PHW2 – Clustering

이상원, 이민서, 이강욱, 심우석

Github - <https://github.com/dntjr41/MachineLearning_termProject>

Source code:

# Import Class Libraries  
import eyeball as eyeball  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import purity as purity  
import seaborn as sns  
import sklearn  
import plotly.express as px  
from sklearn import metrics  
from sklearn.model\_selection import GridSearchCV  
from scipy.stats import stats  
from pyclustering.cluster.clarans import clarans  
from sklearn.mixture import GaussianMixture  
from sklearn.cluster import KMeans, DBSCAN, OPTICS  
from sklearn.decomposition import PCA  
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder  
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler  
from sklearn.metrics import silhouette\_score  
sns.set()  
  
##############################################################################  
# AutoML (X, y = None, scale\_col, encode\_col, scalers = None, encoders = None,  
# feature\_param = None, models = None, model\_param = None,  
# scores = None, score\_param = None)  
#  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Must Read \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# Description = When parameters are put in, the plot and scores are output  
# The method of producing results in AutoML function consists of three main steps  
#  
# Step 1 = Feature Selection (PCA(), RandomSelect(), CustomSelect()) \* model (KMeans(), GMM(), clarans(), DBSCAN(), OPTICS()) = 15,  
# Find a combination with the best silhouette score in each combination  
#  
# Step 2 = If there is a target value, Among the three Feature Selection (PCA(), RandomSelect(), CustomSelect()),  
# check which model has the highest purity and return three results  
#  
# Step 3 = Using the final three combinations (without a target value),  
# we compare with the combinations (with a target value)  
# - The results are checked through the clustering plot and the silhouette score -  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
#  
# Input = X: Data Feature  
# Y: Data Target (If you have a target value, enter it)  
# Scale\_col: columns to scaled  
# Encode\_col: columns to encode  
# Scalers: list of scalers  
# None: [StandardScaler(), RobustScaler(), MinMaxScaler()]  
# If you want to scale other ways, then put the scaler in list.  
# Encoders: list of encoders  
# None: [OrdinalEncoder(), LabelEncoder()]  
# If you want to encode other ways, then put the encoder in list.  
#  
# Feature: list of features  
# None: [PCA(), RandomSelect(), CustomSelect()]  
# If you want to set other ways, then put specific feature in list  
#  
# Feature\_param: feature selection method's parameter  
# PCA()'s None: [n\_components: None (int)]  
# RandomSelect()'s None: [number\_of\_features: None (int)]  
# CustomSelect()'s None: [combination\_of\_features: None (list)]  
#  
# Models: list of models  
# None: [KMeans(), GMM(), clarans(), DBSCAN(), OPTICS()]  
# If you want to fit other ways, then put (Clustering)model in list.  
#  
# Model\_param: list of model's hyperparameter  
# KMeans()’s None: [n\_clusters: None (int), init: None(k-means++, random),  
# n\_init: None (int), Random\_state: None (int), max\_iter: None (int)]  
# GMM()’s None: [n\_components: None (int), covariance\_type: None (spherical, tied, diag),  
# n\_init: None (int), Random\_state: None (int),  
# min\_covar: None (float), tol: None (float)]  
# clarans()’s None: [number\_clusters: None (int), numlocal\_minima: None (int),  
# max\_neighbor: None (int)]  
# DBSCAN()’s None: [eps: None (float), min\_samples: None (int), metric: None (str or callable),  
# p: None (float), Algorithm: None (auto, ball\_tree, kd\_tree, brute)]  
# OPTICS()’s None: [eps: None (float), min\_samples: None (int), p: None (int),  
# cluster\_method: None (xi, dbscan), algorithm: None (auto, ball\_tree, kd\_tree, brute)]  
# If you want to set other ways, then put the hyperparameter in list  
#  
# Scores: list of score methods  
# None: [silhouette\_score(), KelbowVisualizer(), purity(), eyeball()]  
# If you want to see other ways, then put the scoring model in list.  
#  
# Score\_param: list of score method's hyperparameter  
# Silhouette\_score()’s None: [metric: None (str, callable), random\_state: None (int)]  
# Purity()’s None: None  
# eyeball()'s None: None  
#  
# Output = some scores, plots  
  
# Description = Calculate the silhouette score and return the value  
# Input = kind of model, Dataset  
# Output = Silhouette score  
def cv\_silhouette\_scorer(estimator, X):  
 print("그리드 서치중 : ", estimator)  
  
 # If GMM(EM) handle separately  
 if type(estimator) is sklearn.mixture.\_gaussian\_mixture.GaussianMixture:  
 # print("it's GaussianMixture()")  
 labels = estimator.fit\_predict(X)  
 return silhouette\_score(X, labels, metric='euclidean')  
  
 # Calculate and return Silhouette score  
 else:  
 estimator.fit(X)  
 cluster\_labels = estimator.labels\_  
 num\_labels = len(set(cluster\_labels))  
 num\_samples = len(X.index)  
 if num\_labels == 1 or num\_labels == num\_samples:  
 return -1  
 else:  
 return silhouette\_score(X, cluster\_labels)  
  
  
# purity를 구해주는 함수  
def purity\_score(y\_true, y\_pred):  
 # compute contingency matrix  
 contingency\_matrix = metrics.cluster.contingency\_matrix(y\_true, y\_pred)  
 # return purity  
 return np.sum(np.amax(contingency\_matrix, axis=0)) / np.sum(contingency\_matrix)  
  
  
# Description = randomly determines features  
# input = Dataset, number of feature  
# Output = (Random)Dataset  
class RandomSelect:  
 # number of feature (Default:4)  
 n = 4  
  
 # Accept N  
 def set\_params(self, n\_components):  
 self.n = n\_components  
  
 # Pick N and combination  
 def fit\_transform(self, data):  
 choice = np.random.choice(data.columns, self.n)  
 result = pd.DataFrame(data[choice[0]])  
  
 for i in range(1, len(choice)):  
 result = pd.concat([result, data[choice[i]]], axis=1)  
  
 # Return Dataset  
 return result  
  
  
# Description = select specific features  
# input = Dataset, selected features  
# Output = (Selected) Dataset  
class CustomSelect:  
 # Combination of selected features  
 feature = None  
  
 # Accept selected features  
 def set\_params(self, n\_components):  
 self.feature = n\_components  
  
 # Combine the selected features  
 def fit\_transform(self, data):  
 result = pd.DataFrame(data[self.feature[0]])  
  
 for i in range(1, len(self.feature)):  
 result = pd.concat([result, data[self.feature[i]]], axis=1)  
  
 # Return Dataset  
 return result  
  
  
# Description = It converts data according to each feature selection method  
# If PCA is reset column name  
# If RandomSelect is randomly determines features  
# If CustomSelect is select specific features  
# Input = Dataset, selected feature, number of feature  
# Output = (Processed) Dataset  
def makefeatureSubset(X, selection, n\_feature):  
 selection.set\_params(n\_components=n\_feature)  
 x\_result = selection.fit\_transform(X)  
 x\_result = pd.DataFrame(x\_result)  
  
