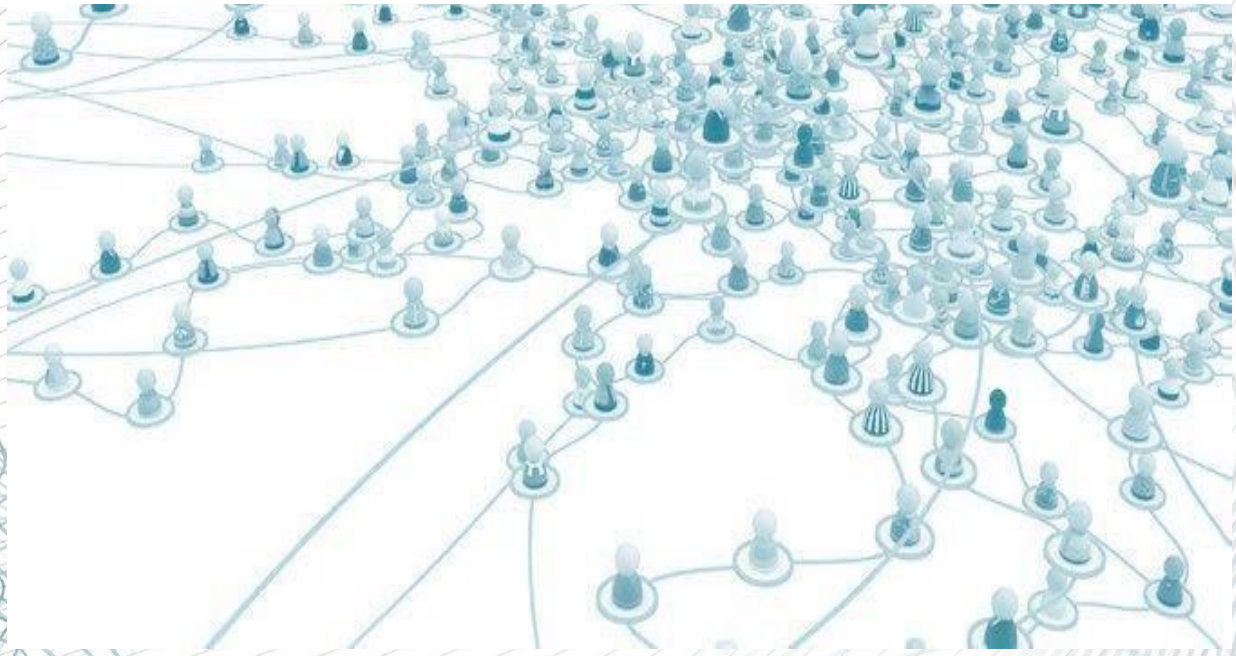
MODUL PRAKTIKUM

SD25-31001 DATA MINING



Program Studi Sains Data
Fakultas Sains
Institut Teknologi Sumatera

2025

MODUL 6

Unsupervised Learning:

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

A. Konsep Dasar

Secara fundamental, semua metode *clustering* menggunakan pendekatan yang serupa, yaitu dengan terlebih dahulu menghitung tingkat kesamaan antar data, kemudian menggunakan informasi tersebut untuk mengelompokkan titik-titik data ke dalam beberapa grup atau himpunan. *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) merupakan metode *unsupervised learning* yang populer dan banyak digunakan dalam pembuatan model serta algoritma machine learning. Metode ini pertama kali diperkenalkan oleh Ester et al. pada tahun 1996. Untuk menjalankan algoritma DBSCAN, tidak diperlukan masukan berupa jumlah kluster. Namun, DBSCAN memerlukan penyetelan terhadap parameter penting, yaitu:

1) Parameter eps (epsilon)

