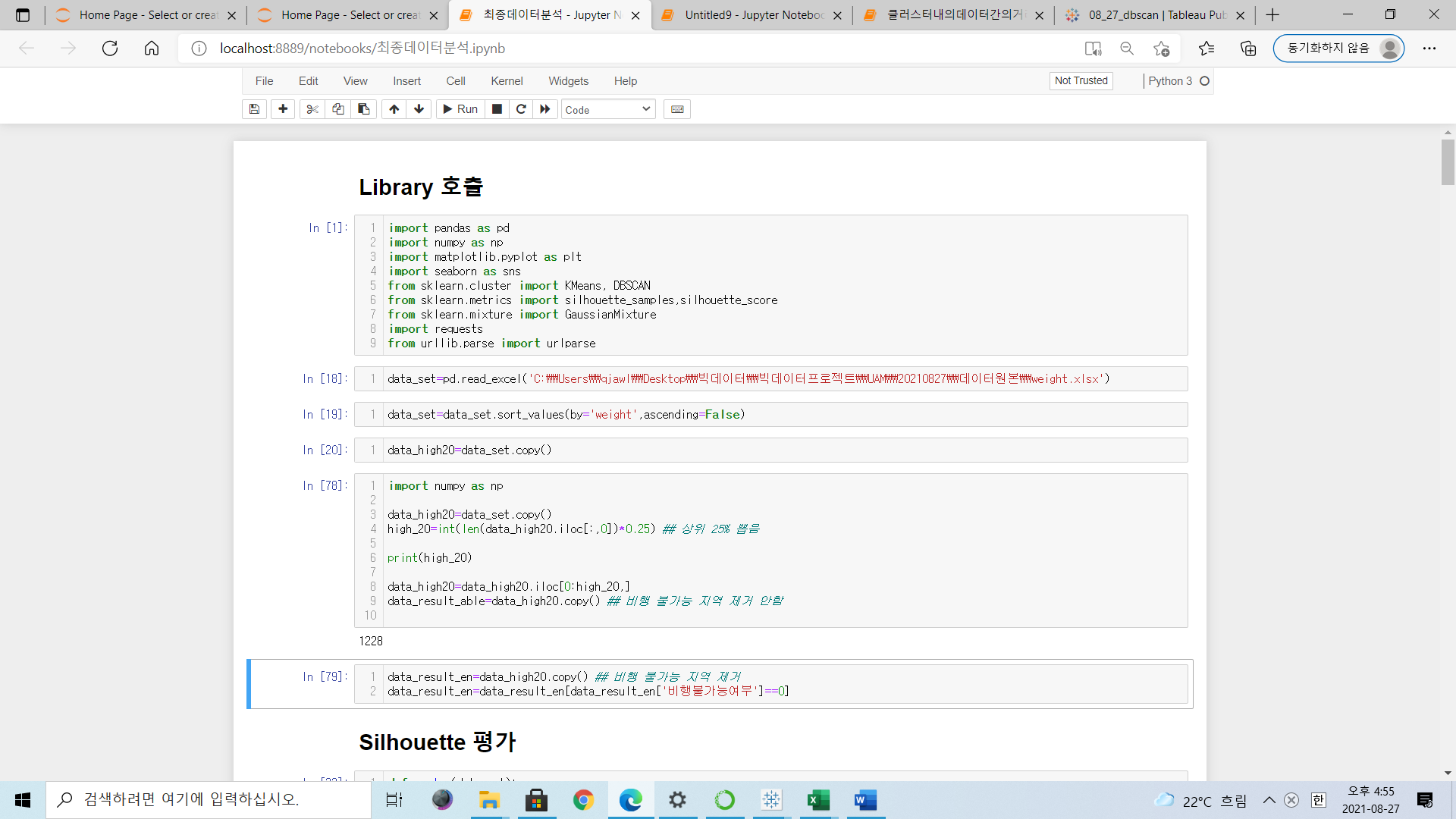
데이터 분석

* **K-Means, GMM, DBSCAN 이용**

테이블이(가) 표시된 사진

자동 생성된 설명엔트로피 지수와 데이터의 행렬 곱셈 결과인 weight의 총합 상위 30%를 가지고 분석을 진행한다.

데이터를 weight 기준으로 내림차순 정렬 한 뒤, 상위 25%의 데이터를 산출하여 비행 불가능 지역 포함 여부를 기준으로 2개의 데이터 로 구분한다.



Data\_result\_able->비행 불가능 지역 제거 안함

Data-result\_en-> 비행 불가능 지역 제거함

**사용 라이브러리**

Pandas : 데이터를 데이터 프레임 단위로 처리 하기 위하여 사용

**텍스트이(가) 표시된 사진

자동 생성된 설명**Numpy : 행렬 연산을 하기 위하여 사용

Matplotlib : 시각화용

Seaborn : 시각화용

Sklearn\_metrics – silhouette\_samples, silhouette\_score : 성능 평가용 실루엣 계산

Sklearn\_cluster – Kmeans,Dbscan,GaussianMixture : 머신러닝 모델링용

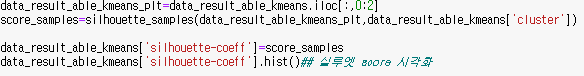
**평가**

eval\_s 함수를 만들어 평가



평가 지표를 확인할 데이터 셋을 인자로 받는다.