 # Reset column name  
 if type(selection) == type(PCA()):  
 if n\_feature == 3:  
 x\_result.columns = ["Principle-1", "Principle-2", "Principle-3"]  
  
 elif n\_feature == 4:  
 x\_result.columns = ["Principle-1", "Principle-2", "Principle-3", "Principle-4"]  
  
 elif n\_feature == 5:  
 x\_result.columns = ["Principle-1", "Principle-2", "Principle-3", "Principle-4", "Principle-5"]  
  
 return x\_result  
  
  
def AutoML(X, y=None, scale\_col=None, encode\_col=None, scalers=None, encoders=None,  
 features=None, feature\_param=None, models=None, model\_param=None,  
 scores=None, score\_param=None):  
  
 # Set Encoder  
 global df\_score\_and\_encode, df\_first\_scaled, df\_new\_score\_and\_encode  
  
 if encoders is None:  
 encode = [OrdinalEncoder(), LabelEncoder()]  
 else:  
 encode = encoders  
  
 # Set Scaler  
 if scalers is None:  
 scale = [StandardScaler(), MinMaxScaler(), RobustScaler()]  
 else:  
 scale = scalers  
  
 # Set Feature  
 # If it's None value, select all features, set PCA, selected features, random select  
 if features is None:  
 feature = [PCA(), RandomSelect(), CustomSelect()]  
 customSelectParameter = [["longitude", "latitude"], ["total\_rooms", "total\_bedrooms"],  
 ["longitude", "latitude", "total\_rooms", "total\_bedrooms"],  
 ["total\_rooms", "total\_bedrooms", "population", "households", "median\_income"]]  
 feature\_parameter = [[3, 4, 5], [3, 4, 5], customSelectParameter]  
  
 else:  
 feature = features  
 feature\_parameter = feature\_param  
  
 # Set Model  
 if models is None:  
 model = [KMeans(), GaussianMixture(), '''clarans()''', DBSCAN(), OPTICS()]  
 else:  
 model = models  
  
 # Set Model parameter  
 if model\_param is None:  
 # KMeas Clustering  
 model\_parameter = [{'n\_clusters': [2, 3, 4, 5], 'init': ["k-means++", "random"],  
 'n\_init': [1, 10, 20], 'random\_state': ["None", 0, 1],  
 'max\_iter': [100, 200]},  
  
 # GMM(EM) Clustering  
 {'n\_components': [2, 3, 4, 5], 'max\_iter': [100, 200],  
 'covariance\_type': ["spherical", "tied", "diag"],  
 'n\_init': [1, 10, 20], 'random\_state': ["None", 0, 1],  
 'min\_covar': [1e-5, 1e-3], 'tol': [1e-5, 1e-3]},  
  
 # Clarans Clustering  
 # {'number\_clusters': [2, 3, 4, 5], 'numlocal\_minima': [2, 3, 4],  
 # 'max\_neighbor': [2, 3, 4, 5]},  
  
 # DBSCAN Clustering  
 {'eps': [0.3, 0.4, 0.5], 'min\_samples': [2, 3, 4, 5],  
 'metric': ["euclidean", "manhattan"], 'p': [1, 2, 3],  
 'algorithm': ["auto", "ball\_tree", "kd\_tree", "brute"]},  
  
 # Optics Clustering  
 {'eps': [0.3, 0.4, 0.5],  
 'min\_samples': [2, 3, 4, 5], 'p': [1, 2, 3],  
 'cluster\_method': ["xi", "dbscan"],  
 'algorithm': ["auto", "ball\_tree", "kd\_tree", "brute"]}]  
 else:  
 model\_parameter = model\_param  
  
 # Set Score  
 if scores is None:  
 score = ['''silhouette\_score(), purity(), eyeball()''']  
 else:  
 score = scores  
  
 # Set Score parameter  
 if score\_param is None:  
 score\_parameter = [None]  
 else:  
 score\_parameter = score\_param  
  
 # First Step's values (feature selection 3 \* model 5) using silhouette score  
 # [ PCA , Model1][ PCA , Model2]...[ PCA , Model5]  
 # [Random, Model1][Random, Model2]...[Random, Model5]  
 # [Custom, Model1][Custom, Model2]...[Custom, Model5]  
 firstScore = [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]  
 firstScoreScaler = [[None, None, None, None, None], [None, None, None, None, None], [None, None, None, None, None]]  
 firstScoreEncoder = [[None, None, None, None, None], [None, None, None, None, None], [None, None, None, None, None]]  
 firstScoreFeature = [[None, None, None, None, None], [None, None, None, None, None], [None, None, None, None, None]]  
 firstScoreModel = [[None, None, None, None, None], [None, None, None, None, None], [None, None, None, None, None]]  
 firstScoreParameter = [[None, None, None, None, None], [None, None, None, None, None],  
 [None, None, None, None, None]]  
  
 cv = [(slice(None), slice(None))]  
  
 # Second Step's values (feature selection 3) using purity and target value (If you have that)  
 secondScore = [0, 0, 0]  
 secondScoreScaler = [None, None, None]  
 secondScoreEncoder = [None, None, None]  
 secondScoreFeature = [None, None, None]  
 secondScoreModel = [None, None, None]  
 secondScoreParameter = [None, None, None]  
  
 ####################################################################  
 # Iterate  
 for i in scale:  
 for j in encode:  
 # Scaling  
 df\_scaled = pd.DataFrame(i.fit\_transform(X[scale\_col]))  
 df\_scaled.columns = scale\_col  
  
 # Encoding  
 if encode\_col is not None:  
 if type(j) == type(OrdinalEncoder()):  
 df\_encoded = j.fit\_transform(X[encode\_col])  
 df\_encoded = pd.DataFrame(df\_encoded)  
 df\_encoded.columns = encode\_col  
 df\_prepro = pd.concat([df\_scaled, df\_encoded], axis=1)  
  
 else:  
 print("NO")  
 dum = pd.DataFrame(pd.get\_dummies(X[encode\_col]))  
 df\_prepro = pd.concat([df\_scaled, dum], axis=1)  
  
 else:  
 df\_prepro = df\_scaled  
  
 # feature selection (find feature subset : PCA, random select, custom select)  
 featureIndex = 0  
 for z, z\_param in zip(feature, feature\_parameter):  
 modelIndex = 0  
 for m in model:  
 for z\_param\_index in z\_param:  
  
 # Step1 - Compare Silhouette score  
 # Feature Selection(PCA(), RandomSelection(), CustomSelect()) \*  
 # model(KMeans(), GMM(), clarans(), DBSCAN(), OPTICS()) = 15  
 # Find a Combination with the best silhouette score in each combination  
  
