai parameter yang dapat memaksimalkan kesesuaian model dengan data. Turunan *log-likelihood* terhadap parameter  $w_j$  diperoleh sebagai berikut.

$$\frac{\partial J(l(b,w))}{\partial w_j} = \sum_{i=1}^n (y_i - p(x_i; b; w))x_{ij}$$

## 2.2. Implementasi Algoritma

Berikut merupakan implementasi algoritma yang didefinisikan berdasarkan fungsi dan tahapan dari implementasinya itu sendiri.

**Tabel 5.2.** Implementasi Algoritma *Logistic Regression*

Fungsi	Deskripsi
<pre> <b>def __init__(self,</b>     learning_rate=0.01,     n_iterations=1000,     optimizer="batch",     batch_size=None,     regularization=0.0,     l1_ratio=0.0,     class_weight=None,     lr_schedule="constant",     lr_decay=0.1,     lr_decay_steps=100,     momentum=0.0,     nesterov=False,     beta1=0.9,     beta2=0.999,     epsilon=1e-8,     use_focal_loss=False,     focal_gamma=2.0,     early_stopping=True,     patience=10,     tol=1e-5):     self.learning_rate = learning_rate </pre>	<p>Fungsi <code>__init__</code> merupakan konstruktor (fungsi yang dijalankan saat objek dibuat) dari kelas <i>Logistic Regression</i>. Konstruktor ini menyimpan beberapa variabel yang akan digunakan di fungsi lainnya dalam kelas <i>Logistic Regression</i>. Berikut merupakan beberapa variabel yang diinisiasi:</p> <ol style="list-style-type: none"> <li>1. <code>learning_rate</code> merupakan penentuan besar langkah pembaruan parameter pada tiap iterasi <i>training</i>.</li> <li>2. <code>initial_lr</code> menyimpan nilai <code>learning_rate</code> awal sebagai referensi.</li> <li>3. <code>n_iterations</code> untuk menentukan jumlah maksimum dari iterasi, atau <i>epoch</i>.</li> <li>4. <code>optimizer</code> menentukan metode optimisasi yang digunakan, yaitu <i>batch gradient descent</i>, <i>stochastic gradient descent</i>, <i>mini-batch gradient descent</i>, atau <i>Adam</i>.</li> <li>5. <code>batch_size</code> untuk menentukan ukuran dari <i>batch</i> pada metode <i>mini-batch</i> dan <i>Adam</i>.</li> <li>6. <code>regularization</code> untuk</li> </ol>

Fungsi	Deskripsi
<pre> self.initial_lr = learning_rate  self.n_iterations = n_iterations  self.optimizer = optimizer.lower()  self.regularization = regularization     self.l1_ratio = l1_ratio  self.class_weight = class_weight  self.lr_schedule = lr_schedule     self.lr_decay = lr_decay  self.lr_decay_steps = lr_decay_steps     self.momentum = momentum     self.nesterov = nesterov     self.beta1 = </pre>	<p>menentukan kekuatan regularisasi, untuk mencegah <i>overfitting</i>.</p> <p>7. l1_ratio untuk menentukan proporsi antara regularisasi L1 dan L2.</p> <p>8. class_weight untuk menangani ketidakseimbangan kelas dengan memberikan bobot pada tiap kelas..</p> <p>9. lr_schedule untuk penyesuaian <i>learning rate</i> selama <i>training</i>.</p> <p>10. lr_decay sebagai faktor pengurang <i>learning rate</i> pada skema penjadwalan.</p> <p>11. lr_decay_steps untuk menentukan interval iterasi untuk penurunan <i>learning rate</i>.</p> <p>12. momentum untuk mempercepat kovergensi dan mengurangi osilasi gradien.</p> <p>13. nesterov menentukan apakah metode <i>Nesterov Accelerated Gradient</i> digunakan.</p> <p>14. beta1, beta2 sebagai parameter momentum pada <i>optimizer Adam</i>.</p> <p>15. epsilon sebagai penjaga stabilitas numerik dalam <i>optimizer</i></p>

Fungsi	Deskripsi
<pre> beta1     self.beta2 = beta2     self.epsilon = epsilon  self.use_focal_loss = use_focal_loss  self.focal_gamma = focal_gamma  self.early_stopping = early_stopping     self.patience = patience     self.tol = tol      if batch_size is None:      self.batch_size = 32 if self.optimizer == "mini-batch" else None     else:      self.batch_size = batch_size </pre>	<p><i>Adam.</i></p> <p>16. <code>use_focal_loss</code> untuk menentukan apakah fungsi <i>loss</i> yang digunakan <i>Focal Loss</i>.</p> <p>17. <code>focal_gamma</code> sebagai parameter fokus pada <i>Focal Loss</i>.</p> <p>18. <code>early_stopping</code> untuk menentukan apakah proses <i>training</i> dihentikan lebih awal atau tidak jika perbaikan <i>loss</i> tidak terjadi.</p> <p>19. <code>patience</code> untuk menentukan jumlah iterasi maksimum tanpa perbaikan <i>loss</i>, sebelum <i>early stioopping</i>.</p> <p>20. <code>tol</code> sebagai ambang batas minimum perbaikan <i>loss</i>.</p> <p>21. <code>weights</code> sebagai bobot model <i>Logistic Regression</i> selama <i>training</i>.</p> <p>22. <code>bias</code> sebagai bias model yang ditambahkan di fungsi linear.</p> <p>23. <code>velocity_w</code>, <code>velocity_b</code> sebagai variabel kecepatan untuk mode optimasi yang <i>momentum-based</i></p> <p>24. <code>m_w</code>, <code>v_w</code>, <code>m_b</code>, <code>v_b</code> sebagai <i>first</i> dan <i>second moment</i> pada <i>optimizer Adam</i>.</p>

## Tugas Besar 2

### IF3070 Dasar Intelektualisasi Artificial

Fungsi	Deskripsi
<pre>         self.weights = None         self.bias = None  self.velocity_w = None  self.velocity_b = None          self.m_w = None         self.v_w = None         self.m_b = None         self.v_b = None          self.t = 0  self.class_weights_ = None  self.loss_history = [] </pre>	<p>25. <code>loss_history</code> menyimpan riwayat <i>loss</i>.</p> <p>26. <code>weight_history</code> menyimpan riwayat <i>weight</i>.</p> <p>27. <code>lr_history</code> menyimpan riwayat <i>learning rate</i>.</p>

Fungsi	Deskripsi
<pre>self.weight_history = [] self.lr_history = []</pre>	
<pre>def     _get_learning_rate(se         lf, iteration):             if                 self.lr_schedule ==                     "constant":                         return                 self.initial_lr             elif                 self.lr_schedule ==                     "step":                         return                 self.initial_lr *                     (self.lr_decay **                         (iteration //                             self.lr_decay_steps))             elif                 self.lr_schedule ==                     "exponential":                         return                 self.initial_lr *                     (self.lr_decay **                         (iteration /                             self.n_iterations))             elif</pre>	<p>Fungsi <code>_get_learning_rate</code> digunakan untuk menentukan nilai <i>learning rate</i> pada setiap iterasi, bergantung pada <i>learning rate schedule</i> yang dipilih. Fungsi akan <i>return</i> nilai <i>learning rate</i>, berdasarkan <i>schedule lr</i> yang digunakan.</p>

Fungsi	Deskripsi
<pre> self.lr_schedule == "cosine":          return self.initial_lr * (1 + np.cos(np.pi * iteration / self.n_iterations)) / 2  else:         return self.initial_lr </pre>	
<pre> <b>def sigmoid</b>(self, z):         z = np.clip(z, -500, 500)          positive_mask = z &gt;= 0         negative_mask = ~positive_mask          result = np.zeros_like(z, dtype=float)  result[positive_mask] = 1 / (1 + np.exp(-z[positive_ma sk])))  exp_z = np.exp(z[negative_mas k]) </pre>	<p>Fungsi <b>sigmoid</b> menerapkan fungsi <i>Sigmoid</i> dalam kode, dengan <i>return</i> nilai pada rentang 0 sampai 1.</p>

Fungsi	Deskripsi
<pre> k] )  result[negative_mask] = exp_z / (1 + exp_z)  return result </pre>	
<pre> <b>def</b> <b>compute_loss</b>(self, y_true, y_pred, sample_weights=None):     n_samples = len(y_true)      epsilon = 1e-15     y_pred = np.clip(y_pred, epsilon, 1 - epsilon)      if self.use_focal_loss:         p_t = np.where(y_true == 1, y_pred, 1 - y_pred)          focal_weight = (1 - p_t) ** self.focal_gamma          loss_per_sample = </pre>	Fungsi <b>compute_loss</b> menghitung nilai fungsi <i>loss</i> dari prediksi model, me-return nilai <i>loss</i> rata-rata.