이 때 데이터 셋 포맷은 Column 기준 0번 index 와 1번 index에는 위도 경도 데이터가 2번 index에는 cluster label이 들어가야한다.

받은 데이터를 가지고 위경도 데이터와 cluster label 데이터로 나누어서 silhouette\_score 라이브러리를 이용하여 척도를 측정한 후 return 한다.

Score\_samples 라이브러리를 이용하여 원본 데이터에 column을 추가하여 클러스터별 silhouette 계수를 시각화 한다.

1. **KMeans**

KMeans 분석은 사이킷런 라이브러리의 KMeans를 이용하였다.

KMeans 알고리즘은 주어진 데이터를 K개의 군집으로 클러스터링 하는 알고리즘으로 K개의 Centriod를 설정한 후 Centriod의 위치를 각 데이터간의 거리합이 최소화가 되는 지점으로 옮겨가며 더 이상 Centriod가 움직이지 않을 때 까지 진행하는 알고리즘이다.





데이터 중에서 클러스터링에 사용될 위도 , 경도 데이터만 추출한다.

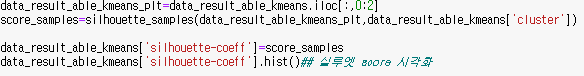
K\_means 는 centroid 를 데이터간 거리가 짧은 지점으로 이동시키면서 군집화 한다. Centriod가 움직이지 않으면 동작을 멈춘다.

Parameter:

n\_clusters = 군집화 하고자 하는 군집 개수

Init = centriod의 초기 위치 , k-means++ -> 임의로 센트로이드 위치를 잡음

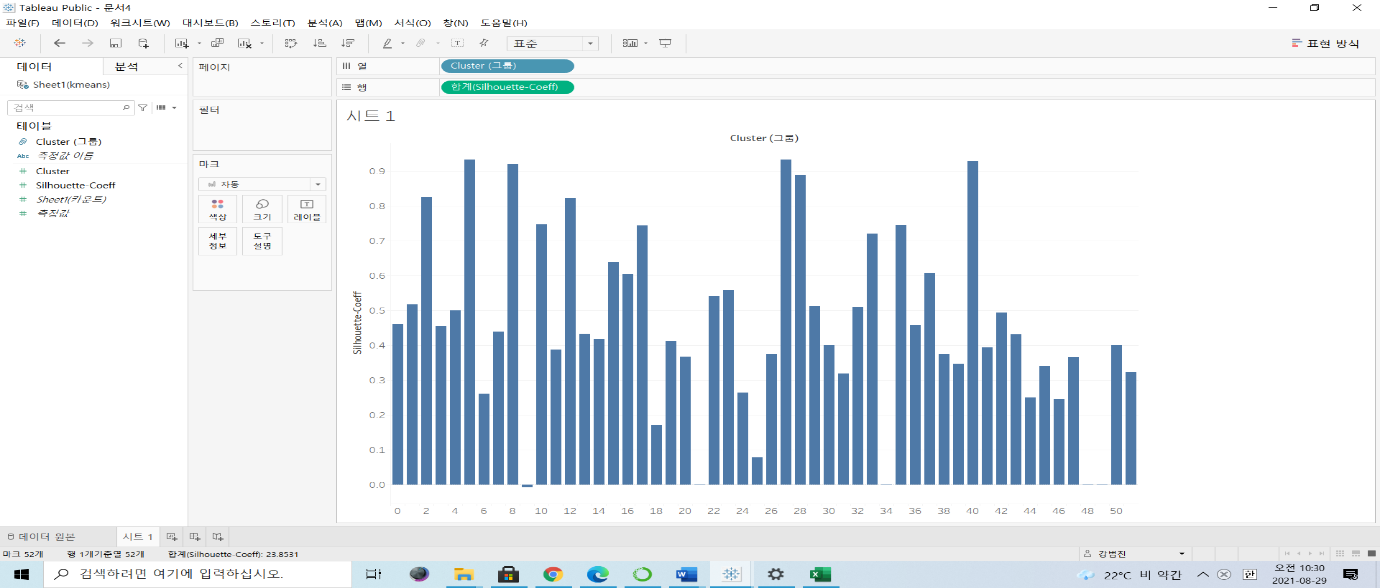
Max\_iter= centriod가 움직일 최대 횟수

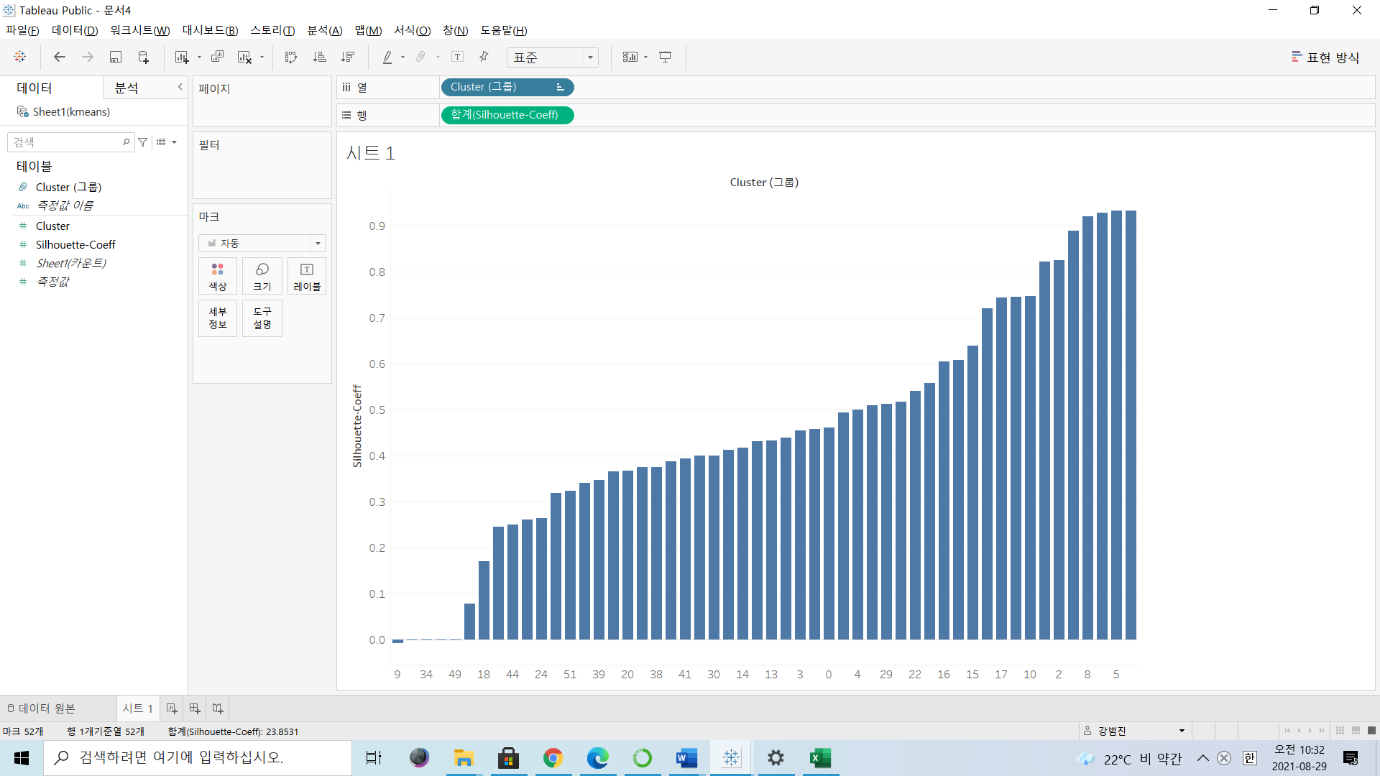
* 비행불가 지역, 비행가능 지역 나누어서 모델링

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

모델링 후 군집 기준 그룹화 후 군집 별 silhouette 분산,표준편차,평균을 계산하고 Tableau를 이용하여 군집 별 실루엣 계수 평균값을 시각화 하기 위하여 백업파일로 만든다.

Tableau 작업 화면에서 열에 cluster 그룹을 행에 실루엣계수 값을 넣으면 다음과 같이 나타나고, cluster를 오름차순 정렬하면 다음과 같이 시각화 가능하다.

**K-Means 분석 시각화의 결과를 보면 클러스터별 실루엣 계수가 비교적 고르게 분포되어 있고 음수 값을 갖는 클러스터가 한 개 존재한다. 실루엣 계수의 평균이 음수의 값을 갖는다는 의미는 해당 군집내에 존재하는 데이터는 다른 군집에 존재하는게 타당할 수 있다는 의미이다.**

**분산은 0.063935, 표준편차는 0.252853정도의 값이 나온다. 이를 다른 군집화 알고리즘과 비교하였을 때 적은 값으로 평균은 높고 분산과 표준편차는 적은 K-means 알고리즘을 채택하였다.**

**2.DBSCAN**

DBSCAN 모델은 사이킷런 DBSCAN 라이브러리를 이용하였다.

DBSCAN은 지정된 eps값 내의 minsamples개의 데이터가 존재하면 군집으로 묶고, 그 군집 중 core point 기준으로 eps 값 내의 minsamples 개 이상의 데이터가 존재하면 동일 군집으로 묶고 그 외의 값들은 outlier 처리하는 밀도기반 군집화 알고리즘이다.

텍스트이(가) 표시된 사진

자동 생성된 설명먼저 DBSCAN 라이브러리의 파라미터를 보면

Eps: 반경

Min\_samples: 최소 데이터셋 개수

Metric : 거리계산 방식

Dbscan 모델은 군집의 개수를 정하고 학습을 시키는 것이 아니다.

본 프로젝트의 목표는 52개의 군집을 찾는 것 이기 때문에 2중 For 문으로 두개의 파라미터 값을 조정하며 52개의 군집이 나오도록 파라미터를 튜닝하고, 튜닝이 끝난 파라미터 중, 성능이 가장 좋은 파라미터를 채택하였다.

