import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import plotly.express as px

import matplotlib.patches as mpatches

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

from sklearn.decomposition import PCA

import library untuk dataset

#import dataset

dataset = pd.read\_csv ('covid19indonesia.csv')

#import dataset: Komentar ini hanya merupakan catatan dalam kode dan tidak mempengaruhi jalannya program. Komentar ini menyatakan bahwa kita akan mengimpor (membaca) dataset.

dataset = pd.read\_csv('covid19indonesia.csv'): Baris ini digunakan untuk mengimpor dataset dari file CSV. Fungsi pd.read\_csv() dari Pandas digunakan untuk membaca data dari file dengan format CSV (Comma-Separated Values). Data yang dibaca dari file CSV akan disimpan dalam bentuk DataFrame dengan nama variabel 'dataset'. DataFrame adalah struktur data tabular (berbentuk tabel) yang sangat berguna dalam manipulasi dan analisis data menggunakan Pandas

#tampilan dataset

print( len(dataset) )

print(dataset.head() )

1. print(len(dataset)): Baris kode ini mencetak jumlah baris yang ada dalam 'dataset'. Fungsi len() digunakan untuk menghitung jumlah baris, dan print() digunakan untuk menampilkan hasilnya ke dalam konsol.
2. print(dataset.head()): Fungsi head() pada Pandas digunakan untuk menampilkan beberapa baris pertama dari DataFrame. Secara default, akan ditampilkan 5 baris pertama, tetapi Anda dapat menentukan jumlah baris yang ingin ditampilkan dengan memberikan argumen bilangan bulat pada fungsi head().

print(dataset['New Cases'])

print(dataset['New Deaths'])

print(dataset['New Recovered'])

print(dataset['Total Cases'])

print(dataset['Total Deaths'])

print(dataset['Total Recovered'])

print(dataset['Total Active Cases'])

1. print(dataset['New Cases']): Baris ini mencetak nilai yang ada di kolom 'New Cases' dari dataset. Kolom ini kemungkinan berisi jumlah kasus COVID-19 yang baru dilaporkan pada tanggal tertentu.
2. print(dataset['Kematian Baru']): Baris ini mencetak nilai yang ada di kolom 'Kematian Baru' dari kumpulan data. Kolom ini kemungkinan berisi jumlah kematian akibat COVID-19 yang baru dilaporkan pada tanggal tertentu.
3. print(dataset['New Recovered']): Baris ini mencetak nilai yang ada di kolom 'New Recovered' dari dataset. Kolom ini kemungkinan berisi jumlah pemulihan COVID-19 yang baru dilaporkan pada tanggal tertentu.
4. print(dataset['Total Cases']): Baris ini mencetak nilai yang ada di kolom 'Total Cases' dari kumpulan data. Kolom ini kemungkinan berisi total kumulatif kasus COVID-19 dari waktu ke waktu.
5. print(dataset['Total Deaths']): Baris ini mencetak nilai yang ada di kolom 'Total Deaths' dari kumpulan data. Kolom ini kemungkinan berisi total kumulatif kematian akibat COVID-19 dari waktu ke waktu.
6. print(dataset['Total Recovered']): Baris ini mencetak nilai yang ada di kolom 'Total Recovered' dari dataset. Kolom ini kemungkinan berisi total kumulatif pemulihan COVID-19 dari waktu ke waktu.
7. print(dataset['Total Active Cases']): Baris ini mencetak nilai yang ada di kolom 'Total Active Cases' dari kumpulan data. Kolom ini kemungkinan berisi kasus COVID-19 aktif saat ini (Total Kasus - Total Meninggal - Total Sembuh) pada tanggal tertentu.

#split dataset

X = dataset.iloc[:, 3:10]

Y = dataset.iloc[:, 10]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, random\_state=0, test\_size=0.2)

