DAFTAR PUSTAKA

Azuri, D. F., Zulhanif, & Pontoh, R. S. (2016). Pengelompokkan Kabupaten/Kota Di Pulau Jawa Berdasarkan Pembangunan ManUmur Berbasis Gender Menggunakan Bisecting K-means. Universitas Padjajaran, 110(25), 78–83.

Basari, A. S. H., Hussin, B., Ananta, I. G. P., & Zeniarja, J. (2013). Opinion mining of movie review using hybrid method of support vector machine and particle swarm optimization. Procedia Engineering, 53, 453-462. https://doi.org/10.1016/j.proeng.2013.02.059

Bhatia, Parteek. 2019. "Data Mining and Data Warehousing Principles and Practical Techniques". New York: Cambridge University. https://doi.org/10.1017/9781108635592

Chen, Liang-Hua., Liao, Hong-Yuan., Wang, Jiing-Yuh., dan Fan, Kuo-Chin. (1999). Automatic Data Capture for Geographic Information Systems. IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, Vol.29, No.2, May 1999. https://doi.org/10.1109/5326.760565

Dinata, R. K., Safwandi, S., Hasdyna, N., & Azizah, N. (2020). Analisis K-means clustering pada data sepeda motor. INFORMAL: Informatics Journal, 5(1), 10-17. https://doi.org/10.19184/isj.v5i1.17071.

Fayyad, Piatetsky-Shapiro, Smyth. (1996). "Data Mining to Knowledge Discovery: an Overview, Advances in Knowldege Discovery and Data Mining". AI magazine, 17(3), 37-37.

Fu, Pinde, Sun, Jiulin. (2011). Webgis Principles and Aplications. California:Esri Press

Han, J., Kamber, M., & Pei, J. (2011). Data Mining Concepts and Techniques, 3rd Edition (The Morgan Kaufmann Series in Data Management Systems).

Hendrawan, A. Y. (2020). Peningkatan Kinerja Algoritma K Means Dengan Menggunakan Particle Swarm Optimization Dalam Pengelompokan Data Penyediaan Akses. Electro Luceat, 6(2), 213-227. https://doi.org/10.32531/jelekn.v6i2.245

Hulu, V. T., Salman, S., Supinganto, A., Amalia, L., Khariri, K., Sianturi, E., & Syamdarniati, S. (2020). Epidemiologi Penyakit Menular: Riwayat, Penularan dan Pencegahan. Yayasan Kita Menulis.

Kemenkes RI. (2014). Pedoman Nasional Pengendalian Tuberkulosis. Jakarta: Kementerian Kesehatan RI.

Kemenkes RI. (2018). Infodatin Tuberkulosis (TB). Pusat Data dan Informasi Kementerian Kesehatan RI. ISSN, 2442-7659.

Kemenkes RI. (2006). Pedoman Pengendalian Demam Tifoid:Keputusan Menteri Kesehatan Republik Indonesia Nomor 364/MENKES/SK/V/2006 tanggal 19 Mei 2006. LKBN Antara.

Kemenkes RI. (2020). Data Kasus Terbaru DBD di Indonesia.

Kemenkes RI. (2011). Modul pengendalian demam berdarah dengue.

Kotu, V. & Desphande, B., (2015). Predictive Analytics and Data Mining. Massachussetts: Elsevier Inc. https://doi.org/10.1016/B978-0-12-801460-8.00002-1

Luthfi.(2012). Tuberkulosis Nosokomial, Jurnal Tuberkulosis Indonesia, 8 : 30-31.

Ordila, R., Wahyuni, R., Irawan, Y., & Sari, M. Y. (2020). Penerapan Data Mining Untuk Pengelompokan Data Rekam Medis Pasien Berdasarkan Jenis Penyakit Dengan Algoritma Clustering (Studi Kasus: Poli Klinik Pt. Inecda). Jurnal Ilmu Komputer, 9(2), 148-153. https://doi.org/10.33060/JIK/2020/Vol9.Iss2.181

Peraturan Menteri Kesehatan No. 82 Tahun 2014 tentang Penanggulangan Penyakit Menular

Permatadevi, M. A., Hendrawan, R. A., & Hafidz, I., (2013). Karakteristik Pelanggan Telepon Kabel Menggunakan Clustering SOM dan K-means untuk Mengurangi Kesalahan Klasifikasi Pelanggan Perusahaan Telekomunikasi. Jurnal Teknik Pomits Vol. 1, No. 1 Hal. 1-6

Purwadhi, Sri Hardiyanti. (1994). Penelitian lingkungan geografis dalam inventarisasi penggunaan lahan dengan teknik penginderaan jauh di Indonesia. Yogyakarta: Forum diskusi mahasiswa Fakultas Geografi Universitas Gadjah Mada

Queensland Health. (2017). Tuberculosis fact sheet Indonesian Last updated 20/01/17. Department of Health

Thinsungnoena, T., Kaoungkub, N., Durongdumronchaib, P., Kerdprasopb, K., & Kerdprasopb, N. (2015). The clustering validity with silhouette and sum of squared errors. learning, 3(7). https://doi.org/10.12792/iciae2015.012

Turban, E., Delen, D., & Sharda, R. (2018). Business intelligence, analytics, and data science: A managerial perspective. Harlow ; Munich: Pearson Prentice Hall.

Wang, S. Liu, Y., Wang, G., Chen, H., & Dong, H., Zhu, X., (2011). "An Improved Particle Swarm Optimization for Feature Selection," Journal of Bionic Engineering, pp. 8(2), 191–200. doi:10.1016/S1672- 6529(11)60020-6. https://doi.org/10.1016/S1672-6529(11)60020-6

WHO. (1997). World Health Organization | Dengue Haemorrhagic Fever Diagnosis, Treatment, Prevention and Control Second Edition. Geneva; 1997. p.1–77.

WHO. (2022). World Health Organization WHO | Global Tuberculosis Report 2022.

Widoyono. (2011). Penyakit Tropis: Epidemiologi, Penularan, Pencegahan dan Pemberantasannya. Jakarta: PT.Gelora Aksara Pratama.

WHO. (2009). World Health Organization | Dengue Guideline For Diagnosis, Treathment, Prevention and Control. Hal.3-4:14-6:25-8:33-41.

Wu, Xindong and Vipin Kumar. (2009). The Top Ten Algorithms in Data Mining. Boca Raton: Chapman & Hall/CRC. https://doi.org/10.1201/9781420089653

Zhelu. (2009). A web-based Geographical Information Sistem prototype on Portuguese traditional food products. Portugal: University of wilhelms.

LAMPIRAN

Lampiran 1. Source Code

```
import pandas as pd
import numpy as np
import random as rd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans as Kmeans1
from sklearn.metrics import silhouette_samples,
silhouette_score
from sklearn.metrics import davies_bouldin_score

from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.preprocessing import Normalizer
from sklearn.linear_model import LinearRegression, Ridge,
Lasso

from adjustText import adjust_text
import random

import geopy
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
from datetime import datetime
from tqdm import tqdm_notebook

import warnings
warnings.filterwarnings('ignore')


data = pd.read_csv('data.csv')
data.head()


# rename column

data = data.copy()
data = data.rename(columns={'Gejala apa yang dirasakan?': 'Gejala',
                            'Jika pasien mengalami demam, apakah demam
yang dialami terus menerus sepanjang hari?': 'Demam Panjang',
                            'Apakah pasien mual/muntah?': 'Mual/Muntah',
                            'Apakah pasien pernah mengalami mimisan?':
'Riwayat Mimisan',
                            'Berapa suhu badan pasien?': 'Suhu Badan',
                            'Apakah sudah menerima imunisasi BCG?':
'Riwayat Vaksin BCG',
                            'Apakah ada orang serumah yang pernah sakit atau
diopname?': 'Riwayat Penyakit Orang Serumah',
                            'Jika iya, karena apa?': 'Daftar Penyakit Orang
Serumah',
                            'Apakah nafsu makan berkurang?': 'Nafsu
Makan',
```

```
                                        'Apakah sering jajan diluar rumah?': 'Kebiasaan
Jajan',
                                        'Berapa luas rumah pasien?': 'Luas Rumah',
                                        'Berapa jumlah penghuni rumah pasien?': 'Jumlah
Penghuni Rumah',
                                        'Bagaimana kondisi pembuangan sampah disekitar
rumah?': 'Kondisi Pembuangan Sampah',
                                        'Apakah tersedia air bersih di rumah?':
'Ketersediaan Air Bersih',
                                        'Bagaimana sistem ventilasi di rumah pasien?':
'Sistem Ventilasi Rumah',
                                        'Apakah sering menggunakan jamban/toilet?':
'Riwayat Pemakaian Jamban/Toilet',
                                        'Apa diagnosa/hasil tes darah pasien?':
'Diagnosa/Tes Darah'})


features_data = data[["Umur",
                      "Alamat",
                      "Nafsu Makan",
                      "Kebiasaan Jajan",
                      "Kondisi Pembuangan Sampah",
                      "Ketersediaan Air Bersih",
                     "Sistem Ventilasi Rumah",
                     "Riwayat Pemakaian Jamban/Toilet",
                     "Diagnosa/Tes Darah"]].copy()

#check missing value
features_data.isnull().sum()


features_data.info()


features_data['Nafsu Makan']=features_data['Nafsu
Makan'].str.split(',\s*')
features_data['Kebiasaan Jajan']=features_data['Kebiasaan
Jajan'].str.split(',\s*')
features_data['Kondisi Pembuangan Sampah']=features_data['Kondisi
Pembuangan Sampah'].str.split(',\s*')
features_data['Ketersediaan Air Bersih']=features_data['Ketersediaan Air
Bersih'].str.split(',\s*')
features_data['Sistem Ventilasi Rumah']=features_data['Sistem Ventilasi
Rumah'].str.split(',\s*')
features_data['Riwayat Pemakaian Jamban/Toilet']=features_data['Riwayat
Pemakaian Jamban/Toilet'].str.split(',\s*')
features_data['Diagnosa/Tes Darah']=features_data['Diagnosa/Tes
Darah'].str.split(',\s*')


mlb = MultiLabelBinarizer()

mlb.fit(features_data['Nafsu Makan'])
```

```
mlb.fit(features_data['Kebiasaan Jajan'])
mlb.fit(features_data['Kondisi Pembuangan Sampah'])
mlb.fit(features_data['Ketersediaan Air Bersih'])
mlb.fit(features_data['Sistem Ventilasi Rumah'])
mlb.fit(features_data['Riwayat Pemakaian Jamban/Toilet'])
mlb.fit(features_data['Diagnosa/Tes Darah'])

MultiLabelBinarizer()

mlb.classes_

features_data['Nafsu Makan'].explode().unique()
features_data['Kebiasaan Jajan'].explode().unique()
features_data['Ketersediaan Air Bersih'].explode().unique()
features_data['Kondisi Pembuangan Sampah'].explode().unique()
features_data['Sistem Ventilasi Rumah'].explode().unique()
features_data['Riwayat Pemakaian Jamban/Toilet'].explode().unique()
features_data['Diagnosa/Tes Darah'].explode().unique()

mlb.transform(features_data['Nafsu Makan'])
mlb.transform(features_data['Kebiasaan Jajan'])
mlb.transform(features_data['Ketersediaan Air Bersih'])
mlb.transform(features_data['Kondisi Pembuangan Sampah'])
mlb.transform(features_data['Sistem Ventilasi Rumah'])
mlb.transform(features_data['Riwayat Pemakaian Jamban/Toilet'])
mlb.transform(features_data['Diagnosa/Tes Darah'])

a1 = pd.DataFrame(mlb.fit_transform(features_data['Nafsu Makan']),
columns=mlb.classes_)
a2 = pd.DataFrame(mlb.fit_transform(features_data['Kebiasaan
Jajan']), columns=mlb.classes_)
a3 = pd.DataFrame(mlb.fit_transform(features_data['Ketersediaan Air
Bersih']), columns=mlb.classes_)
a4 = pd.DataFrame(mlb.fit_transform(features_data['Kondisi
Pembuangan Sampah']), columns=mlb.classes_)
a5 = pd.DataFrame(mlb.fit_transform(features_data['Sistem Ventilasi
Rumah']), columns=mlb.classes_)
a6 = pd.DataFrame(mlb.fit_transform(features_data['Riwayat
Pemakaian Jamban/Toilet']), columns=mlb.classes_)
a7 = pd.DataFrame(mlb.fit_transform(features_data['Diagnosa/Tes
Darah']), columns=mlb.classes_)

# concat data column after transformation
data_features=pd.concat([features_data,a1, a2, a3, a4, a5, a6,
a7], axis='columns')


df = data_features.drop(['Nafsu Makan','Nafsu Makan Normal','Jarang Jajan
Diluar','Tidak tersedia air bersih','Jarang pakai jamban/toilet','Kondisi Pembuangan
Sampah', 'Kebiasaan Jajan', 'Ketersediaan Air Bersih', 'Sistem Ventilasi Rumah',
'Riwayat Pemakaian Jamban/Toilet', 'Diagnosa/Tes Darah'], axis=1)

##Data TB
```

```
data1 = df.copy()
jumlah_tb = data1.loc[(data1['tbc'] == 1)].copy()
jumlah_tb.rename(columns={'Umur': 'Umur_TB'}, inplace=True)
dict_cat = {}
dict_num = {}

for cat in jumlah_tb.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah_tb.select_dtypes(['int64', 'float64']):
    if((num == 'tbc') or (num == 'dbd') or (num == 'tifoid')):
        continue
    dict_num[num] = ['mean']

data_perdesa = jumlah_tb.groupby('Alamat').agg({
    'tbc': 'count',
    **dict_num,
    **dict_cat
})

data_perdesa = data_perdesa[['tbc', 'Umur_TB']]


##Data DBD
data1 = df.copy()
jumlah_dbd = data1.loc[(data1['dbd'] == 1)].copy()
jumlah_dbd.rename(columns={'Umur': 'Umur_DBD'}, inplace=True)
dict_cat = {}
dict_num = {}

for cat in jumlah_dbd.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah_dbd.select_dtypes(['int64', 'float64']):
    if((num == 'tbc') or (num == 'dbd') or (num == 'tifoid')):
        continue
    dict_num[num] = ['mean']

data_perdesa1 = jumlah_dbd.groupby('Alamat').agg({
    'dbd': 'count',
    **dict_num,
    **dict_cat
})

data_perdesa1 = data_perdesa1[['dbd', 'Umur_DBD']]


##Data Tifoid
data1 = df.copy()
jumlah_tifoid = data1.loc[(data1['tifoid'] == 1)].copy()
jumlah_tifoid.rename(columns={'Umur': 'Umur_Tifoid'}, inplace=True)
dict_cat = {}
```

```
dict_num = {}

for cat in jumlah_tifoid.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah_tifoid.select_dtypes(['int64', 'float64']):
    if((num == 'tbc') or (num == 'dbd') or (num == 'tifoid')):
        continue
    dict_num[num] = ['mean']

data_perdesa2 = jumlah_tifoid.groupby('Alamat').agg({
    'tifoid': 'count',
    **dict_num,
    **dict_cat
})

data_perdesa2 = data_perdesa2[['tifoid', 'Umur_Tifoid']]

#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Nafsu Makan Kurang'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa3 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa3 = data_perdesa3[['Nafsu Makan Kurang']]
```