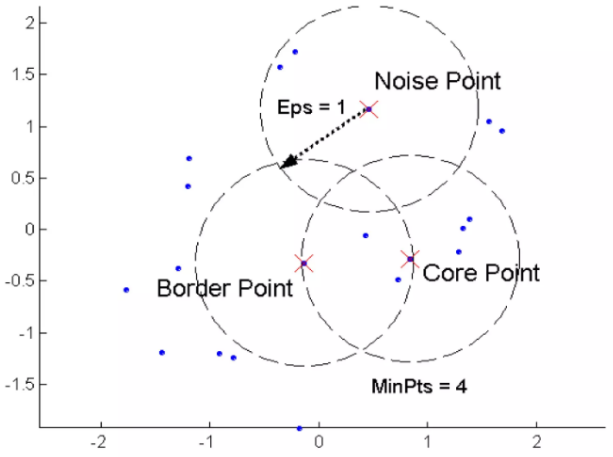
Parameter ini setara dengan jari-jari (radius) lingkaran yang menentukan jarak maksimum antara dua titik data agar dianggap berada dalam lingkungan (neighborhood) yang sama. Jika jarak antara dua titik lebih kecil atau sama dengan nilai eps, maka keduanya dianggap sebagai tetangga. Pemilihan nilai eps yang terlalu kecil dapat menyebabkan banyak titik dianggap sebagai outlier, sedangkan nilai yang terlalu besar dapat menyebabkan kluster saling bergabung sehingga sebagian besar titik data berada dalam satu kluster. Salah satu cara untuk menentukan nilai eps adalah melalui grafik k-distance

2) Parameter MinPts

Parameter ini menunjukkan jumlah minimum titik data dalam suatu lingkungan agar dapat dianggap sebagai kluster. Dengan kata lain, ini adalah jumlah minimum tetangga dalam radius eps. Semakin besar ukuran dataset, semakin besar pula nilai MinPts yang sebaiknya digunakan. Secara umum, nilai minimum MinPts dapat ditentukan berdasarkan jumlah dimensi data (D) dengan rumus: $\text{MinPts} \geq D + 1$. Nilai MinPts minimal yang disarankan adalah 3.

Dalam algoritma DBSCAN, terdapat tiga jenis titik data, yaitu:

1) Core Point: Titik yang memiliki jumlah tetangga lebih dari MinPts dalam radius eps.
2) Border Point: Titik yang memiliki jumlah tetangga kurang dari MinPts, tetapi berada dalam lingkungan dari core point.
3) Noise (Outlier): Titik yang bukan merupakan core point maupun border point.



Gambar 1. Model *DBScan*

Model DBSCAN memiliki beberapa keunggulan diantaranya mampu mendeteksi bentuk geometris rumit yang tidak dapat dideteksi oleh model pengelompokan lain, tidak terpengaruh oleh outlier, karena mampu mengelolanya dengan benar. Selain itu tidak perlu menentukan jumlah klaster secara subjektif, tetapi jumlah klaster akan dibuat secara otomatis, tergantung pada parameter yang telah kita pilih. Namun, ada juga beberapa kekurangannya, seperti tidak deterministik, karena titik-titik yang dapat dijangkau dapat dicapai oleh kluster yang berbeda dan dalam eksekusi yang berbeda, titik-titik tersebut dapat ditetapkan ke kluster yang berbeda. Jika terdapat beberapa wilayah dan masing-masing wilayah memiliki kepadatan berbeda, pemilihan parameter DBSCAN dapat menjadi rumit .

B. Tujuan Praktikum

I. Tujuan Instruksional Umum

Praktikum bertujuan untuk menerapkan teori algoritma *DBSCAN* untuk klasterisasi pada penambangan data.

II. Tujuan Instruksional Khusus

1. Mahasiswa mampu menguasai konsep dasar algoritma *DBSCAN*.
2. Mahasiswa mampu menganalisis studi kasus menggunakan metode *DBSCAN*.
3. Mahasiswa mampu menampilkan visualisasi dan menganalisis hasil klasterisasi dengan algoritma *DBSCAN*.

C. Dataset dan Bahasa Pemrograman Python

Dataset yang digunakan pada praktikum kali ini yaitu dataset 1 berupa fungsi yang menghasilkan titik-titik acak yang membentuk lingkaran dengan radius r, ditambah dengan gangguan (noise) dari distribusi normal. Secara matematis, setiap titik *(x,y)* yang dihasilkan dapat dinyatakan dengan persamaan: dengan rincian sebagai berikut:

$$x_i = r \cdot \cos\left(\frac{2\pi i}{n}\right) + \epsilon_x,$$
$$y_i = r \cdot \sin\left(\frac{2\pi i}{n}\right) + \epsilon_y,$$

Dataset 2 berupa data lokasi stasiun cuaca di USA, download dataset.