Fungsi	Deskripsi
<pre>-focal_weight * np.log(p_t) else:  loss_per_sample = -(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))  if sample_weights is not None:     loss =     np.sum(sample_weights * loss_per_sample) /     np.sum(sample_weights ) else:     loss =     np.mean(loss_per_samp le)  if self.regularization &gt; 0 and self.weights is not None:     l1_term =     self.l1_ratio *     np.sum(np.abs(self.we</pre>	

Fungsi	Deskripsi
<pre> ights))          l2_term =         (1 - self.l1_ratio) *         0.5 *         np.sum(self.weights         ** 2)          loss +=         self.regularization *         (l1_term + l2_term)          return loss     </pre>	
<pre> def     _compute_gradients(se         lf, X, y, y_pred,         sample_weights=None):         n_samples =         len(y)          if             self.use_focal_loss:                 epsilon =                 1e-15                  y_pred_clipped =                 np.clip(y_pred,                 epsilon, 1 - epsilon)                 p_t =                 np.where(y == 1,                 y_pred_clipped, 1 -                 y_pred_clipped)     </pre>	<p>Fungsi <b>_compute_gradients</b> meenghitung gradien <i>loss</i> terhadap bobot dan bias. INi akan <i>return</i> gradien <i>weight and bias</i> (dw &amp; db).</p>

Fungsi	Deskripsi
<pre>        gamma = self.focal_gamma          focal_weight = (1 - p_t) ** gamma          grad_p = np.where(             y == 1,              -focal_weight * (gamma * (1 - y_pred_clipped) * np.log(y_pred_clipped + epsilon) + 1) / (y_pred_clipped + epsilon),              focal_weight * (gamma * y_pred_clipped * np.log(1 - y_pred_clipped + epsilon) + 1) / (1 - y_pred_clipped + epsilon) )          error =</pre>	

Fungsi	Deskripsi
<pre>grad_p * y_pred_clipped * (1 - y_pred_clipped) else:     error =     y_pred - y      if         sample_weights is not None:          weighted_error =         sample_weights *         error         dw =         np.dot(X.T,         weighted_error) /         np.sum(sample_weights )         db =         np.sum(weighted_error ) /         np.sum(sample_weights )     else:         dw = (1 / n_samples) *         np.dot(X.T, error)         db = (1 / n_samples) *</pre>	

Fungsi	Deskripsi
<pre> np.sum(error)          if self.regularization &gt; 0:              l2_grad = (1 - self.l1_ratio) * self.weights              l1_grad = self.l1_ratio * np.sign(self.weights)              dw += self.regularization * (l1_grad + l2_grad)              return dw, db </pre>	
<pre> <b>def</b> <b>_update_parameters</b>(se lf, dw, db, lr):         if self.momentum &gt; 0:              self.velocity_w = self.momentum * self.velocity_w - lr * dw              self.velocity_b = self.momentum * self.velocity_b - lr </pre>	<p>Fungsi <b>_update_parameters</b> digunakan untuk memperbarui <i>weight</i> dan <i>bias</i> berdasarkan gradien yang diperoleh, tanpa <i>return</i> apapun.</p>

Fungsi	Deskripsi
<pre> * db  self.weights += self.velocity_w         self.bias += self.velocity_b else:  self.weights -= lr * dw         self.bias -= lr * db </pre>	
<pre> <b>def fit(self, X, y):</b>         if isinstance(X, pd.DataFrame):         X = X.values         if isinstance(y, (pd.Series, pd.DataFrame)):         y = y.values.flatten()          n_samples, n_features = X.shape </pre>	<p>Fungsi <b>fit</b> merupakan fungsi utama yang digunakan dalam melatih <i>Logistic Regression</i>. Ini diawali dengan inisialisasi <i>parameter</i>, lalu dihitung bobot keals, dipilih <i>optimizer</i>, dan dijlnkan proses <i>training</i>. Terakhir, akan di-return objek model yang telah dilatih.</p>

Fungsi	Deskripsi
<pre>np.random.seed(42)          self.weights =         np.random.randn(n_features) * np.sqrt(2.0         / n_features)          self.bias =         0.0          self.velocity_w =         np.zeros(n_features)          self.velocity_b = 0.0          if         self.class_weight ==         "balanced":             n_classes         = 2          class_counts =         np.bincount(y.astype(         int))          self.class_weights_ =         n_samples /         (n_classes *         class_counts)</pre>	

Fungsi	Deskripsi
<pre>print(f"Class\nweights:\n{self.class_weights_}\")\n\nelse:\n\n    self.class_weights_ =\n    None\n\n        if\n        self.class_weights_\nis not None:\n\n            sample_weights =\n            np.where(y == 1,\n            self.class_weights_[1\n            ],\n            self.class_weights_[0\n            ])\n\n            else:\n\n                sample_weights = None\n\n\n        self.loss_history =\n        []\n\n        self.weight_history =\n        []</pre>	

Fungsi	Deskripsi
<pre>self.lr_history = []  self.weight_history.append(self.weights.co py())          if self.optimizer ==  "batch":   self._fit_batch(X, y, sample_weights)          elif self.optimizer ==  "sgd":   self._fit_sgd(X, y, sample_weights)          elif self.optimizer ==  "mini-batch":   self._fit_mini_batch( X, y, sample_weights)          elif self.optimizer ==  "adam":   self._fit_adam(X, y,</pre>	

Fungsi	Deskripsi
<pre> sample_weights) else:     raise ValueError(f"Unknown optimizer: {self.optimizer}.")  return self </pre>	
<pre> <b>def</b> <b>_cost_function</b>(self, left_df, right_df):     total_df_size = left_df.size + right_df.size     p_left_df = left_df.size / total_df_size     p_right_df = right_df.size / total_df_size     J_left = self._gini_impurity(l eft_df)     J_right = self._gini_impurity(r ight_df)     J = p_left_df*J_left + p_right_df*J_right     return J </pre>	<p>Fungsi <b>_cost_function</b> merupakan fungsi yang menghitung <i>cost</i> atau <i>weighted Gini</i> untuk melakukan <i>split</i>. Akan dilakukan <i>return</i> berupa nilai <i>cost</i> dari Gini, semakin kecil maka kualitas <i>split</i> makin baik.</p>

Fungsi	Deskripsi
<pre> <b>def</b> _fit_batch(self, X, Y, sample_weights=None):     best_loss = float('inf')      patience_counter = 0     best_weights = self.weights.copy()     best_bias = self.bias      <b>for</b> iteration in range(self.n_iteratio ns):     lr = self._get_learning_ra te(iteration)      self.lr_history.append(lr)      <b>if</b> self.nesterov and self.momentum &gt; 0:      self.weights += self.momentum * self.velocity_w </pre>	<p>Fungsi <b>_fit_batch</b> mengimplementasikan <i>Batch Gradient Descent</i> dengan seluruh data digunakan tiap iterasi, diperbarui <i>parameternya</i>.</p>