In [20]:

```
#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Sering Jajan Diluar'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']
```

```
data_perdesa4 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa4 = data_perdesa4[['Sering Jajan Diluar']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Tersedia air bersih'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa5 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa5 = data_perdesa5[['Tersedia air bersih']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Ada ventilasi tapi selalu tertutup'] ==
1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa14 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa14 = data_perdesa14[['Ada ventilasi tapi selalu tertutup']]


#Data Etc
data1 = df.copy()
```

```
jumlah = data1.loc[(data1['Ada ventilasi tapi jarang terbuka'] ==
1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa13 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa13 = data_perdesa13[['Ada ventilasi tapi jarang terbuka']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Ada ventilasi tapi hanya sedikit'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa12 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa12 = data_perdesa12[['Ada ventilasi tapi hanya sedikit']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Bersih tanpa ada sampah'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']
```

```
data_perdesa11 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa11 = data_perdesa11[['Bersih tanpa ada sampah']]

#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Ada sampah tapi tidak banyak'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa10 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa10 = data_perdesa10[['Ada sampah tapi tidak banyak']]

#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Ada sampah dan jarang dibersihkan'] ==
1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa9 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa9 = data_perdesa9[['Ada sampah dan jarang dibersihkan']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Ada lalat/nyamuk di sekitar tumpukan sampah'] ==
1)].copy()
dict_cat = {}
```

```
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa6 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa6 = data_perdesa6[['Ada lalat/nyamuk di sekitar tumpukan
sampah']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Ada ventilasi disemua ruangan'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):
    dict_num[num] = ['count']

data_perdesa7 = jumlah.groupby('Alamat').agg({
    **dict_num,
    **dict_cat
})

data_perdesa7 = data_perdesa7[['Ada ventilasi disemua ruangan']]


#Data Etc
data1 = df.copy()
jumlah = data1.loc[(data1['Sering Pakai Jamban/Toilet'] == 1)].copy()
dict_cat = {}
dict_num = {}

for cat in jumlah.select_dtypes(['object']) :
    if (cat == 'Alamat'):
        continue
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in jumlah.select_dtypes(['int64', 'float64']):

    dict_num[num] = ['count']

data_perdesa8 = jumlah.groupby('Alamat').agg({
```

```
    **dict_num,
    **dict_cat
})

data_perdesa8 = data_perdesa8[['Sering Pakai Jamban/Toilet']]


##merge 4 column data
merge_data = pd.concat([data_perdesa,
                        data_perdesa1,
                        data_perdesa2,
                        data_perdesa3,
                        data_perdesa4,
                        data_perdesa5,
                        data_perdesa6,
                        data_perdesa7,
                        data_perdesa8,
                        data_perdesa9,
                        data_perdesa10,
                        data_perdesa11,
                        data_perdesa12,
                        data_perdesa13,
                        data_perdesa14], axis='columns')

#remove nan data
merge_data  = merge_data .replace(np.NaN, 0)


##fix named columns
merge_data.columns =
merge_data.columns.map(".join).str.strip('|')
merge_data.columns = [
    c.replace('mean', ").replace('count', ") for c in
merge_data.columns
]

#convert to float columns int
merge_data['Sering Pakai Jamban/Toilet'] = merge_data['Sering Pakai
Jamban/Toilet'].astype(float)


merge_data.index.name = None

#COUNT LOGITUDE AND LATITUDE FOR ALL DESA
alamat = []
for i in merge_data.index:
  splitted = i.split('(')
  alamat.append(splitted[0])
geolocator = Nominatim(user_agent="tes")
coord = []
for i in range(0, len(alamat)):
  loc = alamat[i]
  location = geolocator.geocode(loc, timeout=None)
  if location != None:
    m = 0.0025
```

```
    coord.append([[location.latitude + m, location.longitude -
m],
                  [location.latitude - m, location.longitude +
m],
                  [location.latitude + m, location.longitude],
                  [location.latitude, location.longitude - m],
                  [location.latitude - m, location.longitude],
                  [location.latitude, location.longitude +
m]])
    print(loc)
coord1 = [i[0] for i in coord]
coord2 = [i[1] for i in coord]
coord3 = [i[2] for i in coord]
coord4 = [i[3] for i in coord]
coord5 = [i[4] for i in coord]
coord6 = [i[5] for i in coord]
```

Kmeans from Scratch

```
class KMeans:
    def __init__(
            self,
            n_cluster: int,
            init_pp: bool = True,
            max_iter: int = 300,
            tolerance: float = 1e-4,
            seed: int = None):
        self.n_cluster = n_cluster
        self.max_iter = max_iter
        self.tolerance = tolerance
        self.init_pp = init_pp
        self.seed = seed
        self.centroid = None
        self.SSE = None

    def fit(self, data: np.ndarray):
        self.centroid = self._init_centroid(data)
        for _ in range(self.max_iter):
            distance = self._calc_distance(data)
            cluster = self._assign_cluster(distance)
            new_centroid = self._update_centroid(data,
cluster)
            diff = np.abs(self.centroid - new_centroid).mean()
            self.centroid = new_centroid

            if diff <= self.tolerance:
                break

        self.SSE = self._calc_sse(self.centroid, cluster,
data)

    def predict(self, data: np.ndarray):
        distance = self._calc_distance(data)
        cluster = self._assign_cluster(distance)
        return cluster
```

```
    def _init_centroid(self, data: np.ndarray):
        if self.init_pp:
            np.random.seed(self.seed)
            centroid = [int(np.random.uniform() * len(data))]
            while len(centroid) < self.n_cluster:
                dist = np.array([
                    min([np.inner(data[c] - x, data[c] - x)
for c in centroid])
                    for x in data
                ])
                dist /= dist.sum()
                cumdist = np.cumsum(dist)
                prob = np.random.rand()
                for i, c in enumerate(cumdist):
                    if prob < c and i not in centroid:
                        centroid.append(i)
                        break
            centroid = np.array([data[c] for c in centroid])
        else:
            np.random.seed(self.seed)
            idx = np.random.choice(range(len(data)),
size=(self.n_cluster))
            centroid = data[idx]
        return centroid

    def _calc_distance(self, data: np.ndarray):
        distances = np.zeros((data.shape[0], self.n_cluster))
        for i, c in enumerate(self.centroid):
            distance = np.sum((data - c) ** 2, axis=1)
            distances[:, i] = distance
        return distances


    def _assign_cluster(self, distance: np.ndarray):
        cluster = np.argmin(distance, axis=1)
        return cluster

    def _update_centroid(self, data: np.ndarray, cluster:
np.ndarray):
        centroids = []
        for i in range(self.n_cluster):
            idx = np.where(cluster == i)
            centroid = np.mean(data[idx], axis=0)
            centroids.append(centroid)
        centroids = np.array(centroids)
        return centroids

    def _calc_sse(self, centroids, cluster, data):
        sse = 0
        for i in range(self.n_cluster):
            idx = np.where(cluster == i)
            sse += np.sum((data[idx] - centroids[i]) ** 2)
        return sse
```

SSE

```
def calc_sse(centroids: np.ndarray, labels: np.ndarray, data:
np.ndarray):
    distances = 0
    for i, c in enumerate(centroids):
        idx = np.where(labels == i)
        dist = np.sum((data[idx] - c)**2)
        distances += dist
    return distances
```

DBI

```
def calc_dbi(self, data, cluster):
    cluster_centers = self.centroid
    cluster_indices = np.unique(cluster)
    num_clusters = len(cluster_indices)
    intra_cluster_distances = np.zeros(num_clusters)
    inter_cluster_distances = np.zeros((num_clusters,
num_clusters))

    for i in range(num_clusters):
        indices = np.where(cluster == i)
        cluster_data = data[indices]
        centroid = cluster_centers[i]
        intra_cluster_distances[i] =
np.mean(np.linalg.norm(cluster_data - centroid, axis=1))

    for i in range(num_clusters):
        for j in range(num_clusters):
            if i != j:
                dist = np.linalg.norm(cluster_centers[i] -
cluster_centers[j])
                inter_cluster_distances[i, j] = dist

    dbi = 0.0
    for i in range(num_clusters):
        max_value = -np.inf
        for j in range(num_clusters):
            if i != j:
                value = (intra_cluster_distances[i] +
intra_cluster_distances[j]) / inter_cluster_distances[i, j]
                if value > max_value:
                    max_value = value
        dbi += max_value

    dbi /= num_clusters

    return dbi
```

Quantization Error

```
def quantization_error(centroids: np.ndarray, labels:
np.ndarray, data: np.ndarray) -> float:
    error = 0.0
    for i, c in enumerate(centroids):
        idx = np.where(labels == i)
        dist = np.linalg.norm(data[idx] - c)
        dist /= len(idx)
        error += dist
    error /= len(centroids)
    return error
```

Particle for PSO

```
class Particle:
    def __init__(self,
                 n_cluster: int,
                 data: np.ndarray,
                 use_kmeans: bool = False,
                 w: float = 0.9,
                 c1: float = 0.5,
                 c2: float = 0.3):
        index = np.random.choice(list(range(len(data))),
n_cluster)
        self.centroids = data[index].copy()
        if use_kmeans:
            kmeans = KMeans(n_cluster=n_cluster,
init_pp=False)
            kmeans.fit(data)
            self.centroids = kmeans.centroid.copy()
        self.best_position = self.centroids.copy()
        self.best_score = quantization_error(self.centroids,
self._predict(data), data)
        self.best_sse = calc_sse(self.centroids,
self._predict(data), data)
        self.velocity = np.zeros_like(self.centroids)
        self._w = w
        self._c1 = c1
        self._c2 = c2
        self.r1 = 0  # tambahkana variabel r1
        self.r2 = 0  # tambahkan variabel r2

    def update(self, gbest_position: np.ndarray, data:
np.ndarray):
        self._update_velocity(gbest_position)
        self._update_centroids(data)

    def _update_velocity(self, gbest_position: np.ndarray):
        v_old = self._w * self.velocity
        self.r1 = np.random.random()
        self.r2 = np.random.random()
        cognitive_component = self._c1 * self.r1 *
(self.best_position - self.centroids)
        social_component = self._c2 * self.r2 *
(gbest_position - self.centroids)
```

```
        self.velocity = v_old + cognitive_component +
social_component

    def _update_centroids(self, data: np.ndarray):
        self.centroids = self.centroids + self.velocity
        new_score = quantization_error(self.centroids,
self._predict(data), data)
        sse = calc_sse(self.centroids, self._predict(data),
data)
        self.best_sse = min(sse, self.best_sse)
        if new_score < self.best_score:
            self.best_score = new_score
            self.best_position = self.centroids.copy()

    def _predict(self, data: np.ndarray) -> np.ndarray:
        distance = self._calc_distance(data)
        cluster = self._assign_cluster(distance)
        return cluster

    def _calc_distance(self, data: np.ndarray) -> np.ndarray:
        distances = []
        for c in self.centroids:
            distance = np.sum((data - c) * (data - c), axis=1)
            distances.append(distance)

        distances = np.array(distances)
        distances = np.transpose(distances)
        return distances

    def _assign_cluster(self, distance: np.ndarray) ->
np.ndarray:
        cluster = np.argmin(distance, axis=1)
        return cluster

    def print_velocity(self):
        print(self.velocity)

    def print_r1(self):
        print(self.r1)

    def print_r2(self):
        print(self.r2)

    def print_best_position(self):
        print(self.best_position)
```