Berikut ini daftar *library* beserta fungsinya yang digunakan dalam praktikum untuk membantu proses pengolahan data, pelatihan model, evaluasi, dan visualisasi hasil.

Tabel 1. daftar library dan deskripsi

Kode	Deskripsi
sklearn	Digunakan sebagai pustaka utama untuk pemrosesan dan penerapan algoritma machine learning.
from sklearn import datasets	Menyediakan dataset bawaan seperti Iris atau make_blobs untuk percobaan clustering.
matplotlib.pyplot	Membuat visualisasi seperti scatter plot untuk menampilkan hasil clustering DBSCAN.
from sklearn.cluster import DBSCAN	Digunakan untuk melakukan klasterisasi berbasis kepadatan (Density-Based Spatial Clustering).
math	Digunakan untuk operasi matematika seperti sinus, kosinus, dan perhitungan sudut.
matplotlib	Library inti yang digunakan oleh pyplot untuk membuat grafik dan visualisasi data.
from sklearn.neighbors import NearestNeighbors	Digunakan untuk mencari tetangga terdekat, seperti dalam perhitungan nilai epsilon pada DBSCAN.

D. Langkah-langkah implementasi DBSCAN

Berikut adalah langkah-langkah yang dapat dilakukan:

STEP 1: Temukan semua *neighbor points* dalam jarak *eps* dan identifikasi *core points*, yaitu titik yang memiliki lebih dari *MinPts neighbors*.

STEP 2: Untuk setiap *core point* yang belum ditetapkan ke dalam klaster, buat *new cluster.*

STEP 3: Temukan secara rekursif semua *density connected points* dan masukkan ke dalam klaster yang sama dengan *core point* tersebut.

STEP 4: Dua titik, a dan b, dikatakan *density connected* jika terdapat titik c yang memiliki jumlah *neighbors* yang cukup dan kedua titik a dan b berada dalam jarak *eps*. Proses ini bersifat *chaining process* — misalnya jika b adalah *neighbor* dari c, c adalah *neighbor* dari d, d adalah *neighbor* dari e, dan e adalah *neighbor* dari a, maka b juga dianggap sebagai *neighbor* dari a. Ulangi langkah ini untuk semua titik yang belum dikunjungi dalam dataset.

Note: Titik-titik yang tidak termasuk ke dalam klaster mana pun dianggap sebagai *noise*.

1. *Import libraries*

   Beberapa library Python digunakan untuk membangun model DBSCAN, seperti dijelaskan pada Tabel 1.

```python
# Importing Libraries

from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
#from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from sklearn.metrics import silhouette_score
from google.colab import files
from IPython.display import Image
import math
import matplotlib.pyplot as plt
import matplotlib
#from sklearn.datasets import make_blobs # Corrected import
from sklearn.neighbors import NearestNeighbors
import warnings
warnings.filterwarnings("ignore")
```

2. *Import data*

Pada langkah ini dibuat dataset dengan hanya dua fitur agar mudah divisualisasikan. Fungsi PointsInCircum digunakan untuk membuat dataset tersebut, dengan menerima radius dan jumlah titik data sebagai argumen, kemudian menghasilkan array berisi titik-titik data yang, ketika diplot, membentuk sebuah lingkaran. Proses ini dilakukan dengan memanfaatkan kurva sin dan cosine.

```
np.random.seed(42)
# Function for creating datapoints in the form of a circle
def PointsInCircum(r,n=100):
    return [(math.cos(2*math.pi/n*x)*r+np.random.normal(-30,30),
             math.sin(2*math.pi/n*x)*r+np.random.normal(-30,30)) for x in range(1,n+1)]
```