 # If feature selection is PCA -> Iterate n\_components 3,4,5  
 # If feature selection is RandomSelect -> Iterate n\_components 3,4,5  
 # If feature selection is CustomSelect -> Iterate subset  
 # A feature subset that fits the selection and parameter came out  
 df\_featureSubset = makefeatureSubset(df\_prepro, z, z\_param\_index)  
  
 gridSearch = GridSearchCV(estimator=m, param\_grid=model\_parameter[modelIndex],  
 scoring=cv\_silhouette\_scorer, cv=cv)  
 # fit grid search  
 result = gridSearch.fit(df\_featureSubset)  
 best\_model = result.best\_estimator\_  
 best\_params = result.best\_params\_  
 pred = best\_model.fit\_predict(df\_featureSubset)  
 score = silhouette\_score(df\_featureSubset, pred)  
 print("현재 selection : ", z, "\n현재 모델 : ", m)  
 print(best\_model)  
 print(best\_params)  
 print("score: ", score)  
  
 if firstScore[featureIndex][modelIndex] == 0 or firstScore[featureIndex][modelIndex] < score:  
 print(featureIndex)  
 print(modelIndex)  
 print(i)  
 firstScore[featureIndex][modelIndex] = score  
 firstScoreScaler[featureIndex][modelIndex] = i  
 firstScoreEncoder[featureIndex][modelIndex] = j  
 firstScoreFeature[featureIndex][modelIndex] = df\_featureSubset.columns  
 firstScoreModel[featureIndex][modelIndex] = best\_model  
 firstScoreParameter[featureIndex][modelIndex] = best\_params  
  
 modelIndex += 1  
 featureIndex += 1  
  
 # Print step1's result  
 for i in range(0, 3):  
 for j in range(0, 5):  
 print("최종 결과", i, " ", j)  
 print(firstScoreScaler[i][j])  
 print(firstScoreEncoder[i][j])  
 print(firstScoreFeature[i][j])  
 print(firstScoreModel[i][j])  
 print(firstScoreParameter[i][j])  
 print(firstScore)  
 print(print())  
  
 # Step 2 = If there is a target value, Among the three Feature Selection (PCA(), RandomSelect(), CustomSelect()),  
 # check which model has the highest purity and return three results  
 for a in range(1, 3):  
 for b in range(0, 3):  
  
 # scale\_col scaling => X[scale\_col]  
 if firstScoreScaler[a][b] is not None: # If exist scaler  
 df\_first\_scaled = pd.DataFrame(firstScoreScaler[a][b].fit\_transform(X[scale\_col]))  
 df\_first\_scaled.columns = scale\_col  
 # else: # If not exist scaler  
 # df\_first\_scaled = X  
  
 # encode\_col encoding => X[encode\_col]  
 if firstScoreEncoder[a][b] is not None: # If exist encoder  
 df\_first\_encoded = pd.DataFrame(firstScoreEncoder[a][b].fit\_transform(X[encode\_col]))  
 df\_first\_encoded.columns = encode\_col  
 # scaled + encoded  
 df\_score\_and\_encode = pd.concat([df\_first\_scaled, df\_first\_encoded], axis=1)  
 # else: # If not exist encoder  
 # df\_score\_and\_encode = df\_first\_scaled  
  
 # print("\*\*\*\* Combination of Score and Encode \*\*\*\*\n")  
 # print(df\_score\_and\_encode)  
  
 # Extract only features from feature\_selection from scaling and encoded data frames.  
 if firstScoreFeature[a][b] is not None:  
 first\_fture = []  
 for k in firstScoreFeature[a][b]:  
 first\_fture.append(k)  
  
 df\_new\_score\_and\_encode = df\_score\_and\_encode[first\_fture]  
 print("\*\*\*\* Apply feature selection \*\*\*\*\n")  
 print(df\_new\_score\_and\_encode)  
  
 df\_values = df\_new\_score\_and\_encode.values  
  
 # model fitting  
 if firstScoreModel[a][b] is not None:  
 pred\_val = firstScoreModel[a][b].fit\_predict(df\_new\_score\_and\_encode)  
 # print("predict value shape: {}".format(pred\_val.shape))  
 # print("\*\*\*\* Predicted Value \*\*\*\*\n")  
 # print(pred\_val)  
  
 min\_y = np.min(y)  
 max\_y = np.max(y)  
 gap = max\_y - min\_y  
 gap /= len(np.unique(pred\_val))  
  
 labels = []  
 for i in range(len(np.unique(pred\_val))):  
 labels.append(i)  
  
 temp\_df = pd.cut(y["median\_house\_value"], bins=len(np.unique(pred\_val)), labels=labels, include\_lowest=True)  
 temp\_df = temp\_df.to\_numpy()  
  
 print("\*\*\*\* Purity Score \*\*\*\*")  
 purityScore=purity\_score(temp\_df, pred\_val)  
 print(purityScore)  
  
 if purityScore > secondScore[a]:  
 secondScore[a]=purityScore  
 secondScoreScaler[a] = firstScoreScaler[a][b]  
 secondScoreEncoder[a] = firstScoreEncoder[a][b]  
 secondScoreFeature[a]= firstScoreFeature[a][b]  
 secondScoreModel[a] = firstScoreModel[a][b]  
 secondScoreParameter[a] = firstScoreParameter[a][b]  
  
 # Print step2's result  
 print(secondScore)  
 print(secondScoreScaler)  
 print(secondScoreEncoder)  
 print(secondScoreFeature)  
 print(secondScoreModel)  
 print(secondScoreParameter)  
  
 # Step 3 = Using the final three combinations (without a target value),  
 # we compare with the combinations (with a target value)  
 # - The results are checked through the clustering plot and the silhouette score -  
  
 for i in range(1,3):  
 # scale\_col scaling => X[scale\_col]  
 if secondScoreScaler[i] is not None: # If exist scaler  
 df\_second\_scaled = pd.DataFrame(secondScoreScaler[i].fit\_transform(X[scale\_col]))  
 y\_second\_scaled=pd.DataFrame(secondScoreScaler[i].fit\_transform(y))  
 y\_second\_scaled.columns=y.columns  
 df\_second\_scaled.columns = scale\_col  
  
 # encode\_col encoding => X[encode\_col]  
 if secondScoreEncoder[i] is not None: # If exist encoder  
 df\_second\_encoded = pd.DataFrame(secondScoreEncoder[i].fit\_transform(X[encode\_col]))  
 df\_second\_encoded.columns = encode\_col  
 # scaled + encoded  
 df\_score\_and\_encode = pd.concat([df\_second\_scaled, df\_second\_encoded], axis=1)  
  
 # Extract only features from feature\_selection from scaling and encoded data frames.  
 if secondScoreFeature[i] is not None:  
 second\_fture = []  
 for k in secondScoreFeature[i]:  
 second\_fture.append(k)  
 df\_new\_score\_and\_encode = df\_score\_and\_encode[second\_fture]  
 df\_new\_score\_and\_encode\_y=pd.concat([df\_new\_score\_and\_encode,y\_second\_scaled],axis=1)  
  
 # Using the plot and silhouette score  
 # Compare the clustering results with medianHouseValue(target)  
 # feature values in the original dataset  
  
