

TUGAS 2

Kecerdasan Artifisial Lanjut

Teknik Informatika Kelas B

Nama:

Gaung Taqwa Indraswara (235150207111043)

Muhammad Fauzan (235150201111044)

Ferrel Destatiananda Edwardo (235150207111044)

Muhammad Fatir Zaira (23515020011039)

Dosen:

Putra Pandu Adikara, S.Kom., M.Kom.



**Program Studi Teknik Informatika
Fakultas Ilmu Komputer
Universitas Brawijaya**

DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	3
PENDAHULUAN.....	3
1.1. Latar Belakang.....	3
1.2. Rumusan Masalah.....	4
1.3. Tujuan Penelitian.....	4
1.4. Manfaat Penelitian.....	5
BAB II.....	6
KAJIAN PUSTAKA.....	6
2.1. Machine Learning.....	6
2.2. Unsupervised Learning.....	6
2.3. Clustering.....	6
2.4. K-Means.....	6
2.5. Principal Component Analysis (PCA).....	7
2.6. Hierarchical Clustering.....	7
2.7. Segmentasi Pelanggan.....	7
BAB III.....	8
METODE PENELITIAN.....	8
3.1. Diagram Alur.....	8
3.1.1. Pengumpulan Data.....	8
3.1.2. Eksplorasi Data.....	8
3.1.3. Pre-pemrosesan Data.....	8
3.1.4. Modeling.....	9
3.1.5. Evaluasi Model.....	9
3.2. Teknik Pra-Pemrosesan.....	9
3.2.1. Raw Data.....	9
3.2.2. Drop Irrelevant Columns.....	10
3.2.3. Outlier Removal.....	10
3.2.4. Missing Value Handling.....	10
3.2.5. Data Type Casting.....	10
3.2.6. Encoding.....	10
3.2.7. Skewness Correction.....	11
3.2.8. Feature Scaling.....	11
3.2.9. Sampling.....	11
3.2.10. PCA.....	11
3.2.11. Preprocessed Data.....	11
3.3. Tools dan Library.....	12
BAB IV.....	13
IMPLEMENTASI.....	13
4.1. Import Libraries & Loading Data.....	13

4.2. Understanding Data.....	14
4.3. EDA (Exploratory Data Analysis).....	15
4.4. Preprocessing Data.....	28
4.5. Modelling.....	37
4.6. Evaluasi.....	49
4.7. PCA.....	61
4.8. Analisis Segmentasi Customer Hasil Clustering.....	64
BAB V.....	69
PEMBAHASAN & HASIL.....	69
5.1. Analisis Data.....	69
5.2. Fokus Evaluasi.....	69
5.3. Analisis Kinerja Model.....	69
5.3.1. Preprocessing Data.....	69
5.3.2. Reduksi Dimensi.....	69
5.3.3. K-Means Clustering.....	70
5.3.4. Agglomerative Hierarchical Clustering.....	70
5.3.5. Evaluasi Eksternal.....	70
5.4 Analisis Performa.....	70
5.5 Analisis Karakteristik Kluster.....	71
BAB VI.....	73
KESIMPULAN & SARAN.....	73
6.1 Kesimpulan.....	73
6.2 Saran.....	73
STUDI PUSTAKA.....	74

BAB I

PENDAHULUAN

1.1. Latar Belakang

Di era transformasi digital saat ini, perusahaan dituntut untuk dapat memahami pelanggan secara lebih mendalam guna menyusun strategi pemasaran yang lebih efektif dan efisien. Salah satu pendekatan yang semakin berkembang dalam mendukung proses tersebut adalah segmentasi pelanggan, yaitu proses mengelompokkan pelanggan ke dalam beberapa kelompok homogen berdasarkan karakteristik atau perilaku tertentu. Segmentasi ini menjadi dasar bagi pengambilan keputusan yang lebih tepat sasaran, seperti perancangan produk, penawaran promosi, dan pengelolaan loyalitas pelanggan (Setiyawan & Rofifudin, 2024).

Penerapan segmentasi pelanggan menjadi semakin penting terutama dalam konteks bisnis daring seperti platform *e-commerce*. Sebagai contoh, PT Global Loyalty Indonesia melalui aplikasi Alfagift menghadapi tantangan dalam mengidentifikasi kelompok pelanggan secara lebih spesifik karena belum memanfaatkan data riwayat transaksi pelanggan secara maksimal. Kurangnya strategi berbasis data ini berpotensi menyebabkan ketidakefisienan dalam pelaksanaan promosi dan loyalitas pelanggan yang rendah (Perdana et al., 2022). Oleh karena itu, dibutuhkan metode analisis data yang mampu mengolah data besar dan kompleks secara efisien untuk memperoleh wawasan yang bermakna.

Dalam bidang data mining, salah satu teknik yang sering digunakan untuk segmentasi pelanggan adalah algoritma *K-Means Clustering*. Algoritma ini populer karena kesederhanaannya dan efisiensi dalam menangani data skala besar. Metode ini telah banyak digunakan dalam berbagai bidang, mulai dari pemasaran, pendidikan, hingga pertanian (Rahmah, 2024; Hedyati & Suartana, 2021). Namun demikian, penerapan *K-Means* pada data berdimensi tinggi seringkali menemui kendala, seperti meningkatnya waktu komputasi, penurunan akurasi segmentasi, serta pembentukan klaster yang tidak optimal.

Selain metode partisi seperti *K-Means*, teknik *hierarchical clustering* juga merupakan pendekatan yang umum digunakan dalam segmentasi pelanggan. Berbeda dengan *K-Means* yang mewajibkan jumlah klaster ditentukan sejak awal, *hierarchical clustering* membentuk struktur pohon (dendrogram) yang memungkinkan analisis hierarki dari hubungan antar data. Abdurrahman (2019) menjelaskan bahwa algoritma *hierarchical clustering* terdiri dari dua pendekatan utama: *divisive* atau *top-down*, yang memulai dari satu klaster besar yang kemudian dipecah-pecah, dan *agglomerative* atau *bottom-up*, yang dimulai dari setiap data sebagai klaster tunggal dan digabung secara bertahap berdasarkan kedekatan. Karakteristik *hierarchical clustering* membuatnya sangat cocok untuk memahami struktur alami dalam data pelanggan, meskipun metode ini umumnya memiliki waktu komputasi yang lebih tinggi dibanding pendekatan partisi.

Untuk mengatasi permasalahan tersebut, *Principal Component Analysis* (PCA) menjadi salah satu solusi yang dapat diterapkan. PCA merupakan metode reduksi dimensi yang mampu menyederhanakan kompleksitas data tanpa kehilangan informasi penting. Dalam konteks *clustering*, PCA dapat meningkatkan kualitas klaster, mempercepat proses komputasi, serta mengurangi

redundansi antar variabel (Hediyati & Suartana, 2021). Penelitian-penelitian sebelumnya menunjukkan bahwa integrasi antara PCA dan *K-Means* dapat menghasilkan segmentasi yang lebih baik dibandingkan hanya menggunakan *K-Means* secara tradisional.

Selain itu, permasalahan teknis lain yang sering muncul dalam penerapan *K-Means* adalah penentuan jumlah kluster yang optimal dan sensitivitas terhadap inisialisasi *centroid*. Permasalahan ini mempengaruhi hasil akhir klusterisasi dan akurasi segmentasi. Beberapa pendekatan seperti metode *Elbow* dan algoritma *K-Means++* telah diusulkan untuk mengatasi permasalahan tersebut (Rahmah, 2024; Narayana et al., 2022). Dalam praktiknya, kombinasi *K-Means* dengan teknik preprocessing seperti PCA, serta pemilihan jumlah kluster yang tepat, terbukti memberikan hasil segmentasi yang lebih stabil dan akurat.

Dengan mempertimbangkan pentingnya segmentasi pelanggan serta tantangan teknis dalam penerapan algoritma clustering, maka penelitian ini dilakukan untuk mengeksplorasi penerapan *K-Means Clustering* yang dioptimalkan dengan PCA dalam proses segmentasi pelanggan. Penelitian ini diharapkan mampu memberikan kontribusi terhadap praktik analisis data pelanggan yang lebih efektif serta mendukung pengambilan keputusan bisnis berbasis data.

1.2. Rumusan Masalah

1. Bagaimana penerapan algoritma *K-Means Clustering* dalam proses segmentasi pelanggan berdasarkan data historis pelanggan?
2. Bagaimana penerapan algoritma *Hierarchical Clustering* dalam proses segmentasi berdasarkan data historis pelanggan?
3. Apakah integrasi *Principal Component Analysis* (PCA) dapat meningkatkan skor evaluasi model *clustering* yang dihasilkan oleh algoritma *K-Means*?
4. Berapa jumlah kluster optimal yang dapat diperoleh dari data pelanggan melalui pendekatan metode *Elbow*?
5. Bagaimana karakteristik masing-masing segmen pelanggan yang terbentuk berdasarkan hasil klusterisasi?

1.3. Tujuan Penelitian

1. Menerapkan algoritma *K-Means Clustering* untuk melakukan segmentasi pelanggan berdasarkan data riwayat transaksi.
2. Menerapkan algoritma *Hierarchical Clustering* dalam proses segmentasi berdasarkan data historis pelanggan.
3. Mengintegrasikan metode *Principal Component Analysis* (PCA) dalam proses pre-processing untuk mengoptimalkan performa algoritma clustering.
4. Menentukan jumlah kluster optimal menggunakan metode *Elbow* dalam proses segmentasi pelanggan.
5. Menganalisis dan menginterpretasikan karakteristik dari setiap segmen pelanggan yang terbentuk guna mendukung strategi pemasaran yang lebih tepat sasaran.

1.4. Manfaat Penelitian

Penelitian ini dapat memberikan kontribusi dalam pengembangan ilmu pengetahuan di bidang data mining dan machine learning, khususnya pada integrasi antara algoritma clustering dengan teknik reduksi dimensi seperti PCA dalam konteks segmentasi pelanggan.

Bagi pelaku bisnis atau perusahaan, hasil penelitian ini dapat dijadikan acuan dalam menyusun strategi pemasaran berbasis data yang lebih efisien, meningkatkan pemahaman terhadap perilaku pelanggan, serta mendukung pengambilan keputusan dalam pengelompokan pelanggan secara lebih akurat dan terarah.

BAB II

KAJIAN PUSTAKA

2.1. *Machine Learning*

Machine Learning adalah bagian dari kecerdasan buatan yang memungkinkan sistem komputer belajar dari data dan membuat keputusan tanpa diprogram secara eksplisit. *Machine Learning* bekerja dengan mencari pola dalam data untuk menghasilkan prediksi atau klasifikasi. Menurut Saraswat dan Raj (2021), *Machine Learning* terbagi menjadi tiga pendekatan utama, yaitu supervised learning, unsupervised learning, dan reinforcement learning. ML banyak digunakan karena kemampuannya mengolah data dalam skala besar dan menghasilkan wawasan yang berguna di berbagai bidang.

2.2. *Unsupervised Learning*

Unsupervised learning merupakan pendekatan dalam *machine learning* yang digunakan saat data tidak memiliki label. Tujuannya adalah menemukan struktur tersembunyi dalam data, seperti pengelompokan objek yang mirip atau reduksi dimensi. Saraswat dan Raj (2021) menyebut dua teknik utama dalam pendekatan ini, yaitu *K-Means* untuk clustering dan *Principal Component Analysis* (PCA) untuk reduksi dimensi. Pendekatan ini sangat bermanfaat dalam kasus eksplorasi data, seperti segmentasi pelanggan, di mana pola atau kelompok belum diketahui sebelumnya.

2.3. *Clustering*

Clustering adalah metode dalam *unsupervised learning* yang digunakan untuk mengelompokkan data ke dalam beberapa klaster berdasarkan kemiripan karakteristik. Tujuan utama dari teknik ini adalah untuk menemukan pola atau struktur tersembunyi dalam data tanpa adanya label. Proses clustering memungkinkan analisis data secara lebih terarah dengan mengidentifikasi kelompok-kelompok yang memiliki perilaku atau atribut serupa, sehingga sangat berguna dalam berbagai aplikasi seperti segmentasi pelanggan, analisis pasar, dan deteksi anomali (Saligkaras & Papageorgiou, 2021).

2.4. *K-Means*

Algoritma *K-Means* merupakan salah satu metode klasterisasi non-hierarkis yang banyak digunakan dalam analisis data. Teknik ini bekerja dengan membagi data ke dalam sejumlah klaster berdasarkan kemiripan jarak terhadap pusat klaster atau *centroid*. *K-Means* bersifat eksploratif dan efisien, serta dapat digunakan untuk berbagai jenis data berdimensi satu maupun dua. Kelebihan utama dari algoritma ini adalah kemampuannya mengurangi subjektivitas manusia dalam klasifikasi, karena

menggunakan pendekatan kuantitatif berbasis jarak sebagai ukuran kemiripan antar data (Nascimento, 2022).

2.5. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) adalah teknik reduksi dimensi yang mengubah fitur data ke dalam bentuk komponen yang tidak saling berkorelasi, dengan tetap mempertahankan informasi terpenting. PCA menjadi sangat penting dalam machine learning karena mampu menyederhanakan data kompleks tanpa mengurangi akurasi secara signifikan. Modalavalasa dan Makkena (2020) menunjukkan bahwa penerapan PCA secara langsung meningkatkan akurasi dan efisiensi berbagai algoritma pembelajaran, terutama saat fitur-fitur yang tidak relevan atau redundan dieliminasi lebih awal.

2.6. Hierarchical Clustering

Hierarchical Clustering merupakan salah satu metode clustering dalam *unsupervised learning* yang mengelompokkan data ke dalam struktur hirarki berdasarkan kemiripan antar objek. Menurut Abdurrahman (2019), *hierarchical clustering* memiliki karakteristik bahwa setiap data harus termasuk dalam suatu kluster tertentu dan data tersebut tidak dapat berpindah ke kluster lain selama proses *clustering* berlangsung. *Hierarchical clustering* terbagi menjadi dua jenis utama, yaitu *divisive (top-down)* dan *agglomerative (bottom-up)*. Pada pendekatan *divisive*, proses dimulai dengan satu kluster besar yang kemudian dipecah secara bertahap menjadi kluster-kluster lebih kecil. Sebaliknya, pada pendekatan *agglomerative*, setiap data dimulai sebagai kluster tunggal yang kemudian secara bertahap digabungkan menjadi kluster yang lebih besar hingga seluruh data tergabung dalam satu kluster.

2.7. Segmentasi Pelanggan

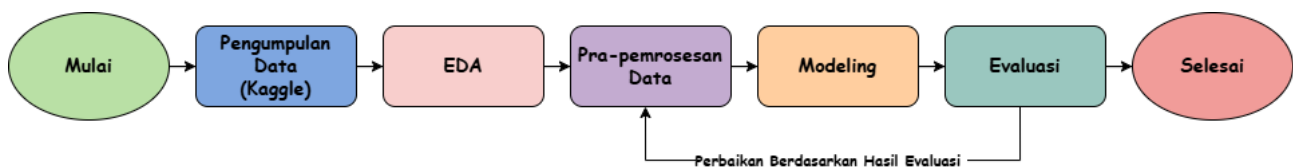
Segmentasi pelanggan adalah proses membagi pelanggan ke dalam kelompok-kelompok yang memiliki karakteristik, perilaku, atau kebutuhan yang serupa. Tujuannya adalah untuk memahami perbedaan di antara pelanggan agar strategi pemasaran dan layanan dapat disesuaikan secara lebih efektif. Menurut Abbas et al. (2021), pendekatan ini memungkinkan organisasi untuk mengeksplorasi pola tersembunyi dalam data pelanggan dan meningkatkan efisiensi dalam pengambilan keputusan berbasis data. Dalam konteks bisnis, segmentasi pelanggan membantu perusahaan mencapai tujuan strategis seperti meningkatkan loyalitas pelanggan, mengoptimalkan alokasi anggaran pemasaran, serta meningkatkan retensi dan akuisisi. Abbas et al. (2021) menunjukkan bahwa segmentasi yang tepat dapat secara signifikan memperkuat efektivitas komunikasi pemasaran dan meningkatkan profitabilitas. Segmentasi memainkan peran penting dalam strategi pemasaran berbasis data karena memungkinkan personalisasi yang lebih tepat sasaran.

BAB III

METODE PENELITIAN

3.1. Diagram Alur

Diagram alur berikut menggambarkan tahapan utama dalam proses pembuatan model prediksi churn pelanggan, dimulai dari pengumpulan data hingga evaluasi dan perbaikan model jika diperlukan. Diagram ini membantu memahami urutan dan hubungan antar proses dalam proyek machine learning.



3.1.1. Pengumpulan Data

Dataset yang digunakan adalah *Customer Segmentation* yang diperoleh dari situs Kaggle. *Dataset* ini berisi demografis konsumen, perilaku konsumen, dan segmentasi yang merupakan label target berupa klasifikasi segmen pelanggan yang terbagi menjadi empat kelompok (A, B, C, dan D). *Dataset* ini dapat dimanfaatkan untuk kasus klasifikasi pelanggan maupun *clustering* guna mengelompokkan pelanggan berdasarkan karakteristik yang serupa.

3.1.2. Eksplorasi Data

Melakukan eksplorasi data untuk memahami karakteristik *dataset* yang digunakan. Menganalisis data untuk mengidentifikasi pola dan anomali pada data. Dari hasil analisis, dapat ditentukan perlakuan yang diperlukan untuk mempersiapkan data di tahap selanjutnya. Langkah ini juga membantu dalam pemilihan fitur yang paling relevan untuk membangun model prediktif.

3.1.3. Pre-pemrosesan Data

Tahap ini bertujuan untuk membersihkan dan mempersiapkan data agar siap digunakan dalam pemodelan. Langkah yang dilakukan mencakup penanganan missing value, penghapusan outlier, encoding fitur kategorikal, normalisasi fitur numerik, serta reduksi dimensi dengan PCA sebagai pendekatan alternatif. Preprocessing yang baik dapat meningkatkan akurasi dan efisiensi model secara signifikan (Almeida et al., 2021).

3.1.4. Modeling

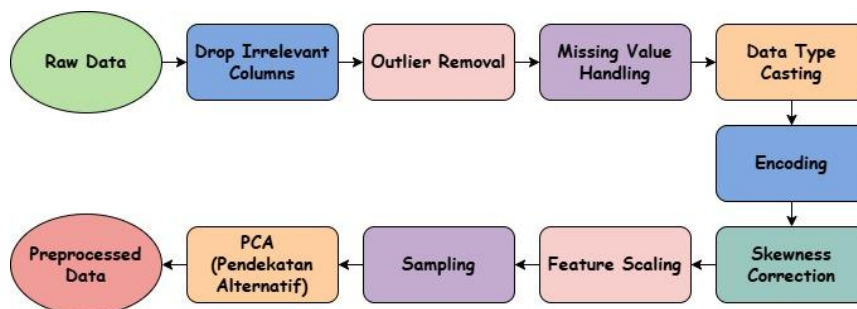
Tahap ini merupakan proses pemodelan metode unsupervised learning, dengan algoritma K-Means dan Hierarchical Clustering. Model digunakan untuk mengelompokkan pelanggan ke dalam segmen berdasarkan kemiripan karakteristik. Pemilihan metode didasarkan pada efektivitas algoritma dalam menangkap struktur klaster dalam data yang tidak berlabel (Sun et al., 2021).

3.1.5. Evaluasi Model

Evaluasi dilakukan untuk menilai kualitas hasil *clustering* menggunakan metrik internal seperti Silhouette Score, Davies-Bouldin Index, dan Calinski-Harabasz Index, serta metrik eksternal seperti Adjusted Rand Index dan Purity Score. Penggunaan berbagai metrik ini memberikan gambaran yang lebih komprehensif terhadap validitas dan konsistensi hasil *clustering* (Zhou et al., 2020).

3.2. Teknik Pra-Pemrosesan

Diagram teknik pra-pemrosesan merinci langkah-langkah yang dilakukan untuk membersihkan dan menyiapkan data mentah sebelum digunakan dalam pemodelan. Setiap tahap, mulai dari drop data yang tidak relevan, penanganan nilai yang hilang dan tahap pra-pemrosesan lain yang diperlukan. Tahap ini bertujuan untuk memastikan data dalam kondisi optimal untuk analisis dan pelatihan model.