PSO from Scratch

```
class ParticleSwarmOptimizedClustering:
    def __init__(self,
                 n_cluster: int,
                 n_particles: int,
                 data: np.ndarray,
                 hybrid: bool = True,
                 max_iter: int = 2000,
```

```
                 print_debug: int = 10):
        self.n_cluster = n_cluster
        self.n_particles = n_particles
        self.data = data
        self.max_iter = max_iter
        self.particles = []
        self.hybrid = hybrid

        self.print_debug = print_debug
        self.gbest_score = np.inf
        self.gbest_centroids = None
        self.gbest_position = None
        self.gbest_sse = np.inf
        self._init_particles()

    def _init_particles(self):
        for i in range(self.n_particles):
            particle = None
            if i == 0 and self.hybrid:
                particle = Particle(self.n_cluster, self.data,
use_kmeans=True)
            else:
                particle = Particle(self.n_cluster, self.data,
use_kmeans=False)
            if particle.best_score < self.gbest_score:
                self.gbest_centroids =
particle.centroids.copy()
                self.gbest_score = particle.best_score
            self.particles.append(particle)
            self.gbest_sse = min(particle.best_sse,
self.gbest_sse)

    def run(self):
        print('Initial global best score', self.gbest_score)
        history = []
        for i in range(self.max_iter):
            for particle in self.particles:
                particle.update(self.gbest_centroids,
self.data)
                # print(i, particle.best_score,
self.gbest_score)
            for particle in self.particles:
                if particle.best_score < self.gbest_score:
                    self.gbest_centroids =
particle.centroids.copy()
                    self.gbest_score = particle.best_score
            history.append(self.gbest_score)
            if i % self.print_debug == 0:
                print('Iteration {:04d}/{:04d} current gbest score
{:.18f}'.format(
                    i + 1, self.max_iter, self.gbest_score))
        print('Finish with gbest score {:.18f}'.format(self.gbest_score))
        return history
```

```
test_data = merge_data.copy()
index_awal = test_data.index

# Normalisasi L2
normalizer = Normalizer(norm='l2')
X_normalized = normalizer.transform(test_data)

# Create a new DataFrame with the normalized features
X_normalized_df = pd.DataFrame(X_normalized,
columns=test_data.columns, index=index_awal)
X_normalized_df[:10]


data = merge_data.copy()


selected_columns = ['tbc', 'Umur_TB', 'Tersedia air bersih', 'Ada lalat/nyamuk
di sekitar tumpukan sampah', 'Ada sampah dan jarang dibersihkan',
                    'Ada sampah tapi tidak banyak', 'Bersih tanpa ada sampah',
'Ada ventilasi tapi hanya sedikit', 'Ada ventilasi tapi jarang terbuka',
                    'Ada ventilasi tapi selalu tertutup', 'Ada ventilasi disemua
ruangan']
data_selected = data[selected_columns]


# Normalisasi L2
normalizer = Normalizer(norm='l2')
data_normalized = normalizer.transform(data_selected)


# Data yang telah dinormalisasi
data_normalized = data_normalized.copy()

# Inisialisasi range nilai k
min_k = 2
max_k = 11

# List untuk menyimpan nilai SSE
sse_scores = []
# List untuk menyimpan nilai Silhouette Score
silhouette_scores = []

# Loop untuk mencari SSE dan Silhouette Score pada setiap
nilai k
for k in range(min_k, max_k+1):
    kmeans = Kmeans1(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(data_normalized)
    sse = kmeans.inertia_
    sse_scores.append(sse)
    if len(np.unique(cluster_labels)) > 1:
        silhouette_avg = silhouette_score(data_normalized,
cluster_labels)
        silhouette_scores.append(silhouette_avg)
    else:
```

```
            silhouette_scores.append(0)

    # Mencari nilai k optimal berdasarkan Silhouette Score
    optimal_k = min_k + np.argmax(silhouette_scores)

    # Plot grafik SSE dan Silhouette Score
    fig, ax1 = plt.subplots()

    ax1.plot(range(min_k, min_k + len(sse_scores)), sse_scores,
    marker='o')
    ax1.set_xlabel('Number of Clusters (k)')
    ax1.set_ylabel('SSE', color='blue')
    ax1.tick_params('y', colors='blue')

    ax2 = ax1.twinx()
    ax2.plot(range(min_k, min_k + len(silhouette_scores)),
    silhouette_scores, marker='o', color='red')
    ax2.set_ylabel('Silhouette Score', color='red')
    ax2.tick_params('y', colors='red')

    # Tambahkan garis vertikal untuk nilai k optimal
    ax1.axvline(x=optimal_k, color='green', linestyle='--')
    ax2.axvline(x=optimal_k, color='green', linestyle='--')

    plt.title('Elbow Method and Silhouette Score')
    plt.xticks(range(min_k, min_k + len(sse_scores)))
    plt.grid(True)
    plt.show()
```

