LAPORAN MACHINE LEARNING CASE BASED 2
KODE DOSEN: IZA

Disusun oleh:

RYAN OKTAVIANDI SUSILO WIBOWO 1301204289

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
BANDUNG
2022

Saya mengerjakan tugas ini dengan cara yang tidak melanggar aturan perkuliahan dan kode etik akademisi.

DAFTAR ISI

I. Pendahuluan

Dataset yang digunakan pada tugas *Machine Learning Case Based 2* ini adalah *water-treatment.data*, tujuan dari adanya *dataset* ini adalah untuk membantu auditor dengan membangun clustering. Dalam data yang digunakan memiliki panjang data 527 dengan 38 atribut.

1.1 Import Data

Data akan di load dan disimpan pada sebuah variabel yang bernama water_treatment, dan kemudian kolom 0 pada dataframe akan dihapus dikarenakan data yang terdapat pada kolom ke 0 merupakan data tanggal, yang mengakibatkan data tersebut tidak relevan atau tidak terpakai di selanjutnya, kemudian seluruh data yang belum ada nilainya atau dalam dataframe bernilai ? maka akan diganti dengan NaN

```
[54] water_treatment = pd.read_csv("https://raw.githubusercontent.com/mobs3288/ML_Case-Based-2/main/water-treatment.data", header = None)
     df=pd.DataFrame(water_treatment)
     df.drop(df.columns[0], inplace = True, axis = 1)

     df=df.replace("?",np.NaN)
     df
```

	1	2	3	4	5	6	7	8	9	10	...	29	30	31	32	33	34	35	36	37	38
0	44101	1.50	7.8	NaN	407	166	66.3	4.5	2110	7.9	...	2000	NaN	58.8	95.5	NaN	70.0	NaN	79.4	87.3	99.6
1	39024	3.00	7.7	NaN	443	214	69.2	6.5	2660	7.7	...	2590	NaN	60.7	94.8	NaN	80.8	NaN	79.5	92.1	100
2	32229	5.00	7.6	NaN	528	186	69.9	3.4	1666	7.7	...	1888	NaN	58.2	95.6	NaN	52.9	NaN	75.8	88.7	98.5
3	35023	3.50	7.9	205	588	192	65.6	4.5	2430	7.8	...	1840	33.1	64.2	95.3	87.3	72.3	90.2	82.3	89.6	100
4	36924	1.50	8.0	242	496	176	64.8	4.0	2110	7.9	...	2120	NaN	62.7	95.6	NaN	71.0	92.1	78.2	87.5	99.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
522	32723	0.16	7.7	93	252	176	56.8	2.3	894	7.7	...	942	NaN	62.3	93.3	69.8	75.9	79.6	78.6	96.6	99.6
523	33535	0.32	7.8	192	346	172	68.6	4.0	988	7.8	...	950	NaN	58.3	97.8	83.0	59.1	91.1	74.6	90.7	100
524	32922	0.30	7.4	139	367	180	64.4	3.0	1060	7.5	...	1136	NaN	65.0	97.1	76.2	66.4	82.0	77.1	88.9	99
525	32190	0.30	7.3	200	545	258	65.1	4.0	1260	7.4	...	1326	39.8	65.9	97.1	81.7	70.9	89.5	87.0	89.5	99.8
526	30488	0.21	7.5	152	300	132	69.7	NaN	1073	7.4	...	1224	NaN	69.5	NaN	81.7	76.4	NaN	81.7	86.4	NaN

II. Dataset Preprocessing

Pada Preprocessing ini digunakan dua metode, yaitu: *Data Cleaning*, dan *Normalization*. *Data cleaning* adalah penanganan *missing value* dan *noise*, dan *Normalization* adalah mengubah nilai kolom numerik dalam himpunan data untuk menggunakan skala umum, tanpa mendistorsi perbedaan dalam rentang nilai atau kehilangan informasi.

2.1 Data Cleaning

Data yang bernilai NaN pada dataframe akan diganti dengan mean/rata-rata nilai dari setiap kolom data tersebut berada.

```
[55] df = df.apply(pd.to_numeric, errors='coerce')

     df = df.fillna(df.mean())
     df_clean = df.to_numpy()
```

Berikut merupakan data setelah dilakukannya data Cleaning dengan cara memasukan nilai mean setiap kolom ke dalam data yang bernilai NaN

```
[56] dataframe = pd.DataFrame(df_clean, columns = ["Q_E", "ZN_E", "PH_E", "DBO_E", "DQO_E", "SS_E",
        "SSV_E", "SED_E", "COND_E", "PH_P", "DBO_P", "SS_P", "SSV_P", "SED_P", "COND_P",
        "PH_D", "DBO_D", "DQO_D", "SS_D", "SSV_D", "SED_D", "COND_D", "PH_S", "DBO_S",
        "DQO_S", "SS_S", "SSV_S", "SED_S", "COND_S", "RD_DBO_P", "RD_SS_P", "RD_SED_P",
        "RD_DBO_S", "RD_DQO_S", "RD_DBO_G", "RD_DQO_G", "RD_SS_G", "RD_SED_G"])

     dataframe
```

