

## ✓ Clustering Earthquakes

In this notebook, we use density based clustering functions - DBSCAN, HDBSCAN and OPTICS.


### Importing Baisc Libraries

1 Start coding or [generate](#) with AI.

```
1 import numpy as np
2 import pandas as pd
3 import geopandas as gpd
4 import matplotlib.pyplot as plt
```

### ✓ Reading Data

```
1 df = pd.read_csv('/content/EQ_data.csv')
2 df.head()
```

 <ipython-input-3-cf255d2903b4>:1: DtypeWarning: Columns (3,4,5) have mixed types. Specify dtype option  
df = pd.read\_csv('/content/EQ\_data.csv')

	#date	lat	lon	smajax	sminax	strike	q	depth	unc	q.1	...	mtp	mtt	str1	dip1	ra
0	1904-04-04 10:02:34.56	41.802	23.108	8.6	6.6	164.2	B	15.0	4.8	C	...					
1	1904-04-04 10:26:00.88	41.758	23.249	8.3	6.9	15.2	B	15.0	4.8	C	...					
2	1904-06-25 14:45:39.14	51.424	161.638	33.6	18.7	116.2	C	15.0	25.0	C	...					
3	1904-06-25 21:00:38.72	52.763	160.277	28.6	14.6	43.1	C	30.0	10.3	C	...					
4	1904-08-30 11:43:20.85	30.684	100.608	16.9	14.4	118.4	C	15.0	25.0	C	...					

5 rows × 31 columns



```
1 len(df)
```

 70881

```
1 locations = []
2 for i in range(len(df)):
3     locations.append((df['lat'][i], df['lon'][i]))
```

```
1 gdf = gpd.GeoDataFrame(geometry=gpd.points_from_xy([lon for lat, lon in locations],
2                                                     [lat for lat, lon in locations]))
3 gdf.head()
```



geometry



```
0 POINT (23.108 41.802)
1 POINT (23.249 41.758)
2 POINT (161.638 51.424)
3 POINT (160.277 52.763)
4 POINT (100.608 30.684)
```



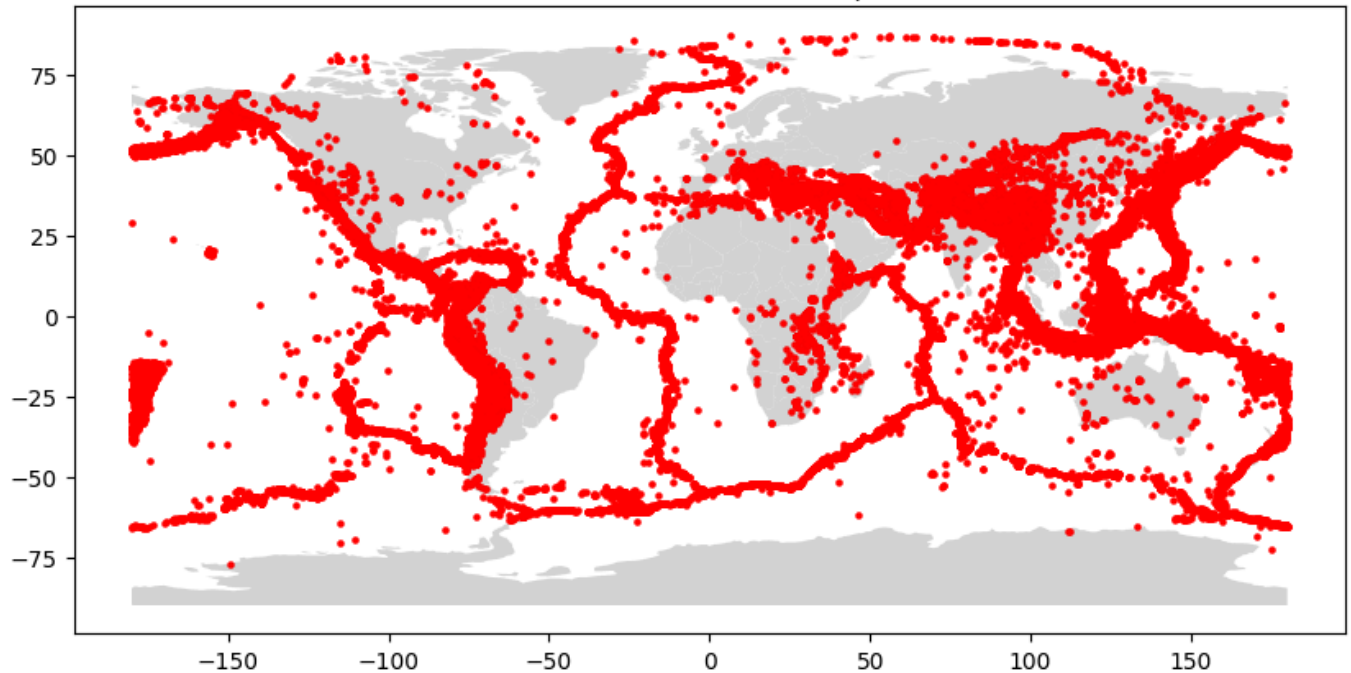
Next steps:

[Generate code with gdf](#)
[View recommended plots](#)
[New interactive sheet](#)

```
1 world = gpd.read_file("/content/ne_110m_admin_0_countries.shp")
2
3 fig, ax = plt.subplots(figsize=(10, 6))
4 world.plot(ax=ax, color='lightgray')
5
6 gdf.plot(ax=ax, color='red', markersize=5)
7
8 plt.title("Locations on World Map")
9 plt.show()
```



Locations on World Map



## ✓ Clustering Algorithms

### ✓ Importing Libraries for Clustering

```
1 from sklearn.cluster import DBSCAN
2 from sklearn import metrics
```

```

3 from sklearn.datasets import make_blobs
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.cluster import KMeans
6 from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
7 import hdbscan
8 from sklearn.cluster import OPTICS

```

## ▼ DBScan

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) groups points based on density. It works well when clusters have different shapes, but it struggles when densities vary too much. The main challenge is choosing eps—too small and you get too many noise points, too large and clusters merge incorrectly.

```

1 for i in range(len(locations)):
2     locations[i] = locations[i] + (df['depth'][i],)
3
4 locations = np.array(locations)
5 print(locations)

```

```

⇒ [[ 41.802  23.108  15.   ]
   [ 41.758  23.249  15.   ]
   [ 51.424 161.638  15.   ]
   ...
   [ -8.733 122.233 130.8   ]
   [ -0.793 146.809  14.4   ]
   [ -9.127 119.007  97.6   ]]

```

```

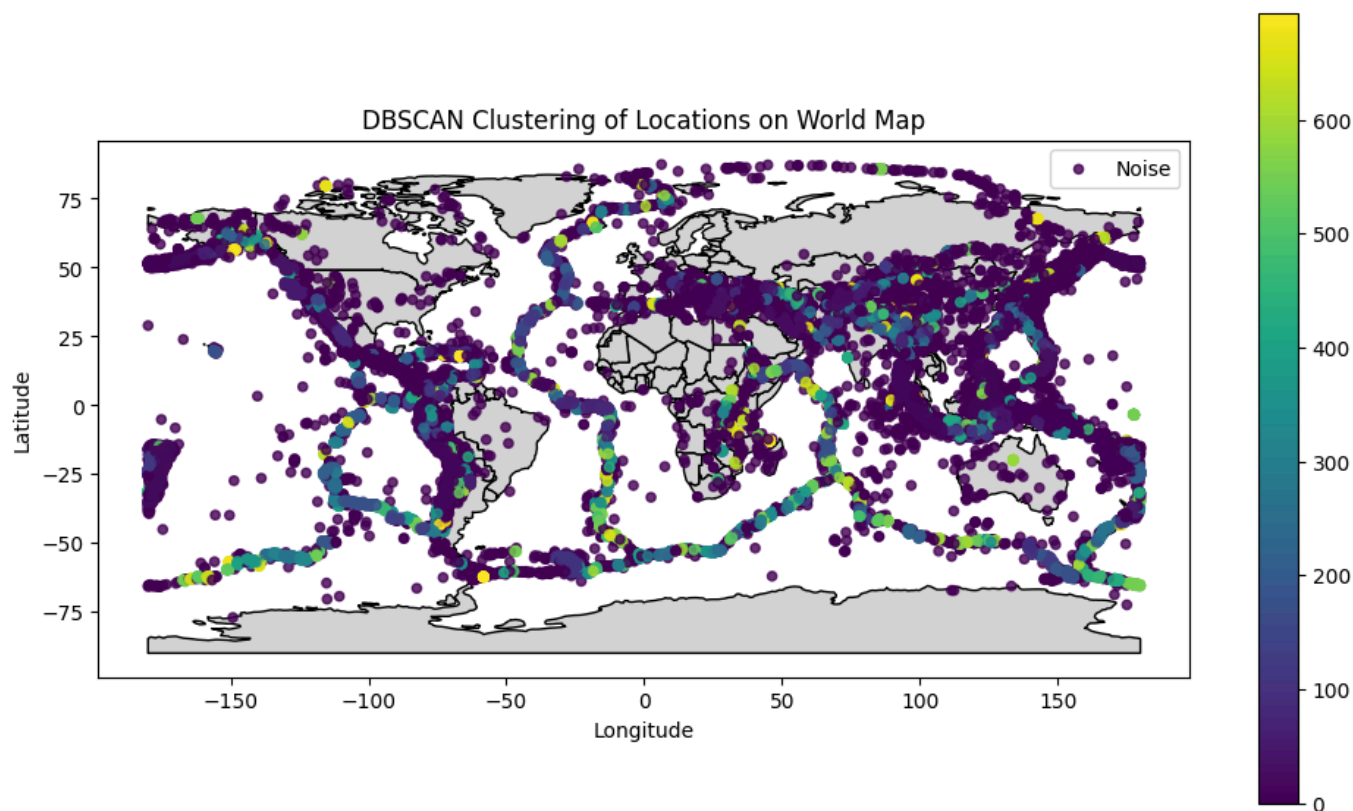
1 locations = np.array(locations)
2
3 # Normalize depth to balance its scale with latitude and longitude
4 scaler = StandardScaler()
5 locations[:, 2] = scaler.fit_transform(locations[:, 2].reshape(-1, 1)).flatten()
6
7 # Apply DBSCAN clustering
8 eps = 0.5 # Adjust based on data scale
9 min_samples = 7
10 db = DBSCAN(eps=eps, min_samples=min_samples, metric="euclidean").fit(locations)
11
12 # Get cluster labels
13 labels = db.labels_
14 unique_labels = set(labels)
15
16 # Convert clustered points into a GeoDataFrame
17 gdf = gpd.GeoDataFrame(
18     geometry=gpd.points_from_xy(locations[:, 1], locations[:, 0]),
19     data={"Cluster": labels}
20 )
21
22 # Plot world map
23 fig, ax = plt.subplots(figsize=(12, 7))
24 world.plot(ax=ax, color="lightgray", edgecolor="black")
25
26 # Plot clustered points with different colors
27 gdf.plot(ax=ax, column="Cluster", cmap="viridis", markersize=20, legend=True, alpha=0.8)
28
29 # Labels & Title