Fungsi	Deskripsi
<pre>self.bias += self.momentum * self.velocity_b  z = np.dot(X, self.weights) + self.bias  y_pred = self.sigmoid(z)  loss = self.compute_loss(y, y_pred, sample_weights)  self.loss_history.append(loss)  if self.early_stopping:     if loss &lt; best_loss - self.tol:      best_loss = loss  best_weights = self.weights.copy()</pre>	

Fungsi	Deskripsi
<pre>best_bias = self.bias  patience_counter = 0 else:      patience_counter += 1      if patience_counter     &gt;= self.patience:          print(f"Early               stopping at iteration               {iteration}, loss:               {loss:.6f}")          self.weights =         best_weights          self.bias = best_bias          break          dw, db =         self._compute_gradien         ts(X, y, y_pred,         sample_weights)          self._update_paramete         rs(dw, db, lr)</pre>	

Fungsi	Deskripsi
<pre>         if iteration % 10 == 0:  self.weight_history.a ppend(self.weights.co py())          if iteration % 50 == 0:  print(f"Iteration {iteration}: loss = {loss:.6f}, lr = {lr:.6f}")  self.weight_history.a ppend(self.weights.co py()) </pre>	
<pre> <b>def _fit_sgd</b>(self, X, Y, sample_weights=None):     n_samples = len(y)      best_loss = float('inf')      patience_counter = 0 </pre>	<p>Fungsi <b>_fit_sgd</b> mengimplementasikan <i>Stochastic Gradient Descent</i> dengan pembaruan <i>parameter</i> tiap sampel data.</p>

Fungsi	Deskripsi
<pre>        for iteration in range(self.n_iteratio ns):     lr = self._get_learning_ra te(iteration)      self.lr_history.append(lr)      indices = np.random.permutation (n_samples)      for i in indices:         xi = X[i:i+1]         yi = y[i:i+1]         sw_i = sample_weights[i:i+1] if sample_weights is not None else None          z = np.dot(xi, self.weights) +</pre>	

Fungsi	Deskripsi
<pre>self.bias  y_pred = self.sigmoid(z)  dw, db = self._compute_gradien ts(xi, yi, y_pred, sw_i)  self._update_paramete rs(dw, db, lr)  z_full = np.dot(X, self.weights) + self.bias  y_pred_full = self.sigmoid(z_full) loss = self.compute_loss(y, y_pred_full, sample_weights)  self.loss_history.app end(loss)  if self.early_stopping:</pre>	

Fungsi	Deskripsi
<pre>        if loss &lt; best_loss - self.tol:  best_loss = loss  patience_counter = 0 else:      patience_counter += 1      if patience_counter     &gt;= self.patience:          print(f"Early stopping at iteration {iteration}, loss: {loss:.6f}")          break          if iteration % 10 == 0:              self.weight_history.a ppend(self.weights.co py())          if iteration % 50 == 0:</pre>	

Fungsi	Deskripsi
<pre> print(f"Iteration {iteration}: loss = {loss:.6f}, lr = {lr:.6f}")  self.weight_history.a ppend(self.weights.co py()) </pre>	
<pre> <b>def</b> <b>_fit_mini_batch</b>(self, X, Y, sample_weights=None):     n_samples = len(Y)      batch_size = min(self.batch_size, n_samples)      best_loss = float('inf')      patience_counter = 0     best_weights = self.weights.copy()     best_bias = self.bias      for iteration in </pre>	<p>Fungsi <b>_fit_mini_batch</b> mengimplementasikan <i>Mini-Batch Gradient Descent</i> yang menggunakan sebagian data tiap iterasi.</p>

Fungsi	Deskripsi
<pre>range(self.n_iterations):     lr =     self._get_learning_rate(iteration)      self.lr_history.append(lr)      indices =     np.random.permutation(n_samples)      X_shuffled =     X[indices]      y_shuffled =     y[indices]      sw_shuffled =     sample_weights[indices] if sample_weights     is not None else None      for     start_idx in range(0,     n_samples,     batch_size):          end_idx =</pre>	

Fungsi	Deskripsi
<pre>min(start_idx + batch_size, n_samples)  x_batch = x_shuffled[start_idx: end_idx]  y_batch = y_shuffled[start_idx: end_idx]  sw_batch = sw_shuffled[start_idx :end_idx] if sw_shuffled is not None else None  if self.nesterov and self.momentum &gt; 0:      self.weights +=     self.momentum *     self.velocity_w      self.bias +=     self.momentum *     self.velocity_b</pre>	

Fungsi	Deskripsi
<pre>z = np.dot(X_batch, self.weights) + self.bias  y_pred = self.sigmoid(z)  dw, db = self._compute_gradien ts(X_batch, y_batch, y_pred, sw_batch)  self._update_paramete rs(dw, db, lr)  z_full = np.dot(X, self.weights) + self.bias  y_pred_full = self.sigmoid(z_full) loss = self.compute_loss(y, y_pred_full, sample_weights)</pre>	

Fungsi	Deskripsi
<pre>self.loss_history.append(loss)          if self.early_stopping:             if loss &lt; best_loss - self.tol:  best_loss = loss  best_weights = self.weights.copy()  best_bias = self.bias  patience_counter = 0             else:  patience_counter += 1  if patience_counter &gt;= self.patience:  print(f"Early stopping at iteration {iteration}, loss: {loss:.6f}")  self.weights =</pre>	

Fungsi	Deskripsi
<pre> best_weights  self.bias = best_bias  break      if iteration % 10 == 0:          self.weight_history.a         ppend(self.weights.co               py())          if iteration % 50 == 0:              print(f"Iteration {iteration}: loss = {loss:.6f}, lr = {lr:.6f}")          self.weight_history.a         ppend(self.weights.co               py()) </pre>	
<b>def __fit_adam(self,</b> X, y, sample_weights=None):     n_samples = len(y)	<p><i>Method __fit_adam</i></p> <p>mengimplementasikan <i>Adam optimizer</i> yang menggabungkan <i>momentum and adaptive learning rate</i>. Ini memperbarui <i>parameter</i> dengan estimasi <i>first moment</i></p>

Fungsi	Deskripsi
<pre> n_features = x.shape[1]  batch_size = self.batch_size if self.batch_size else 32  batch_size = min(batch_size, n_samples)  self.m_w = np.zeros(n_features)  self.v_w = np.zeros(n_features)  self.m_b = 0.0  self.v_b = 0.0  self.t = 0  best_loss = float('inf')  patience_counter = 0 best_weights = self.weights.copy() best_bias = self.bias  for iteration </pre>	<p>and second moment dari gradien, dilakukan juga <i>early stopping</i> agar tidak <i>overfit</i>.</p>

Fungsi	Deskripsi
<pre>in range(self.n_iterations):     lr = self._get_learning_rate(iteration)  self.lr_history.append(lr)  indices = np.random.permutation(n_samples)  X_shuffled = X[indices]  y_shuffled = y[indices]  sw_shuffled = sample_weights[indices] if sample_weights is not None else None  for start_idx in range(0, n_samples, batch_size):</pre>	

Fungsi	Deskripsi
<pre>end_idx = min(start_idx + batch_size, n_samples)  x_batch = x_shuffled[start_idx: end_idx]  y_batch = y_shuffled[start_idx: end_idx]  sw_batch = sw_shuffled[start_idx :end_idx] if sw_shuffled is not None else None  z = np.dot(X_batch, self.weights) + self.bias  y_pred = self.sigmoid(z)  dw, db =</pre>	

*Tugas Besar 2*

*IF3070 Dasar Intelelegensi Artifisial*

Fungsi	Deskripsi
<pre>self._compute_gradien ts(X_batch, y_batch, y_pred, sw_batch)  self.t += 1  self.m_w = self.beta1 * self.m_w + (1 - self.beta1) * dw  self.m_b = self.beta1 * self.m_b + (1 - self.beta1) * db  self.v_w = self.beta2 * self.v_w + (1 - self.beta2) * (dw ** 2)  self.v_b = self.beta2 * self.v_b + (1 - self.beta2) * (db ** 2)  m_w_corrected = self.m_w / (1 -</pre>	

