Clustering Task - The California Housing Prices Dataset

Before run this manual, please make sure the install and import following packages.

!pip install pyclustering

**import** **random**

**import** **warnings**

warnings.filterwarnings('ignore')

**import** **pandas** **as** **pd**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **numpy** **as** **np**

**from** **sklearn.preprocessing** **import** OneHotEncoder

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **sklearn.preprocessing** **import** StandardScaler,MaxAbsScaler,MinMaxScaler,RobustScaler

**from** **sklearn.cluster** **import** KMeans

**from** **sklearn.cluster** **import** DBSCAN

**from** **sklearn.cluster** **import** MeanShift

**from** **sklearn.cluster** **import** estimate\_bandwidth

**from** **sklearn.mixture** **import** GaussianMixture

**from** **sklearn.metrics** **import** silhouette\_score

**from** **sklearn.metrics** **import** silhouette\_samples

**from** **sklearn.model\_selection** **import** GridSearchCV

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** \*

**from** **pyclustering.cluster.clarans** **import** clarans;

**from** **pyclustering.utils** **import** timedcall;

**from** **sklearn** **import** datasets

**from** **pyclustering.cluster** **import** cluster\_visualizer\_multidim

**Loading a Dataset**

*#Loading a dataset*

df = pd.read\_csv('/content/drive/MyDrive/housing.csv', delimiter = ",")

df\_original = df.copy()

print(df.shape)

print(df.isnull().sum())

테이블이(가) 표시된 사진

자동 생성된 설명

out:

**Visualize it with a heat map.**

housing\_corr\_matrix = df.corr()

*#set the matplotlib figure*

fig, axe = plt.subplots(figsize=(12,8))

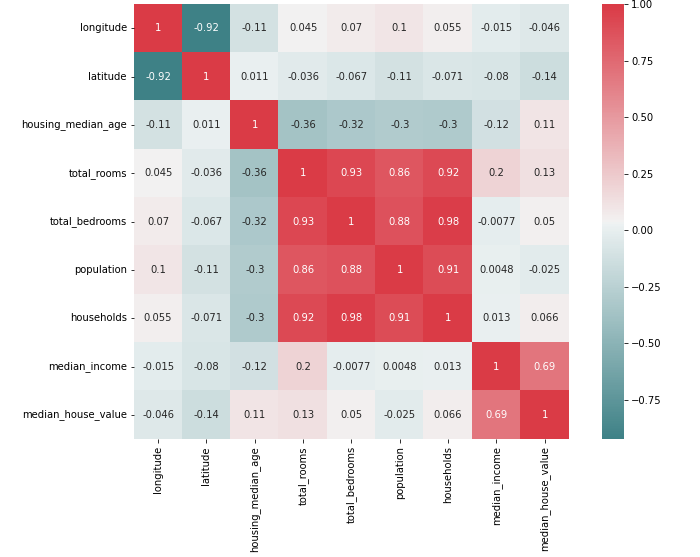
*#Generate color palettes*

cmap = sns.diverging\_palette(200, 10, center = "light", as\_cmap=**True**)

*#draw the heatmap*

sns.heatmap(housing\_corr\_matrix, vmax=1, square =**True**, cmap=cmap, annot=**True** )

plt.show()

out: 

**Setting up the combinations.**

You can freely add or delete the elements you want.

encoders = [LabelEncoder(), OneHotEncoder()]

scalers = [StandardScaler(), MinMaxScaler(), MaxAbsScaler(), RobustScaler()]

models = ['K\_Means','MeanShift','CLARANS','DBSCAN','GMM']

hyperparams = {

*#'K\_Means\_params':{}*

*#'GMM\_params':{}*

*#'CLARANS\_params':{}*

'DBSCAN\_params': {

'eps': [0.005, 0.01]

'min\_samples': [10, 20]

},

'MeanShift\_params': {

'n': [10, 50, 100]

},

'k': range(2, 13)

}

**Various combinations of the features**

We selected two features randomly and conducted the experiment.

combi = []

combi.append(['longitude', 'latitude'])

combi.append(['total\_rooms', 'total\_bedrooms'])

combi.append(['population','households'])

**Clean and prepare a dataset**

**Function**

preprocessing(df)

**Parameters**

* **df :** dataset

**Source code**

**'''**

< preprocessing >

Input : df

- Remove needless features.(median\_house\_value)

- fill the missing values

Output : modified dataframe

'''

**def** preprocessing(df):

df.drop(columns=["median\_house\_value"], inplace=**True**)

df.total\_bedrooms.fillna(df.total\_bedrooms.median(), inplace=**True**)

**return** df

**Utils**

Implement silhouette score function and elbow curve function

**def** cv\_silhouette\_scorer(estimator, X):

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)

**def** elbow\_curve(distortions):

fig = plt.figure(figsize=(15, 5))

plt.plot(range(2, 13), distortions)

plt.grid(**True**)

plt.title('Elbow curve')

plt.show()

**Clustering**

We defined functions to cluster automatically with computing all combination of parameters that specified scaler, models and hyperparameters.

**Function**

clustering(df, models, hyperparams)

**Parameters**

* **df :** dataset
* **models** : list of models

*['K\_Means',' MeanShift', 'CLARANS', 'DBSCAN', 'GMM']*

* **hyperparams** : list of models’ hyperparameters

*hyperparams = {*

*'DBSCAN\_params': { 'eps': [0.01, 0.003] },*

*'MeanShift\_params': { 'n': [10, 50, 100] },*

*'k': range(2, 13)*

*}*

**Source code**

**'''**

< clustering >

Input : df(dataframe), models(list), hyperparams(dict)

- clustering with various models and hyperparameter values.

- plotting

Output : plot results, silhouette Score, Quantile comparison score

'''

**def** clustering(df, y, models, hyperparams):

*# Experiment with various models*

**for** model **in** models:

**print**("Current model: ", model)

*# Apply various hyperparameters in each models*

**if** model == 'K\_Means':

distortions = []

**for** k **in** hyperparams['k']:

kmeans = KMeans(n\_clusters=k, init='k-means++')

cluster = kmeans.fit(df)

labels = kmeans.predict(df)

cluster\_id = pd.DataFrame(cluster.labels\_)

distortions.append(kmeans.inertia\_)

d1 = pd.concat([df, cluster\_id], axis=1)

d1.columns = [0, 1, "cluster"]

sns.scatterplot(d1[0], d1[1], hue=d1['cluster'], legend="full")

sns.scatterplot(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], label='Centroids')

plt.title("KMeans Clustering")

plt.legend()

*#plt.show()*

**print**('Silhouette Score(euclidean):', metrics.silhouette\_score(df, labels, metric='euclidean'), " ", k, "-clusters)")

**print**('Silhouette Score(manhattan):', metrics.silhouette\_score(df, labels, metric='manhattan'))

*#Compare the clustering results with N (where 2<= N <= 10) quantiles of the medianHouseValue feature values in the original dataset.*

