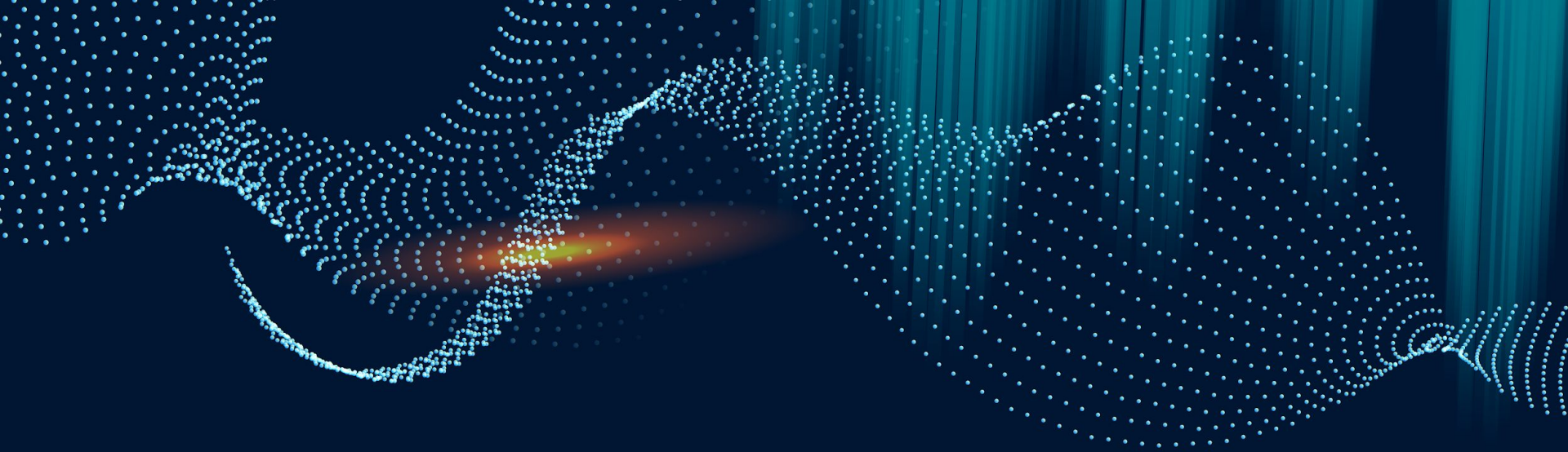
Machine learning

Clustering project

박유나, 임근영



01 K-Means

What is K-Means algorithm?

K-Means Algorithm

- 레이블이 없는 데이터셋을 서로 다른 클러스터로 그룹화하는 비지도 학습 알고리즘
- K = 사전 정의된 생성해야하는 클러스터의 수
- **Working mechanism**

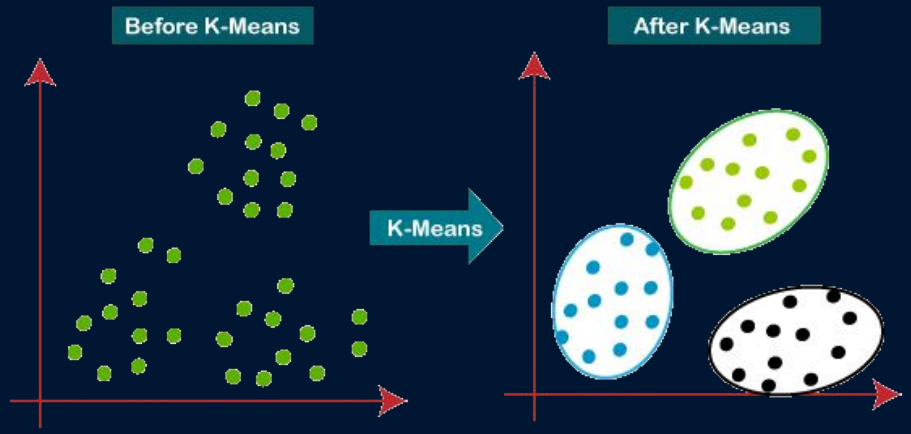
단계 **1** 클러스터의 수 **(K)**를 지정합니다 .

단계 **2.** K개의 인스턴스를 무작위로 선택하여 중심점 **(centroid)**을 초기화합니다 .

단계 **3.** 각 데이터 포인트를 거리 측정법을 사용하여 가장 가까운 중심점에 할당합니다 .

단계 **4.** 클러스터 내 모든 포인트의 평균을 계산하여 중심점을 재계산합니다 .

단계 **5.** 알고리즘이 중지 기준을 충족할 때까지 단계 **3**과 **4**를 반복합니다 .



출처 : https://www.analyticsvidhya.com/blog/2021/04/k-means-clustering-simplified-in-python/

K-Means Algorithm

입력인자 : 데이터(points), 최대 반복수(max_iter), 클러스터 수(k), initial(첫 centroids, 기본은 None), 수렴 판단 기준(tol, 기본은 0.0001)

```python
# kmeans 알고리즘
def kmeans(points, k, max_iter, initial = None, tol = 1e-4):
    # initial이 지정되지 않으면 주어진 데이터에서 k개 비복원추출
    if initial is None:
        initial = random.sample(list(points), k) # 파이썬의 random 알고리즘 사용

    # clustering하기
    centroids = initial
    iter = 0 # 언제 수렴하는지 알고자 추가
    for _ in range(max_iter):
        # 각 포인트 별로 가까운 centroids에 따라 cluster 만들기
        clusters = cluster(centroids, points)
        # 각 cluster의 평균으로 새로운 centroids 구하기
        new_centroids = update_centorids(points, clusters, k)
        # stop criterion : 이전 centorids와 이후 centroids의 차가 tol 보다 작아지면 수렴이라 판단하고 반복 중지
        if np.all(abs(centroids - new_centroids) < tol):
            break
        centroids = new_centroids # centroids를 새로운 centroids로 업데이트
        iter+= 1 # 반복 횟수 1 증가
    return clusters, centroids, iter
```

