Nama : Yulitta Irvine

NIM : 225314134

LISTING CODE

**Data Preparatation**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  import numpy as np  from sklearn import metrics  from sklearn.preprocessing import MinMaxScaler  from sklearn.cluster import KMeans  from kneed import KneeLocator  from sklearn.cluster import AgglomerativeClustering  from sklearn.metrics import silhouette\_score  from sklearn.cluster import DBSCAN  from sklearn.neighbors import NearestNeighbors  from yellowbrick.cluster import SilhouetteVisualizer  import warnings  warnings.filterwarnings("ignore") | Mengimport libraries 6yang dibutuhkan |
| df = pd.read\_csv("Wholesale customers data.csv")  df | Membaca dataset Wholesale customers data.csv dan disimpan dalam dataframe df |

**Data Wrangling**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| print(df.columns)  print(df.info) | Mencetak kolom dan informasi tiep data pada dataframe df |
| # checking the shape of the data  print(df.shape)  # check for missing values  df.isnull().sum() | Mencetak ukuran df |
| # Cari duplikasi data dan perbaiki  print("\nJumlah duplikasi sebelum perbaikan:", df.duplicated().sum())  df.drop\_duplicates(inplace=True) | Mencetak duplikasi sebelu perbaikan |

**Data Preprocessing**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| # Inisialisasi objek MinMaxScaler  scaler = MinMaxScaler()  # Mengambil hanya fitur-fitur numerik untuk dinormalisasi kecuali target data  numeric\_features = df[['Fresh', 'Milk', 'Grocery', 'Frozen',  'Detergents\_Paper', 'Delicassen']]  # Melakukan normalisasi pada fitur-fitur numerik  normalized\_features = scaler.fit\_transform(numeric\_features)  # Membuat DataFrame baru dari fitur-fitur yang sudah dinormalisasi  df\_normalized = pd.DataFrame(normalized\_features, columns=numeric\_features.columns)  # Menampilkan hasil normalisasi  print("Hasil Normalisasi:")  print(df\_normalized.head()) | Menormalisasikan df dengan minmac scaler untuk data yang bertipe numerik. Lalu menampilkan hasil normalisasi dataset. |
| df\_normalized.head() | Menampilkan 5 data teratas dari data yang sudah dinormalisasi |
| df\_normalized.info() | Menampilkan informasi dataset terkait tipe data |
| df\_normalized.describe() | Menampilkan rangkuman statistic dari df\_normalized. |