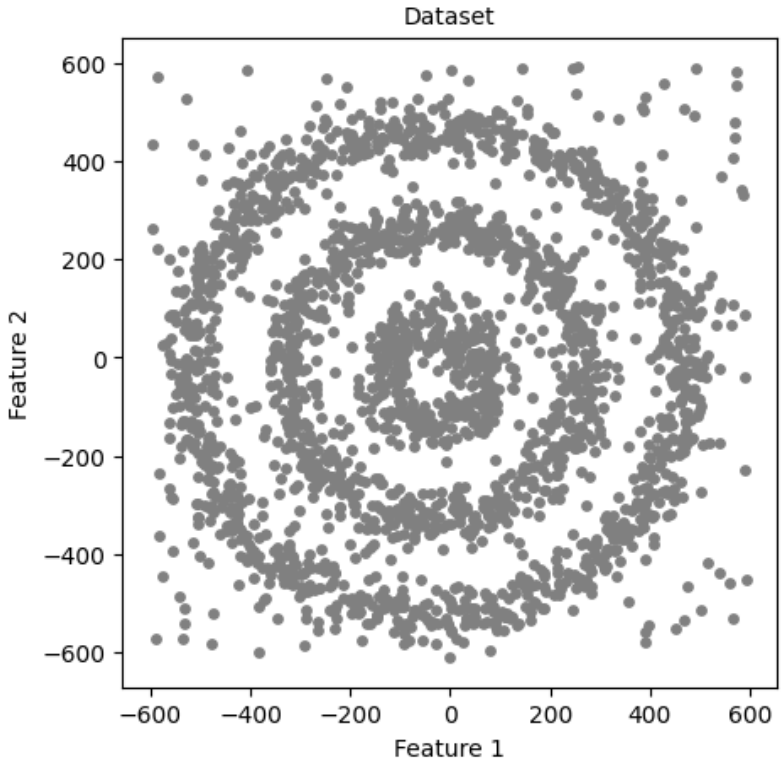
Satu lingkaran saja tidak cukup untuk melihat kemampuan clustering dari DBSCAN. Oleh karena itu, dibuat tiga lingkaran konsentris dengan jari-jari yang berbeda. Selain itu, ditambahkan pula noise pada data agar dapat diamati bagaimana berbagai algoritma clustering menangani keberadaan noise.

```
# Creating data points in the form of a circle
df1=pd.DataFrame(PointsInCircum(500,1000))
df2=pd.DataFrame(PointsInCircum(300,700))
df3=pd.DataFrame(PointsInCircum(100,300))

# Adding noise to the dataset
noise_df = pd.DataFrame([(np.random.randint(-600,600),np.random.randint(-600,600)) for i in range(300)])

# Concatenate the DataFrames
df = pd.concat([df1, df2, df3, noise_df], ignore_index=True)

print(df.head())
```

Tampilkan plot titik-titik data tersebut dalam feature space, Pada tahap ini digunakan scatter plot untuk menampilkan titik-titik data tersebut. Gunakan sintaks berikut:

```
plt.figure(figsize=(5,5))
plt.scatter(df[0],df[1],s=15,color='grey')
plt.title('Dataset',fontsize=10)
plt.xlabel('Feature 1',fontsize=10)
plt.ylabel('Feature 2',fontsize=10)
plt.show()
```

Output:



3. *DBSCAN default mode*

Pada bagian ini, nilai epsilon (eps) ditetapkan sebesar 0.5, dan min_samples atau minPoints bernilai 5.