```

```

30 plt.xlabel("Longitude")
31 plt.ylabel("Latitude")
32 plt.title("DBSCAN Clustering of Locations on World Map")
33 plt.legend(["Noise", "Clusters"])
34 plt.show()

```



## ✓ Performance Metrics

We used Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Score to evaluate clustering quality.

Silhouette measures how well-separated clusters are, DB Index checks how compact they are, and CH Score favors distinct, well-spread clusters.

```

1 scaler = StandardScaler()
2 locations[:, 2] = scaler.fit_transform(locations[:, 2].reshape(-1, 1)).flatten()
3
4 # Define hyperparameter ranges
5 eps_values = np.linspace(0.5, 1.5, 6)
6 min_samples_values = range(3, 8)
7
8 # Store results
9 results = []
10
11 # Grid Search Over Parameters
12 for eps in eps_values:
13     print(eps)
14     for min_samples in min_samples_values:
15         print(min_samples, end = " ")
16

```

```

17 db = DBSCAN(eps=eps, min_samples=min_samples, metric="euclidean").fit(locations)
18 labels = db.labels_
19
20 # Remove noise points (-1) for evaluation
21 core_samples = locations[labels != -1]
22 core_labels = labels[labels != -1]
23
24 # Compute metrics only if we have at least 2 clusters
25 if len(set(core_labels)) > 1:
26     silhouette = silhouette_score(core_samples, core_labels)
27     db_index = davies_bouldin_score(core_samples, core_labels)
28     ch_score = calinski_harabasz_score(core_samples, core_labels)
29 else:
30     silhouette, db_index, ch_score = None, None, None
31
32 # Store results
33 results.append((eps, min_samples, silhouette, db_index, ch_score, len(set(core_labels))))
34
35 # Convert results to array
36 results = np.array(results, dtype=object)
37
38 # Extract metrics for plotting
39 eps_grid = results[:, 0].astype(float)
40 min_samples_grid = results[:, 1].astype(int)
41 sil_scores = np.array([x if x is not None else 0 for x in results[:, 2]])
42 db_scores = np.array([x if x is not None else np.nan for x in results[:, 3]])
43 ch_scores = np.array([x if x is not None else 0 for x in results[:, 4]])
44 num_clusters = results[:, 5].astype(int)
45
46 # Plot Results
47 fig, axes = plt.subplots(1, 3, figsize=(18, 5))
48
49 # Silhouette Score Plot
50 sc = axes[0].scatter(eps_grid, min_samples_grid, c=sil_scores, cmap="viridis", edgecolors="k")
51 axes[0].set_title("Silhouette Score")
52 axes[0].set_xlabel("Epsilon (eps)")
53 axes[0].set_ylabel("Min Samples")
54 fig.colorbar(sc, ax=axes[0])
55
56 # Davies-Bouldin Score Plot
57 sc = axes[1].scatter(eps_grid, min_samples_grid, c=db_scores, cmap="coolwarm", edgecolors="k")
58 axes[1].set_title("Davies-Bouldin Index (Lower is Better)")
59 axes[1].set_xlabel("Epsilon (eps)")
60 fig.colorbar(sc, ax=axes[1])
61
62 # Number of Clusters Found
63 sc = axes[2].scatter(eps_grid, min_samples_grid, c=num_clusters, cmap="plasma", edgecolors="k")
64 axes[2].set_title("Number of Clusters Found")
65 axes[2].set_xlabel("Epsilon (eps)")
66 fig.colorbar(sc, ax=axes[2])
67
68 plt.tight_layout()
69 plt.show()

