LAMPIRAN

Lampiran 1. *Source Code* Program

```
# menghapus Outliers dengan IQR method
plt.figure(figsize=(8, 6))
plt.boxplot(data['penghasilan_ortu'])
plt.title('Box Plot Fitur Penghasilan Orang Tua')
plt.xlabel('penghasilan_ortu')
plt.ylabel('Value')
plt.show()

# Menghitung Q1, Q3, dan IQR
Q1 = np.percentile(data.penghasilan_ortu, 35)
Q3 = np.percentile(data.penghasilan_ortu, 65)
IQR = Q3 - Q1

# Menghitung batas atas dan batas bawah untuk outlier
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Hapus outlier berdasarkan IQR
data_clean = data[(data.penghasilan_ortu >= lower_bound) &
(data.penghasilan_ortu  <= upper_bound)]

# drop fitur kode mahasiswa
data_cleaned = data_clean.drop(['kode_mhs'], axis = 1)
```

L1 Source Code *Outliers Removing*

```
# Import MinMaxScaler dari library sklearn.preprocessing
from sklearn.preprocessing import MinMaxScale

# Lakukan Min-Max normalization ke numerical columns
numerical_columns = ['jmlh_semester','ips_1', 'ips_2', 'ips_3',
'ips_4', 'ips_5', 'ips_6', 'ipk_6', 'sks_1', 'sks_2', 'sks_3',
'sks_4', 'sks_5', 'sks_6', 'sks_kom_6', 'nominal_ukt',
'penghasilan_ortu', 'jumlah_saudara',
'beasiswa','non_beasiswa', 'buleleng', 'luar_buleleng',
'luar_bali']
scaler = MinMaxScaler()
data_cleaned[numerical_columns] =
scaler.fit_transform(data_cleaned[numerical_columns])
```

L2 *Source Code Data Normalization*

```
# Import PCA dari sklearn.decomposition
from sklearn.decomposition import PCA

# Lakukan PCA
n_components = 2  # Set the number of principal components
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(data_cleaned)

# Hasil of PCA
print("PCA Result:")
print(X_pca)
```

L3 *Source Code* PCA dalam *Feature selection*

```
# Inisialisasi list untuk menyimpan nilai SSE

sse = []


for k in range(1, 11):

    kmeans = KMeans(n_clusters=k, random_state=0)

    kmeans.fit(X_pca)

    sse.append(kmeans.inertia_)


# Plot Elbow dengan SSE

plt.figure(figsize=(8, 6))

plt.plot(range(1, 11), sse, marker='o')

plt.xlabel('Jumlah Klaster')

plt.ylabel('SSE (Within-Cluster Sum of Squares)')

plt.title('Elbow Method for Optimal K')

plt.grid(True)

plt.show()

# mencari elbow point using KneeLocator

kneedle = KneeLocator(range(1, 11), sse, curve='convex',
direction='decreasing')

knee_point = kneedle.knee


print("Knee Point (Optimal Number of Clusters):", knee_point)
```

L4 *Source Code* metode *Elbow*

```
import math
import csv
# Set a random seed
np.random.seed(50)

#euclidean_distance function
def euclidean_distance(a, b):
    if len(a) != len(b):
       raise ValueError("the points should have the same
       dimensions")
    sum_sq = 0.0
    for i in range(len(a)):
        sum_sq += (a[i] - b[i]) ** 2
    return math.sqrt(sum_sq)

def kmeans_clustering(data, k, max_iterations=100):
    n_samples, n_features = data.shape
    centroids_idx = np.random.choice(n_samples, size=k,
     replace=False)
    centroids = data[centroids_idx]
    labels = np.zeros(n_samples)
    for _ in range(max_iterations):
        # Assign each sample to the nearest centroid
        for i in range(n_samples):
            distances = [euclidean_distance(data[i], centroid)
            for centroid in centroids]
            labels[i] = np.argmin(distances) #minimum distance
        # Update centroids
        new_centroids = np.array([data[labels ==
        j].mean(axis=0) for j in range(k)])
        # Check for convergence
        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids

    return labels, centroids
```

L5 *Source Code* K-Means *Clustering*

```
# melakukan K-means clustering
labels, centroids = k_means_clustering(X_pca, k=3)

# Memvisualisaikan hasil K-means clustering
def plot_clusters(X, labels, centroids):
    plt.figure(figsize=(8, 6))
    for i in range(len(np.unique(labels))):
        plt.scatter(X[labels == i, 0], X[labels == i, 1],
         label=f'Cluster {i+1}')
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='X',
     s=10, c='black', label='Centroids')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('K-means Clustering')
    plt.legend()
    # plt.grid(True)
    plt.show()

# Plot
plot_clusters(X_pca, labels, centroids)
```

L5 *Source Code* K-Means *Clustering*

```
from sklearn.metrics import silhouette_score

# Menghitung Silhouette Coefficient
silhouette_avg_kmeans = silhouette_score(X_pca, labels)

# Print Silhouette Coefficient
print("Silhouette Coefficient:", silhouette_avg_kmeans)
```