	Q_E	ZN_E	PH_E	DBO_E	DQO_E	SS_E	SSV_E	SED_E	COND_E	PH_P	...	COND_S	RD_DBO_P	RD_SS_P	RD_SED_P	RD_DBO_S	RD_DQO_S	RD_DBO_G	RD_DQO_G	RD_SS_G	RD_SED_G
0	44101.0	1.50	7.8	188.714286	407.0	166.0	66.3	4.500000	2110.0	7.9	...	2000.0	39.085806	58.8	95.5000	83.448049	70.0	89.013646	79.4	87.3	99.60000
1	39024.0	3.00	7.7	188.714286	443.0	214.0	69.2	6.500000	2660.0	7.7	...	2590.0	39.085806	60.7	94.8000	83.448049	80.8	89.013646	79.5	92.1	100.00000
2	32229.0	5.00	7.6	188.714286	528.0	186.0	69.9	3.400000	1666.0	7.7	...	1888.0	39.085806	58.2	95.6000	83.448049	52.9	89.013646	75.8	88.7	98.50000
3	35023.0	3.50	7.9	205.000000	588.0	192.0	65.6	4.500000	2430.0	7.8	...	1840.0	33.100000	64.2	95.3000	87.300000	72.3	90.200000	82.3	89.6	100.00000
4	36924.0	1.50	8.0	242.000000	496.0	176.0	64.8	4.000000	2110.0	7.9	...	2120.0	39.085806	62.7	95.6000	83.448049	71.0	92.100000	78.2	87.5	99.50000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
522	32723.0	0.16	7.7	93.000000	252.0	176.0	56.8	2.300000	894.0	7.7	...	942.0	39.085806	62.3	93.3000	69.800000	75.9	79.600000	78.6	96.6	99.60000
523	33535.0	0.32	7.8	192.000000	346.0	172.0	68.6	4.000000	988.0	7.8	...	950.0	39.085806	58.3	97.8000	83.000000	59.1	91.100000	74.6	90.7	100.00000
524	32922.0	0.30	7.4	139.000000	367.0	180.0	64.4	3.000000	1060.0	7.5	...	1136.0	39.085806	65.0	97.1000	76.200000	66.4	82.000000	77.1	88.9	99.00000
525	32190.0	0.30	7.3	200.000000	545.0	258.0	65.1	4.000000	1260.0	7.4	...	1326.0	39.800000	65.9	97.1000	81.700000	70.9	89.500000	87.0	89.5	99.80000
526	30488.0	0.21	7.5	152.000000	300.0	132.0	69.7	4.593825	1073.0	7.4	...	1224.0	39.085806	69.5	90.5542	81.700000	76.4	89.013646	81.7	86.4	99.08629

Dan, berikut merupakan data yang telah di cleaning dalam bentuk array.

```
[57] df_clean

     array([[4.41010000e+04, 1.50000000e+00, 7.80000000e+00, ...,
             7.94000000e+01, 8.73000000e+01, 9.96000000e+01],
            [3.90240000e+04, 3.00000000e+00, 7.70000000e+00, ...,
             7.95000000e+01, 9.21000000e+01, 1.00000000e+02],
            [3.22290000e+04, 5.00000000e+00, 7.60000000e+00, ...,
             7.58000000e+01, 8.87000000e+01, 9.85000000e+01],
            ...,
            [3.29220000e+04, 3.00000000e-01, 7.40000000e+00, ...,
             7.71000000e+01, 8.89000000e+01, 9.90000000e+01],
            [3.21900000e+04, 3.00000000e-01, 7.30000000e+00, ...,
             8.70000000e+01, 8.95000000e+01, 9.98000000e+01],
            [3.04880000e+04, 2.10000000e-01, 7.50000000e+00, ...,
             8.17000000e+01, 8.64000000e+01, 9.90862903e+01]])
```

2.2 Normalization

Data yang telah dibersihkan atau telah melakukan proses cleaning, selanjutnya akan dilakukan proses normalization dengan metode min-max, Cara kerja dari metode ini adalah setiap nilai pada sebuah fitur dikurangi dengan nilai minimum fitur tersebut, kemudian dibagi dengan rentang nilai atau nilai maksimum dikurangi nilai minimum dari fitur tersebut.

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

```
[58] normalized_df=(df_clean-df_clean.min())/(df_clean.max()-df_clean.min())
```

Berikut merupakan hasil data yang telah di normalisasi.

```
[59] dataframe_clean = pd.DataFrame(normalized_df, columns = ["Q_E", "ZN_E", "PH_E", "DBO_E", "DQO_E", "SS_E",
        "SSV_E", "SED_E", "COND_E", "PH_P", "DBO_P", "SS_P", "SSV_P", "SED_P", "COND_P",
        "PH_D", "DBO_D", "DQO_D", "SS_D", "SSV_D", "SED_D", "COND_D", "PH_S", "DBO_S",
        "DQO_S", "SS_S", "SSV_S", "SED_S", "COND_S", "RD_DBO_P", "RD_SS_P", "RD_SED_P",
        "RD_DBO_S", "RD_DQO_S", "RD_DBO_G", "RD_DQO_G", "RD_SS_G", "RD_SED_G"])

    dataframe_clean
```

	Q_E	ZN_E	PH_E	DBO_E	DQO_E	SS_E	SSV_E	SED_E	COND_E	PH_P	...
0	0.734026	0.000025	0.000130	0.003141	0.006774	0.002763	0.001104	0.000075	0.035119	0.000131	...
1	0.649523	0.000050	0.000128	0.003141	0.007373	0.003562	0.001152	0.000108	0.044274	0.000128	...
2	0.536426	0.000083	0.000126	0.003141	0.008788	0.003096	0.001163	0.000057	0.027729	0.000128	...
3	0.582930	0.000058	0.000131	0.003412	0.009787	0.003196	0.001092	0.000075	0.040445	0.000130	...
4	0.614570	0.000025	0.000133	0.004028	0.008256	0.002929	0.001079	0.000067	0.035119	0.000131	...
...	...	...	...	...	...	...	...	...	...	...	...
522	0.544648	0.000003	0.000128	0.001548	0.004194	0.002929	0.000945	0.000038	0.014880	0.000128	...
523	0.558163	0.000005	0.000130	0.003196	0.005759	0.002863	0.001142	0.000067	0.016444	0.000130	...
524	0.547960	0.000005	0.000123	0.002314	0.006108	0.002996	0.001072	0.000050	0.017643	0.000125	...
525	0.535777	0.000005	0.000122	0.003329	0.009071	0.004294	0.001084	0.000067	0.020972	0.000123	...
526	0.507448	0.000003	0.000125	0.002530	0.004993	0.002197	0.001160	0.000076	0.017859	0.000123	...