Fungsi	Deskripsi
<pre>self.betal ** self.t)  m_b_corrected = self.m_b / (1 - self.betal ** self.t)  v_w_corrected = self.v_w / (1 - self.beta2 ** self.t)  v_b_corrected = self.v_b / (1 - self.beta2 ** self.t)  self.weights -= lr * m_w_corrected / (np.sqrt(v_w_correcte d) + self.epsilon)  self.bias -= lr * m_b_corrected / (np.sqrt(v_b_correcte d) + self.epsilon)  z_full = np.dot(X, self.weights) + self.bias</pre>	

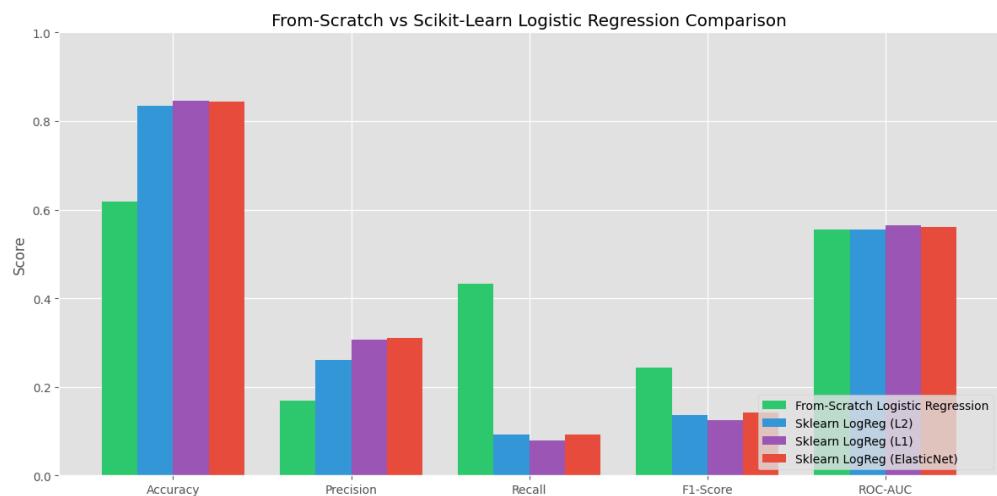
Fungsi	Deskripsi
<pre>y_pred_full = self.sigmoid(z_full) loss = self.compute_loss(y, y_pred_full, sample_weights)  self.loss_history.append(loss)  if self.early_stopping:     if loss &lt; best_loss - self.tol:      best_loss = loss      best_weights = self.weights.copy()      best_bias = self.bias      patience_counter = 0 else:      patience_counter += 1  if patience_counter &gt;= self.patience:</pre>	

Fungsi	Deskripsi
<pre>print(f"Early stopping at iteration {iteration}, loss: {loss:.6f}")  self.weights = best_weights  self.bias = best_bias  break  if iteration % 10 == 0:  self.weight_history.a ppend(self.weights.co py())  if iteration % 50 == 0:  print(f"Iteration {iteration}: loss = {loss:.6f}, lr = {lr:.6f}")  self.weight_history.a</pre>	

Fungsi	Deskripsi
ppend(self.weights.co py())	
<pre><b>def</b>  <b>predict_proba</b>(self,  X):      <b>if</b>  <b>isinstance</b>(X,  pd.DataFrame):          X =  X.values            z = np.dot(X,  self.weights) +  self.bias          <b>return</b>  self.sigmoid(z)</pre>	Fungsi <b>predict_proba</b> menghasilkan probabilitas prediksi untuk tiap sampel data yang dimasukkan, berdasarkan model <i>Logistic Regression</i> yang dilatih, dengan <i>return</i> berupa nilai probabilitas prediksi.
<pre><b>def predict</b>(self, X,  threshold=0.5):      probabilities =  self.predict_proba(X)      <b>return</b>  (probabilities &gt;=  threshold).astype(int  )</pre>	Fungsi <b>predict</b> menghasilkan label kelas akhir, berdasarkan probabilitas prediksi, dengan <i>return</i> dalam bentuk biner.

### 2.3. Hasil Implementasi

Berikut merupakan hasil perbandingan dari hasil implementasi dengan dan tanpa pustaka untuk algoritma *Logistic Regression*.



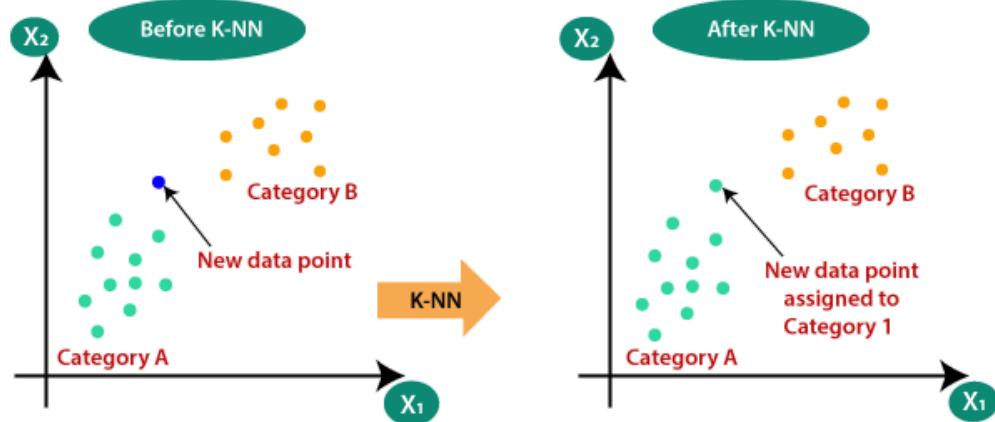
**Gambar 5.4.** Perbandingan *Logistic Regression*

Dapat dilihat bahwa algoritma yang diimplementasikan dari scratch menunjukkan nilai ROC-AUC yang relatif sebanding dengan implementasi scikit-learn. Selain itu, model from-scratch menghasilkan nilai F1-score dan recall yang lebih tinggi, namun peningkatan tersebut dicapai dengan mengorbankan precision dan accuracy. Hal ini mengindikasikan adanya trade-off antara kemampuan mendeteksi kelas positif dan tingkat kesalahan prediksi secara keseluruhan. Salah satu faktor yang berkontribusi terhadap perbedaan tersebut adalah penggunaan hyperparameter yang telah dituning serta penerapan solver Adam optimizer versi kustom pada algoritma from-scratch, yang dalam kasus ini menunjukkan kinerja yang lebih baik dibandingkan dengan konfigurasi default pada scikit-learn.

### 3. Algoritma *K-Nearest Neighbor (KNN)*

*K-Nearest Neighbor (KNN)* merupakan algoritma pengklasifikasi pembelajaran non-parametrik dan terawasi, yang menggunakan kedekatan untuk membuat klasifikasi atau prediksi tentang pengelompokan titik data individu.