<- Step1: Number of Cluster

<- Step2: Initializing

<- Step5 : Iterating

<- Step3 : Labeling Points

<- Step4 : Calculating New Centroids

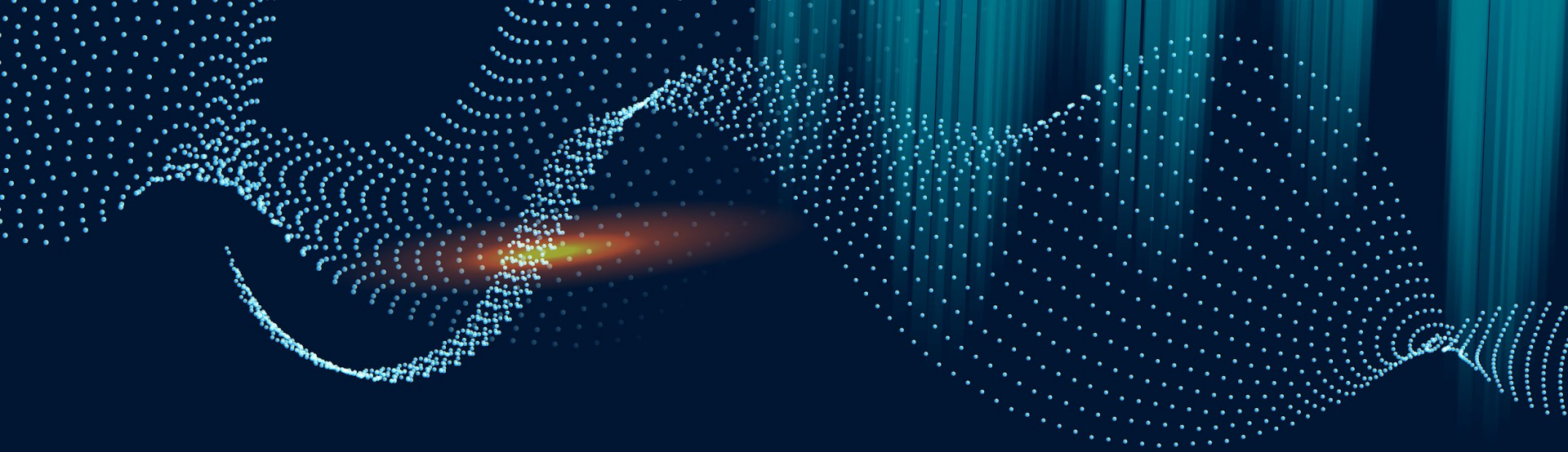
<- Stop Criterion

K-Means Algorithm

```python
# 유클리디안 거리 구하기
def euclidean(x, y):
    dist = np.sqrt(np.sum((x - y)**2))
    return dist

# 각 point 별로 가까운 centroids에 따라 cluster 만들기
def cluster(centroids, points):
    cluster = list()
    for point in points:
        # 포인트와 각 centroid마다의 거리 구하기
        distance = [euclidean(point, centroid) for centroid in centroids]
        # 거리가 가장 짧은 centroid(index)를 cluster에 추가하기
        cluster.append(np.argmin(distance))
    return np.array(cluster)

# cluster별 평균으로 새로운 centroids 구하기
def update_centorids(points, cluster, k):
    new_centroids = list()
    for i in range(k): # cluster 수 만큼 반복
        cluster_points = points[cluster == i] # 각 cluster에 속하는 포인트 선택
        if len(cluster_points) == 0: # cluster에 데이터가 하나도 없을 경우 랜덤으로 centroids를 만듦
            new_centroids.append(random.choice(list(points)))
        else:
            # 각 cluster에 속하는 포인트의 평균으로 새로운 centroids 계산
            new_centroids.append(np.mean(cluster_points, axis=0))
    return np.array(new_centroids)
```