	COND_S	RD_DBO_P	RD_SS_P	RD_SED_P	RD_DBO_S	RD_DQO_S	RD_DBO_G	RD_DQO_G	RD_SS_G	RD_SED_G
0	0.033288	0.000651	0.000979	0.001590	0.001389	0.001165	0.001482	0.001322	0.001453	0.001658
1	0.043108	0.000651	0.001010	0.001578	0.001389	0.001345	0.001482	0.001323	0.001533	0.001664
2	0.031424	0.000651	0.000969	0.001591	0.001389	0.000880	0.001482	0.001262	0.001476	0.001639
3	0.030625	0.000551	0.001069	0.001586	0.001453	0.001203	0.001501	0.001370	0.001491	0.001664
4	0.035286	0.000651	0.001044	0.001591	0.001389	0.001182	0.001533	0.001302	0.001456	0.001656
...	...	...	...	...	...	...	...	...	...	...
522	0.015679	0.000651	0.001037	0.001553	0.001162	0.001263	0.001325	0.001308	0.001608	0.001658
523	0.015812	0.000651	0.000970	0.001628	0.001381	0.000984	0.001516	0.001242	0.001510	0.001664
524	0.018908	0.000651	0.001082	0.001616	0.001268	0.001105	0.001365	0.001283	0.001480	0.001648
525	0.022070	0.000662	0.001097	0.001616	0.001360	0.001180	0.001490	0.001448	0.001490	0.001661
526	0.020372	0.000651	0.001157	0.001507	0.001360	0.001272	0.001482	0.001360	0.001438	0.001649

Dapat dilihat bahwa seluruh data sekarang bernilai desimal, ini membuktikan bahwa normalisasi data telah berhasil dilakukan, dan berikut merupakan data yang telah di normalisasi dalam bentuk array.

```
normalized_df

array([[7.34025732e-01, 2.49662955e-05, 1.29824737e-04, ...,
         1.32154924e-03, 1.45303840e-03, 1.65776202e-03],
        [6.49523144e-01, 4.99325910e-05, 1.28160317e-04, ...,
         1.32321366e-03, 1.53293054e-03, 1.66441970e-03],
        [5.36425825e-01, 8.32209850e-05, 1.26495897e-04, ...,
         1.26163013e-03, 1.47634027e-03, 1.63945340e-03],
        ...,
        [5.47960254e-01, 4.99325910e-06, 1.23167058e-04, ...,
         1.28326759e-03, 1.47966911e-03, 1.64777550e-03],
        [5.35776701e-01, 4.99325910e-06, 1.21502638e-04, ...,
         1.44804514e-03, 1.48965563e-03, 1.66109086e-03],
        [5.07448278e-01, 3.49528137e-06, 1.24831478e-04, ...,
         1.35983089e-03, 1.43805862e-03, 1.64921174e-03]])
```

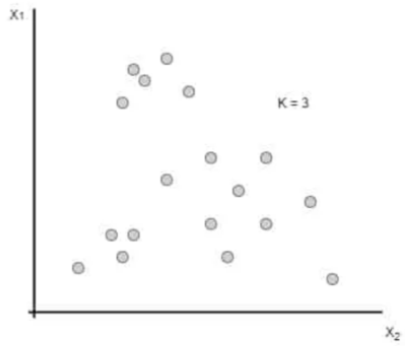
III. K-Means

Algoritma K-means membantu kelompok objek (seperti hewan di kebun binatang) menjadi lebih dekat satu sama lain dengan mengelompokkannya berdasarkan kesamaannya. Pengelompokan K-means adalah jenis algoritma pembelajaran mesin yang membantu Anda mengelompokkan item serupa. K-Means clustering adalah cara untuk mengelompokkan data secara otomatis. Algoritma ini bekerja dengan menyortir data ke dalam grup, dan kemudian menggunakan seperangkat aturan untuk menyatukan setiap grup.

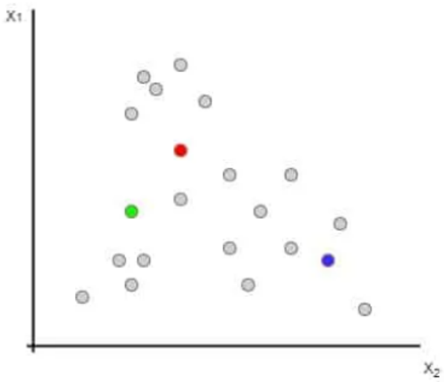
Pada algoritma pembelajaran ini, komputer mengelompokkan sendiri data-data yang menjadi masukannya tanpa mengetahui terlebih dulu target kelasnya. Pembelajaran ini termasuk dalam unsupervised learning. Masukan yang diterima adalah data atau objek dan k buah kelompok (cluster) yang diinginkan. Algoritma ini akan mengelompokkan data atau objek ke dalam k buah kelompok tersebut. Pada setiap cluster terdapat titik pusat (centroid) yang merepresentasikan cluster tersebut.

Cara Kerja Algoritma ini dapat dilihat pada gambar dibawah ini.

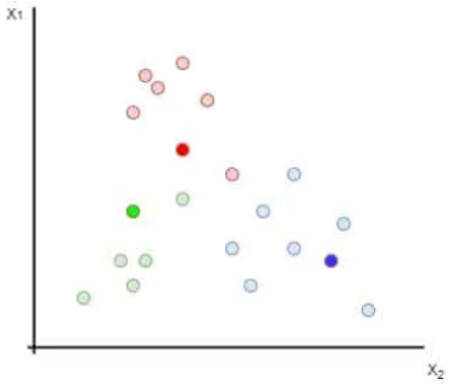
- Tentukan Jumlah Cluster (K).



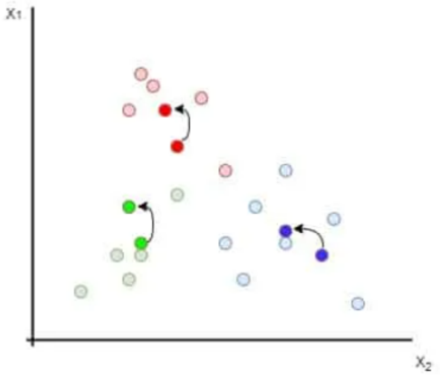
- Ambil titik acak Sebanyak K.
  Titik ini merupakan titik seed dan akan menjadi titik centroid proses pertama. Titik ini tidak harus titik data kita



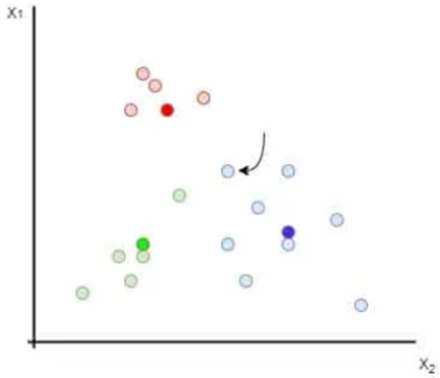
- Labeli seluruh data berdasarkan titik centroid terdekat
  Semua data diberikan label mengikuti titik centroid dari setiap klaster. Perhitungan jarak ini bisa menggunakan algoritma jarak tertentu, secara default dilakukan dengan Euclidean Distance



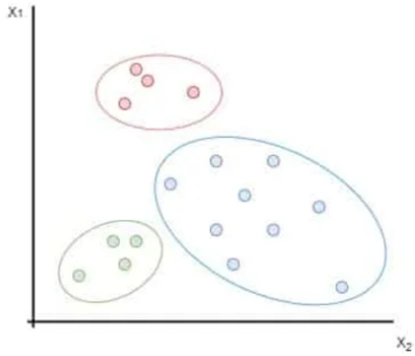
- Tentukan titik centroid berdasarkan cluster yang terbentuk
  Titik centroid selanjutnya "berpindah" ke lokasi centroid setiap cluster yang telah terbentuk.