먼저텍스트이(가) 표시된 사진

자동 생성된 설명 list\_array 배열에는 eps의 후보 값들이 들어있다.

최대값 1 부터 0까지 0.001씩 줄여가며 파라미터 튜닝을 진행하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Min\_samples 의 값은 1에서부터 7로 1씩 증가시키며 튜닝을 진행하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명먼저 input data 는 위에서 말한 kmeans의 입력 데이터의 형태와 동일하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* **사용자 정의함수 db\_scan**

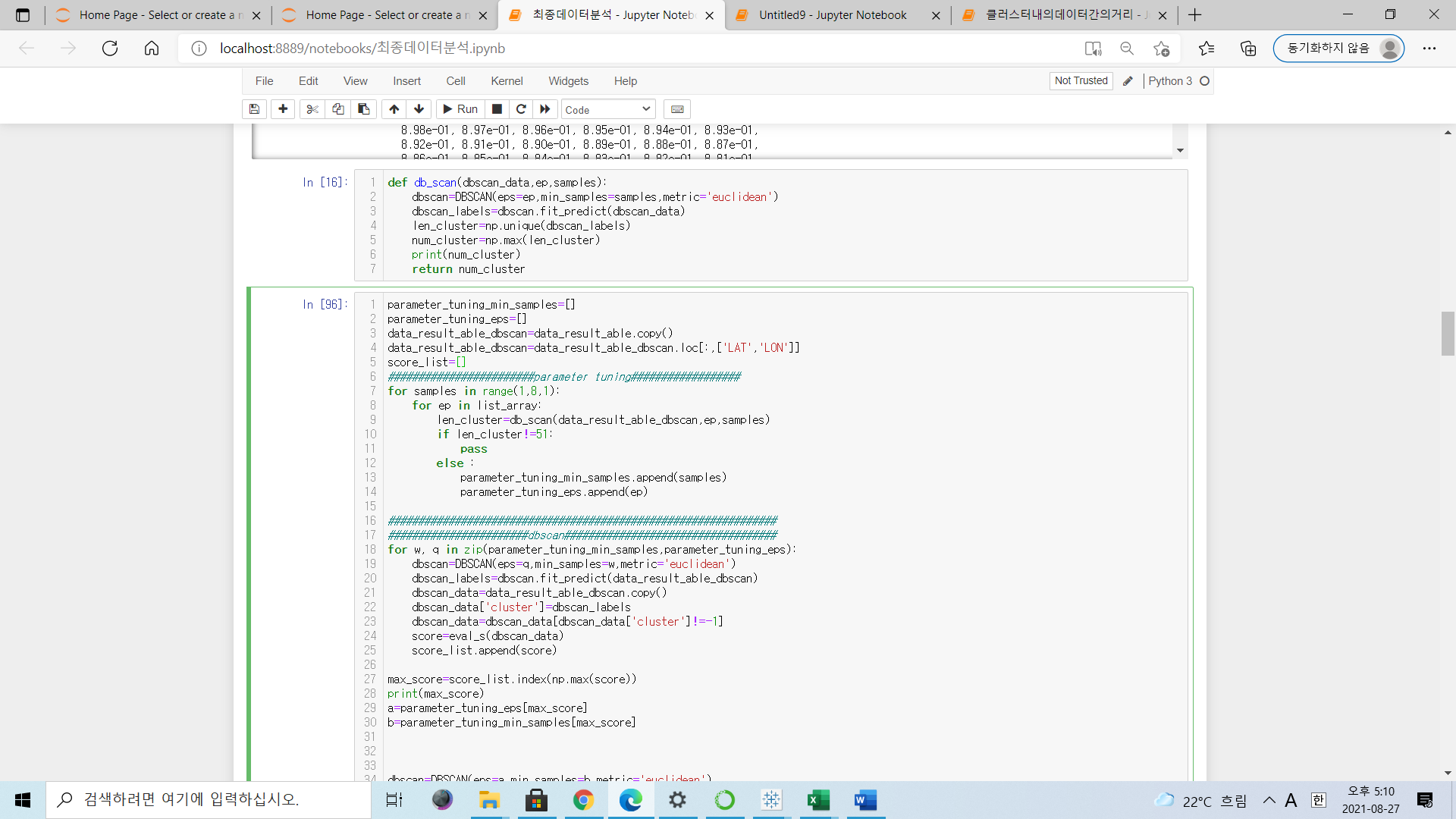
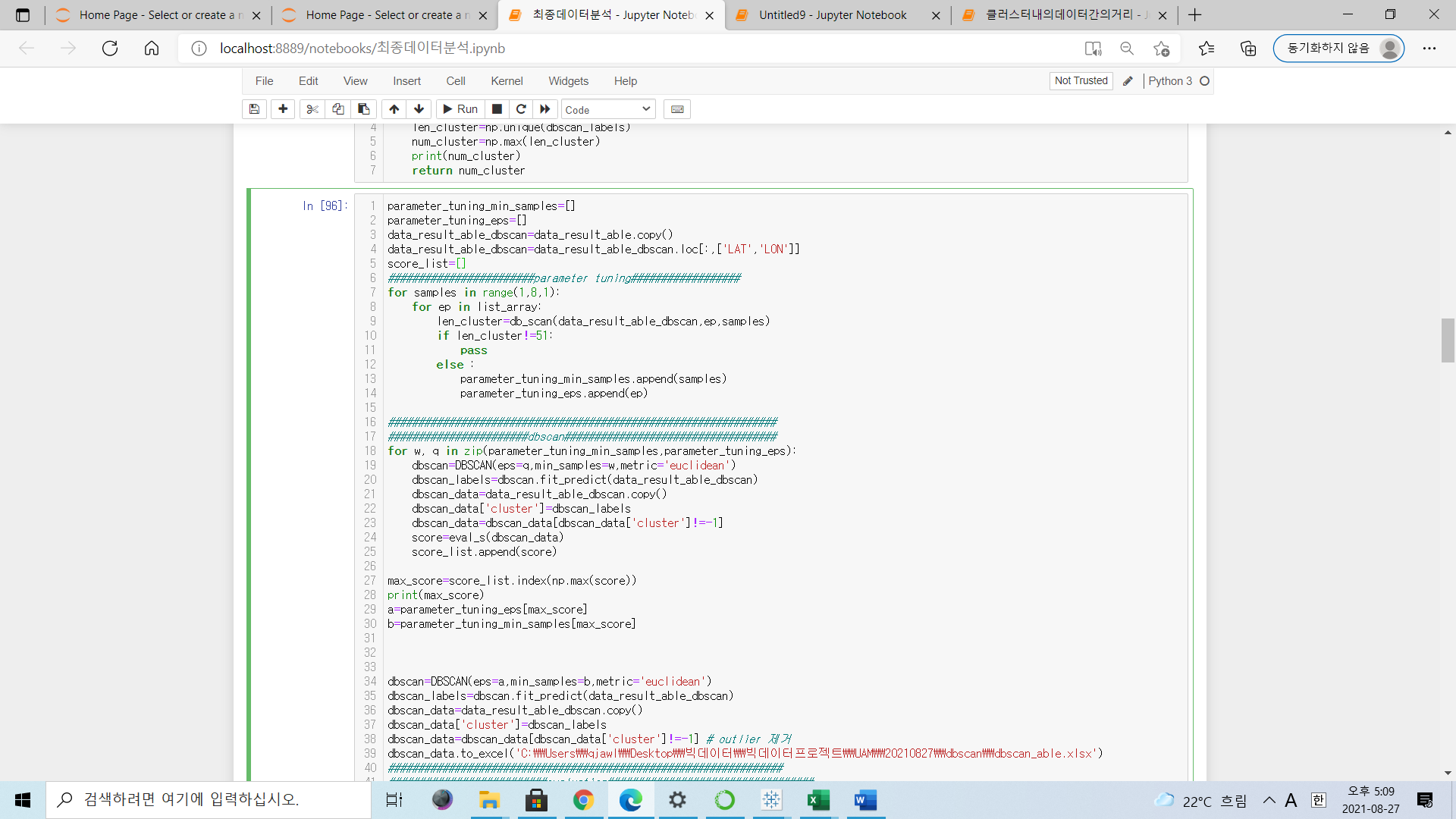
Db\_scan 함수는 DBSCAN에 쓰일 데이터와 eps와 min\_samples 를 인자로 받는다