1. #split dataset: Komentar ini hanya merupakan catatan dalam kode dan tidak mempengaruhi jalannya program. Komentar ini menyatakan bahwa kita akan membagi (membagi menjadi subset) dataset.
2. X = dataset.iloc[:, 3:10]: Baris ini mengambil subset dari dataset untuk digunakan sebagai fitur (features) atau variabel independen. Di sini, kita mengambil kolom data mulai dari kolom indeks ke-3 hingga kolom indeks ke-9. Kolom data ini akan menjadi fitur atau variabel independen yang akan digunakan untuk melakukan prediksi.
3. Y = dataset.iloc[:, 10]: Baris ini mengambil subset dari dataset untuk digunakan sebagai target (label) atau variabel dependen. Di sini, kita mengambil kolom data pada indeks ke-10. Kolom data ini akan menjadi target atau variabel dependen yang ingin kita prediksi.
4. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, random\_state=0, test\_size=0.2): Baris ini membagi dataset menjadi subset data pelatihan (train) dan data pengujian (test) untuk fitur dan target. Fungsi train\_test\_split() dari scikit-learn (sklearn) digunakan untuk melakukan pemisahan ini secara acak. Parameter random\_state=0 digunakan untuk memastikan bahwa pemisahan data yang dihasilkan akan selalu sama setiap kali kode ini dijalankan. Parameter test\_size=0.2 menentukan bahwa 20% dari data akan digunakan sebagai data pengujian (test) dan 80% akan digunakan sebagai data pelatihan (train).

#feature scaling

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

1. #feature scaling: Komentar ini hanya merupakan catatan dalam kode dan tidak mempengaruhi jalannya program. Komentar ini menyatakan bahwa kita akan melakukan penskalaan fitur (feature scaling).
2. sc\_X = StandardScaler(): Baris ini membuat sebuah objek dari kelas StandardScaler dari scikit-learn (sklearn). StandardScaler adalah salah satu teknik penskalaan fitur yang akan digunakan untuk menstandarisasi fitur-fitur dalam dataset. Penskalaan fitur ini penting karena beberapa algoritma machine learning mungkin sensitif terhadap skala fitur.
3. X\_train = sc\_X.fit\_transform(X\_train): Baris ini menerapkan penskalaan fitur pada data pelatihan (X\_train). Fungsi fit\_transform() dari objek StandardScaler digunakan untuk melakukan penskalaan fitur pada data pelatihan dan mengubah data pelatihan menjadi data yang telah distandarisasi.
4. X\_test = sc\_X.transform(X\_test): Baris ini menerapkan penskalaan fitur pada data pengujian (X\_test). Fungsi transform() dari objek StandardScaler digunakan untuk mengubah data pengujian menjadi data yang telah distandarisasi dengan menggunakan mean dan standar deviasi yang telah dihitung dari data pelatihan. Perlu dicatat bahwa kita menggunakan transform() pada data pengujian, bukan fit\_transform(), karena kita ingin menggunakan parameter penskalaan yang sama seperti yang dihitung dari data pelatihan.

dataset=pd.read\_csv("covid19indonesia.csv",parse\_dates=["Date"],index\_col="Province")

dataset.columns = dataset.columns.str.replace(' ','\_')

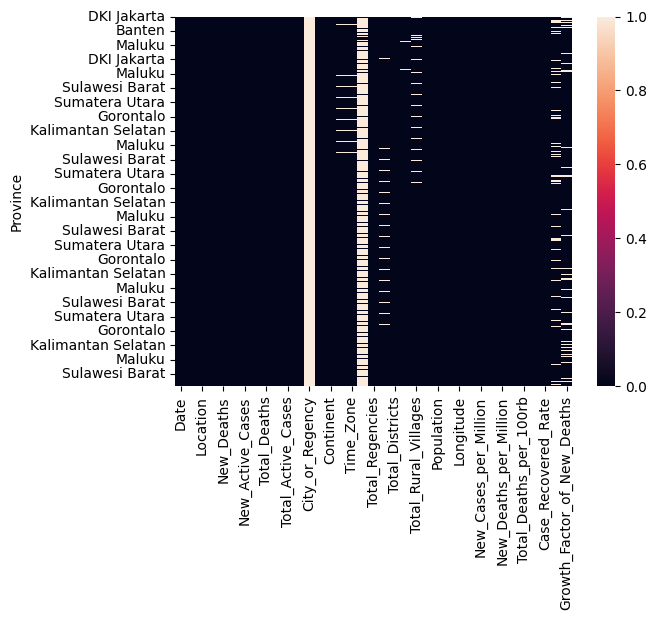
dataset.tail()