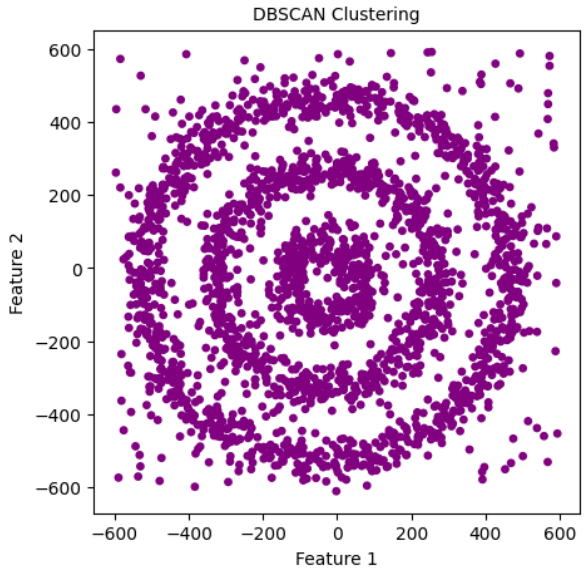
```
from sklearn.cluster import DBSCAN

dbscan=DBSCAN() #Here, epsilon is 0.5, and min_samples or minPoints is 5.
dbscan.fit(df[[0,1]])
```

```
DBSCAN
DBSCAN()
```

Selanjutnya, hasil dari model tersebut akan divisualisasikan.

```
# DB Scan Plot
df['DBSCAN_labels']=dbscan.labels_

# Plotting resulting clusters
colors=['purple','red','blue','green'] # Define colors here
plt.figure(figsize=(5,5))
plt.scatter(df[0],df[1],c=df['DBSCAN_labels'],cmap=matplotlib.colors.ListedColormap(colors),s=15)
plt.title('DBSCAN Clustering',fontsize=10)
plt.xlabel('Feature 1',fontsize=10)
plt.ylabel('Feature 2',fontsize=10)
plt.show()
```

Output:



Semua titik data kini berwarna ungu, yang berarti semuanya dianggap sebagai noise. Hal ini terjadi karena nilai epsilon terlalu kecil dan parameter belum dioptimalkan. Oleh karena itu, perlu dilakukan pencarian nilai epsilon dan minPoints yang tepat, kemudian model dilatih kembali.

4. *Cara memilih parameter di DBSCAN*

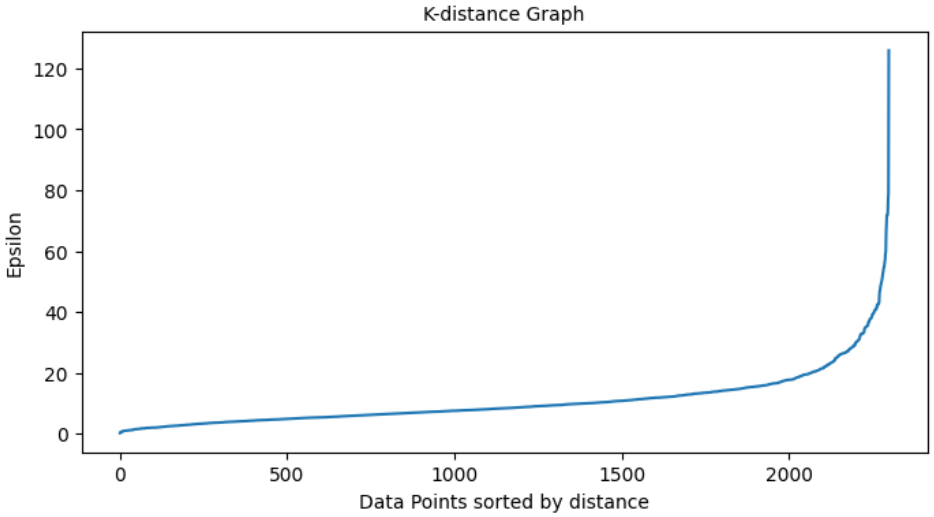
Menentukan nilai epsilon, digunakan K-distance graph. Untuk membuat *K-distance graph*, diperlukan jarak antara setiap titik dengan nearest data point-nya untuk semua titik dalam dataset. Jarak tersebut diperoleh menggunakan *NearestNeighbors* dari modul *sklearn.neighbors*.

```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(df[[0,1]])
distances, indices = nbrs.kneighbors(df[[0,1]])
```

Variabel distance berisi array yang merepresentasikan jarak antara setiap titik data dengan nearest data point-nya di seluruh dataset. Selanjutnya, plot K-distance graph untuk menentukan nilai epsilon. Gunakan sintaks berikut:

```
# Plotting K-distance Graph
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(8,4))
plt.plot(distances)
plt.title('K-distance Graph',fontsize=10)
plt.xlabel('Data Points sorted by distance',fontsize=10)
plt.ylabel('Epsilon',fontsize=10)
plt.show()
```

Output:



Nilai epsilon yang optimal terletak pada titik dengan kurva paling tajam pada *K-Distance Graph*, yaitu pada nilai 30 dalam kasus ini. Selanjutnya, perlu ditentukan nilai *minPoints*, yang umumnya bergantung pada domain *knowledge*. Pada tahap ini, nilai *minPoints* ditetapkan sebesar 6.