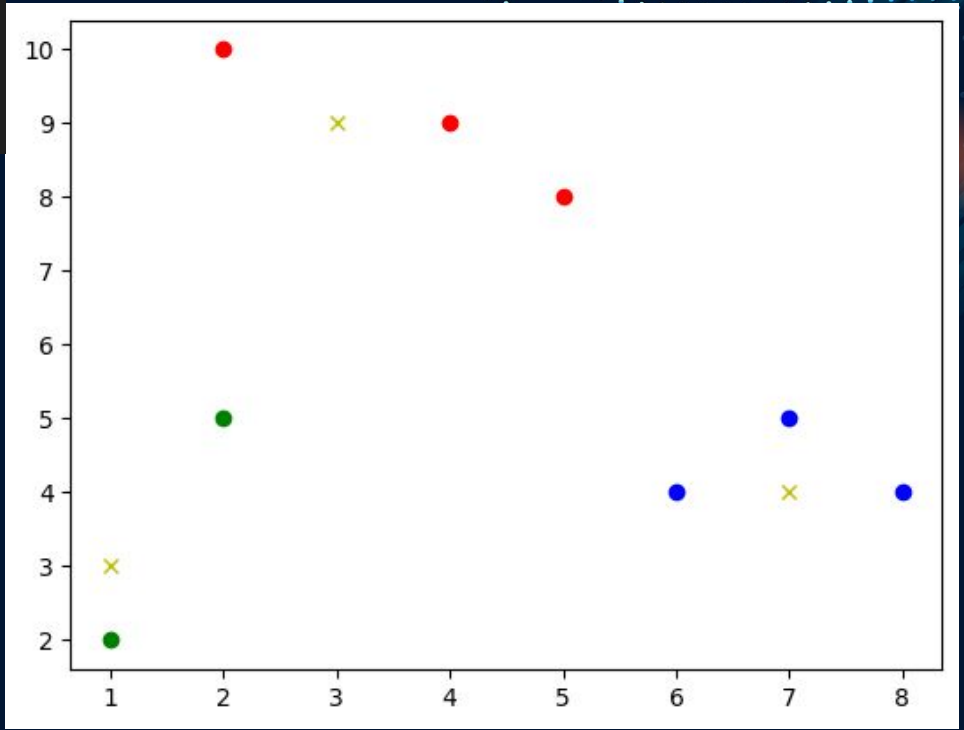
02 | Problem 1

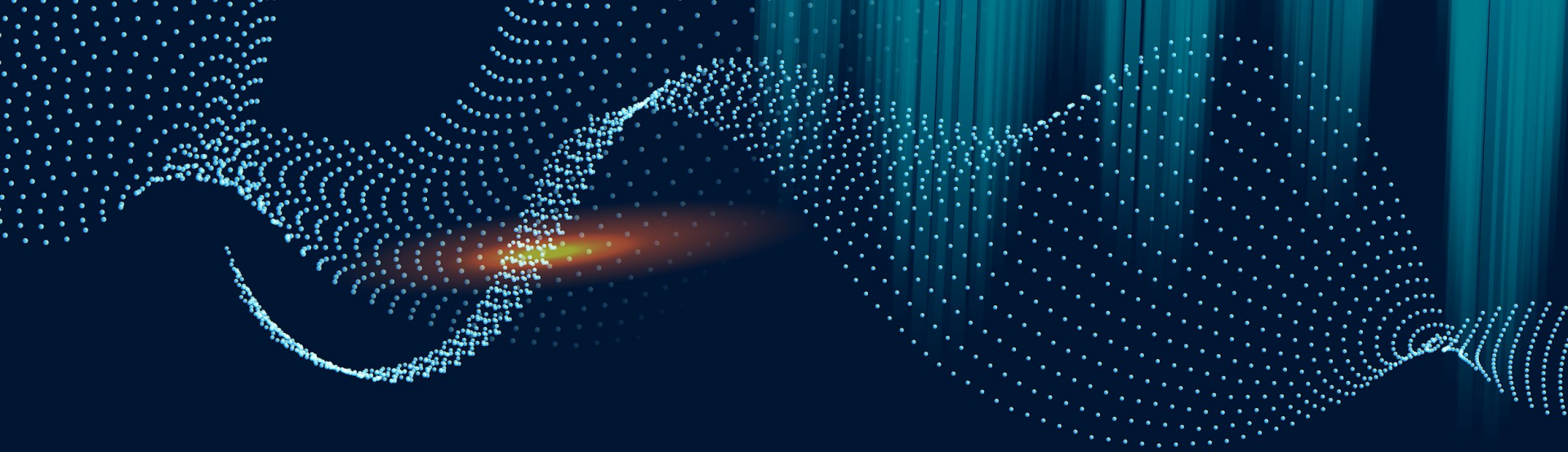
Problem 1 : toy problem

```
Clusters: [0 2 1 0 1 1 2 0]
Final Centroids: [[3.66666667 9.        ]
 [7.         4.33333333]
 [1.5        3.5       ]]
Iterations: 3
```