 # Without target value  
 model = secondScoreModel[i]  
 print(secondScoreScaler[i])  
 print(secondScoreEncoder[i])  
 print(secondScoreFeature[i])  
 print(secondScoreModel[i])  
 print(secondScoreParameter[i])  
 cluster\_no=secondScoreModel[i].fit(df\_new\_score\_and\_encode)  
 label=cluster\_no.labels\_  
 print(label)  
  
 fig = px.scatter(df\_new\_score\_and\_encode, color=label)  
 fig.show()  
  
 pred\_no = cluster\_no.fit\_predict(df\_new\_score\_and\_encode)  
 score = silhouette\_score(df\_new\_score\_and\_encode, pred\_no)  
 print("Silhouette score = ", score)  
  
 # With target value  
 cluster\_yes = secondScoreModel[i].fit(df\_new\_score\_and\_encode\_y)  
 label=cluster\_yes.labels\_  
 print(label)  
  
 fig = px.scatter(df\_new\_score\_and\_encode\_y, color=label)  
 fig.show()  
  
 pred\_yes = cluster\_yes.fit\_predict(df\_new\_score\_and\_encode\_y)  
 score = silhouette\_score(df\_new\_score\_and\_encode\_y, pred\_yes)  
 print("Silhouette score = ", score)  
  
##############################################################################  
# Dataset = California housing price  
# Feature = longitude, latitude, housing\_median\_age, total\_rooms, total\_bedrooms  
# population, households, median\_income, ocean\_proximity  
# Target = median\_house\_value  
  
# Number of dataset = 20640  
# Numerical value = longitude, latitude, housing\_median\_age, total\_rooms, total\_bedrooms,  
# population, households, median\_income, median\_house\_value  
# Categorical value = ocean\_proximity  
  
df = pd.read\_csv("housing.csv")  
  
feature\_label = ['longitude', 'latitude', 'housing\_median\_age', 'total\_rooms', 'total\_bedrooms',  
 'population', 'households', 'median\_income', 'ocean\_proximity']  
target\_label = ['median\_house\_value']  
  
# Print housing data's information  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* housing \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.head())  
#  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.describe())  
#  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.info())  
  
# Check null value  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check null \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.isna().sum())  
  
# Fill null value  
df['total\_bedrooms'].fillna(df['total\_bedrooms'].mean(), inplace=True)  
# print("\n\*\*\*\*\*\*\*\* Check null (Cleaned data) \*\*\*\*\*\*")  
# print(df.isna().sum())  
  
# Remove Outliers with z-score  
# Description = Use the z-score to handle outlier over mean +- 3SD  
# Input = dataframe's column  
# Output = index  
df\_cate = df['ocean\_proximity']  
  
def find\_outliers(col):  
 z = np.abs(stats.zscore(col))  
 idx\_outliers = np.where(z > 3, True, False)  
 return pd.Series(idx\_outliers, index=col.index)  
  
  
for n in range(len(feature\_label)):  
 idx = None  
 idx = find\_outliers(df.iloc[:, n])  
 df = df.loc[idx == False]  
  
# print("\n\*\*\*\*\*\*\*\* Removed Outlier \*\*\*\*\*\*")  
# print(df.info())  
  
# Set X, y data  
y\_data = df.loc[:, target\_label]  
X\_data = df.drop(target\_label, axis=1)  
  
scale\_col = ["longitude", "latitude", "housing\_median\_age", "total\_rooms", "total\_bedrooms", "population", "households",  
 "median\_income"]  
end\_col = ["ocean\_proximity"]  
  
AutoML(X\_data,y\_data, scale\_col=scale\_col, encode\_col=end\_col, models=None, model\_param=None)

Result:

Step1 - Feature Selection (PCA(), RandomSelection (), CustomSelect()) \*

model (KMeans(), GMM(), clarans(), DBSCAN(), OPTICS()) = 15

Find a Combination with the best silhouette score in each combination

##############################################################################

# Step1 Result - Compare Silhouette score #############################################

firstScore = [[0.36422508875626597, 0.3583197737781512, -0.18922996991959803, 0, 0],

[0.5226162531098414, 0.5021989520509015, 0.15173322443353157, 0, 0],

[0.6442539672913747, 0.5735714041022789, 0.7001271118588158, 0, 0]]

firstScoreScaler = [[StandardScaler(), StandardScaler(), StandardScaler(), None, None],

[StandardScaler(), StandardScaler(), StandardScaler(), None, None],

[StandardScaler(), StandardScaler(), StandardScaler(), None, None]]

firstScoreEncoder = [[OrdinalEncoder(), OrdinalEncoder(), OrdinalEncoder(), None, None],

[OrdinalEncoder(), OrdinalEncoder(), OrdinalEncoder(), None, None],

[OrdinalEncoder(), OrdinalEncoder(), OrdinalEncoder(), None, None]]

firstScoreFeature = [[['Principle-1', 'Principle-2', 'Principle-3'], ['Principle-1', 'Principle-2', 'Principle-3'],

['Principle-1', 'Principle-2', 'Principle-3'], None, None],

[['housing\_median\_age', 'ocean\_proximity', 'housing\_median\_age'],

['longitude', 'median\_income', 'ocean\_proximity'],

['ocean\_proximity', 'population', 'population'], None, None],

[["longitude", "latitude"], ["longitude", "latitude"], ["total\_rooms", "total\_bedrooms"], None, None]]

firstScoreModel = [[KMeans(init='random', max\_iter=100, n\_clusters=3),

GaussianMixture(covariance\_type='spherical', n\_components=2, n\_init=10, tol=1e-05),

DBSCAN(eps=0.4, min\_samples=3, p=1), None, None],

[KMeans(max\_iter=100, n\_clusters=2, n\_init=1),

GaussianMixture(covariance\_type='spherical', n\_components=2, tol=1e-05),

DBSCAN(eps=0.3, min\_samples=3, p=1), None, None],

[KMeans(init='random', n\_clusters=3, n\_init=1), GaussianMixture(covariance\_type='spherical', n\_components=4), DBSCAN(min\_samples=4, p=2), None, None]]

firstScoreParameter = [[{'init': 'random', 'max\_iter': 100, 'n\_clusters': 3, 'n\_init': 10},

{'covariance\_type': 'spherical', 'max\_iter': 100, 'n\_components': 2, 'n\_init': 10, 'tol': 1e-05},

{'algorithm': 'auto', 'eps': 0.4, 'metric': 'euclidean', 'min\_samples': 3, 'p': 1}, None, None],

[{'init': 'k-means++', 'max\_iter': 100, 'n\_clusters': 2, 'n\_init': 1},

{'covariance\_type': 'spherical', 'max\_iter': 100, 'n\_components': 2, 'n\_init': 1, 'tol': 1e-05},