3.2.1. Raw Data

Raw data atau data mentah merupakan data asli yang diperoleh langsung sumber dan belum melalui proses pengolahan. Data mentah sering memiliki berbagai kekurangan seperti nilai yang hilang, data duplikat, pencilan, dan sebagainya. Memahami data mentah menjadi langkah awal dalam *pipeline* analisis data untuk menentukan pemrosesan yang tepat.

3.2.2. Drop Irrelevant Columns

Drop irrelevant columns merupakan tahap dimana kolom yang tidak memiliki kontribusi bermakna dalam proses *clustering* dihapus dari *dataset*. Penghapusan fitur yang tidak relevan merupakan praktik umum yang dilakukan dalam pra-pemrosesan untuk mengurangi noise dan memperbaiki kinerja model (Almeida et al., 2021)

3.2.3. Outlier Removal

Deteksi dan penghapusan pencilan data dilakukan menggunakan metode *Interquartile Range* (IQR). Metode ini efektif dalam penanganan *outlier* karena mengukur sebaran data tengah tanpa terpengaruh nilai ekstrem, sehingga dapat mengidentifikasi data yang jauh dari distribusi utama secara *robust* (Rousseeuw & Hubert, 2011).

3.2.4. Missing Value Handling

Missing Value Handling merupakan tahap penanganan terhadap nilai yang hilang. Penanganan nilai yang hilang dapat berupa pengisian data dengan rata-rata, median, atau modus, tergantung pada tipe data dan distribusinya. Tujuan dari *handling missing value* adalah untuk memastikan bahwa analisis data atau proses machine learning berjalan dengan akurat dan efisien tanpa terganggu oleh data yang tidak lengkap.

3.2.5. Data Type Casting

Data type casting merupakan tahap dimana kolom dengan tipe data float diubah menjadi integer karena nilainya bersifat diskret. Perubahan tipe data dilakukan supaya pada tahap pembuatan model tidak memakan banyak ram dan memakan waktu yang lebih. Perubahan tipe data ini dilakukan untuk menjaga keakuratan semantik dan validitas analisis, serta untuk menghindari kesalahan dalam proses encoding dan pemodelan selanjutnya (Kaur & Singh, 2021).

3.2.6. Encoding

Encoding merupakan transformasi data kategorik menjadi format numerik yang dapat diproses oleh algoritma *machine learning*, kebanyakan algoritma tidak dapat langsung memproses data tekstual sehingga teknik *encoding* perlu digunakan. Beberapa teknik *encoding* yang umum digunakan adalah *one-hot encoding*, *label encoding*, dan *target encoding*. Pemilihan teknik encoding yang didasari pada kardinalitas fitur, hubungan antar kategori, dan kebutuhan algoritma yang digunakan.

3.2.7. Skewness Correction

Skewness Correction merupakan tahap pengoreksian terhadap fitur yang distribusi datanya tidak normal atau *skewed*. Distribusi yang tidak normal dikoreksi menggunakan transformasi logaritmik pada fitur numerik. Koreksi ini bertujuan untuk mengurangi ketidakseimbangan data yang dapat menyebabkan menurunnya performa model.

3.2.8. Feature Scaling

Feature Scaling adalah proses menyesuaikan rentang nilai fitur numerik agar berada dalam skala yang sebanding. Tahap ini mencegah fitur dengan nilai besar mendominasi yang bernilai kecil dalam perhitungan jarak atau bobot. Metode umum termasuk *Min-Max Scaling*, *Standardization*, dan *Robust Scaling*. *Scaling* sangat penting untuk algoritma yang sensitif terhadap skala dan metode berbasis gradien. *Scaling* mengurangi jarak antara unit data dengan membuatnya lebih generik (Sharma, 2022).

3.2.9. Sampling

Untuk mengurangi kompleksitas komputasi metode Hierarchical Clustering yang memiliki kompleksitas $O(n^3)$ terutama pada model yang dibuat secara manual, dilakukan sampling terhadap data hasil preprocessing. Teknik stratified sampling digunakan agar distribusi proporsi dari masing-masing label tetap terjaga. Sampling mempercepat proses clustering tanpa mengorbankan representativitas data (Sharma & Mirajkar, 2020).

3.2.10. PCA

Principal Component Analysis (PCA) adalah teknik reduksi dimensi yang digunakan untuk menyederhanakan data berdimensi tinggi dengan memproyeksikannya ke dalam sejumlah komponen utama yang mempertahankan variansi terbesar dari data asli. Dalam konteks clustering, PCA berguna untuk meningkatkan efisiensi komputasi, mengurangi noise, serta membantu visualisasi struktur data dalam ruang dua atau tiga dimensi. Meskipun terjadi pengurangan jumlah fitur, informasi penting dalam data tetap dipertahankan sebanyak mungkin sehingga model tetap akurat dan interpretatif (Jolliffe & Cadima, 2016; Sun et al., 2021).

3.2.11. Preprocessed Data

Preprocessed data adalah data yang telah melewati tahapan pembersihan dan transformasi. Pada tahap ini, data siap untuk digunakan dalam pemodelan. Data ini memiliki format yang konsisten, bebas dari anomali, dan strukturnya telah disesuaikan dengan kebutuhan dari algoritma yang akan diterapkan. Kualitas *preprocessed data* memengaruhi performa model

akhir, sehingga proses pra-pemrosesan harus dilakukan dengan cermat dengan mempertimbangkan karakteristik data dan tujuan analisis.

3.3. Tools dan Library

Dalam pengerjaan digunakan beberapa *tools* dan *library* dari Python untuk membantu proses analisis dan pembuatan model *machine learning*. Berikut adalah *tools* dan *library* yang digunakan:

1. Python
Bahasa pemrograman utama yang digunakan untuk mengembangkan model machine learning dan melakukan analisis data.
2. Pandas
Digunakan untuk manipulasi dan analisis data, termasuk membaca file CSV, menangani data yang hilang, serta melakukan encoding terhadap data kategori.
3. NumPy
Digunakan untuk operasi matematika dan array numerik, mendukung komputasi numerik yang efisien.
4. Matplotlib dan Seaborn
Kedua *library* ini digunakan untuk visualisasi data, membantu memahami distribusi data dan hubungan antar fitur.
5. Scikit-learn
Library utama yang digunakan untuk membangun model machine learning mencakup beberapa komponen penting, seperti *preprocessing* data menggunakan *StandardScaler*, *OneHotEncoder*, dan *LabelEncoder*; pembangunan model menggunakan *K-Means*, dan *AgglomerativeClustering*; evaluasi performa model dengan *silhouette_score*, *davies_bouldin_score*, *calinski_harabasz_score*, dan *adjusted_rand_score*.

BAB IV

IMPLEMENTASI

4.1. *Import Libraries & Loading Data*

a. Kode

```
import pandas as pd
import kagglehub
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
import numpy as np
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score, adjusted_rand_score
from random import randint
import copy

# Download latest version
path = kagglehub.dataset_download("vetrirah/customer")

print("Path to dataset files:", path)

print(os.listdir(path))

data_train = pd.read_csv(os.path.join(path, "Train.csv"))
data_test = pd.read_csv(os.path.join(path, "Test.csv"))
sample_submission = pd.read_csv(os.path.join(path, "sample_submission.csv"))
```

b. Output

```
▼ IMPORT LIBRARIES

[1] import pandas as pd
import kagglehub
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
import numpy as np
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score, adjusted_rand_score
from random import randint
import copy

▼ LOADING DATA

[2] # Download latest version
path = kagglehub.dataset_download("vetrirah/customer")

print("Path to dataset files:", path)

Downloading from https://www.kaggle.com/api/v1/datasets/download/vetrirah/customer?dataset_version_number=1...
100%|██████████| 105k/105k [00:00<00:00, 45.1MB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/vetrirah/customer/versions/1

[3] print(os.listdir(path))

['Test.csv', 'sample_submission.csv', 'Train.csv']

[4] data_train = pd.read_csv(os.path.join(path, "Train.csv"))
data_test = pd.read_csv(os.path.join(path, "Test.csv"))
sample_submission = pd.read_csv(os.path.join(path, "sample_submission.csv"))
```

4.2. Understanding Data**a. Kode**

```
data_train.head()

data_test.head()

sample_submission.head()
```

b. Output

Deskripsi Dataset

- data_train = digunakan untuk training model
- data_test = digunakan untuk menguji performa model yang telah dilatih.
- sample_submission = contoh sample submission atau output yang diberikan oleh sumber (kaggle)

`data_train.head()`

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation
0	462809	Male	No	22	No	Healthcare	1.0	Low	4.0	Cat_4	D
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	3.0	Cat_4	A
2	468315	Female	Yes	67	Yes	Engineer	1.0	Low	1.0	Cat_6	B
3	461735	Male	Yes	67	Yes	Lawyer	0.0	High	2.0	Cat_6	B
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	6.0	Cat_6	A

`data_test.head()`

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
0	458989	Female	Yes	36	Yes	Engineer	0.0	Low	1.0	Cat_6
1	458994	Male	Yes	37	Yes	Healthcare	8.0	Average	4.0	Cat_6
2	458996	Female	Yes	69	No	NaN	0.0	Low	1.0	Cat_6
3	459000	Male	Yes	59	No	Executive	11.0	High	2.0	Cat_6
4	459001	Female	No	19	No	Marketing	NaN	Low	4.0	Cat_6

`sample_submission.head()`

	ID	Segmentation
0	458989	A
1	458994	A
2	458996	A
3	459000	A
4	459001	A

4.3. EDA (Exploratory Data Analysis)

a. Kode

```
# Persiapan data untuk EDA
num = data_train.select_dtypes(include=['int64', 'float64'])
num = num.drop(columns=['ID'])
cat = data_train.select_dtypes(include=['object', 'category'])
cat = cat.drop(columns=['Segmentation'])
```

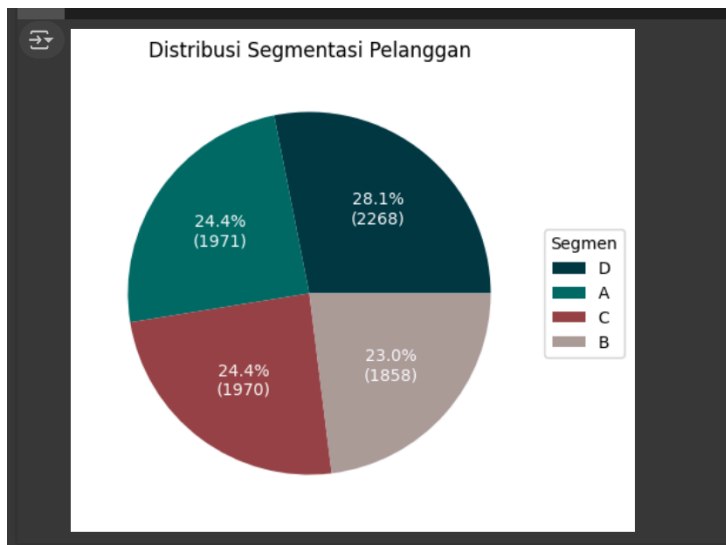

4.3.1. Distribusi Segmentasi

a. Kode

```
segment_counts = data_train['Segmentation'].value_counts()
colors = ['#003844', '#006C67', '#984447', '#AD9B9A']
plt.figure(figsize=(5, 5))
plt.pie(segment_counts,
        labels=segment_counts.index,
        colors=colors,
        textprops={'color': 'white'},
        autopct=lambda p: f'{p:.1f}%\n({int(p * sum(segment_counts)/100)})')

labels = [f'{label} " for label, count in zip(segment_counts.index, segment_counts)]
plt.legend(labels, title="Segmen", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))
plt.title('Distribusi Segmentasi Pelanggan')
plt.show()
```

b. Output



c. Penjelasan

Dari *pie chart* terlihat persebaran cukup merata untuk setiap kelas segmentasi yang diberikan di train set, dengan segmen 'D' yang paling dominan.

4.3.2. Distribusi Numerik

4.3.2.1. Fitur Numerik

a. Kode

```
data_train[num.columns].describe()

print("Unique values dan jumlahnya untuk Work_Experience:")
print(data_train['Work_Experience'].value_counts())

print("\nUnique values dan jumlahnya untuk Family_Size:")
print(data_train['Family_Size'].value_counts())

print("\nUnique values dan jumlahnya untuk Age:")
print(data_train['Age'].value_counts())

# Hitung jumlah unique values untuk Work_Experience dan Family_Size
unique_work = data_train['Work_Experience'].nunique()
unique_family = data_train['Family_Size'].nunique()
unique_age = data_train['Age'].nunique()

hist_color = '#003844'
box_color = '#984447'

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

# Plot Work_Experience
plt.subplot(2, 2, 1)
sns.histplot(
    data_train['Work_Experience'],
    bins=unique_work,
    discrete=True,
    color=hist_color,
    edgecolor='white',
)
plt.title('Distribusi Work_Experience')
plt.xticks(range(0, 15))

# Plot Family_Size
```

```
plt.subplot(2, 2, 2)
sns.histplot(
    data_train['Family_Size'],
    bins=unique_family,
    discrete=True,
    color=hist_color,
    edgecolor='white',
)
plt.title('Distribusi Family_Size')
plt.xticks(range(1, 10))

# Plot Age
plt.subplot(2, 2, (3,4))
sns.histplot(
    data_train['Age'],
    bins=7,
    color=hist_color,
    edgecolor='white',
)
plt.title('Distribusi Age')

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 8))
for i, col in enumerate(num.columns):
    plt.subplot(3, 3, i+1)
    sns.boxplot(y=data_train[col], color=box_color)
    plt.title(f'Boxplot {col}')
    plt.tight_layout()
plt.show()
```

b. *Output*

	Age	Work_Experience	Family_Size
count	8068.000000	7239.000000	7733.000000
mean	43.466906	2.641663	2.850123
std	16.711696	3.406763	1.531413
min	18.000000	0.000000	1.000000
25%	30.000000	0.000000	2.000000
50%	40.000000	1.000000	3.000000
75%	53.000000	4.000000	4.000000
max	89.000000	14.000000	9.000000

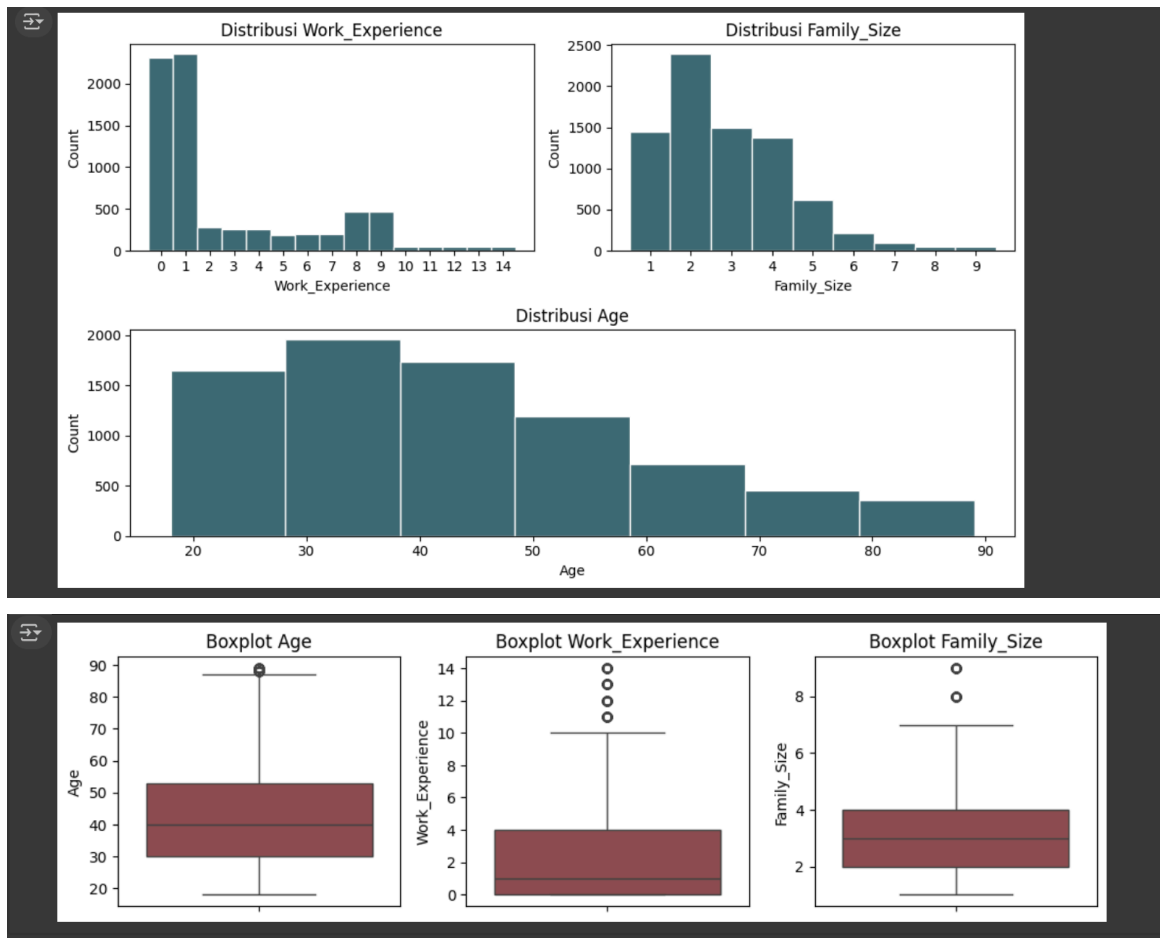
```

Unique values dan jumlahnya untuk Work_Experience:
Work_Experience
1.0    2354
0.0    2318
9.0     474
8.0     463
2.0     286
3.0     255
4.0     253
6.0     204
7.0     196
5.0     194
10.0     53
11.0     50
12.0     48
13.0     46
14.0     45
Name: count, dtype: int64

Unique values dan jumlahnya untuk Family_Size:
Family_Size
2.0    2390
3.0    1497
1.0    1453
4.0    1379
5.0     612
6.0     212
7.0      96
8.0      50
9.0      44
Name: count, dtype: int64

Unique values dan jumlahnya untuk Age:
Age
35     250
37     234
42     232
33     232
27     229
...
78      29
87      28
76      27
80      24
85      22
Name: count, Length: 67, dtype: int64

```



c. Penjelasan

Berdasarkan hasil visualisasi histogram dan boxplot dari ketiga fitur *Age*, *Work_Experience*, dan *Family_Size*, kami memperoleh beberapa *insight* penting untuk keperluan *preprocessing*. Distribusi *Age* terlihat *right-skewed* dan juga menunjukkan kehadiran sedikit outlier pada kelompok usia lanjut (85++). *Work_Experience* sangat condong ke kiri (*right skew*) dengan mayoritas data berada pada rentang 0–2 tahun, serta terdapat banyak *outlier* pada nilai tinggi. Selain itu *Family_Size* juga memiliki distribusi yang *right skewed*, dengan mayoritas ukuran keluarga berada di angka 2 hingga 4, dan juga tetap terdapat beberapa outlier di atas nilai 7. Temuan ini menandakan perlunya penanganan terhadap skala dan distribusi fitur agar hasil clustering lebih optimal.

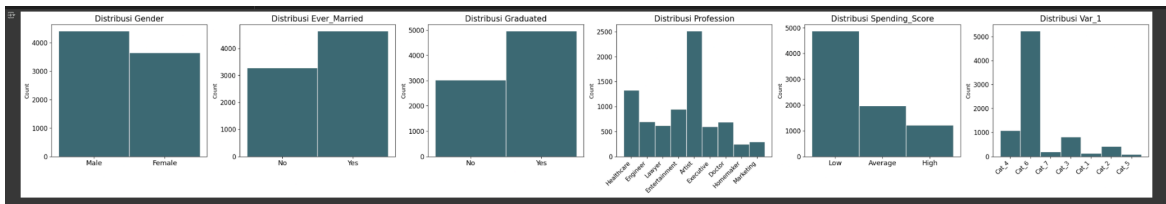
Sebagai langkah *preprocessing*, pertama kita akan melakukan penghapusan *outlier* dengan menggunakan metode IQR. Setelahnya kami akan menerapkan *StandardScaler* pada fitur numerikal diatas untuk menyamakan skala dan mencegah fitur dominan mengganggu hasil *clustering*. Namun sebelum itu, akan diterapkan log transformation terlebih dahulu menggunakan fungsi *log1p* untuk mengatasi kemiringan distribusi (*skewness*) sebelum dilakukan scaling. Ini dilakukan agar semua fitur lebih mendekati distribusi normal, yang sesuai dengan asumsi dasar *StandardScaler*.