**Exploratory Data Analysis**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| # Set figure size  plt.figure(figsize=(20, 8))  # Iterate over columns and plot KDE  data = df\_normalized  for col in data.columns.values:  sns.kdeplot(data[col], label=col)  # Add legend outside the plot  plt.legend(loc=”upper center”, bbox\_to\_anchor=(0.5, 1.05), ncol=2, prop={‘size’: 15})  # Show the plot  plt.show() | Kode untuk memplotkan hasil normalisasi |
| df\_normalized[‘Channel’] = df[‘Channel’] | Menambahkan chanel ke df yang sudah dinormalisasi |
| # bivariate & multivariate analysis  fig, axes = plt.subplots(3, 2, figsize=(18, 18))  fig.suptitle(‘Boxplot for all categorical variables in the dataset’)  sns.boxplot(ax=axes[0, 0], x=’Channel’, y=’Fresh’,data=df\_normalized, color=’blue’, palette=”Set2”)  sns.boxplot(ax=axes[0, 1], x=’Channel’, y=’Milk’,data=df\_normalized, color=’blue’, palette=”Set2”)  sns.boxplot(ax=axes[1, 0], x=’Channel’, y=’Grocery’,data=df\_normalized, color=’blue’, palette=”Set2”)  sns.boxplot(ax=axes[1, 1], x=’Channel’, y=’Frozen’,data=df\_normalized, color=’blue’, palette=”Set2”)  sns.boxplot(ax=axes[2, 0], x=’Channel’, y=’Detergents\_Paper’,data=df\_normalized, color=’blue’, palette=”Set2”)  sns.boxplot(ax=axes[2, 1], x=’Channel’, y=’Delicassen’,data=df\_normalized, color=’blue’, palette=”Set2”)  axes[1, 1].tick\_params(labelrotation=45)  axes[2, 0].tick\_params(labelrotation=90)  axes[2, 1].tick\_params(labelrotation=90)  plt.show() | Kode untukbivariate dan multivariate analysis, bivariate untuk kolom kategorical dan multivariate untuk kolom numerik.  Diplotkan seluruh nya dalam bentuk boxplot per masing-masing chanel. Ditampilkan dengan layout 2x3. |
| df\_normalized = df\_normalized.drop(‘Channel’, axis = 1) | Mendropkan kolom channel pada df\_normalized untuk proses numerik |
| # Display the correlation heatmap  corr = df\_normalized.corr()  plt.figure(figsize = (10,5))  ax = sns.heatmap(corr, annot=True)  ax.legend(loc=0, prop={‘size’: 15}) | Menampilkan heatmap untuk df\_normalized yang sudah di drop kolom chanel nya |
| # Produce a scatter matrix for each pair of features in the data  g = sns.pairplot(df\_normalized, diag\_kind = ‘kde’) | Membuat pairplot atas variabel dari df\_normalized |
| # remove kolom yang memiliki outliers  selected\_data = df\_normalized[df\_normalized.isna().any(axis=1)]  # Fungsi untuk menghapus outlier menggunakan metode IQR  def remove\_outliers(column):  Q1 = column.quantile(0.25)  Q3 = column.quantile(0.75)  IQR = Q3 – Q1  lower\_bound = Q1 - 1.5 \* IQR  upper\_bound = Q3 + 1.5 \* IQR  return column[(column >= lower\_bound) & (column <= upper\_bound)]  # Loop melalui setiap kolom yang dipilih dan hapus outlier  for col in selected\_data:  df\_normalized[col] = remove\_outliers(df\_normalized[col]) | # mengahpus kolom yang memiliki outliers. |
| nan\_values = df\_normalized.isna().sum()  print("\nJumlah nilai NaN dalam setiap kolom:")  print(nan\_values) | Mencetak nilai nan value dalam setiap kolom |
| # impute missing values  df\_imputed = df\_normalized.fillna(df\_normalized.mean())  print(df\_imputed.isna().sum()) | Mengisikan nilai missing value dengan nilai mean lalu menyimpannya dalam df\_imputed, lalu mencetak df\_imputed |
| df\_imputed.describe() | Mencetak rangkuman statistic dari df\_imputed |

**Principal Component Analysis (PCA)**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| from sklearn.decomposition import PCA  pca = PCA() | Memanggil methode PCA() |
| pcomps = pca.fit\_transform(df\_imputed)  pcomps.shape | me  Mencetak ukuran data pcomps |
| eigen\_values = pca.explained\_variance\_  eigen\_vector = pca.components\_  print("Eigen values : \n",eigen\_values)  print("Eigen vector : \n",eigen\_vector) | Mencari eigen value dan eigen vektor |
| # memilih komponen utama PCA berdasarkan eigen value  total = sum(eigen\_values)  print("Sum of Eigen Values : ",total)  variance\_eigen\_value = [i\*100/total for i in eigen\_values]  print("Variance Eigen Values : \n",variance\_eigen\_value)  cumulative\_variance\_eigen\_value = np.cumsum(variance\_eigen\_value)  print("Cumulative Variance Eigen Value : \n",cumulative\_variance\_eigen\_value) | Memilih komponen utama PCA berdasarkan eigen values lalu mencetak total dan variansnya, kemuadian mencetak total varians nya |
| # validation  pcomps\_res = np.dot(df\_imputed,eigen\_vector.T)  pcomps\_res | Memvalidasi hasil pca |
| pca\_df = pd.DataFrame(pcomps,columns=["PC1","PC2","PC3","PC4","PC5","PC6"])  pca\_df.head() | Membuat dataframe Bernama pca\_df yang hasil PCA1, PC2, PC3, PC4, PC5 |
| # Create factor analysis object and perform factor analysis  from factor\_analyzer.factor\_analyzer import calculate\_kmo  kmo\_all,kmo\_model=calculate\_kmo(df)  kmo\_model | Membuat factor analysis untuk mendapatkan model kmo |
| from factor\_analyzer import FactorAnalyzer  # Create factor analysis object and perform factor analysis  fa = FactorAnalyzer()  fa.fit(df\_imputed, 6)  # Check Eigenvalues  ev, v = fa.get\_eigenvalues()  ev | Membuat factor analisis dengan methode FactorAnalyzer, lalu menyesuaikan df\_imputed ke model fa |
| fa.loadings\_ | Memanggil nilai loading untuk fa |
| # Create scree plot using matplotlib  plt.scatter(range(1,df\_imputed.shape[1]+1),ev)  plt.plot(range(1,df\_imputed.shape[1]+1),ev)  plt.title('Scree Plot')  plt.xlabel('Factors')  plt.ylabel('Eigenvalue')  plt.grid()  plt.show() | Membuat scree plot dengan matplotlib sumbu x untuk factors dan sumbu y untuk nilai eigenvalue |
| final\_pca = pca\_df.iloc[:,:2]  final\_pca.head() | Menyimpan 2 kolom pertama di pca\_df ke dalam final\_pca, lalu memanggil 5 data teratas dengan methode head |