**print**('Quantile comparison score(purity\_score):', purity\_score(y, labels))

elbow\_curve(distortions)

**elif** model == 'GMM':

**for** k **in** hyperparams['k']:

gmm = GaussianMixture(n\_components=k)

gmm.fit(df)

labels = gmm.predict(df)

frame = pd.DataFrame(df)

frame['cluster'] = labels

frame.columns = [df.columns[0], df.columns[1], 'cluster']

**for** i **in** range(0, k + 1):

data = frame[frame["cluster"] == i]

plt.scatter(data[data.columns[0]], data[data.columns[1]])

plt.show()

**print**('Silhouette Score(euclidean):', metrics.silhouette\_score(df, labels, metric='euclidean'), " (", k, "-components)")

**print**('Silhouette Score(manhattan):', metrics.silhouette\_score(df, labels, metric='manhattan'))

*#Compare the clustering results with N (where 2<= N <= 10) quantiles of the medianHouseValue feature values in the original dataset.*

**print**('Quantile comparison score(purity\_score):', purity\_score(y, labels))

**elif** model == 'CLARANS':

data = df.values.tolist()

**for** k **in** hyperparams['k']:

cl\_data = random.sample(data, 250)

clarans\_obj = clarans(cl\_data, k, 3, 5)

(tks, res) = timedcall(clarans\_obj.process)

clst = clarans\_obj.get\_clusters()

med = clarans\_obj.get\_medoids()

*#print("Index of clusters' points :\n", clst)*

*#print("\nIndex of the best medoids : ", med)*

labels = pd.DataFrame(clst).T.melt(var\_name='clusters').dropna()

labels['value'] = labels.value.astype(int)

labels = labels.sort\_values(['value']).set\_index('value').values.flatten()

vis = cluster\_visualizer\_multidim()

vis.append\_clusters(clst, cl\_data, marker="\*", markersize=5)

vis.show(max\_row\_size=3)

**print**('Silhouette Score(euclidean):', metrics.silhouette\_score(cl\_data, labels, metric='euclidean'), " (", k, "-clusters)")

**print**('Silhouette Score(manhattan):', metrics.silhouette\_score(cl\_data, labels, metric='manhattan'))

**elif** model == 'DBSCAN':

eps = hyperparams['DBSCAN\_params']['eps']

minsam = hyperparams['DBSCAN\_params']['min\_samples']

**for** i **in** eps:

**for** j **in** minsam:

db = DBSCAN(eps=i, min\_samples=j)

cluster = db.fit(df)

cluster\_id = pd.DataFrame(cluster.labels\_)

d2 = pd.DataFrame()

d2 = pd.concat([df, cluster\_id], axis=1)

d2.columns = [0, 1, "cluster"]

sns.scatterplot(d2[0], d2[1], hue=d2['cluster'], legend="full")

plt.title('DBSCAN with eps {}'.format(i))

plt.show()

**print**('Silhouette Score(euclidean):', metrics.silhouette\_score(d2.iloc[:, :-1], d2['cluster'], metric='euclidean'), " (eps=", i, ")", " (min\_samples=", j, ")")

**print**('Silhouette Score(manhattan):', metrics.silhouette\_score(d2.iloc[:, :-1], d2['cluster'], metric='manhattan'))

**elif** model == 'MeanShift':

n = hyperparams['MeanShift\_params']['n']

**for** i **in** n:

bandwidth = estimate\_bandwidth(df, quantile=0.2, n\_samples=i)

ms = MeanShift(bandwidth=bandwidth)

cluster = ms.fit(df)

cluster\_id = pd.DataFrame(cluster.labels\_)

d6 = pd.DataFrame()

d6 = pd.concat([df, cluster\_id], axis=1)

d6.columns = [0, 1, "cluster"]

sns.scatterplot(d6[0], d6[1], hue=d6['cluster'], legend="full")

plt.title('Mean Shift with {} samples'.format(i))

plt.show()

**print**('n\_samples(estimate\_bandwidth) = {}'.format(i))

**print**('Silhouette Coefficient(euclidean): ',metrics.silhouette\_score(d6.iloc[:, :-1], d6['cluster'], metric='euclidean'))

**print**('Silhouette Coefficient(manhattan): ',metrics.silhouette\_score(d6.iloc[:, :-1], d6['cluster'], metric='manhattan'))

**Main**

All of these processes are executed automatically by calling the main function.

'''

< main >

INPUT : df(dataframe), scalers(list), models(list), hyperparams(dict), combi(list)

- preprocessing

- scaling

- Call 'clustering()' function to cluster and plot

'''

**def** main(df, scalers, models, hyperparams, combi):

new\_df = preprocessing(df)

**for** i **in** combi:

X = new\_df[i]

print("Current combination", i)

**for** scaler **in** scalers:

print("Current scaler:", scaler)

scaled\_X = scaler.fit\_transform(X)

data\_df = pd.DataFrame(scaled\_X)

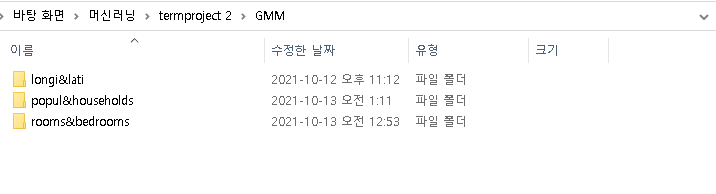
clustering(data\_df, models, hyperparams)

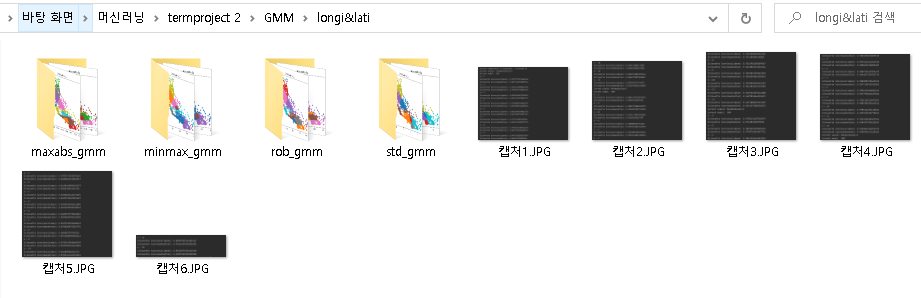
main(df, scalers, models, hyperparams, combi)

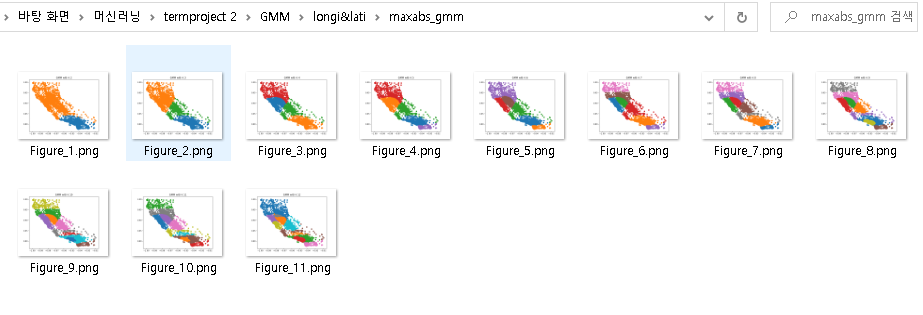
**Result**

Due to the large number of results, only a few representative plots were attached.