1. dataset = pd.read\_csv("covid19indonesia.csv", parse\_dates=["Date"], index\_col="Province"): Baris ini membaca dataset dari file CSV "covid19indonesia.csv" menggunakan Pandas. Fungsi read\_csv() digunakan untuk membaca data dari file CSV dan menghasilkan DataFrame. Argumen parse\_dates=["Date"] mengindikasikan bahwa kolom "Date" dalam dataset harus diubah menjadi tipe data tanggal (datetime). Argumen index\_col="Province" menetapkan bahwa kolom "Province" akan digunakan sebagai indeks (index) untuk DataFrame, sehingga DataFrame akan diindeks berdasarkan nama provinsi.
2. dataset.columns = dataset.columns.str.replace(' ', '\_'): Baris ini mengganti spasi (' ') dalam nama kolom dengan garis bawah ('\_'). Ini dilakukan untuk mengganti spasi dengan garis bawah dalam semua nama kolom DataFrame agar lebih mudah dan konsisten saat mengakses kolom menggunakan sintaks Python.
3. dataset.tail(): Baris ini digunakan untuk menampilkan beberapa baris terakhir dari DataFrame 'dataset'. Fungsi tail() pada Pandas akan menampilkan lima baris terakhir secara default. Tujuannya adalah untuk melihat bagian terakhir dari dataset dan memastikan bahwa data telah diimpor dengan benar.

sns.heatmap(dataset.isna())

1. sns: Ini adalah alias untuk Seaborn, sebuah library visualisasi data di Python yang dibangun di atas Matplotlib. Seaborn menyediakan antarmuka tingkat tinggi untuk membuat grafik statistik yang menarik secara estetika dan informatif.
2. dataset.isna(): Ekspresi ini digunakan untuk membuat DataFrame baru yang memiliki bentuk yang sama dengan dataset asli, di mana setiap sel berisi nilai boolean yang menunjukkan apakah sel yang sesuai di dataset asli mengandung nilai yang hilang (NaN) atau tidak. True mengindikasikan nilai yang hilang, dan False mengindikasikan nilai yang tidak hilang.
3. sns.heatmap(): Fungsi heatmap() pada Seaborn digunakan untuk membuat peta panas (heatmap) berdasarkan data yang diberikan. Peta panas menampilkan data dalam bentuk matriks dengan warna yang berbeda-beda, sehingga memungkinkan kita untuk memvisualisasikan pola dan hubungan dalam data secara intuitif.

Akan menampilakan tabel seperti dibawah :



dataset = dataset[['Total\_Cases','Total\_Deaths','Total\_Recovered','Population\_Density',

        'Population','Total\_Cases\_per\_Million','Total\_Deaths\_per\_Million']]

        #Membuat dataframe baru dengan feature yang dipilih

dataset

1. dataset = dataset[['Total\_Cases','Total\_Deaths','Total\_Recovered','Population\_Density','Population','Total\_Cases\_per\_Million','Total\_Deaths\_per\_Million']]: Baris ini menciptakan DataFrame baru dengan memilih kolom-kolom tertentu dari DataFrame asli 'dataset'. DataFrame yang baru akan berisi kolom-kolom dengan nama 'Total\_Cases', 'Total\_Deaths', 'Total\_Recovered', 'Population\_Density', 'Population', 'Total\_Cases\_per\_Million', dan 'Total\_Deaths\_per\_Million'. Dengan menggunakan double bracket '[[]]', kita dapat memilih beberapa kolom sekaligus.
2. #Membuat dataframe baru dengan feature yang dipilih: Ini adalah komentar dalam kode dan tidak mempengaruhi jalannya program. Komentar ini menjelaskan bahwa kita sedang membuat DataFrame baru dengan memilih fitur-fitur (kolom-kolom) tertentu dari dataset.
3. dataset: Baris ini menampilkan DataFrame yang telah dibuat dengan memilih fitur-fitur tertentu. DataFrame baru ini akan berisi hanya kolom-kolom yang dipilih sesuai dengan nama kolom yang dijelaskan pada baris pertama.

#Menggantikan NaN menjadi nilai 0

dataset = dataset.fillna(0)

dataset = dataset.fillna(0): Baris ini menggantikan nilai-nilai yang hilang (NaN) dalam DataFrame 'dataset' dengan nilai 0. Fungsi fillna() dari Pandas digunakan untuk mengisi nilai yang hilang dengan nilai yang ditentukan, dalam kasus ini, kita mengisi dengan 0.