In [ ]:

```
    sse = []
    silhouette_scores = []

    for n_clusters in range(2, 11):
        kmeans = Kmeans1(n_clusters=n_clusters, random_state=42)
        labels = kmeans.fit_predict(data_normalized)
        sse.append(kmeans.inertia_)
        silhouette_avg = silhouette_score(data_normalized, labels)
        silhouette_scores.append(silhouette_avg)

    table_elbow = {
        'K': range(2, 11),
        'sse': sse,
        'silhouette': silhouette_scores
    }
    df = pd.DataFrame(table_elbow)
    df

    data_cluster1 = merge_data[['tbc', 'Umur_TB', 'Tersedia air bersih', 'Ada
    lalat/nyamuk di sekitar tumpukan sampah', 'Ada sampah dan jarang dibersihkan',
                                'Ada sampah tapi tidak banyak', 'Bersih tanpa ada sampah',
    'Ada ventilasi tapi hanya sedikit', 'Ada ventilasi tapi jarang terbuka',
                                'Ada ventilasi tapi selalu tertutup', 'Ada ventilasi disemua
    ruangan']]
```

```
#normalisasi data
cluster1 = data_normalized.copy()
kmeans1 = KMeans(n_cluster=8, init_pp=False, seed=2014)
kmeans1.fit(cluster1)

predicted_kmeans1 = kmeans1.predict(cluster1)
print('Silhouette:', silhouette_score(cluster1, predicted_kmeans1))
print('SSE:', kmeans1.SSE)

Silhouette: 0.13786066757747145
SSE: 5.437251912054526

frame1 = pd.DataFrame(cluster1)
frame1['cluster'] = predicted_kmeans1
frame1['cluster'].value_counts()


predicted_kmeans1


# Pemetaan warna klaster
color_map = {0: 'pink', 1: 'grey', 2: 'green', 3: 'blue', 4: 'black',
5:'yellow', 6: 'orange', 7: 'brown'}  # Atur warna untuk masing-masing
klaster

fig, ax = plt.subplots(figsize=(15, 9))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster1.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster1.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_kmeans1])

# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster1.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster1[predicted_kmeans1 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
```

```
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()

pso1 = ParticleSwarmOptimizedClustering(n_cluster=8,
                                        n_particles=10,
                                        data=cluster1,
                                        hybrid=True,
                                        max_iter=4000,
                                        print_debug=50)


hist1 = pso1.run()


pso_kmeans1 = KMeans(n_cluster=8)
pso_kmeans1.centroid = pso1.gbest_centroids.copy()

predicted_pso1 = pso_kmeans1.predict(cluster1)

print('Silhouette:', silhouette_score(cluster1, predicted_pso1))
print('SSE:', calc_sse(centroids=pso1.gbest_centroids,
                       data=cluster1,
                       labels=predicted_pso1))

frame1 = pd.DataFrame(cluster1)
frame1['cluster'] = predicted_pso1
frame1['cluster'].value_counts()

predicted_pso1

# Pemetaan warna klaster
color_map = {0: '#FF0000', 1: '#FFA500', 2: '#FFFF00', 3: '#00FF00', 4:
'#00FFFF', 5:'#0000FF', 6: '#800080', 7: '#FFC0CB'}

fig, ax = plt.subplots(figsize=(15, 9))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
```

```
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster1.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster1.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_pso1])
# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster1.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster1[predicted_pso1 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()


#evaluasi sse dan silhoute kmeans dengan 20 iterasi
kmeanspp = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    kmean_rep = KMeans(n_cluster=8, init_pp=True, seed=2014)
    kmean_rep.fit(cluster1)
    predicted_kmean_rep = kmean_rep.predict(cluster1)
    silhouette = silhouette_score(cluster1,
```

```
                                        predicted_kmean_rep)
    sse = kmean_rep.SSE
    dbi = davies_bouldin_score(cluster1, predicted_kmean_rep)
    kmeanspp['silhouette'].append(silhouette)
    kmeanspp['sse'].append(sse)
    kmeanspp['dbi'].append(dbi)


#evaluasi sse dan silhoute pso-kmeans dengan 20 iterasi
pso_hybrid = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(10):
    pso_rep2 = ParticleSwarmOptimizedClustering(n_cluster=8,
                                                n_particles=20,
                                                data=cluster1,
                                                hybrid=True,
                                                max_iter=2000,

print_debug=2000)
    pso_rep2.run()
    pso_kmeans2 = KMeans(n_cluster=8,
                         init_pp=False,
                         seed=144)
    pso_kmeans2.centroid = pso_rep2.gbest_centroids.copy()
    predicted_pso_rep2 = pso_kmeans2.predict(cluster1)

    silhouette = silhouette_score(cluster1,
                                  predicted_pso_rep2)
    sse = calc_sse(centroids=pso_rep2.gbest_centroids,
                   data=cluster1,
                   labels=predicted_pso_rep2)
    dbi = davies_bouldin_score(cluster1, predicted_pso_rep2)

    pso_hybrid['silhouette'].append(silhouette)
    pso_hybrid['sse'].append(sse)
    pso_hybrid['dbi'].append(dbi)


evaluasi = {
    'method' : ['K-Means', 'PSO Kmeans'],
    'sse_mean' : [
        np.around(np.mean(kmeanspp['sse']), decimals=10),
        np.around(np.mean(pso_hybrid['sse']), decimals=10),
    ],
    'silhouette_mean' : [
        np.around(np.mean(kmeanspp['silhouette']), decimals=10),
        np.around(np.mean(pso_hybrid['silhouette']), decimals=10),
    ],
    'dbi_mean': [
        np.around(np.mean(kmeanspp['dbi']), decimals=10),
        np.around(np.mean(pso_hybrid['dbi']), decimals=10),
```

```
    ]
}


eval = pd.DataFrame.from_dict(evaluasi)
eval

# Menghitung dan menampilkan nilai rentang usia di tengah
setiap klaster
age_ranges = []
for i in range(8):
    cluster_data = data_cluster1[predicted_pso1 == i]
    min_age = cluster_data["Umur_TB"].min()
    max_age = cluster_data["Umur_TB"].max()

    # Penanganan nilai 0 atau NaN
    if np.isnan(max_age):
        age_range = f"{int(min_age)}+"
    else:
        if np.isnan(min_age):
            age_range = f"0-{int(max_age)}"
        else:
            age_range = f'{int(min_age)}-{int(max_age)}'

    age_ranges.append(age_range)

# Mengganti semua nilai pada setiap klaster menjadi rentang
usia
for i in range(8):
    data_cluster1.loc[predicted_pso1 == i, 'Rentang_Usia_TB'] =
age_ranges[i]

data_cluster1 = data_cluster1.drop(['Umur_TB'], axis=1)

#create new column for cluster labels associated with each
subject
data_cluster1['labels'] = predicted_kmeans1
data_cluster1['Segment'] = data_cluster1['labels'].map({0: 'First', 1:
'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sistxh', 6: 'Seventh', 7:
'Eighth'})


#create new column for cluster labels associated with each
subject
data_cluster1['labels'] = predicted_pso1
data_cluster1['Segment'] = data_cluster1['labels'].map({0: 'First', 1:
'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sistxh', 6: 'Seventh', 7:
'Eighth'})

#Order the cluster
data_cluster1['Segment'] = data_cluster1['Segment'].astype('category')
data_cluster1['Segment'] =
data_cluster1['Segment'].cat.reorder_categories(['First', 'Second',
'Third', 'Fourth', 'Fifth', 'Sistxh', 'Seventh', 'Eighth'])
```

```
data_cluster1[data_cluster1['labels']==0]
dict_cat = {}
dict_num = {}
for cat in data_cluster1.select_dtypes('object'):
  if len(data_cluster1[cat]) > 0:  # Periksa apakah ada data
tersedia
    dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in data_cluster1.select_dtypes(['int64', 'float64', 'object']):
  if (num == 'Total'):
    continue
  dict_num[num] = ['mean']

data_cluster1.rename(columns={'labels': 'Total'}, inplace=True)
data_percluster1 = data_cluster1.groupby('Segment').agg({
 'Total': 'count',
 **dict_num,
 **dict_cat
}).T

data_cluster1['coord'] = coord1

data_cluster1 = data_cluster1.T
data_percluster1.to_json(r'cluster1_result.json')
data_cluster1.to_json(r'cluster1_df.json')


data = merge_data.copy()

selected_columns = ['dbd', 'Umur_DBD','Tersedia air bersih', 'Ada lalat/nyamuk
di sekitar tumpukan sampah', 'Ada sampah dan jarang dibersihkan',
                    'Ada sampah tapi tidak banyak', 'Bersih tanpa ada sampah',
'Ada ventilasi tapi hanya sedikit', 'Ada ventilasi tapi jarang terbuka',
                    'Ada ventilasi tapi selalu tertutup', 'Ada ventilasi disemua
ruangan']
data_selected = data[selected_columns]

# Normalisasi L2
normalizer = Normalizer(norm='l2')
data_normalized = normalizer.transform(data_selected)

# Data yang telah dinormalisasi
data_normalized = data_normalized.copy()

# Inisialisasi range nilai k
min_k = 2
max_k = 11

# List untuk menyimpan nilai SSE
sse_scores = []
# List untuk menyimpan nilai Silhouette Score
silhouette_scores = []
```

```
# Loop untuk mencari SSE dan Silhouette Score pada setiap
nilai k
for k in range(min_k, max_k+1):
    kmeans = Kmeans1(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(data_normalized)
    sse = kmeans.inertia_
    sse_scores.append(sse)
    if len(np.unique(cluster_labels)) > 1:
        silhouette_avg = silhouette_score(data_normalized,
cluster_labels)
        silhouette_scores.append(silhouette_avg)
    else:
        silhouette_scores.append(0)

# Mencari nilai k optimal berdasarkan Silhouette Score
optimal_k = min_k + np.argmax(silhouette_scores)

# Plot grafik SSE dan Silhouette Score
fig, ax1 = plt.subplots()

ax1.plot(range(min_k, min_k + len(sse_scores)), sse_scores,
marker='o')
ax1.set_xlabel('Number of Clusters (k)')
ax1.set_ylabel('SSE', color='blue')
ax1.tick_params('y', colors='blue')

ax2 = ax1.twinx()
ax2.plot(range(min_k, min_k + len(silhouette_scores)),
silhouette_scores, marker='o', color='red')
ax2.set_ylabel('Silhouette Score', color='red')
ax2.tick_params('y', colors='red')

# Tambahkan garis vertikal untuk nilai k optimal
ax1.axvline(x=optimal_k, color='green', linestyle='--')
ax2.axvline(x=optimal_k, color='green', linestyle='--')

plt.title('Elbow Method and Silhouette Score')
plt.xticks(range(min_k, min_k + len(sse_scores)))
plt.grid(True)
plt.show()

sse = []
silhouette_scores = []

for n_clusters in range(2, 11):
    kmeans = Kmeans1(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(data_normalized)
    sse.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(data_normalized, labels)
    silhouette_scores.append(silhouette_avg)

table_elbow = {
    'K': range(2, 11),
    'sse': sse,
    'silhouette': silhouette_scores
```

```
}
df = pd.DataFrame(table_elbow)
df

data_cluster2 = merge_data[['dbd', 'Umur_DBD','Tersedia air bersih', 'Ada
lalat/nyamuk di sekitar tumpukan sampah', 'Ada sampah dan jarang dibersihkan',
                            'Ada sampah tapi tidak banyak', 'Bersih tanpa ada sampah',
'Ada ventilasi tapi hanya sedikit', 'Ada ventilasi tapi jarang terbuka',
                            'Ada ventilasi tapi selalu tertutup', 'Ada ventilasi disemua
ruangan']]

# Standardize data
cluster2 = data_normalized.copy()

kmeans2 = KMeans(n_cluster=9)
kmeans2.fit(cluster2)

predicted_kmeans2 = kmeans2.predict(cluster2)
print('Silhouette:', silhouette_score(cluster2, predicted_kmeans2))
print('SSE:', kmeans2.SSE)

frame2 = pd.DataFrame(cluster2)
frame2['cluster'] = predicted_kmeans2
frame2['cluster'].value_counts()

predicted_kmeans2

pso2 = ParticleSwarmOptimizedClustering(n_cluster=9,
                                        n_particles=10,
                                        data=cluster2,
                                        hybrid=True,
                                        max_iter=4000,
                                        print_debug=50)


hist = pso2.run()
pso_kmeans2 = KMeans(n_cluster=9)
pso_kmeans2.centroid = pso2.gbest_centroids.copy()

predicted_pso2 = pso_kmeans2.predict(cluster2)

print('Silhouette:', silhouette_score(cluster2, predicted_pso2))
print('SSE:', calc_sse(centroids=pso2.gbest_centroids,
                       data=cluster2,
                       labels=predicted_pso2))


frame2 = pd.DataFrame(cluster2)
frame2['cluster'] = predicted_pso2
frame2['cluster'].value_counts()


predicted_pso2

# Pemetaan warna klaster
```

```
color_map = {0: '#FFD700', 1: '#008000', 2: '#FF00FF', 3: '#000080', 4:
'#800000', 5:'#C0C0C0', 6: '#008080', 7: '#FF6347', 8: '#F08080'}

fig, ax = plt.subplots(figsize=(15, 9))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster2.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster2.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_pso1])
# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster2.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = pso_kmeans2.centroid
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster2[predicted_pso2 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()


#evaluasi sse dan silhoute kmeans dengan 20 iterasi
kmeanspp = {
    'silhouette': [],
    'sse' : [],
```

```
        'dbi': [],
    }
    for _ in range(10):
        kmean_rep = KMeans(n_cluster=9, init_pp=True, seed=2014)
        kmean_rep.fit(cluster2)
        predicted_kmean_rep = kmean_rep.predict(cluster2)
        silhouette = silhouette_score(cluster2,
                                      predicted_kmean_rep)
        sse = kmean_rep.SSE
        dbi = davies_bouldin_score(cluster2, predicted_kmean_rep)
        kmeanspp['silhouette'].append(silhouette)
        kmeanspp['sse'].append(sse)
        kmeanspp['dbi'].append(dbi)

    #evaluasi sse dan silhoute pso-kmeans dengan 20 iterasi
    pso_hybrid = {
        'silhouette': [],
        'sse' : [],
        'dbi': [],
    }
    for _ in range(20):
        pso_rep2 = ParticleSwarmOptimizedClustering(n_cluster=9,
                                                    n_particles=10,
                                                    data=cluster2,
                                                    hybrid=True,
                                                    max_iter=2000,
                                                    print_debug=50)
        pso_rep2.run()
        pso_kmeans2 = KMeans(n_cluster=9,
                             init_pp=False,
                             seed=144)
        pso_kmeans2.centroid = pso_rep2.gbest_centroids.copy()
        predicted_pso_rep2 = pso_kmeans2.predict(cluster2)

        silhouette = silhouette_score(cluster2,
                                      predicted_pso_rep2)
        sse = calc_sse(centroids=pso_rep2.gbest_centroids,
                       data=cluster2,
                       labels=predicted_pso_rep2)
        dbi = davies_bouldin_score(cluster2, predicted_pso_rep2)

        pso_hybrid['silhouette'].append(silhouette)
        pso_hybrid['sse'].append(sse)
        pso_hybrid['dbi'].append(dbi)

    evaluasi = {
        'method' : ['K-Means', 'PSO Kmeans'],
        'sse_mean' : [
            np.around(np.mean(kmeanspp['sse']), decimals=10),
            np.around(np.mean(pso_hybrid['sse']), decimals=10),
        ],
        'silhouette_mean' : [
            np.around(np.mean(kmeanspp['silhouette']), decimals=10),
            np.around(np.mean(pso_hybrid['silhouette']), decimals=10),
```

```
        ],
        'dbi_mean': [
            np.around(np.mean(kmeanspp['dbi']), decimals=10),
            np.around(np.mean(pso_hybrid['dbi']), decimals=10),
        ]
    }

    eval = pd.DataFrame.from_dict(evaluasi)
    eval

    # Menghitung dan menampilkan nilai rentang usia di tengah
    setiap klaster
    age_ranges = []
    for i in range(9):
        cluster_data = data_cluster2[predicted_pso2 == i]
        min_age = cluster_data["Umur_DBD"].min()
        max_age = cluster_data["Umur_DBD"].max()

        # Penanganan nilai 0 atau NaN
        if np.isnan(max_age):
            age_range = f"{int(min_age)}+"
        else:
            if np.isnan(min_age):
                age_range = f"0-{int(max_age)}"
            else:
                age_range = f'{int(min_age)}-{int(max_age)}'

        age_ranges.append(age_range)

    # Mengganti semua nilai pada setiap klaster menjadi rentang
    usia
    for i in range(9):
        data_cluster2.loc[predicted_pso2 == i, 'Rentang_Usia_DBD'] =
    age_ranges[i]

    data_cluster2 = data_cluster2.drop(['Umur_DBD'], axis=1)

    #create new column for cluster labels associated with each
    subject
    data_cluster2['labels'] = predicted_pso2
    data_cluster2['Segment'] = data_cluster2['labels'].map({0: 'First', 1:
    'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sistxh', 6: 'Seventh', 7:
    'Eighth', 8: 'Ninth'})


    #Order the cluster
    data_cluster2['Segment'] = data_cluster2['Segment'].astype('category')
    data_cluster2['Segment'] =
    data_cluster2['Segment'].cat.reorder_categories(['First', 'Second',
    'Third', 'Fourth', 'Fifth', 'Sistxh', 'Seventh', 'Eighth', 'Ninth'])


    # ==== CREATE DICTIONARY FOR OBJECT AND NUMERICAL COLUMNS
    ====#
```

```
dict_cat = {}
dict_num = {}
for cat in data_cluster2.select_dtypes('object'):
 dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in data_cluster2.select_dtypes(['int64', 'float64']):
  if (num == 'Total'):
    continue
  dict_num[num] = ['mean']
# ============ CREATE TABLE FOR EACH CLUSTER ============#
data_cluster2.rename(columns={'labels': 'Total'}, inplace=True)
data_percluster2 = data_cluster2.groupby('Segment').agg({
 'Total': 'count',
 **dict_num,
 **dict_cat
}).T

data_cluster2['coord'] = coord2

data_cluster2 = data_cluster2.T
data_percluster2.to_json(r'cluster2_result.json')
data_cluster2.to_json(r'cluster2_df.json')

data = merge_data.copy()

selected_columns = ['tifoid', 'Umur_Tifoid','Tersedia air bersih', 'Ada
lalat/nyamuk di sekitar tumpukan sampah', 'Ada sampah dan jarang dibersihkan',
                    'Ada sampah tapi tidak banyak', 'Bersih tanpa ada sampah',
'Ada ventilasi tapi hanya sedikit', 'Ada ventilasi tapi jarang terbuka',
                    'Ada ventilasi tapi selalu tertutup', 'Ada ventilasi disemua
ruangan']  # Ganti dengan nama variabel yang ingin Anda gunakan
data_selected = data[selected_columns]


# Normalisasi L2
normalizer = Normalizer(norm='l2')
data_normalized = normalizer.transform(data_selected)

# Data yang telah dinormalisasi
data_normalized = data_normalized.copy()

# Inisialisasi range nilai k
min_k = 2
max_k = 11

# List untuk menyimpan nilai SSE
sse_scores = []
# List untuk menyimpan nilai Silhouette Score
silhouette_scores = []

# Loop untuk mencari SSE dan Silhouette Score pada setiap
nilai k
for k in range(min_k, max_k+1):
    kmeans = Kmeans1(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(data_normalized)
```

```
        sse = kmeans.inertia_
        sse_scores.append(sse)
        if len(np.unique(cluster_labels)) > 1:
            silhouette_avg = silhouette_score(data_normalized,
cluster_labels)
            silhouette_scores.append(silhouette_avg)
        else:
            silhouette_scores.append(0)

    # Mencari nilai k optimal berdasarkan Silhouette Score
    optimal_k = min_k + np.argmax(silhouette_scores)

    # Plot grafik SSE dan Silhouette Score
    fig, ax1 = plt.subplots()

    ax1.plot(range(min_k, min_k + len(sse_scores)), sse_scores,
    marker='o')
    ax1.set_xlabel('Number of Clusters (k)')
    ax1.set_ylabel('SSE', color='blue')
    ax1.tick_params('y', colors='blue')

    ax2 = ax1.twinx()
    ax2.plot(range(min_k, min_k + len(silhouette_scores)),
    silhouette_scores, marker='o', color='red')
    ax2.set_ylabel('Silhouette Score', color='red')
    ax2.tick_params('y', colors='red')

    # Tambahkan garis vertikal untuk nilai k optimal
    ax1.axvline(x=optimal_k, color='green', linestyle='--')
    ax2.axvline(x=optimal_k, color='green', linestyle='--')

    plt.title('Elbow Method and Silhouette Score')
    plt.xticks(range(min_k, min_k + len(sse_scores)))
    plt.grid(True)
    plt.show()


    sse = []
    silhouette_scores = []

    for n_clusters in range(2, 12):
        kmeans = Kmeans1(n_clusters=n_clusters, random_state=42)
        labels = kmeans.fit_predict(data_normalized)
        sse.append(kmeans.inertia_)
        silhouette_avg = silhouette_score(data_normalized, labels)
        silhouette_scores.append(silhouette_avg)

    table_elbow = {
        'K': range(2, 12),
        'sse': sse,
        'silhouette': silhouette_scores
    }
    df = pd.DataFrame(table_elbow)
    df
```

```
data_cluster3 = merge_data[['tifoid','Umur_Tifoid','Tersedia air bersih', 'Ada
lalat/nyamuk di sekitar tumpukan sampah', 'Ada sampah dan jarang dibersihkan',
                            'Ada sampah tapi tidak banyak', 'Bersih tanpa ada sampah',
'Ada ventilasi tapi hanya sedikit', 'Ada ventilasi tapi jarang terbuka',
                            'Ada ventilasi tapi selalu tertutup', 'Ada ventilasi disemua
ruangan' ]]

# Standardize data
cluster3 = data_normalized.copy()

kmeans3 = KMeans(n_cluster=11)
kmeans3.fit(cluster3)


predicted_kmeans3 = kmeans3.predict(cluster3)
print('Silhouette:', silhouette_score(cluster3, predicted_kmeans3))
print('SSE:', kmeans3.SSE)

frame1 = pd.DataFrame(cluster3)
frame1['cluster'] = predicted_kmeans3
frame1['cluster'].value_counts()


predicted_kmeans3

# Pemetaan warna klaster
color_map = {0: 'pink', 1: 'grey', 2: 'green', 3: 'blue', 4: 'black',
5:'yellow', 6: 'orange', 7: 'brown', 8: 'purple', 9: 'navy', 10: 'red'}  #
Atur warna untuk masing-masing klaster

fig, ax = plt.subplots(figsize=(15, 9))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster3.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster3.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_kmeans3])
# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster3.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
```

```
    cluster_data = data_cluster3[predicted_kmeans3 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()


pso3 = ParticleSwarmOptimizedClustering(n_cluster=11,
                                        n_particles=10,
                                        data=cluster3,
                                        hybrid=True,
                                        max_iter=4000,
                                        print_debug=100)


hist3 = pso3.run()

pso_kmeans3 = KMeans(n_cluster=11)
pso_kmeans3.centroid = pso3.gbest_centroids.copy()


predicted_pso3 = pso_kmeans3.predict(cluster3)


print('Silhouette:', silhouette_score(cluster3, predicted_pso3))
print('SSE:', calc_sse(centroids=pso3.gbest_centroids,
                       data=cluster3,
                       labels=predicted_pso3))

frame1 = pd.DataFrame(cluster3)
frame1['cluster'] = predicted_pso3
frame1['cluster'].value_counts()

predicted_pso3


# Pemetaan warna klaster
```

```
color_map = {0: '#808080', 1: '#DC143C', 2: '#2E8B57', 3: '#FFA07A',
4: '#FFFFE0', 5: '#DAA520', 6:'#B8860B', 7: '#FF4500', 8: '#DA70D6', 9:
'#9370DB', 10: '#6A5ACD'}  # Atur warna untuk masing-masing
klaster

fig, ax = plt.subplots(figsize=(15, 9))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster3.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster3.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_pso3])
# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster3.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster3[predicted_pso3 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()


#evaluasi sse dan silhoute kmeans dengan 20 iterasi
kmeanspp = {
```

```
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    kmean_rep = KMeans(n_cluster=11, init_pp=True, seed=2014)
    kmean_rep.fit(cluster3)
    predicted_kmean_rep = kmean_rep.predict(cluster3)
    silhouette = silhouette_score(cluster3,
                                  predicted_kmean_rep)
    sse = kmean_rep.SSE
    dbi = davies_bouldin_score(cluster3, predicted_kmean_rep)
    kmeanspp['silhouette'].append(silhouette)
    kmeanspp['sse'].append(sse)
    kmeanspp['dbi'].append(dbi)


#evaluasi sse dan silhoute pso-kmeans dengan 20 iterasi
pso_hybrid = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    pso_rep2 = ParticleSwarmOptimizedClustering(n_cluster=11,
                                                n_particles=40,
                                                data=cluster3,
                                                hybrid=True,
                                                max_iter=2000,

print_debug=2000)
    pso_rep2.run()
    pso_kmeans2 = KMeans(n_cluster=11,
                         init_pp=False,
                         seed=144)
    pso_kmeans2.centroid = pso_rep2.gbest_centroids.copy()
    predicted_pso_rep2 = pso_kmeans2.predict(cluster3)

    silhouette = silhouette_score(cluster3,
                                  predicted_pso_rep2)
    sse = calc_sse(centroids=pso_rep2.gbest_centroids,
                   data=cluster3,
                   labels=predicted_pso_rep2)
    dbi = davies_bouldin_score(cluster3, predicted_pso_rep2)

    pso_hybrid['silhouette'].append(silhouette)
    pso_hybrid['sse'].append(sse)
    pso_hybrid['dbi'].append(dbi)


evaluasi = {
    'method' : ['K-Means', 'PSO Kmeans'],
    'sse_mean' : [
        np.around(np.mean(kmeanspp['sse']), decimals=10),
```

```
            np.around(np.mean(pso_hybrid['sse']), decimals=10),
        ],
        'silhouette_mean' : [
            np.around(np.mean(kmeanspp['silhouette']), decimals=10),
            np.around(np.mean(pso_hybrid['silhouette']), decimals=10),
        ],
        'dbi_mean': [
            np.around(np.mean(kmeanspp['dbi']), decimals=10),
            np.around(np.mean(pso_hybrid['dbi']), decimals=10),
        ]
    }


    eval = pd.DataFrame.from_dict(evaluasi)
    eval

    # Menghitung dan menampilkan nilai rentang usia di tengah
    setiap klaster
    age_ranges = []
    for i in range(11):
        cluster_data = data_cluster3[predicted_pso3 == i]
        min_age = cluster_data["Umur_Tifoid"].min()
        max_age = cluster_data["Umur_Tifoid"].max()

        # Penanganan nilai 0 atau NaN
        if np.isnan(max_age):
            age_range = f"{int(min_age)}+"
        else:
            if np.isnan(min_age):
                age_range = f"0-{int(max_age)}"
            else:
                age_range = f'{int(min_age)}-{int(max_age)}'

        age_ranges.append(age_range)

    # Mengganti semua nilai pada setiap klaster menjadi rentang
    usia
    for i in range(11):
        data_cluster3.loc[predicted_pso3 == i, 'Rentang_Usia_Tifoid'] =
    age_ranges[i]

    data_cluster3 = data_cluster3.drop(['Umur_Tifoid'], axis=1)


    #create new column for cluster labels associated with each
    subject
    data_cluster3['labels'] = predicted_pso3
    data_cluster3['Segment'] = data_cluster3['labels'].map({0: 'First', 1:
    'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sixth', 6: 'Seventh', 7:
    'Eighth', 8: 'Ninth', 9: 'Tenth', 10: 'Eleventh'})

    #Order the cluster
    data_cluster3['Segment'] = data_cluster3['Segment'].astype('category')
```

```
data_cluster3['Segment'] =
data_cluster3['Segment'].cat.reorder_categories(['First', 'Second',
'Third', 'Fourth', 'Fifth', 'Sixth', 'Seventh', 'Eighth', 'Ninth', 'Tenth', 'Eleventh'])


# ==== CREATE DICTIONARY FOR OBJECT AND NUMERICAL COLUMNS
====#
dict_cat = {}
dict_num = {}
for cat in data_cluster3.select_dtypes('object'):
 dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in data_cluster3.select_dtypes(['int64', 'float64']):
  if (num == 'Total'):
    continue
  dict_num[num] = ['mean']
# ============ CREATE TABLE FOR EACH CLUSTER ============#
data_cluster3.rename(columns={'labels': 'Total'}, inplace=True)
data_percluster3 = data_cluster3.groupby('Segment').agg({
 'Total': 'count',
 **dict_num,
 **dict_cat
}).T

data_cluster3['coord'] = coord3

data_cluster3 = data_cluster3.T
data_percluster3.to_json(r'cluster3_result.json')
data_cluster3.to_json(r'cluster3_df.json')



data = merge_data.copy()

selected_columns = ['tbc', 'Umur_TB','Nafsu Makan Kurang',
                    'Sering Jajan Diluar',
                    'Sering Pakai Jamban/Toilet']
data_selected = data[selected_columns]


# Normalisasi L2
normalizer = Normalizer(norm='l2')
data_normalized = normalizer.transform(data_selected)

# Data yang telah dinormalisasi
data_normalized = data_normalized.copy()

# Inisialisasi range nilai k
min_k = 2
max_k = 11

# List untuk menyimpan nilai SSE
sse_scores = []
# List untuk menyimpan nilai Silhouette Score
silhouette_scores = []
```

```
# Loop untuk mencari SSE dan Silhouette Score pada setiap
nilai k
for k in range(min_k, max_k+1):
    kmeans = Kmeans1(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(data_normalized)
    sse = kmeans.inertia_
    sse_scores.append(sse)
    if len(np.unique(cluster_labels)) > 1:
        silhouette_avg = silhouette_score(data_normalized,
cluster_labels)
        silhouette_scores.append(silhouette_avg)
    else:
        silhouette_scores.append(0)

# Mencari nilai k optimal berdasarkan Silhouette Score
optimal_k = min_k + np.argmax(silhouette_scores)

# Plot grafik SSE dan Silhouette Score
fig, ax1 = plt.subplots()

ax1.plot(range(min_k, min_k + len(sse_scores)), sse_scores,
marker='o')
ax1.set_xlabel('Number of Clusters (k)')
ax1.set_ylabel('SSE', color='blue')
ax1.tick_params('y', colors='blue')

ax2 = ax1.twinx()
ax2.plot(range(min_k, min_k + len(silhouette_scores)),
silhouette_scores, marker='o', color='red')
ax2.set_ylabel('Silhouette Score', color='red')
ax2.tick_params('y', colors='red')

# Tambahkan garis vertikal untuk nilai k optimal
ax1.axvline(x=optimal_k, color='green', linestyle='--')
ax2.axvline(x=optimal_k, color='green', linestyle='--')

plt.title('Elbow Method and Silhouette Score')
plt.xticks(range(min_k, min_k + len(sse_scores)))
plt.grid(True)
plt.show()

sse = []
silhouette_scores = []

for n_clusters in range(2, 12):
    kmeans = Kmeans1(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(data_normalized)
    sse.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(data_normalized, labels)
    silhouette_scores.append(silhouette_avg)

table_elbow = {
    'K': range(2, 12),
    'sse': sse,
```

```
    'silhouette': silhouette_scores
}
df = pd.DataFrame(table_elbow)
df

data_cluster4 = merge_data[['tbc', 'Umur_TB','Nafsu Makan Kurang',
                      'Sering Jajan Diluar',
                      'Sering Pakai Jamban/Toilet']]

# Standardize data
cluster4 = data_normalized.copy()

kmeans4 = KMeans(n_cluster=10)
kmeans4.fit(cluster4)

predicted_kmeans4 = kmeans4.predict(cluster4)
print('Silhouette:', silhouette_score(cluster4, predicted_kmeans4))
print('SSE:', kmeans4.SSE)

Silhouette: 0.7171578823691377
SSE: 0.3434657411672276
frame4 = pd.DataFrame(cluster4)
frame4['cluster'] = predicted_kmeans4
frame4['cluster'].value_counts()


predicted_kmeans4

# Pemetaan warna klaster
color_map = {0: 'pink', 1: 'grey', 2: 'green', 3: 'blue', 4: 'black',
5:'yellow', 6: 'orange', 7: 'brown', 8: 'purple', 9: 'navy'}

fig, ax = plt.subplots(figsize=(15, 9))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster4.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster4.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_kmeans4])
# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster4.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
```

```
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster4[predicted_kmeans4 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()



pso4 = ParticleSwarmOptimizedClustering(n_cluster=10,
                                        n_particles=10,
                                        data=cluster4,
                                        hybrid=True,
                                        max_iter=4000,
                                        print_debug=50)


hist4 = pso4.run()

pso_kmeans4 = KMeans(n_cluster=10)
pso_kmeans4.centroid = pso4.gbest_centroids.copy()

predicted_pso4 = pso_kmeans4.predict(cluster4)


predicted_pso4 = pso_kmeans4.predict(cluster4)
print('Silhouette:', silhouette_score(cluster4, predicted_pso4))
print('SSE:', calc_sse(centroids=pso4.gbest_centroids,
                       data=cluster4,
                       labels=predicted_pso4))

frame4 = pd.DataFrame(cluster4)
frame4['cluster'] = predicted_pso4
frame4['cluster'].value_counts()


predicted_pso4
```

```
# Pemetaan warna klaster
color_map = {0: '#7B68EE', 1: '#48D1CC', 2: '#ADFF2F', 3: '#20B2AA',
4: '#00FA9A', 5:'#3CB371', 6: '#FF69B4', 7: '#7FFF00', 8: '#00CED1', 9:
'#4682B4'}  # Atur warna untuk masing-masing klaster

fig, ax = plt.subplots(figsize=(15, 8))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster4.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster4.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_pso4])

# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster4.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster4[predicted_pso4 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()

#evaluasi sse dan silhoute kmeans dengan 20 iterasi
kmeanspp = {
```

```
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    kmean_rep = KMeans(n_cluster=7, init_pp=True, seed=2014)
    kmean_rep.fit(cluster4)
    predicted_kmean_rep = kmean_rep.predict(cluster4)
    silhouette = silhouette_score(cluster4,
                                  predicted_kmean_rep)
    sse = kmean_rep.SSE
    dbi = davies_bouldin_score(cluster4, predicted_kmean_rep)
    kmeanspp['silhouette'].append(silhouette)
    kmeanspp['sse'].append(sse)
    kmeanspp['dbi'].append(dbi)
```