L6 *Source Code Validasi* K-Means *Clustering*

```
import numpy as np
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt

k = 4
nbrs = NearestNeighbors(n_neighbors=k).fit(X_pca)
distances, indices = nbrs.kneighbors(X_pca)

# Urutkan jarak-jarak untuk setiap titik untuk mendapatkan
jarak tetangga ke-4
sorted_distances = np.sort(distances[:, -1])

# Plot jarak-jarak yang diurutkan untuk memvisualisasikan
grafik k-dist
plt.plot(np.arange(1, len(sorted_distances) + 1),
sorted_distances)
plt.xlabel('Indeks Titik Data')
plt.ylabel('Nilai 4-Dist')

# Temukan titik ambang (titik knee) di mana jarak mulai
meningkat secara signifikan
kneedle = KneeLocator(np.arange(1, len(sorted_distances) + 1),
sorted_distances, curve='convex', direction='increasing')
threshold_point = kneedle.knee

# Tambahkan garis vertikal pada titik ambang untuk menunjukkan
knee
plt.axvline(x=threshold_point, color='red', linestyle='--',
label='Titik Knee')
plt.legend()

print("Indeks Titik Ambang:", threshold_point)
print("Jarak Ambang (Nilai 4-Dist):",
sorted_distances[threshold_point])

plt.show()
```

L7 *Source Code* KNN 4-dist Untuk mencari Nilai Eps

```
#membuat euclidean_distance function
def euclidean_distance(a, b):
    if len(a) != len(b):
        raise ValueError("the points should have the same
        dimensions")
    sum_sq = 0.0
    for i in range(len(a)):
        sum_sq += (a[i] - b[i]) ** 2
    return math.sqrt(sum_sq)
# membuat fungsi region untuk menemukan semua titik dalam jarak
epsilon (eps) yang diberikan
def region(data, indeks_titik, eps):
    neighbors = []
    for i in range(len(data)):
        if jarak_euclidean(data[indeks_titik], data[i]) <= eps:
            neighbors.append(i)
    return neighbors
# dbscan function
def dbscan_clustering(data, eps, min_points):
    n_samples = len(data) #create n_sample var to determine the
     number of iteration
    cluster_label = 0
    labels = np.zeros(n_samples, dtype=int)
    # Loop melalui semua data point
    for i in range(n_samples):
        if labels[i] != 0:
            continue
        # memanggil region query untuk menemukan titik-titik
           tetangga
        neighbors = region(data, i, eps)
        if len(neighbors) < min_points:
            labels[i] = -1  # Noise label
        else
            cluster_label += 1
            expand_cluster(data, labels, i, neighbors,
cluster_label, eps, min_points)
    return labels
```

L8 *Source Code* DBSCAN *Clustering*

```
# Function untuk memperluas cluster and memberikann cluster
labels to connected data points
def expand_cluster(data, labels, point_index, neighbors,
cluster_label, eps, min_points):
    labels[point_index] = cluster_label
    i = 0

    while i < len(neighbors):
        current_point = neighbors[i]
        if labels[current_point] == -1:
            labels[current_point] = cluster_label
        elif labels[current_point] == 0:
            labels[current_point] = cluster_label
            current_neighbors = region(data, current_point,eps)
            if len(current_neighbors) >= min_points:
                neighbors = neighbors + current_neighbors
        i += 1

# melakukan DBSCAN clustering
epsilon = 0.02
min_points = 4
dbscan_labels = dbscan_clustering(X_pca, epsilon, min_points)

# memvisualisasikan hasil cluster
unique_labels = np.unique(dbscan_labels)
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
for label, color in zip(unique_labels, colors):
    if label == -1:
        color = 'black'
    x = X_pca[dbscan_labels == label, 0]
    y = X_pca[dbscan_labels == label, 1]
    plt.scatter(x, y, color=color, label=f'Cluster {label}')
plt.title('DBSCAN Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(loc='best')
plt.show()
```

L8 *Source Code* DBSCAN *Clustering*

```
# menghitung Silhouette Coefficient DBSCAN
silhouette_avg_dbscan = silhouette_score(X_pca, dbscan_labels)

print("Silhouette Coefficient:", silhouette_avg_dbscan)
```

L9 *Source Code Validasi* DBSCAN *Clustering*

RIWAYAT HIDUP

Gusti Ayu Gita Mulya Sari lahir di Seririt pada tanggal 26 November Tahun 20000. Penulis lahir dari pasangan suami istri, Bapak Gusti Ketut Muliarta dan ibu Luh Sariani. Penulis berkebangsaan Indonesia dan beragama Hindu. Kini penulis beralamat di Jalan Sudirman no 38, Seririt, Buleleng, Bali. Penulis menyelesaikan pendidikan dasar di SD Negeri 1 Seririt dan lulus pada 2013. Kemudian penulis melanjutkan di SMP Negeri 1 Seririt dan lulus pada tahun 2016. Pada tahun 2019, penulis lulus dari SMA Negeri 1 Singaraja jurusan MIPA. Penulis terdaftar sebagai mahasiswa Program Studi S1 Ilmu Komputer di Universitas Pendidikan Ganesha