```

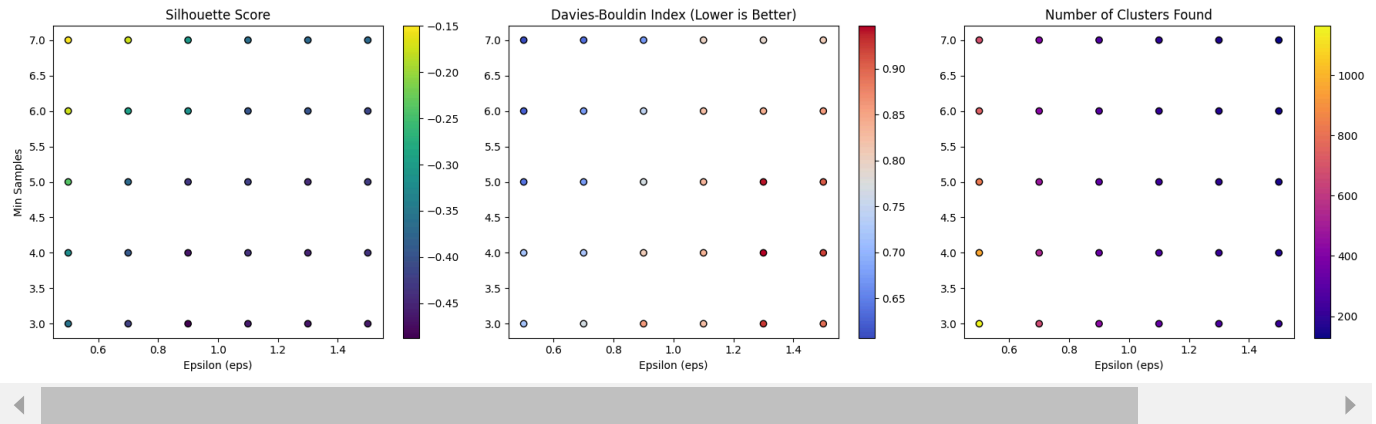


0.5

```

3 4 5 6 7 0.7
3 4 5 6 7 0.9
3 4 5 6 7 1.1
3 4 5 6 7 1.3
3 4 5 6 7 1.5
3 4 5 6 7

```



```

1 # Filter out None values from Silhouette Scores
2 valid_results = results[~np.isnan(sil_scores)]
3
4 if valid_results.size > 0:
5     # Get the index of the best Silhouette Score
6     best_idx = np.argmax(sil_scores)
7
8     # Extract the best parameters
9     best_eps = results[best_idx, 0]
10    best_min_samples = results[best_idx, 1]
11    best_silhouette_score = results[best_idx, 2]
12
13    print(f"✅ Best Hyperparameters Found:")
14    print(f"    - Epsilon (eps): {best_eps}")
15    print(f"    - Min Samples: {best_min_samples}")
16    print(f"    - Best Silhouette Score: {best_silhouette_score:.4f}")
17 else:
18    print("⚠️ No valid clustering found. Try adjusting the eps or min_samples range.")
19    best_eps, best_min_samples = None, None
20

```



```

✅ Best Hyperparameters Found:
- Epsilon (eps): 0.5
- Min Samples: 7
- Best Silhouette Score: -0.1499

```

A negative Silhouette Score means points are closer to other clusters than their own. This happens when clusters overlap or when too many points are marked as noise, making the remaining clusters poorly defined.