인자로 받은 값들을 가지고 DBSCAN을 돌리고 결과 label의 unique 값들을 len\_cluster 배열에 저장한다. Cluster label은 0부터 오름차순으로 매겨지기 대문에 len\_cluster 배열의 최대값이 최종 cluster의 개수가 되고, 이 값을 return 하여 cluster의 개수가 52개 되는 지점을 찾는다.

* **파라미터 튜닝**

코드에서 보면 2중 For문을 돌려가며 파라미터를 튜닝한다.

빈 배열 parameter\_tuning\_min\_samples와 parameter\_tuning\_eps를 선언 하고



첫번째 for 문에서 sample의 개수를 조정하고, 2번째 for문에서 위에서 만들어 놓은 1에서 0까지 -0.001씩 줄여가며 만들어 놓은 값들을 대입하여 위에서 언급한 사용자 정의함수 db\_scan에 인자로 넣어가며 클러스터의 개수를 확인한다

Cluster label의 indexing은 0부터 시작하기 대문에 len\_Cluster가 51 일 때 cluster의 개수가 52개가 되었다고 할 수 있다.

따라서 52개가 아닌 경우에는 if문 pass, 52인 경우에는 처음에 설정한 빈 배열 parameter\_tuning\_min\_samples, parameter\_tuning\_eps 배열에 해당 파라미터들을 저장한다. 동시에 저장하기에 같은 index에 쌍으로 저장되어서 문제될 점은 존재하지 않는다. 최종적으로 parameter\_tuning\_min\_samples와 parameter\_tuning\_eps의 배열에는 cluster의 개수가 52개가 되는 파라미터들이 존재하게 된다.