ex)



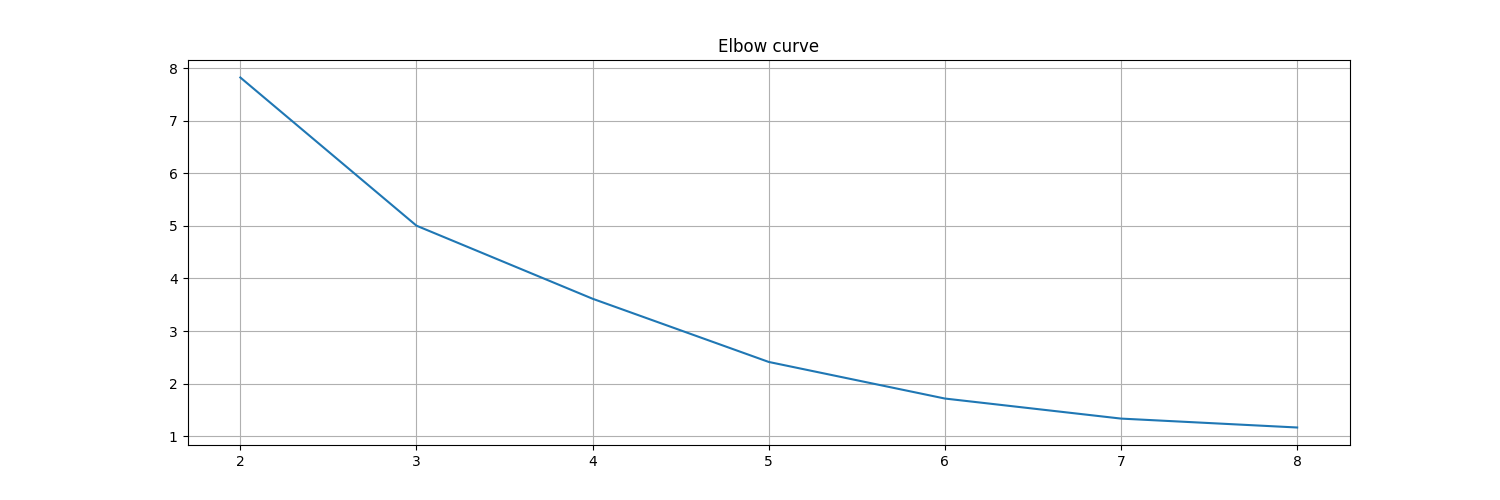




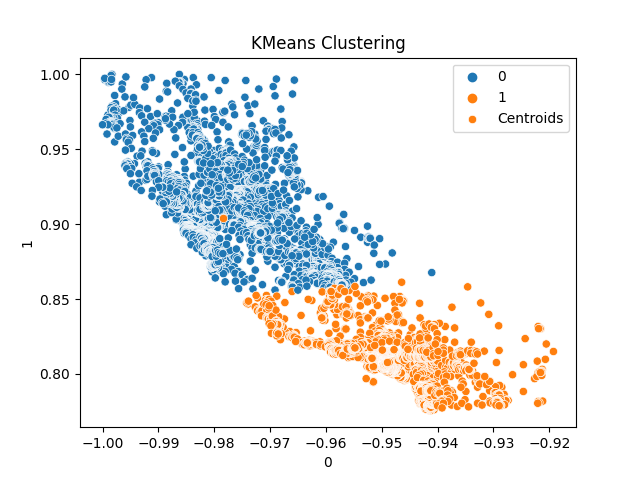
**['longitude', 'latitude']**

**K-means**

Best combination: MaxAbsScaler, k=2

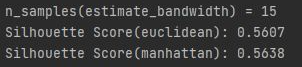


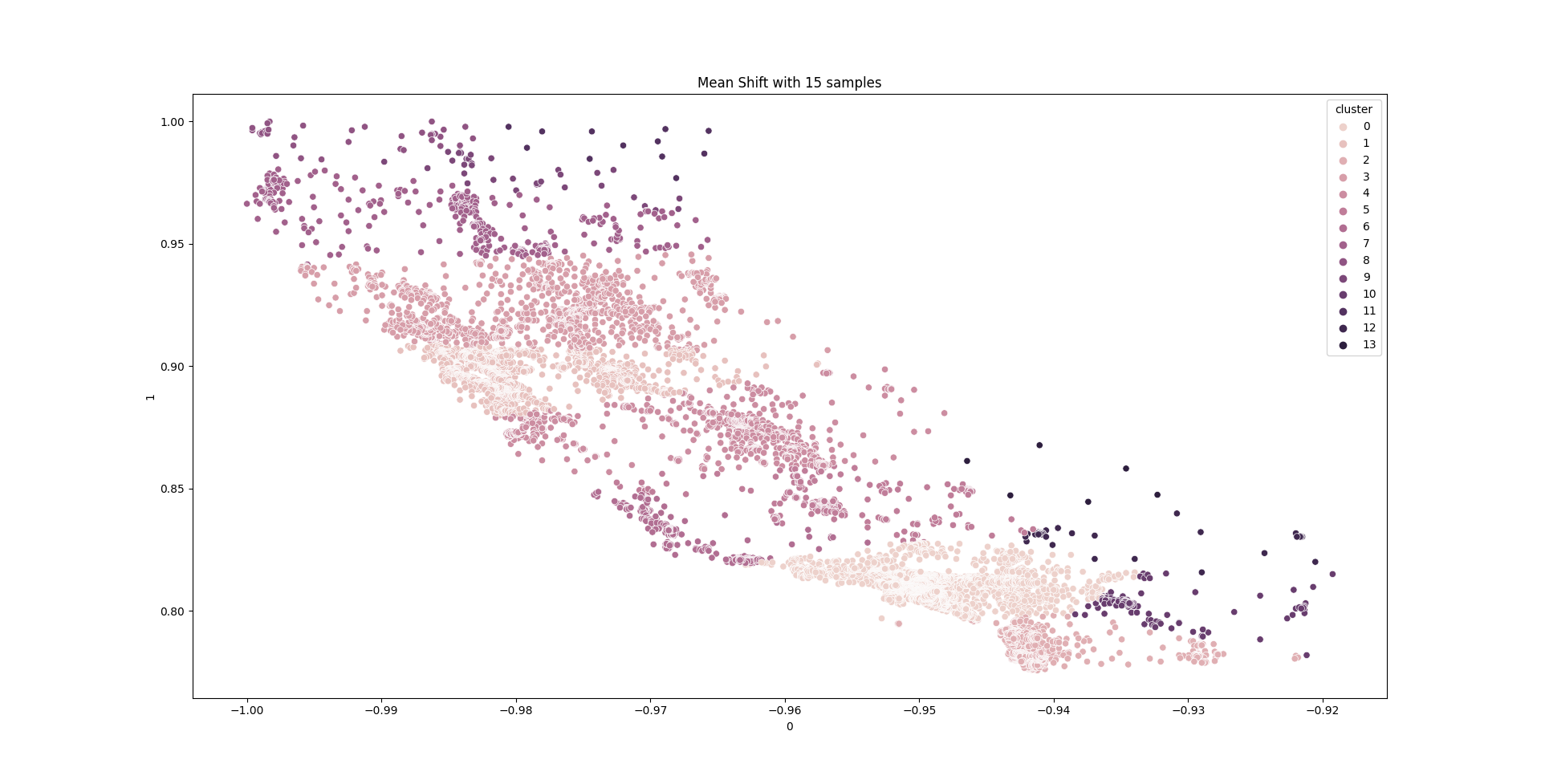