- Labeli ulang seluruh data berdasarkan titik centroid terbaru



- Ulangi 2 langkah sebelumnya hingga tidak ada perubahan lagi



3.1 Algoritma K-Mean 1

3.1.1 Algoritma K-Means

Berikut merupakan algoritma K-Means yang telah dibangun.

```
[61] class K_Means:
       def __init__(self, k, tolerance = 0.001, max_iterations = 100):
         self.k = k
         self.tolerance = tolerance
         self.max_iterations = max_iterations

       def fit(self, data):

         self.centroids = {}

         for i in range(self.k):
           self.centroids[i] = data[i]

         for i in range(self.max_iterations):
           self.classes = {}
           for i in range(self.k):
             self.classes[i] = []

           for features in data:
             distances = [np.linalg.norm(features - self.centroids[centroid]) for centroid in self.centroids]
             classification = distances.index(min(distances))
             self.classes[classification].append(features)

           previous = dict(self.centroids)

           for classification in self.classes:
             self.centroids[classification] = np.average(self.classes[classification], axis = 0)

           isOptimal = True

           for centroid in self.centroids:

             original_centroid = previous[centroid]
             curr = self.centroids[centroid]

             if np.sum((curr - original_centroid)/original_centroid * 100.0) > self.tolerance:
               isOptimal = False

           if isOptimal:
             break

       def pred(self, data):
         distances = [np.linalg.norm(data - self.centroids[centroid]) for centroid in self.centroids]
         classification = distances.index(min(distances))
         return classification
```

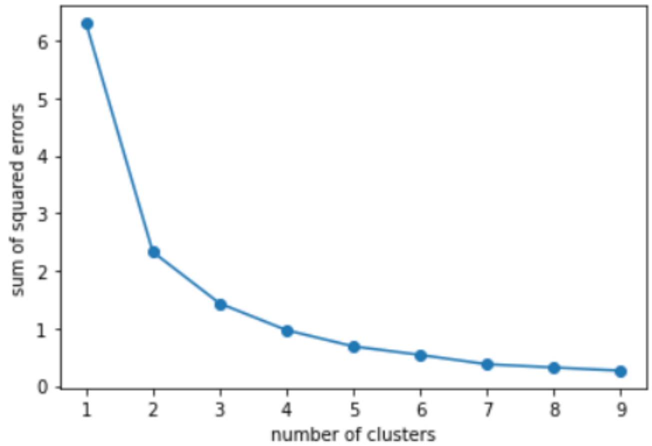
3.1.2 Algoritma Elbow

Algoritma ini digunakan untuk menampilkan grafik dari data yang digunakan, dan dengan algoritma ini, dapat memastikan nilai optimum K ada di angka berapa.

```
[62] import matplotlib.pyplot as plt
     from matplotlib import style

     # Library K-Means disini hanya untuk menampilkan elbow
     from sklearn.cluster import KMeans
```

```
[63] sum_of_squared_errors = []
     for i in range(1, 10):
         model = KMeans(n_clusters=i, random_state=0, init='random')
         model.fit(normalized_df)
         sum_of_squared_errors.append(model.inertia_)

     plt.plot(range(1, 10), sum_of_squared_errors, marker='o')
     plt.xlabel('number of clusters')
     plt.ylabel('sum of squared errors')
     plt.show()
```

Dari hasil grafik yang ditampilkan dapat disimpulkan bahwa nilai optimum K berada di angka 3, nilai K disini diambil pada saat grafik menunjukkan akan menurun dan nantinya akan static di satu nilai, dalam kasus ini elbow nya terdapat pada angka 3.



Namun untuk memastikan nya kembali, disini menggunakan satu library guna pengecekan kembali

```
[64] pip install kneed

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: kneed in /usr/local/lib/python3.7/dist-packages (0.8.1)
     Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from kneed) (1.7.3)
     Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.7/dist-packages (from kneed) (1.21.6)
```

Dapat dilihat bahwa hasil dari fungsi yang berasal library memberikan hasil nilai optimum K berada di angka 3, ini menandakan bahwa grafik yang dihasilkan sudah benar.

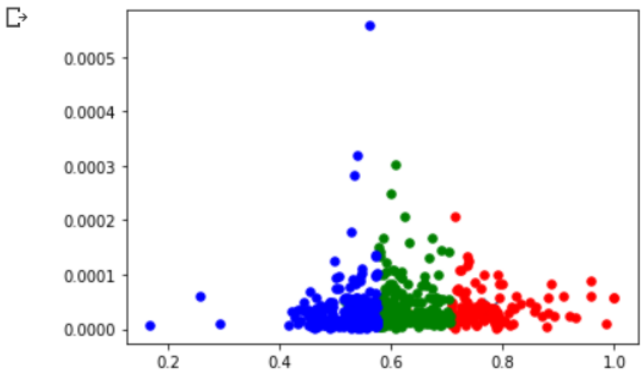
```
[65] from kneed import KneeLocator
     kn = KneeLocator(range(1, 10), sum_of_squared_errors, curve='convex', direction='decreasing')
     optimum_K= (kn.knee)
     print("Optimum K:",optimum_K)

     Optimum K: 3
```

3.1.3 Main Function

Berikut merupakan main function dari algoritma ini.

```
[71] K = optimum_K

     k_means = K_Means(K)
     k_means.fit(normalized_df)

     colors = 10*["r", "g", "b"]

     for centroid in k_means.centroids:
       plt.scatter(k_means.centroids[centroid][0], k_means.centroids[centroid][1], s = 130, marker = "x")

     for classification in k_means.classes:
       color = colors[classification]
       for features in k_means.classes[classification]:
         plt.scatter(features[0], features[1], color = color,s = 30)

     plt.show()
```