* **텍스트이(가) 표시된 사진

  자동 생성된 설명성능 최적화**

텍스트이(가) 표시된 사진

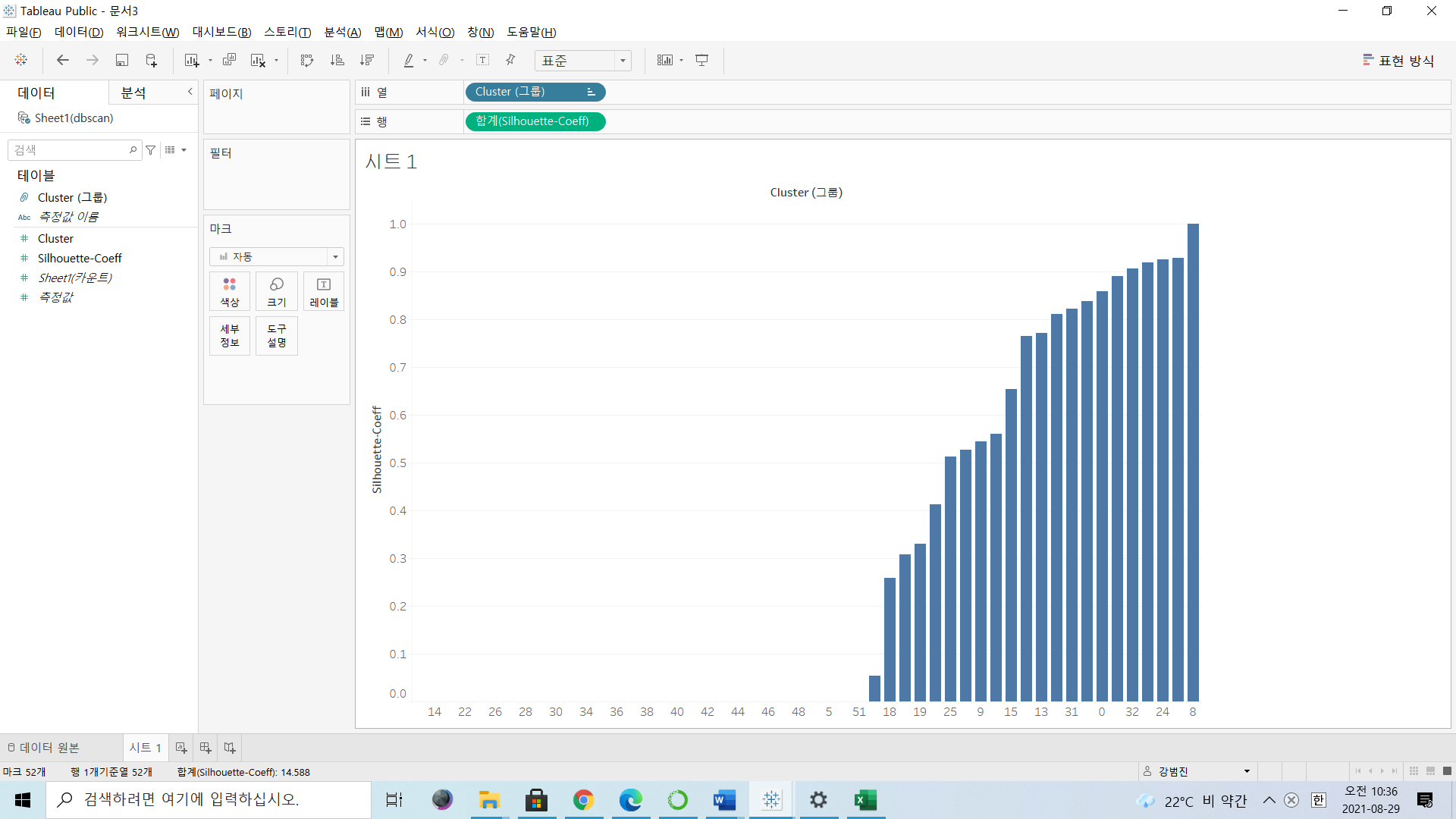
자동 생성된 설명Parameter\_tuning\_min과 parameter\_tuing\_eps의 값들을 zip함수를 이용하여 w,q변수에 대입하며 각각 파라미터의 경우의 수에 대한 score를 측정하여 score 배열에 저장한다. 이 때 score 배열과 parameter\_tuning 데이터에 채워지는 데이터의 순서를 보면 DBSCAN을 돌린 WQ에 대한 score값이 score 배열에 저장 되기 때문에 최고 점수일때의 index를 가지고 parameter\_tuning 데이터의 값에 접근하여 최고 점수일때의 eps와 minsamplse를 변수 a,b에 할당하고 이를 가지고 다시 DBSCAN을 돌린다.

Cluster label이 -1인 데이터들은 outlier로 처리된 값들 이기 때문에 제외하고 dbscan\_data에 저장한 후 백업한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

silhouette 라이브러리를 이용하여 dbscan\_data의 silhouette 계수를 구하고, dbscan\_data에 silhouette\_coeff 라는 column을 추가하여 silhouette 계수 데이터를 합치고, 클러스터 기준으로 그룹화 하고 Tableau 에서 시각화 하기 위해서 백업파일로 저장한다.