**K-MEANS clustering**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| # menentukan klastering dengan elbow  # Elbow Methode - Inertias plot  wcss = []  # melakukan perulangan untuk menghitung inertia untuk setiap k  for k in range(2,15):  model = KMeans(n\_clusters=k, random\_state = 42)  model.fit(final\_pca)  wcss.append(model.inertia\_)  # membuat plot dengan plt untuk menampilkan grafik elbow untuk menemukan jumlah klaster yang paling maksimal  plt.plot(range(2,15), wcss, marker='o')  plt.title('Elbow method')  plt.xlabel('Number of clusters')  plt.ylabel('WCSS')  plt.show() | Mencari clustering terbaik dengan menggunakan wcss untuk mencari jumlah kuadrat jarak antara setiap titik dan pusat massa dalam sebuah cluster.  Dipilih range clusternya 2-14 dengan random state – 42, lalu memodelkan final\_pca dan menambahkan nilai model.inertia pada urutan terakhir dengan append.  Selanjutnya memvisualisasikan elbow dengan sumbu x adalah number of clusters dan y adalah nilai WCSS |
| kmeans\_kwargs = {"init" : 'random' ,  "n\_init" : 10 ,  "max\_iter" : 300 ,  "random\_state" : 42,}  error=[]  for k in range(2,15) :  km=KMeans(n\_clusters=k , \*\*kmeans\_kwargs)  km.fit(final\_pca)  error.append(km.inertia\_) | Membuat variabel kmeans\_kwargs yang menyimpan konfigurasi atau argument untuk algoritma K-Means di dalamnya terdapat parameter init untuk inisialisasi pusat cluster, percobaan inisialisasi (n\_init), max\_iter, dan random \_state.  Melakukan perulangan untuk menentukan menghitung nilai error nya dengan menambahkan nilai km\_inertia |
| #membuat objek KneeLocator dengan parameter yg ditentukan  kn = KneeLocator(range(2,15),  error,  curve='convex',  direction='decreasing') | Membuat objek KneeLocator untuk menentukan nilai elbow yang paling optimal |
| #Mendapatkan jumlah cluster yang dianggap optimal  kn.elbow | Memanggil hasil kn.elbow |
| km = KMeans(n\_clusters = 3, random\_state = 42)  cluster = km.fit\_predict(final\_pca) | Melakukan permodelan dataframe final\_pca dengan nilai \_n\_clusters = 3 |
| # Menambahkan kolom cluster ke dataset  final\_pca['cluster'] = cluster | Menambahkan kolom cluster ke dataset final\_pca |
| # Inertia (Sum of squared distances of samples to their closest cluster center)  inertia = model.inertia\_  print(f"Inertia: {inertia}") | Mencari nilai inertia dari model kmeans yang telah dibuat, lalu mencetak nilainya. |
| # Silhouette Coefficient (Koefisien Silhouette)  # Menghitung koefisien Silhouette untuk berbagai jumlah cluster (k) dari 2 hingga 11.  sc = []  for k in range(2, 12):  sil = KMeans(n\_clusters=k, random\_state=42)  sil.fit(df\_imputed)  score = silhouette\_score(final\_pca,sil.labels\_)  sc.append(score)  #Membuat grafik garis yang menunjukkan perubahan skor siluet terhadap jumlah klaster  plt.style.use("fivethirtyeight")  plt.plot(range(2, 12), sc)  plt.xticks(range(2, 12))  plt.xlabel('Jumlah Klaster')  plt.ylabel('Silhouette Score')  plt.show() | Mencari nilai koefisien silhouette dengan perulangan k sebanyak 2-11. Hasil silhouette yang diperoleh dijumlahkan ke array sc  Memvisulaisasikan hasil silhouette score dengan matplotlib. Untuk sumbu x adalah jumlah kluster dan sumbu y adalah nilai silhouette nya |