### 3.1. Penjelasan Teori

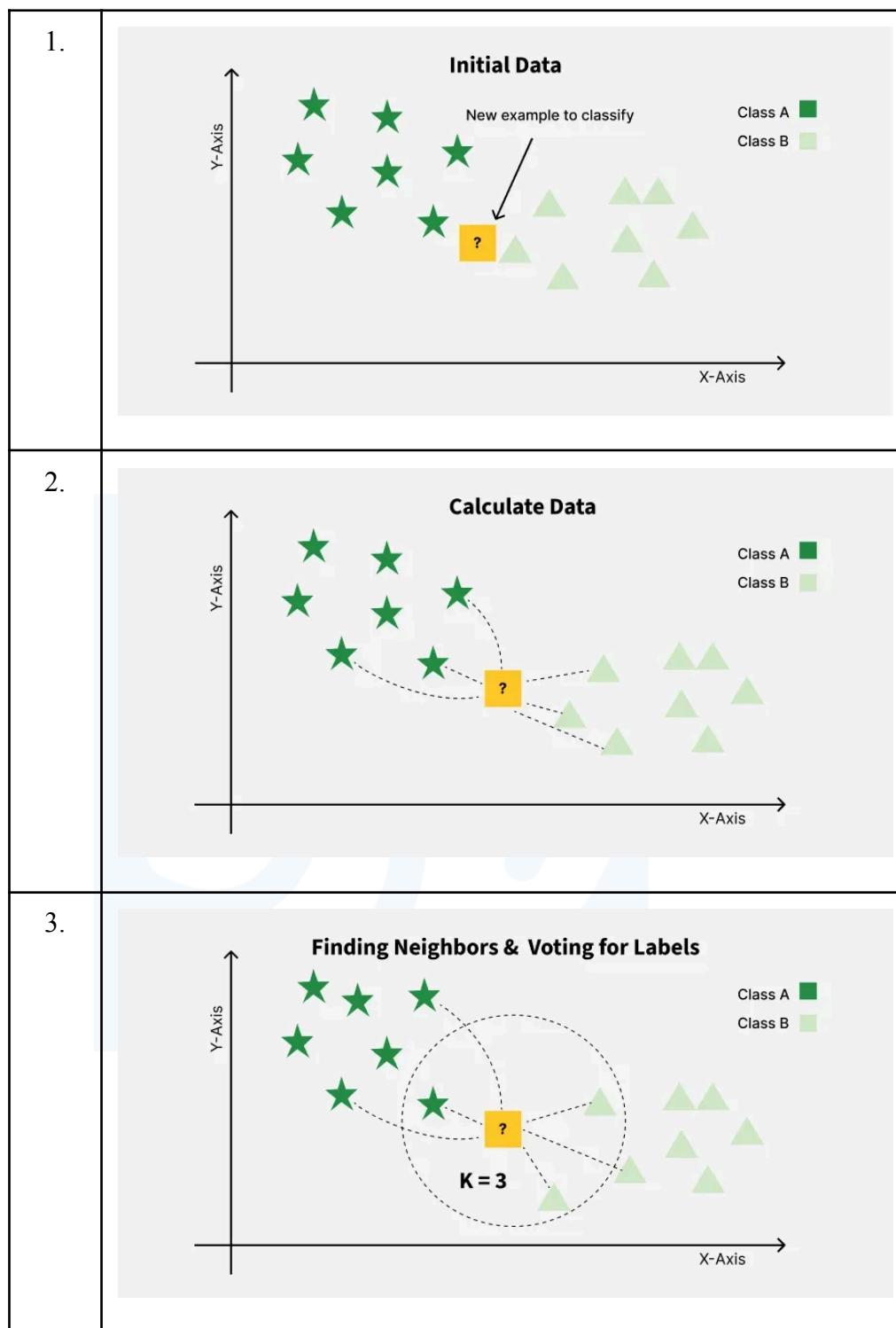


**Gambar 5.5. K-Nearest Neighbor (KNN)**

*K-Nearest Neighbor (KNN)* bekerja dengan cara mencari sejumlah tetangga terdekat dari data uji dan menentukan kelas data uji tersebut berdasarkan mayoritas kelas dari tetangga terdekat (data latih) yang ditemukan. Meskipun algoritma KNN dapat digunakan untuk masalah regresi atau klasifikasi, algoritma ini biasanya digunakan sebagai algoritma klasifikasi, dengan asumsi bahwa titik-titik yang serupa dapat ditemukan berdekatan satu sama lain. *K-Nearest Neighbor (KNN)* juga dikenal sebagai algoritma *lazy learner* karena tidak "belajar" dari *training set* secara langsung. *K-Nearest Neighbor (KNN)* akan menyimpan keseluruhan *dataset* dan melakukan komputasi hanya pada saat klasifikasi.

**Tabel 5.3.** Visualisasi Konsep KNN

No.	Visualisasi Tahapan
-----	---------------------



*K-Nearest Neighbor (KNN)* menggunakan metrik jarak untuk mengidentifikasi tetangga atau *neighbor* terdekat untuk klasifikasi dan regresi. Terdapat 3 metrik yang digunakan untuk identifikasi jarak tetangga, antara lain:

1. ***Euclidean Distance***

*Euclidean distance* atau jarak *euclidean* didefinisikan sebagai sebuah garis lurus yang berada di antara dua poin dalam sebuah ruang. Garis tersebut menjadi jarak terpendek dari dua poin tersebut. Berikut merupakan persamaan dari *euclidean distance*:

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^d (x_j - X_{i,j})^2}$$

## 2. *Manhattan Distance*

*Manhattan distance* adalah total jarak yang ditempuh jika hanya dapat bergerak ke arah horizontal dan vertikal, seperti menyusuri jalan-jalan kota yang berbentuk kotak-kotak (sebagaimana di kota Manhattan). *Manhattan distance* juga disebut sebagai *taxis cab distance* karena karena taksi hanya bisa berjalan mengikuti jalan kota yang tersusun seperti grid, bukan secara diagonal. Berikut merupakan persamaan dari *manhattan distance*:

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$

## 3. *Minkowski Distance*

*Minkowski distance* merupakan gabungan dari *euclidean distance* dan *manhattan distance* sebagai kasus khusus. Berikut merupakan persamaan dari *minkowski distance*:

$$d(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Berdasarkan persamaan tersebut, jika  $p = 2$ , maka hasil akan sama persis dengan *euclidean distance* (jarak lurus). Namun, jika  $p = 1$ , maka hasilnya akan berubah menjadi *manhattan distance*.

### 3.2. Implementasi Algoritma (Tanpa Library)

Berikut merupakan implementasi dari algoritma *K-Nearest Neighbor (KNN)* yang diterapkan dalam tugas besar ini. Sayangnya, *K-Nearest Neighbor (KNN)* menjadi algoritma terlelah jika dibandingkan dengan *Decision Tree Learning* dan *Logistic Regression*.

**Tabel 5.4.** Implementasi Algoritma *K-Nearest Neighbor (KNN)*

Fungsi	Deskripsi
<pre> <b>def</b> __init__(self, n_neighbors=5, weighted=False, metric='euclidean', p=2):      self.n_neighbors = n_neighbors     self.weighted = weighted     self.metric = metric     self.p = p     self.X_train = None     self.y_train = None </pre>	<p>Fungsi <code>__init__</code> merupakan konstruktor (fungsi yang dijalankan saat objek dibuat) dari kelas KNN. Konstruktor ini menyimpan beberapa variabel yang akan digunakan di fungsi lainnya dalam kelas KNN. Berikut merupakan beberapa variabel yang diinisiasi:</p> <ol style="list-style-type: none"> <li>1. <code>n_neighbors</code> merupakan jumlah tetangga terdekat yang akan dipakai. Dalam kelas ini, angka 5 dipilih sebagai jumlah <i>default</i> dari tetangga.</li> <li>2. <code>weighted=False</code> merupakan variabel yang digunakan untuk mengecek apakah tetangga diberi bobot atau tidak. Hal ini akan membantu menentukan jarak yang dekat akan lebih berpengaruh.</li> <li>3. <code>metric</code> menentukan jenis perhitungan jarak yang digunakan (<i>euclidean</i>, <i>manhattan</i>, atau <i>minkowski</i>).</li> <li>4. <code>p</code> yang diinisiasikan khusus untuk jarak minkowski. Biasanya jika <code>p</code> bernilai 2 maka metrik yang digunakan adalah <i>euclidean</i>.</li> <li>5. <code>self.X_train</code> merupakan</li> </ol>

