PHW2\_방준석, 신민오, 송수현

**<Intermediate output>** Used various Scalers, Encoders, Silhouette\_score, Clustering Algorithms etc…

We used the combinations of a small number of parameters for this output example. This is because using all combinations takes a long time to check the results, so we couldn't check the results.텍스트, 신문이(가) 표시된 사진

자동 생성된 설명

텍스트, 신문, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 신문, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

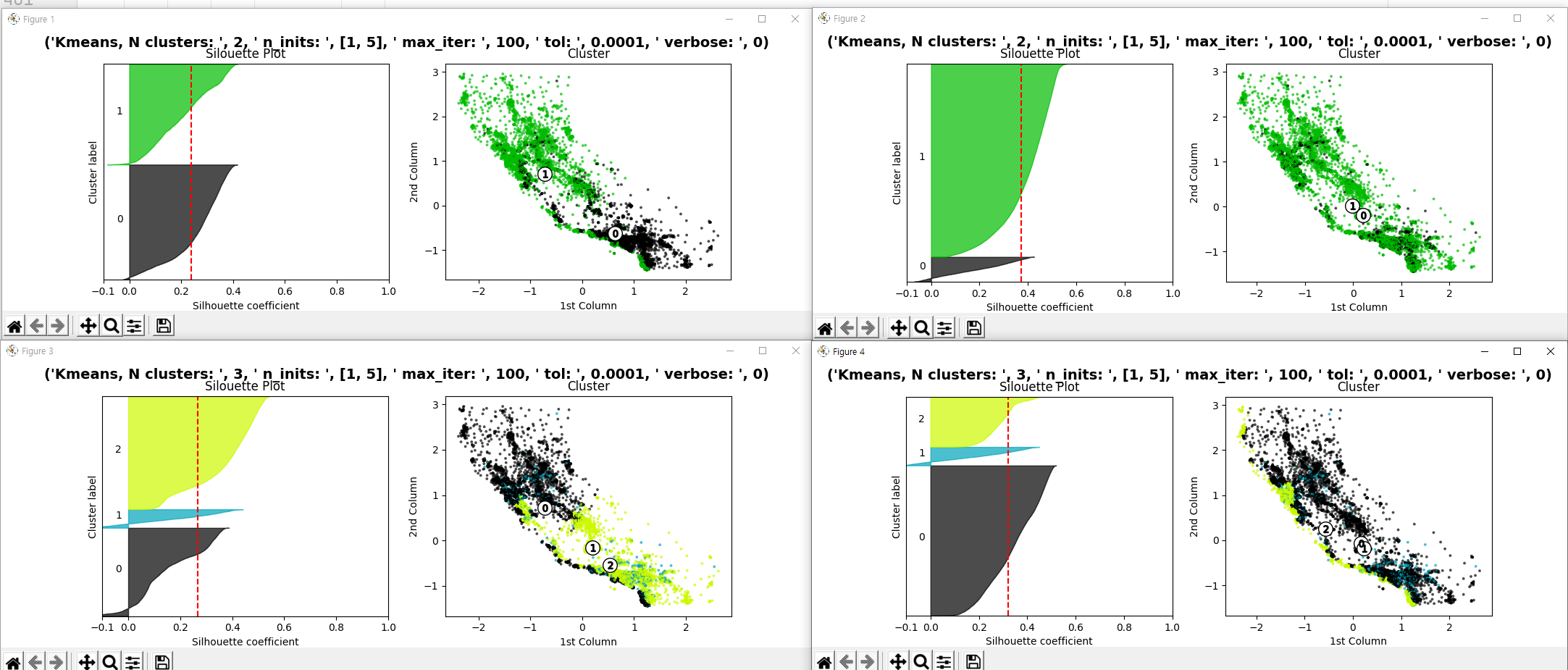
자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명



->Plotting Examples.  
The graph on the right shows clusters on the graph but, it is just feature1 and feature2’s graph. so It does not seems to be the result of the kmeans clustering algorithm.

**<Conclusion>**

we use various clustering algorithms **without ‘medianHouseValue’**

After clustering, we compared clusters and housing value

**<Insight> Compare clusters and medianHouseValue for each clustering Algorithms**

1. Combination Features: [housing\_median\_age, total\_rooms, total\_bedrooms, population, households, median\_income, ocean\_proximity]

텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* **We choose the KMeans Algorithm that shows the highest silhouette score.**

We compared the average housing Value of each cluster.

텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

the highest silhouette score was obtained when the cluster was divided into two.

3 features, housing\_median\_age, median\_income, and ocean\_proximity, which have large differences in values between clusters, effect on Mean\_House\_Value, and the remaining 4 features were small. It can be seen that 4 features had little effect on the Mean\_House\_Value.

2. Combination Features: [ housing\_median\_age, total\_rooms, total\_bedrooms]

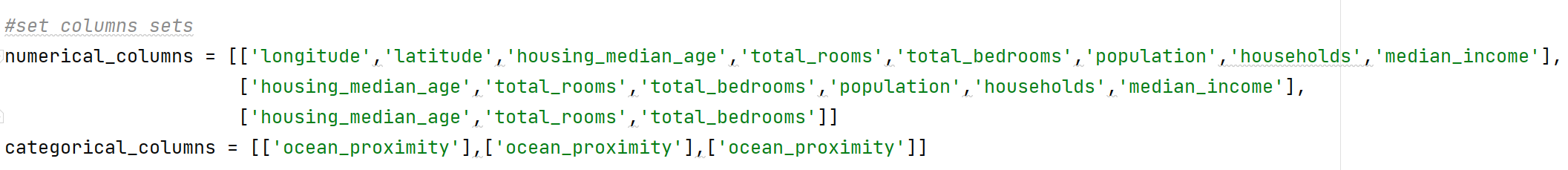


As a result of clustering a combination of three such as housing\_median\_age, total\_rooms, and total bedrooms, the highest silhouette score was obtained when the cluster was divided into two.