| # Silhouette  kmean = KMeans(n\_clusters=3, random\_state=10)  kmean.fit(final\_pca)  # Memperoleh label klaster untuk setiap sampel  labels = kmean.labels\_  # Silhouette Score  silhouette\_score = metrics.silhouette\_score(final\_pca, labels)  print(f"Silhouette Score: {silhouette\_score}") | Memodelkan hasil klastering dengan n\_clsuter = 3 dan mencetak nilai silhouette scorenya |
| # visualisasi untuk n = 3  model = KMeans(n\_clusters =3, random\_state = 42)  visualizer = SilhouetteVisualizer(model,  colors = 'yellowbrick')  visualizer.fit(final\_pca)  visualizer.show() | Memvisualisasikan hasil silhouette scrore dengan n\_cluster = 3 |
| # visualisasi untuk n = 2  model = KMeans(n\_clusters =2, random\_state = 42)  visualizer = SilhouetteVisualizer(model,  colors = 'yellowbrick')  visualizer.fit(final\_pca)  visualizer.show() | Memvisualisasikan hasil silhouette scrore dengan n\_cluster = 2 |
| # visualisasi untuk n = 4  model = KMeans(n\_clusters =4, random\_state = 42)  visualizer = SilhouetteVisualizer(model,  colors = 'yellowbrick')  visualizer.fit(final\_pca)  visualizer.show() | Memvisualisasikan hasil silhouette scrore dengan n\_cluster = 4 |
| # Mendapatkan pusat (centroid) dari setiap cluster  centroids = kmean.cluster\_centers\_  cluster\_labels = final\_pca["cluster"].unique()  cluster\_0 = final\_pca[final\_pca["cluster"] == 0]  cluster\_1 = final\_pca[final\_pca["cluster"] == 1]  cluster\_2 = final\_pca[final\_pca["cluster"] == 2]  # Visualizing clusters  plt.figure(figsize=(8, 6))  plt.scatter(cluster\_0['PC1'], cluster\_0['PC2'], label='Cluster 0', c='blue', marker='^')  plt.scatter(cluster\_1['PC1'], cluster\_1['PC2'], label='Cluster 1', c='red', marker='^')  plt.scatter(cluster\_2['PC1'], cluster\_2['PC2'], label='Cluster 2', c='green', marker='^')  plt.scatter(centroids[:, 0], centroids[:, 1],marker='x', s=200, linewidths=3, color='black', label='Centroids')  plt.xlabel('Komponen 1')  plt.ylabel('Komponen 2')  plt.legend()  plt.show() | Memplotting hasil clustering dengan kmeans Cluster 0 berwarna biru, cluster 1 berwarna merah dan cluster 2 berwarna hijau .  Sumbu x untuk Komponen 1 dan sumbu y untuk Komponen 2. |
| # Menghitung jumlah titik data dalam setiap cluster  print('jumlah cluster 0 : ', final\_pca[final\_pca['cluster'] == 0].shape[0])  print('jumlah cluster 1 : ' , final\_pca[final\_pca['cluster'] == 1].shape[0])  print('jumlah cluster 2 : ', final\_pca[final\_pca['cluster'] == 2].shape[0]) | Menghitung jumlah titik data dalam setiap cluster Kemudian menampilkan jumlah anggota dalam masing-masing kluster. |