****

****

**MeanShift**

Best combination: MaxAbsScaler, n\_samples=15

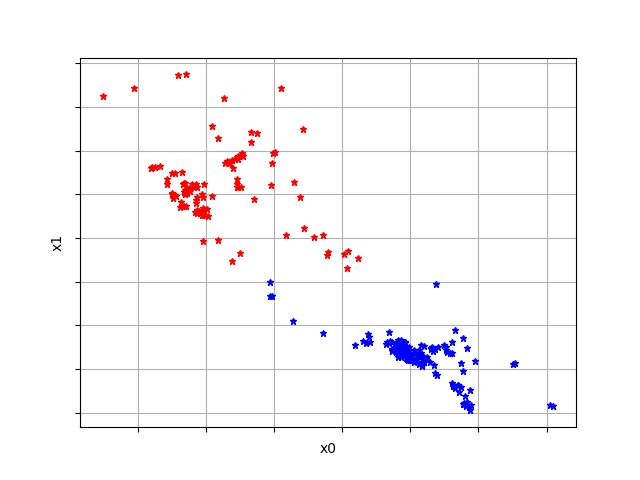




**CLARANS**

Best combination: MaxAbsScaler, k=2

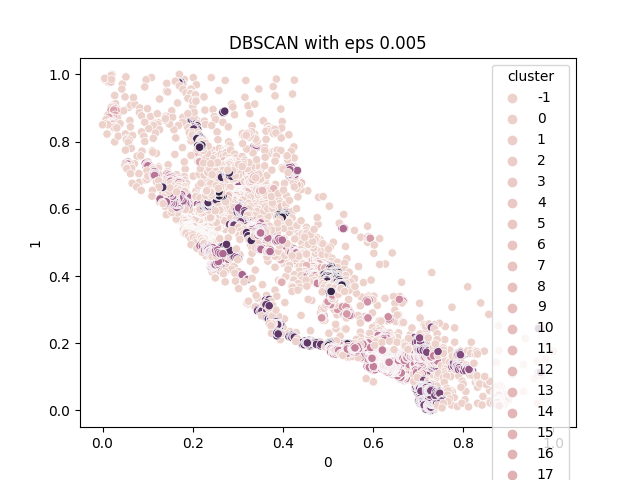




**DBSCAN**

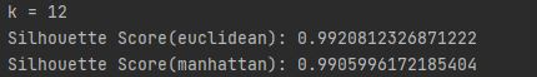
Best combination: MinMaxScaler, eps=0.005, min\_samples=5