5. *DBSCAN* dengan parameter baru

Nilai epsilon yang optimal hasil dari *K-Distance Graph*, yaitu pada nilai 30, dengan *Points* ditetapkan sebesar 6 (menyesuaikan)

```
dbscan_opt=DBSCAN(eps=30,min_samples=6)
dbscan_opt.fit(df[[0,1]])
```

```
DBSCAN(eps=30, min_samples=6)
```

Hal paling menarik dari DBSCAN adalah kemampuannya memisahkan noise dari dataset dengan sangat baik. Dalam hasil ini, label 0, 1, dan 2 mewakili tiga klaster yang berbeda, sedangkan label -1 menunjukkan noise.

```
df['DBSCAN_opt_labels']=dbscan_opt.labels_
df['DBSCAN_opt_labels'].value_counts()
```

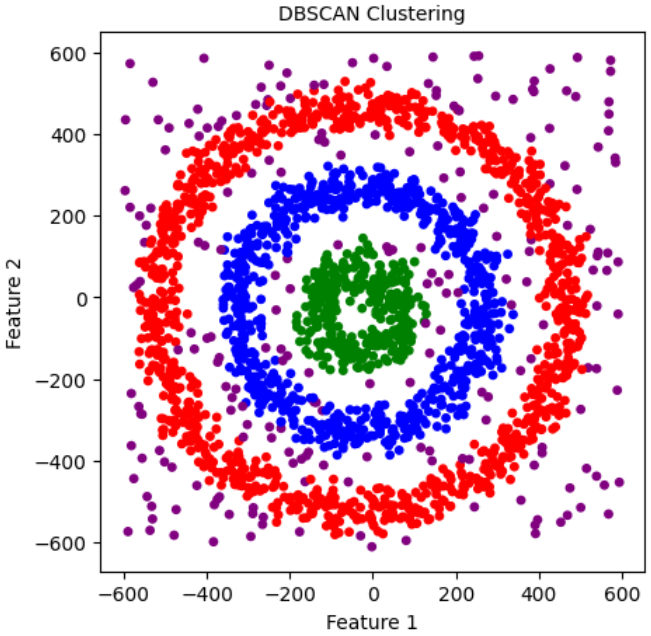
DBSCAN_opt_labels	count
0	1030
1	730
2	318
-1	222

dtype: int64

Selanjutnya, hasil tersebut akan diplot untuk melihat visualisasinya.

```
# Plotting the resulting clusters
plt.figure(figsize=(5,5))
plt.scatter(df[0],df[1],c=df['DBSCAN_opt_labels'],cmap=matplotlib.colors.ListedColormap(colors),s=15)
plt.title('DBSCAN Clustering',fontsize=10)
plt.xlabel('Feature 1',fontsize=10)
plt.ylabel('Feature 2',fontsize=10)
plt.show()
```

Output:



6. *DBSCAN Algorithm from Scratch in Python*

Pendekatan DBSCAN dapat dilakukan dengan membangun algoritma secara manual tanpa menggunakan pustaka seperti *sklearn*, sehingga setiap langkah — mulai dari perhitungan jarak hingga pembentukan cluster — dapat dipahami secara mendalam. Berbeda dengan versi *sklearn* yang otomatis, pendekatan ini memberikan fleksibilitas untuk memodifikasi logika dan mengamati langsung pengaruh parameter terhadap hasil klasterisasi. Code DBSCAN Algorithm from Scratch dapat anda download pada link berikut: link code DBSCAN.