	<p>tempat menyimpan data latih (fitur) yang awalnya kosong.</p> <p>6. <code>self.y_train</code> tempat menyimpan label data latih yang awalnya kosong.</p>
<pre>def     _calculate_distances(         self, x):         diff =             self.X_train - x             if                 self.metric ==                     'manhattan':                         return                     np.sum(np.abs(diff),                         axis=1)                 elif                     self.metric ==                         'minkowski':                             return                             np.power(np.sum(np.ab                                 s(diff) ** self.p,                                 axis=1), 1/self.p)                 else:                     return                     np.sqrt(np.sum(diff                         ** 2, axis=1))</pre>	<p>Fungsi <code>_calculate_distances</code> merupakan fungsi yang digunakan untuk menghitung jarak antara 1 data uji (<code>x</code>) dengan semua data latih (<code>X_train</code>).</p> <ol style="list-style-type: none"> <li>1. Terdapat <code>def _calculate_distances(self, x)</code> yang digunakan untuk mendefinisikan fungsi untuk menghitung jarak satu data uji.</li> <li>2. <code>diff = self.X_train - x</code> yang menghitung selisih setiap data latih dengan data uji</li> <li>3. <code>if self.metric == 'manhattan'</code> yang digunakan untuk mengecek apakah jarak yang digunakan adalah <i>Manhattan</i>.</li> <li>4. <code>np.sum(np.abs(diff), axis=1)</code> yang digunakan untuk menghitung jarak <i>Manhattan</i> dengan menjumlahkan nilai absolut per data latih.</li> <li>5. <code>elif self.metric == 'minkowski'</code> yang digunakan untuk mengecek apakah jarak</li> </ol>

	<p>yang digunakan adalah <i>Minkowski</i>.</p> <ol style="list-style-type: none"> <li>6. <code>np.power(np.sum(np.abs(diff) ** self.p, axis=1), 1/self.p)</code> yang menghitung jarak <i>Minkowski</i> dengan parameter p. Namun, jika tidak ditentukan, gunakan jarak <i>Euclidean (default)</i>.</li> <li>7. <code>np.sqrt(np.sum(diff ** 2, axis=1))</code> yang digunakan untuk menghitung jarak <i>Euclidean</i>, yaitu jarak garis lurus sebagai jarak terpendek.</li> <li>8. Terdapat <code>axis=1</code> yang dijadikan inisiasi untuk perhitungan yang dilakukan per baris data latih.</li> </ol>
<pre><b>def fit(self,</b>        X_train, y_train):     self.X_train     = np.array(X_train)     self.y_train     = np.array(y_train)</pre>	<p>Fungsi <code>fit</code> merupakan fungsi yang digunakan untuk menyimpan data latih dan labelnya ke dalam model agar dapat digunakan saat proses prediksi.</p> <ol style="list-style-type: none"> <li>1. <code>self.X_train = np.array(X_train)</code>, akan mengubah data latih menjadi <i>array NumPy</i> dan akan disimpan ke dalam model.</li> <li>2. <code>self.y_train = np.array(y_train)</code>, akan mengubah label data latih menjadi <i>array NumPy</i> dan menyimpannya</li> </ol>

	di model.
<pre> def predict(self, X, show_progress=False):     X =     np.array(X)     predictions =     []     n = len(X)     for i, x in     enumerate(X):         if         show_progress and (i + 1) % 1000 == 0:             print(f" Progress: {i+1}/{n}")      predictions.append(se lf._predict(x))     return     np.array(predictions) </pre>	<p>Fungsi <b>predict</b> merupakan fungsi yang digunakan untuk memprediksi label dari data uji (<math>x</math>) satu per satu. Fungsi <b>predict</b> menggunakan model yang sebelumnya sudah disesuaikan melalui fungsi <b>fit</b>.</p> <ol style="list-style-type: none"> <li>1. <math>X = \text{np.array}(X)</math> disini akan mengubah data uji menjadi <i>array NumPy</i> agar lebih mudah untuk diproses.</li> <li>2. <math>\text{predictions} = []</math>, akan menyiapkan <i>list</i> kosong untuk menyimpan hasil prediksi.</li> <li>3. <math>n = \text{len}(X)</math> di mana <math>n</math> akan digunakan untuk menyimpan jumlah data uji menggunakan <b>len</b>.</li> <li>4. <math>\text{for } i, x \text{ in } \text{enumerate}(X):</math> berfungsi untuk melakukan iterasi atau <i>loop</i> yang digunakan untuk memproses setiap data satu per satu.</li> <li>5. <math>\text{if } \text{show\_progress} \text{ and } (i + 1) \% 1000 == 0:</math>, <i>conditional</i> ini digunakan untuk mengecek apakah progress perlu ditampilkan setiap 1000 data.</li> <li>6. <math>\text{print}(f" Progress: {i+1}/{n}" )</math>, yang menampilkan progres prediksi ke</li> </ol>

	<p>layar.</p> <p>7. <code>predictions.append(self._predict(x))</code> digunakan untuk memanggil fungsi <code>_predict</code> yang sudah didefinisikan sebelumnya untuk satu data uji dan kemudian hasilnya akan disimpan.</p> <p>8. <code>return np.array(predictions)</code>, digunakan untuk mengembalikan semua hasil prediksi dalam bentuk <i>array NumPy</i>.</p>
<pre>def __predict(self, x):     distances = self._calculate_distances(x)     k_indices = np.argsort(distances)[:self.n_neighbors]     k_labels = self.y_train[k_indices]      if self.weighted:         weights = 1 / (distances[k_indices] + 1e-8)</pre>	<p>Fungsi <code>_predict</code> merupakan fungsi yang digunakan untuk memprediksi label dari satu data uji (<code>x</code>) dengan melihat <code>k</code> dari tetangga atau <i>neighbor</i> dari data latih.</p> <ol style="list-style-type: none"> <li>1. <code>distances = self._calculate_distances(x)</code>, digunakan untuk menghitung jarak antara <code>x</code> dan semua data latih yang tersedia.</li> <li>2. <code>k_indices = np.argsort(distances)[:, :self.n_neighbors]</code> yang digunakan untuk mengambil indeks <code>k</code> data latih dengan jarak yang paling kecil.</li> <li>3. <code>k_labels =</code></li> </ol>

```

class_weights = {}
for
label, weight in
zip(k_labels,
weights):
    class_weights[label] =
    class_weights.get(lab
el, 0) + weight
return
max(class_weights,
key=class_weights.get
)
else:
    return
Counter(k_labels).mos
t_common(1)[0][0]

```

`self.y_train[k_indices]`, yang digunakan untuk mengambil label dari k tetangga terdekat.

Fungsi ini menggunakan *conditionals* untuk menentukan proses selanjutnya,

1. Kondisi *True*, jika `weighted = True`, pada fungsi KNN (*K-Nearest Neighbors*) setiap tetangga diberi bobot berdasarkan jaraknya, sehingga tetangga yang lebih dekat memiliki pengaruh lebih besar. Bobot tersebut dijumlahkan untuk tiap kelas, lalu kelas dengan total bobot terbesar dipilih sebagai hasil prediksi.
2. Kondisi *False*, jika `weighted = False`, pada fungsi KNN (*K-Nearest Neighbors*) prediksi kelas dilakukan dengan *voting* mayoritas, yaitu mengambil label yang paling sering muncul di antara tetangga terdekat menggunakan `Counter(k_labels).most_common(1)[0][0]`.

```

def score(self, x,
y):
    return
np.mean(self.predict(

```

Fungsi `score` merupakan fungsi yang digunakan mengukur akurasi model. Akurasi model adalah seberapa banyak prediksi yang benar dibandingkan dengan