#Menambah kolom feature Fatality Rate dan Recovery Rate

dataset["Fatality\_Rate"] = dataset.Total\_Deaths / dataset.Total\_Cases

dataset["Recovery\_Rate"] = dataset.Total\_Recovered / dataset.Total\_Cases

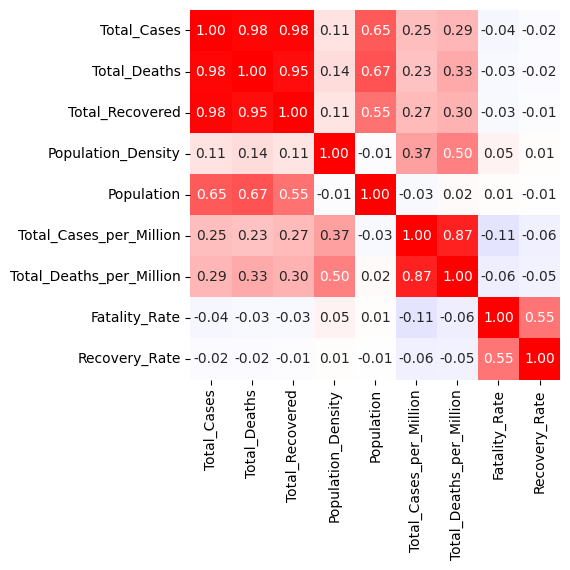
dataset.head()

1. dataset["Fatality\_Rate"] = dataset.Total\_Deaths / dataset.Total\_Cases: Baris ini menambahkan kolom baru bernama "Fatality\_Rate" ke dalam DataFrame 'dataset'. Kolom ini akan berisi nilai dari perbandingan antara jumlah kematian (Total\_Deaths) dan jumlah kasus total (Total\_Cases) di setiap baris. Dengan cara ini, kita dapat menghitung persentase rasio kematian dari total kasus pada setiap baris.
2. dataset["Recovery\_Rate"] = dataset.Total\_Recovered / dataset.Total\_Cases: Baris ini juga menambahkan kolom baru bernama "Recovery\_Rate" ke dalam DataFrame 'dataset'. Kolom ini akan berisi nilai dari perbandingan antara jumlah pemulihan (Total\_Recovered) dan jumlah kasus total (Total\_Cases) di setiap baris. Dengan cara ini, kita dapat menghitung persentase rasio pemulihan dari total kasus pada setiap baris.
3. dataset.head(): Baris ini menampilkan beberapa baris pertama dari DataFrame 'dataset', yang sudah termasuk kolom-kolom baru "Fatality\_Rate" dan "Recovery\_Rate". Tujuannya adalah untuk melihat bagian awal dari dataset setelah kolom-kolom baru ditambahkan.

sns.heatmap(dataset.corr(),cmap="bwr",cbar=False,vmin=-1,vmax=1,annot=True,square=True, fmt =".2f")

1. sns: Ini adalah alias untuk Seaborn, sebuah library visualisasi data di Python yang dibangun di atas Matplotlib.
2. dataset.corr(): Ekspresi ini digunakan untuk menghitung korelasi antara kolom-kolom dalam DataFrame 'dataset'. Fungsi corr() dari Pandas menghasilkan DataFrame baru yang berisi nilai korelasi antara setiap pasangan kolom dalam 'dataset'.
3. cmap="bwr": Parameter ini menentukan peta warna yang digunakan untuk peta panas (heatmap). Nilai "bwr" menunjukkan bahwa kita akan menggunakan peta warna biru-putih-merah, dengan warna biru menunjukkan korelasi negatif, warna putih menunjukkan korelasi netral, dan warna merah menunjukkan korelasi positif.
4. cbar=False: Parameter ini menentukan apakah batang warna (colorbar) akan ditampilkan atau tidak pada peta panas. Dalam kasus ini, kita menetapkan False untuk tidak menampilkan batang warna.
5. vmin=-1, vmax=1: Parameter ini menentukan nilai minimum dan maksimum untuk rentang warna pada peta panas. Dalam hal ini, kita menetapkan -1 sebagai nilai minimum dan 1 sebagai nilai maksimum, sesuai dengan rentang korelasi yang diinginkan (-1 hingga 1).
6. annot=True: Parameter ini menentukan apakah nilai-nilai korelasi akan ditampilkan di dalam sel-sel peta panas. Jika diatur sebagai True, nilai-nilai korelasi akan ditampilkan.
7. square=True: Parameter ini menentukan apakah bentuk peta panas akan diatur sebagai persegi (square) atau tidak. Jika diatur sebagai True, peta panas akan berbentuk persegi.
8. fmt=".2f": Parameter ini menentukan format tampilan untuk nilai-nilai yang ditampilkan di dalam sel-sel peta panas. ".2f" berarti nilai akan ditampilkan dalam format bilangan desimal dengan dua angka di belakang koma.

akan menampilkan tabel dibawah ini :



plt.scatter(X['New Cases'], X['New Deaths'], label='True Position')

plt.xlabel("kasus baru")