텍스트이(가) 표시된 사진

자동 생성된 설명

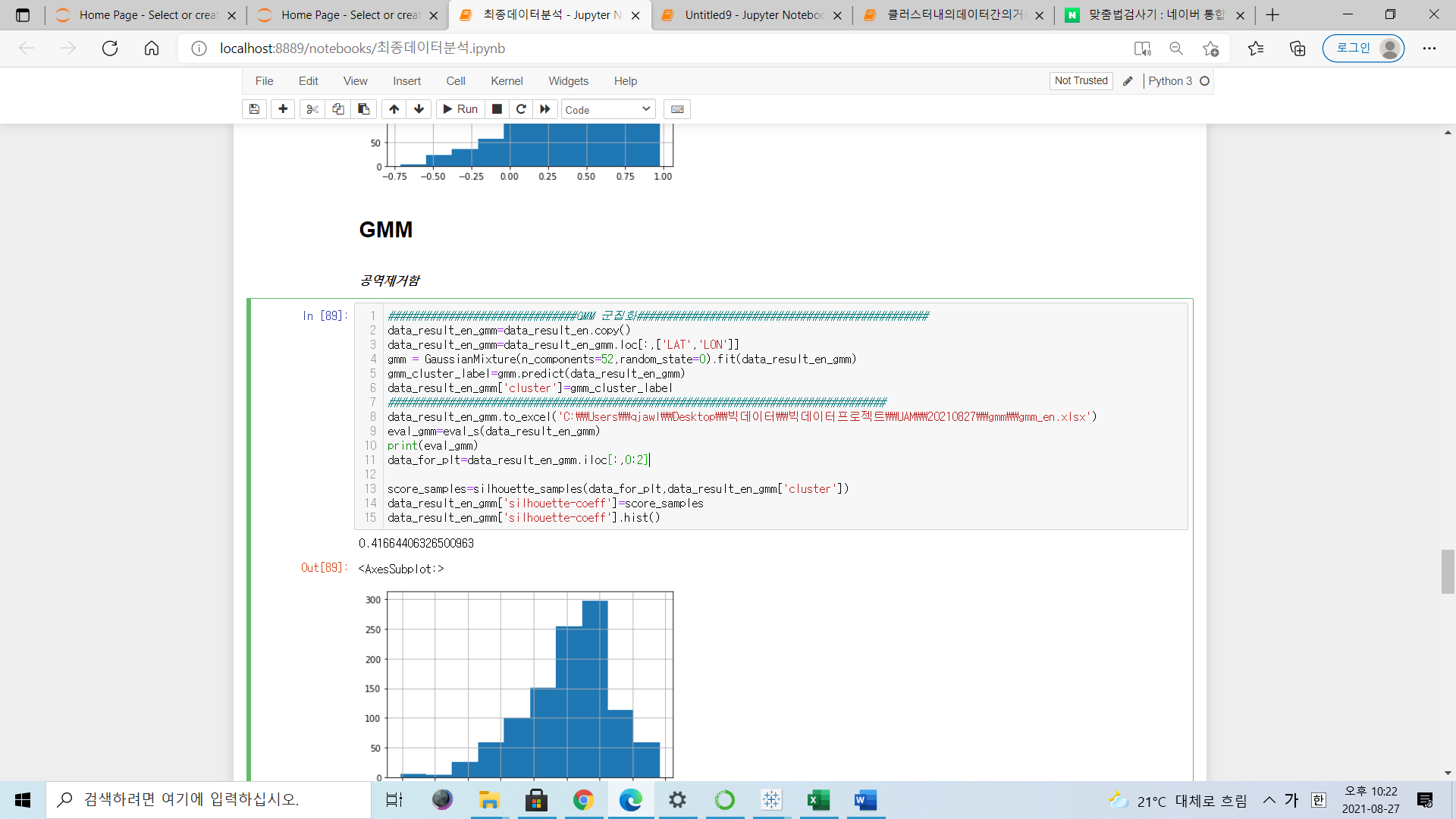
실루엣 계수의 분산, 표준편차를 구한다.

DBSCAN 알고리즘의 결과를 보면 분산은 0.368, 표준편차는 0.281 의 값이 나온다. 이는 KMEANS와 비교해보면 다소 큰 값을 갖는다. 또 시각화 결과를 보면 실루엣 계수의 평균이 0인 클러스터가 많이 존재한다. 이는 다른 군집과의 거리가 매우 가깝다는 의미이고, K\_means와 비교하였을 때 비교적 성능이 떨어진다는 의미이다. DBSCAN은 outlier를 제거하며 군집화를 진행하는 알고리즘으로 성능이 좋을 것으로 예상되었지만, 비교적 성능이 안 좋게 나온 이유 다음과 같다. 우리는 정부의 목적에 맞춰서 52개의 군집을 대상으로 군집화를 진행하였다. DBSCAN은 군집의 개수를 정하고 돌리는 알고리즘이 아니다. 따라서 우리는 목적에 맞게 클러스터 개수가 52개 나올 때 까지 파라미터 튜닝을 진행하였고, 그 과정에서 어쩔 수 없이 성능이 떨어진다. 만약 군집의 개수를 상수가 아닌 범위로 잡고 진행하게 된다면 DBSCAN의 성능은 더 좋아질 것으로 예상된다.