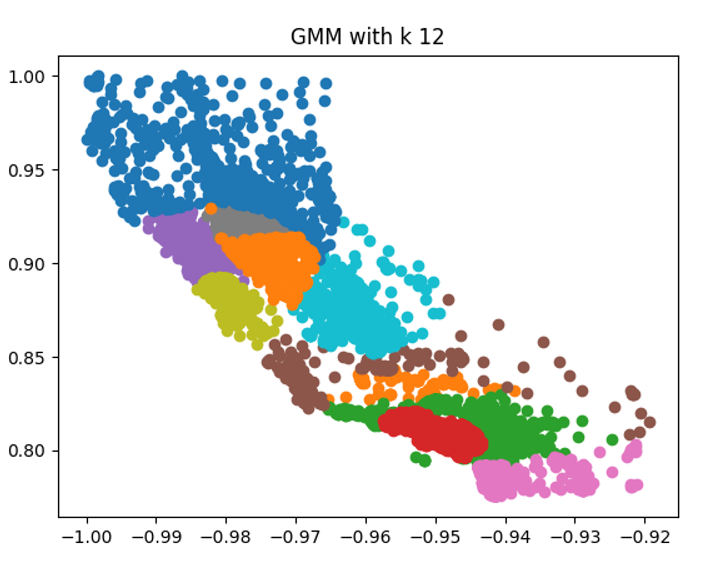




**GMM**

Best combination: MaxAbsScaler, k=12

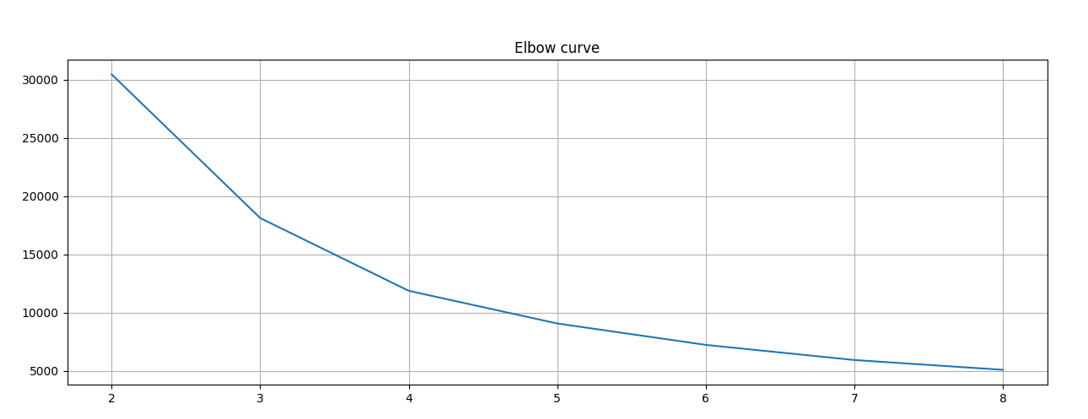


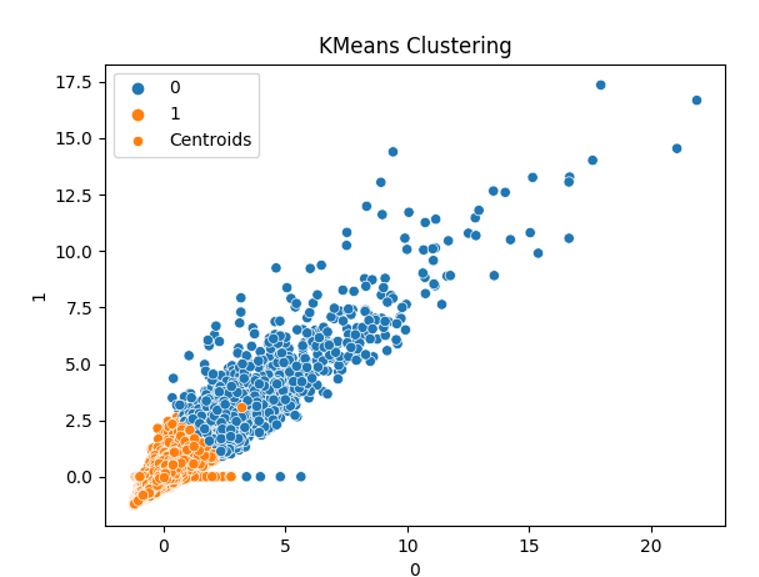
****

**['total\_rooms', 'total\_bedrooms']**

**K-means**

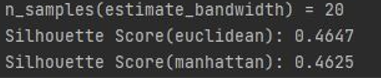
Best combination: RobustScaler, k=2

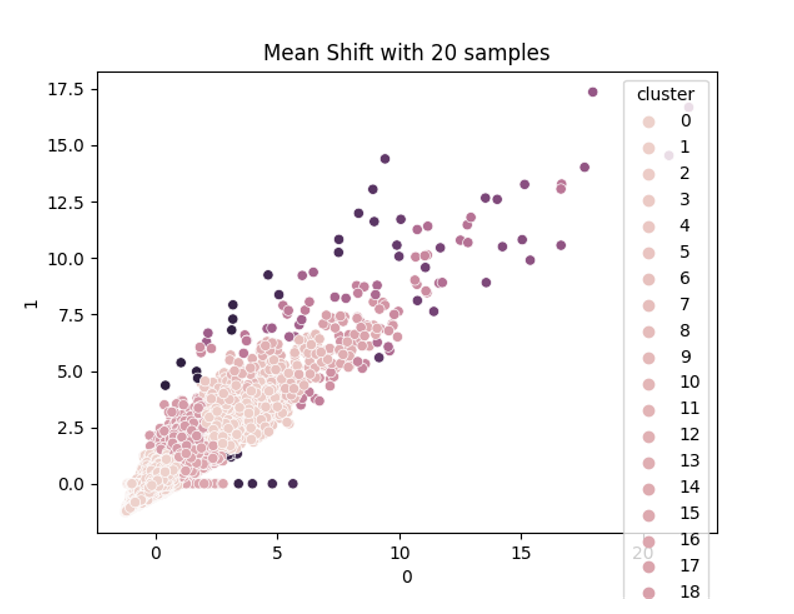




**MeanShift**

Best combination: RobustScaler, n\_samples=20,

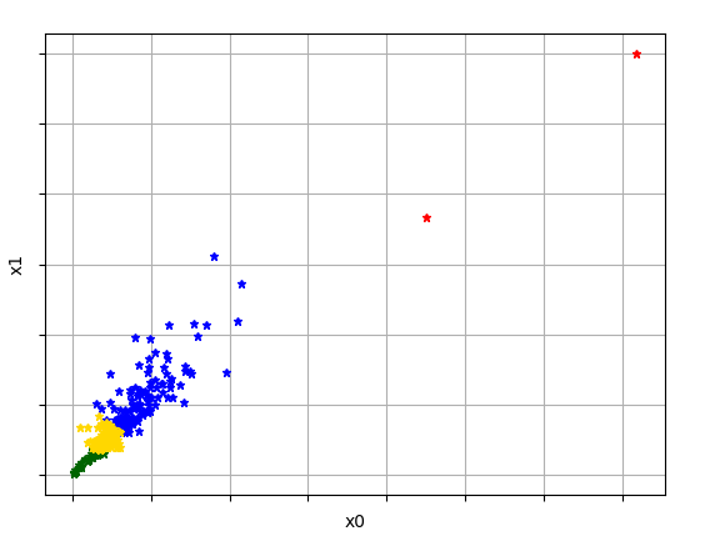




**CLARANS**

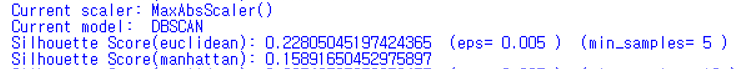
Best combination: MinMaxScaler, k=3

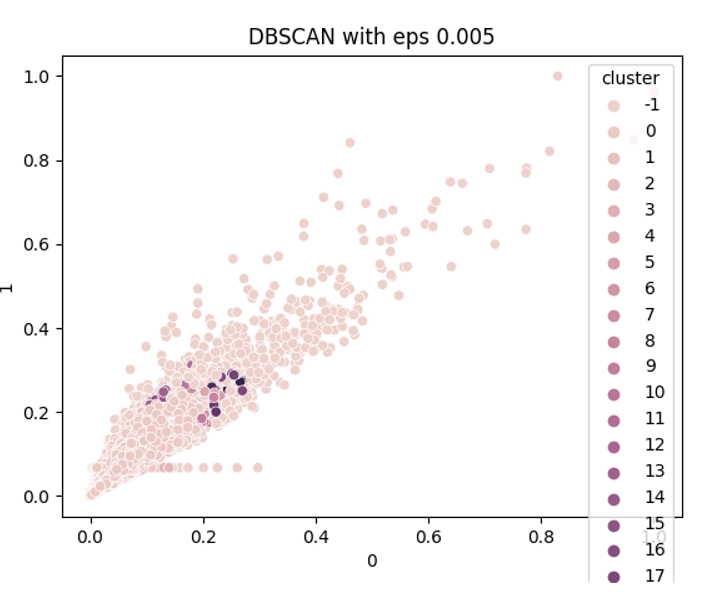




**DBSCAN**

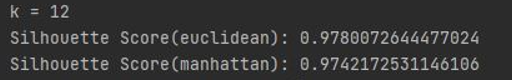
Best combination: MaxAbsScaler, eps=0.005, min\_samples=5