Basic idea

- K-Means



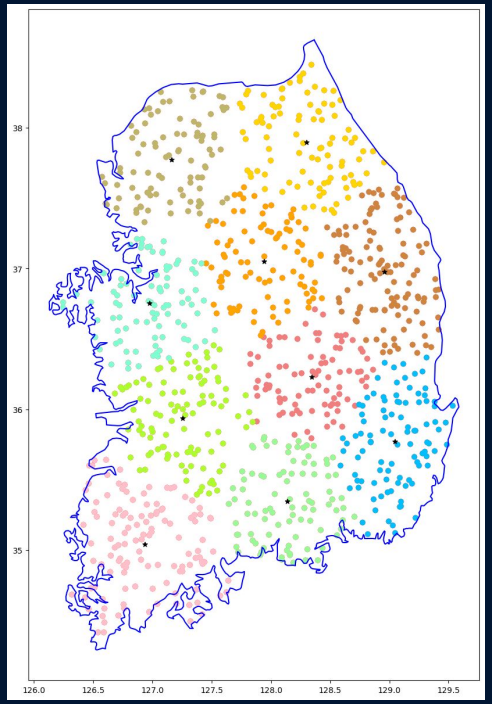


03 | Problem 2

Problem 2
: Open-ended problem

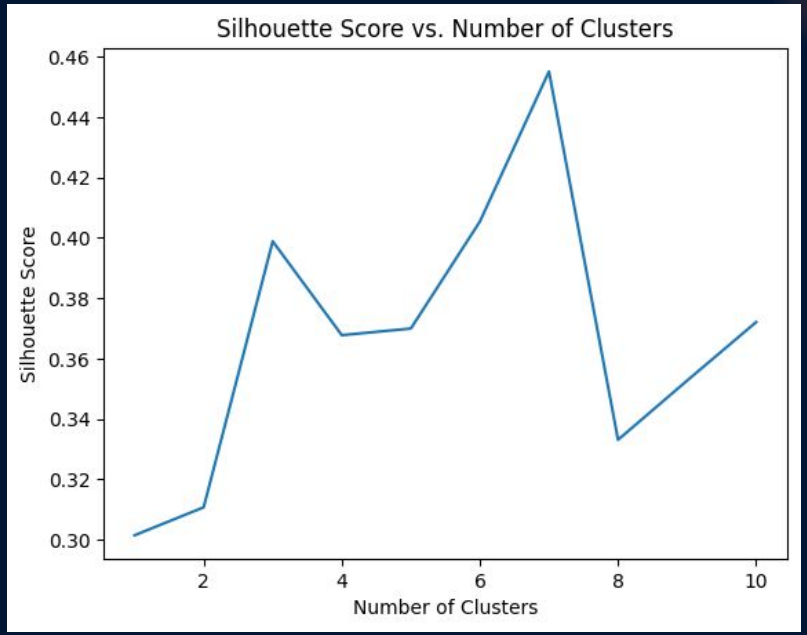
Basic idea

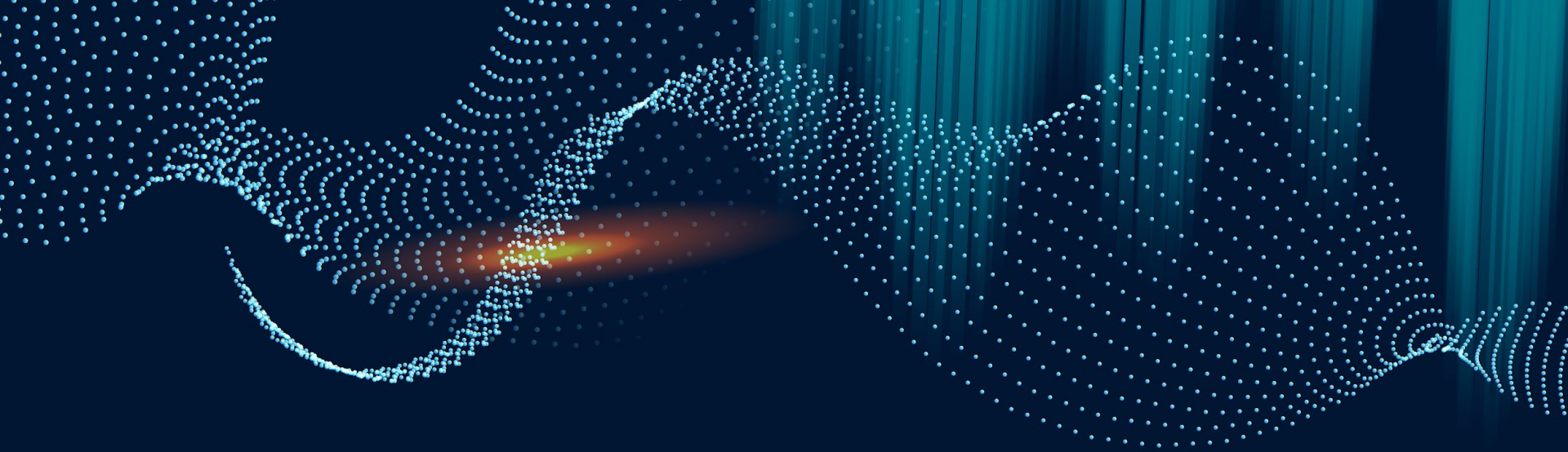
- Random point
- K-Means
- Stop criteria



Problem 2
: Open-ended problem

Model evaluation
- Silhouette score





04 | Problem 3

K-means when K = 17

```
coordinates = vertiport_candidates[['Latitude (deg)', 'Longitude (deg)']].values
num_clusters = 17

# kmeans 알고리즘 실시하기
clusters, final_centroids, iterations  = kmeans(k=num_clusters, max_iter = 100,
                                                initial= None, points = coordinates, tol = 1e-4)
print("Clusters:", clusters)
print("Final Centroids:", final_centroids)
print("Iterations:", iterations)
```

✓ 13.0s

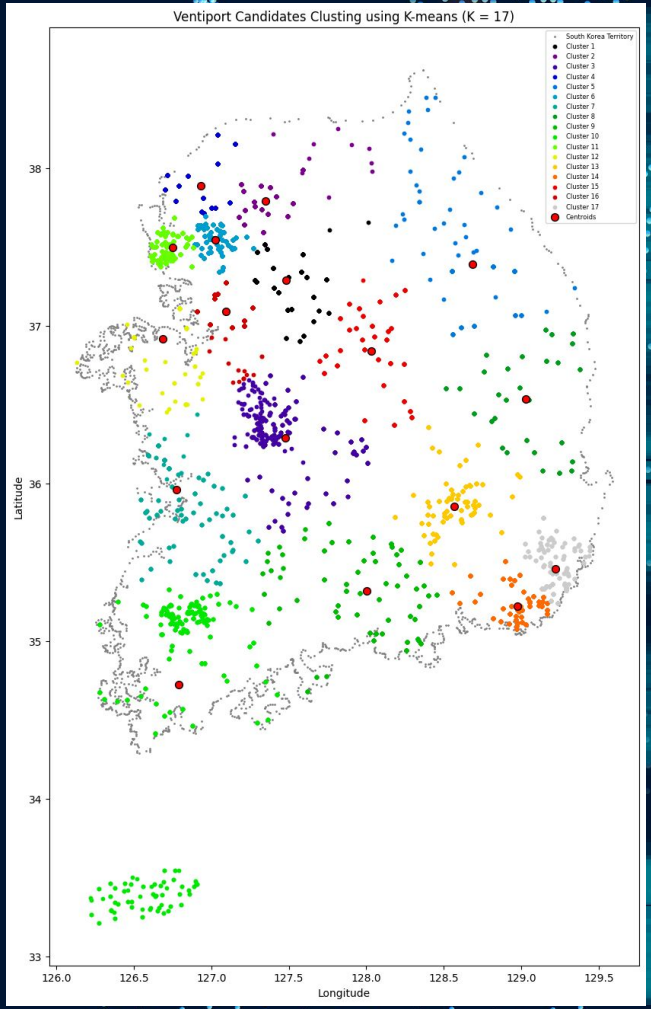
```
Clusters: [5 5 5 ... 9 9 9]
Final Centroids: [[ 37.29153823 127.48162439]
 [ 37.79345132 127.35235455]
 [ 36.28976283 127.4778338 ]
 [ 37.88662478 126.9316216 ]
 [ 37.38961221 128.68338789]
 [ 37.54402536 127.02819114]
 [ 35.96118538 126.77602665]
 [ 36.53520621 129.02940193]
 [ 35.31724455 128.00078798]
 [ 34.72322345 126.78986549]
 [ 37.50007047 126.74897414]
 [ 36.9171189  126.68589938]
 [ 35.85681049 128.56714398]
 [ 35.22508823 128.97511764]
 [ 36.8422553  128.03070805]
 [ 37.09153485 127.09407103]
 [ 35.4578539  129.2189276 ]]
Iterations: 13
```