**Hierarchical Clustering**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| # Dendrograms dengan metode ward  plt.figure(figsize=(10, 7))  plt.title("Customers Dendrogram with line")  clusters = shc.linkage(final\_pca, method='ward', metric="euclidean")  shc.dendrogram(clusters)  plt.axhline(y=10, color='g', linestyle='-')  plt.show() | Membuat clustering dengan menghasilkan dendrogram dengan methode ward berukuran 10x7 |
| # Dendrograms dengan metode single  plt.figure(figsize=(10, 7))  plt.title("Customers Dendrogram with line")  clusters = shc.linkage(final\_pca, method='single', metric="euclidean")  shc.dendrogram(clusters)  plt.axhline(y=0.8, color='g', linestyle='-')  plt.show() | Membuat clustering dengan menghasilkan dendrogram dengan methode single berukuran 10x7 |
| # Dendrograms dengan metode complete  plt.figure(figsize=(10, 7))  plt.title("Customers Dendrogram with line")  clusters = shc.linkage(final\_pca, method=’complete’, metric="euclidean")  shc.dendrogram(clusters)  plt.axhline(y=0.8, color='g', linestyle='-')  plt.show() | Membuat clustering dengan menghasilkan dendrogram dengan methode complete berukuran 10x7 |
| # Silhouette untuk methode ward  clusterW = AgglomerativeClustering(n\_clusters=3,  affinity='euclidean', linkage='ward')  clusterW.fit\_predict(final\_pca)  # Memperoleh label klaster untuk setiap sampel  labelsW = clusterW.labels\_  # Silhouette Score  silhouette\_scoreAW = metrics.silhouette\_score(final\_pca, labelsW)  print(f"Silhouette Score: {silhouette\_scoreAW}") | Mencetak nilai silhouette score dan hasil klusteringnya dengan methode ward |
| # Silhouette untuk methode single  clusterS = AgglomerativeClustering(n\_clusters=3,  affinity='euclidean', linkage='single')  clusterS.fit\_predict(final\_pca)  # Memperoleh label klaster untuk setiap sampel  labelsS = clusterS.labels\_  # Silhouette Score  silhouette\_scoreAS = metrics.silhouette\_score(final\_pca, labelsS)  print(f"Silhouette Score: {silhouette\_scoreAS}") | Mencetak nilai silhouette score dan hasil klusteringnya dengan methode single |
| # Silhouette untuk methode complete  clusterW = AgglomerativeClustering(n\_clusters=3,  affinity='euclidean', linkage=’complete’)  clusterW.fit\_predict(final\_pca)  # Memperoleh label klaster untuk setiap sampel  labelsW = clusterW.labels\_  # Silhouette Score  silhouette\_scoreAW = metrics.silhouette\_score(final\_pca, labelsW)  print(f"Silhouette Score: {silhouette\_scoreAW}") | Mencetak nilai silhouette score dan hasil klusteringnya dengan methode complete |