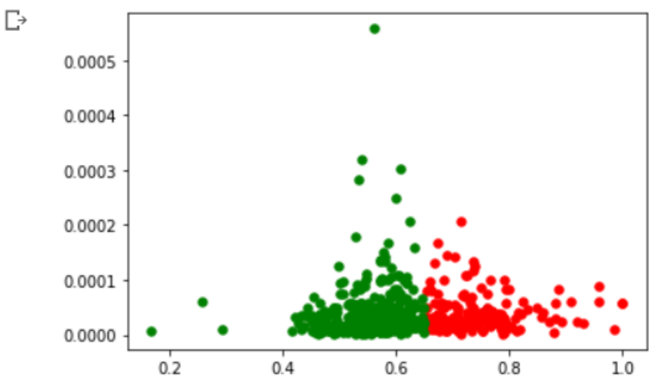
Berikut merupakan hasil dari algoritma K-Means dengan nilai Optimum K = 3



Selanjutnya akan dilakukan pengecekan apabila nilai K = 2:

```
K = 2

k_means = K_Means(K)
k_means.fit(normalized_df)


colors = 10*["r", "g", "b"]

for centroid in k_means.centroids:
  plt.scatter(k_means.centroids[centroid][0], k_means.centroids[centroid][1], s = 130, marker = "x")

for classification in k_means.classes:
  color = colors[classification]
  for features in k_means.classes[classification]:
    plt.scatter(features[0], features[1], color = color,s = 30)

plt.show()
```

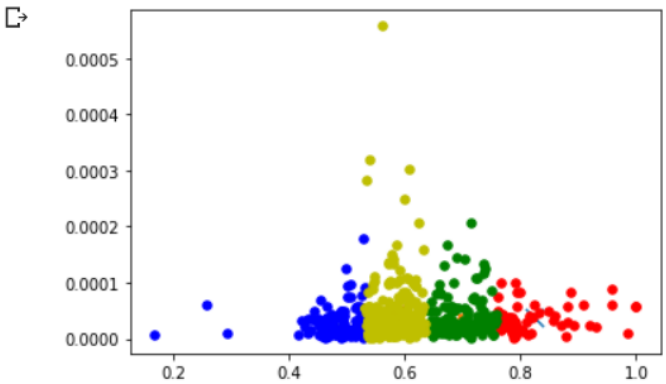
Berikut merupakan hasil dari algoritma K-Means apabila nilai K = 2



Selanjutnya akan dilakukan pengecekan apabila nilai K = 4:

```
[68] K = 4

     k_means = K_Means(K)
     k_means.fit(normalized_df)


     colors = 10*["r", "g", "b", "y"]

     for centroid in k_means.centroids:
       plt.scatter(k_means.centroids[centroid][0], k_means.centroids[centroid][1], s = 130, marker = "x")

     for classification in k_means.classes:
       color = colors[classification]
       for features in k_means.classes[classification]:
         plt.scatter(features[0], features[1], color = color,s = 30)

     plt.show()
```

Berikut merupakan hasil dari algoritma K-Means apabila nilai K = 2



3.2 Algoritma K-Mean 2

Library yang digunakan pada Algoritma K-Means 2 ini adalah sebagai berikut:

```
[72] import logging
     import numpy
     import random
     import seaborn as sns
     from collections import defaultdict
     import matplotlib.pyplot as plt
```