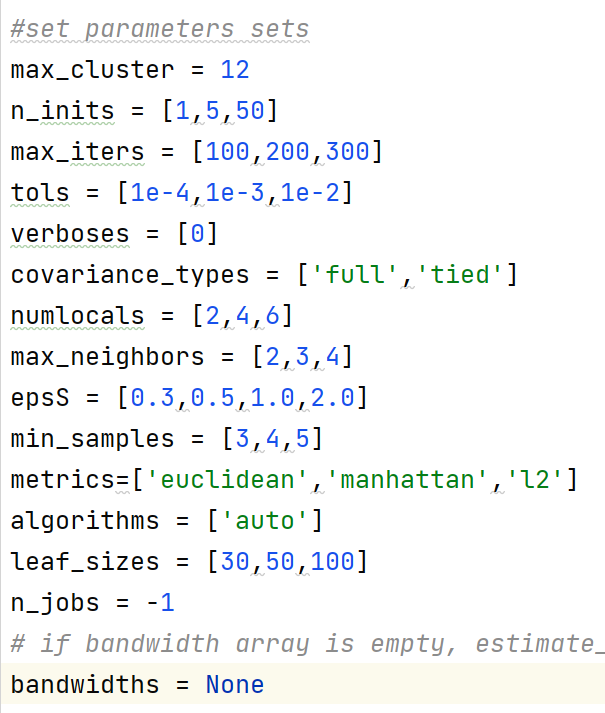
Comparing the results of clustering with Mean\_House\_Value, it was found that, as expected, **house prices were generally higher when new construction, many rooms, and the number of bedrooms were large.**

**3. The conditions of the assignment.**

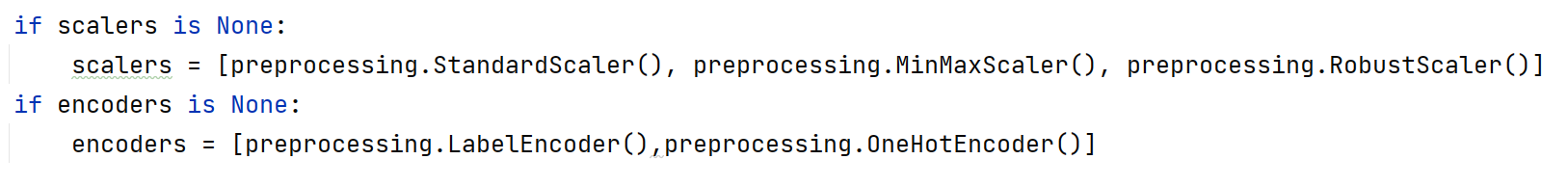
various combinations of features



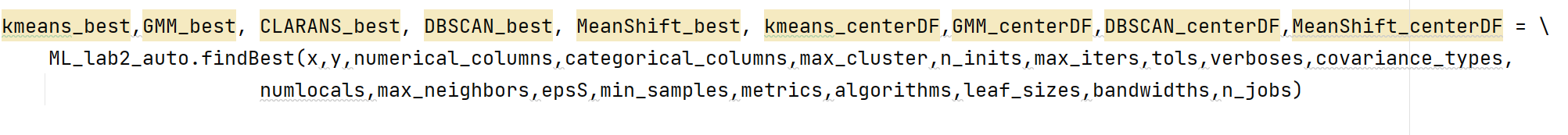
various number of clusters and parameters

****

various combinations of scalers and encoders

****

Step in the direction of “AutoML”

****

**4. Source code**

**4.1. ML\_lab2.py**

import pandas as pd  
import ML\_lab2\_auto  
  
pd.set\_option('display.max\_columns',None)  
*#pd.set\_option('display.max\_rows',None)*a = pd.read\_csv("D:/housing.csv")  
a.dropna(axis=0,inplace=True)  
a.reset\_index(inplace=True)  
y = a.loc[:,'median\_house\_value']  
x = a.loc[:,['longitude','latitude','housing\_median\_age','total\_rooms','total\_bedrooms','population','households','median\_income','ocean\_proximity']]  
  
*#set columns sets*numerical\_columns = [['longitude','latitude','housing\_median\_age','total\_rooms','total\_bedrooms','population','households','median\_income'],  
 ['housing\_median\_age','total\_rooms','total\_bedrooms','population','households','median\_income'],  
 ['housing\_median\_age','total\_rooms','total\_bedrooms']]  
categorical\_columns = [['ocean\_proximity'],['ocean\_proximity'],['ocean\_proximity']]  
  
*#set parameters sets*max\_cluster = 12  
n\_inits = [1,5,50]  
max\_iters = [100,200,300]  
tols = [1e-4,1e-3,1e-2]  
verboses = [0]  
covariance\_types = ['full','tied']  
numlocals = [2,4,6]  
max\_neighbors = [2,3,4]  
epsS = [0.3,0.5,1.0,2.0]  
min\_samples = [3,4,5]  
metrics=['euclidean','manhattan','l2']  
algorithms = ['auto']  
leaf\_sizes = [30,50,100]  
n\_jobs = -1  
*# if bandwidth array is empty, estimate\_bandwidth function estimate the bandwidth to use with the mean-shift algorithm.*bandwidths = None  
  
kmeans\_best = [-1,'scale','encode','params']  
GMM\_best = [-1,'scale','encode','params']  
CLARANS\_best = [-1,'scale','encode','params']  
DBSCAN\_best = [-1,'scale','encode','params']  
MeanShift\_best = [-1,'scale','encode','params']  
kmeans\_centerDF = pd.DataFrame()  
GMM\_centerDF = pd.DataFrame()  
DBSCAN\_centerDF = pd.DataFrame()  
MeanShift\_centerDF = pd.DataFrame()  
  
  
kmeans\_best,GMM\_best, CLARANS\_best, DBSCAN\_best, MeanShift\_best, kmeans\_centerDF,GMM\_centerDF,DBSCAN\_centerDF,MeanShift\_centerDF = \  
 ML\_lab2\_auto.findBest(x,y,numerical\_columns,categorical\_columns,max\_cluster,n\_inits,max\_iters,tols,verboses,covariance\_types,  
 numlocals,max\_neighbors,epsS,min\_samples,metrics,algorithms,leaf\_sizes,bandwidths,n\_jobs)  
  
print("===================Result===================\n")  
print("----------------KMeans----------------")  
print("Scaler: ", kmeans\_best[1], "Encoder: ", kmeans\_best[2])  
print("Silhouette Score: ", kmeans\_best[0])  
print(kmeans\_best[3])  
print("\*\*\*\*\*\*\*\*\*Best Center\*\*\*\*\*\*\*\*\*\*")  
print(kmeans\_centerDF)  
  
print("\n----------------GMM----------------")  
print("Scaler: ", GMM\_best[1], "Encoder: ", GMM\_best[2])  
print("Silhouette Score: ", GMM\_best[0])  
print(GMM\_best[3])  
print("\*\*\*\*\*\*\*\*\*Best Center\*\*\*\*\*\*\*\*\*\*")  
print(GMM\_centerDF)  
  