최종 후보지의
위도와 경도

Cluster 별 데이터 포인트의 수

```
OrderedDict([(0, 512),
             (1, 542),
             (2, 447),
             (3, 417),
             (4, 191),
             (5, 1457),
             (6, 414),
             (7, 184),
             (8, 435),
             (9, 534),
             (10, 719),
             (11, 145),
             (12, 495),
             (13, 477),
             (14, 182),
             (15, 312),
             (16, 311)])
```

평균적으로 **15~30 iteration** 내에서 수렴함



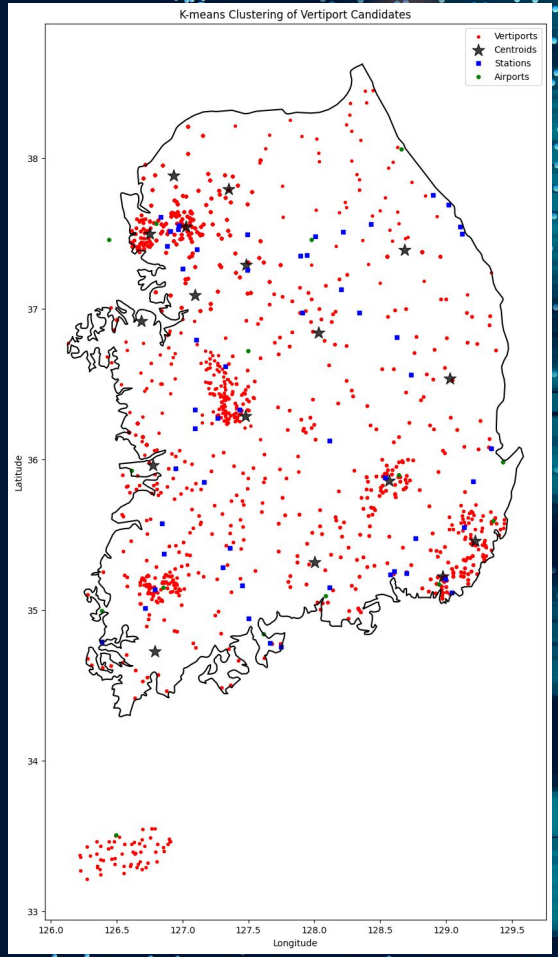
K-means when K = 17

Vertiport와 다른 교통 수단과의 연계와 **vertiport** 접근성를 고려

최종적으로 결정된 **vertiport**위치에서 주변 역, 공항 근처로 수정

	A	B	C	D	E
1	Longitude	Latitude			
2	126.801	37.5655	김포공항		
3	126.442	37.4587	인천공항		
4	128.947	35.1729	김해공항		
5	128.639	35.8991	대구공항		
6	128.663	38.0588	양양공항		
7	127.496	36.722	청주공항		
8	126.388	34.99357	무안공항		
9	126.493	33.5071	제주공항		

	A	B	C	D	E
1	Longitude	Latitude			
2	126.834	37.61185	행신역		
3	126.972	37.5559	서울역		
4	126.885	37.4165	광명역		
5	127.001	37.2668	수원역		
6	127.105	36.7946	천안아산역		
7	127.327	36.6205	오송역		
8	127.434	36.3323	대전역		
9	128.115	36.1234	김천역		
10	128.54	35.8815	서대구역		
11	128.771	35.4745	밀양역		
12	128.997	35.2055	구포역		



Clustering model evaluation

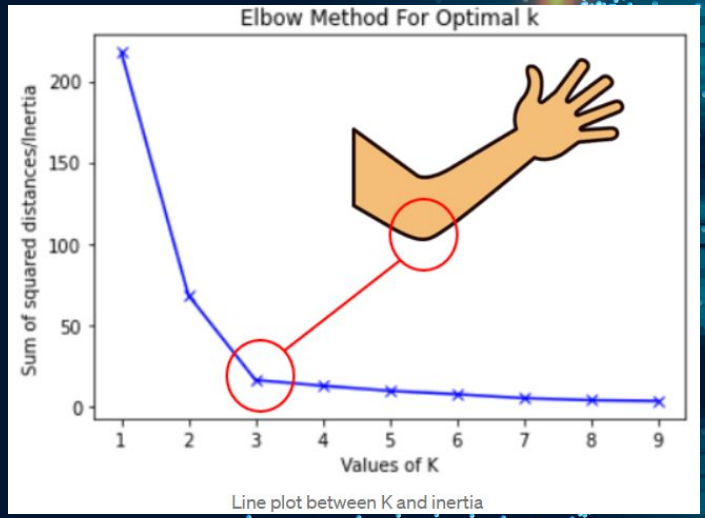
1. Elbow method

- 클러스터링에서 최적의 클러스터 수(K)를 결정하는 데 사용됨

- 각 클러스터 내의 데이터 포인트들과 해당 클러스터 중심점 간의 거리 제곱합(SSE)를 계산하여, SSE가 급격히 감소하다가 완만해지는 지점을 찾음.