**DBSCAN clustering**

|  |  |
| --- | --- |
| **Kode** | **Komentar Program** |
| # menghitung jarak ke tetangga terdekat (nearest neighbors)  nearest\_neighbors = NearestNeighbors(n\_neighbors=6) #bisa mencoba-coba n\_neighborsnya  neighbors = nearest\_neighbors.fit(final\_pca)  distances, indices = neighbors.kneighbors(final\_pca)  # mengurutkan jarak  distances = np.sort(distances, axis=0)  distances = distances[:,1]  # memvisualisasikan jarak dengan plot  plt.figure(figsize=(10,10))  plt.plot(distances)  plt.title('K-distance Graph',fontsize=20)  plt.xlabel('Data Points sorted by distance',fontsize=14)  plt.ylabel('Epsilon',fontsize=14)  plt.show() | Menghirung jarak masing-masing titik ke tetangga terdekat, di sini dicoba untuk memsukkan n\_neighbors = 6.  Setelah didapat hasil distance, lalu diurutkan jaraknya  Lalu memvisualisasikan dengan plot dengan sumbu x adalah data points sorted by distances, dan sumbu y adalah epsilon. |
| from sklearn.metrics import silhouette\_score  best\_score = -1  best\_eps = 0  best\_min\_samples = 0  for eps in np.arange(0.02, 0.05, 0.01): # Adjust the range as needed  for min\_samples in range(2, 14): # Adjust the range as needed  dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)  db = dbscan.fit\_predict(final\_pca)  if len(np.unique(db)) == 1:  print(f'skipping check for eps = {eps} and min\_samples = {min\_samples}')  else:  score = silhouette\_score(final\_pca, db)  if score > best\_score:  best\_score = score  best\_eps = eps  best\_min\_samples = min\_samples  print(f"Best silhouette score: {best\_score} with eps={best\_eps} and min\_samples={best\_min\_samples}") | Encari best\_score, best\_eps dan best\_min\_samples dengan perulangan. Karena pada gambar silhouette yang terbentuk 0.02 sampai 0.05, maka dilakukan perulangan dari range tersebut dan penambahannya sejumlah 0.01. untuk min\_samplesnya diambil 2 sampai dengan 14.  Lalu memodelkan dara final\_pca ke dalam dbscan.  Jika panjang nilai unique pada db hanya 1, maka bisa dileawti untuk nilai epsilon dan minimum samplenya, jika tidak maka lanjut mencari nilai silhouette dari final pca dan db. Jika score nya > best score sementara, maka tukar nilai best\_score dengan nilai score nya, begitu juga dengan epsilon dan best minimum sample.  Mencetak nilai best\_score, dengan nilai epsilon dan minimum samples nya |
| #DBSCAN  # Membuat model DBSCAN  y = labels # hasil clustering dengan kmeans  db = DBSCAN(eps=0.04, min\_samples=6).fit(final\_pca) # mencoba parameter epsilon = 0.04  # Mendapatkan label klaster untuk setiap sampel  final\_pca['cluster'] = db.labels\_  #menampilkan perbandingan cluster pada label yang diprediksi dari dbscan dan dari kmeans dalam array  core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)  core\_samples\_mask[db.core\_sample\_indices\_] = True  print(db)  print(y) | Karena sudah diperoleh nilai epsilon yang optimum ada di eps = 0.04 dan min\_sample =6, selanjutnya dimodelkan dengan dbscan  Mendapatkan label klister untuk setiap sampel dan disimpan dalam final\_pca[‘cluster’]  Menampilkan perbandingan cluster pada label yang diprediksi dari dbscan dan dari kmeans karena dataset yang dipiih tidak memiliki target data. Dengan membuat core\_samples\_mask dengan panjang yang sama dengan label dari DBSCAN kemudian larik tersebut diisi dengan nilai true pada indeks yang merupakan core sample dan cluster yang dihasilkan DBSCAN, sementara nilal yang lain diisi dengan false |
| # Menghitung jumlah klaster dan titik noise  n\_clusters\_ = len(set(db.labels\_)) - (1 if -1 in db.labels\_ else 0)  n\_noise\_ = list(db.labels\_).count(-1)  print('Informasi perfomansi clustering dengan DBSCAN eps = 0.03')  print('Estimated number of clusters: %d' % n\_clusters\_)  print('Estimated number of noise points: %d' % n\_noise\_)  print("Homogeneity: %0.3f" % metrics.homogeneity\_score(labels, final\_pca['cluster']))  print("Completeness: %0.3f" % metrics.completeness\_score(labels, final\_pca['cluster']))  print("V-measure: %0.3f" % metrics.v\_measure\_score(labels, final\_pca['cluster']))  print("Adjusted Rand Index: %0.3f"% metrics.adjusted\_rand\_score(labels, final\_pca['cluster']))  print("Adjusted Mutual Information: %0.3f"% metrics.adjusted\_mutual\_info\_score(labels, final\_pca['cluster']))  print("Silhouette Coefficient: %0.3f"% metrics.silhouette\_score(final\_pca, final\_pca['cluster'])) | Menghitung jumlah cluster dan titik noise. Untuk jumlah jumlah cluster dicari berdasar panjang dari set label yang diperoleh dari hasil klastering, dikurangi satu jika terdapat label -1. sementara noise dicari berdasar label yang memiliki nilai -1  Menampilkan evaluasi performansi clustering yang dihasilkan dari DBSCAN. evaluasinya berupa jumlah clster, jumlah noise, homogeneity, kelengkapan, V-Measure. Adjusted rand score, adjusted mutual info, dan silhouette score nya. |
| outliers = final\_pca[final\_pca['cluster'] == -1]  plt.figure(figsize=(12, 5))  sns.scatterplot('PC1', 'PC2',  data=final\_pca[final\_pca['cluster'] != -1],  hue='cluster', palette='Set2', legend='full', s=200)  plt.scatter(outliers['PC1'], outliers['PC2'], s=10, label='outliers', c="k")  plt.legend(fontsize='12')  plt.legend()  plt.show() | Menyimpan -1 dalam cluster adalah outlier  Membuat ukuran plot 12x5 yang berisikan scatterplot untuk sumvu x adalah PC 1 dan sumbu y adalah PC 2 dengan ukuran scatter = 200.  Menambahkan label untuk outlier yang dengan ukuran scatter = 10  Menampilkan hasil clustering dengan DBSCAN |