In [ ]:

```
#evaluasi sse dan silhoute pso-kmeans dengan 20 iterasi
pso_hybrid = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    pso_rep2 = ParticleSwarmOptimizedClustering(n_cluster=10,
                                                n_particles=20,
                                                data=cluster4,
                                                hybrid=True,
                                                max_iter=2000,

print_debug=2000)
    pso_rep2.run()
    pso_kmeans2 = KMeans(n_cluster=10,
                         init_pp=False,
                         seed=144)
    pso_kmeans2.centroid = pso_rep2.gbest_centroids.copy()
    predicted_pso_rep2 = pso_kmeans2.predict(cluster4)

    silhouette = silhouette_score(cluster4,
                                  predicted_pso_rep2)
    sse = calc_sse(centroids=pso_rep2.gbest_centroids,
                   data=cluster4,
                   labels=predicted_pso_rep2)
    dbi = davies_bouldin_score(cluster4, predicted_pso_rep2)

    pso_hybrid['silhouette'].append(silhouette)
    pso_hybrid['sse'].append(sse)
    pso_hybrid['dbi'].append(dbi)


evaluasi = {
    'method' : ['K-Means', 'PSO Kmeans'],
    'sse_mean' : [
```

```
            np.around(np.mean(kmeanspp['sse']), decimals=10),
            np.around(np.mean(pso_hybrid['sse']), decimals=10),
        ],
        'silhouette_mean' : [
            np.around(np.mean(kmeanspp['silhouette']), decimals=10),
            np.around(np.mean(pso_hybrid['silhouette']), decimals=10),
        ],
        'dbi_mean': [
            np.around(np.mean(kmeanspp['dbi']), decimals=10),
            np.around(np.mean(pso_hybrid['dbi']), decimals=10),
        ]
    }
```