4.3.3.2. Fitur non-Numerik

a. Kode

```
plt.figure(figsize=(30, 5))
for i, col in enumerate(cat.columns):
    plt.subplot(1, 6, i+1)
    sns.histplot(data_train[col], bins=10, color=hist_color, edgecolor='white',
kde=False)
    plt.title(f'Distribusi {col}', fontsize=15)
    plt.xlabel(None)
    plt.xticks(fontsize=13)
    plt.yticks(fontsize=12)
    if col == 'Profession' or col == 'Var_1':
        plt.xticks(rotation=45, ha='right', fontsize=11)
    plt.tight_layout()
plt.show()
```

b. Output



c. Penjelasan

Dari visualisasi distribusi kategori ini, kita dapat melihat beberapa hal penting. Distribusi antara gender cukup seimbang namun sedikit lebih banyak pria. Mayoritas pelanggan telah/pernah menikah dan lulus pendidikan tinggi, yang bisa menunjukkan segmen pelanggan yang sudah mapan secara sosial dan ekonomi. Profesi “Artist” mendominasi dengan jumlah tertinggi, sementara “Homemaker” memiliki jumlah yang relatif sedikit, hal ini dapat mempengaruhi hasil modeling karena distribusi yang tidak seimbang dapat membuat algoritma berat sebelah terhadap kelas mayoritas. Untuk *Spending Score*, kategori “Low” paling banyak, dan ini bisa menjadi sinyal adanya potensi segmentasi konsumen dengan pengeluaran rendah yang dominan.

Distribusi kolom Var_1 sangat timpang, dengan mayoritas nilai berada di kategori Cat 6. Ketidakseimbangan ini bisa menyebabkan hasil *clustering* terlalu terpengaruh oleh kategori tersebut. Karena kita tidak mengetahui makna dari Var_1 dan distribusinya sangat timpang, sebaiknya kolom ini tidak digunakan terlebih dahulu dalam proses clustering. Langkah

selanjutnya adalah melakukan *encoding* pada semua fitur kategori (selain ID dan *Segmentation* yang akan dibuang).

Untuk penanganan fitur kategorikal, pendekatan *encoding* yang akan digunakan adalah *One-Hot Encoding* untuk fitur kategorik *non-ordinal* seperti *Profession*, karena tidak memiliki urutan intrinsik antar kategorinya. Ini akan menghindari asumsi palsu tentang urutan yang bisa membingungkan model berbasis jarak. Sementara itu, untuk fitur *Spending_Score*, akan dilakukan *Ordinal Encoding* karena kategori *Low*, *Average*, dan *High* memiliki urutan alami yang dapat berkontribusi pada pembentukan *cluster*. Terakhir untuk Fitur-Fitur Biner akan kita lakukan *Label Encoding* agar tidak menambahkan dimensi dari data yang akan sangat berpengaruh bagi perhitungan jarak di algoritma clustering nantinya

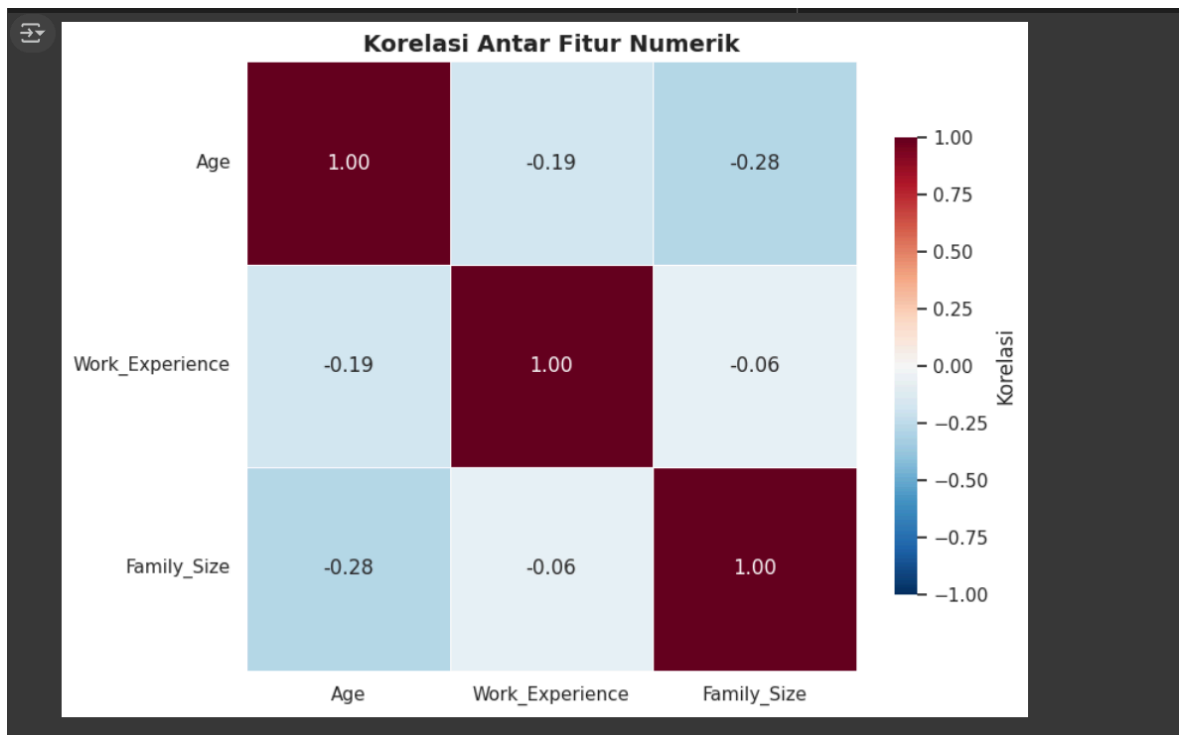
4.3.3.3. Heatmap Korelasi Fitur Numerik

a. Kode

```
plt.figure(figsize=(8, 6))
sns.set(style="whitegrid")
corr = num.corr()

sns.heatmap(corr, annot=True, fmt=".2f", cmap='RdBu_r', vmin=-1, vmax=1,
linewidths=0.5, square=True, cbar_kws={"shrink": 0.75, "label": "Korelasi"})

plt.title("Korelasi Antar Fitur Numerik", fontsize=14, fontweight='bold')
plt.xticks(rotation=0)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

b. Output**c. Penjelasan**

Berdasarkan heatmap korelasi yang ditampilkan, terlihat bahwa korelasi antar fitur numerik seperti *Age*, *Work_Experience*, dan *Family_Size* relatif rendah satu sama lain. Nilai korelasi tertinggi (selain diagonal utama) adalah -0.28 antara *Age* dan *Family_Size*, yang menunjukkan hubungan negatif lemah. Hal ini menunjukkan bahwa fitur-fitur ini cukup independen dan tidak redundan, sehingga semuanya layak dipertahankan dalam proses clustering karena masing-masing membawa informasi unik.

4.3.3.4. Missing Value Analysis**a. Kode**

```
data_train.isnull().sum()

data_missing = data_train.copy()
data_missing = data_missing.drop(columns=['Segmentation'])
missing_rows = data_missing[data_missing.isnull().any(axis=1)]

print("Jumlah baris yang memiliki missing value:", len(missing_rows))
missing_rows.head()
missing_rows = data_missing[data_missing['Ever_Married'].isnull()]
```



```
missing_rows.head()

missing_rows = data_missing[data_missing['Graduated'].isnull()]
missing_rows.head()


missing_rows = data_missing[data_missing['Profession'].isnull()]
missing_rows.head()

missing_rows = data_missing[data_missing['Work_Experience'].isnull()]
missing_rows.head()

missing_rows = data_missing[data_missing['Family_Size'].isnull()]
missing_rows.head()

missing_rows = data_missing[data_missing['Var_1'].isnull()]
missing_rows.head()

# Persentasi Data Mising Tiap Kolom
missing_pct = data_missing.isnull().mean().sort_values(ascending=False)
print(missing_pct)
```

b. Output

	0
ID	0
Gender	0
Ever_Married	140
Age	0
Graduated	78
Profession	124
Work_Experience	829
Spending_Score	0
Family_Size	335
Var_1	76
Segmentation	0
dtype:	int64

Jumlah baris yang memiliki missing value: 1403

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	3.0	Cat_4
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	6.0	Cat_6
12	461230	Female	No	19	No	Executive	0.0	Low	NaN	Cat_3
13	459573	Male	Yes	70	No	Lawyer	NaN	Low	1.0	Cat_6
24	461021	Female	NaN	58	No	Executive	1.0	Average	3.0	Cat_3

Jumlah baris yang memiliki missing value: 1403

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	3.0	Cat_4
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	6.0	Cat_6
12	461230	Female	No	19	No	Executive	0.0	Low	NaN	Cat_3
13	459573	Male	Yes	70	No	Lawyer	NaN	Low	1.0	Cat_6
24	461021	Female	NaN	58	No	Executive	1.0	Average	3.0	Cat_3

Jumlah baris yang memiliki missing value: 1403

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
24	461021	Female	NaN	58	No	Executive	1.0	Average	3.0	Cat_3
108	466466	Female	NaN	19	No	Healthcare	6.0	Low	5.0	Cat_3
201	466065	Male	NaN	19	No	Healthcare	9.0	Low	3.0	Cat_3
213	460516	Female	NaN	85	No	Lawyer	0.0	High	1.0	Cat_3
272	464841	Male	NaN	19	No	Entertainment	0.0	High	3.0	Cat_4

Jumlah baris yang memiliki missing value: 1403

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
79	466567	Female	No	49	Yes	NaN	1.0	Low	1.0	Cat_6
118	465827	Male	No	27	No	NaN	2.0	Low	7.0	Cat_4
219	465837	Male	No	62	Yes	NaN	0.0	Low	1.0	Cat_6
237	467252	Female	No	33	Yes	NaN	0.0	Low	4.0	NaN
437	461410	Male	Yes	79	No	NaN	0.0	Average	2.0	NaN

Jumlah baris yang memiliki missing value: 1403

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
57	462267	Male	No	32	NaN	Doctor	8.0	Low	2.0	Cat_6
220	464613	Female	No	35	NaN	Artist	0.0	Low	3.0	Cat_6
290	465058	Female	No	43	NaN	Entertainment	NaN	Low	1.0	Cat_6
431	462548	Male	No	18	NaN	Executive	NaN	Low	5.0	Cat_4
510	460685	Male	No	51	NaN	Artist	6.0	Low	4.0	Cat_4

Jumlah baris yang memiliki missing value: 1403

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	3.0	Cat_4
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	6.0	Cat_6
13	459573	Male	Yes	70	No	Lawyer	NaN	Low	1.0	Cat_6
39	467442	Male	Yes	56	Yes	Artist	NaN	Average	2.0	Cat_6
45	463156	Female	Yes	79	No	Lawyer	NaN	High	2.0	Cat_6

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
43	466006	Female	Yes	49	Yes	Artist	1.0	Low	2.0	NaN
163	463595	Female	No	32	No	Healthcare	1.0	Low	5.0	NaN
186	459576	Female	Yes	85	No	Lawyer	NaN	Low	NaN	NaN
231	463467	Female	No	23	No	Healthcare	0.0	Low	4.0	NaN
233	461282	Male	No	21	No	Healthcare	0.0	Low	4.0	NaN

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1
12	461230	Female	No	19	No	Executive	0.0	Low	NaN	Cat_3
33	467010	Male	No	26	No	Homemaker	9.0	Low	NaN	Cat_6
59	460881	Male	Yes	72	Yes	Lawyer	1.0	Low	NaN	Cat_4
112	467758	Female	Yes	50	Yes	Doctor	1.0	Low	NaN	Cat_6
126	466295	Female	Yes	42	No	Engineer	0.0	Low	NaN	Cat_6

Work_Experience

0.102752

Family_Size

0.041522

Ever_Married

0.017353

Profession

0.015369

Graduated

0.009668

Var_1

0.009420

Gender

0.000000

ID

0.000000

Age

0.000000

Spending_Score

0.000000

dtype: float64

c. Penjelasan

Dari hasil penelusuran tidak ada pola untuk dari sebuah data yang menunjukkan bahwa sebuah *null* value berarti sesuatu, dengan kata lain *null* value pure dari kelalaian pengisian atau terjadinya data yang korup atau sebagainya (*Missing Completely at Random*) karena *missing value* di setiap kolom tidak ada yang lebih dari 20% maka akan kami lakukan imputasi di bagian *preprocessing* nantinya daripada *drop* kolom tersebut,

Berdasarkan analisis distribusi data dan persentase *missing values*, penanganan *missing values* akan dilakukan menggunakan metode sederhana yang sesuai dengan karakteristik setiap kolom. Untuk kolom biner yaitu *Ever_Married* (140 *missing values*, 1.7%) dan *Graduated* (78 *missing values*, 0.9%) yang merupakan variabel biner, akan digunakan mode *imputation* dengan mengisi nilai yang hilang menggunakan kategori yang paling sering muncul dalam dataset. Dari distribusi yang terlihat, kategori "Yes" lebih dominan pada kedua kolom ini.

Untuk kolom numerikal yaitu *Work_Experience* (829 *missing values*, 10.2%) yang memiliki persentase *missing values* tertinggi, akan digunakan median *imputation* karena distribusi data yang sangat *right-skewed* dengan sebagian besar nilai berkisar antara 0-5 tahun. Median lebih *robust* terhadap *outlier* dibandingkan *mean* pada distribusi yang tidak normal seperti ini. Kolom *Family_Size* (335 *missing values*, 4.1%) yang juga menunjukkan distribusi

right-skewed dengan mayoritas keluarga berukuran 2-4 orang akan ditangani dengan median *imputation* untuk alasan yang sama, yaitu ketahanan terhadap nilai ekstrem dan kemampuan menangkap tendensi sentral data yang lebih representatif.

Kolom *Profession* (124 *missing values*, 1.5%) yang merupakan variabel kategorikal dengan 9 kategori berbeda akan ditangani menggunakan mode *imputation*, dimana *missing values* akan diisi dengan profesi yang paling sering muncul dalam dataset. Pendekatan ini dipilih karena kesederhanaannya dalam implementasi dan efektivitasnya untuk dataset dengan persentase *missing values* yang relatif rendah, sehingga tidak akan mengubah distribusi asli data secara signifikan.

4.3.3.5. Data Type Analysis

Dalam section understanding data sebelumnya, kami awalnya menyadari bahwa beberapa kolom seperti *Work Experience* dan *Family Size* memiliki tipe data *float*, namun seluruh nilainya merupakan bilangan bulat tanpa angka desimal. Hal ini kami temukan melalui eksplorasi menggunakan `.describe()`, yang menunjukkan bahwa meskipun bertipe *float*, kedua fitur tersebut secara semantik bersifat diskret. Meskipun kami memahami bahwa seluruh fitur nantinya akan dikonversi kembali menjadi *float* setelah proses *scaling*, kami memutuskan untuk tetap melakukan *casting* ke tipe *int* sebagai bentuk eksplisitasi makna data yang sebenarnya. Casting ini membantu menjaga kejelasan semantik, memudahkan validasi data (misalnya mencegah masuknya nilai non-logis seperti 2.5 anggota keluarga), dan membuat pipeline *preprocessing* kami menjadi lebih rapi dan terstruktur.

a. Kode

```
data_train[num.columns].describe()

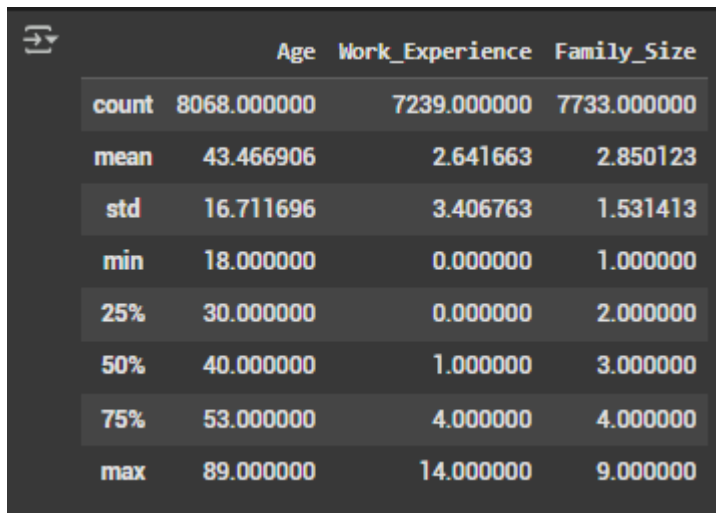
# Cek apakah 3 kolom ini adalah bilangan bulat
work_experience_decimal_check = (data_train['Work_Experience'].dropna() % 1 == 0).all()
family_size_decimal_check = (data_train['Family_Size'].dropna() % 1 == 0).all()

print(f"Apakah semua angka di kolom Work_Experience (tanpa NaN) di belakang koma adalah 0? {work_experience_decimal_check}")
print(f"Apakah semua angka di kolom Family_Size (tanpa NaN) di belakang koma adalah 0? {family_size_decimal_check}")

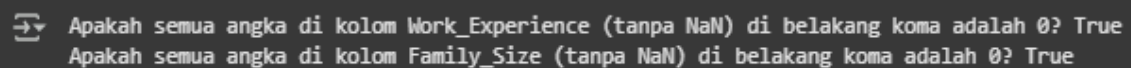
# Menampilkan angka tidak yang bukan bilangan bulat jika ada
if not work_experience_decimal_check:
    print("\nBaris di kolom Work_Experience dengan angka di belakang koma selain 0:")
```

```
print(data_train[data_train['Work_Experience'].dropna() % 1 != 0])
if not family_size_decimal_check:
    print("\nBaris di kolom Family_Size dengan angka di belakang koma selain 0:")
    print(data_train[data_train['Family_Size'].dropna() % 1 != 0])
```

b. Output



	Age	Work_Experience	Family_Size
count	8068.000000	7239.000000	7733.000000
mean	43.466906	2.641663	2.850123
std	16.711696	3.406763	1.531413
min	18.000000	0.000000	1.000000
25%	30.000000	0.000000	2.000000
50%	40.000000	1.000000	3.000000
75%	53.000000	4.000000	4.000000
max	89.000000	14.000000	9.000000



```
Apakah semua angka di kolom Work_Experience (tanpa NaN) di belakang koma adalah 0? True
Apakah semua angka di kolom Family_Size (tanpa NaN) di belakang koma adalah 0? True
```

4.4. Preprocessing Data

4.4.1. Persiapan data untuk *Preprocessing*

a. Kode

```
data = data_train.copy()

num_cols = ["Age", "Work_Experience", "Family_Size"]
binary_cols = ["Gender", "Ever_Married", "Graduated"]
cat_cols = ["Profession", "Spending_Score"]

data.info()
```

b. Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8068 entries, 0 to 8067
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   ID                    8068 non-null   int64
1   Gender                8068 non-null   object
2   Ever_Married          7928 non-null   object
3   Age                   8068 non-null   int64
4   Graduated             7990 non-null   object
5   Profession             7944 non-null   object
6   Work_Experience        7239 non-null   float64
7   Spending_Score         8068 non-null   object
8   Family_Size            7733 non-null   float64
9   Var_1                 7992 non-null   object
10  Segmentation           8068 non-null   object
dtypes: float64(2), int64(2), object(7)
memory usage: 693.5+ KB
```

4.4.2. Drop Kolom yang Tidak Relevan

a. Kode

```
data = data.drop(columns=['ID', 'Var_1'])
```

b. Penjelasan

Kode diatas akan menghapus kolom “ID” karena tidak relevan sebagai input dalam melakukan *clustering* dan kolom “Var_1” karena menyebabkan *curse of dimensionality* (ada 6 kategori yang akan di *one hot encoding*), persebaran data yang *skew* menyebabkan bias, dan juga tidak jelas apa yang direpresentasikan oleh *value* dalam kolom tersebut.