- 장점 : 간단하고 직관적, 계산이 빠름

- 단점 : 주관성, 데이터 포인트들이 밀집해 있으면 엘보우가 나타나지 않을 수도 있음



출처 :
https://media.licdn.com/dms/image/D4D12AQF-yYtbzPvNFg/article-cover_image-shrink_600_2000/0/1682277078758?e=2147483647&v=beta&t=VhzheKDjy7bEcsYyrjql3NQAUcTaMBCTzhZWSVVSeNg

Clustering model evaluation

2. Silhouette score

- 클러스터링의 품질을 평가하는 지표로, 각 데이터 포인트가 자신의 클러스터 내에 잘 맞는지, 또는 다른 클러스터와 더 유사한지를 나타냄
-실루엣 스코어는 -1에서 1 사이의 값을 가지며, 값이 클수록 클러스터들이 잘 퍼져 있으며, 클러스터 내의 데이터 포인트들이 유사함을 의미

$$s(i) = \frac{(b(i) - a(i))}{(\max(a(i), b(i))}$$

a(i) : 데이터 포인트 **i**와 같은 클러스터에 속한 데이터 포인트들과의 거리의 평균
b(i) : **i**와 다른 클러스터 중 가장 가까운 클러스터까지의 평균 거리

- 장점 : 클러스터 내부에 데이터 포인트 간의 응집력과 클러스터 간의 분리된 정도를 동시에 고려하므로 클러스터의 품질을 종합적으로 결정 가능, **K**값에 따라 비교하며 최적의 **K**를 찾을 수 있음.

- 단점 : 계산 비용이 비쌈

Clustering model evaluation

```python
def silhouette(points, clusters):
    sillhouette_list = list()  # 각 데이터 포인트의 실루엣 스코어를 저장할 리스트를 초기화
    k = len(np.unique(clusters))  # 클러스터의 수

    # 각 데이터 포인트에 대해 반복
    for i in range(len(points)):
        # 같은 클러스터에 속하는 데이터 포인트를 구하는데, 본인은 제외
        same_cluster = points[(clusters == clusters[i]) & (np.arange(len(points)) != i)]

        # 다른 클러스터에 속하는 데이터 포인트들을 저장할 리스트를 초기화
        other_cluster = list()
        for j in range(k):
            if j != clusters[i]:
                # 각 클러스터마다 현재의 데이터 포인트와 다른 클러스터에 속하는 데이터 포인트들을 구한다.
                other_cluster.append(points[clusters == j])

        # 같은 클러스터에 데이터 포인트가 하나도 없으면 실루엣 스코어를 0으로 설정
        if len(same_cluster) == 0:
            sillhouette_list.append(0)

        else:
            # ai 계산: 같은 클러스터의 데이터 포인트들과의 평균 거리를 계산
            same_dist = list()
            for same in same_cluster:
                same_dist.append(euclidean(points[i], same))
            ai = np.mean(same_dist)

            # bi 계산: 다른 클러스터의 데이터 포인트들과의 평균 거리를 계산하고, 그 중 가장 작은 값을 선택
            bi_list = list()
            for cluster in other_cluster:
                other_dist = list()
                for other in cluster:
                    other_dist.append(euclidean(points[i], other))
                bi_list.append(np.mean(other_dist))
            bi = min(bi_list) # 여러 클러스터내의 평균 거리중 가장 작은 값을 선택

            # 공식에 따라 실루엣 점수를 계산
            sillhouette_list.append((bi - ai) / max(bi, ai))

    # 모든 데이터 포인트의 실루엣 스코어의 평균을 구함
    sillhouette_score = np.mean(sillhouette_list)

    return sillhouette_list, sillhouette_score
```

Clustering model evaluation

Problem 1

```
clusters, final_centroids, iterations = kmeans(points, initial=initial_centroids,
sillhouette_list, average = silhouette(points, clusters)
print(average)
print(sillhouette_list)
```

```
0.5708086620582917
[0.552786404500042, 0.30827652333975814, 0.7424971893898715, 0.4084491934010904, 0.72
```

```
from sklearn.metrics import silhouette_score, silhouette_samples
print(silhouette_score(points, clusters))
silhouette_samples(points, clusters)
```

```
0.5708086620582917

array([0.5527864 , 0.30827652, 0.74249719, 0.40844919, 0.72936507,
       0.6409217 , 0.51031856, 0.67385466])
```

Problem 3

```
sillhouette_list, average = silhouette(coordinates, clusters)
print(average)
print(sillhouette_list)
```

```
0.441120961875548
[0.5756027704808483, 0.3637934425231505, 0.13021803210421312, 0.417904336
```

```
print(silhouette_score(coordinates, clusters))
silhouette_samples(coordinates, clusters)
```

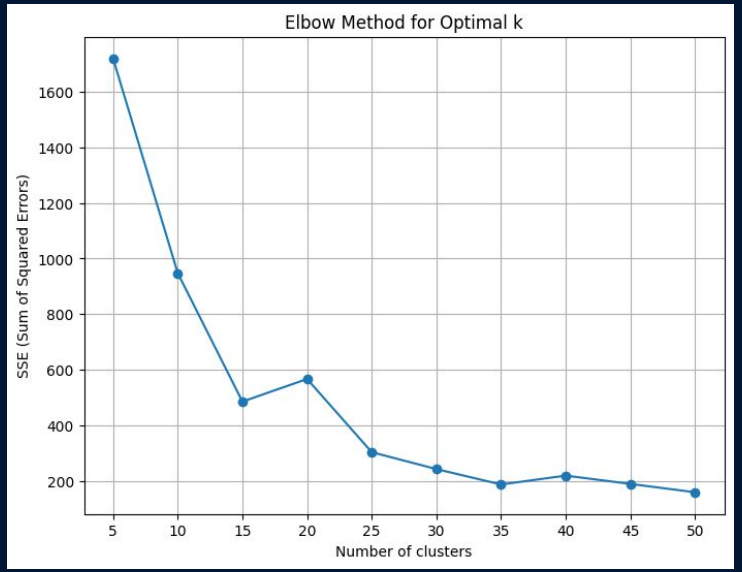
```
0.4411209373872431

array([0.57560277, 0.36379344, 0.13021803, ..., 0.42329766, 0.39051706,
       0.3809257 ])
```