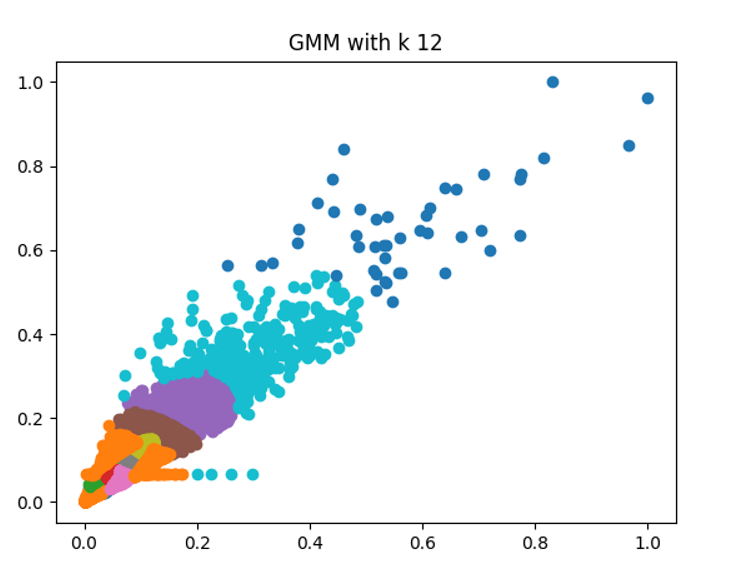




**GMM**

Best combination: MinMaxScaler, k=12

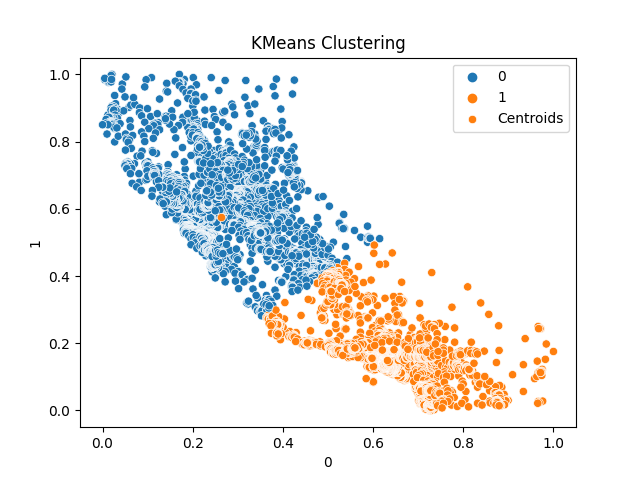
****

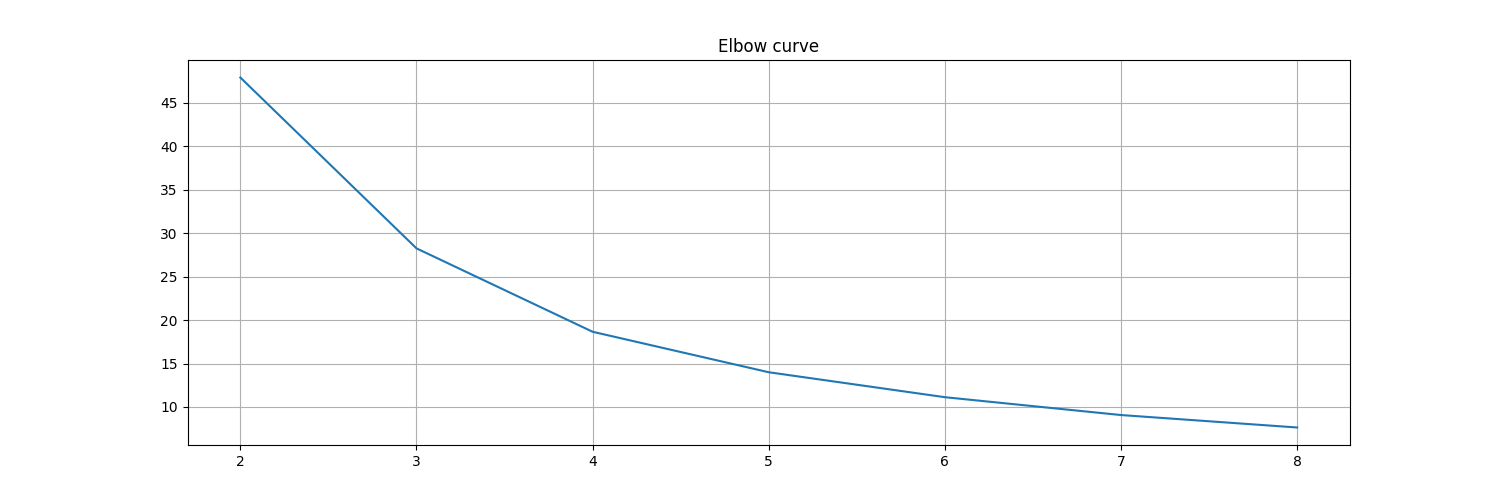
****

**['population', 'households']**

**K-means**

Best combination: MinMaxScaler, k=2

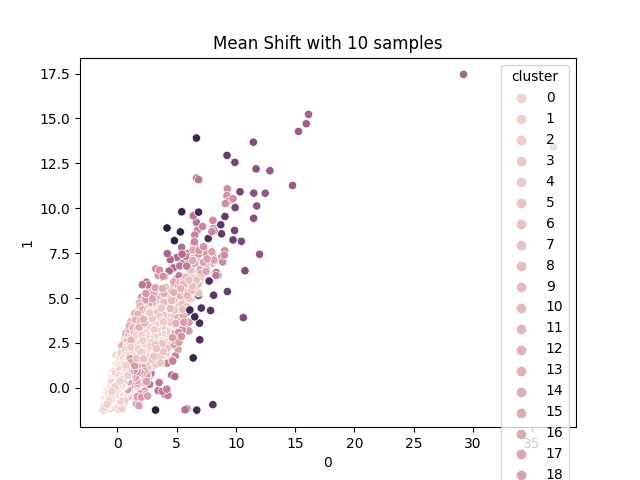
****

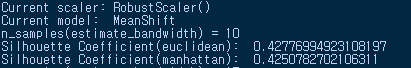
****

****

**MeanShift**

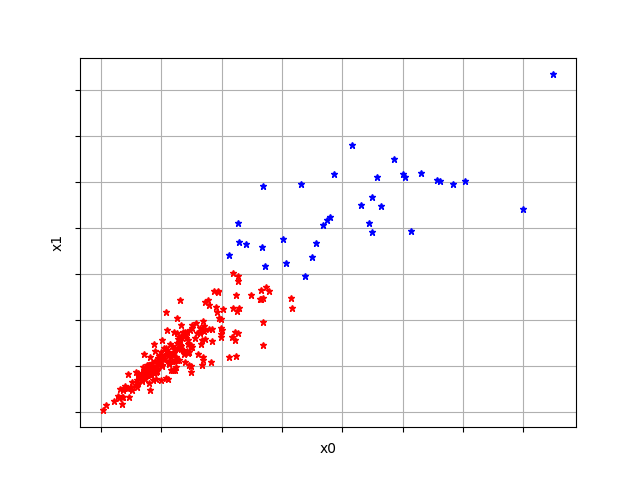
Best combination: RobustScaler, n\_samples = 10