In [ ]:

```
eval = pd.DataFrame.from_dict(evaluasi)
eval


# Menghitung dan menampilkan nilai rentang usia di tengah
setiap klaster
age_ranges = []
for i in range(10):
    cluster_data = data_cluster4[predicted_pso4 == i]
    min_age = cluster_data["Umur_TB"].min()
    max_age = cluster_data["Umur_TB"].max()

    # Penanganan nilai 0 atau NaN
    if np.isnan(max_age):
        age_range = f"{int(min_age)}+"
    else:
        if np.isnan(min_age):
            age_range = f"0-{int(max_age)}"
        else:
            age_range = f'{int(min_age)}-{int(max_age)}'

    age_ranges.append(age_range)

# Mengganti semua nilai pada setiap klaster menjadi rentang
usia
for i in range(10):
    data_cluster4.loc[predicted_pso4 == i, 'Rentang_Usia_TB'] =
age_ranges[i]

data_cluster4 = data_cluster4.drop(['Umur_TB'], axis=1)


#create new column for cluster labels associated with each
subject
data_cluster4['labels'] = predicted_pso4
data_cluster4['Segment'] = data_cluster4['labels'].map({0: 'First', 1:
'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sixth', 6: 'Seventh', 7:
'Eighth', 8: 'Ninth', 9: 'Tenth'})
```

```
#Order the cluster
data_cluster4['Segment'] = data_cluster4['Segment'].astype('category')
data_cluster4['Segment'] =
data_cluster4['Segment'].cat.reorder_categories(['First', 'Second',
'Third', 'Fourth', 'Fifth', 'Sixth', 'Seventh', 'Eighth', 'Ninth', 'Tenth'])


# ==== CREATE DICTIONARY FOR OBJECT AND NUMERICAL COLUMNS
====#
dict_cat = {}
dict_num = {}
for cat in data_cluster4.select_dtypes('object'):
 dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in data_cluster4.select_dtypes(['int64', 'float64']):
  if (num == 'Total'):
    continue
  dict_num[num] = ['mean']
# ============ CREATE TABLE FOR EACH CLUSTER ============#
data_cluster4.rename(columns={'labels': 'Total'}, inplace=True)
data_percluster4 = data_cluster4.groupby('Segment').agg({
 'Total': 'count',
 **dict_num,
 **dict_cat
}).T

data_cluster4['coord'] = coord4

data_cluster4 = data_cluster4.T
data_percluster4.to_json(r'cluster4_result.json')
data_cluster4.to_json(r'cluster4_df.json')


data = merge_data.copy()

selected_columns = ['dbd', 'Umur_DBD', 'Nafsu Makan Kurang',
                    'Sering Jajan Diluar',
                    'Sering Pakai Jamban/Toilet']
data_selected = data[selected_columns]

# Normalisasi L2
normalizer = Normalizer(norm='l2')
data_normalized = normalizer.transform(data_selected)


# Data yang telah dinormalisasi
data_normalized = data_normalized.copy()

# Inisialisasi range nilai k
min_k = 2
max_k = 11

# List untuk menyimpan nilai SSE
sse_scores = []
```

```
# List untuk menyimpan nilai Silhouette Score
silhouette_scores = []

# Loop untuk mencari SSE dan Silhouette Score pada setiap
nilai k
for k in range(min_k, max_k+1):
    kmeans = Kmeans1(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(data_normalized)
    sse = kmeans.inertia_
    sse_scores.append(sse)
    if len(np.unique(cluster_labels)) > 1:
        silhouette_avg = silhouette_score(data_normalized,
cluster_labels)
        silhouette_scores.append(silhouette_avg)
    else:
        silhouette_scores.append(0)

# Mencari nilai k optimal berdasarkan Silhouette Score
optimal_k = min_k + np.argmax(silhouette_scores)

# Plot grafik SSE dan Silhouette Score
fig, ax1 = plt.subplots()

ax1.plot(range(min_k, min_k + len(sse_scores)), sse_scores,
marker='o')
ax1.set_xlabel('Number of Clusters (k)')
ax1.set_ylabel('SSE', color='blue')
ax1.tick_params('y', colors='blue')

ax2 = ax1.twinx()
ax2.plot(range(min_k, min_k + len(silhouette_scores)),
silhouette_scores, marker='o', color='red')
ax2.set_ylabel('Silhouette Score', color='red')
ax2.tick_params('y', colors='red')

# Tambahkan garis vertikal untuk nilai k optimal
ax1.axvline(x=optimal_k, color='green', linestyle='--')
ax2.axvline(x=optimal_k, color='green', linestyle='--')

plt.title('Elbow Method and Silhouette Score')
plt.xticks(range(min_k, min_k + len(sse_scores)))
plt.grid(True)
plt.show()


sse = []
silhouette_scores = []

for n_clusters in range(2, 11):
    kmeans = Kmeans1(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(data_normalized)
    sse.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(data_normalized, labels)
    silhouette_scores.append(silhouette_avg)
```

```
table_elbow = {
    'K': range(2, 11),
    'sse': sse,
    'silhouette': silhouette_scores
}
df = pd.DataFrame(table_elbow)
df


data_cluster5 = merge_data[['dbd', 'Umur_DBD', 'Nafsu Makan Kurang',
                            'Sering Jajan Diluar',
                            'Sering Pakai Jamban/Toilet']]

# Standardize data
cluster5 = data_normalized.copy()

kmeans5 = KMeans(n_cluster=8)
kmeans5.fit(cluster5)


predicted_kmeans5 = kmeans5.predict(cluster5)
print('Silhouette:', silhouette_score(cluster5, predicted_kmeans5))
print('SSE:', kmeans5.SSE)


frame5 = pd.DataFrame(cluster5)
frame5['cluster'] = predicted_kmeans5
frame5['cluster'].value_counts()

predicted_kmeans5


pso5 = ParticleSwarmOptimizedClustering(n_cluster=8,
                                        n_particles=10,
                                        data=cluster5,
                                        hybrid=True,
                                        max_iter=4000,
                                        print_debug=100)


hist5 = pso5.run()

pso_kmeans5 = KMeans(n_cluster=8)
pso_kmeans5.centroid = pso5.gbest_centroids.copy()

predicted_pso5 = pso_kmeans5.predict(cluster5)


print('Silhouette:', silhouette_score(cluster5, predicted_pso5))
print('SSE:', calc_sse(centroids=pso5.gbest_centroids,
                       data=cluster5,
                       labels=predicted_pso5))


frame5 = pd.DataFrame(cluster5)
```

```
frame5['cluster'] = predicted_pso5
frame5['cluster'].value_counts()

predicted_pso5

# Pemetaan warna klaster
color_map = {0: '#8B4513', 1: '#FF8C00', 2: '#FF1493', 3: '#1E90FF', 4:
'#9932CC', 5:'#8B008B', 6: '#556B2F', 7: '#FF00FF'}  # Atur warna untuk
masing-masing klaster

fig, ax = plt.subplots(figsize=(15, 8))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster5.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster5.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_pso5])

# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster5.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster5[predicted_pso5 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))
```

```
plt.show()


#evaluasi sse dan silhoute kmeans dengan 20 iterasi
kmeanspp = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    kmean_rep = KMeans(n_cluster=8, init_pp=True)
    kmean_rep.fit(cluster5)
    predicted_kmean_rep = kmean_rep.predict(cluster5)
    silhouette = silhouette_score(cluster5,
                                  predicted_kmean_rep)
    sse = kmean_rep.SSE
    dbi = davies_bouldin_score(cluster5, predicted_kmean_rep)
    kmeanspp['silhouette'].append(silhouette)
    kmeanspp['sse'].append(sse)
    kmeanspp['dbi'].append(dbi)


#evaluasi sse dan silhoute pso-kmeans dengan 20 iterasi
pso_hybrid = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    pso_rep2 = ParticleSwarmOptimizedClustering(n_cluster=8,
                                                n_particles=30,
                                                data=cluster5,
                                                hybrid=True,
                                                max_iter=2000,

print_debug=2000)
    pso_rep2.run()
    pso_kmeans2 = KMeans(n_cluster=8,
                         init_pp=False,
                         seed=144)
    pso_kmeans2.centroid = pso_rep2.gbest_centroids.copy()
    predicted_pso_rep2 = pso_kmeans2.predict(cluster5)

    silhouette = silhouette_score(cluster5,
                                  predicted_pso_rep2)
    sse = calc_sse(centroids=pso_rep2.gbest_centroids,
                   data=cluster5,
                   labels=predicted_pso_rep2)
    dbi = davies_bouldin_score(cluster5, predicted_pso_rep2)

    pso_hybrid['silhouette'].append(silhouette)
    pso_hybrid['sse'].append(sse)
    pso_hybrid['dbi'].append(dbi)
```

```
evaluasi = {
    'method' : ['K-Means', 'PSO Kmeans'],
    'sse_mean' : [
        np.around(np.mean(kmeanspp['sse']), decimals=10),
        np.around(np.mean(pso_hybrid['sse']), decimals=10),
    ],
    'silhouette_mean' : [
        np.around(np.mean(kmeanspp['silhouette']), decimals=10),
        np.around(np.mean(pso_hybrid['silhouette']), decimals=10),
    ],
    'dbi_mean': [
        np.around(np.mean(pso_hybrid['dbi']), decimals=10),
        np.around(np.mean(kmeanspp['dbi']), decimals=10),
    ]
}
eval = pd.DataFrame.from_dict(evaluasi)
eval


# Menghitung dan menampilkan nilai rentang usia di tengah
setiap klaster
age_ranges = []
for i in range(8):
    cluster_data = data_cluster5[predicted_pso5 == i]
    min_age = cluster_data["Umur_DBD"].min()
    max_age = cluster_data["Umur_DBD"].max()

    # Penanganan nilai 0 atau NaN
    if np.isnan(max_age):
        age_range = f"{int(min_age)}+"
    else:
        if np.isnan(min_age):
            age_range = f"0-{int(max_age)}"
        else:
            age_range = f'{int(min_age)}-{int(max_age)}'

    age_ranges.append(age_range)

# Mengganti semua nilai pada setiap klaster menjadi rentang
usia
for i in range(8):
    data_cluster5.loc[predicted_pso5 == i, 'Rentang_Usia_DBD'] =
age_ranges[i]

data_cluster5 = data_cluster5.drop(['Umur_DBD'], axis=1)


#create new column for cluster labels associated with each
subject
data_cluster5['labels'] = predicted_pso5
data_cluster5['Segment'] = data_cluster5['labels'].map({0: 'First', 1:
'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sistxh', 6: 'Seventh', 7:
'Eighth'})
```

```
#Order the cluster
data_cluster5['Segment'] = data_cluster5['Segment'].astype('category')
data_cluster5['Segment'] =
data_cluster5['Segment'].cat.reorder_categories(['First', 'Second',
'Third', 'Fourth', 'Fifth', 'Sistxh', 'Seventh', 'Eighth'])

# ==== CREATE DICTIONARY FOR OBJECT AND NUMERICAL COLUMNS
====#
dict_cat = {}
dict_num = {}
for cat in data_cluster5.select_dtypes('object'):
 dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in data_cluster5.select_dtypes(['int64', 'float64']):
  if (num == 'Total'):
    continue
  dict_num[num] = ['mean']
# ============ CREATE TABLE FOR EACH CLUSTER ============#
data_cluster5.rename(columns={'labels': 'Total'}, inplace=True)
data_percluster5 = data_cluster5.groupby('Segment').agg({
 'Total': 'count',
 **dict_num,
 **dict_cat
}).T

data_cluster5['coord'] = coord5

data_cluster5 = data_cluster5.T
data_percluster5.to_json(r'cluster5_result.json')
data_cluster5.to_json(r'cluster5_df.json')


data = merge_data.copy()

selected_columns = ['tifoid', 'Umur_Tifoid', 'Nafsu Makan Kurang',
                    'Sering Jajan Diluar',
                    'Sering Pakai Jamban/Toilet']
data_selected = data[selected_columns]


# Normalisasi L2
normalizer = Normalizer(norm='l2')
data_normalized = normalizer.transform(data_selected)


# Data yang telah dinormalisasi
data_normalized = data_normalized.copy()

# Inisialisasi range nilai k
min_k = 2
max_k = 11

# List untuk menyimpan nilai SSE
sse_scores = []
# List untuk menyimpan nilai Silhouette Score
```

```
silhouette_scores = []

# Loop untuk mencari SSE dan Silhouette Score pada setiap
nilai k
for k in range(min_k, max_k+1):
    kmeans = Kmeans1(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(data_normalized)
    sse = kmeans.inertia_
    sse_scores.append(sse)
    if len(np.unique(cluster_labels)) > 1:
        silhouette_avg = silhouette_score(data_normalized,
cluster_labels)
        silhouette_scores.append(silhouette_avg)
    else:
        silhouette_scores.append(0)

# Mencari nilai k optimal berdasarkan Silhouette Score
optimal_k = min_k + np.argmax(silhouette_scores)

# Plot grafik SSE dan Silhouette Score
fig, ax1 = plt.subplots()

ax1.plot(range(min_k, min_k + len(sse_scores)), sse_scores,
marker='o')
ax1.set_xlabel('Number of Clusters (k)')
ax1.set_ylabel('SSE', color='blue')
ax1.tick_params('y', colors='blue')

ax2 = ax1.twinx()
ax2.plot(range(min_k, min_k + len(silhouette_scores)),
silhouette_scores, marker='o', color='red')
ax2.set_ylabel('Silhouette Score', color='red')
ax2.tick_params('y', colors='red')

# Tambahkan garis vertikal untuk nilai k optimal
ax1.axvline(x=optimal_k, color='green', linestyle='--')
ax2.axvline(x=optimal_k, color='green', linestyle='--')

plt.title('Elbow Method and Silhouette Score')
plt.xticks(range(min_k, min_k + len(sse_scores)))
plt.grid(True)
plt.show()

sse = []
silhouette_scores = []

for n_clusters in range(2, 11):
    kmeans = Kmeans1(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(data_normalized)
    sse.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(data_normalized, labels)
    silhouette_scores.append(silhouette_avg)

table_elbow = {
    'K': range(2, 11),
```

```
    'sse': sse,
    'silhouette': silhouette_scores
}
df = pd.DataFrame(table_elbow)
df


data_cluster6 = merge_data[['tifoid', 'Umur_Tifoid', 'Nafsu Makan Kurang',
                            'Sering Jajan Diluar',
                            'Sering Pakai Jamban/Toilet']]

# Standardize data
cluster6 = data_normalized.copy()

kmeans6 = KMeans(n_cluster=6)
kmeans6.fit(cluster6)

predicted_kmeans6 = kmeans6.predict(cluster6)
print('Silhouette:', silhouette_score(cluster6, predicted_kmeans6))
print('SSE:', kmeans6.SSE)

frame6 = pd.DataFrame(cluster6)
frame6['cluster'] = predicted_kmeans6
frame6['cluster'].value_counts()


predicted_kmeans6


pso6 = ParticleSwarmOptimizedClustering(n_cluster=6,
                                        n_particles=40,
                                        data=cluster6,
                                        hybrid=True,
                                        max_iter=4000,
                                        print_debug=100)


hist6 = pso6.run()

pso_kmeans6 = KMeans(n_cluster=6)
pso_kmeans6.centroid = pso6.gbest_centroids.copy()

predicted_pso6 = pso_kmeans6.predict(cluster6)

print('Silhouette:', silhouette_score(cluster6, predicted_pso6))
print('SSE:', calc_sse(centroids=pso6.gbest_centroids,
                       data=cluster6,
                       labels=predicted_pso6))


frame6 = pd.DataFrame(cluster6)
frame6['cluster'] = predicted_pso6
frame6['cluster'].value_counts()

predicted_pso6
```

```
# Pemetaan warna klaster
color_map = {0: '#4B0082', 1: '#FFD700', 2: '#ADFF2F', 3: '#20B2AA',
4: '#DA70D6', 5:'#9370DB'}  # Atur warna untuk masing-masing
klaster

fig, ax = plt.subplots(figsize=(15, 8))

# Jittering pada posisi titik data
jitter = 2  # Besar jittering, sesuaikan sesuai kebutuhan
jittered_x = [x + random.uniform(-jitter, jitter) for x in
data_cluster6.iloc[:, 0]]
jittered_y = [y + random.uniform(-jitter, jitter) for y in
data_cluster6.iloc[:, 1]]
scatter = ax.scatter(jittered_x, jittered_y, c=[color_map[i]
for i in predicted_pso6])

# Scatter plot dengan indeks dan keterangan klaster pada
setiap titik data klaster
texts = []
for i, (x, y, alamat) in enumerate(zip(jittered_x, jittered_y,
data_cluster5.index)):
    texts.append(ax.text(x, y, f'{alamat}', ha='center', va='bottom'))

# Menyesuaikan posisi tulisan agar tidak bertumpuk
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='black'))

# Menghitung dan menampilkan nilai centroid di tengah setiap
klaster
centroids = kmeans.cluster_centers_
for i, centroid in enumerate(centroids):
    cluster_data = data_cluster6[predicted_pso6 == i]
    centroid_x = np.mean(cluster_data.iloc[:, 0])
    centroid_y = np.mean(cluster_data.iloc[:, 1])
    plt.scatter(centroid_x, centroid_y, marker='o', color='red',
s=200)
    plt.annotate(f'Centroid {i+1}', (centroid_x, centroid_y),
textcoords="offset points", xytext=(0, 10), ha='center', color='black',
weight='bold')

plt.xlabel('X1', fontsize=18)
plt.ylabel('X2', fontsize=16)
plt.title('Hasil Clustering')

# Menampilkan keterangan warna
legend_labels = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor=color_map[i], markersize=10)
                 for i in range(len(color_map))]
legend_texts = [f'Cluster {i+1}' for i in range(len(color_map))]
plt.legend(legend_labels, legend_texts, loc='upper center',
bbox_to_anchor=(0.5, -0.1), ncol=len(color_map))

plt.show()
```

```
#evaluasi sse dan silhoute kmeans dengan 20 iterasi
kmeanspp = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    kmean_rep = KMeans(n_cluster=6, init_pp=True)
    kmean_rep.fit(cluster6)
    predicted_kmean_rep = kmean_rep.predict(cluster6)
    silhouette = silhouette_score(cluster6,
                                  predicted_kmean_rep)
    sse = kmean_rep.SSE
    dbi = davies_bouldin_score(cluster6, predicted_kmean_rep)
    kmeanspp['silhouette'].append(silhouette)
    kmeanspp['sse'].append(sse)
    kmeanspp['dbi'].append(dbi)

#evaluasi sse dan silhoute pso-kmeans dengan 20 iterasi
pso_hybrid = {
    'silhouette': [],
    'sse' : [],
    'dbi': [],
}
for _ in range(20):
    pso_rep2 = ParticleSwarmOptimizedClustering(n_cluster=6,
                                                n_particles=30,
                                                data=cluster5,
                                                hybrid=True,
                                                max_iter=2000,

print_debug=2000)
    pso_rep2.run()
    pso_kmeans2 = KMeans(n_cluster=6 ,
                         init_pp=False,
                         seed=144)
    pso_kmeans2.centroid = pso_rep2.gbest_centroids.copy()
    predicted_pso_rep2 = pso_kmeans2.predict(cluster6)

    silhouette = silhouette_score(cluster6,
                                  predicted_pso_rep2)
    sse = calc_sse(centroids=pso_rep2.gbest_centroids,
                   data=cluster6,
                   labels=predicted_pso_rep2)
    dbi = davies_bouldin_score(cluster6, predicted_pso_rep2)

    pso_hybrid['silhouette'].append(silhouette)
    pso_hybrid['sse'].append(sse)
    pso_hybrid['dbi'].append(dbi)

evaluasi = {
    'method' : ['K-Means', 'PSO Kmeans'],
    'sse_mean' : [
        np.around(np.mean(kmeanspp['sse']), decimals=10),
        np.around(np.mean(pso_hybrid['sse']), decimals=10),
```

```
        ],
        'silhouette_mean' : [
            np.around(np.mean(kmeanspp['silhouette']), decimals=10),
            np.around(np.mean(pso_hybrid['silhouette']), decimals=10),
        ],
        'dbi_mean': [
            np.around(np.mean(pso_hybrid['dbi']), decimals=10),
            np.around(np.mean(kmeanspp['dbi']), decimals=10),
        ]
    }


    eval = pd.DataFrame.from_dict(evaluasi)
    eval

    # Menghitung dan menampilkan nilai rentang usia di tengah
    setiap klaster
    age_ranges = []
    for i in range(6):
        cluster_data = data_cluster6[predicted_pso6 == i]
        min_age = cluster_data["Umur_Tifoid"].min()
        max_age = cluster_data["Umur_Tifoid"].max()

        # Penanganan nilai 0 atau NaN
        if np.isnan(max_age):
            age_range = f"{int(min_age)}+"
        else:
            if np.isnan(min_age):
                age_range = f"0-{int(max_age)}"
            else:
                age_range = f'{int(min_age)}-{int(max_age)}'

        age_ranges.append(age_range)

    # Mengganti semua nilai pada setiap klaster menjadi rentang
    usia
    for i in range(6):
        data_cluster6.loc[predicted_pso6 == i, 'Rentang_Usia_Tifoid'] =
    age_ranges[i]

    data_cluster6 = data_cluster6.drop(['Umur_Tifoid'], axis=1)

    #create new column for cluster labels associated with each
    subject
    data_cluster6['labels'] = predicted_pso6
    data_cluster6['Segment'] = data_cluster6['labels'].map({0: 'First', 1:
    'Second', 2: 'Third', 3: 'Fourth', 4: 'Fifth', 5: 'Sistxh'})


    #Order the cluster
    data_cluster6['Segment'] = data_cluster6['Segment'].astype('category')
    data_cluster6['Segment'] =
    data_cluster6['Segment'].cat.reorder_categories(['First', 'Second',
    'Third', 'Fourth', 'Fifth', 'Sistxh'])
```

```
# ==== CREATE DICTIONARY FOR OBJECT AND NUMERICAL COLUMNS
====#
dict_cat = {}
dict_num = {}
for cat in data_cluster6.select_dtypes('object'):
 dict_cat[cat] = lambda x: x.value_counts().index[0]
for num in data_cluster6.select_dtypes(['int64', 'float64']):
  if (num == 'Total'):
    continue
  dict_num[num] = ['mean']
# ============ CREATE TABLE FOR EACH CLUSTER ============#
data_cluster6.rename(columns={'labels': 'Total'}, inplace=True)
data_percluster6 = data_cluster6.groupby('Segment').agg({
 'Total': 'count',
 **dict_num,
 **dict_cat
}).T

data_cluster6['coord'] = coord6

data_cluster6 = data_cluster6.T
data_percluster6.to_json(r'cluster6_result.json')
data_cluster6.to_json(r'cluster6_df.json')
```