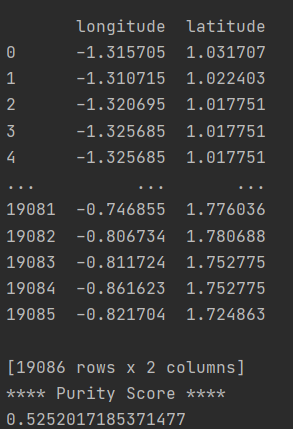
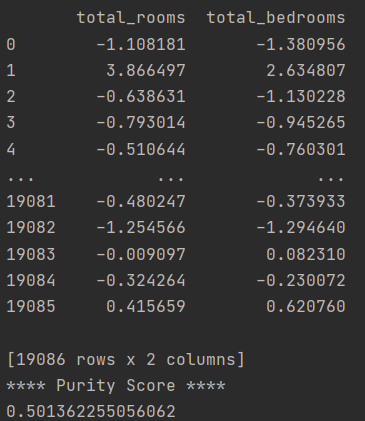
{'algorithm': 'auto', 'eps': 0.3, 'metric': 'euclidean', 'min\_samples': 3, 'p': 1}, None, None],

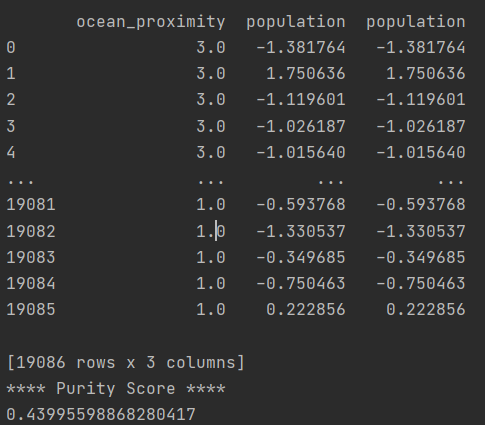
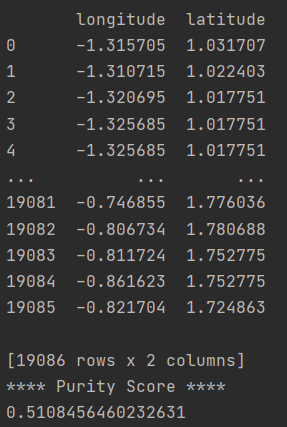
[{'init': 'random', 'n\_clusters': 3, 'n\_init': 1}, {'covariance\_type': 'spherical', 'n\_components': 4, 'n\_init': 1, 'tol': 0.001}, {'algorithm': 'auto', 'eps': 0.5, 'min\_samples': 4, 'p': 2}, None, None]]

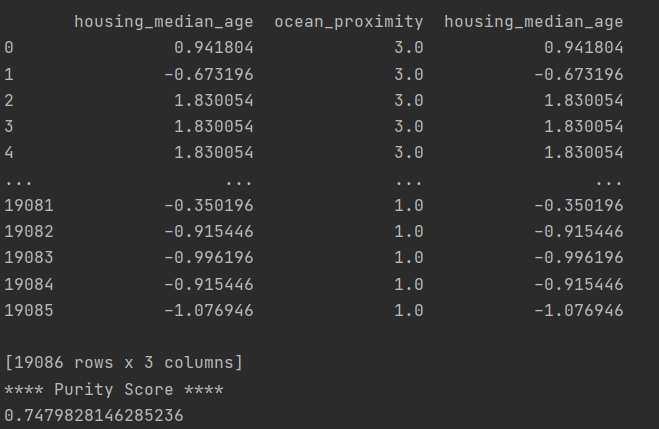
# Feature selection 3 \* model 5 = 15 combinations have the highest silhouette score

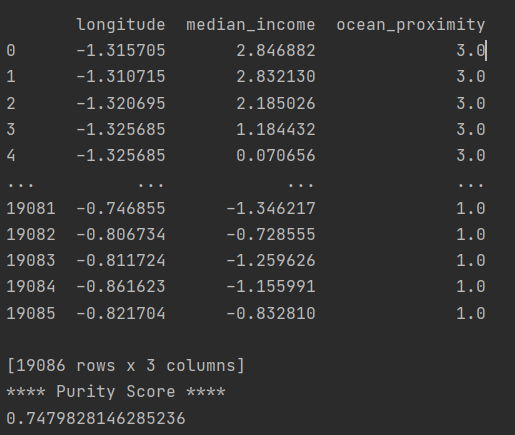
############################################################################

Step 2 = If there is a target value, Among the three Feature Selection (PCA(), RandomSelect(), CustomSelect()), check which model has the highest purity and return three results