Sklearn에서 제공하는 silhouette_score과 같은 값을 산출함을 확인

Optimizing K

엘보우 방법으로 **5, 50**까지를 단위 **5**로 비교




```
For n_clusters = 5 SSE = 1719.4721922687486
For n_clusters = 10 SSE = 947.7110943855582
For n_clusters = 15 SSE = 485.4642223963629
For n_clusters = 20 SSE = 566.8309599226565
For n_clusters = 25 SSE = 303.66902305863016
For n_clusters = 30 SSE = 242.8112265316132
For n_clusters = 35 SSE = 187.4592692250831
For n_clusters = 40 SSE = 219.35865243007822
For n_clusters = 45 SSE = 189.45762879464223
For n_clusters = 50 SSE = 159.44531472026964
```

⇒ **25**일 때 급격히 감소하며 이후 완만해짐

⇒ **25~30** 사이에 최적 **K**가 있을 것

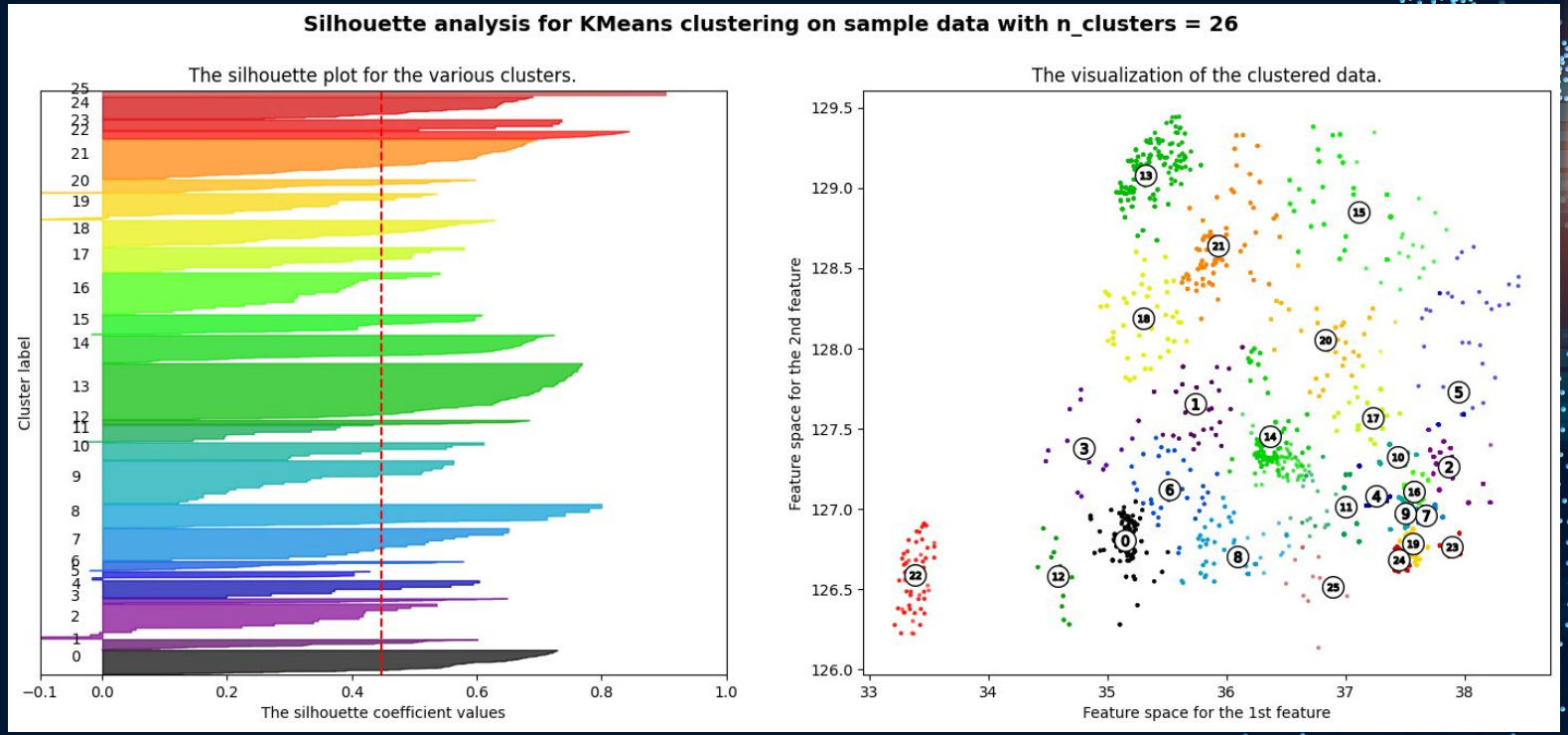
Optimizing K

실루엣 방법으로 **25, 30**까지를 비교

```
For n_clusters = 25 The average silhouette_score is : 0.43202351262147803
For n_clusters = 26 The average silhouette_score is : 0.44734162100266698
For n_clusters = 27 The average silhouette_score is : 0.43234606048666335
For n_clusters = 28 The average silhouette_score is : 0.44702699917881O1
For n_clusters = 29 The average silhouette_score is : 0.43357405118170833
For n_clusters = 30 The average silhouette_score is : 0.42958050349814164
```