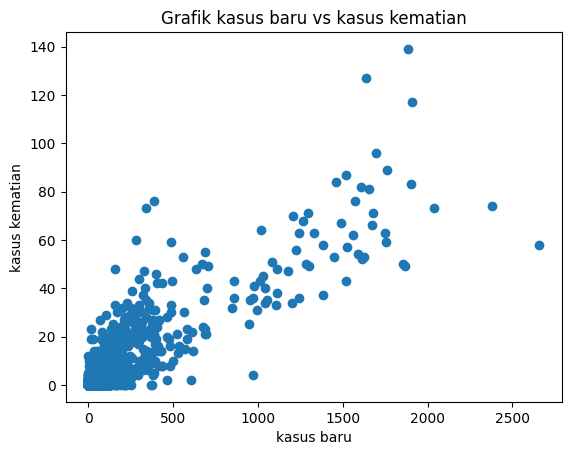
plt.ylabel("kasus kematian")

plt.title("Grafik kasus baru vs kasus kematian")

plt.show()

1. plt: Ini adalah alias untuk Matplotlib, sebuah library visualisasi data di Python yang digunakan untuk membuat berbagai jenis plot, termasuk scatter plot.
2. plt.scatter(X['New Cases'], X['New Deaths'], label='True Position'): Baris ini membuat scatter plot berdasarkan data dari kolom 'New Cases' dan 'New Deaths' dalam DataFrame 'X'. Scatter plot merupakan jenis grafik yang menampilkan titik-titik data pada bidang kartesian, di mana sumbu x menunjukkan data dari kolom 'New Cases', dan sumbu y menunjukkan data dari kolom 'New Deaths'. Parameter label='True Position' digunakan untuk memberi label pada scatter plot, meskipun dalam contoh ini, mungkin sebaiknya diganti dengan label yang lebih sesuai dengan data yang sebenarnya.
3. plt.xlabel("kasus baru"): Baris ini memberikan label sumbu x pada scatter plot, yaitu "kasus baru". Ini akan menjadi label yang ditampilkan pada sumbu x.
4. plt.ylabel("kasus kematian"): Baris ini memberikan label sumbu y pada scatter plot, yaitu "kasus kematian". Ini akan menjadi label yang ditampilkan pada sumbu y.
5. plt.title("Grafik kasus baru vs kasus kematian"): Baris ini memberikan judul pada scatter plot, yaitu "Grafik kasus baru vs kasus kematian". Ini akan menjadi judul yang ditampilkan di atas scatter plot.
6. plt.show(): Baris ini menampilkan scatter plot yang telah dibuat dengan semua konfigurasi sebelumnya. Fungsi show() dari Matplotlib digunakan untuk menampilkan plot ke dalam gambar (figure) dan menampilkannya di layar.

Akan menampilkan tabel dibawah ini :



#Scaling dataset dengan MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(dataset)

scaled\_dataset = pd.DataFrame(scaler.transform(dataset),columns= dataset.columns)

scaled\_dataset

1. scaler = MinMaxScaler(): Baris ini menciptakan sebuah objek dari kelas MinMaxScaler dari scikit-learn (sklearn). MinMaxScaler adalah salah satu teknik penskalaan fitur yang akan digunakan untuk melakukan normalisasi data dalam rentang tertentu.
2. scaler.fit(dataset): Baris ini menghitung nilai minimum dan maksimum untuk setiap fitur (kolom) dalam DataFrame 'dataset' menggunakan objek scaler yang telah dibuat sebelumnya. Nilai minimum dan maksimum ini akan digunakan untuk melakukan normalisasi data.
3. scaled\_dataset = pd.DataFrame(scaler.transform(dataset), columns=dataset.columns): Baris ini menerapkan normalisasi data pada DataFrame 'dataset' menggunakan objek scaler. Fungsi transform() dari objek scaler digunakan untuk mengubah data dalam 'dataset' menjadi data yang telah dinormalisasi. Hasilnya disimpan dalam DataFrame baru yang disebut 'scaled\_dataset'. Nama-nama kolom (fitur) akan tetap sama seperti di 'dataset'.
4. scaled\_dataset: Baris ini menampilkan DataFrame 'scaled\_dataset', yang berisi data yang telah dinormalisasi dalam rentang tertentu. Data ini memiliki skala yang seragam antara 0 dan 1, setelah melalui proses normalisasi.