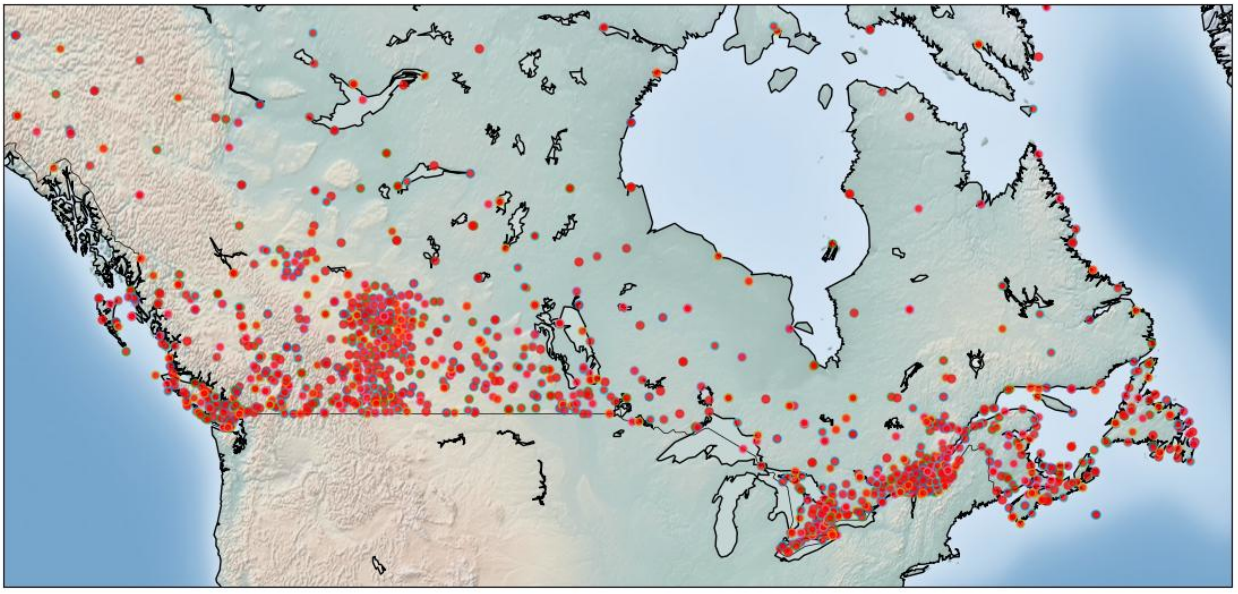
7. Implementasi DBSCAN (Contoh kasus: Clustering weather stations in USA)

DBSCAN sangat baik digunakan untuk tugas seperti identifikasi kelas dalam konteks spasial. Keunggulan utama algoritma DBSCAN adalah kemampuannya menemukan klaster dengan bentuk yang tidak beraturan tanpa terpengaruh oleh noise. Sebagai contoh, pada kasus ini DBSCAN digunakan untuk mengelompokkan lokasi stasiun cuaca di USA.