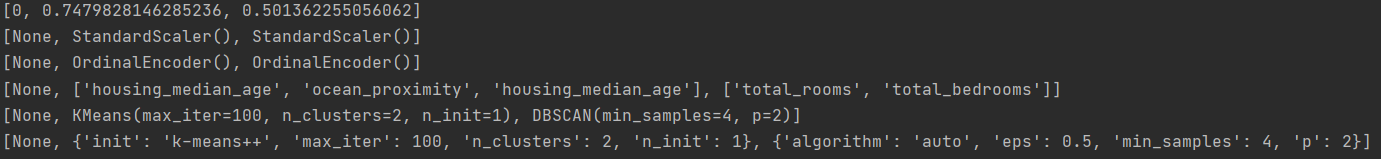
 





So, Step 2’s Result is

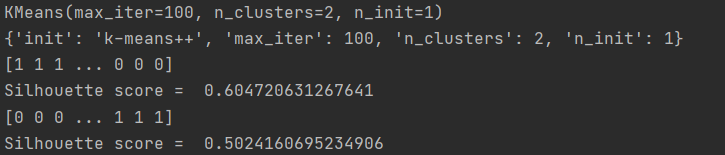


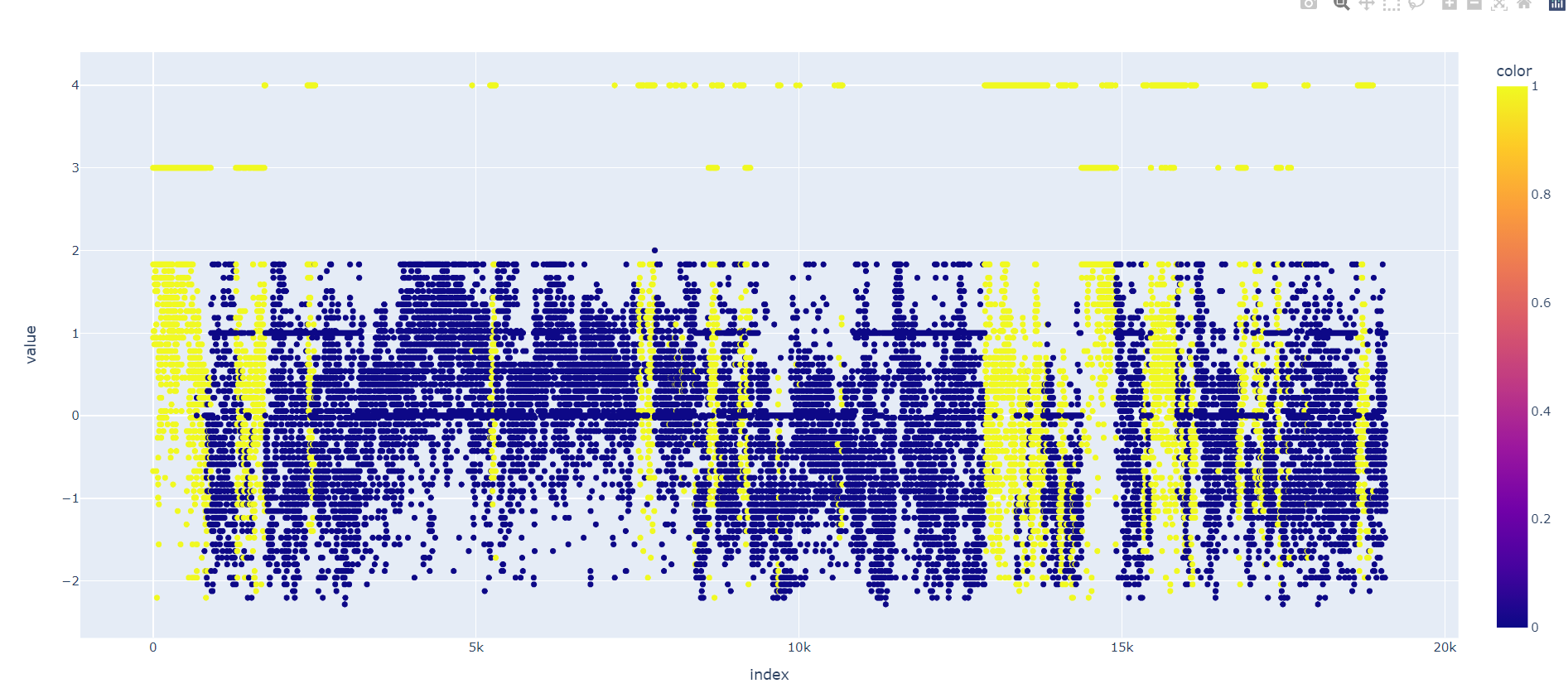
PCA was excluded because there was an error.

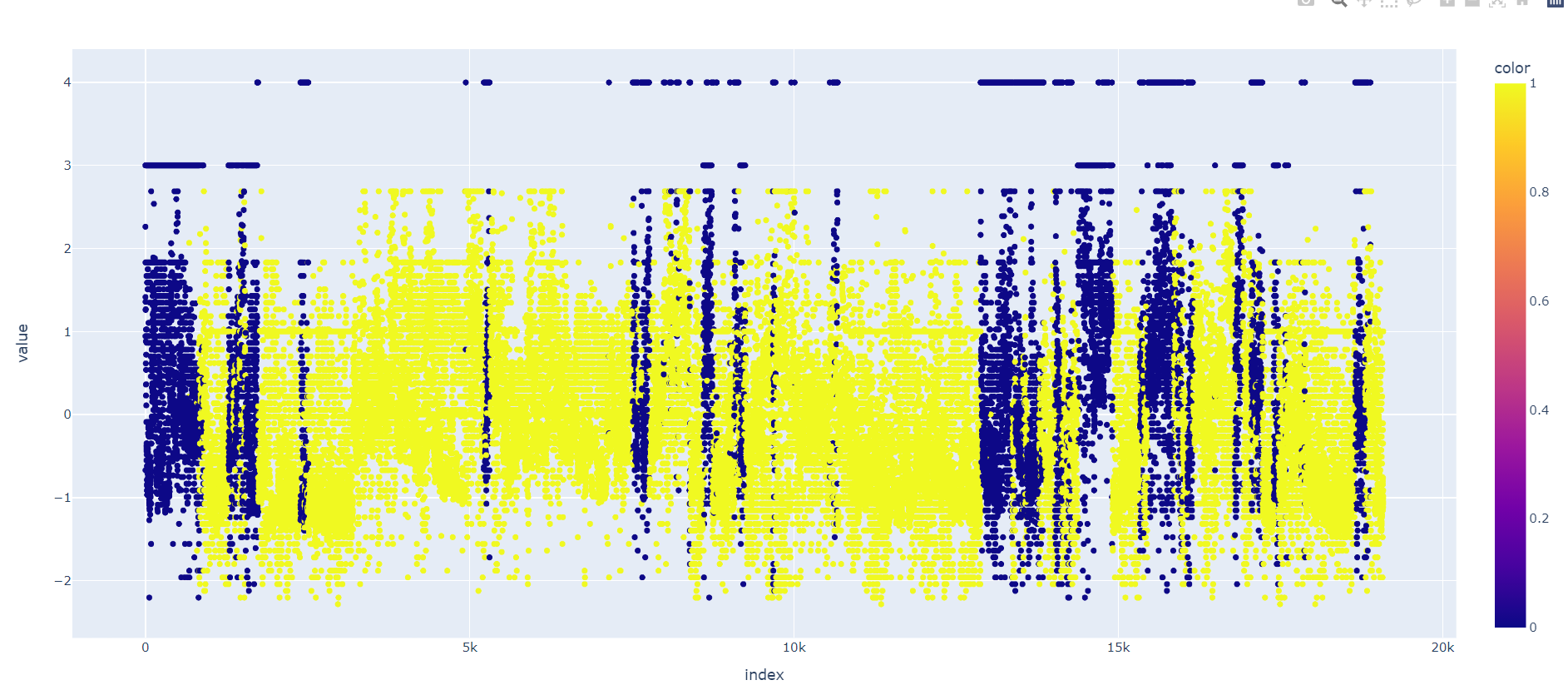
Step 3 = Using the final three combinations (without a target value),

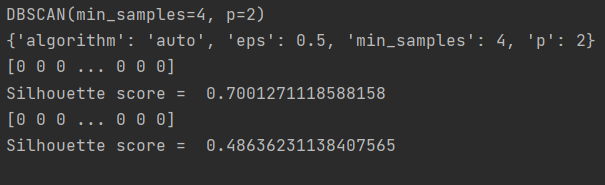
we compare with the combinations (with a target value)