print("\n----------------CLARANS----------------")  
print("Scaler: ", CLARANS\_best[1], "Encoder: ", CLARANS\_best[2])  
print("Silhouette Score: ", CLARANS\_best[0])  
print(CLARANS\_best[3])  
  
print("----------------DBSCAN----------------")  
print("Scaler: ", DBSCAN\_best[1], "Encoder: ", DBSCAN\_best[2])  
print("Silhouette Score: ", DBSCAN\_best[0])  
print(DBSCAN\_best[3])  
print("\*\*\*\*\*\*\*\*\*Best Center\*\*\*\*\*\*\*\*\*\*")  
print(DBSCAN\_centerDF)  
  
print("\n----------------MeanShift----------------")  
print("Scaler: ", MeanShift\_best[1], "Encoder: ", MeanShift\_best[2])  
print("Silhouette Score: ", MeanShift\_best[0])  
print(MeanShift\_best[3])  
print("\*\*\*\*\*\*\*\*\*Best Center\*\*\*\*\*\*\*\*\*\*")  
print(MeanShift\_centerDF)

**3.1. ML\_lab2\_auto.py**

import numpy as np  
import pandas as pd  
from matplotlib import pyplot as plt, cm  
from sklearn import preprocessing, mixture, metrics  
from sklearn.cluster import KMeans, DBSCAN, MeanShift, estimate\_bandwidth  
from sklearn.metrics import silhouette\_score, silhouette\_samples  
  
def getEncode(df,name,encoder):  
 encoder.fit(df[name])  
 labels = encoder.transform(df[name])  
 df.loc[:, name] = labels  
  
def onehotEncode(df, name):  
 le = preprocessing.OneHotEncoder(handle\_unknown='ignore')  
 enc = df[[name]]  
 enc = le.fit\_transform(enc).toarray()  
 enc\_df = pd.DataFrame(enc, columns=le.categories\_[0])  
 df.loc[:, le.categories\_[0]] = enc\_df  
 df.drop(columns=[name], inplace=True)  
  
*#label encoding*def labelEncode(df, name):  
 encoder = preprocessing.LabelEncoder()  
 encoder.fit(df[name])  
 labels = encoder.transform(df[name])  
 df.loc[:, name] = labels  
  
"""  
:param X: feature values  
:param numerical\_columns: name of numerical columns (array of string)  
:param categorical\_columns: name of categorical columns (array of string)  
:param scalers: array of scalers  
:param encoders: array of encoders   
:param scaler\_name: name of scalers (array of string)  
:param encoder\_name: name of encoders (array of string)  
:return: 2d array that is scaled and encoded X   
"""  
def get\_various\_encode\_scale(X, numerical\_columns, categorical\_columns, scalers=None, encoders= None,scaler\_name=None,encoder\_name=None):  
  
 if categorical\_columns is None:  
 categorical\_columns = []  
 if numerical\_columns is None:  
 numerical\_columns = []  
 if len(categorical\_columns) == 0:  
 return get\_various\_scale(X,numerical\_columns,scalers,scaler\_name)  
 if len(numerical\_columns) == 0:  
 return get\_various\_encode(X,categorical\_columns,encoders,encoder\_name)  
  
 if scalers is None:  
 scalers = [preprocessing.StandardScaler(), preprocessing.MinMaxScaler(), preprocessing.RobustScaler()]  
 if encoders is None:  
 encoders = [preprocessing.LabelEncoder(),preprocessing.OneHotEncoder()]  
  
 after\_scale\_encode = [[0 for col in range(len(encoders))] for row in range(len(scalers))]  
  
 i=0  
 for scale in scalers:  
 for encode in encoders:  
 after\_scale\_encode[i].pop()  
 for encode in encoders:  
 after\_scale\_encode[i].append(X.copy())  
 i=i+1  
  
 for name in numerical\_columns:  
 i=0  
 for scaler in scalers:  
 j=0  
 for encoder in encoders:  
 after\_scale\_encode[i][j][name] = scaler.fit\_transform(X[name].values.reshape(-1, 1))  
 j=j+1  
 i=i+1  
  
 for new in categorical\_columns:  
 i=0  
 for scaler in scalers:  
 j=0  
 for encoder in encoders:  
 if (str(type(encoder)) == "<class 'sklearn.preprocessing.\_label.LabelEncoder'>"):  
 labelEncode(after\_scale\_encode[i][j], new)  
 elif (str(type(encoder)) == "<class 'sklearn.preprocessing.\_encoders.OneHotEncoder'>"):  
 onehotEncode(after\_scale\_encode[i][j], new)  
 else:  
 getEncode(after\_scale\_encode[i][j], new, encoder)  
 j=j+1  
 i=i+1  
  
 return after\_scale\_encode,scalers,encoders  
  
"""  
If there aren't categorical value, do this function  
This function only scales given X  
Return: 1d array of scaled X  
"""  
def get\_various\_scale(X, numerical\_columns, scalers=None,scaler\_name=None):  
  
 *"""  
 Test scale/encoder sets  
 """* if scalers is None:  
 scalers = [preprocessing.StandardScaler(), preprocessing.MinMaxScaler(), preprocessing.RobustScaler()]  
 *#scalers = [preprocessing.StandardScaler()]* encoders = ["None"]  
  
 after\_scale = [[0 for col in range(1)] for row in range(len(scalers))]  
  
 i = 0  
 for scale in scalers:  
 for encode in encoders:  
 after\_scale[i].pop()  
 for encode in encoders:  
 after\_scale[i].append(X.copy())  
 i = i + 1  
  
 for name in numerical\_columns:  
 i=0  
 for scaler in scalers:  
 after\_scale[i][0][name] = scaler.fit\_transform(X[name].values.reshape(-1,1))  
 i=i+1  
  
 return after\_scale,scalers,["None"]  
  
"""  
If there aren't numerical value, do this function  
This function only encodes given X  
Return: 1d array of encoded X  
"""  
def get\_various\_encode(X, categorical\_columns, encoders=None,encoder\_name=None):  
  