## ✓ HDBSCAN

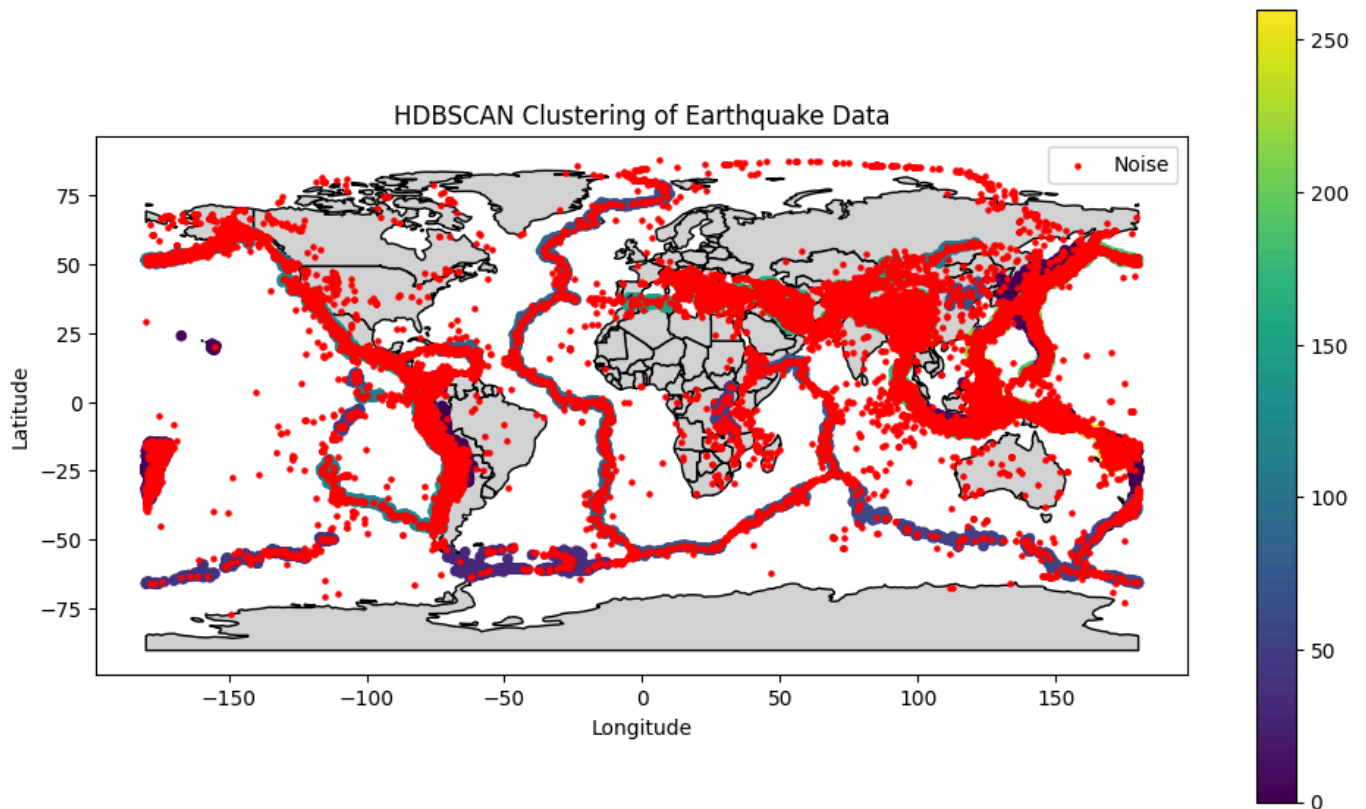
HDBSCAN is an improvement over DBSCAN that automatically finds the best eps and can identify clusters of different densities. It's great for large datasets and handles variable-density clustering better than DBSCAN. However, it still struggles when data lacks clear density variations.

```

1 clusterer = hdbscan.HDBSCAN(min_cluster_size=50, min_samples=10, metric="euclidean")
2 labels = clusterer.fit_predict(locations)
3
4 # Convert to GeoDataFrame for plotting
5 gdf = gpd.GeoDataFrame(
6     geometry=gpd.points_from_xy(locations[:, 1], locations[:, 0]),
7     data={"Cluster": labels}
8 )
9
10 # Plot results
11 fig, ax = plt.subplots(figsize=(12, 7))
12 world.plot(ax=ax, color="lightgray", edgecolor="black")
13
14 # Plot clustered points
15 gdf[gdf["Cluster"] != -1].plot(ax=ax, column="Cluster", cmap="viridis", markersize=20, legend=True)
16
17 # Plot noise points
18 gdf[gdf["Cluster"] == -1].plot(ax=ax, color="red", markersize=5, label="Noise")
19
20 plt.xlabel("Longitude")
21 plt.ylabel("Latitude")
22 plt.title("HDBSCAN Clustering of Earthquake Data")
23 plt.legend()
24 plt.show()

```

 /usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force\_all\_finite' was renamed to 'warn\_only' in 1.2.0. Please use 'warn\_only' instead of 'force\_all\_finite'.
 warnings.warn(
 /usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force\_all\_finite' was renamed to 'warn\_only' in 1.2.0. Please use 'warn\_only' instead of 'force\_all\_finite'.
 warnings.warn(



```

1 core_samples = locations[labels != -1]
2 core_labels = labels[labels != -1]
3
4 # Compute silhouette score only if at least 2 clusters exist
5 if len(set(core_labels)) > 1:
6     silhouette = silhouette_score(core_samples, core_labels)
7 else:
8     silhouette = None
9
10 print(f"✅ Silhouette Score: {silhouette}")

```

↗️ ✅ Silhouette Score: 0.44544709741024097

## ✓ Trying HDBSCAN for samller subset of the data

As HDBSCAN is producing decent results in comparison to other methods, we will try to reduce number of earthquakes to reduce density and evaluate performance