4.4.3. Hapus *Outlier* dengan Metode IQR

a. Kode

```
data.describe()

Q1 = data[num_cols].quantile(0.25)
Q3 = data[num_cols].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```

outliers = ((data[num_cols] < lower_bound) | (data[num_cols] > upper_bound))

data = data[~outliers.any(axis=1)]
print("Shape after removing outliers:", data.shape)

# Box Plot Setelah Pembersihan Outlier
plt.figure(figsize=(10, 8))
for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    sns.boxplot(y=data[col], color=box_color)
    plt.title(f'Boxplot {col}')
    plt.tight_layout()
plt.show()

data.describe()

```

b. Output

	Age	Work_Experience	Family_Size
count	8068.000000	7239.000000	7733.000000
mean	43.466906	2.641663	2.850123
std	16.711696	3.406763	1.531413
min	18.000000	0.000000	1.000000
25%	30.000000	0.000000	2.000000
50%	40.000000	1.000000	3.000000
75%	53.000000	4.000000	4.000000
max	89.000000	14.000000	9.000000

```

Q1 = data[num_cols].quantile(0.25)
Q3 = data[num_cols].quantile(0.75)
IQR = Q3 - Q1

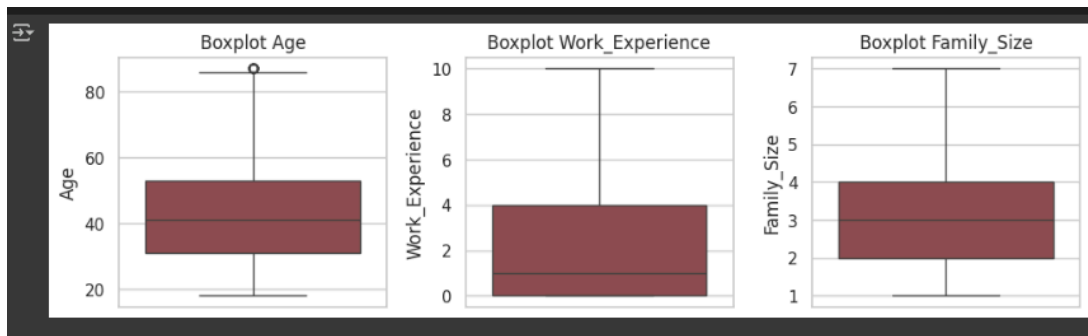
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = ((data[num_cols] < lower_bound) | (data[num_cols] > upper_bound))

data = data[~outliers.any(axis=1)]
print("Shape after removing outliers:", data.shape)

Shape after removing outliers: (7720, 9)

```

	Age	Work_Experience	Family_Size
count	7720.000000	6911.000000	7399.000000
mean	43.331606	2.392128	2.798757
std	16.306127	3.047557	1.407572
min	18.000000	0.000000	1.000000
25%	31.000000	0.000000	2.000000
50%	41.000000	1.000000	3.000000
75%	53.000000	4.000000	4.000000
max	87.000000	10.000000	7.000000

c. Penjelasan

Banyaknya data turun dari 8068 menjadi 7720 (berkurang ~4%) setelah dilakukan pembersihan outlier menggunakan metode IQR

4.4.4. Imputasi *Missing Value*

a. Kode

```
# Numerik: isi dengan median
for col in num_cols:
    data[col] = data[col].fillna(data[col].median())

# Kategorikal: isi dengan modus
for col in cat_cols:
    data[col] = data[col].fillna(data[col].mode()[0])

# Biner: isi dengan modus
for col in binary_cols:
```



```
data[col] = data[col].fillna(data[col].mode()[0])
```

4.4.5. Casting Tipe Data

a. Kode

```
data['Family_Size'] = data['Family_Size'].astype(int)
data['Work_Experience'] = data['Work_Experience'].astype(int)

data.info()
```

b. Output

```
<class 'pandas.core.frame.DataFrame'>
Index: 7720 entries, 0 to 8067
Data columns (total 9 columns):
#   Column             Non-Null Count  Dtype
---  ---
0   Gender              7720 non-null   object
1   Ever_Married        7720 non-null   object
2   Age                 7720 non-null   int64
3   Graduated           7720 non-null   object
4   Profession           7720 non-null   object
5   Work_Experience      7720 non-null   int64
6   Spending_Score      7720 non-null   object
7   Family_Size          7720 non-null   int64
8   Segmentation        7720 non-null   object
dtypes: int64(3), object(6)
memory usage: 603.1+ KB
```

4.4.6. Encoding

4.4.6.1. Label Encoding untuk Kolom Biner dan Segmentation

a. Kode

```
le = LabelEncoder()
data['Gender'] = le.fit_transform(data['Gender']) # Female=0, Male=1
data['Ever_Married'] = le.fit_transform(data['Ever_Married']) # No=0, Yes=1
data['Graduated'] = le.fit_transform(data['Graduated']) # No=0, Yes=1
data['Segmentation'] = le.fit_transform(data['Segmentation']) # A=0, B=1, C=2, D=3

data.head()
```

b. Output

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Segmentation
0	1	0	22	0	Healthcare	1	Low	4	3
1	0	1	38	1	Engineer	1	Average	3	0
2	0	1	67	1	Engineer	1	Low	1	1
3	1	1	67	1	Lawyer	0	High	2	1
4	0	1	40	1	Entertainment	1	High	6	0

4.4.6.2. Ordinal Encoding untuk Spending Score**a. Kode**

```

spending_score_map = {'Low': 0, 'Average': 1, 'High': 2}
data['Spending_Score'] = data['Spending_Score'].map(spending_score_map)

data.head()

```

b. Output

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Segmentation
0	1	0	22	0	Healthcare	1	0	4	3
1	0	1	38	1	Engineer	1	1	3	0
2	0	1	67	1	Engineer	1	0	1	1
3	1	1	67	1	Lawyer	0	2	2	1
4	0	1	40	1	Entertainment	1	2	6	0

4.4.6.3. One-Hot Encoding untuk Fitur Profession**a. Kode**

```

data_encoded = pd.get_dummies(data, columns=["Profession"], drop_first=True)

data_encoded.head()

```

b. Output

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Segmentation	Profession_Doctor	Profession_Engineer	Profession_Entertainment	Profession_Executive	Profession_Healthcare	Profession_Homemaker	Profession_Lawyer	Profession_Marketing
0	1	0	22	0	1	0	4	3	False	False	False	False	True	False	False	False
1	0	1	38	1	1	1	3	0	False	True	False	False	False	False	False	False
2	0	1	67	1	1	0	1	1	False	True	False	False	False	False	False	False
3	1	1	67	1	0	2	2	1	False	False	False	False	False	False	True	False
4	0	1	40	1	1	2	6	0	False	False	True	False	False	False	False	False

4.4.7. Fix Skewness

a. Kode

```
for col in num_cols:
    data_encoded[col] = np.log1p(data_encoded[col])

hist_color = '#003844'
box_color = '#984447'

plt.figure(figsize=(10, 8))
for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    sns.histplot(data_encoded[col], bins=6, color=hist_color, edgecolor='white',
kde=False)
    plt.title(f'Distribusi {col}')
plt.tight_layout()
plt.show()
```

b. Output



4.4.8. Feature Scaling

a. Kode

```
scaler = StandardScaler()
scaled_cols = ["Age", "Work_Experience", "Family_Size", "Spending_Score"]
data_scaled = data_encoded.copy()
data_scaled[scaled_cols] = scaler.fit_transform(data_encoded[scaled_cols])

data_scaled.head()
```

b. Output

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Segmentation	Profession_Doctor	Profession_Engineer	Profession_Entertainment	Profession_Executive	Profession_Healthcare	Profession_Homemaker	Profession_Lawyer	Profession_Marketing
0	1	0	-1.586412	0	-0.202423	-0.738594	0.926293	3	False	False	False	False	True	False	False	False
1	0	1	-0.163103	1	-0.202423	0.617398	0.314899	0	False	True	False	False	False	False	False	False
2	0	1	1.335348	1	-0.202423	-0.738594	-1.584262	1	False	True	False	False	False	False	False	False
3	1	1	1.335348	1	-1.100537	1.973391	-0.473324	1	False	False	False	False	False	False	True	False
4	0	1	-0.028309	1	-0.202423	1.973391	1.848197	0	False	False	True	False	False	False	False	False

4.4.9. Ambil Subset Dataset**a. Kode**

```

proportions = data_scaled["Segmentation"].value_counts(normalize=True)
samples_per_class = (proportions * 1000).round().astype(int)

subset = pd.concat([
    data_scaled[data_scaled["Segmentation"] == cls].sample(n=n,
random_state=42)
    for cls, n in samples_per_class.items()])

data_kecil = subset.sample(frac=1, random_state=42).reset_index(drop=True) #
shuffle

data_kecil

data_kecil["Segmentation"].value_counts()

label = data_kecil["Segmentation"]

data = data_kecil.drop(columns=("Segmentation"))

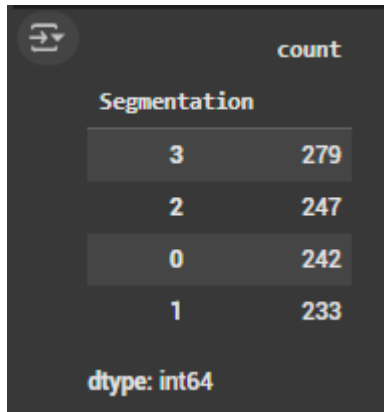
data.info()

```

b. Output

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Segmentation	Profession_Doctor	Profession_Engineer	Profession_Entertainment	Profession_Executive	Profession_Healthcare	Profession_Homemaker	Profession_Lawyer	Profession_Marketing
0	1	1	2.030280	1	-0.202423	1.973391	-0.473324	2	False	False	False	True	False	False	False	False
1	0	1	1.254885	1	-0.202423	-0.738594	-1.584262	1	False	True	False	False	False	False	False	False
2	1	0	0.339806	0	-1.100537	0.617398	0.314899	0	False	False	True	False	False	False	False	False
3	0	0	-1.056216	1	-1.100537	-0.738594	0.926293	1	False	False	False	True	False	False	False	False
4	1	0	-0.094863	1	-0.202423	-0.738594	-1.584262	2	False	False	False	False	False	False	False	False
...
996	1	0	-0.613367	0	-1.100537	-0.738594	0.926293	3	False	False	False	False	False	False	False	False
997	0	0	-1.056216	0	-0.202423	-0.738594	0.314899	3	False	False	False	False	False	False	False	False
998	0	0	-0.163103	1	-0.202423	-0.738594	-1.584262	1	False	False	False	False	False	False	False	False
999	0	1	0.714014	1	-0.202423	-0.738594	-0.473324	2	False	False	False	False	False	False	False	False
1000	1	0	-1.255960	1	-0.202423	-0.738594	0.926293	3	False	False	False	False	True	False	False	False

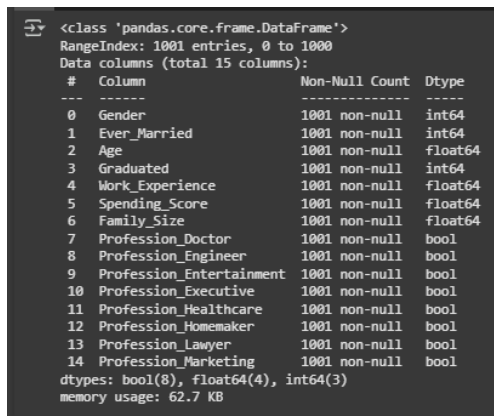
1001 rows x 16 columns



A Jupyter Notebook cell displaying a count of values for the 'Segmentation' variable. The output is a table with two columns: the segmentation value and its count. The dtype is int64.

Segmentation	count
3	279
2	247
0	242
1	233

dtype: int64



A Jupyter Notebook cell displaying the output of the `data_train[num.columns].describe()` command. It shows the data type, range, and non-null count for each of the 15 columns in the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001 entries, 0 to 1000
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Gender                1001 non-null  int64
1   Ever_Married          1001 non-null  int64
2   Age                   1001 non-null  float64
3   Graduated             1001 non-null  int64
4   Work_Experience       1001 non-null  float64
5   Spending_Score       1001 non-null  float64
6   Family_Size           1001 non-null  float64
7   Profession_Doctor     1001 non-null  bool
8   Profession_Engineer   1001 non-null  bool
9   Profession_Entertainment 1001 non-null  bool
10  Profession_Executive  1001 non-null  bool
11  Profession_Healthcare 1001 non-null  bool
12  Profession_Homemaker   1001 non-null  bool
13  Profession_Lawyer      1001 non-null  bool
14  Profession_Marketing   1001 non-null  bool
dtypes: bool(8), float64(4), int64(3)
memory usage: 62.7 KB
```

c. Penjelasan

Sayangnya karena besarnya biaya komputasi untuk modeling *hierarchical manual* $O(n^3)$, kami harus mengurangi jumlah data, disini akan kami buat subset data yang *stratified* terhadap label *Segmentation* agar persebaran kelas tetap merata dan memiliki karakteristik yang sama dengan data awalnya, kita akan ambil subset berjumlah 1000 baris

4.4.10. Persiapan data untuk *Preprocessing*

a. Kode

```
data_train[num.columns].describe()

# Cek apakah 3 kolom ini adalah bilangan bulat
work_experience_decimal_check = (data_train['Work_Experience'].dropna() % 1 == 0).all()
family_size_decimal_check = (data_train['Family_Size'].dropna() % 1 == 0).all()

print(f'Apakah semua angka di kolom Work_Experience (tanpa NaN) di belakang koma adalah 0? {work_experience_decimal_check}')
print(f'Apakah semua angka di kolom Family_Size (tanpa NaN) di belakang koma adalah 0? {family_size_decimal_check}')
```

```
# Menampilkan angka tidak yang bukan bilangan bulat jika ada
if not work_experience_decimal_check:
    print("\nBaris di kolom Work_Experience dengan angka di belakang koma selain 0:")
    print(data_train[data_train['Work_Experience'].dropna() % 1 != 0])
if not family_size_decimal_check:
    print("\nBaris di kolom Family_Size dengan angka di belakang koma selain 0:")
    print(data_train[data_train['Family_Size'].dropna() % 1 != 0])
```

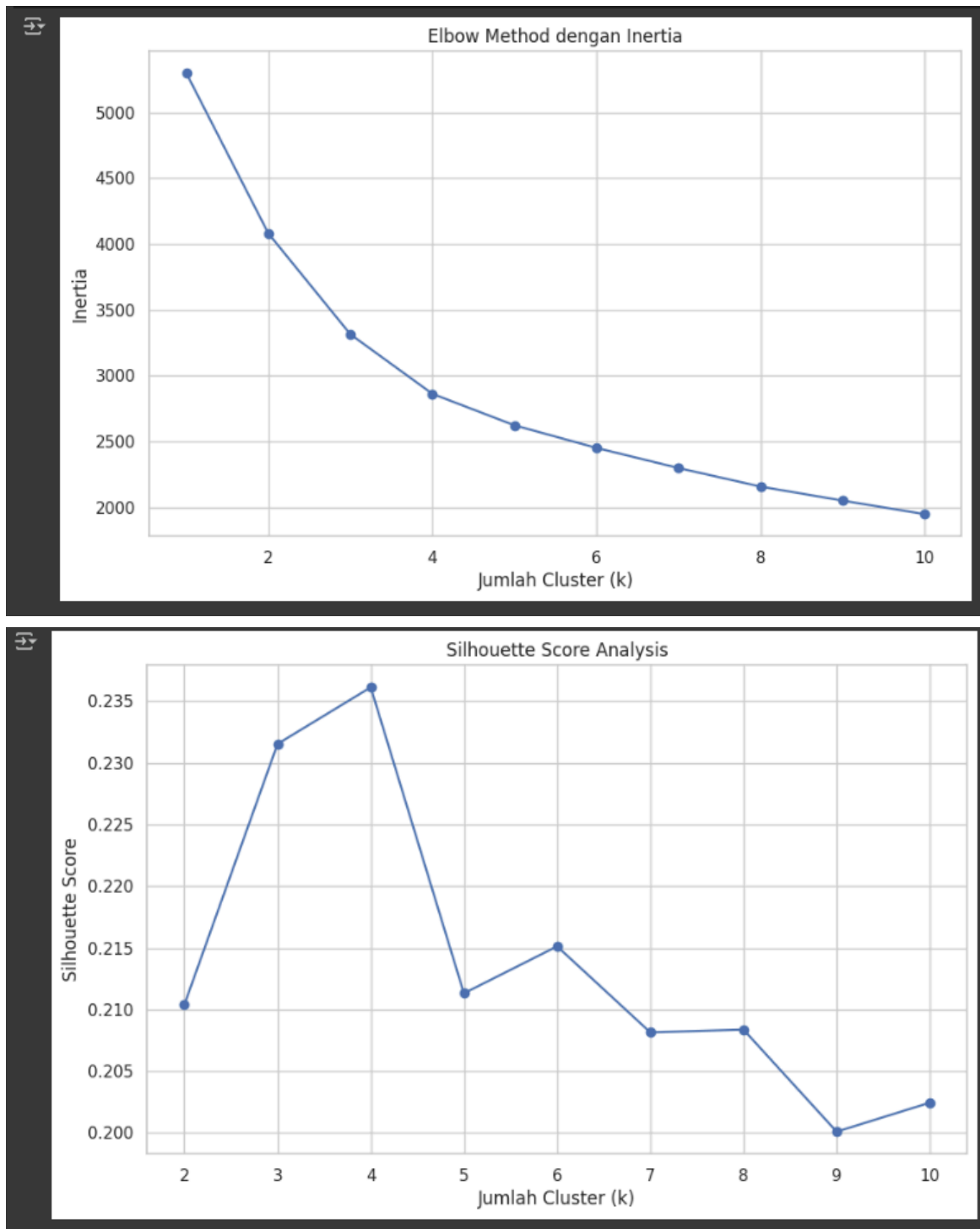
4.5. Modelling

4.5.1. Menentukan Jumlah Cluster Optimal dengan Elbow Method

a. Kode

```
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)
    kmeans.fit(data)
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method dengan Inertia')
plt.xlabel('Jumlah Cluster (k)')
plt.ylabel('Inertia')
plt.show()

silhouette_scores = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)
    kmeans.fit(data)
    score = silhouette_score(data, kmeans.labels_)
    silhouette_scores.append(score)
plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.title('Silhouette Score Analysis')
plt.xlabel('Jumlah Cluster (k)')
plt.ylabel('Silhouette Score')
plt.show()
```

b. Output**c. Penjelasan**

Bisa dilihat *elbow* terdapat di $k = 4$ sesuai dengan label segmentasi yang diberikan oleh team *marketing* sebelumnya, setelah $k=4$ *inertia* mulai konvergen. Sesuai dengan *elbow method* $k=4$ memiliki nilai tertinggi untuk *silhouette* nya.

4.5.2. Perhitungan Jarak

a. Kode

```
def euclidean(data1, data2):  
    jarak=np.square(data1-data2)  
    jarak=np.sum(jarak)  
    return np.sqrt(jarak)  
  
def manhattan(data1, data2):  
    jarak=np.abs(data1-data2)  
    jarak=np.sum(jarak)  
    return jarak
```

4.5.3. K-Means

4.5.3.1. K-Means Library

a. Kode

```
kmeans_library = KMeans(n_clusters=4, random_state=42, n_init=10)  
kmeans_library_labels = kmeans_library.fit_predict(data)
```

4.5.3.2. K-Means Manual

a. Kode