 *"""  
 Test scale/encoder sets  
 """* if encoders is None:  
 *#encoders = [preprocessing.LabelEncoder(),preprocessing.OneHotEncoder()]* encoders = [preprocessing.LabelEncoder()]  
 scalers = ["None"]  
  
 after\_encode = [[0 for col in range(1)] for row in range(len(scalers))]  
  
 i = 0  
 for scale in scalers:  
 for encode in encoders:  
 after\_encode[i].pop()  
 for encode in encoders:  
 after\_encode[i].append(X.copy())  
 i = i + 1  
  
 for new in categorical\_columns:  
 j = 0  
 for encoder in encoders:  
 if (str(type(encoder)) == "<class 'sklearn.preprocessing.\_label.LabelEncoder'>"):  
 labelEncode(after\_encode[0][j], new)  
 elif (str(type(encoder)) == "<class 'sklearn.preprocessing.\_encoders.OneHotEncoder'>"):  
 onehotEncode(after\_encode[0][j], new)  
 else:  
 getEncode(after\_encode[0][j], new, encoder)  
 j = j + 1  
  
  
 return after\_encode,["None"],encoders  
  
"""  
:param X: dataset  
:param max\_cluster: maximum number of clusters  
:param n\_inits: Number of time the k-means algorithm will be run with different centroid seeds.  
:param max\_iters: Maximum number of iterations of the k-means algorithm for a single run  
:param tols: Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence.  
:param verboses: Verbosity mode.  
:param random\_state  
"""  
def kmeans(X,y,max\_cluster=None, n\_inits=None, max\_iters=None, tols=None, verboses=None, random\_state=None):  
  
 if max\_cluster is None:  
 max\_cluster = 7  
 max\_cluster = max\_cluster + 1  
  
 range\_n\_clusters = list(range(max\_cluster))  
 range\_n\_clusters.remove(0)  
 range\_n\_clusters.remove(1)  
  
 if n\_inits is None:  
 n\_inits = [10]  
 if max\_iters is None:  
 max\_iters = [300]  
 if tols is None:  
 tols = [1e-4]  
 if verboses is None:  
 verboses = [0]  
  
 best\_cluster = -1  
 best\_silhouette= -1  
 best\_n\_inits = 0  
 best\_max\_iters = 0  
 best\_tols = 0  
 best\_verboses = 0  
  
 centerDF = pd.DataFrame  
  
 for n\_clusters in range\_n\_clusters:  
 for n\_init in n\_inits:  
 for max\_iter in max\_iters:  
 for tol in tols:  
 for verbose in verboses:  
 print("number of clusters: ", n\_clusters,"/ n\_init:", n\_init,"/ max\_iter:", max\_iter,"/ tol:", tol,"/ verbose:", verbose)  
  
 fig, (ax1, ax2) = plt.subplots(1, 2)  
 fig.set\_size\_inches(18, 7)  
  
 ax1.set\_xlim([-0.1, 1])  
 ax1.set\_ylim([0, len(X) + (n\_clusters + 1) \* 10])  
  
 clusterer = KMeans(n\_clusters=n\_clusters,n\_init=n\_init,max\_iter=max\_iter,tol=tol,verbose=verbose, random\_state=random\_state)  
 cluster\_labels = clusterer.fit\_predict(X)  
  
 silhouette\_avg = silhouette\_score(X, cluster\_labels)  
 centers = clusterer.cluster\_centers\_  
  
 if best\_silhouette<silhouette\_avg:  
 best\_silhouette = silhouette\_avg  
 best\_cluster = n\_clusters  
 best\_n\_inits = n\_init  
 best\_max\_iters = max\_iter  
 best\_tols = tol  
 best\_verboses = verbose  
  
 sum = [0 for row in range(n\_clusters)]  
 num = [0 for row in range(n\_clusters)]  
  
 j = 0  
 for i in cluster\_labels:  
 sum[i] = sum[i] + y[j]  
 num[i] = num[i] + 1  
 j = j + 1  
  
 for i in range(n\_clusters):  
 sum[i] = sum[i] / num[i]  
 centerDF = pd.DataFrame(centers)  
 centerDF.loc[:, 'Mean House Value'] = sum  
  
  
 print("The average silhouette\_score is :", silhouette\_avg)  
  
 sample\_silhouette\_values = silhouette\_samples(X, cluster\_labels)  
  
 y\_lower = 10  
 for i in range(n\_clusters):  
  
 ith\_cluster\_silhouette\_values = sample\_silhouette\_values[cluster\_labels == i]  
  
 ith\_cluster\_silhouette\_values.sort()  
  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
  
 color = cm.nipy\_spectral(float(i) / n\_clusters)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper),  
 0, ith\_cluster\_silhouette\_values,  
 facecolor=color, edgecolor=color, alpha=0.7)  
  
 ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
  
 y\_lower = y\_upper + 10 *# 10 for the 0 samples* ax1.set\_title("Silouette Plot")  
 ax1.set\_xlabel("Silhouette coefficient")  
 ax1.set\_ylabel("Cluster label")  
  
 ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")  
  
 ax1.set\_yticks([]) *# Clear the yaxis labels / ticks* ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 colors = cm.nipy\_spectral(cluster\_labels.astype(float) / n\_clusters)  
 ax2.scatter(X.iloc[:, 0], X.iloc[:, 1], marker='.', s=30, lw=0, alpha=0.7,  
 c=colors, edgecolor='k')  
  
 ax2.scatter(centers[:, 0], centers[:, 1], marker='o',c="white", alpha=1, s=200, edgecolor='k')  
  
 for i, c in enumerate(centers):  
 ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,s=50, edgecolor='k')  
  
 ax2.set\_title("Cluster")  
 ax2.set\_xlabel("1st Column")  
 ax2.set\_ylabel("2nd Column")  
  
 plt.suptitle(("Kmeans, N clusters: ",n\_clusters ," n\_inits: ",n\_inits," max\_iter: ",max\_iter," tol: ",tol," verbose: ",verbose),fontsize=14, fontweight='bold')  
  
 plt.show()  
 df = centerDF.copy()  
 print("\nThe highest silhouette score is ", best\_silhouette, " with ",best\_cluster," clusters")  
 print("Best params\_/ n\_init:",best\_n\_inits,"/ max\_iter:",best\_max\_iters,"/ tol:",best\_tols,"/ verbose:",best\_verboses,"\n")  
 param = 'Best params\_/ best cluster: '+str(best\_cluster)+ '/ n\_init: '+str(best\_n\_inits)+' / max\_iter: '+str(best\_max\_iters)+'/ tol: '+str(best\_tols)+'/ verbose: '+str(best\_verboses)  
 return best\_silhouette, param, df  
"""  
:param X: dataset  
:param max\_cluster: maximum number of clusters  
:param covariance\_types: String describing the type of covariance parameters to use.   
:param tols: The convergence threshold.  
:param max\_iters: The number of initializations to perform.  
:param n\_inits: The number of initializations to perform.  
:param random\_state  
"""  
def GMM(X,y,max\_cluster=None,covariance\_types=None,tols=None,max\_iters=None,n\_inits=None,random\_state=None):  
  