3.2.1 Algoritma K-Means

Berikut merupakan algoritma K-Means yang telah dibangun.

```
[78] class KMeans(object):
         def __init__(self, dataset_numpy_array, k_number_of_clusters, number_of_centroid_initializations,
                      max_number_of_iterations=30):
             self.dataset = dataset_numpy_array
             self.k_number_of_clusters = k_number_of_clusters
             self.number_of_instances, self.number_of_features = self.dataset.shape
             self.number_of_centroid_initializations = number_of_centroid_initializations
             self.inertia_values = []
             self.max_number_of_iterations = max_number_of_iterations
             self.clusters_all_iterations_record = []  # all centroids and clustered dataset points

         @staticmethod
         def get_euclidean_distance(n_dimensional_numpy_array_0, n_dimensional_numpy_array_1):
             return numpy.linalg.norm(n_dimensional_numpy_array_0 - n_dimensional_numpy_array_1)

         def create_random_initial_centroids(self):
             random_dataset_indices = random.sample(range(0, self.number_of_instances), self.k_number_of_clusters)
             random_initial_centroids = self.dataset[random_dataset_indices]
             return random_initial_centroids

         def assign_dataset_points_to_closest_centroid(self, centroids):
             logging.info("clustering dataset points to centroids...")
             cluster_single_iteration_record = defaultdict(list)
             for dataset_point in self.dataset:
                 euclidean_distances_between_dataset_point_and_centroids = []
                 for centroid in centroids:
                     distance_between_centroid_and_dataset_point = self.get_euclidean_distance(centroid, dataset_point)
                     logging.debug("Euclidean distance between dataset point {} and centroid {} is {}".format(
                         dataset_point, centroid, distance_between_centroid_and_dataset_point))
                     euclidean_distances_between_dataset_point_and_centroids.append(distance_between_centroid_and_dataset_point)
                 index_of_closest_centroid = numpy.argmin(euclidean_distances_between_dataset_point_and_centroids)
                 closest_centroid = tuple(centroids[index_of_closest_centroid])
                 logging.debug("dataset point {} is closest to centroid {}".format(dataset_point, centroid))
                 logging.debug("dataset point {} now belongs to cluster with centroid {}".format(dataset_point, centroid))
                 cluster_single_iteration_record[closest_centroid].append(dataset_point)
             logging.debug("cluster_single_iteration_record: {0}".format(cluster_single_iteration_record))
             return cluster_single_iteration_record
```

```
    def run_kmeans_initialized_centroid(self, initialization_number):
        centroids = self.create_random_initial_centroids()
        logging.info("random initial centroids are {}".format(centroids))
        self.clusters_all_iterations_record.append([])  # list of record of iteration centroids and clustered points

        for iteration in range(1, self.max_number_of_iterations+1):
            logging.info("starting iteration number {}...".format(iteration))
            cluster_single_iteration_record = self.assign_dataset_points_to_closest_centroid(centroids=centroids)
            self.clusters_all_iterations_record[initialization_number].append(cluster_single_iteration_record)
            updated_centroids = []
            for centroid in cluster_single_iteration_record:
                cluster_dataset_points = cluster_single_iteration_record[centroid]
                logging.debug("calculating the mean of {} clustered dataset points associated with centroid {}".format(
                    len(cluster_dataset_points), centroid))
                updated_centroid = numpy.mean(cluster_dataset_points, axis=0)
                logging.info("mean of the clustered dataset points is the new centroid at {}".format(updated_centroid))
                updated_centroids.append(updated_centroid)
            logging.debug("check if we meet early stopping criteria...")
            if self.get_euclidean_distance(numpy.array(updated_centroids), centroids) == 0:
                logging.info("updated centroids {} are the same as previous iteration centroids {}".format(
                    updated_centroids, centroids))
                logging.info("we've reached convergence of centroid values; end clustering")
                break
            logging.debug("use new updated_centroids values for next iteration...")
            centroids = updated_centroids
        return None
```

```
def fit(self):
    logging.info("perform K-Means {} times with new centroids at each start".format(self.number_of_centroid_initializations))
    for initialization_number in range(self.number_of_centroid_initializations):
        self.run_kmeans_initialized_centroid(initialization_number=initialization_number)

        # index of -1 is for the last cluster assignment of the iteration
        inertia_of_last_cluster_record = self.inertia(self.clusters_all_iterations_record[initialization_number][-1])
        self.inertia_values.append(inertia_of_last_cluster_record)
    return None

def inertia(self, clusters):
    cluster_sum_of_squares_points_to_clusters = 0
    logging.debug("cluster points: {}".format(clusters))

    for centroid, cluster_points in clusters.items():
        logging.debug("the cluster has a centroid at: {}".format(centroid))

        logging.debug("calculate sum of squares from centroid to all points in that cluster...")
        for cluster_point in cluster_points:
            euclidean_norm_distance = self.get_euclidean_distance(cluster_point, centroid)
            euclidean_norm_distance_squared = euclidean_norm_distance**2
            logging.debug("squared euclidean dist from centroid {} to point {} is {}".format(centroid, cluster_point,
                                                                                                euclidean_norm_distance_squared))
            cluster_sum_of_squares_points_to_clusters += euclidean_norm_distance_squared
    logging.info("inertia is: {}".format(cluster_sum_of_squares_points_to_clusters))
    return cluster_sum_of_squares_points_to_clusters
```

```
def index_lowest_inertia_cluster(self):
    minimum_inertia_value = min(self.inertia_values)
    logging.debug("minimum_inertia_value: {}".format(minimum_inertia_value))
    index_lowest_inertia = self.inertia_values.index(minimum_inertia_value)
    logging.debug("index_lowest_inertia: {}".format(index_lowest_inertia))
    return index_lowest_inertia

def final_iteration_optimal_cluster(self):
    # -1 gets us the final iteration from a centroid initialization of running K-Means
    return self.clusters_all_iterations_record[self.index_lowest_inertia_cluster()][-1]

def final_iteration_optimal_cluster_centroids(self):
    return list(self.final_iteration_optimal_cluster().keys())
```

```
@staticmethod
def plot_clusters(clusters, x_axis_label="", y_axis_label="", plot_title=""):
    list_of_colors = ['firebrick', 'gold', 'navy', 'lightseagreen', 'deepskyblue', 'mediumpurple',
                      'darkmagenta', 'palevioletred', 'darkgreen',  'darkorange', 'darkslategray', 'dimgrey']
    color_index = 0
    for centroid, cluster_points in clusters.items():
        cluster_color = list_of_colors[color_index]
        x_values_index = 0
        y_values_index = 1

        logging.debug("plot centroid {} as {}".format(centroid, cluster_color))
        plt.scatter(centroid[x_values_index], centroid[y_values_index], color=cluster_color, s=500, alpha=0.5)

        logging.debug("create lists of x-values and y-values for cluster points...")
        cluster_points_x_values = [cluster_point[x_values_index] for cluster_point in cluster_points]
        cluster_points_y_values = [cluster_point[y_values_index] for cluster_point in cluster_points]

        logging.debug("plot dataset points in cluster with centroid {} as {}".format(centroid, cluster_color))
        plt.scatter(cluster_points_x_values, cluster_points_y_values, color=cluster_color, s=100, marker='o')

        color_index += 1
    plt.title(plot_title)
    plt.xlabel(x_axis_label)
    plt.ylabel(y_axis_label)
    plt.show()
    return None

def predict(self, n_dimensional_numpy_array):
    # initially assign closest_centroid as large value; we'll reassign it later
    closest_centroid = numpy.inf
    for centroid in self.final_iteration_optimal_cluster_centroids():
        distance = self.get_euclidean_distance(centroid, n_dimensional_numpy_array)
        if distance < closest_centroid:
            closest_centroid = centroid
    return closest_centroid
```

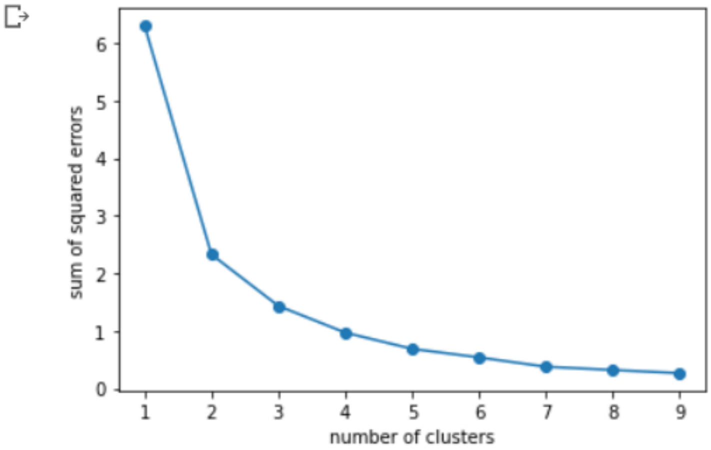
3.2.2 Algoritma Elbow

Algoritma ini digunakan untuk menampilkan grafik dari data yang digunakan, dan dengan algoritma ini, dapat memastikan nilai optimum K ada di angka berapa.

```
[62] import matplotlib.pyplot as plt
     from matplotlib import style

     # Library K-Means disini hanya untuk menampilkan elbow
     from sklearn.cluster import KMeans
```

```
[63] sum_of_squared_errors = []
     for i in range(1, 10):
         model = KMeans(n_clusters=i, random_state=0, init='random')
         model.fit(normalized_df)
         sum_of_squared_errors.append(model.inertia_)

     plt.plot(range(1, 10), sum_of_squared_errors, marker='o')
     plt.xlabel('number of clusters')
     plt.ylabel('sum of squared errors')
     plt.show()
```

Dari hasil grafik yang ditampilkan dapat disimpulkan bahwa nilai optimum K berada di angka 3, nilai K disini diambil pada saat grafik menunjukkan akan menurun yang nantinya akan membentuk seperti tangan, dan nantinya akan static di satu nilai, dalam kasus ini elbow nya terdapat pada angka 3.