We will use random 10,000 recorded earthquakes

```

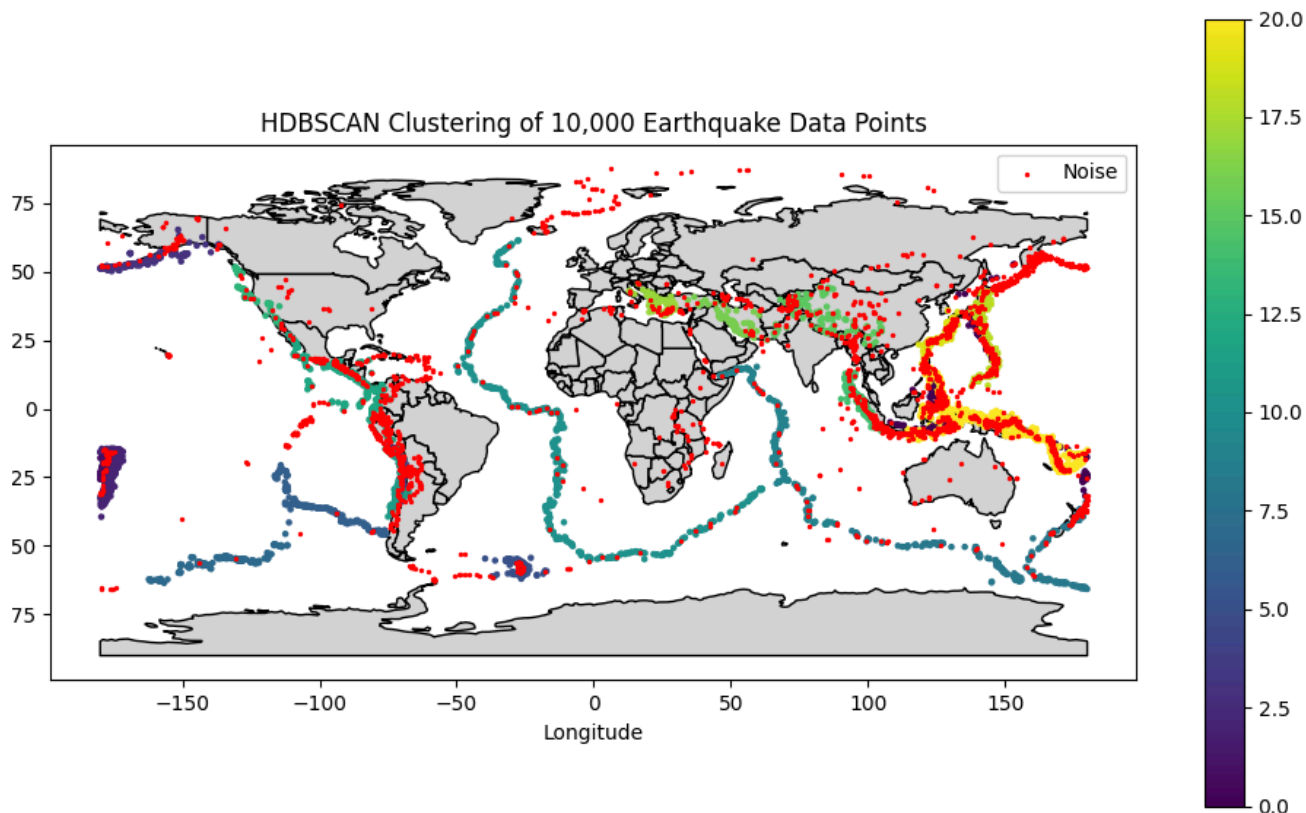
1 num_samples = 10000
2
3 # Randomly select 10,000 points without replacement
4 indices = np.random.choice(locations.shape[0], num_samples, replace=False)
5 location_new = locations[indices]
6
7 # Run HDBSCAN
8 clusterer = hdbscan.HDBSCAN(min_cluster_size=100, min_samples=10, metric="euclidean")
9 labels = clusterer.fit_predict(locations_new)
10
11 # Convert to GeoDataFrame for plotting
12 gdf = gpd.GeoDataFrame(
13     geometry=gpd.points_from_xy(locations_new[:, 1], locations_new[:, 0]), # Longitude, Latitude
14     data={"Cluster": labels}
15 )
16
17 # Plot results
18 fig, ax = plt.subplots(figsize=(12, 7))
19 world.plot(ax=ax, color="lightgray", edgecolor="black")
20
21 # Plot clustered points
22 gdf[gdf["Cluster"] != -1].plot(ax=ax, column="Cluster", cmap="viridis", markersize=5, legend=True)
23
24 # Plot noise points
25 gdf[gdf["Cluster"] == -1].plot(ax=ax, color="red", markersize=2, label="Noise")
26
27 plt.xlabel("Longitude")
28 plt.ylabel("Latitude")
29 plt.title("HDBSCAN Clustering of 10,000 Earthquake Data Points")
30 plt.legend()
31 plt.show()

```





```
'local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_fin:
nings.warn(
'local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_fin:
nings.warn(
```



```
1 core_samples = locations_new[labels != -1]
2 core_labels = labels[labels != -1]
3
4 # Compute silhouette score only if at least 2 clusters exist
5 if len(set(core_labels)) > 1:
6     silhouette = silhouette_score(core_samples, core_labels)
7 else:
8     silhouette = None
9
10 print(f"✅ Silhouette Score: {silhouette}")
```



✅ Silhouette Score: 0.3482536508616005

As observed when the dataset is reduced in size, the number of noise points reduces significantly and we get better results.