****

****

**CLARANS**

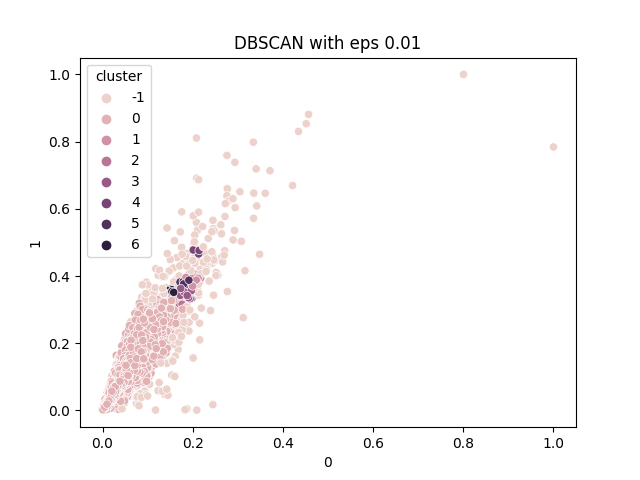
Best combination: MaxAbsScaler, k = 2

****

****

**DBSCAN**

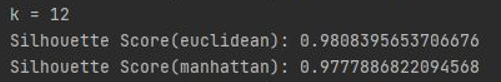
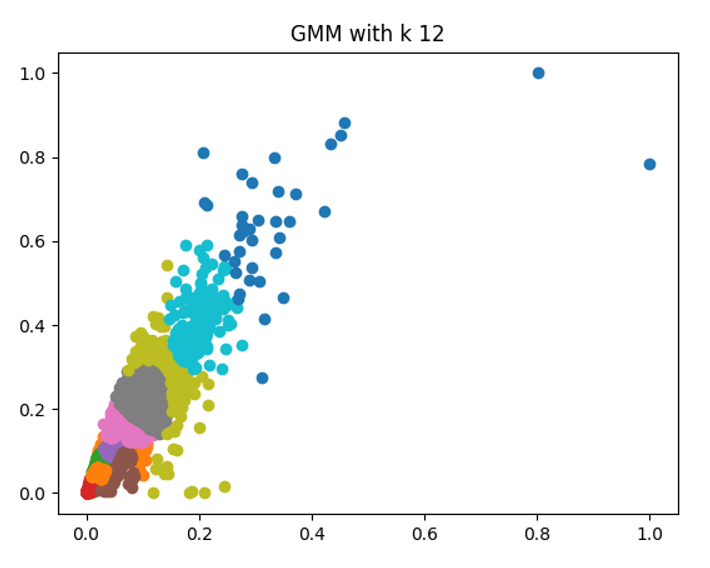
Best combination: MaxAbsScaler, eps = 0.01, min\_samples = 5

****

****

**GMM**

Best combination: MaxAbsScaler, k=12



**BEST OF ALL – [longitude, latitude], MaxAbs, GMM, k=12**

**BEST SCORE = 0.992…**

**Compare the clustering results**

**Compare the results with N quantiles of the medianHouseValue feature values in the original dataset. In this case, we compared with N=4, N=5,and N=8.**

**Source code**

df = pd.read\_csv('housing.csv', delimiter=",")

df\_original = df.copy()

df=df.sort\_values(by=['median\_house\_value'], axis=0)

median\_house\_value=df['median\_house\_value']

n=4

k=1

tmp=0

median\_house\_value\_q=[]

total\_rooms\_q=[]

total\_bedrooms\_q=[]

longitude\_q=[]

latitude\_q=[]

household\_q=[]

population\_q=[]

**while** **True**:

tmp=(1/n)\*k

**if** tmp>1:

**break**

print(tmp)

q\_v=df.quantile(tmp)

print(q\_v)

median\_house\_value\_q.append(q\_v['median\_house\_value'])

total\_rooms\_q.append(q\_v['total\_rooms'])

total\_bedrooms\_q.append(q\_v['total\_bedrooms'])

longitude\_q.append(q\_v['longitude'])

latitude\_q.append(q\_v['latitude'])

household\_q.append(q\_v['households'])

population\_q.append(q\_v['population'])

k=k+1

sns.scatterplot(df['population'], df['households'], hue=median\_house\_value)

sns.scatterplot(population\_q, household\_q, hue=median\_house\_value\_q,palette=['red' **for** i **in** range(0,n)])

plt.show()

sns.scatterplot(df['total\_rooms'], df['total\_bedrooms'], hue=median\_house\_value)

sns.scatterplot(total\_rooms\_q, total\_bedrooms\_q, hue=median\_house\_value\_q,palette=['red' **for** i **in** range(0,n)])

plt.show()

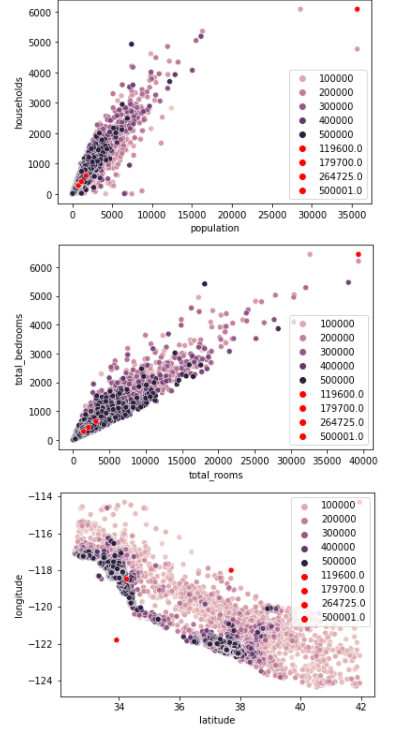
sns.scatterplot(df['latitude'], df['longitude'], hue=median\_house\_value)

sns.scatterplot(latitude\_q, longitude\_q, hue=median\_house\_value\_q,palette=['red' **for** i **in** range(0,n)])

plt.show()

**N=4**

4 quantiles of the median House Value feature values are 119600(25%), 179700(50%), 264725(75%), 500001(100%).



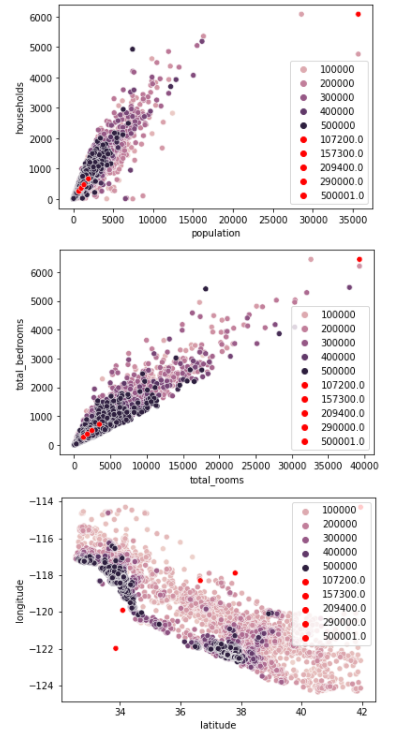
Best purity score(with n=4): **0.327...**

Best purity combination: [longitude, latitude], RobustScaler, GMM, k=7

****

**N=5**

5 quantiles of the median House Value feature values are 107200(20%), 157300(40%), 209400(60%), 290000(80%), 500001(100%)