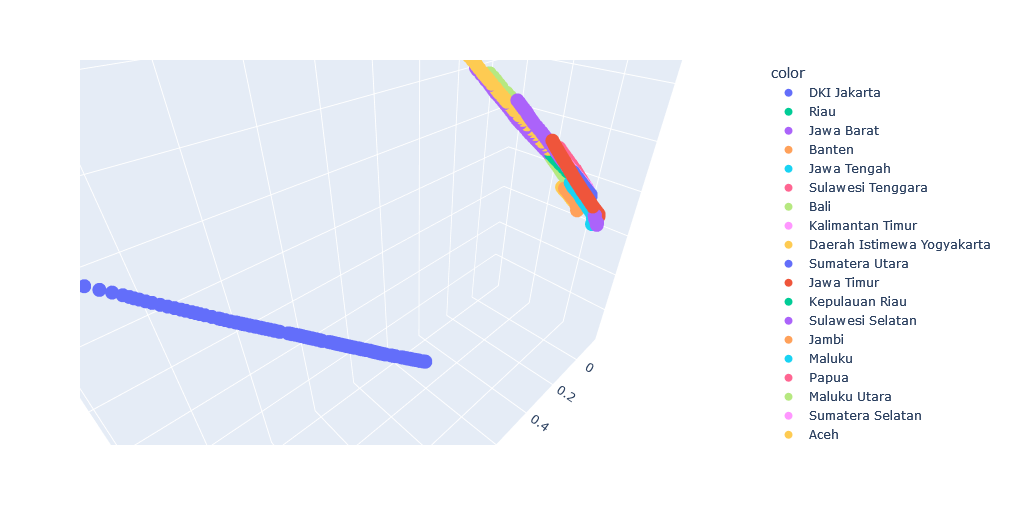
pca = PCA(n\_components = 3,random\_state = 42)

pca\_data= pd.DataFrame(pca.fit\_transform(scaled\_dataset), columns=(["PC1","PC2", "PC3"]))

pca\_data.describe().T

1. pca = PCA(n\_components=3, random\_state=42): Baris ini menciptakan sebuah objek dari kelas PCA (Principal Component Analysis) dari scikit-learn (sklearn). PCA adalah sebuah teknik analisis yang digunakan untuk mengurangi dimensi data dengan memproyeksikan data ke dalam ruang fitur yang lebih rendah, disebut sebagai komponen utama.
2. pca\_data = pd.DataFrame(pca.fit\_transform(scaled\_dataset), columns=["PC1", "PC2", "PC3"]): Baris ini mengurangi dimensi data dalam DataFrame 'scaled\_dataset' menggunakan objek pca yang telah dibuat sebelumnya. Fungsi fit\_transform() dari objek pca digunakan untuk melakukan transformasi PCA pada data dalam 'scaled\_dataset'. Hasil transformasi PCA ini disimpan dalam DataFrame baru yang disebut 'pca\_data', dengan kolom-kolom baru yang dinamakan "PC1", "PC2", dan "PC3".
3. pca\_data.describe().T: Baris ini menampilkan statistik deskriptif dari DataFrame 'pca\_data'. Fungsi describe() dari Pandas digunakan untuk menghitung statistik deskriptif seperti mean, median, kuartil, dan lainnya dari setiap kolom dalam 'pca\_data'. Parameter T digunakan untuk mentranspose hasil statistik deskriptif, sehingga baris akan menjadi kolom dan sebaliknya.
4. x =pca\_data["PC1"]
5. y =pca\_data["PC2"]
6. z =pca\_data["PC3"]
7. fig = px.scatter\_3d(pca\_data, x, y, z,color=dataset.index)
8. fig.show()
9. pca = PCA(n\_components=3, random\_state=42): Baris ini menciptakan sebuah objek dari kelas PCA (Principal Component Analysis) dari scikit-learn (sklearn). PCA adalah sebuah teknik analisis yang digunakan untuk mengurangi dimensi data dengan memproyeksikan data ke dalam ruang fitur yang lebih rendah, disebut sebagai komponen utama.
10. pca\_data = pd.DataFrame(pca.fit\_transform(scaled\_dataset), columns=["PC1", "PC2", "PC3"]): Baris ini mengurangi dimensi data dalam DataFrame 'scaled\_dataset' menggunakan objek pca yang telah dibuat sebelumnya. Fungsi fit\_transform() dari objek pca digunakan untuk melakukan transformasi PCA pada data dalam 'scaled\_dataset'. Hasil transformasi PCA ini disimpan dalam DataFrame baru yang disebut 'pca\_data', dengan kolom-kolom baru yang dinamakan "PC1", "PC2", dan "PC3".
11. pca\_data.describe().T: Baris ini menampilkan statistik deskriptif dari DataFrame 'pca\_data'. Fungsi describe() dari Pandas digunakan untuk menghitung statistik deskriptif seperti mean, median, kuartil, dan lainnya dari setiap kolom dalam 'pca\_data'. Parameter T digunakan untuk mentranspose hasil statistik deskriptif, sehingga baris akan menjadi kolom dan sebaliknya.