```
def kmeans_manual(data_input, k, iter_max=200, random_state=None):  
    if random_state is not None:  
        np.random.seed(random_state)  
  
    n_data = data_input.shape[0]  
    n_fitur = data_input.shape[1]  
    iterasi = 1  
  
    cluster = np.random.randint(0, k, size=n_data) # memberi cluster acak sebagai awalan  
    centroid = np.zeros((k, n_fitur))  
    cluster_tidak_berubah = False  
  
    while not cluster_tidak_berubah and iterasi < iter_max:
```



```

    jarak = np.zeros((n_data, k))

    for i in range(k):
        anggota = data_input.iloc[np.where(cluster == i)[0]] # ambil data anggota
        klaster ke-i
        if not anggota.empty:
            centroid[i, :] = anggota.mean().values # (1) Hitung centroid sebagai
            rata-rata anggota
        else:
            centroid[i, :] = data_input.sample(n=1, random_state=random_state).values
            # Jika tidak ada anggota, sampling satu data acak sebagai centroid

    for i in range(n_data): # Hitung jarak dari setiap data ke setiap centroid
        for j in range(k):
            jarak[i][j] = euclidean(data_input.iloc[i], centroid[j]) # (2) Hitung jarak
            Euclidean

    cluster_baru = [np.argmin(jarak[i]) for i in range(n_data)] # Tentukan klaster
    baru berdasarkan centroid terdekat
    cluster_tidak_berubah = np.array_equal(cluster_baru, cluster) # Periksa apakah
    klaster sudah tidak berubah (konvergen)
    cluster = np.array(copy.deepcopy(cluster_baru)) # Perbarui klaster untuk iterasi
    berikutnya
    iterasi += 1 # naikkan iterasi

    return cluster, centroid, jarak

kmeans_manual_labels, centroids_manual, jarak_manual = kmeans_manual(data, 4,
random_state=42)

kmeans_manual_labels

centroids_manual

jarak_manual

```

b. Output

```

[ ] kmeans_manual_labels
→ array([2, 3, 2, ..., 3, 3, 1])

```

```
array([[ 0.51075269,  0.47849462, -0.30579473,  0.72580645,  1.56915604,
        -0.39595112, -0.25725076,  0.08064516,  0.06451613,  0.16129032,
         0.02688172,  0.15053763,  0.06451613,  0.01612903,  0.04301075],
       [ 0.55909091,  0.16363636, -1.10130796,  0.28181818, -0.45785423,
        -0.68312199,  0.79351207,  0.14545455,  0.05909091,  0.08181818,
         0.02727273,  0.5       ,  0.01818182,  0.00454545,  0.05       ],
       [ 0.60968661,  1.       ,  0.65178432,  0.70940171, -0.34045909,
        1.20847188,  0.1898177 ,  0.05413105,  0.08262108,  0.1025641 ,
        0.17948718,  0.01139601,  0.00854701,  0.13960114,  0.02564103],
       [ 0.53688525,  0.57786885,  0.48853207,  0.69672131, -0.52695235,
        -0.73303706, -0.94696175,  0.09836066,  0.08606557,  0.13934426,
         0.02459016,  0.06967213,  0.04098361,  0.1557377 ,  0.02868852]])
```

```
array([[3.98187036, 4.6015327 , 1.97352967, 3.39640573],
       [2.9996502 , 3.73743328, 2.94702814, 1.59806173],
       [3.26805356, 2.5504706 , 1.60105268, 2.33135234],
       ...,
       [2.38087421, 2.7825249 , 3.02498387, 1.3111068 ],
       [2.2398548 , 2.60411205, 2.18695131, 1.00382173],
       [2.62073459, 1.06384729, 3.20606381, 2.8683516 ]])
```

c. Penjelasan

Fungsi `kmeans_manual` merupakan implementasi algoritma *K-Means* yang mengelompokkan data menjadi *k* kluster berdasarkan jarak *Euclidean*. Jika parameter `random_state` diberikan, maka fungsi menetapkan *seed* untuk memastikan hasil acak yang konsisten. Setiap data diberi label kluster secara acak, dan *centroid* diinisialisasi sebagai array kosong.

Proses inti berlangsung dalam *loop* yang berjalan hingga kluster tidak berubah lagi atau jumlah iterasi mencapai `iter_max`. Pada setiap iterasi, *centroid* dihitung ulang sebagai rata-rata dari data yang termasuk dalam masing-masing kluster. Jika ada kluster kosong, *centroid*-nya diisi dengan titik data acak untuk menghindari *error*.

Selanjutnya, fungsi menghitung jarak antara setiap titik data dan seluruh *centroid* menggunakan fungsi *euclidean*, yang mengembalikan akar kuadrat dari jumlah selisih kuadrat antar fitur. Berdasarkan matriks jarak ini, setiap data diberi kluster baru dengan memilih *centroid* terdekat. Iterasi berlanjut jika terjadi perubahan kluster, dan berhenti jika tidak ada perubahan. Fungsi mengembalikan tiga hal yaitu, label kluster akhir (*cluster*), posisi *centroid*, dan matriks jarak (*jarak*) antara setiap titik data dan *centroid*. Fungsi ini dapat digunakan untuk memahami cara kerja *K-Means* tanpa menggunakan pustaka eksternal seperti *scikit-learn*.

4.5.4. Hierarchical Clustering

4.5.4.1. Hierarchical Library

a. Kode

```
from scipy.cluster.hierarchy import linkage, dendrogram

# Single Linkage Euclidean
linked_single_eu = linkage(data, method='single', metric='euclidean')

# Single Linkage Manhattan
linked_single_mh = linkage(data, method='single', metric='cityblock')

# Complete Linkage Euclidean
linked_complete_eu = linkage(data, method='complete', metric='euclidean')

# Complete Linkage Manhattan
linked_complete_mh = linkage(data, method='complete', metric='cityblock')

from scipy.cluster.hierarchy import fcluster

# Single Linkage Euclidean
single_eu_labels = fcluster(linked_single_eu, 4, criterion='maxclust')

# Single Linkage Manhattan
single_mh_labels = fcluster(linked_single_mh, 4, criterion='maxclust')

# Complete Linkage Euclidean
complete_eu_labels = fcluster(linked_complete_eu, 4, criterion='maxclust')

# Complete Linkage Manhattan
complete_mh_labels = fcluster(linked_complete_mh, 4, criterion='maxclust')
```

b. Penjelasan

`linked_single_eu = linkage(data, method='single', metric='euclidean')` → Mengelompokkan data dengan *single linkage*, yaitu menggabungkan dua klaster berdasarkan jarak terpendek antar titik dari kedua klaster, menggunakan jarak *Euclidean* sebagai pengukurnya.

`linked_single_mh = linkage(data, method='single', metric='cityblock')` → Sama seperti sebelumnya menggunakan *single linkage*, tapi jarak antar titik diukur menggunakan jarak *Manhattan (cityblock)*, yaitu jumlah selisih absolut antar dimensi.

`linked_complete_eu = linkage(data, method='complete', metric='euclidean') → Mengelompokkan data dengan complete linkage, yaitu menggabungkan dua klaster berdasarkan jarak terjauh antar titik dari kedua klaster, dengan jarak Euclidean.`

`linked_complete_mh = linkage(data, method='complete', metric='cityblock') → Sama seperti sebelumnya menggunakan complete linkage, tapi menggunakan jarak Manhattan untuk menghitung jarak antar titik.`

Output dari kode ini adalah struktur data yang berisi informasi tentang bagaimana data akan digabungkan secara bertahap menjadi klaster, yaitu hasil proses penggabungan klaster secara hierarkis. *Dendrogram* bisa dibuat melalui data ini.

`single_eu_labels = fcluster(linked_single_eu, 4, criterion='maxclust') → Menghasilkan label klaster dengan memotong dendrogram hasil single linkage menggunakan jarak Euclidean sehingga terbentuk 4 klaster.`

`single_mh_labels = fcluster(linked_single_mh, 4, criterion='maxclust') → Menghasilkan label klaster dengan memotong dendrogram hasil single linkage menggunakan jarak Manhattan untuk membentuk 4 klaster.`

`complete_eu_labels = fcluster(linked_complete_eu, 4, criterion='maxclust') → Menghasilkan label klaster dari dendrogram hasil complete linkage dengan jarak Euclidean, dipotong untuk membentuk 4 klaster.`

`complete_mh_labels = fcluster(linked_complete_mh, 4, criterion='maxclust') → Menghasilkan label klaster dari dendrogram hasil complete linkage dengan jarak Manhattan, dipotong untuk membentuk 4 klaster.`

Linkage membuat model cluster hirarkis lengkap berupa *dendrogram* yang menggambarkan penggabungan data secara bertahap tanpa menentukan jumlah klaster akhir, sedangkan *fcluster* menggunakan hasil *dendrogram* tersebut untuk memotong dan membagi data menjadi klaster datar sesuai jumlah klaster yang ditentukan (dalam kasus ini 4 klaster). Jadi, *linkage* membangun struktur klaster secara menyeluruh, dan *fcluster* memilih pembagian akhir berdasarkan kriteria yang diinginkan tanpa membuat model baru.

4.5.4.2. Hierarchical Manual

a. Kode

```
def buat_matrix_jarak(data, metric='euclidean'):
    data = np.array(data)
    n_samples = data.shape[0]
    distance_matrix = np.zeros((n_samples, n_samples))

    # Pilih fungsi jarak
    if metric == 'euclidean':
        distance_func = euclidean
    elif metric == 'manhattan':
```

```
distance_func = manhattan
else:
    raise ValueError("Metric harus 'euclidean' atau 'manhattan'")

# Hitung jarak untuk setiap pasangan titik
for i in range(n_samples):
    for j in range(i + 1, n_samples):
        dist = distance_func(data[i], data[j])
        distance_matrix[i, j] = dist
        distance_matrix[j, i] = dist # Matriks simetris

return distance_matrix

def jarak_single_linkage(cluster1_indices, cluster2_indices, distance_matrix):
    min_distance = float('inf')

    # Cari jarak minimum antar semua pasangan titik
    for i in cluster1_indices:
        for j in cluster2_indices:
            distance = distance_matrix[i, j]
            if distance < min_distance:
                min_distance = distance

    return min_distance

def jarak_complete_linkage(cluster1_indices, cluster2_indices, distance_matrix):
    max_distance = 0

    # Cari jarak maksimum antar semua pasangan titik
    for i in cluster1_indices:
        for j in cluster2_indices:
            distance = distance_matrix[i, j]
            if distance > max_distance:
                max_distance = distance

    return max_distance

def hitung_jarak_cluster(cluster1_indices, cluster2_indices, distance_matrix,
linkage='single'):
    if linkage == 'single':
        return jarak_single_linkage(cluster1_indices, cluster2_indices, distance_matrix)
```

```
elif linkage == 'complete':
    return jarak_complete_linkage(cluster1_indices, cluster2_indices,
distance_matrix)
else:
    raise ValueError("Linkage harus 'single' atau 'complete'")

def cari_cluster_terdekat(clusters, distance_matrix, linkage='single'):
    min_distance = float('inf')
    closest_pair = (-1, -1)

    # Bandingkan semua pasangan cluster
    for i in range(len(clusters)):
        for j in range(i + 1, len(clusters)):
            distance = hitung_jarak_cluster(
                clusters[i], clusters[j], distance_matrix, linkage
            )

            if distance < min_distance:
                min_distance = distance
                closest_pair = (i, j)

    return closest_pair[0], closest_pair[1], min_distance

def agglomerative_clustering(data, metric='euclidean', linkage='single'):
    data = np.array(data)
    n_samples = data.shape[0]

    # Langkah 1: Buat matriks jarak antar titik data
    distance_matrix = buat_matrix_jarak(data, metric)

    # Langkah 2: Inisialisasi - setiap titik dianggap sebagai cluster sendiri
    # Sekarang clusters akan menyimpan list dari cluster, dan kita juga perlu
    # menyimpan ID permanen untuk setiap cluster (original index atau index merge)
    clusters = [[i] for i in range(n_samples)]
    # Initial IDs are the indices of the data points
    current_cluster_ids = list(range(n_samples))

    # List untuk menyimpan linkage matrix
    linkage_matrix = []

    # Langkah 3: Proses penggabungan cluster hingga tersisa satu cluster
```

```
# Melakukan n_samples - 1 penggabungan
for _ in range(n_samples - 1):

    # Cari dua cluster terdekat berdasarkan linkage dan metrik jarak
    # cari_cluster_terdekat sekarang bekerja dengan list of lists (clusters)
    # dan mengembalikan index di list 'clusters'
    i, j, min_dist = cari_cluster_terdekat(clusters, distance_matrix, linkage)

    # Ambil ID asli dari cluster yang akan digabung
    id1 = current_cluster_ids[i]
    id2 = current_cluster_ids[j]

    # Gabungkan kedua cluster (list of point indices)
    new_cluster_points = clusters[i] + clusters[j]

    # Hapus cluster yang sudah digabung dari daftar dan ID mereka
    # Hapus index j dulu agar index i tidak berubah
    del clusters[j]
    del clusters[i]
    del current_cluster_ids[j]
    del current_cluster_ids[i]

    # Tambahkan cluster hasil gabungan ke daftar cluster baru
    clusters.append(new_cluster_points)
    # ID untuk cluster baru adalah n_samples + (jumlah merge yang sudah
    dilakukan)
    new_cluster_id = n_samples + len(linkage_matrix)
    current_cluster_ids.append(new_cluster_id)

    # Tambahkan informasi penggabungan ke linkage matrix:
    # [id_cluster1, id_cluster2, jarak_penggabungan, jumlah_total_anggota_cluster]
    # Urutkan ID cluster agar sesuai format SciPy (ID lebih kecil di kiri)
    if id1 > id2:
        id1, id2 = id2, id1

    linkage_matrix.append([
        id1,
        id2,
        min_dist,
        len(new_cluster_points)
    ])
])
```

```
# Kembalikan linkage matrix dalam bentuk array numpy
return np.array(linkage_matrix)

# Hierarchical Clustering Manual Single Linkage Euclidean
matrix_single_eu = agglomerative_clustering(data, metric='euclidean',
linkage='single')

# Hierarchical Clustering Manual Single Linkage Manhattan
matrix_single_mh = agglomerative_clustering(data, metric='manhattan',
linkage='single')

# Hierarchical Clustering Manual Complete Linkage Euclidean
matrix_complete_eu = agglomerative_clustering(data, metric='euclidean',
linkage='complete')

# Hierarchical Clustering Manual Complete Linkage Manhattan
matrix_complete_mh = agglomerative_clustering(data, metric='manhattan',
linkage='complete')

from scipy.cluster.hierarchy import fcluster

# Single Linkage Euclidean
label_single_eu = fcluster(matrix_single_eu, 4, criterion='maxclust')

# Single Linkage Manhattan
label_single_mh = fcluster(matrix_single_mh, 4, criterion='maxclust')

# Complete Linkage Euclidean
label_complete_eu = fcluster(matrix_complete_eu, 4, criterion='maxclust')

# Complete Linkage Manhattan
label_complete_mh = fcluster(matrix_complete_mh, 4, criterion='maxclust')
```

b. Penjelasan

Fungsi `buat_matriks_jarak` digunakan untuk menghitung matriks jarak antar semua pasangan titik dalam dataset. Data terlebih dahulu diubah ke array NumPy dan disiapkan dalam bentuk matriks kosong berukuran $n \times n$. Kemudian, fungsi memilih metode pengukuran jarak yang akan digunakan, yaitu *Euclidean* (jarak lurus) atau *Manhattan* (jumlah selisih absolut antar dimensi). Setelah itu, fungsi menghitung jarak untuk setiap

pasangan titik unik (i, j), menyimpan hasilnya dalam matriks jarak, dan menjamin bahwa matriks tersebut bersifat simetris. Matriks inilah yang nantinya digunakan sebagai dasar dalam penggabungan klaster.

Fungsi `jarak_single_linkage` menghitung jarak antar dua klaster berdasarkan pendekatan *single linkage*, yaitu dengan mencari jarak terpendek antara satu titik dari klaster pertama dan satu titik dari klaster kedua. Fungsi ini mengevaluasi semua kombinasi titik dari dua klaster yang diberikan dan menyimpan nilai minimum dari semua jarak tersebut. Hasilnya adalah satu nilai jarak minimum yang merepresentasikan kedekatan antar dua klaster menurut *single linkage*.

Berbeda dengan *single linkage*, fungsi `jarak_complete_linkage` menerapkan metode *complete linkage* untuk menghitung jarak antar dua klaster. Ia mencari jarak maksimum dari semua pasangan titik yang mungkin antara dua klaster. Dengan kata lain, fungsi ini menganggap dua klaster “jauh” bila ada dua titik yang sangat berjauhan. Nilai jarak maksimum ini digunakan sebagai dasar penggabungan klaster dalam metode *complete linkage*.

Fungsi `hitung_jarak_cluster` bertugas memilih metode *linkage* yang akan digunakan untuk menghitung jarak antar dua klaster. Jika parameter `linkage` bernilai *single*, maka fungsi memanggil `jarak_single_linkage`, dan jika *complete*, maka dipanggil `jarak_complete_linkage`. Jika metode *linkage* yang diberikan tidak dikenali, maka fungsi akan menghasilkan *error*. Fungsi ini membuat proses pemanggilan jarak antar klaster menjadi fleksibel tergantung metode *linkage* yang dipilih.

Fungsi `cari_cluster_terdekat` mencari dua klaster yang paling dekat satu sama lain dari daftar klaster yang ada, berdasarkan metode *linkage* yang dipilih. Dengan membandingkan semua pasangan klaster (kombinasi dua klaster dari seluruh daftar), fungsi ini menghitung jarak antar klaster menggunakan `hitung_jarak_cluster`, lalu menyimpan pasangan dengan jarak terkecil. Hasil akhirnya adalah indeks dua klaster terdekat dan nilai jarak minimum di antara mereka. Fungsi ini menjadi inti dari proses pemilihan penggabungan klaster berikutnya dalam algoritma hierarkis.

Fungsi `agglomerative_clustering` merupakan implementasi lengkap dari algoritma *Agglomerative Hierarchical Clustering* secara manual. Langkah pertama adalah menghitung matriks jarak antar titik menggunakan metrik *Euclidean* atau *Manhattan*. Kemudian, setiap titik diasumsikan sebagai klaster tunggal dan disimpan dalam daftar. Selanjutnya, dilakukan proses penggabungan klaster secara bertahap: dua klaster terdekat dipilih dan digabung menjadi satu, lalu dicatat dalam *linkage matrix* dengan format `[id_klaster1, id_klaster2, jarak, jumlah_anggota]`. ID klaster juga dikelola agar sesuai dengan format standar SciPy. Proses ini diulang hingga semua titik tergabung menjadi satu klaster besar. Hasil akhir dari fungsi ini adalah *linkage matrix* dalam bentuk array NumPy, yang bisa digunakan untuk visualisasi dendrogram atau pemotongan klaster.

1. `matrix_single_eu = agglomerative_clustering(data, metric='euclidean', linkage='single')` → Mengelompokkan data secara manual dengan metode *hierarchical clustering* menggunakan *single linkage*, yaitu menggabungkan dua klaster berdasarkan jarak terpendek antar titik dari kedua klaster, dan menghitung jaraknya menggunakan metrik *Euclidean* (jarak lurus antar titik).

2. `matrix_single_mh = agglomerative_clustering(data, metric='manhattan', linkage='single')` → Sama seperti sebelumnya menggunakan *single linkage*, namun jarak antar titik dihitung dengan jarak *Manhattan (cityblock)*, yaitu penjumlahan selisih absolut antar dimensi.
3. `matrix_complete_eu = agglomerative_clustering(data, metric='euclidean', linkage='complete')` → Mengelompokkan data secara manual dengan metode *complete linkage*, yaitu menggabungkan dua kluster berdasarkan jarak terjauh antar titik dari kedua kluster, dengan pengukuran jarak menggunakan metrik *Euclidean*.
4. `matrix_complete_mh = agglomerative_clustering(data, metric='manhattan', linkage='complete')` → Sama seperti sebelumnya menggunakan *complete linkage*, namun menghitung jarak antar titik dengan metrik *Manhattan* (jumlah selisih absolut antar dimensi).

Output dari keempat baris kode ini adalah struktur data *linkage matrix* buatan sendiri, yang berisi informasi penggabungan kluster secara bertahap. Struktur ini mirip dengan hasil dari `scipy.linkage()` dan dapat digunakan untuk membangun *dendrogram*, yang menggambarkan proses pengelompokan hierarkis.

1. `label_single_eu = fcluster(matrix_single_eu, 4, criterion='maxclust')` → Menghasilkan label kluster dari hasil *hierarchical clustering* manual (*single linkage* dengan jarak *Euclidean*), dengan memotong *dendrogram* pada jumlah maksimum 4 kluster. Artinya, data dikelompokkan menjadi 4 kluster berdasarkan struktur `matrix_single_eu`.
2. `label_single_mh = fcluster(matrix_complete_mh, 4, criterion='maxclust')` → Sama seperti sebelumnya, namun label kluster diambil dari hasil *complete linkage* dengan jarak *Manhattan* (`matrix_complete_mh`), dan jumlah kluster yang diinginkan tetap 4.
3. `label_complete_eu = fcluster(matrix_complete_eu, 4, criterion='maxclust')` → Menghasilkan label kluster dari hasil *complete linkage* dengan metrik *Euclidean*, yaitu mengelompokkan data menjadi 4 kluster berdasarkan struktur `matrix_complete_eu`.
4. `label_complete_mh = fcluster(matrix_complete_mh, 4, criterion='maxclust')` → Sama seperti sebelumnya, tapi label diambil dari hasil *hierarchical clustering complete linkage* dengan metrik *Manhattan* (`matrix_complete_mh`), dan dibagi menjadi 4 kluster.

Keempat baris kode ini bertugas mengubah hasil *hierarchical clustering* (yang berupa *linkage matrix*) menjadi label kluster eksplisit untuk masing-masing data, berdasarkan jumlah kluster yang diinginkan.

4.6. Evaluasi

Evaluasi clustering yang digunakan dalam analisis ini meliputi *Silhouette Score*, *Davies-Bouldin Index (DBI)*, *Calinski-Harabasz (CH) Score*, *Rand Index*, dan *Purity*.

1. *Silhouette Score* mengukur seberapa mirip suatu data dengan kluster sendiri dibandingkan dengan kluster lain; nilai mendekati 1 menunjukkan kluster yang baik.
2. *Davies-Bouldin Index* mengukur rata-rata kemiripan antar kluster; nilai yang lebih rendah menunjukkan pemisahan kluster yang lebih baik.

3. *Calinski-Harabasz Score* menilai rasio dispersi antar-kluster dengan intra-kluster; nilai yang lebih tinggi menunjukkan struktur kluster yang lebih baik.
4. *Rand Index* membandingkan kesesuaian antara kluster hasil algoritma dan label sebenarnya; nilai 1 menunjukkan kesesuaian sempurna.
5. *Purity* menghitung proporsi data terbanyak dari satu kelas dalam setiap kluster; semakin mendekati 1, semakin sesuai kluster dengan label aslinya.
6. Untuk *Hierarchical Clustering*, juga digunakan visualisasi berupa *dendogram*, yaitu diagram pohon yang menunjukkan proses penggabungan kluster secara bertahap

4.6.1. Fungsi Untuk Evaluasi

a. Kode