 if max\_cluster is None:  
 max\_cluster = 7  
 max\_cluster = max\_cluster + 1  
  
 if covariance\_types is None:  
 covariance\_types = ['full']  
 if tols is None:  
 tols = [1e-3]  
 if max\_iters is None:  
 max\_iters = [100]  
 if n\_inits is None:  
 n\_inits = [1]  
  
 range\_n\_clusters = list(range(max\_cluster))  
 range\_n\_clusters.remove(0)  
 range\_n\_clusters.remove(1)  
  
 best\_cluster = -1  
 best\_silhouette = -1  
 best\_covariance\_type = ''  
 best\_tol = 0  
 best\_max\_iter = 0  
 best\_n\_init = 0  
  
 centerDF = pd.DataFrame  
  
 for n\_clusters in range\_n\_clusters:  
 for covariance\_type in covariance\_types:  
 for tol in tols:  
 for max\_iter in max\_iters:  
 for n\_init in n\_inits:  
 print("number of clusters: ", n\_clusters, "/ covariance type:", covariance\_type, "/ n\_init:", n\_init, "/ max\_iter:", max\_iter,  
 "/ tol:", tol)  
  
 fig, (ax1, ax2) = plt.subplots(1, 2)  
 fig.set\_size\_inches(18, 7)  
  
 ax1.set\_xlim([-0.1, 1])  
 ax1.set\_ylim([0, len(X) + (n\_clusters + 1) \* 10])  
  
 clusterer = mixture.GaussianMixture(n\_components=n\_clusters, covariance\_type=covariance\_type,tol=tol,max\_iter=max\_iter,n\_init=n\_init)  
 clusterer.fit(X)  
 cluster\_labels = clusterer.predict(X)  
  
 silhouette\_avg = silhouette\_score(X, cluster\_labels)  
 print("The average silhouette\_score is :", silhouette\_avg)  
  
 *# Labeling the clusters* centers = clusterer.means\_  
  
 if best\_silhouette<silhouette\_avg:  
 best\_silhouette = silhouette\_avg  
 best\_cluster = n\_clusters  
 best\_covariance\_type = covariance\_type  
 best\_tol = tol  
 best\_max\_iter = max\_iter  
 best\_n\_init = n\_init  
  
 sum = [0 for row in range(n\_clusters)]  
 num = [0 for row in range(n\_clusters)]  
  
 j = 0  
 for i in cluster\_labels:  
 sum[i] = sum[i] + y[j]  
 num[i] = num[i] + 1  
 j = j + 1  
  
 for i in range(n\_clusters):  
 sum[i] = sum[i] / num[i]  
 centerDF = pd.DataFrame(centers)  
 centerDF.loc[:, 'Mean House Value'] = sum  
  
 sample\_silhouette\_values = silhouette\_samples(X, cluster\_labels)  
  
 y\_lower = 10  
 for i in range(n\_clusters):  
 ith\_cluster\_silhouette\_values = sample\_silhouette\_values[cluster\_labels == i]  
  
 ith\_cluster\_silhouette\_values.sort()  
  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
  
 color = cm.nipy\_spectral(float(i) / n\_clusters)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper),0, ith\_cluster\_silhouette\_values,facecolor=color, edgecolor=color, alpha=0.7)  
  
 ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
  
 y\_lower = y\_upper + 10 *# 10 for the 0 samples* ax1.set\_title("Silouette Plot")  
 ax1.set\_xlabel("Silhouette coefficient")  
 ax1.set\_ylabel("Cluster label")  
  
 ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")  
  
 ax1.set\_yticks([]) *# Clear the yaxis labels / ticks* ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 *# 2nd Plot showing the actual clusters formed* colors = cm.nipy\_spectral(cluster\_labels.astype(float) / n\_clusters)  
 ax2.scatter(X.iloc[:, 0], X.iloc[:, 1], marker='.', s=30, lw=0, alpha=0.7,  
 c=colors, edgecolor='k')  
  
 *# Draw white circles at cluster centers* ax2.scatter(centers[:, 0], centers[:, 1], marker='o',  
 c="white", alpha=1, s=200, edgecolor='k')  
  
 for i, c in enumerate(centers):  
 ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,  
 s=50, edgecolor='k')  
  
 ax2.set\_title("Cluster")  
 ax2.set\_xlabel("1st Column")  
 ax2.set\_ylabel("2nd Column")  
  
 plt.suptitle(("GMM n\_clusters: ",n\_clusters," "),  
 fontsize=14, fontweight='bold')  
  
 plt.show()  
  
 print("\nThe highest silhouette score is ", best\_silhouette, " with ", best\_cluster, " clusters")  
 print("Best params\_/ covariance\_types:", best\_covariance\_type, "/ max\_iter:", best\_max\_iter, "/ tol:", best\_tol, "/ n\_init:",  
 best\_n\_init,"\n")  
 param = "Best params\_/ cluster: "+str(best\_cluster)+"/ covariance\_types:"+ best\_covariance\_type+ "/ max\_iter:"+ str(best\_max\_iter)+ "/ tol:"+ str(best\_tol)+ "/ n\_init:"+str(best\_n\_init)  
 return best\_silhouette, param, centerDF  
  
"""  
:param X: dataset  
:param max\_cluster: maxinum number of clusters  
:param numlocals: The number of local minima obtained  
:param maxneighbors: The maximum number of neighbors examined  
"""  
def clarans(X,y,max\_cluster=None, numlocals=None,maxneighbors=None):  
  
 from pyclustering.cluster.clarans import clarans;  
  
 if max\_cluster is None:  
 max\_cluster = 7  
 max\_cluster = max\_cluster + 1  
  
 if numlocals is None:  
 numlocals = [2]  
 if maxneighbors is None:  
 maxneighbors = [2]  
  
 range\_n\_clusters = list(range(max\_cluster))  
 range\_n\_clusters.remove(0)  
 range\_n\_clusters.remove(1)  
  
 best\_cluster = -1  
 best\_silhouette = -1  
 best\_numlocal = 0  
 best\_maxneighbor = 0  
  
 for n\_clusters in range\_n\_clusters:  
 for numlocal in numlocals:  
 for maxneighbor in maxneighbors:  
  