- The results are checked through the clustering plot and the silhouette score -

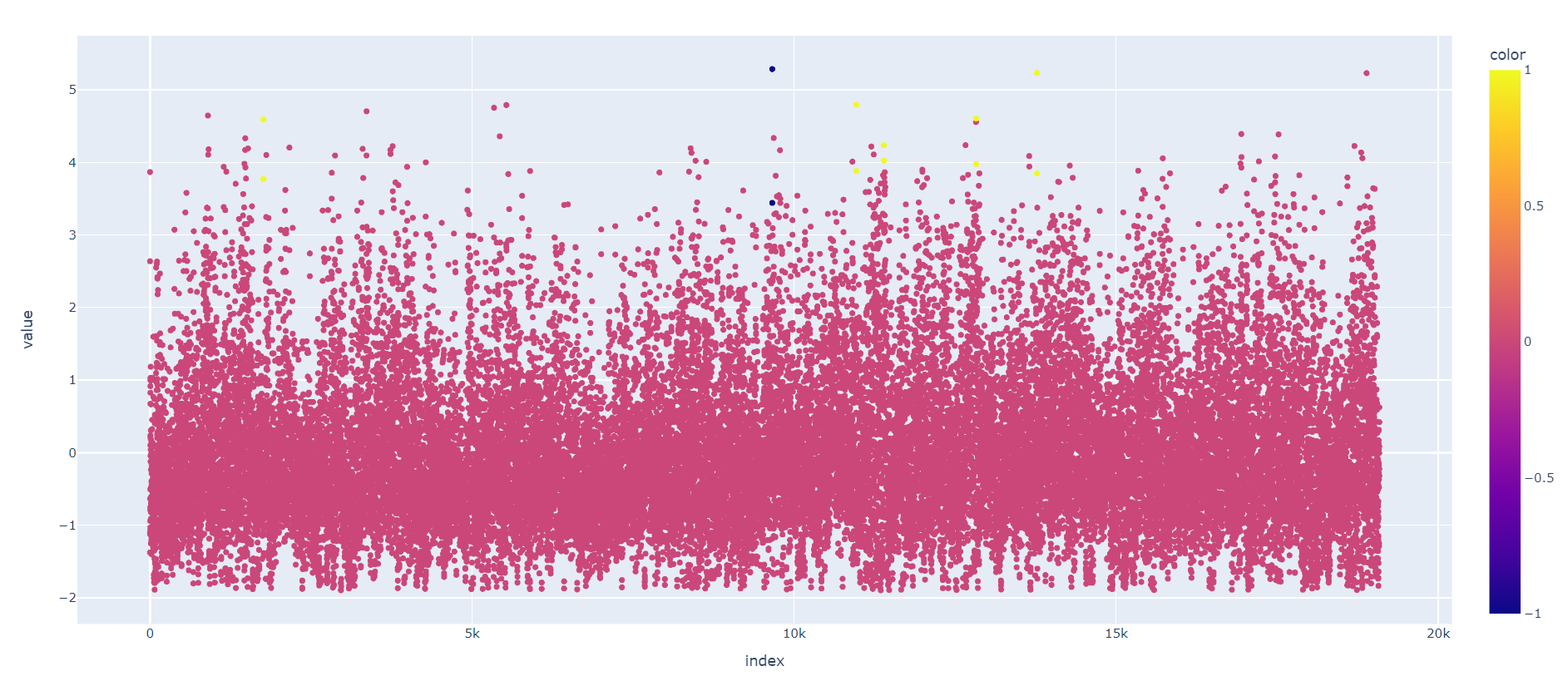
(KMeans with random selected features)

(without a target value)

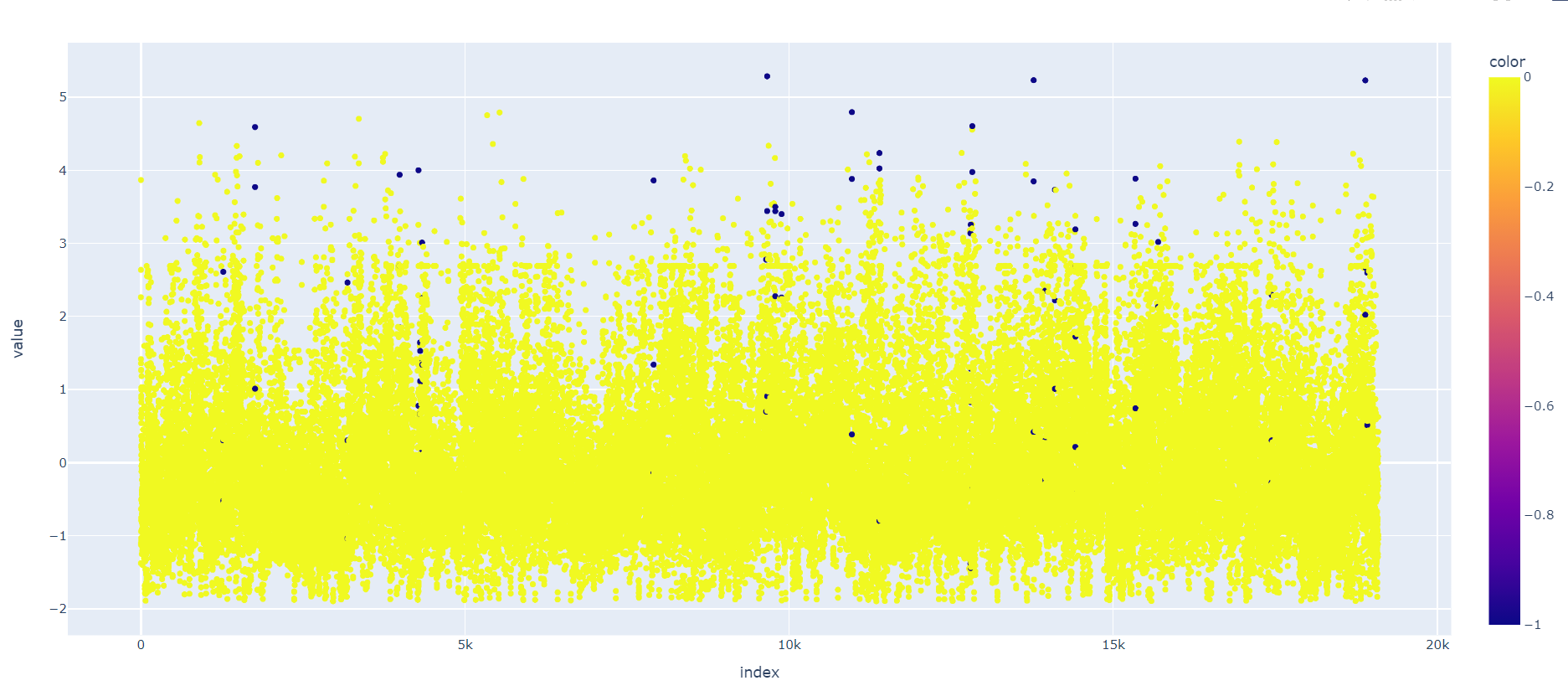
(with a target value)

(DBSCAN with selected features)

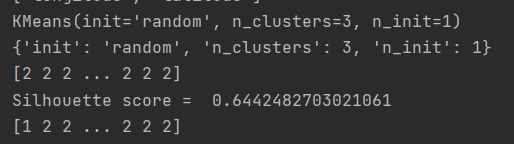
(without a target value)