```
def purity_score(y_true, y_pred) :
    # Compute contingency matrix
    contingency_matrix = np.zeros ((np.max(y_pred) +1, np.max(y_true)+1))
    for i in range (len(y_true) ):
        contingency_matrix[y_pred[i]] [y_true[i]] += 1

    return np.sum(np.amax (contingency_matrix, axis=1) ) /
    np.sum(contingency_matrix)
```

b. Penjelasan

Fungsi `purity_score(y_true, y_pred)` digunakan untuk menghitung nilai *purity*, yaitu ukuran seberapa "murni" hasil clustering dibandingkan dengan label asli (*ground truth*). *Purity* menggambarkan seberapa banyak data dalam satu kluster yang berasal dari kelas yang sama.

Rumus Purity:

$$Purity(C, W) = \frac{1}{N} \sum_{i=1}^k \max_j |c_i \cap w_j|$$

Keterangan:

- N : Jumlah total data $\rightarrow N = \text{len}(y_true)$ (jumlah elemen dalam label asli)
- k : Jumlah kluster hasil prediksi $\rightarrow k = \text{np.max}(y_pred) + 1$ (jumlah label unik dalam prediksi, diasumsikan label mulai dari 0)
- c_i : Himpunan data pada kluster ke- $i \rightarrow$ Data dengan $y_pred == i$
- w_j : Himpunan data dari kelas ground truth ke- $j \rightarrow$ Data dengan $y_true == j$
- $|c_i \cap w_j|$: Banyaknya data yang ada di kluster i dan juga termasuk dalam kelas $j \rightarrow \text{contingency_matrix}[i][j]$ (nilai frekuensi untuk pasangan kluster i dan kelas j)
- $\max_j |c_i \cap w_j|$: Jumlah anggota terbanyak dari satu kelas (kelas mayoritas) dalam kluster $i \rightarrow \text{np.amax}(\text{contingency_matrix}, \text{axis}=1)[i]$

Cara Kerja Kode:

1. Membuat *contingency matrix* berukuran (jumlah_klaster \times jumlah_kelas) yang menyimpan frekuensi data yang ada di masing-masing kombinasi klaster i dan kelas j .
2. Untuk setiap baris (klaster), diambil nilai maksimum, yaitu jumlah anggota kelas mayoritas dalam klaster tersebut.
3. Semua nilai maksimum dijumlahkan, lalu dibagi dengan total jumlah data (N) untuk mendapatkan nilai purity.

Nilai *purity* berada dalam rentang 0 hingga 1, dimana nilai yang lebih tinggi menunjukkan bahwa hasil *clustering* semakin sesuai dengan label kelas sebenarnya.

4.6.2. Persiapan Data**a. Kode**

```
X = data
y = np.array(label)
```

4.6.3. K-Means**4.6.3.1. K-Means Library****a. Kode**

```
sil_score = silhouette_score(X, kmeans_library_labels)
db_index = davies_bouldin_score(X, kmeans_library_labels)
ch_score = calinski_harabasz_score(X, kmeans_library_labels)
rand_index = adjusted_rand_score(y, kmeans_library_labels)

print(f'Silhouette Score: {sil_score}')
print(f'Davies-Bouldin Index: {db_index}')
print(f'Calinski-Harabasz Index: {ch_score}')
print(f'Rand score: {rand_index} ")

purity = purity_score(y, kmeans_library_labels)
print("Purity Score : ", purity)
```

4.6.3.2. K-means Manual

a. Kode

```

sil_score = silhouette_score(X, kmeans_manual_labels)
db_index = davies_bouldin_score(X, kmeans_manual_labels)
ch_score = calinski_harabasz_score(X, kmeans_manual_labels)
rand_index = adjusted_rand_score(y, kmeans_manual_labels)

print(f'Silhouette Score: {sil_score}')
print(f'Davies-Bouldin Index: {db_index}')
print(f'Calinski-Harabasz Index: {ch_score}')
print(f'Rand score: {rand_index} ')

def purity_score(y_true, y_pred):
    # Compute contingency matrix
    contingency_matrix = np.zeros((np.max(y_pred)+1, np.max(y_true)+1))
    for i in range(len(y_true)):
        contingency_matrix[y_pred[i]][y_true[i]] += 1

    return np.sum(np.amax(contingency_matrix, axis=1)) /
np.sum(contingency_matrix)

purity = purity_score(y, kmeans_manual_labels)
print("Purity Score : ", purity)

```

b. Penjelasan

Berdasarkan hasil evaluasi, model *K-Means* dari library sedikit lebih baik dibandingkan dengan *K-Means* manual. Hal ini terlihat dari nilai *Silhouette Score* dan *Calinski-Harabasz Index* yang sedikit lebih tinggi, serta *Davies-Bouldin Index* yang sedikit lebih rendah, menunjukkan bahwa struktur kluster yang dihasilkan lebih kompak dan terpisah dengan lebih baik. Selain itu, nilai *Rand Score* dan *Purity* juga sedikit lebih tinggi pada versi *library*, mengindikasikan hasil kluster yang lebih konsisten dengan label sebenarnya.

Beberapa Hal yang Mungkin/Dapat Menjadi Alasan Mengapa *K-Means Library* Lebih Baik daripada *K-Means Manual*

1. Optimasi numerik yang lebih baik: *Library* seperti scikit-learn menggunakan implementasi yang telah dioptimalkan dengan algoritma seperti *k-means++* initialization dan operasi vektorisasi penuh, sehingga proses konvergensi lebih efisien dan hasil kluster lebih stabil.
2. Inisialisasi centroid yang lebih cerdas: *K-Means* dari library menggunakan strategi inisialisasi *k-means++* yang memilih *centroid* awal secara strategis untuk

- memaksimalkan jarak antar *centroid*. Ini mengurangi risiko hasil buruk akibat inisialisasi acak seperti pada *K-Means* manual.
3. Vektorisasi dan pengelolaan numerik yang lebih akurat: Perhitungan jarak dan *centroid* dilakukan dengan *precision* tinggi dan lebih cepat, serta lebih tahan terhadap *noise* dan *outlier* dibanding implementasi manual yang lebih sederhana.
 4. Evaluasi iteratif yang lebih matang: Implementasi *library* sudah memperhitungkan toleransi konvergensi, maksimisasi efisiensi memori, dan penanganan kondisi ekstrem (misalnya *centroid* tanpa anggota) dengan cara yang lebih robust dibanding pendekatan from scratch.

Secara keseluruhan, meskipun implementasi manual dapat digunakan untuk pembelajaran konsep dasar, penggunaan *library* memberikan hasil yang lebih dapat diandalkan dan efisien dalam praktik nyata.

4.6.4. Hierarchical Clustering

4.6.4.1. Hierarchical Library

a. Kode

```
# Score HC Single Euclidean Library

sil_score = silhouette_score(X, single_eu_labels)
db_index = davies_bouldin_score(X, single_eu_labels)
ch_score = calinski_harabasz_score(X, single_eu_labels)
rand_index = adjusted_rand_score(y, single_eu_labels)

print(f'Silhouette Score: {sil_score}')
print(f'Davies-Bouldin Index: {db_index}')
print(f'Calinski-Harabasz Index: {ch_score}')
print(f'Rand score: {rand_index}')

purity = purity_score(y, single_eu_labels)
print("Purity Score:", purity)

# Score HC Single Manhattan Library

sil_score = silhouette_score(X, single_mh_labels)
db_index = davies_bouldin_score(X, single_mh_labels)
ch_score = calinski_harabasz_score(X, single_mh_labels)
rand_index = adjusted_rand_score(y, single_mh_labels)

print(f'Silhouette Score: {sil_score}')
```



```
print(f'Davies-Bouldin Index: {db_index}")
print(f'Calinski-Harabasz Index: {ch_score}")
print(f'Rand score: {rand_index}")

purity = purity_score(y, single_mh_labels)
print("Purity Score:", purity)

# Score HC Complete Euclidean Library

sil_score = silhouette_score(X, complete_eu_labels)
db_index = davies_bouldin_score(X, complete_eu_labels)
ch_score = calinski_harabasz_score(X, complete_eu_labels)
rand_index = adjusted_rand_score(y, complete_eu_labels)

print(f'Silhouette Score: {sil_score}")
print(f'Davies-Bouldin Index: {db_index}")
print(f'Calinski-Harabasz Index: {ch_score}")
print(f'Rand score: {rand_index}")

purity = purity_score(y, complete_eu_labels)
print("Purity Score:", purity)

# Score HC Complete Manhattan Library

sil_score = silhouette_score(X, complete_mh_labels)
db_index = davies_bouldin_score(X, complete_mh_labels)
ch_score = calinski_harabasz_score(X, complete_mh_labels)
rand_index = adjusted_rand_score(y, complete_mh_labels)

print(f'Silhouette Score: {sil_score}")
print(f'Davies-Bouldin Index: {db_index}")
print(f'Calinski-Harabasz Index: {ch_score}")
print(f'Rand score: {rand_index}")

purity = purity_score(y, complete_mh_labels)
print("Purity Score:", purity)
```

b. Output

Metode	Silhouette Score	Davies-Bouldin Index	Calinski-Harabasz Index	Rand Score	Purity Score
HC Single Euclidean Library	-0.00298	0.75418	1.74158	-0.000077	0.28072
HC Single Manhattan Library	0.04493	0.76455	1.65547	-0.000045	0.28072
HC Complete Euclidean Library	0.12003	1.92545	169.61902	0.09090	0.41558
HC Complete Manhattan Library	0.12776	2.10939	173.70221	0.06037	0.38561

c. Penjelasan

Analisis Singkat:

- *Silhouette Score* tertinggi diperoleh oleh *HC Complete Manhattan*, menunjukkan pemisahan cluster yang relatif lebih baik dibanding metode lain.
- *Davies-Bouldin Index (DBI)* terendah dimiliki oleh *HC Single Euclidean*, namun skor lainnya rendah karena *Silhouette*-nya negatif.
- *Calinski-Harabasz Index (CHI)* tertinggi ada pada *HC Complete Manhattan*, mengindikasikan pemisahan antar cluster cukup optimal.
- *Rand Score* dan *Purity Score* tertinggi diperoleh oleh *HC Complete Euclidean*.

HC Complete Euclidean Library adalah model terbaik secara keseluruhan karena memiliki kombinasi *Rand Score* dan *Purity Score* tertinggi, serta nilai *Silhouette* dan *CHI* yang cukup baik, meskipun *DBI*-nya sedikit lebih tinggi dibanding metode lain. Ini menunjukkan model tersebut paling mendekati struktur label sebenarnya dan membentuk cluster yang cukup baik.

4.6.4.2. Dendrogram

a. Kode

```
# Single Linkage Euclidean
plt.figure(figsize=(8, 4))
dendrogram(linked_single_eu, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title('Library - Single Linkage (Euclidean Distance) ')
plt.show()

# Single Linkage Manhattan
plt.figure(figsize=(8, 4))
dendrogram(linked_single_mh, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title('Library - Single Linkage (Manhattan Distance) ')
plt.show()

# Complete Linkage Euclidean
```



```
plt.figure(figsize=(8, 4))
dendrogram(linked_complete_eu, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title('Library - Complete Linkage (Euclidean Distance) ')
plt.show()

# Complete Linkage Manhattan
plt.figure(figsize=(8, 4))
dendrogram(linked_complete_mh, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title('Library - Complete Linkage (Manhattan Distance) ')
plt.show()
```

4.6.4.3. Hierarchical Manual

a. Kode

```
# Score HC Single Euclidean Manual

sil_score = silhouette_score(X, label_single_eu)
db_index = davies_bouldin_score(X, label_single_eu)
ch_score = calinski_harabasz_score(X, label_single_eu)
rand_index = adjusted_rand_score(y, label_single_eu)

print(f'Silhouette Score: {sil_score}')
print(f'Davies-Bouldin Index: {db_index}')
print(f'Calinski-Harabasz Index: {ch_score}')
print(f'Rand score: {rand_index}')

purity = purity_score(y, label_single_eu)
print("Purity Score:", purity)

# Score HC Single Manhattan Manual

sil_score = silhouette_score(X, label_single_mh)
db_index = davies_bouldin_score(X, label_single_mh)
ch_score = calinski_harabasz_score(X, label_single_mh)
rand_index = adjusted_rand_score(y, label_single_mh)

print(f'Silhouette Score: {sil_score}')
```

```
print(f'Davies-Bouldin Index: {db_index}")
print(f'Calinski-Harabasz Index: {ch_score}")
print(f'Rand score: {rand_index}")

purity = purity_score(y, label_single_mh)
print("Purity Score:", purity)

# Score HC Complete Euclidean Manual

sil_score = silhouette_score(X, label_complete_eu)
db_index = davies_bouldin_score(X, label_complete_eu)
ch_score = calinski_harabasz_score(X, label_complete_eu)
rand_index = adjusted_rand_score(y, label_complete_eu)

print(f'Silhouette Score: {sil_score}")
print(f'Davies-Bouldin Index: {db_index}")
print(f'Calinski-Harabasz Index: {ch_score}")
print(f'Rand score: {rand_index}")

purity = purity_score(y, label_complete_eu)
print("Purity Score:", purity)

# Score HC Complete Manhattan Manual

sil_score = silhouette_score(X, label_complete_mh)
db_index = davies_bouldin_score(X, label_complete_mh)
ch_score = calinski_harabasz_score(X, label_complete_mh)
rand_index = adjusted_rand_score(y, label_complete_mh)

print(f'Silhouette Score: {sil_score}")
print(f'Davies-Bouldin Index: {db_index}")
print(f'Calinski-Harabasz Index: {ch_score}")
print(f'Rand score: {rand_index}")

purity = purity_score(y, label_complete_mh)
print("Purity Score:", purity)
```

b. Output

Metode	Silhouette Score	Davies-Bouldin Index	Calinski-Harabasz Index	Rand Score	Purity Score
HC Single Euclidean Manual	-0.00298	0.75418	1.74158	-0.000077	0.28072
HC Single Manhattan Manual	0.12776	2.10939	173.70221	0.06037	0.38561
HC Complete Euclidean Manual	0.13908	2.32269	188.19509	0.08407	0.42258
HC Complete Manhattan Manual	0.12776	2.10939	173.70221	0.06037	0.38561

c. Penjelasan

Analisis Singkat:

- *Silhouette Score* tertinggi dimiliki oleh *HC Complete Euclidean Manual*, menunjukkan bahwa cluster yang dibentuk cukup terpisah dan kompak.
- *Davies-Bouldin Index (DBI)* terendah ada pada *HC Single Euclidean*, tapi metode ini memiliki *Silhouette* yang negatif dan *CHI* sangat rendah, mengindikasikan clustering yang buruk.
- *Calinski-Harabasz Index (CHI)* tertinggi juga dimiliki oleh *HC Complete Euclidean Manual*, memperkuat indikasi pemisahan cluster yang baik.
- *Rand Score* dan *Purity Score* tertinggi juga diperoleh oleh *HC Complete Euclidean Manual*, yang menunjukkan kesesuaian hasil clustering dengan label asli.

Model *HC Complete Euclidean Manual* merupakan metode paling unggul berdasarkan evaluasi. Ia unggul di semua aspek penting seperti *Silhouette Score*, *Calinski-Harabasz Index*, *Rand Score*, dan *Purity Score*. Meskipun *DBI*-nya tidak paling rendah, nilai tersebut masih wajar dan tidak mengurangi kualitas model secara keseluruhan.

4.6.4.4. Dendrogram

a. Kode

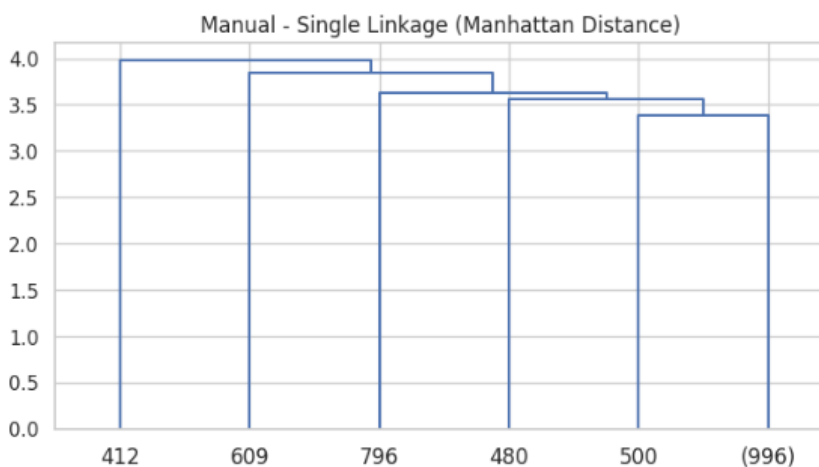
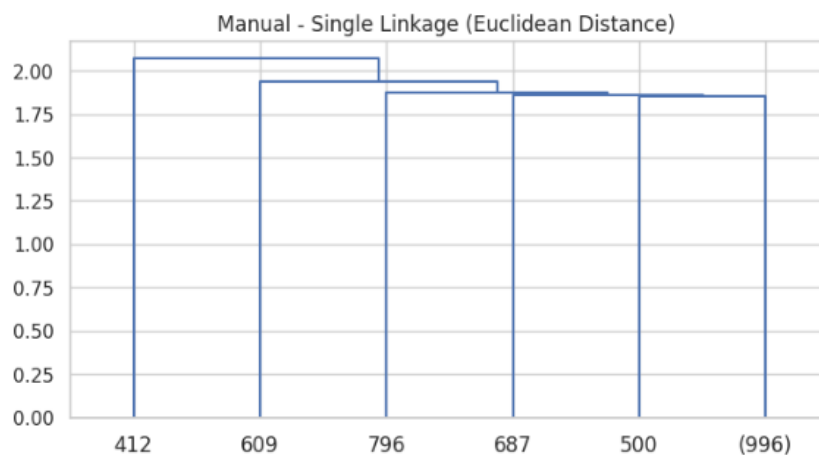
```
# Manual Single Linkage Euclidean
plt.figure(figsize=(8, 4))
dendrogram(matrix_single_eu, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title('Manual - Single Linkage (Euclidean Distance) ')
plt.show()