akan menampilkan tabel dibawah ini :



#menentukan KMeans dengan method elbow

wcss = []

for i in range(1,11):

    kmeans = KMeans(n\_clusters= i,

    init = 'k-means++', max\_iter= 300, n\_init= 10)

    kmeans.fit(pca\_data)

    wcss.append(kmeans.inertia\_)

plt.plot(range(1,11), wcss)

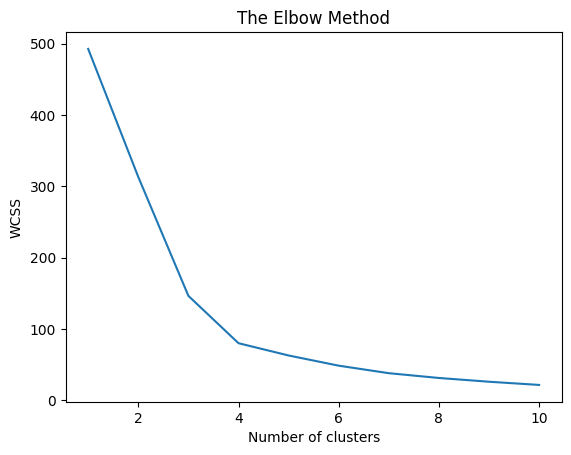
plt.title("The Elbow Method")

plt.xlabel("Number of clusters")

plt.ylabel("WCSS")

plt.show()

1. wcss = []: Ini menginisialisasi sebuah list kosong wcss (Within-Cluster Sum of Squares) yang akan digunakan untuk menyimpan jumlah kuadrat jarak antara setiap titik data dengan pusat klaster yang ditetapkan.
2. for i in range(1, 11): Ini menyiapkan perulangan yang berjalan dari 1 hingga 10 (inklusif), yang akan mencoba berbagai nilai n\_clusters (jumlah klaster) dari 1 hingga 10.
3. kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10): Ini menciptakan sebuah objek dari kelas KMeans dari scikit-learn (sklearn) dengan nilai n\_clusters (jumlah klaster) yang sedang diuji pada setiap iterasi perulangan. init='k-means++' menentukan bahwa pusat klaster awal akan diinisialisasi dengan menggunakan metode 'k-means++', max\_iter=300 menentukan jumlah iterasi maksimum dalam algoritma KMeans, dan n\_init=10 menentukan berapa kali algoritma KMeans akan dijalankan dengan inisialisasi pusat klaster yang berbeda.
4. kmeans.fit(pca\_data): Baris ini menggunakan data yang telah direduksi dimensinya (pca\_data) untuk melatih model KMeans dengan jumlah klaster yang ditetapkan pada setiap iterasi perulangan.
5. wcss.append(kmeans.inertia\_): Ini menambahkan nilai inertia\_ (Within-Cluster Sum of Squares) yang dihitung oleh model KMeans pada iterasi tersebut ke dalam list wcss.
6. plt.plot(range(1, 11), wcss): Ini membuat plot dari jumlah klaster versus WCSS untuk setiap percobaan nilai klaster dari 1 hingga 10.
7. plt.title("The Elbow Method"): Ini memberikan judul pada plot, yaitu "Metode Elbow".
8. plt.xlabel("Number of clusters"): Ini memberikan label sumbu x pada plot, yaitu "Jumlah klaster".
9. plt.ylabel("WCSS"): Ini memberikan label sumbu y pada plot, yaitu "Within-Cluster Sum of Squares".
10. plt.show(): Ini menampilkan plot hasil dari metode elbow, yang akan membantu kita menentukan jumlah klaster yang optimal berdasarkan bentuk siku (elbow) dalam plot.