<pre>x) == y)</pre>	<p>label sebenarnya.</p> <ol style="list-style-type: none"> <li><code>self.predict(X)</code> dibandingkan dengan label asli y sehingga menghasilkan <i>array boolean</i>, lalu <code>np.mean(self.predict(X) == y)</code> menghitung proporsi nilai <i>True</i>, dan nilai tersebut dikembalikan sebagai akurasi model dalam rentang 0–1.</li> </ol>
<pre>def calculate_metrics(y_true, y_pred):     y_true, y_pred =         np.array(y_true),         np.array(y_pred)     tp =         np.sum((y_pred == 1) &amp; (y_true == 1))     tn =         np.sum((y_pred == 0) &amp; (y_true == 0))     fp =         np.sum((y_pred == 1) &amp; (y_true == 0))     fn =         np.sum((y_pred == 0) &amp; (y_true == 1))      accuracy = (tp + tn) / (tp + tn + fp +</pre>	<p>Fungsi <b>calculate_metrics</b> merupakan fungsi yang digunakan untuk menghitung metrik evaluasi model klasifikasi biner, seperti akurasi, <i>precision</i>, <i>recall</i>, dan <i>F1-score</i>.</p> <ol style="list-style-type: none"> <li>Nilai TP, TN, FP, dan FN dihitung dengan membandingkan setiap pasangan nilai antara <i>y_true</i> dan <i>y_pred</i>. TP adalah jumlah data yang diprediksi positif dan memang positif, TN adalah data yang diprediksi negatif dan memang negatif, FP adalah data yang diprediksi positif tetapi sebenarnya negatif, sedangkan FN adalah data yang diprediksi negatif tetapi sebenarnya positif.</li> <li><i>Accuracy</i> dihitung dengan membagi jumlah prediksi yang benar (TP + TN) dengan total</li> </ol>

```

fn)

    precision = tp / 
(tp + fp) if (tp + 
fp) > 0 else 0
    recall = tp / (tp 
+ fn) if (tp + fn) > 
0 else 0
    f1 = 2 * 
precision * recall / 
(precision + recall)
if (precision + 
recall) > 0 else 0

    return

{'accuracy': 
accuracy,
'precision': 
precision, 'recall': 
recall, 'f1_score': 
f1,

'confusion_matrix': 
{'TP': tp, 'TN': tn,
'FP': fp, 'FN': fn} }

```

seluruh data (TP + TN + FP + FN). Nilai ini menunjukkan seberapa sering model membuat prediksi yang benar secara keseluruhan.

3. *Precision* dihitung dengan membagi TP dengan jumlah seluruh prediksi positif (TP + FP). Metrik ini menunjukkan seberapa akurat prediksi positif yang dihasilkan model, dan akan bernilai 0 jika tidak ada prediksi positif untuk menghindari pembagian dengan nol.
4. *Recall* dihitung dengan membagi TP dengan jumlah seluruh data yang benar-benar positif (TP + FN). Nilai ini menunjukkan kemampuan model dalam menemukan atau menangkap kasus positif yang sebenarnya, dan di-set ke 0 jika tidak ada data positif.
5. Terakhir terdapat *F1-score* dihitung sebagai rata-rata harmonik antara *precision* dan *recall*, yaitu

$$\frac{precision \times recall}{precision + recall}$$

Metrik ini akan digunakan untuk menyeimbangkan *precision* dan *recall*.

	<p>6. <i>Confusion matrix</i> dikembalikan dalam bentuk ringkasan TP, TN, FP, dan FN untuk memberikan gambaran lengkap tentang jenis kesalahan dan keberhasilan model.</p>
<pre>def <b>find_best_k</b>(X_train, y_train, X_val, y_val, k_values=[3, 5, 7, 9, 11], sample_size=5000):     # Sample for     speed     if len(X_train) &gt;     sample_size:         np.random.seed(42)         idx =         np.random.choice(len(         X_train),         sample_size,         replace=False)         X_train_s,         y_train_s =         X_train[idx],         y_train[idx]     else:         X_train_s,         y_train_s = X_train,         y_train</pre>	<p>Fungsi <b>find_best_k</b> merupakan fungsi yang digunakan untuk mencari nilai <i>k</i> terbaik pada KNN dengan cara membandingkan performa (<i>F1-score</i>) beberapa nilai <i>k</i> menggunakan data validasi.</p> <ol style="list-style-type: none"> <li>1. <i>Sampling</i> data melalui (<i>X_train_s</i>, <i>y_train_s</i>, <i>X_val_s</i>, <i>y_val_s</i>) merupakan <i>sampling</i> yang mengambil sebagian data training dan validasi untuk mempercepat proses pencarian nilai <i>k</i>.</li> <li>2. <i>Training</i> &amp; prediksi (KNN, fit, predict) merupakan tahapan untuk memprediksi data validasi untuk setiap nilai di <i>k_values</i>, model KNN dibuat dan dilatih dengan <i>fit</i>.</li> <li>3. <i>F1-score</i> akan dihitung menggunakan <i>calculate_metrics</i>, dan nilai <i>k</i> dengan <i>F1-score</i> tertinggi disimpan sebagai <i>best_k</i> untuk</li> </ol>

```
    val_size =
min(1000, len(X_val))
    val_idx =
np.random.choice(len(
X_val), val_size,
replace=False)
    X_val_s, y_val_s
= X_val[val_idx],
y_val[val_idx]

    best_k, best_f1 =
k_values[0], 0
    for k in
k_values:
        knn =
KNN(n_neighbors=k,
weighted=True)

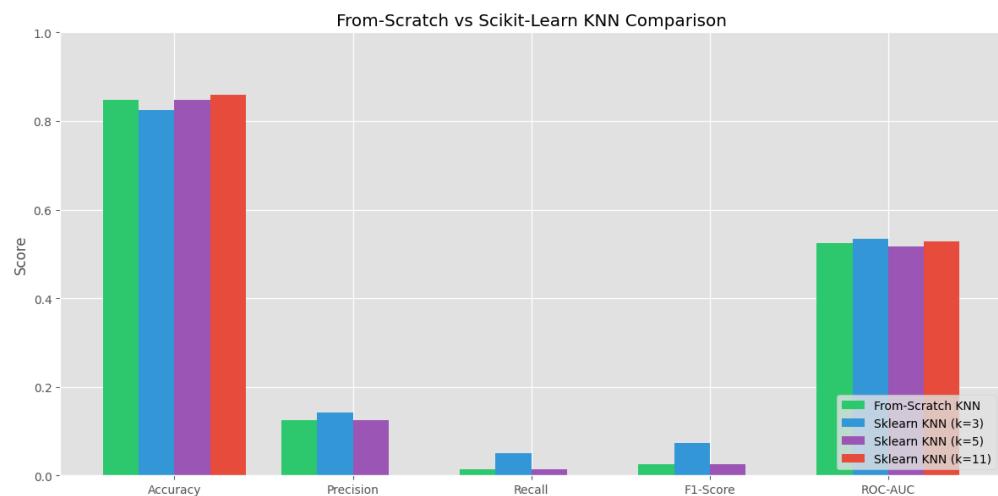
    knn.fit(X_train_s,
y_train_s)
        metrics =
calculate_metrics(y_v
al_s,
knn.predict(X_val_s))
        if
metrics['f1_score'] >
best_f1:
            best_f1,
best_k =
metrics['f1_score'],
k
```

mengakses dan memilih nilai terbaik dari  $k$ .

return best_k	
def <b>create_submission</b> (tes t_ids, predictions, filename='knn_submiss ion.csv'): submission = pd.DataFrame({'ID': test_ids, 'is_fraud': predictions})  submission.to_csv(fil ename, index=False)	Fungsi <b>create_submission</b> untuk membuat <i>file submission (CSV)</i> yang berisikan ID data <i>test</i> dan hasil prediksi model. File <i>submission.csv</i> inilah yang digunakan untuk di-submit ke dalam Kaggle.  1. pd.DataFrame({'ID': test_ids, 'is_fraud': predictions}) akan digunakan untuk membuat sebuah <i>DataFrame</i> dengan dua kolom, yaitu ID data <i>test</i> dan hasil prediksi.  2. submission.to_csv(filen ame, index=False) yang akan digunakan untuk menyimpan <i>DataFrame</i> ke <i>file CSV</i> tanpa menyertakan indeks.

### 3.3. Hasil Implementasi

Berikut merupakan hasil perbandingan dari hasil implementasi dengan dan tanpa pustaka untuk algoritma *K-Nearest Neighbor (KNN)*.