Number of clusters in this case = 9

## ✓ OPTICS

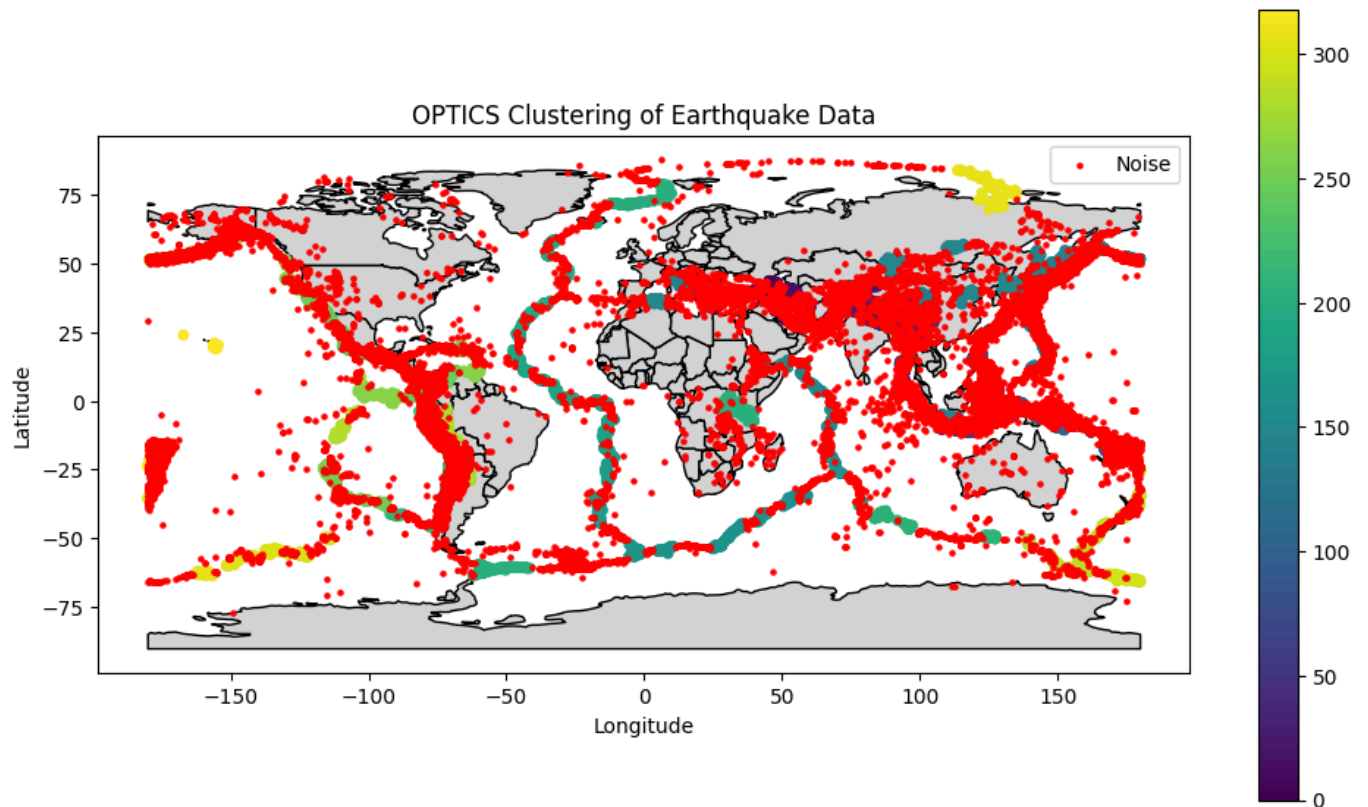
OPTICS is like DBSCAN but doesn't require a fixed eps. It orders points based on density, making it useful for finding clusters at multiple scales. It's more flexible than DBSCAN but can still produce too many noise points if the dataset

is highly variable.

```

1 optics = OPTICS(min_samples=10, xi=0.05, min_cluster_size=50, metric="euclidean")
2 labels = optics.fit_predict(locations)
3
4 # Convert to GeoDataFrame for plotting
5 gdf = gpd.GeoDataFrame(
6     geometry=gpd.points_from_xy(locations[:, 1], locations[:, 0]),
7     data={"Cluster": labels}
8 )
9
10 # Plot results
11 fig, ax = plt.subplots(figsize=(12, 7))
12 world.plot(ax=ax, color="lightgray", edgecolor="black")
13
14 # Plot clustered points
15 gdf[gdf["Cluster"] != -1].plot(ax=ax, column="Cluster", cmap="viridis", markersize=20, legend=True)
16
17 # Plot noise points
18 gdf[gdf["Cluster"] == -1].plot(ax=ax, color="red", markersize=5, label="Noise")
19
20 plt.xlabel("Longitude")
21 plt.ylabel("Latitude")
22 plt.title("OPTICS Clustering of Earthquake Data")
23 plt.legend()
24 plt.show()