LEMBAR PERBAIKAN SKRIPSI

"OPTIMASI K-MEANS CLUSTERING PADA PENYAKIT MENULAR DENGAN ALGORITMA PARTICLE SWARM OPTIMIZATION"

OLEH:

ANDI RUSMIATI
D121191079

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 9 Juni 2023.
Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Elly Warni, S.T., M.T.	
Sekretaris	Ir. Christoforus Yohannes, M.T.	
Anggota	Dr. Ir. Zahir Zainuddin, M.Sc	
	Tyanita Puti Marindah Wardani, S.T., M.Inf.	

Persetujuan perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Elly Warni, S.T., M.T.	
II	Ir. Christoforus Yohannes, M.T.	

KEMENTERIAN PENDIDIKAN , KEBUDAYAAN
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INFORMATIKA
Kampus FakultasTeknikUnhas, Jl. PorosMalino, Gowa
http://eng.unhas.ac.id/informatika, Email : informatika@unhas.ac.id

DAFTAR HADIR SEMINAR HASIL

Nama/Stambuk : 1. Andi Rusmiati D121191079

Judul Skipsi/T.A : **"Optimasi K-means Clustering pada Penyakit Menular dengan Algoritma Particle Swarm Optimization"**

Hari/Tanggal : Rabu, 17 Mei 2023
Jam : 13.30 Wita – Selesai
Tempat : Ruang Lab. CBS Departeman Teknik Informatika Gowa

No.	Jabatan	Nama Dosen	Tanda Tangan
L.	Pembimbing I	1. Elly Warni,ST.,M.T	1.....
	Pembimbing II	2. Ir. Christoforus Yohannes,M.T	2 .....
II.	Anggota Penguji	3. Dr. Ir. Zahir Zainuddin, M.Sc	3 .....
		4. Tyanita Puti Marindah Wardhani,ST.,M.Inf	4 .....