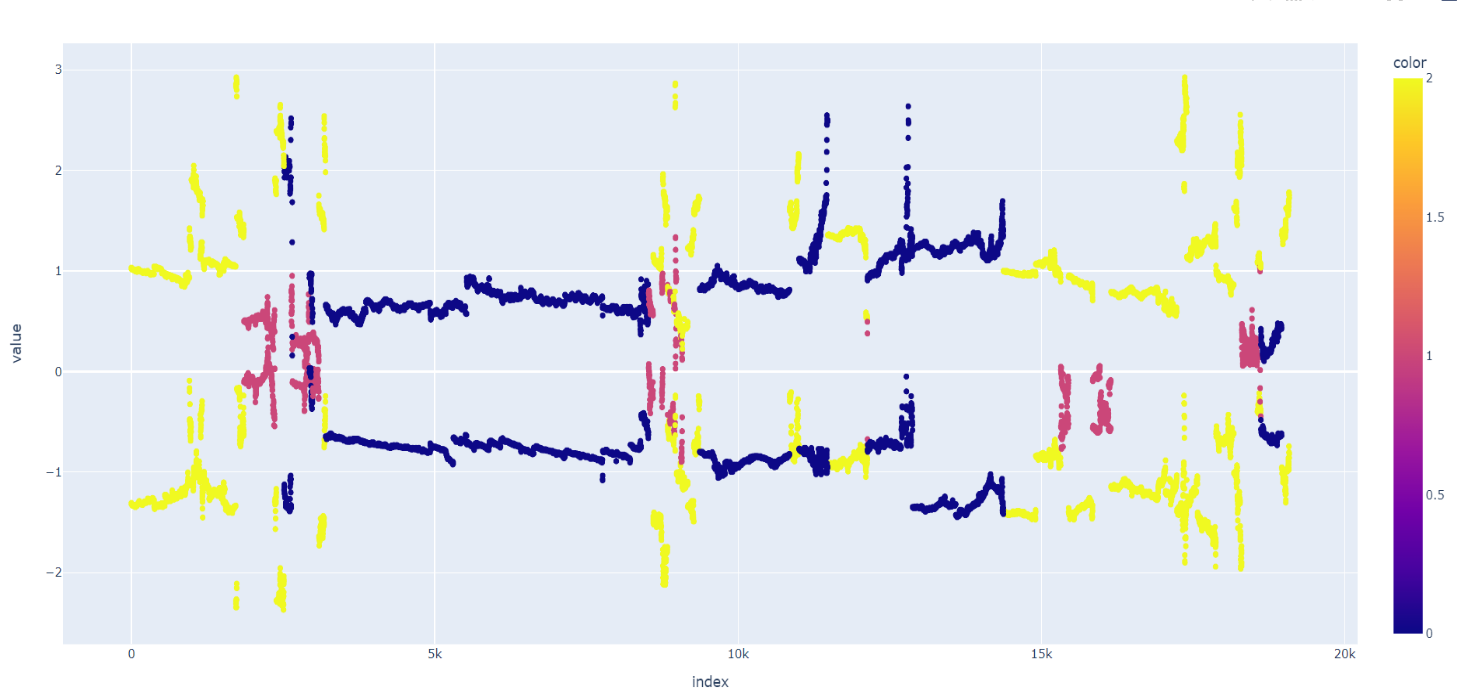
(with a target value)



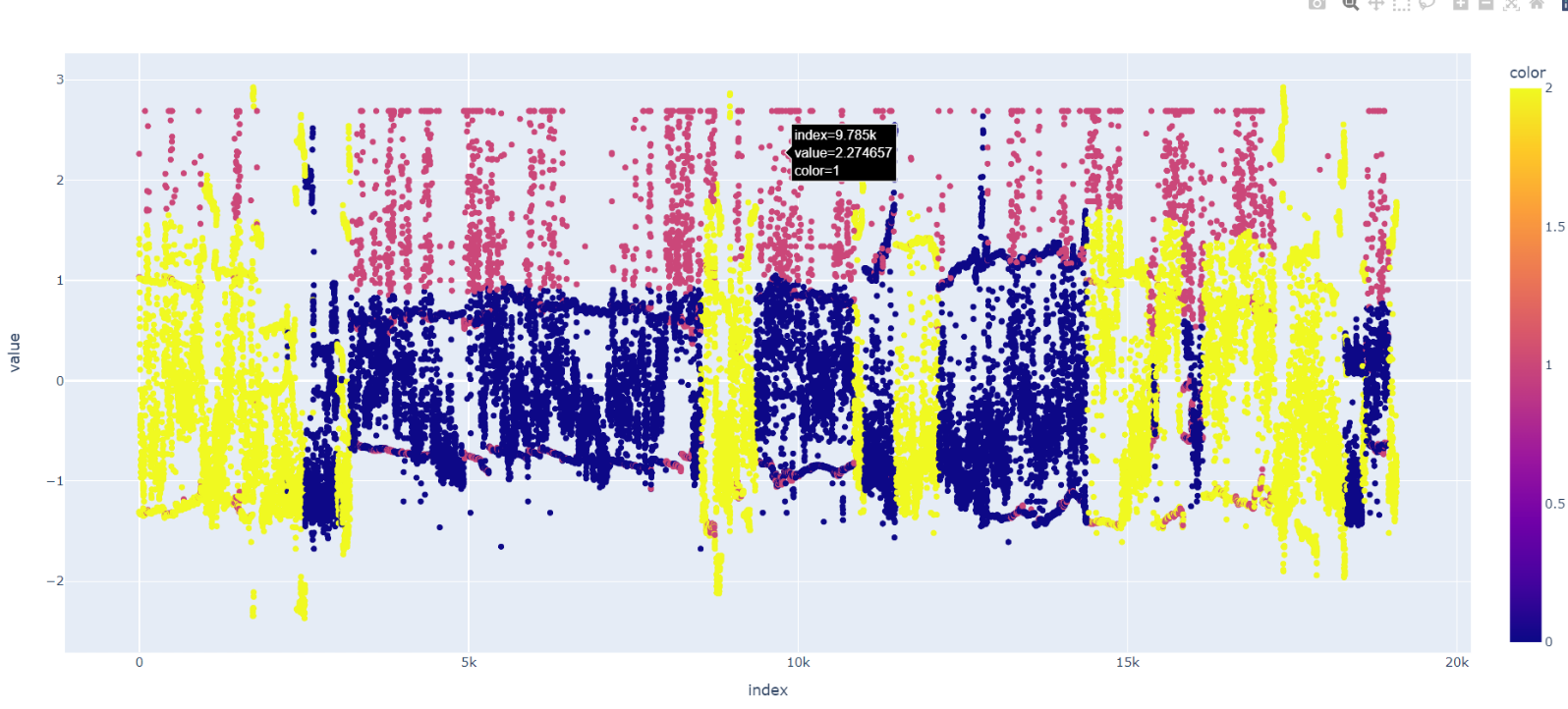
(KMeans with selected features)



(without a target value)



(with a target value)



Unlike classification, clustering was very difficult because there were so many things to worry about.

We tried clustering by varying feature selection and setting a lot of parameters, but no significant results came out.

And We think again that unsupervised learning was very difficult.

Whatever combination of clustering, the results were not satisfactory.

For refinement, We divided it into three steps, and We were able to learn a lot in designing the process.