**Gambar 5.6.** Perbandingan KNN

Dapat dilihat bahwa berdasarkan skor ROC-AUC, model KNN yang diimplementasikan dari *scratch* menunjukkan kemampuan pemisahan kelas yang relatif sebanding dengan model KNN dari *scikit-learn*. Hal ini mengindikasikan bahwa meski implementasi dilakukan manual, performa model masih baik.

## BAB VI

### PEMBAGIAN TUGAS

#### 1. Pembagian Tugas

Berikut merupakan pembagian tugas dalam pengerojan tugas besar IF3070 Dasar Intelelegensi Artifisial.

**Tabel 6.1** Pembagian Tugas

NIM	Nama	Pembagian Tugas
182223003	Wisa Ahmaduta Dinutama	Mengerjakan <i>data cleaning</i> dan <i>preprocessing</i> Mengerjakan <i>error analysis</i> Memperbaiki dan optimasi kode dan model yang sudah dibuat (inisialisasi, EDA, dll) Menyatukan dan menyeleraskan <i>file-file</i> terpisah ke dalam <i>notebook</i>
182223033	Persada Ramiiza Abyudaya	Mengerjakan model <i>logistic regression</i> Melakukan optimasi dari <i>logistic regression</i> Menyatukan dan menyeleraskan <i>file-file</i> terpisah ke dalam <i>notebook final</i> Mengerjakan laporan Bab III
182223035	Inggried Amelia Deswenty	Mengerjakan EDA dan tahapan awal <i>notebook</i> Mengerjakan model <i>K-Nearest</i>

*Tugas Besar 2*

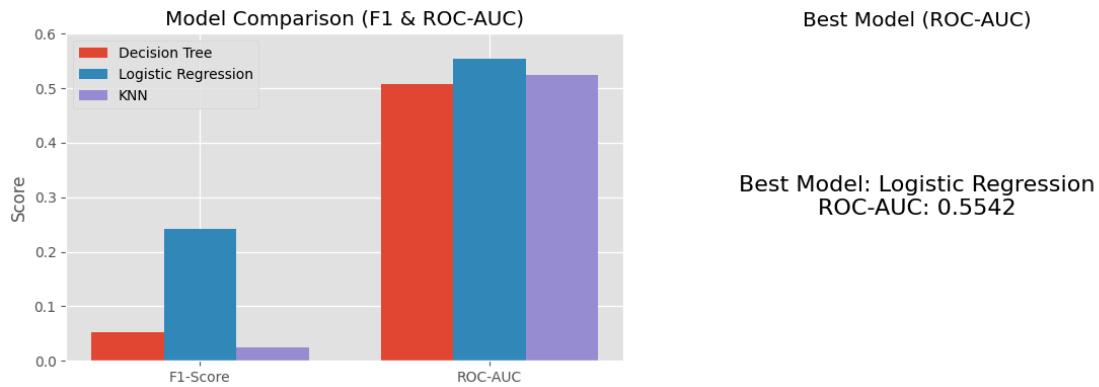
*IF3070 Dasar Intelelegensi Artifisial*

		<i>Neigbors (KNN)</i>
		Mengerjakan optimasi <i>K-Nearest Neigbors (KNN)</i>
		Mengerjakan <i>template</i> laporan dan laporan Bab I, II, IV, V, VI, dan VII
18223088	Wijaksara Aptaluhung	Mengerjakan model <i>Decision Tree Learning (DTL)</i>
		Mengerjakan optimasi model-model yang dibuat
		Mengerjakan <i>template</i> laporan dan laporan Bab I, III, IV, dan V

## BAB VII

### KESIMPULAN & SARAN

#### 1. Kesimpulan



Gambar 7.1. Model Comparison

Berdasarkan hasil evaluasi, *logistic regression* merupakan model yang memberikan kinerja paling baik dibandingkan model lainnya. Hal ini disebabkan oleh karakteristik data yang bersifat sintetis dan cenderung acak, sehingga model *non-linear* seperti *Decision Tree Learning (DTL)* dan KNN menjadi kurang stabil. Sebaliknya, *logistic regression* yang bersifat linear lebih mampu menangani pola data tersebut secara konsisten.

Secara umum, performa model *from-scratch* sudah sebanding dengan implementasi *scikit-learn*, khususnya jika dilihat dari metrik ROC-AUC. Perbedaan pada metrik lain seperti *accuracy*, *precision*, dan *recall* terutama disebabkan oleh belum diterapkannya *threshold tuning*. Selain itu, proses *preprocessing* memiliki peran yang sangat penting. *Feature scaling* terbukti meningkatkan performa model, terutama pada *logistic regression* yang sensitif terhadap perbedaan skala fitur. Penanganan *missing value* juga lebih efektif dilakukan dengan imputasi mean

Dengan demikian, dapat disimpulkan bahwa kombinasi *preprocessing* yang tepat dan pemilihan model yang sesuai, dalam hal ini *logistic regression*, merupakan kunci untuk memperoleh performa klasifikasi yang optimal.

## **2. Saran**

Berdasarkan implementasi, eksperimen, serta analisis lebih jauh terhadap seluruh model, didapatkan beberapa saran sebagai berikut.

1. Penggunaan *class weighting*, *over* atau *under sampling* dapat digunakan untuk menangani kasus yang tergolong sebagai kasus minoritas.
2. Pemilihan pemodelan dan penentuan prioritas optimasi model ditentukan sedari awal setelah melewati tahap *data understanding* dan *data preparation*.
3. Dapat diterapkannya *threshold tuning* untuk kasus serupa di masa depan.



## REFERENSI

Bhattacharjee, A. (1996). Implementing the k-nearest neighbors (KNN) algorithm from scratch in Python. Medium.

<https://medium.com/@avijit.bhattacharjee1996/implementing-the-k-nearest-neighbors-knn-algorithm-from-scratch-in-python-3b83a4fe8>

Elmenshawii, F. (n.d.). KNN from scratch. Kaggle.

<https://www.kaggle.com/code/fareselmenshawii/knn-from-scratch>

GeeksforGeeks. (n.d.). K-nearest neighbours (KNN) in machine learning.

<https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/>

IBM. (n.d.). Apa itu K-nearest neighbors (KNN)?

<https://www.ibm.com/id-id/think/topics/knn>

Irina, E. A. (n.d.). Algoritma k-nearest neighbor (KNN): Penjelasan dan implementasi untuk klasifikasi kanker. Medium.

<https://esairina.medium.com/algoritma-k-nearest-neighbor-knn-penjelasan-dan-implementasi-untuk-klasifikasi-kanker-ff9b7fbe0a4>

GeeksforGeeks. (2025, November 18). *Understanding logistic regression*. GeeksforGeeks.

<https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression/>  
[GeeksforGeeks](#)

Kaggle. (n.d.). *Decision Tree CART from Scratch – Full Tutorial*. Kaggle.

<https://www.kaggle.com/code/egazakharenko/decision-tree-cart-from-scratch-full-tutorial>  
Medium

FavTutor. (2021, June 9). *Gini impurity A-Z (decision tree)*. FavTutor.

<https://favtutor.com/blogs/gini-impurity> FavTutor

Patadiya, R. (2020). *Gini index & CART decision algorithm in machine learning*. Medium.

<https://medium.com/@riyapatadiya/gini-index-cart-decision-algorithm-in-machine-learning-1a4ed5d6140d> Medium

GeeksforGeeks. (2025). *Decision tree*. GeeksforGeeks.

<https://www.geeksforgeeks.org/machine-learning/decision-tree/> GeeksforGeeks

GeeksforGeeks. (2025, July 23). *Implementing CART (classification and regression tree) in Python*. GeeksforGeeks.

<https://www.geeksforgeeks.org/machine-learning/implementing-cart-classification-and-regression-tree-in-python/>

## LAMPIRAN



4 AbyuDAlya Ganbatte  0.62707 69 4h

Your Best Entry!  
Your most recent submission scored 0.62656, which is not an improvement of your previous score. Keep trying!

**AbyuDAlya Ganbatte SIGNING OUT from DAI!**