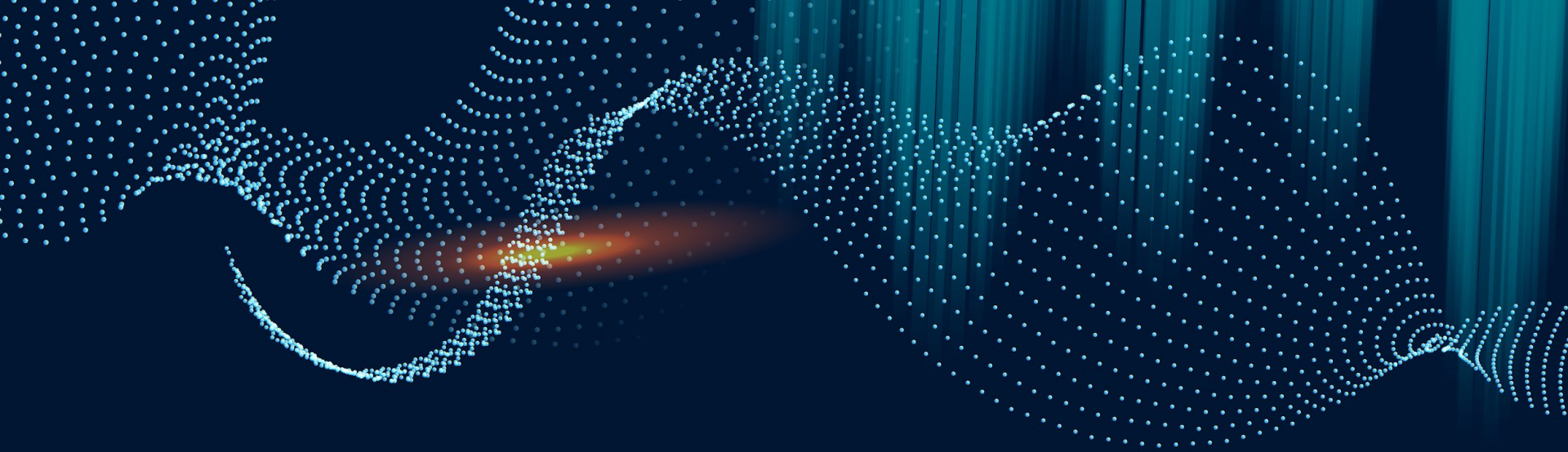
K가 **26**일 때의 평균 실루엣 스코어가 가장 높음

Optimizing K



⇒ 대부분의 클러스터가 **Silhouette score** 라인을 넘음

⇒ 각 지역별로 고르게 분포하되, 수도권에 중심점이 모여있음을 확인할 수 있음



05 Discussion

Discussion

Number of cluster(k)

k=1,2,3...

Number of iteration & Stop criteria

Initialization

Data shape

Number of cluster

K-Means 알고리즘의 경우, **hard clustering method** 이기 때문에 적절한 **K**값을 찾는 것이 **algorithm** 의 성능을 결정

적절한 **K**값을 찾기 위해 **Elbow method, Silhouette score** 를 이용

Iteration & Criteria

Python code 에서 **K-Means algorithm** 을 통해 최적의 해를 얻기 위해 반복 계산 실행

Stop criteria 와 **number of iteration** 을 동시에 정하는 것이 효과적

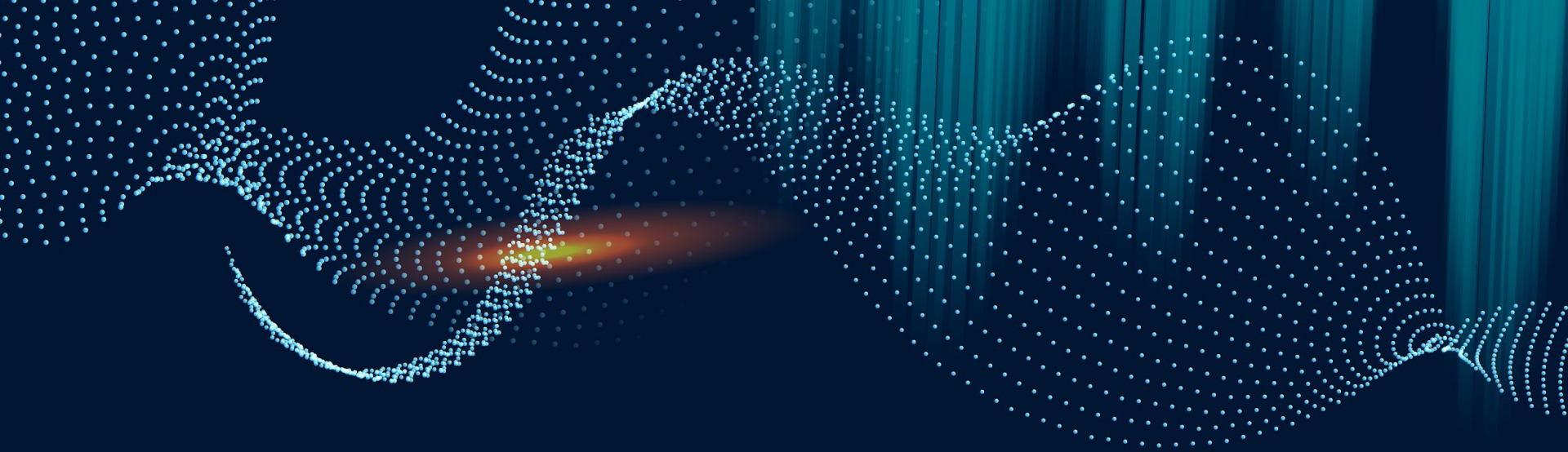
Initialization

K-Means algorithm 은 **Initialization** 에 따라 결과가 달라지기 때문에 적절한 **Initialization** 을 설정해야함

-> 초기 **centroid point** 를 무작위로 생성, 일정 구역 내에서 생성..

Data shape

K-Means algorithm 은 **spectral circle data** 에 사용할 수 없으므로 데이터를 적절하게 생성



06 Reference

- iAI KAIST . (2019/09/07) . *E week 1-05 K-means 클러스터링 파이썬 실습 1* . [Video]. https://www.youtube.com/watch?v=tk9LGtOYVL8
- OpenAI . (2024/07/29) . K-means 모델링 . *ChatGPT*
- sungboklove@gmail.com . (2024/06/26) . *지도로 위도 경도 좌표 찾기* . 지도로 위도 경도 좌표 찾기 (findlatlng.org)
- Scikitlearn, Selecting the number of clusters with silhouette analysis on KMeans clustering, https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
- OpenAI. (2024). ChatGPT [Large language model]. https://chat.openai.com
- Medium, The Art and Science of K-means Clustering: A Practical Guide, sachinsoni600517, https://medium.com/@sachinsoni600517/the-art-and-science-of-k-means-clustering-a-practical-guide-e71b11638867

THANKS!