```



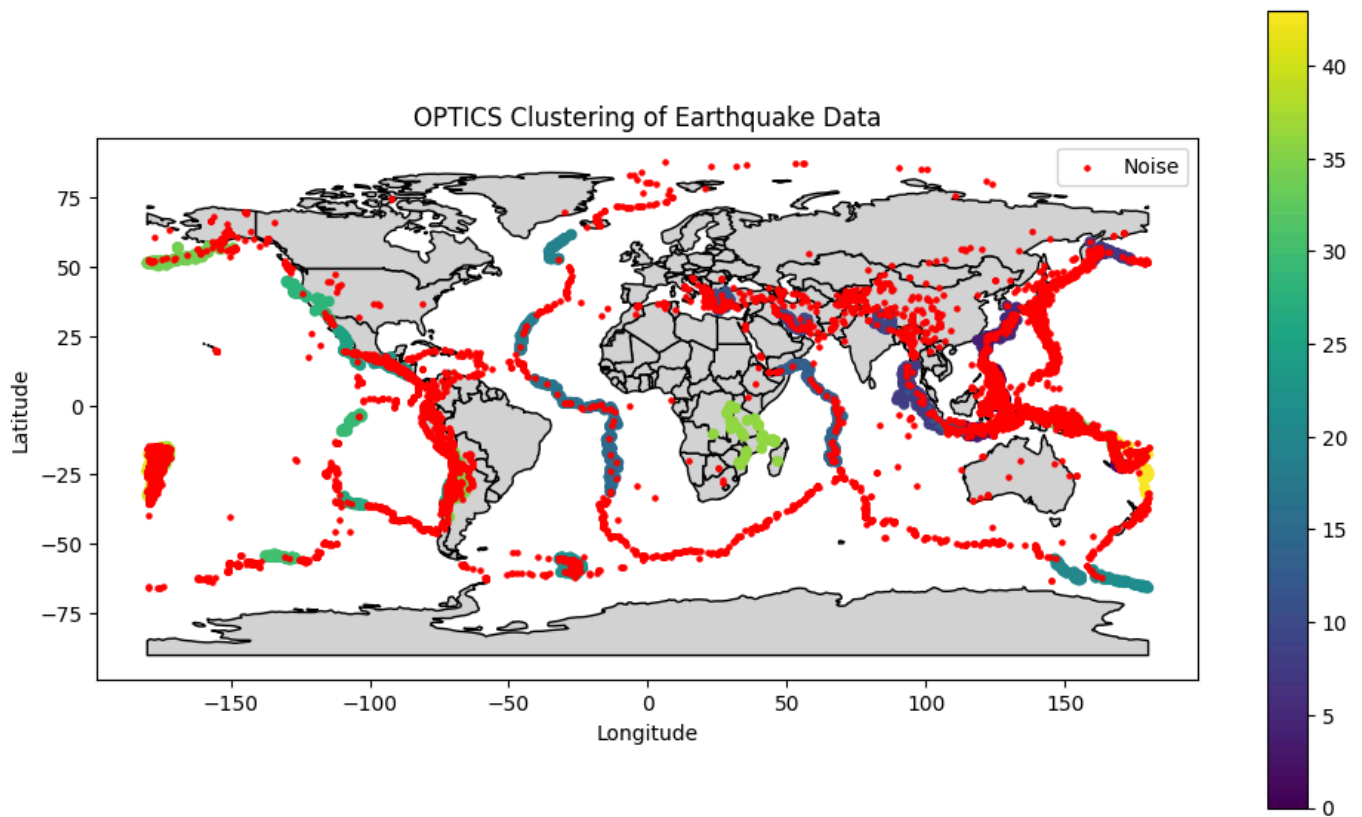
✓ Trying OPTICS for a smaller dataset

We will try OPTICS with last 10,000 recorded earthquakes

```

1 optics = OPTICS(min_samples=10, xi=0.05, min_cluster_size=50, metric="euclidean")
2 labels = optics.fit_predict(locations_new)
3
4 # Convert to GeoDataFrame for plotting
5 gdf = gpd.GeoDataFrame(
6     geometry=gpd.points_from_xy(locations_new[:, 1], locations_new[:, 0]),
7     data={"Cluster": labels}
8 )
9
10 # Plot results
11 fig, ax = plt.subplots(figsize=(12, 7))
12 world.plot(ax=ax, color="lightgray", edgecolor="black")
13
14 # Plot clustered points
15 gdf[gdf["Cluster"] != -1].plot(ax=ax, column="Cluster", cmap="viridis", markersize=20, legend=True)
16
17 # Plot noise points
18 gdf[gdf["Cluster"] == -1].plot(ax=ax, color="red", markersize=5, label="Noise")
19
20 plt.xlabel("Longitude")
21 plt.ylabel("Latitude")
22 plt.title("OPTICS Clustering of Earthquake Data")
23 plt.legend()
24 plt.show()

```



## Overall Conclusion

1. Density-based methods work well for spatial data like earthquakes but struggle when density varies too much or data is too sparse.
2. Since DBSCAN, HDBSCAN, and OPTICS all rely on density estimation, they can misclassify many points as noise.
3. For large datasets (70,000+ points), model-based clustering like Gaussian Mixture Models (GMM) or hierarchical clustering might perform better.
4. When we limit the number of points, we seemingly get better results in these density based functions.

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.