PANITIA UJIAN

Ketua,

Elly Warni,ST.,M.T

Sekretaris,

Ir. Christoforus Yohannes,M.T

KEMENTERIAN PENDIDIKAN , KEBUDAYAAN
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INFORMATIKA
Kampus FakultasTeknikUnhas, Jl. PorosMalino, Gowa
http://eng.unhas.ac.id/informatika, Email : informatika@unhas.ac.id

BERITA ACARA SEMINAR HASIL

Pada hari ini **Rabu,** tanggal **17 Mei 2023** Pukul **13.30 WITA - Selesai** bertempat di **Ruang** Lab. **CBS Teknik Informatika,** telah dilaksanakan Seminar Hasil bagi Saudara :

Nama : Andi Rusmiati
No. Stambuk : D121191079
Fakultas/Departemen : Teknik/Teknik Informatika
Judul Skripsi : **"Optimasi K-means Clustering pada Penyakit Menular dengan Algoritma Particle Swarm Optimization"**

Yang dihadiri oleh Tim Penguji Seminar Hasil sebagai berikut :

No.	N a m a	Jabatan	Tanda tangan
1.	Elly Warni,ST.,M.T	Pemb I/Ketua	1.....
2.	Ir. Christoforus Yohannes,M.T	Pemb II/Sekretaris	2...
3.	Dr. Ir. Zahir Zainuddin, M.Sc	Anggota	3.....
4.	Tyanita Puti Marindah Wardhani,ST.,M.Inf	Anggota	4.....

Hasil keputusan Tim Penguji Seminar Hasil : **Lulus / Tidak lulus** dengan nilai angka ..... dan huruf .....

Makassar, 17 Mei 2023

Ketua/Sekretaris Panitia Ujian,

Elly Warni,ST.,M.T

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INFORMATIKA

Kampus Fakultas Teknik Unhas, Jl. Poros Malino, Gowa
http://eng.unhas.ac.id/informatika, Email : informatika@unhas.ac.id

Nomor : 416/UN4.7.7/TD.06/2023
Lamp : -
Hal : Penerbitan Surat Penugasan Panitia Seminar Hasil Strata Satu (S1)

Kepada Yth :

Wakil Dekan Bidang Akademik dan Kemahasiswaan
Fakultas Teknik Universitas Hasanuddin

Di-

Gowa

Dengan hormat,
Berdasarkan Persetujuan Pembimbing Mahasiswa, Bersama ini diusulkan susunan Panitia Seminar Hasil Strata Satu (S1) bagi mahasiswa Departemen Teknik Informatika Fakultas Teknik tersebut di bawah ini :

Nama / Stambuk : Andi Rusmiati D121191079
Judul TA : Optimasi K-means Clustering Pada Penyakit Menular Dengan Algoritma Particle Swarm Optimization

Dengan ini kami sampaikan Susunan Panitia Seminar Hasil Program Strata Satu (S1) Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin dengan susunan sebagai berikut :

Pembimbing I/ Ketua : 1. Elly Warni, ST.,M.T.
Pembimbing II / Sekretaris : 2. Ir. Christoforus Yohannes, M.T
Anggota : 3. Dr. Ir. Zahir Zainuddin, M.Sc.
4. Tyanita Puti Marindah Wardhani ST., M.Inf.

Untuk dapat diterbitkan surat penugasannya

Demikian penyampaian kami, atas perhatian dan kerjasamanya diucapkan terima kasih.

Gowa, 12 Mei 2023
Ketua Departemen Tek.Informatika,

Prof. Dr. Ir. Indrabayu.,ST, MT, M.Bus.Sys., IPM, ASEAN.Eng
Nip.19750716 200212 1 004

Tembusan :
1. Arsip

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
Poros Malino Km.6Bontomarannu(92172) Gowa, Sulawesi Selatan 92172, Sulawesi Selatan
Telp. (0411) 586015, 586262 Fax (0411) 586015
http://eng.unhas.ac.id, Email : teknik@unitas.ac.id

SURAT PENUGASAN
No. 9455 UN4.7.1 TD.06 2023

Dari : Dekan Fakultas Teknik Universitas Hasanuddin

Kepada : Mereka yang tercantum namanya dibawah ini

Isi : 1. **Bahwa berdasarkan peraturan Akademik Universitas Hasanuddin Tahun 2018 pasal 18 (SK.Rektor Unhas nomor : 2781/UN4.1/KEP/2018).** dengan ini menugaskan Saudara sebagai PANITIA SEMINAR HASIL Program Strata Satu (S1) Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin dengan susunan sebagai berikut :

Pembimbing I/ Ketua	**: 1. Elly Warni, ST.,M.T**
Pembimbing II / Sekretaris	**: 2. Ir. Christoforus Yohannes, M.T**
Anggota	**: 3. Dr. Ir. Zahir Zainuddin, M.Sc.**
	4. Tyanita Puti Marindah Wardhani ST., M.Inf.

Untuk menguji bagi mahasiswa tersebut dibawah ini :

Nama NIM	: Andi Rusmiati	D121191079
Program Studi	: Teknik Informatika	
Judul thesis Skrips	: **Optimasi K-means Clustering Pada Penyakit Menular Dengan Algoritma Particle Swarm Optimization**	

2. Waktu seminar ditetapkan oleh Panitia Seminar Hasil Program Strata Satu (S1)
3. Agar Surat Penugasan ini dilaksanakan sebaik-baiknya dengan penuh rasa tanggung jawab.
4. Surat penugasa ini berlaku sejak tanggal ditetapkan sampai dengan berakhirnya seminar tersebut dengan ketentuan bahwa segala sesuatunya akan ditinjau dan diperbaiki sebagaimana mestinya apabila dikemudia hari terdapat kekeliruan dalam keputusan ini.

Ditetapkan di Gowa
Pada tanggal 12 Mei 2023
a.n. Dekan,
Wakil Dekan Bidang Akademik dan Kemahasiswaan
Fakultas Teknik Unhas

Dr. Amil Ahmad Ilham, ST., M.IT
NIP. 197310101998021001

Tembusan :
1. Dekan Fak. Teknik Unhas
2. Ketua Departemen Teknik Informatika FT-UH
3. Mahasiswa yang bersangkutan

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INFORMATIKA
Kampus FakultasTeknikUnhas, Jl. PorosMalino, Gowa
http://eng.unhas.ac.id/informatika, Email : informatika@unhas.ac.id

DAFTAR HADIR UJIAN SKRIPSI MAHASISWA FAKULTAS TEKNIK UNHAS

Nama/Stambuk : 1. Andi Rusmiati D121191079

Judul Skipsi/T.A : **"Optimasi K-means Clustering pada Penyakit Menular dengan Algoritma Particle Swarm Optimization"**

Hari/Tanggal : Jumat, 9 Juni 2023
Jam : 13.00 Wita – Selesai
Tempat : Ruang Lab.Komputer Departemen Teknik Informatika Gowa

No.	Jabatan	Nama Dosen	Tanda Tangan
L.	Pembimbing I	1. Elly Warni,ST.,M.T	1.....
	Pembimbing II	2. Ir. Christoforus Yohannes,M.T	2 ....
II.	Anggota Penguji	3. Dr. Ir. Zahir Zainuddin, M.Sc	3 .....
		4. Tyanita Puti Marindah Wardhani,ST.,M.Inf	4 .....

PANITIA UJIAN

Ketua,

Elly Warni,ST.,M.T

Sekretaris,

Ir. Christoforus Yohannes,M.T

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INFORMATIKA
Kampus FakultasTeknikUnhas, Jl. PorosMalino, Gowa
http://eng.unhas.ac.id/informatika, Email : informatika@unhas.ac.id

BERITA ACARA UJIAN SKRIPSI

Pada hari ini **Jumat,** tanggal **9 Juni 2023** Pukul **13.00 WITA - Selesai** bertempat di **Ruang Lab.Komputer Departemen Teknik Informatika Gowa,** telah dilaksanakan Ujian Skripsi bagi Saudara :

Nama : Andi Rusmiati
No. Stambuk : D121191079
Fakultas/Departemen : Teknik/Teknik Informatika
Judul Skripsi : **"Optimasi K-means Clustering pada Penyakit Menular dengan Algoritma Particle Swarm Optimization"**

Yang dihadiri oleh Tim Penguji Ujian Skripsi sebagai berikut :

No.	Nama	Jabatan	Tanda tangan
1.	Elly Warni,ST.,M.T	Pemb I/Ketua	1.
2.	Ir. Christoforus Yohannes,M.T	Pemb II/Sekretaris	2.
3.	Dr. Ir. Zahir Zainuddin, M.Sc	Anggota	3.
4.	Tyanita Puti Marindah Wardhani,ST.,M.Inf	Anggota	4.

Hasil keputusan Tim Penguji Ujian Skripsi/Tugas Akhir : **Lulus / Tidak lulus** dengan nilai angka .....87..... dan huruf ...A.....

Gowa, 9 Juni 2023

Ketua/Sekretaris Panitia Ujian,

Elly Warni,ST.,M.T

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INFORMATIKA
Kampus Fakultas Teknik Unhas, Jl. Poros Malino, Gowa
http://eng.unhas.ac.id/informatika, Email : informatika@unhas.ac.id

Gowa, 7 Juni 2023

Nomor : 471/UN4.7.7.1/TD.06/2023
Lamp : -
Hal : Usulan Susunan Panitia Ujian Sarjana

Yth. : Bapak Wakil Dekan Bidang Akademik dan Kemahasiswaan
Fakultas Teknik Unhas
Di
Gowa

Dalam rangka penyelesaian studi pada Departemen Teknik Informatika Fakultas Teknik Unhas, bersama ini kami usulkan susunan Panitia Ujian Sarjana Program Strata Satu (S1) bagi mahasiswa Departemen Teknik Informatika Fakultas Teknik Uniersitas Hasanuddin atas nama :

Pembimbing I / Ketua : 1. Elly Warni, ST.,M.T
Pembimbing II / Sekretaris : 2. Ir. Christoforus Yohannes, M.T
Anggota : 3. Dr. Ir. Zahir Zainuddin, M.Sc
4. Tyanita Puti Marindah Wardhani ST., M.Inf.

Untuk Bertugas sebagai Penguji/ Penanggap Ujian Sarjana bagi Mahasiswa :

Nama : Andi Rusmiati
Stambuk : D121 19 1079

Dengan Judul Skripsi
" Optimasi K-Means Clustering pada Penyakit Menular dengan Algoritma Particle Swarm Optimization "

Pada :
Hari/Tanggal : ~~Rabu~~ Jumat, 9 Juni 2023
Jam : 08.00 Wita - Selesai
Tempat : Ruang Sidang Lab. Komputer

Demikian penyampaian kami, atas perhatiannya diucapkan terimah kasih.

Ketua Departemen Tek.Informatika,

Prof. Dr. Ir. Indrabayu.,ST, MT, M.Bus.Sys., IPM, ASEAN.Eng
Nip.197507016 200212 1 004

Tembusan :
1. Arsip

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET DAN TEKNOLOGI
UNIVERSITAS HASANUDDIN
FAKULTAS TEKNIK
Poros Malino Km.6Bontomarannu(92172) Gowa, Sulawesi Selatan 92172, Sulawesi Selatan
Telp. (0411) 586015, 586262 Fax (0411) 586015
http://eng.unhas.ac.id, Email : teknik@unhas.ac.id

SURAT PENUGASAN

No. *11861/UN4.7 1/TD.06/2023*

Dari : Dekan Fakultas Teknik Universitas Hasanuddin.

Kepada. : Mereka yang tercantum namanya di bawah ini.

Isi : 1. Bahwa berdasarkan peraturan Akademik Universitas Hasanuddin Tahun 2018 pasal 19 (SK.Rektor Unhas nomor : 2781/UN4.1/KEP/2018), dengan ini menugaskan Saudara sebagai PANITIA UJIAN SARJANA Program Strata Satu (S1) Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin dengan susunan sebagai berikut :

Pembimbing I / Ketua : 1. Elly Warni, ST., M.T
Pembimbing II / Sekretaris : 2. Ir. Christoforus Yohannes, M.T
Anggota : 3. Dr. Ir. Zahir Zainuddin, M.Sc
4. Tyanita Puti Marindah Wardhani ST., M.Inf.

untuk menguji bagi mahasiswa tersebut di bawah ini :

Nama/NIM : Andi Rusmiati D121191079
Program Studi : Teknik Informatika
Judul Thesis/Skripsi : Optimasi K-Means Clustering pada Penyakit Menular dengan Algoritma Particle Swarm Optimization

2. Waktu Ujian ditetapkan oleh Panitia Ujian Sarjana Program Strata Satu (S1).
3. Agar Surat penugasan ini dilaksanakan sebaik-baiknya dengan penuh rasa tanggung jawab.
4. Surat penugasan ini berlaku sejak tanggal ditetapkan sampai dengan berakhirnya Ujian Sarjana tersebut, dengan ketentuan bahwa segala sesuatunya akan ditinjau dan diperbaiki sebagaimana mestinya apabila dikemudian hari ternyata terdapat kekeliruan dalam keputusan ini.

Ditetapkan di Gowa,
Pada tanggal 7 Juni 2023
a.n. Dekan
Wakil Dekan Bidang Akademik dan Kemahasiswaan
Fakultas Teknik Unhas

Dr. Amil Ahmad Ilham, ST., M.IT
NIP.197310101998021001

Tembusan :
1. Dekan Fak. Teknik Unhas
2. Ketua Departemen Teknik Informatika FT-UH
3. Kasubag. Umum dan Perlengkapan FT-UH