**3.GMM**

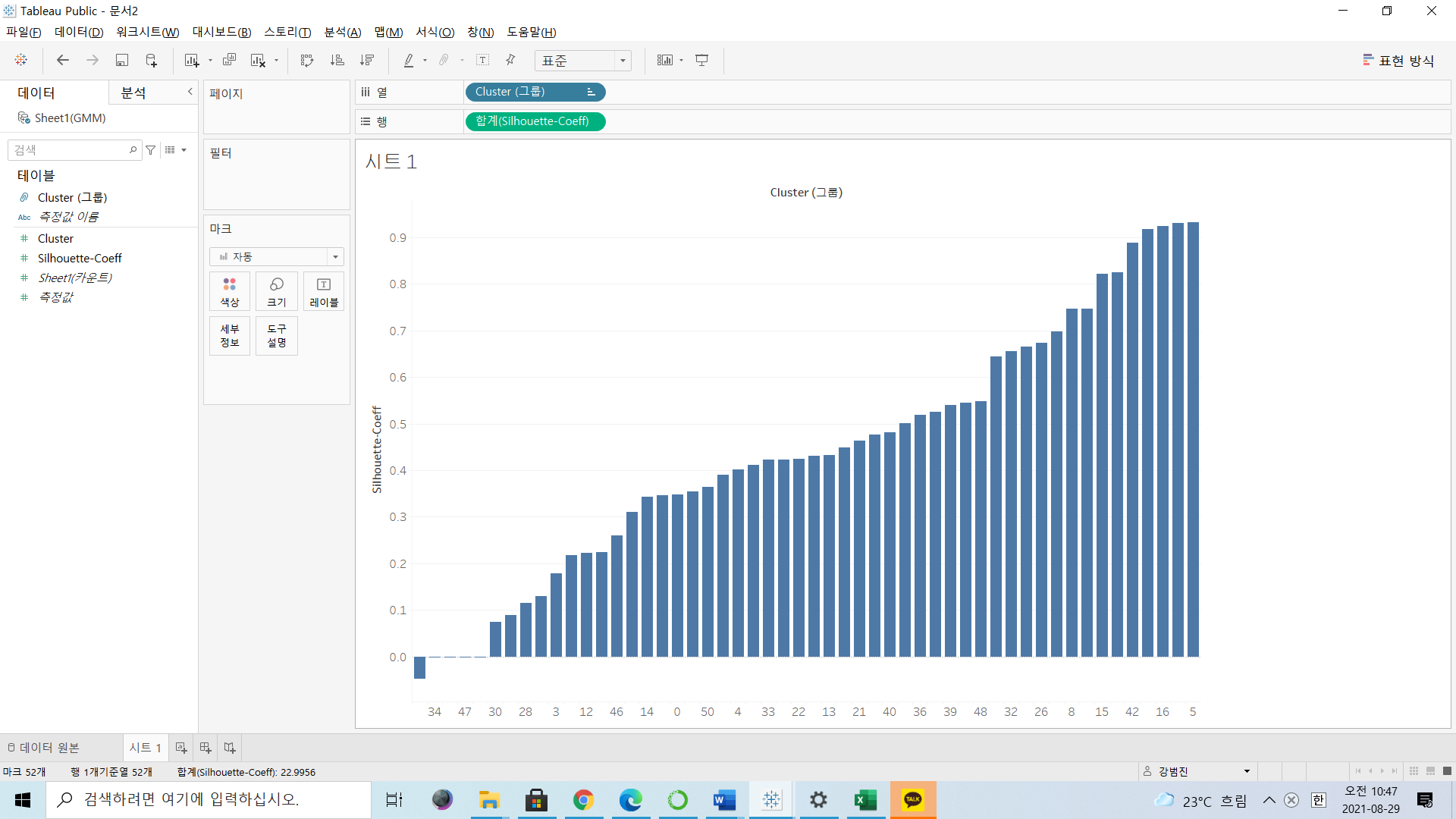
GMM 분석은 사이킷런 GaussianMixture 라이브러리를 이용하였다.

GMM 알고리즘은 Gaussian Mixture Model로 주어진 데이터를 N개의 군집으로 묶을 때 N개의 가우시안 분포를 혼합하여 각각 데이터가 어떤 가우시안 분포에 해당되는지 찾아 군집화하는 알고리즘이다.

Input 데이터의 형태는 위에서 언급한 kmeans의 input 데이터 형태와 동일하다.

텍스트이(가) 표시된 사진

자동 생성된 설명Cluster 52개를 기준으로 군집화를 진행한 후 cluster column을 추가하여 결과 데이터를 저장한다.

클러스터별로 그룹화 하여 실루엣 계수의 평균, 분산, 표준편차를 구했고 이를 Tableau로 시각화 하기위해 백업파일로 저장했다.

평균텍스트이(가) 표시된 사진

자동 생성된 설명,표준편차,분산을 보면 DBSCAN보단 성능이 좋지만, K-MEANS보단 다소 떨어지는 모습이다.