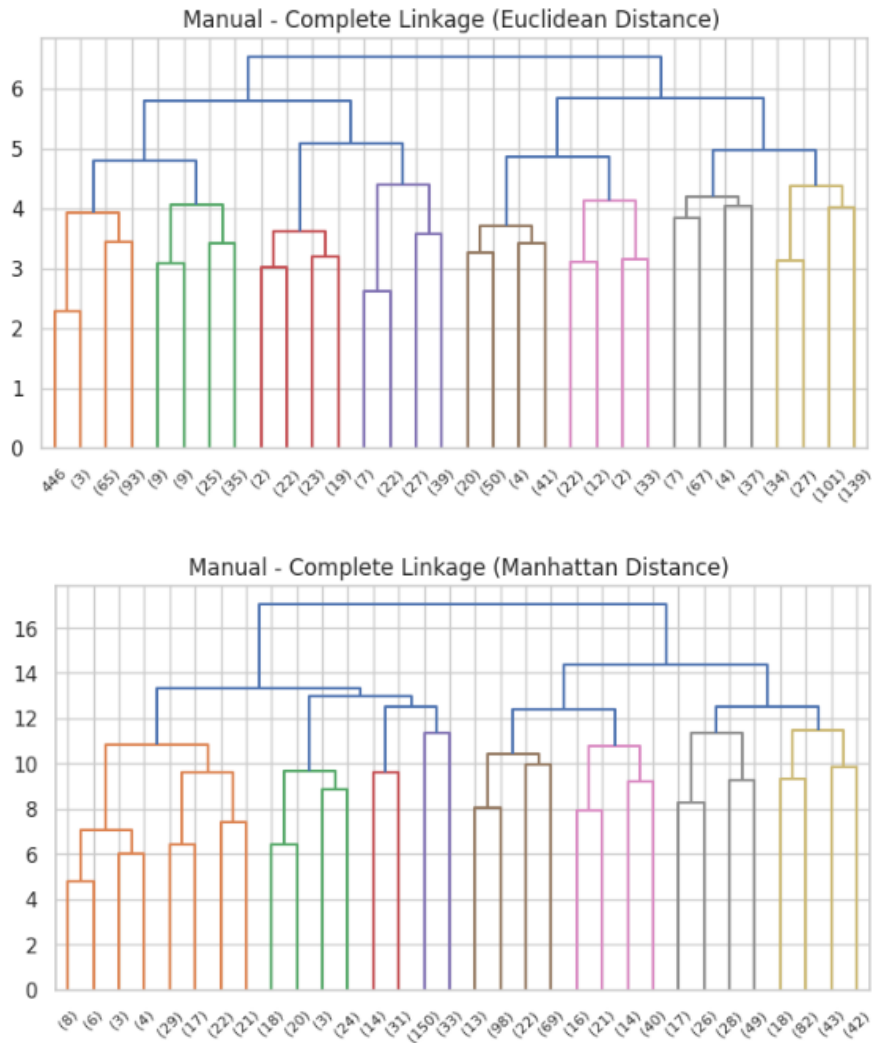
# Manual Single Linkage Manhattan
plt.figure(figsize=(8, 4))
dendrogram(matrix_single_mh, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title('Manual - Single Linkage (Manhattan Distance) ')
plt.show()
```

```
# Manual Complete Linkage Euclidean
plt.figure(figsize=(8, 4))
dendrogram(linked_complete_eu, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title ('Manual - Complete Linkage (Euclidean Distance) ')
plt.show()

# Manual Complete Linkage Manhattan
plt.figure(figsize=(8, 4))
dendrogram(linked_complete_mh, labels=data.index, truncate_mode='level',
orientation='top', p=4)
plt.title ('Manual - Complete Linkage (Manhattan Distance) ')
plt.show()
```

b. Output





4.6.5. Hasil Evaluasi

Hasil clustering yang kurang optimal dapat disebabkan oleh berbagai faktor, salah satunya adalah karakteristik data yang digunakan. Pada dataset yang digunakan struktur datanya mungkin belum cukup mendukung pembentukan kluster yang jelas. Hal ini bisa terjadi karena fitur-fitur yang tersedia belum cukup representatif untuk membedakan kelompok yang bermakna, baik dari sisi statistik maupun bisnis. Selain itu, tingginya variasi perilaku pelanggan dan adanya *noise* dalam data transaksi juga dapat mempersulit proses segmentasi. Oleh karena itu, hasil *clustering* yang kurang baik tidak selalu mencerminkan kelemahan algoritma, tetapi juga bisa menunjukkan perlunya eksplorasi lebih lanjut terhadap kualitas data dan pemilihan fitur yang digunakan.

4.7. PCA

4.7.1. Data PCA

a. Kode

```
X = data.copy()

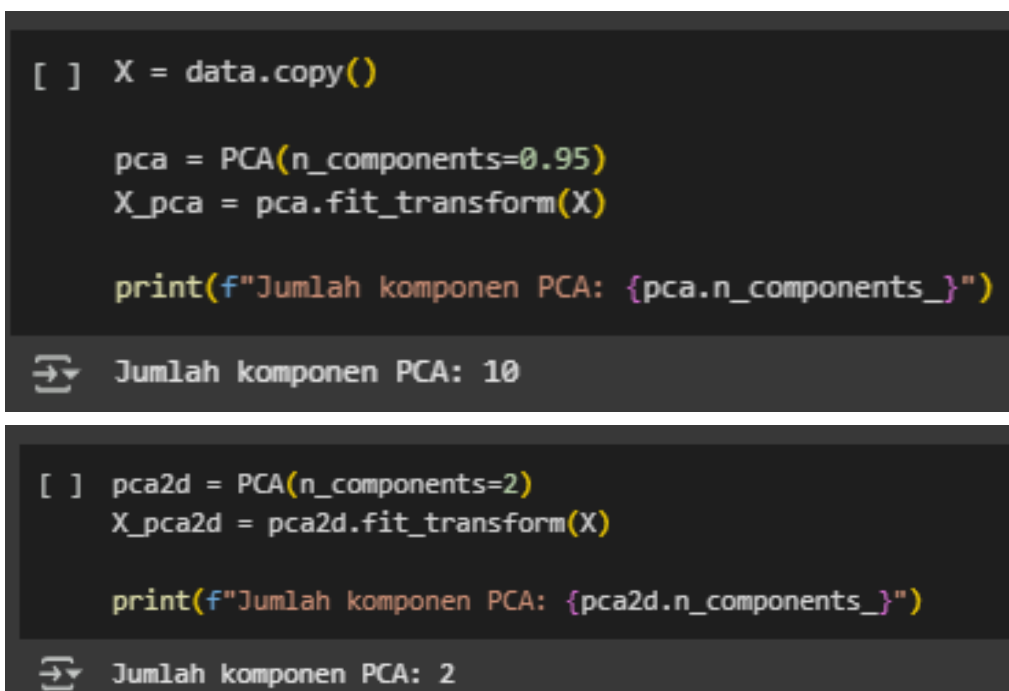
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X)

print(f"Jumlah komponen PCA: {pca.n_components_}")

pca2d = PCA(n_components=2)
X_pca2d = pca2d.fit_transform(X)

print(f"Jumlah komponen PCA: {pca2d.n_components_}")
```

b. Output



The image shows two screenshots of a Jupyter Notebook. The first screenshot displays the execution of the first part of the code: `X = data.copy()`, `pca = PCA(n_components=0.95)`, `X_pca = pca.fit_transform(X)`, and `print(f"Jumlah komponen PCA: {pca.n_components_}")`. The output below the code is `Jumlah komponen PCA: 10`. The second screenshot displays the execution of the second part of the code: `pca2d = PCA(n_components=2)`, `X_pca2d = pca2d.fit_transform(X)`, and `print(f"Jumlah komponen PCA: {pca2d.n_components_}")`. The output below the code is `Jumlah komponen PCA: 2`.

c. Penjelasan

`X_pca` adalah hasil transformasi data dengan PCA yang mempertahankan total variansi $\geq 95\%$, artinya jumlah komponen yang dipilih otomatis oleh PCA adalah sebanyak mungkin hingga kumulatif variansinya mencapai minimal 95%. Ini bertujuan untuk mengurangi dimensi tanpa kehilangan terlalu banyak informasi. Sementara itu, `X_pca2d` adalah hasil transformasi dari PCA dengan jumlah komponen tetap = 2, yaitu dua komponen utama

terbaik berdasarkan variansi tertinggi. Tujuannya adalah untuk visualisasi dan eksplorasi struktur data dalam ruang dua dimensi.

4.7.2. *K-means*

a. Kode

```
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans_labels = kmeans.fit_predict(X_pca)

kmeans2d = KMeans(n_clusters=4, random_state=42)
kmeans_labels2d = kmeans2d.fit_predict(X_pca2d)
```

b. Penjelasan

Kode diatas menerapkan algoritma *K-Means clustering* untuk membentuk empat klaster dari data yang telah direduksi dimensinya menggunakan PCA. Proses pertama (*kmeans*) dilakukan pada data PCA berdimensi penuh (*X_pca*), sedangkan proses kedua (*kmeans2d*) diterapkan pada data PCA yang direduksi menjadi dua dimensi (*X_pca2d*) untuk keperluan visualisasi. Kedua model menggunakan parameter *random_state=42* agar hasil klasterisasi bersifat konsisten.

4.7.3. *Hierarchical*

a. Kode

```
hc = AgglomerativeClustering(n_clusters=4, linkage='complete', metric='euclidean')
hc_labels = hc.fit_predict(X_pca)

hc2d = AgglomerativeClustering(n_clusters=4, linkage='complete',
metric='euclidean')
hc2d_labels = hc2d.fit_predict(X_pca2d)
```

b. Penjelasan

Kode tersebut menerapkan algoritma *Agglomerative Clustering* dengan jumlah klaster sebanyak empat, menggunakan metode penggabungan *complete linkage* dan metrik jarak *euclidean*. Proses klasterisasi pertama (*hc*) dilakukan pada data hasil reduksi dimensi penuh (*X_pca*), sementara proses kedua (*hc2d*) diterapkan pada data dua dimensi (*X_pca2d*) untuk visualisasi. Kedua model bertujuan mengelompokkan data secara hierarkis berdasarkan kemiripan antar data.

4.7.4. Evaluasi untuk PCA

a. Kode

```
def evaluate_clustering(X, y_true, y_pred):
    sil_score = silhouette_score(X, y_pred)
    db_index = davies_bouldin_score(X, y_pred)
    ch_score = calinski_harabasz_score(X, y_pred)
    rand_index = adjusted_rand_score(y_true, y_pred)
    purity = purity_score(y_true, y_pred)

    print(f'Silhouette Score: {sil_score}')
    print(f'Davies-Bouldin Index: {db_index}')
    print(f'Calinski-Harabasz Index: {ch_score}')
    print(f'Rand score: {rand_index}')
    print(f'Purity Score: {purity}')

print('Kmeans PCA 95%')
evaluate_clustering(X_pca, label, kmeans_labels)

print('\nKmeans PCA 2D')
evaluate_clustering(X_pca2d, label, kmeans_labels2d)

print('\nHC PCA 95%')
evaluate_clustering(X_pca, label, hc_labels)

print('\nHC PCA 2D')
evaluate_clustering(X_pca2d, label, hc2d_labels)
```


b. Output

```

Kmeans PCA 95%
Silhouette Score: 0.24887299727342618
Davies-Bouldin Index: 1.3454043012493135
Calinski-Harabasz Index: 304.6653028113468
Rand score: 0.07523826575336025
Purity Score: 0.4095904095904096

Kmeans PCA 2D
Silhouette Score: 0.4137801386626036
Davies-Bouldin Index: 0.8517035060873877
Calinski-Harabasz Index: 970.7987591411581
Rand score: 0.08090457309373494
Purity Score: 0.4165834165834166

HC PCA 95%
Silhouette Score: 0.1353311213147043
Davies-Bouldin Index: 1.9705818130397699
Calinski-Harabasz Index: 182.9356164358595
Rand score: 0.06289293764794952
Purity Score: 0.3786213786213786

HC PCA 2D
Silhouette Score: 0.332203572960161
Davies-Bouldin Index: 0.8373794331829988
Calinski-Harabasz Index: 606.2458311161539
Rand score: 0.05602444958510288
Purity Score: 0.3786213786213786

```

c. Penjelasan

Dataset hasil transformasi `X_pca` dan `X_pca2d` digunakan sebagai input untuk evaluasi metode *clustering*, dengan tujuan mengukur performa menggunakan lima metrik: *Silhouette Score*, *Davies-Bouldin Index (DBI)*, *Calinski-Harabasz Score (CH Score)*, *Rand Index*, dan *Purity Score*.

4.8. Analisis Segmentasi Customer Hasil Clustering

4.8.1. Load Data

a. Kode

```

data_analisis = data.copy()
data_analisis

```

b. Output

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Profession_Artist	Profession_Doctor	Profession_Engineer	Profession_Entertainment	Profession_Executive	Profession_Healthcare	Profession_Housemaker	Profession_Lawyer	Profession_Marketing
0	1	1	2.030280	1	-0.202423	1.973391	-0.473324	False	False	False	False	True	False	False	False	False
1	0	1	1.254885	1	-0.202423	-0.738594	-1.584262	False	False	True	False	False	False	False	False	False
2	1	1	0.339806	0	-1.100537	0.617398	0.314899	False	False	False	True	False	False	False	False	False
3	0	0	-1.056216	1	-1.100537	-0.738594	0.926293	False	False	False	False	False	True	False	False	False
4	1	0	-0.094863	1	-0.202423	-0.738594	-1.584262	True	False	False	False	False	False	False	False	False
...
996	1	0	-0.613367	0	-1.100537	-0.738594	0.926293	True	False	False	False	False	False	False	False	False
997	0	0	-1.056216	0	-0.202423	-0.738594	0.314899	True	False	False	False	False	False	False	False	False
998	0	0	-0.163103	1	-0.202423	-0.738594	-1.584262	True	False	False	False	False	False	False	False	False
999	0	1	0.714014	1	-0.202423	-0.738594	-0.473324	True	False	False	False	False	False	False	False	False
1000	1	0	-1.255960	1	-0.202423	-0.738594	0.926293	False	False	False	False	False	True	False	False	False

c. Penjelasan

Kode diatas membuat salinan dari *dataframe* “data” dengan nama “data_analisis”.

4.8.2. Reverse Transform Data

a. Kode

```

ohe_cols = ["Profession_Artist","Profession_Doctor", "Profession_Engineer",
"Profession_Entertainment", "Profession_Executive", "Profession_Healthcare",
"Profession_Homemaker", "Profession_Lawyer", "Profession_Marketing"]

data_analisis['Profession'] = data_analisis[ohe_cols].idxmax(axis=1)
data_analisis['Profession'] = data_analisis['Profession'].str.replace('Profession_', '')
data_analisis.drop(ohe_cols, axis=1, inplace=True)
data_analisis
scaled_cols = ["Age", "Work_Experience", "Family_Size", "Spending_Score"]

# Inverse scaling
data_analisis[scaled_cols] = scaler.inverse_transform(data_analisis[scaled_cols])

# Kalau pakai log1p sebelumnya:
log_cols = ["Age", "Work_Experience", "Family_Size"]

# Inverse log1p
data_analisis[log_cols] = np.expm1(data_analisis[log_cols])
data_analisis

```

b. Output

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Profession
0	1	1	2.030280	1	-0.202423	1.973391	-0.473324	Executive
1	0	1	1.254885	1	-0.202423	-0.738594	-1.584262	Engineer
2	1	1	0.339806	0	-1.100537	0.617398	0.314899	Entertainment
3	0	0	-1.056216	1	-1.100537	-0.738594	0.926293	Healthcare
4	1	0	-0.094863	1	-0.202423	-0.738594	-1.584262	Artist
...
996	1	0	-0.613367	0	-1.100537	-0.738594	0.926293	Artist
997	0	0	-1.056216	0	-0.202423	-0.738594	0.314899	Artist
998	0	0	-0.163103	1	-0.202423	-0.738594	-1.584262	Artist
999	0	1	0.714014	1	-0.202423	-0.738594	-0.473324	Artist
1000	1	0	-1.255960	1	-0.202423	-0.738594	0.926293	Healthcare

c. Penjelasan

Kode tersebut digunakan untuk merekonstruksi kembali data asli setelah proses transformasi sebelumnya seperti *one-hot encoding*, *scaling*, dan *log transformation*. Pertama, kolom-kolom profesi yang telah di-*one-hot encode* dikembalikan ke format kategorikal melalui `idxmax()` dan penghapusan *prefix*, kemudian kolom-kolom *one-hot* dihapus dari data. Selanjutnya, data pada kolom numerik yang telah dinormalisasi (*Age*, *Work_Experience*, *Family_Size*, dan *Spending_Score*) dikembalikan ke skala aslinya menggunakan *inverse_transform*. Terakhir, transformasi logaritmik pada kolom *Age*, *Work_Experience*, dan *Family_Size* di-*reverse* menggunakan `np.expml()` agar nilainya kembali mendekati kondisi awal sebelum dilakukan logaritma.

4.8.3. Menambahkan Kolom "*Cluster*" Sebagai Label

a. Kode

```
data_analisis['Cluster'] = kmeans_labels2d

data_analisis['Cluster'].value_counts()

print(f"\n=== Cluster {0} ===")
display(data_analisis[data_analisis['Cluster'] == 0].describe())

print(f"\n=== Cluster {1} ===")
display(data_analisis[data_analisis['Cluster'] == 1].describe())

print(f"\n=== Cluster {2} ===")
display(data_analisis[data_analisis['Cluster'] == 2].describe())

print(f"\n=== Cluster {3} ===")
display(data_analisis[data_analisis['Cluster'] == 3].describe())