Bisa ditentukan K nya 3

#dapat KMeans = 3

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=3,random\_state=42) #n\_clusters merupakan Jumlah cluster yang dipilih

#dan random\_state digunakan untuk mengontrol ke-random-an saat rerun kernel.

labels = kmeans.fit\_predict(pca\_data)

np.unique(labels)

1. from sklearn.cluster import KMeans: Baris ini mengimpor kelas KMeans dari modul sklearn.cluster. KMeans adalah algoritma clustering yang digunakan untuk membentuk klaster berdasarkan posisi pusat klaster (centroid) dan jarak antara data dengan centroid tersebut.
2. kmeans = KMeans(n\_clusters=3, random\_state=42): Baris ini menciptakan sebuah objek dari kelas KMeans dengan jumlah klaster yang ditetapkan sebanyak 3. Parameter random\_state=42 digunakan untuk mengontrol keacak-an dalam inisialisasi pusat klaster agar hasilnya dapat direproduksi ketika algoritma dijalankan kembali.
3. labels = kmeans.fit\_predict(pca\_data): Baris ini melatih model KMeans pada data yang sudah direduksi dimensinya (pca\_data) dan mengeluarkan label klaster untuk setiap data. Label klaster ini akan menunjukkan keanggotaan setiap data pada klaster yang terbentuk.
4. np.unique(labels): Baris ini menghitung dan menampilkan nilai unik dari label klaster yang dihasilkan. Fungsi unique() dari NumPy digunakan untuk mengidentifikasi nilai unik dalam array.

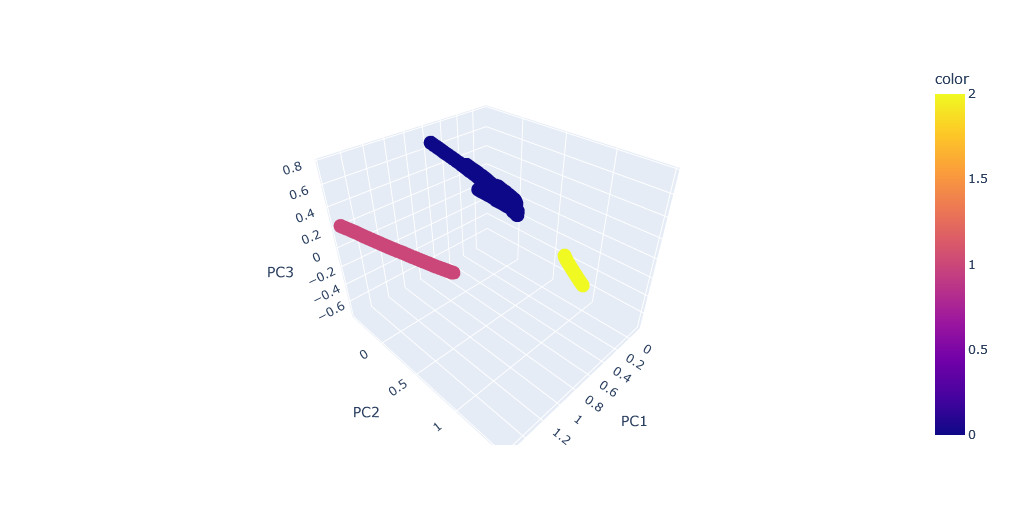
x =pca\_data["PC1"]

y =pca\_data["PC2"]

z =pca\_data["PC3"]

fig = px.scatter\_3d(pca\_data, x, y, z,color=labels)

fig.show()



#masukan kolum kluster pada dataframe awal

dataset["Clusters"] = labels

pl = sns.countplot(x=dataset["Clusters"])

pl.set\_title("Distribusi Cluster")

plt.show()

1. dataset["Clusters"] = labels: Baris ini menambahkan kolom baru "Clusters" ke dalam DataFrame 'dataset' yang berisi label klaster (clusters) yang telah dihasilkan oleh model KMeans sebelumnya. Setiap baris dalam DataFrame 'dataset' akan memiliki nilai label klaster (0, 1, atau 2) sesuai dengan klaster yang di-assign oleh algoritma KMeans.
2. pl = sns.countplot(x=dataset["Clusters"]): Baris ini menggunakan Seaborn untuk membuat countplot berdasarkan kolom "Clusters" dalam DataFrame 'dataset'. Countplot akan menghitung jumlah data points dalam setiap klaster dan menampilkan histogram vertikal yang menunjukkan distribusi data pada masing-masing klaster.
3. pl.set\_title("Distribusi Cluster"): Baris ini memberikan judul pada countplot, yaitu "Distribusi Cluster".
4. plt.show(): Baris ini menampilkan countplot yang telah dibuat menggunakan Seaborn.