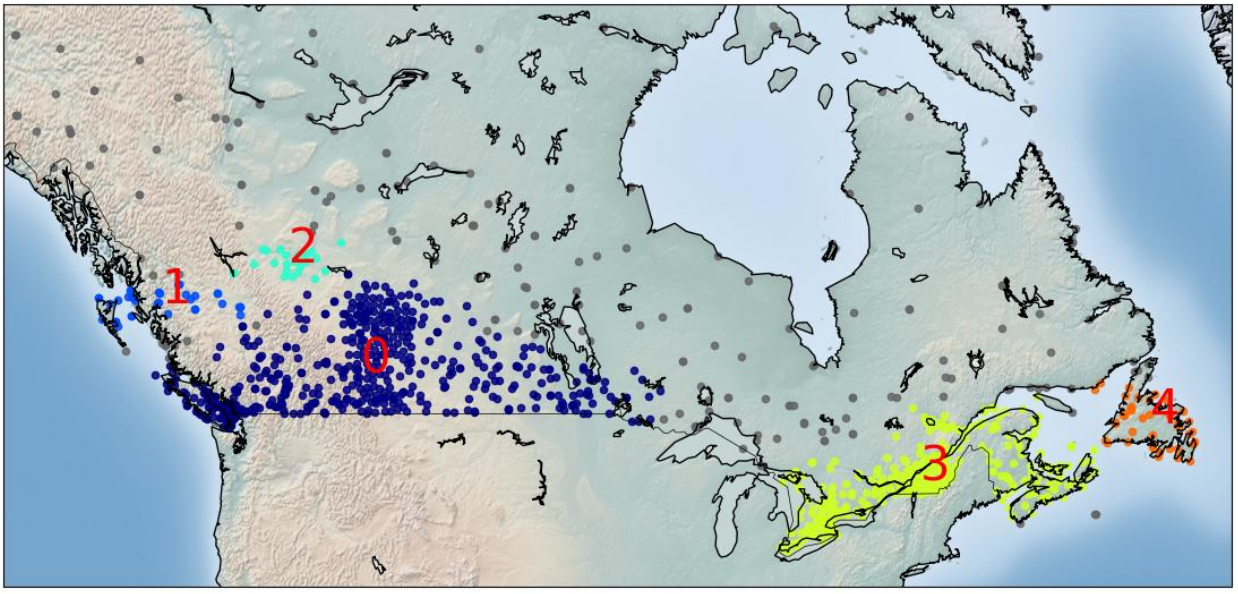
DBSCAN dapat membantu menemukan kelompok stasiun yang menunjukkan kondisi cuaca serupa. Seperti terlihat, algoritma ini tidak hanya mampu mendeteksi

klaster dengan bentuk yang beragam, tetapi juga dapat menemukan area dengan kepadatan tinggi pada data sambil mengabaikan area yang kurang padat atau dianggap noise. Download link Code.

Plot persebaran data:



Hasil Klasterisasi dengan DBSCAN:



E. Tugas Individu

1. Pilih minimal 2 dari fungsi berikut untuk menghasilkan data. Lakukan klasterisasi dengan DBSCAN dari sklearn (ikuti langkah langkah-langkah pada bagian D 1-5).

```
def archimedean_spiral(a, b, turns, n_points):
    theta = np.linspace(0, turns * 2 * np.pi, n_points)
    r = a + b * theta
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return np.vstack((x, y)).T + np.random.normal(scale=0.1, size=(n_points, 2))

def lemniscate(a, n_points):
    theta = np.linspace(0, 2*np.pi, n_points*2)
    cos2 = np.cos(2*theta)
    valid = cos2 >= 0
    theta = theta[valid][:n_points]
    r = a * np.sqrt(np.cos(2*theta))
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return np.vstack((x, y)).T + np.random.normal(scale=0.05, size=(len(theta), 2))

def cardioid(a, n_points):
    theta = np.linspace(0, 2*np.pi, n_points)
    r = a * (1 - np.cos(theta))
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return np.vstack((x, y)).T + np.random.normal(scale=0.05, size=(n_points, 2))

def concentric_circles(radii, points_per_circle):
    pts = []
    for r in radii:
        theta = np.linspace(0, 2*np.pi, points_per_circle, endpoint=False)
        x = r * np.cos(theta)
        y = r * np.sin(theta)
        p = np.vstack((x, y)).T + np.random.normal(scale=0.03, size=(points_per_circle, 2))
        pts.append(p)
    return np.vstack(pts)

def sine_wave_cluster(x_min, x_max, n_points, amplitude, frequency, offset):
    x = np.linspace(x_min, x_max, n_points)
    y = amplitude * np.sin(frequency * x) + offset
    return np.vstack((x, y)).T + np.random.normal(scale=0.08, size=(n_points, 2))

def gaussian_blob(center, cov, n_points):
    return np.random.multivariate_normal(mean=center, cov=cov, size=n_points)
```

2. Menggunakan data yang sama dengan nomor 1. Lakukan klasterisasi dengan model DBSCAN tanpa library (gunakan code pada D 6 untuk model DBSCAN).
3. Pilih salah satu dataset pada link berikut untuk klasterisasi dengan DBSCAN (gunakan code D 7)