# Persentase Pekerjaan tiap Cluster
pd.crosstab(data_analisis['Cluster'], data_analisis['Profession'], normalize='index') *
100
```

b. Output

=== Cluster 0 ===								
	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Cluster
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	0.508251	0.531353	46.706271	0.735974	2.752475	0.069307	1.415842	0.0
std	0.500759	0.499842	15.063976	0.441543	3.247228	0.254395	0.544703	0.0
min	0.000000	0.000000	22.000000	0.000000	0.000000	0.000000	1.000000	0.0
25%	0.000000	0.000000	37.000000	0.000000	1.000000	0.000000	1.000000	0.0
50%	1.000000	1.000000	43.000000	1.000000	1.000000	0.000000	1.000000	0.0
75%	1.000000	1.000000	53.000000	1.000000	5.000000	0.000000	2.000000	0.0
max	1.000000	1.000000	87.000000	1.000000	10.000000	1.000000	4.000000	0.0

=== Cluster 1 ===								
	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Cluster
count	153.000000	153.0	153.000000	153.000000	153.000000	153.000000	153.000000	153.0
mean	0.549020	1.0	66.568627	0.699346	0.869281	1.653595	2.313725	1.0
std	0.499225	0.0	11.670230	0.460048	1.422067	0.529651	0.622713	0.0
min	0.000000	1.0	33.000000	0.000000	0.000000	0.000000	1.000000	1.0
25%	0.000000	1.0	58.000000	0.000000	0.000000	1.000000	2.000000	1.0
50%	1.000000	1.0	67.000000	1.000000	1.000000	2.000000	2.000000	1.0
75%	1.000000	1.0	76.000000	1.000000	1.000000	2.000000	3.000000	1.0
max	1.000000	1.0	87.000000	1.000000	9.000000	2.000000	4.000000	1.0

=== Cluster 2 ===								
	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Cluster
count	274.000000	274.000000	274.000000	274.000000	274.000000	274.000000	274.000000	274.0
mean	0.532847	0.131387	27.788321	0.332117	2.452555	0.021898	3.781022	2.0
std	0.499833	0.338441	6.347600	0.471834	3.113178	0.146618	1.130434	0.0
min	0.000000	0.000000	18.000000	0.000000	0.000000	0.000000	1.000000	2.0
25%	0.000000	0.000000	23.000000	0.000000	0.000000	0.000000	3.000000	2.0
50%	1.000000	0.000000	28.000000	0.000000	1.000000	0.000000	4.000000	2.0
75%	1.000000	0.000000	32.000000	1.000000	4.000000	0.000000	4.000000	2.0
max	1.000000	1.000000	50.000000	1.000000	9.000000	1.000000	7.000000	2.0

=== Cluster 3 ===								
	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Cluster
count	271.000000	271.000000	271.000000	271.000000	271.000000	271.000000	271.000000	271.0
mean	0.660517	0.985240	45.335793	0.719557	1.653137	1.036900	3.446494	3.0
std	0.474410	0.120815	10.395234	0.450047	2.308916	0.582558	1.090318	0.0
min	0.000000	0.000000	22.000000	0.000000	0.000000	0.000000	1.000000	3.0
25%	0.000000	1.000000	38.000000	0.000000	0.000000	1.000000	3.000000	3.0
50%	1.000000	1.000000	46.000000	1.000000	1.000000	1.000000	3.000000	3.0
75%	1.000000	1.000000	51.500000	1.000000	1.000000	1.000000	4.000000	3.0
max	1.000000	1.000000	81.000000	1.000000	10.000000	2.000000	7.000000	3.0

Profession	Artist	Doctor	Engineer	Entertainment	Executive	Healthcare	Homemaker	Lawyer	Marketing
Cluster									
0	42.244224	8.910891	7.590759	15.511551	1.650165	5.610561	3.630363	11.221122	3.630363
1	31.372549	1.960784	6.535948	2.614379	20.261438	0.653595	0.000000	33.986928	2.614379
2	10.948905	12.773723	6.569343	9.854015	2.189781	47.810219	4.014599	0.364964	5.474453
3	43.542435	9.225092	8.856089	14.760148	14.022140	3.690037	2.583026	1.476015	1.845018

c. Penjelasan

Kode tersebut digunakan untuk mengevaluasi hasil klasterisasi dengan menambahkan label klaster dari hasil *K-Means* dua dimensi ke dalam dataset. Kemudian, jumlah anggota tiap klaster dihitung menggunakan `value_counts()`. Untuk memahami karakteristik masing-masing klaster, dilakukan analisis deskriptif statistik (`describe()`) per klaster. Terakhir, distribusi profesi dalam setiap klaster dihitung dalam bentuk persentase menggunakan `crosstab()` dengan normalisasi per baris, sehingga dapat diketahui profesi dominan di masing-masing klaster.

BAB V

PEMBAHASAN & HASIL

5.1. Analisis Data

Ditemukan bahwa dataset *data_train* terdiri dari 8068 baris dan 11 kolom. Terdapat fitur kategorikal seperti *Gender*, *Ever_Married*, *Graduated*, *Profession*, *Spending_Score*, dan *Segmentation* serta numerik seperti *Age*, *Work_Experience*, dan *Family_Size*. Beberapa fitur seperti *Ever_Married*, *Graduated*, *Profession*, *Work_Experience*, *Family_Size*, dan *Var_1* memiliki *missing value* yang perlu ditangani. Tidak ditemukan baris duplikat dalam data, namun perlu dilakukan pembersihan data melalui imputasi nilai hilang dan encoding fitur kategorikal untuk keperluan analisis lebih lanjut.

5.2. Fokus Evaluasi

Fokus evaluasi utama dalam tugas ini adalah melakukan clustering terhadap data pelanggan menggunakan metode *unsupervised learning*, khususnya *K-Means* dan *Agglomerative Clustering*, lalu mengevaluasi hasil clustering dengan metrik internal dan eksternal seperti *Silhouette Score*, *Davies-Bouldin Index*, *Calinski-Harabasz Score*, dan *Adjusted Rand Index (ARI)*. Evaluasi dilakukan dengan membandingkan hasil prediksi cluster dengan label ground truth yang tersedia, yaitu kolom *Segmentation*.

5.3. Analisis Kinerja Model

5.3.1. Preprocessing Data

Tahapan pra-pemrosesan dilakukan secara menyeluruh untuk memastikan data sudah siap digunakan dalam pemodelan *clustering*. Penanganan nilai hilang dilakukan berdasarkan karakteristik fitur: imputasi median digunakan untuk fitur numerikal yang *skewed* (*Work_Experience*, *Family_Size*), sementara imputasi modus digunakan untuk fitur kategorikal seperti *Profession*, *Ever_Married*, dan *Graduated*. Fitur yang dianggap tidak relevan atau bias seperti *Var_1* dihapus. Kemudian, data dikonversi ke bentuk numerik melalui teknik encoding: *Label Encoding* untuk fitur biner, *Ordinal Encoding* untuk *Spending_Score*, dan *One-Hot Encoding* untuk *Profession*. Setelah itu, dilakukan log transformation pada fitur numerikal untuk mengurangi *skewness*, diikuti dengan standard scaling menggunakan *StandardScaler*.

5.3.2. Reduksi Dimensi

Reduksi dimensi dilakukan menggunakan *Principal Component Analysis (PCA)* dengan dua pendekatan, yaitu PCA yang mempertahankan $\geq 95\%$ variansi (dengan jumlah komponen yang ditentukan secara otomatis) dan PCA 2D yang digunakan untuk keperluan visualisasi.

Penerapan PCA terbukti membantu meningkatkan efisiensi pemrosesan serta mempermudah eksplorasi visual hasil clustering. Selain itu, PCA juga sedikit meningkatkan performa model, terutama pada metode *clustering* yang sensitif terhadap korelasi antar fitur seperti *Hierarchical Clustering*.

5.3.3. K-Means Clustering

Metode *K-Means* digunakan dengan jumlah cluster 4, sesuai jumlah segmen pada label *Segmentation*. Dua implementasi digunakan: versi *library* scikit-learn dan versi manual dari awal. *K-Means library* memberikan hasil lebih baik dengan *Silhouette Score* dan *Calinski-Harabasz Index* yang lebih tinggi, serta *Davies-Bouldin Index* yang lebih rendah. Nilai *Adjusted Rand Index (ARI)* dan *Purity* juga lebih tinggi pada versi *library*, mengindikasikan kesesuaian yang lebih baik dengan ground truth.

5.3.4. Agglomerative Hierarchical Clustering

Metode *Agglomerative Hierarchical Clustering* diuji menggunakan berbagai kombinasi teknik linkage (*Single* dan *Complete*) serta metrik jarak (*Euclidean* dan *Manhattan*). Hasil evaluasi menunjukkan bahwa pendekatan *Complete Linkage* dengan *Euclidean Distance*, baik yang diimplementasikan melalui library maupun secara manual, memberikan performa terbaik secara keseluruhan. Hal ini ditunjukkan oleh pencapaian nilai tertinggi pada metrik evaluasi seperti *Silhouette Score*, *Calinski-Harabasz Index*, *Adjusted Rand Index (ARI)*, dan *Purity Score* dibandingkan dengan varian metode lainnya.

5.3.5. Evaluasi Eksternal

Evaluasi eksternal dilakukan menggunakan dua metrik utama, yaitu *Adjusted Rand Index (ARI)* dan *Purity Score*. Berdasarkan Hasil evaluasi menunjukkan bahwa model *K-Means (library)* dan *Hierarchical Clustering* dengan *Complete Linkage Euclidean* sama-sama memberikan hasil terbaik dalam hal kesesuaian terhadap label sebenarnya yaitu *segmentation*. Namun, versi *Hierarchical Complete Euclidean* yang dibuat secara manual menunjukkan performa yang paling stabil dan konsisten di seluruh metrik evaluasi, sehingga dianggap sebagai model terbaik untuk tugas ini.

5.4 Analisis Performa

Analisis performa dilakukan untuk membandingkan efektivitas model-model *clustering* yang telah diimplementasikan, baik secara internal maupun eksternal. Pengukuran dilakukan menggunakan metrik-metrik seperti *Silhouette Score*, *Davies-Bouldin Index (DBI)*, *Calinski-Harabasz Index (CHI)*, *Rand Index*, dan *Purity Score*. Selain itu, hasil dari implementasi manual dan *library* juga dibandingkan untuk menilai keandalan masing-masing.

Model *K-Means* dari *library* memberikan hasil terbaik secara keseluruhan di antara metode *K-Means*. Model ini memiliki *Silhouette Score* dan CHI yang tinggi, serta DBI yang rendah, yang

mengindikasikan pembentukan kluster yang kompak dan terpisah. Nilai *Rand Index* dan *Purity Score* juga relatif tinggi, menunjukkan kecocokan dengan label *ground truth*.

Di sisi lain, pada model *Hierarchical Clustering*, metode terbaik diperoleh dari *Complete Linkage* dengan *Euclidean Distance*, baik dalam versi *library* maupun manual. Model ini unggul dalam *Silhouette Score*, *Calinski-Harabasz*, dan memiliki nilai *Rand Score* serta *Purity* tertinggi. Ini menunjukkan bahwa metode ini mampu menangkap struktur kluster yang paling mendekati label asli segmentasi pelanggan.

Namun, metode *Single Linkage*, baik dengan *Euclidean* maupun *Manhattan*, menghasilkan *Silhouette Score* negatif atau rendah, serta CHI yang sangat kecil. Ini mengindikasikan pembentukan kluster yang buruk dan kurang terpisah. Dengan demikian, analisis performa mendukung bahwa pemilihan metode clustering harus mempertimbangkan baik efektivitas pemisahan kluster maupun kesesuaian dengan tujuan segmentasi aktual.

5.5 Analisis Karakteristik Kluster

Analisis karakteristik kluster merupakan tahap penting dalam memahami segmentasi data secara lebih mendalam. Melalui pendekatan ini, masing-masing kluster dapat diinterpretasikan berdasarkan variabel demografis, sosial, dan perilaku konsumsi yang dominan. Tujuannya adalah untuk mengidentifikasi pola umum yang membedakan satu kelompok dengan kelompok lainnya, sehingga dapat digunakan sebagai dasar dalam pengambilan keputusan yang lebih tepat sasaran, baik dalam konteks bisnis maupun penelitian lanjutan.

Cluster	Julukan	Umur Rata-rata	<i>Spending Score</i>	Dominan Profesi
0	<i>Silent Spenders</i>	47	Sangat rendah	<i>Artist, Entertainment</i>
1	<i>Mature Big Spenders</i>	66	Sangat tinggi	<i>Lawyer, Executive</i>
2	<i>Young Essentials</i>	28	Sangat rendah	<i>Healthcare</i>
3	<i>Creative Families</i>	45	Sedang-tinggi	<i>Artist, Executive</i>

1. **Cluster 0** terdiri dari individu berusia rata-rata 47 tahun, sebagian besar telah menikah (53%) dan berpendidikan tinggi (73% lulusan). Rata-rata pengalaman kerja sebesar 2,75 tahun menunjukkan tingkat kematangan profesional. Namun, skor pengeluaran sangat rendah (0,07), dengan ukuran keluarga kecil (1–2 anggota) dan proporsi gender seimbang. Profesi dominan adalah seniman (42%), diikuti pekerja hiburan (15%) dan pengacara (11%).
2. **Cluster 1** merupakan kelompok dengan rata-rata usia tertinggi (66 tahun) dan seluruh anggotanya telah menikah (100%). Sebanyak 70% merupakan lulusan perguruan tinggi, namun memiliki pengalaman kerja rendah (0,8 tahun), yang mengindikasikan status pensiun. Skor pengeluaran sangat tinggi (1,65), dengan ukuran keluarga sedang (2–3 anggota) dan proporsi pria sedikit lebih tinggi (55%). Profesi dominan meliputi pengacara (34%), seniman (31%), dan eksekutif (20%).

3. **Cluster 2** terdiri dari individu berusia paling muda (rata-rata 28 tahun), dengan mayoritas belum menikah (87%) dan tingkat pendidikan rendah (33% lulusan). Pengalaman kerja rata-rata sebesar 2,45 tahun. Skor pengeluaran sangat rendah (0,02), meskipun ukuran keluarga tergolong besar (rata-rata 3,8 anggota). Distribusi gender seimbang, dengan dominasi profesi di bidang kesehatan (48%) dan seni (11%).
4. **Cluster 3** mencakup individu berusia rata-rata 45 tahun, hampir seluruhnya telah menikah (98%) dan berpendidikan tinggi (72% lulusan). Pengalaman kerja sebesar 1,65 tahun, dengan skor pengeluaran menengah ke tinggi (1,03) dan ukuran keluarga besar (rata-rata 3,4 anggota). Mayoritas berjenis kelamin pria (66%), dengan profesi dominan seniman (43%), pekerja hiburan (14%), dan eksekutif (14%).

BAB VI

KESIMPULAN & SARAN

6.1 Kesimpulan

Berdasarkan serangkaian tahapan eksplorasi, *preprocessing*, dan penerapan metode clustering yang telah dilakukan, dapat disimpulkan bahwa proses segmentasi data berhasil dilakukan dengan pendekatan yang sistematis dan mempertimbangkan karakteristik unik dari masing-masing fitur. Proses dimulai dari pembersihan data, penanganan missing value, transformasi fitur numerik, serta encoding fitur kategorikal. Tahapan-tahapan tersebut terbukti penting untuk menghasilkan representasi data yang siap digunakan dalam pemodelan.

Penerapan algoritma clustering seperti *K-Means* dan *Hierarchical Clustering* menunjukkan hasil yang cukup informatif, terutama dalam mengelompokkan data berdasarkan pola-pola yang tersembunyi. Meskipun tidak semua hasil *clustering* dapat langsung ditafsirkan secara praktis, analisis terhadap karakteristik masing-masing cluster memberikan wawasan awal yang berguna untuk pengambilan keputusan lebih lanjut.

Secara keseluruhan, implementasi dan analisis ini menegaskan pentingnya proses data preparation sebelum masuk ke tahap modeling. Selain itu, hasil yang diperoleh juga menunjukkan potensi clustering sebagai metode yang efektif dalam mengenali pola dan segmentasi dalam data, terutama jika dikombinasikan dengan evaluasi metrik yang sesuai seperti silhouette score atau visualisasi distribusi cluster.

6.2 Saran

Evaluasi penggunaan metode *clustering* lain seperti *DBSCAN* atau *Gaussian Mixture Model (GMM)* dapat dilakukan untuk mengetahui apakah terdapat pendekatan lain yang mampu menghasilkan segmentasi yang lebih baik, khususnya dalam menangani data dengan distribusi tidak merata atau adanya outlier.

Gunakan teknik validasi eksternal yang lebih luas, seperti *mutual information score* atau *normalized mutual information (NMI)*, untuk melengkapi evaluasi eksternal dan memperkuat keandalan hasil clustering terhadap *ground truth*.

Terapkan segmentasi hasil clustering pada strategi bisnis nyata, misalnya dalam personalisasi promosi, pengelompokan loyalitas pelanggan, atau rekomendasi produk. Validasi terhadap efektivitas segmentasi ini dalam praktik akan memperkuat manfaat penelitian secara langsung.

Perluasan fitur dan data *real-time* dari transaksi pelanggan atau perilaku digital dapat meningkatkan kedalaman segmentasi. Penambahan fitur seperti waktu transaksi, frekuensi kunjungan, dan lokasi geografis berpotensi menghasilkan klaster yang lebih relevan.

Dengan memperhatikan saran-saran di atas, diharapkan penelitian selanjutnya dapat mengembangkan model segmentasi pelanggan yang lebih adaptif, akurat, dan berdampak langsung pada strategi pengambilan keputusan berbasis data.

STUDI PUSTAKA

Perdana, S. A., Florentin, S. F., & Santoso, A. (2022). Analisis segmentasi pelanggan menggunakan K-Means Clustering studi kasus aplikasi Alfagift. *Sebatik*, 26(2). <https://jurnal.wicida.ac.id/index.php/sebatik/article/view/1991>

Setiyawan, A., & Rofifudin, F. M. (2024). Urgensi segmentasi pasar bagi perusahaan: Literature review. *JIEMBI: Jurnal Ilmu Ekonomi, Manajemen dan Bisnis*, 2(2), 70–72.

Lakshman Narayana, V., Sirisha, S., Divya, G., Pooja, N. L. S., & Nouf, S. A. (2022). Mall customer segmentation using machine learning. In *Proceedings of the International Conference on Electronics and Renewable Systems (ICEARS 2022)*. IEEE. <https://doi.org/10.1109/ICEARS53579.2022.9752447>

Aulia, S. (2024). Review tantangan dan aplikasi algoritma K-Means dalam data mining. *Djtechno: Jurnal Teknologi Informasi*, 5(2), 298–303. <https://jurnal.dharmawangsa.ac.id/index.php/djtechno>

Izzuddin, A. (2021). Optimasi cluster pada algoritma K-Means dengan reduksi dimensi dataset menggunakan Principal Component Analysis untuk pemetaan kinerja dosen. *JIEET: Journal Information Engineering and Educational Technology*, 5(2), 51–53.

Saraswat, P., & Raj, S. (2021). A brief review on machine learning and its various techniques. *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, 9(6), 110–113. <https://doi.org/10.55524/ijircst.2021.9.6.25>

Saligkaras, D., & Papageorgiou, V. E. (2022). Seeking the truth beyond the data: An unsupervised machine learning approach. *Journal of Physics: Conference Series*, 2382(1), 012043. <https://doi.org/10.1088/1742-6596/2382/1/012043>

Nascimento, M. L. F. (2022). In search of star clusters: An introduction to the K-means algorithm. *Journal of Humanistic Mathematics*, 12(1), 243–255. <https://doi.org/10.5642/jhummath.202201.19>

Modalavalasa, H. K., & Makkena, M. L. (2020). An experimental review on effect of Principal Component Analysis on machine learning techniques. *International Journal of Engineering Applied Sciences and Technology*, 5(1), 290–296. <http://www.ijeast.com>

Abbas, Q., Khalid, R., Hussain, M., & Sajjad, M. (2021). Incorporating K-means, hierarchical clustering and PCA in customer segmentation. *Journal of King Saud University - Computer and Information Sciences*. <https://doi.org/10.1016/j.jksuci.2021.03.004>

Almeida, J., Braga, A., & Pereira, R. (2021). Data preprocessing impact on clustering algorithms: A review. *Information*, 12(8), 317. <https://doi.org/10.3390/info12080317>

Rousseeuw, P. J., & Hubert, M. (2011). Robust statistics for outlier detection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1), 73–79. <https://doi.org/10.1002/widm.2>

Abdurrahman, G. (2019). Clustering Data Kredit Bank Menggunakan Algoritma Agglomerative Hierarchical Clustering Average Linkage. *JUSTINDO (Jurnal Sistem & Teknologi Informasi Indonesia)*, 4(1), 13–20.

Kaur, H., & Singh, D. (2021). Importance of data type conversion and data integrity in data preprocessing. *Journal of Emerging Technologies and Innovative Research*, 8(3), 45–49.

Sharma, S., & Mirajkar, R. (2020). Efficient stratified sampling techniques for big data

analytics. *Procedia Computer Science*, 171, 1272–1279. <https://doi.org/10.1016/j.procs.2020.04.136>