Namun untuk memastikan nya kembali, disini menggunakan satu library guna pengecekan kembali

```
[64] pip install kneed

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: kneed in /usr/local/lib/python3.7/dist-packages (0.8.1)
     Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from kneed) (1.7.3)
     Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.7/dist-packages (from kneed) (1.21.6)
```

Dapat dilihat bahwa hasil dari fungsi yang berasal library memberikan hasil nilai optimum K berada di angka 3, ini menandakan bahwa grafik yang dihasilkan sudah benar.

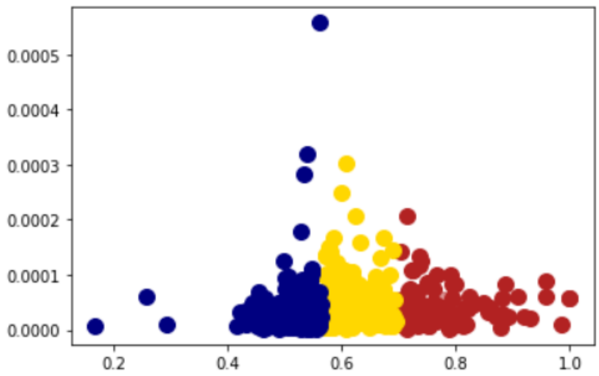
```
[65] from kneed import KneeLocator
     kn = KneeLocator(range(1, 10), sum_of_squared_errors, curve='convex', direction='decreasing')
     optimum_K= (kn.knee)
     print("Optimum K:",optimum_K)

     Optimum K: 3
```

3.2.3 Main Function

Berikut merupakan main function dari algoritma ini.

```
points = normalized_df
k_means_object = KMeans(dataset_numpy_array=points, k_number_of_clusters=3, number_of_centroid_initializations=1)
k_means_object.fit()
logging.info("inertia values: {0}".format(k_means_object.inertia_values))
index_lowest_inertia_cluster = k_means_object.index_lowest_inertia_cluster()
logging.info("lowest inertia value at centroid initialization #{}".format(index_lowest_inertia_cluster))
optimal_cluster_assignment = k_means_object.final_iteration_optimal_cluster()
logging.info("optimal_cluster_assignment: {}".format(optimal_cluster_assignment))
optimal_centroids = k_means_object.final_iteration_optimal_cluster_centroids()
logging.info("optimal centroids: {}".format(optimal_centroids))
k_means_object.plot_clusters(optimal_cluster_assignment)
```

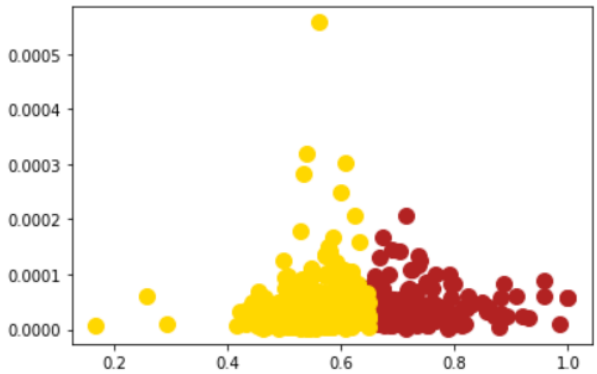
Berikut merupakan hasil dari algoritma K-Means dengan nilai Optimum K = 3



Selanjutnya akan dilakukan pengecekan apabila nilai K = 2:

```
[81] points = normalized_df
k_means_object = KMeans(dataset_numpy_array=points, k_number_of_clusters=2, number_of_centroid_initializations=1)
k_means_object.fit()
logging.info("inertia values: {0}".format(k_means_object.inertia_values))
index_lowest_inertia_cluster = k_means_object.index_lowest_inertia_cluster()
logging.info("lowest inertia value at centroid initialization #{}".format(index_lowest_inertia_cluster))
optimal_cluster_assignment = k_means_object.final_iteration_optimal_cluster()
logging.info("optimal_cluster_assignment: {}".format(optimal_cluster_assignment))
optimal_centroids = k_means_object.final_iteration_optimal_cluster_centroids()
logging.info("optimal centroids: {}".format(optimal_centroids))
k_means_object.plot_clusters(optimal_cluster_assignment)
```

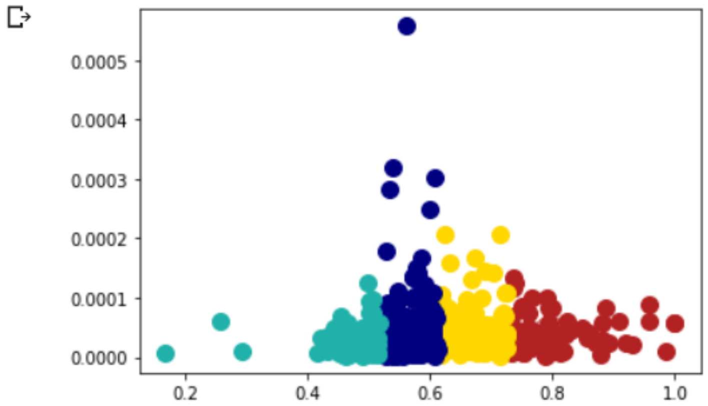
Berikut merupakan hasil dari algoritma K-Means apabila nilai K = 2



Selanjutnya akan dilakukan pengecekan apabila nilai K = 4:

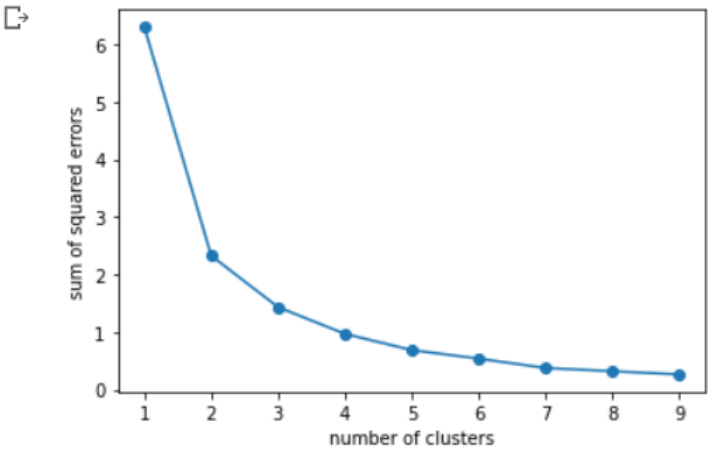
```
[82] points = normalized_df
     k_means_object = KMeans(dataset_numpy_array=points, k_number_of_clusters=4, number_of_centroid_initializations=1)
     k_means_object.fit()
     logging.info("inertia values: {0}".format(k_means_object.inertia_values))
     index_lowest_inertia_cluster = k_means_object.index_lowest_inertia_cluster()
     logging.info("lowest inertia value at centroid initialization #{}".format(index_lowest_inertia_cluster))
     optimal_cluster_assignment = k_means_object.final_iteration_optimal_cluster()
     logging.info("optimal_cluster_assignment: {}".format(optimal_cluster_assignment))
     optimal_centroids = k_means_object.final_iteration_optimal_cluster_centroids()
     logging.info("optimal centroids: {}".format(optimal_centroids))
     k_means_object.plot_clusters(optimal_cluster_assignment)
```

Berikut merupakan hasil dari algoritma K-Means apabila nilai K = 2

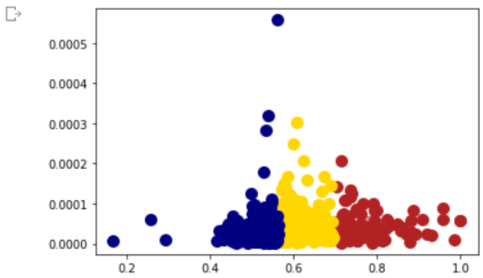


IV. Evaluasi Hasil

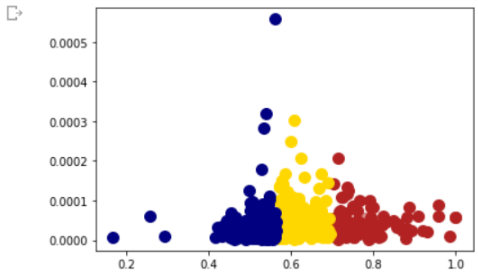
Dari pengamatan yang telah dilakukan, didapati bahwa nilai Optimum K untuk algoritma K-Mean yang telah dibangun adalah 3.



Nilai 3 Sendiri didapat dari hasil algoritma Elbow Method, setelah itu dibandingkan lah hasil dari Algoritma K-Mean 1 dan Algoritma K-Mean 2, didapati bahwa kedua algoritma menghasilkan hasil yang sama persis.



Algoritma K-Mean 1



Algoritma K-Mean 2

Dapat dilihat dari gambar diatas bahwa hasil clustering dari kedua algoritma sama persis. Hasil dari program yang telah dibuat berupa clustering sebanyak 3 cluster terhadap dataset water_treatment.

Lampiran

Colab
https://colab.research.google.com/drive/1I-CDFg5FpvEmXU3d79RdwhmOQOzyWxCp?usp=sharing

Github
https://github.com/mobs3288/ML_Case-Based-2

Youtube

Referensi

https://www.skytowner.com/explore/fill_missing_values_with_the_mean_of_the_column
https://www.folkstalk.com/tech/how-to-fill-missing-values-dataframe-with-mean-with-code-examples/
https://www.statology.org/pandas-fillna-with-mean/
https://www.makeuseof.com/fill-missing-data-with-pandas/
https://www.geeksforgeeks.org/how-to-fill-nan-values-with-mean-in-pandas/
https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/
https://www.projectpro.io/recipes/impute-missing-values-with-means-in-python
https://stackoverflow.com/questions/26414913/normalize-columns-of-a-dataframe
https://towardsdatascience.com/imputing-missing-values-using-the-simpleimputer-class-in-sklearn-99706afaff46
https://github.com/Gaurav-Reddy/Clustering-analysis/blob/bc67bc0e6c00336fb0bf0201b940a6ffb859780d/Knn_afterPCA.py
https://www.geeksforgeeks.org/ml-handle-missing-data-with-simple-imputer/
https://datatofish.com/numpy-array-to-pandas-dataframe/
https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/
https://www.kaggle.com/code/kevinarvai/knee-elbow-point-detection
https://www.geeksforgeeks.org/ml-determine-the-optimal-value-of-k-in-k-means-clustering/
https://www.machinelearningplus.com/predictive-modeling/k-means-clustering/
https://github.com/hsayedi/K-Means-Clustering-From-Scratch/blob/master/KMeans_Airbnb_DataMining.ipynb
https://github.com/tugot17/K-Means-Algorithm-From-Scratch/blob/master/K-means.ipynb
https://github.com/madhug-nadig/Machine-Learning-Algorithms-from-Scratch/blob/master/K%20Means%20Clustering.py
https://stackoverflow.com/questions/13187778/convert-pandas-dataframe-to-numpy-array
https://www.statology.org/pandas-fillna-with-mean/
https://dfrieds.com/machine-learning/k-means-from-scratch-python.html
https://stackoverflow.com/questions/14463277/how-to-disable-python-warnings
https://geospasialis.com/k-means-clustering/
https://raharja.ac.id/2020/04/19/k-means-clustering/
https://informatika.unsyiah.ac.id/~viska/basisdata/Bab%208%20-%20NORMALISASI%20DATA.ppt
https://learn.microsoft.com/id-id/azure/machine-learning/component-reference/normalize-data
https://ilmudatapy.com/metode-normalisasi-data/#:~:text=Min%2DMax,nilai%20minimum%20dari%20fitur%20tersebut.