 *#CLARANS takes too much time so, reduce the number of row* data = X.iloc[0:500,:]  
 data = data.values.tolist()  
  
 print("number of clusters: ", n\_clusters, "/ numlocal:", numlocal, "/ maxneighbor:", maxneighbor)  
  
 fig, (ax1, ax2) = plt.subplots(1, 2)  
 fig.set\_size\_inches(18, 7)  
  
 ax1.set\_xlim([-0.1, 1])  
 ax1.set\_ylim([0, len(data) + (n\_clusters + 1) \* 10])  
  
 clarans\_instance = clarans(data, n\_clusters, numlocal, maxneighbor);  
  
 clarans\_instance.process()  
 clusters = clarans\_instance.get\_clusters();  
  
 i = 0  
 a = []  
 for cluster in clusters:  
 for index in cluster:  
 a.insert(index, i)  
 i = i + 1  
  
 cluster\_labels = np.array(a)  
  
 silhouette\_avg = silhouette\_score(data, cluster\_labels)  
 print("The average silhouette\_score is :", silhouette\_avg)  
  
 if best\_silhouette<silhouette\_avg:  
 best\_silhouette = silhouette\_avg  
 best\_cluster = n\_clusters  
 best\_numlocal = numlocal  
 best\_maxneighbor = maxneighbor  
  
 sample\_silhouette\_values = silhouette\_samples(data, cluster\_labels)  
  
 y\_lower = 10  
 for i in range(n\_clusters):  
 ith\_cluster\_silhouette\_values = sample\_silhouette\_values[cluster\_labels == i]  
  
 ith\_cluster\_silhouette\_values.sort()  
  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
  
 color = cm.nipy\_spectral(float(i) / n\_clusters)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_cluster\_silhouette\_values, facecolor=color,  
 edgecolor=color, alpha=0.7)  
  
 ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
  
 y\_lower = y\_upper + 10 *# 10 for the 0 samples* ax1.set\_title("Silouette Plot")  
 ax1.set\_xlabel("Silhouette coefficient")  
 ax1.set\_ylabel("Cluster label")  
  
 ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")  
  
 ax1.set\_yticks([])  
 ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 colors = cm.nipy\_spectral(cluster\_labels.astype(float) / n\_clusters)  
   
 ax2.scatter(data.iloc[:, 0], data.iloc[:, 1], marker='.', s=30, lw=0, alpha=0.7,  
 c=colors, edgecolor='k')  
  
 ax2.set\_title("Cluster")  
 ax2.set\_xlabel("1st Column")  
 ax2.set\_ylabel("2nd Column")  
  
 plt.suptitle(("Silhouette analysis for CLARANS clustering on sample data "  
 "with n\_clusters = %d" % n\_clusters),  
 fontsize=14, fontweight='bold')  
  
 plt.show()  
  
 print("\nThe highest silhouette score is ", best\_silhouette, " with ", best\_cluster, " clusters")  
 print("Best params\_/ numlocal:", best\_numlocal, "/ max\_neighbor:", best\_maxneighbor,"\n")  
 param = "Best params\_/ cluster: "+str(best\_cluster)+"/ numlocal:"+ str(best\_numlocal)+ "/ max\_neighbor:"+str(best\_maxneighbor)  
 return best\_silhouette, param  
  
"""  
:param X: datasets  
:param epsS: The maximum distance between two samples for one to be considered as in the neighborhood of the other.  
:param min\_samples: The number of samples (or total weight) in a neighborhood for a point to be considered as a core point.  
:param metrics: The metric to use when calculating distance between instances in a feature array  
:param algorithms: The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors  
:param leaf\_sizes  
 """  
def DBSCANs(X,y, epsS=None,min\_samples=None,metrics=None,algorithms=None,leaf\_sizes=None):  
  
 if epsS is None:  
 epsS = [0.8]  
 if min\_samples is None:  
 min\_samples = [3]  
 if metrics is None:  
 metrics = ['euclidean']  
 if algorithms is None:  
 algorithms = ['auto']  
 if leaf\_sizes is None:  
 leaf\_sizes = [30]  
  
 best\_silhouette = -1  
 best\_cluster = -1  
 best\_eps = 0  
 best\_min\_sample=0  
 best\_metric = ''  
 best\_algorithm = ''  
 best\_leaf\_size = 0  
  
 centerDF = pd.DataFrame  
  
 for eps in epsS:  
 for min\_sample in min\_samples:  
 for metric in metrics:  
 for algorithm in algorithms:  
 for leaf\_size in leaf\_sizes:  
 np.set\_printoptions(threshold=100000,linewidth=np.inf)  
  
 fig, (ax1, ax2) = plt.subplots(1, 2)  
 fig.set\_size\_inches(18, 7)  
  
 clusterer = DBSCAN(eps=eps,min\_samples=min\_sample,metric=metric,algorithm=algorithm,leaf\_size=leaf\_size).fit(X)  
 cluster\_labels = clusterer.labels\_  
  
 n\_clusters = len(set(clusterer.labels\_))  
  
 unique\_set = set(clusterer.labels\_)  
 unique\_list = list(unique\_set)  
 if unique\_list.count(-1):  
 unique\_list.remove(-1)  
  
 a = np.array([[0 for col in range(len(X.iloc[0,:]))] for row in range(len(set(unique\_list)))],dtype=np.float64)  
 num = np.array([0 for row in range(len(set(unique\_list)))])  
  
 i = 0  
 for cluster in cluster\_labels:  
 if (cluster != -1):  
 a[cluster] = a[cluster] + X.iloc[i,:]  
 num[cluster] = num[cluster] + 1  
 i = i + 1  
  
 i = 0  
  
 for cluster in unique\_list:  
 a[cluster] = a[cluster] / num[cluster]  
  
  
 ax1.set\_xlim([-0.1, 1])  
 ax1.set\_ylim([0, len(X) + (n\_clusters + 1) \* 10])  
  
 silhouette\_avg = silhouette\_score(X, cluster\_labels)  
 print("number of clusters: ", n\_clusters, "/ eps:", eps, "/ min\_sample:", min\_sample,  
 "/ metric:", metric, "/ algorithm:", algorithm, "/ leaf\_size:", leaf\_size)  
 print("The average silhouette\_score is :", silhouette\_avg)  
  
 centers = np.array(a)  
  
  
 if best\_silhouette < silhouette\_avg:  
 best\_silhouette = silhouette\_avg  
 best\_cluster = n\_clusters  
 best\_eps = eps  
 best\_metric = metric  
 best\_algorithm = algorithm  
 best\_leaf\_size = leaf\_size  
 best\_min\_sample = min\_sample  
  
 sum = [0 for row in range(n\_clusters)]  
 num = [0 for row in range(n\_clusters)]  
 j = 0  
 for i in cluster\_labels:  
 if i>=0:  
 sum[i] = sum[i] + y[j]  
 num[i] = num[i] + 1  
 j = j + 1  
  
 for i in range(n\_clusters):  
 if num[i]!=0:  
 sum[i] = sum[i] / num[i]  
 centerDF = pd.DataFrame(centers)  
 if len(sum)!=len(centerDF):  
 sum.pop()  
 centerDF.loc[:, 'Mean House Value'] = sum  
  
 sample\_silhouette\_values = silhouette\_samples(X, cluster\_labels)  
  
 y\_lower = 10  
 for i in range(n\_clusters):  
 ith\_cluster\_silhouette\_values = \  
 sample\_silhouette\_values[cluster\_labels == i]  
  
 ith\_cluster\_silhouette\_values.sort()  
  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
  
 color = cm.nipy\_spectral(float(i) / n\_clusters)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_cluster\_silhouette\_values, facecolor=color,  
 edgecolor=color, alpha=0.7)  
  
 ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
  
 y\_lower = y\_upper + 10 *# 10 for the 0 samples* ax1.set\_title("Silouette Plot")  
 ax1.set\_xlabel("Silhouette coefficient")  
 ax1.set\_ylabel("Cluster label")  
  
 ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")  
  
 ax1.set\_yticks([]) *# Clear the yaxis labels / ticks* ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 colors = cm.nipy\_spectral(cluster\_labels.astype(float) / n\_clusters)  
 ax2.scatter(X.iloc[:, 0], X.iloc[:, 1], marker='.', s=30, lw=0, alpha=0.7,  
 c=colors, edgecolor='k')  
  
 ax2.scatter(centers[:, 0], centers[:, 1], marker='o',  
 c="white", alpha=1, s=200, edgecolor='k')  
  
 for i, c in enumerate(centers):  
 ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,  
 s=50, edgecolor='k')  
  
 ax2.set\_title("Cluster")  
 ax2.set\_xlabel("1st Column")  
 ax2.set\_ylabel("2nd Column")  
  
 plt.suptitle(("Silhouette analysis for DBSCAN clustering on sample data "  
 "with n\_clusters = %d" % n\_clusters),  
 fontsize=14, fontweight='bold')  
  
  
 print("---------------------\n",centerDF)  
 plt.show()  
 print("\nThe highest silhouette score is ", best\_silhouette, " with ", best\_cluster, " clusters")  
 print("Best params\_/ eps:", best\_eps, "/ min\_sample:", best\_min\_sample,"/ metric:", best\_metric,"/ algorithm:", best\_algorithm,"/ leaf\_size:", best\_leaf\_size,"\n")  
 param = "Best params\_/ cluster: "+str(best\_cluster)+ "/ eps:"+ str(best\_eps)+ "/ min\_sample:"+ str(best\_min\_sample)+"/ metric:"+ best\_metric+"/ algorithm:"+ best\_algorithm+"/ leaf\_size:"+ str(best\_leaf\_size)  
 return best\_silhouette, param, centerDF  
"""  
:param X: dataset  
:param bandwidths: bandwidth used in the RBF kernel   
:param max\_iters: Maximum numer of iteration  
:param n\_job: The number of jobs to use for the computation.  
"""  
def MeanShifts(X,y,bandwidths=None,max\_iters=None,n\_job=None):  
  
 fig, (ax1, ax2) = plt.subplots(1, 2)  
 fig.set\_size\_inches(18, 7)  
  
 if bandwidths is None:  
 bandwidths = [estimate\_bandwidth(X, quantile=0.75)]  
 if max\_iters is None:  
 max\_iters = [300]  
 if n\_job is None:  
 n\_job = -1  
  
 best\_silhouette = -1  
 best\_cluster = -1  
 best\_max\_iter = 0  
 best\_bandwidth = 0  
  
 centerDF = pd.DataFrame  
  
 for bandwidth in bandwidths:  
 for max\_iter in max\_iters:  
  
 clusterer = MeanShift(bandwidth=bandwidth,max\_iter=max\_iter,n\_jobs=n\_job)  
 clusterer.fit(X)  
 cluster\_labels = clusterer.labels\_  
 n\_clusters = len(clusterer.cluster\_centers\_)  
  
 ax1.set\_xlim([-0.1, 1])  
 ax1.set\_ylim([0, len(X) + (n\_clusters + 1) \* 10])  
  
 silhouette\_avg = silhouette\_score(X, cluster\_labels)  
 print("number of clusters: ", n\_clusters, "/ bandwidth:", bandwidth, "/ max\_iter:", max\_iter)  
 print("The average silhouette\_score is :", silhouette\_avg)  
  
 centers = clusterer.cluster\_centers\_  
  
 if best\_silhouette < silhouette\_avg:  
 best\_silhouette = silhouette\_avg  
 best\_cluster = n\_clusters  
 best\_bandwidth = bandwidth  
 best\_max\_iter = max\_iter  
  
 sum = [0 for row in range(n\_clusters)]  
 num = [0 for row in range(n\_clusters)]  
  
 j = 0  
 for i in cluster\_labels:  
 sum[i] = sum[i] + y[j]  
 num[i] = num[i] + 1  
 j = j + 1  
  
 for i in range(n\_clusters):  
 sum[i] = sum[i] / num[i]  
 centerDF = pd.DataFrame(centers)  
 centerDF.loc[:, 'Mean House Value'] = sum  
  
 sample\_silhouette\_values = silhouette\_samples(X, cluster\_labels)  
  
 y\_lower = 10  
 for i in range(n\_clusters):  
 ith\_cluster\_silhouette\_values = \  
 sample\_silhouette\_values[cluster\_labels == i]  
  
 ith\_cluster\_silhouette\_values.sort()  
  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
  
 color = cm.nipy\_spectral(float(i) / n\_clusters)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_cluster\_silhouette\_values, facecolor=color,  
 edgecolor=color, alpha=0.7)  
  
 ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
  
 y\_lower = y\_upper + 10 *# 10 for the 0 samples* ax1.set\_title("Silouette Plot")  
 ax1.set\_xlabel("Silhouette coefficient")  
 ax1.set\_ylabel("Cluster label")  
  
 ax1.axvline(x=silhouette\_avg, color="red", linestyle="--")  
  
 ax1.set\_yticks([])  
 ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 colors = cm.nipy\_spectral(cluster\_labels.astype(float) / n\_clusters)  
  
 ax2.scatter(X.iloc[:, 0], X.iloc[:, 1], marker='.', s=30, lw=0, alpha=0.7,  
 c=colors, edgecolor='k')  
   
 ax2.scatter(centers[:, 0], centers[:, 1], marker='o',  
 c="white", alpha=1, s=200, edgecolor='k')  
  
 for i, c in enumerate(centers):  
 ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,  
 s=50, edgecolor='k')  
  
 ax2.set\_title("Cluster")  
 ax2.set\_xlabel("1st Column")  
 ax2.set\_ylabel("2nd Column")  
  
  
 plt.suptitle(("Silhouette analysis for MeanShift clustering on sample data "  
 "with n\_clusters = %d" % n\_clusters),  
 fontsize=14, fontweight='bold')  
  
 df = centerDF.copy()  
  
 plt.show()  
 print("\nThe highest silhouette score is ", best\_silhouette, " with ", best\_cluster, " clusters")  
 print("Best params\_/ bandwidth:", best\_bandwidth, "/ max\_iter:", best\_max\_iter,"\n")  
 param = "Best params\_/ bandwidth:"+ str(best\_bandwidth)+ "/ max\_iter:"+ str(best\_max\_iter)  
 return best\_silhouette, param,df  
  
"""  
Do clustering and get silhouette index for various combinations of parameters  
do scale and encode dataset with given scalers and encoders (select features to use for this program)  
do kmeans, GMM, CLARANS, DBSCAN, MeanShift clustering to all combination datasets of encoders and scalers  
do clustering with all combinations of parameters(including 'k' number of clusters)  
show every result (silhouette index) and show parameters with the most highest silhouette index  
"""  
def findBest(or\_data,y,numerical\_columns,categorical\_columns,max\_cluster=None,n\_inits=None,max\_iters=None,tols=None,verboses=None,covariance\_types=None,  
 numlocals=None,max\_neighbors=None,epsS=None,min\_samples=None,metrics=None,algorithms=None,leaf\_sizes=None,bandwidths=None,n\_job=None):  
  
  
  
 kmeans\_best = [-1,'scale','encode','params']  
 GMM\_best = [-1,'scale','encode','params']  
 CLARANS\_best = [-1,'scale','encode','params']  
 DBSCAN\_best = [-1,'scale','encode','params']  
 MeanShift\_best = [-1,'scale','encode','params']  
 silhouette\_score = 0  
 params=""  
 kmeans\_centerDF\_ex = pd.DataFrame()  
 GMM\_centerDF\_ex = pd.DataFrame()  
 DBSCAN\_centerDF\_ex = pd.DataFrame()  
 MeanShift\_centerDF\_ex = pd.DataFrame()  
  
 for numerical\_column,categorical\_column in zip(numerical\_columns,categorical\_columns):  
  
 print("columns: ",numerical\_column, categorical\_column)  
  
 total\_columns = numerical\_column + categorical\_column + ['Mean\_House\_Value']  
 x = pd.DataFrame()  
 data = or\_data.copy()  
  
 for numerical\_column\_ind in numerical\_column:  
 x.loc[:, numerical\_column\_ind] = data.loc[:, numerical\_column\_ind]  
 for categorical\_column\_ind in categorical\_column:  
 x.loc[:, categorical\_column\_ind] = data.loc[:, categorical\_column\_ind]  
  
 x, scalers, encoders = get\_various\_encode\_scale(x, numerical\_column, categorical\_column)  
  
 i = 0  
 for scaler in scalers:  
 j = 0  
 for encoder in encoders:  
 print(scaler, encoder)  
 print("--------Kmeans--------")  
  
 silhouette\_score,params,kmeans\_centerDF\_ex = kmeans(x[i][j],y, max\_cluster=max\_cluster, n\_inits=n\_inits,max\_iters=max\_iters,tols=tols,verboses=verboses)  
 if silhouette\_score > kmeans\_best[0]:  
 kmeans\_best[0] = silhouette\_score  
 kmeans\_best[1] = scaler  
 kmeans\_best[2] = encoder  
 kmeans\_best[3] = params  
 kmeans\_centerDF = kmeans\_centerDF\_ex.copy()  
 print("--------GMM--------")  
  
 silhouette\_score, params, GMM\_centerDF\_ex = GMM(x[i][j],y, max\_cluster=max\_cluster, covariance\_types=covariance\_types,tols=tols,max\_iters=max\_iters,n\_inits=n\_inits)  
 if silhouette\_score > GMM\_best[0]:  
 GMM\_best[0] = silhouette\_score  
 GMM\_best[1] = scaler  
 GMM\_best[2] = encoder  
 GMM\_best[3] = params  
 GMM\_centerDF = GMM\_centerDF\_ex.copy()  
 print("--------CLARANS--------")  
  
 silhouette\_score, params = clarans(x[i][j],y, max\_cluster=max\_cluster,numlocals=numlocals,maxneighbors=max\_neighbors)  
 if silhouette\_score > CLARANS\_best[0]:  
 CLARANS\_best[0] = silhouette\_score  
 CLARANS\_best[1] = scaler  
 CLARANS\_best[2] = encoder  
 CLARANS\_best[3] = params  
 print("--------DBSCAN--------")  
  
 silhouette\_score, params , DBSCAN\_centerDF\_ex= DBSCANs(x[i][j],y,epsS=epsS,min\_samples=min\_samples,metrics=metrics,algorithms=algorithms,leaf\_sizes=leaf\_sizes)  
 if silhouette\_score > DBSCAN\_best[0]:  
 DBSCAN\_best[0] = silhouette\_score  
 DBSCAN\_best[1] = scaler  
 DBSCAN\_best[2] = encoder  
 DBSCAN\_best[3] = params  
 DBSCAN\_centerDF = DBSCAN\_centerDF\_ex.copy()  
 print("--------MeanShift--------")  
  
 silhouette\_score, params,MeanShift\_centerDF\_ex =MeanShifts(x[i][j],y,bandwidths=bandwidths,max\_iters=max\_iters,n\_job=n\_job)  
 if silhouette\_score > MeanShift\_best[0]:  
 MeanShift\_best[0] = silhouette\_score  
 MeanShift\_best[1] = scaler  
 MeanShift\_best[2] = encoder  
 MeanShift\_best[3] = params  
 MeanShift\_centerDF = MeanShift\_centerDF\_ex.copy()  
 j = j + 1  
 i = i + 1  
  
 kmeans\_centerDF.columns = total\_columns  
 GMM\_centerDF.columns = total\_columns  
 DBSCAN\_centerDF.columns = total\_columns  
 MeanShift\_centerDF.columns = total\_columns  
  
 return kmeans\_best, GMM\_best, CLARANS\_best, DBSCAN\_best, MeanShift\_best, kmeans\_centerDF, GMM\_centerDF, DBSCAN\_centerDF, MeanShift\_centerDF