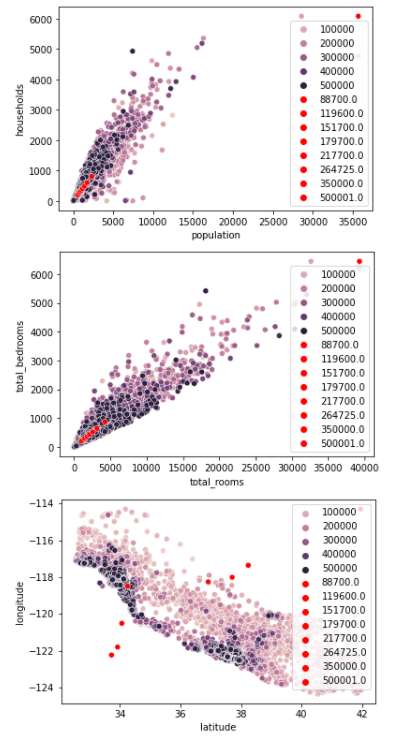
Best purity score(with n=5): **0.292...**

Best purity combination: [longitude, latitude], StandardScaler, GMM, k=7

****

**N=8**

8 quantiles of the median House Value feature values are 88700(12.5%), 119600(25%), 151700(37.5%), 179700(50%), 217700(62.5%), 264725(75%),350000(87.5%), 500001(100%)

****

Best purity score(with n=5): **0.2078...**

Best purity combination: [longitude, latitude], StandardScaler, K\_Means, k=8



**Total source code**

**import random**

**import warnings**

**warnings.filterwarnings('ignore')**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import numpy as np**

**from sklearn.preprocessing import OneHotEncoder**

**from sklearn.preprocessing import LabelEncoder**

**from sklearn.preprocessing import StandardScaler,MaxAbsScaler,MinMaxScaler,RobustScaler**

**from sklearn.cluster import KMeans**

**from sklearn.cluster import DBSCAN**

**from sklearn.cluster import MeanShift**

**from sklearn.cluster import estimate\_bandwidth**

***#from pyclustering.cluster.clarans import clarans;***

**from sklearn.mixture import GaussianMixture**

**from sklearn.metrics import silhouette\_score**

**from sklearn.metrics import silhouette\_samples**

**from sklearn.model\_selection import GridSearchCV**

**from sklearn import metrics**

**from sklearn.metrics import \***

**from pyclustering.cluster.clarans import clarans;**

**from pyclustering.utils import timedcall;**

**from sklearn import datasets**

**from pyclustering.cluster import cluster\_visualizer\_multidim**

**df = pd.read\_csv('D:/housing.csv', delimiter=",")**

**df\_original = df.copy()**

**print(df.shape)**

**print(df.isnull().sum())**

**housing\_corr\_matrix = df.corr()**

***# set the matplotlib figure***

**fig, axe = plt.subplots(figsize=(12, 8))**

***# Generate color palettes***

**cmap = sns.diverging\_palette(200, 10, center="light", as\_cmap=True)**

***# draw the heatmap***

**sns.heatmap(housing\_corr\_matrix, vmax=1, square=True, cmap=cmap, annot=True)**

**plt.show()**

**encoders = [LabelEncoder(), OneHotEncoder()]**

**scalers = [StandardScaler(), MinMaxScaler(), MaxAbsScaler(), RobustScaler()]**

**models = ['K\_Means','DBSCAN','CLARANS','MeanShift','GMM']**

**hyperparams = {**

***# 'K\_Means\_params':{}***

***# 'GMM\_params':{}***

***# 'CLARANS\_params':{}***

**'DBSCAN\_params': {**

**'eps': [0.005, 0.01],**

**'min\_samples':[5, 10]**

***# 'eps':[0.1, 0.2, 0.3, 0.4, 0.5]***

**},**

**'MeanShift\_params': {**

**'n': [10, 15, 20]**

**},**

**'k': range(2, 9)**

**}**

**def preprocessing(df):**

**df.total\_bedrooms.fillna(df.total\_bedrooms.median(), inplace=True)**

**return df**

**def main(df, y, scalers, models, hyperparams, combi):**

**new\_df = preprocessing(df)**

**for i in combi:**

**X = new\_df[i]**

**print("Current combination", i)**

**for scaler in scalers:**

**print("Current scaler:", scaler)**

**scaled\_X = scaler.fit\_transform(X)**

**data\_df = pd.DataFrame(scaled\_X)**

**clustering(data\_df, y, models, hyperparams)**

**def elbow\_curve(distortions):**

**fig = plt.figure(figsize=(15, 5))**

**plt.plot(range(2, 9), distortions)**

**plt.grid(True)**

**plt.title('Elbow curve')**

***#plt.show()***

**def purity\_score(y\_true, y\_pred):**

***# compute contingency matrix (also called confusion 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)**

**def clustering(df, y, models, hyperparams):**

***# Experiment with various models***

**for model in models:**

**print("Current model: ", model)**

***# Apply various hyperparameters in each models***

**if model == 'K\_Means':**

**distortions = []**

**for k in hyperparams['k']:**

**kmeans = KMeans(n\_clusters=k, init='k-means++')**

**cluster = kmeans.fit(df)**

**labels = kmeans.predict(df)**

**cluster\_id = pd.DataFrame(cluster.labels\_)**

**distortions.append(kmeans.inertia\_)**

**d1 = pd.concat([df, cluster\_id], axis=1)**

**d1.columns = [0, 1, "cluster"]**

**sns.scatterplot(d1[0], d1[1], hue=d1['cluster'], legend="full")**

**sns.scatterplot(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], label='Centroids')**

**plt.title("KMeans Clustering")**

**plt.legend()**

**plt.show()**

**print('Silhouette Score(euclidean):', metrics.silhouette\_score(df, labels, metric='euclidean'), " ", k, "-clusters)")**

**print('Silhouette Score(manhattan):', metrics.silhouette\_score(df, labels, metric='manhattan'))**

**print('Quantile comparison score(purity\_score):', purity\_score(y, labels))**

**elbow\_curve(distortions)**

**elif model == 'GMM':**

**for k in hyperparams['k']:**

**gmm = GaussianMixture(n\_components=k)**

**gmm.fit(df)**

**labels = gmm.predict(df)**

**frame = pd.DataFrame(df)**

**frame['cluster'] = labels**

**frame.columns = [df.columns[0], df.columns[1], 'cluster']**

**for i in range(0, k + 1):**

**data = frame[frame["cluster"] == i]**

**plt.scatter(data[data.columns[0]], data[data.columns[1]])**

**plt.show()**

**print('Silhouette Score(euclidean):', metrics.silhouette\_score(df, labels, metric='euclidean'), " (", k, "-components)")**

**print('Silhouette Score(manhattan):', metrics.silhouette\_score(df, labels, metric='manhattan'))**

**print('Quantile comparison score(purity\_score):', purity\_score(y, labels))**

**elif model == 'CLARANS':**

**data = df.values.tolist()**

**for k in hyperparams['k']:**

**cl\_data = random.sample(data, 250)**

**clarans\_obj = clarans(cl\_data, k, 3, 5)**

**(tks, res) = timedcall(clarans\_obj.process)**

**clst = clarans\_obj.get\_clusters()**

**med = clarans\_obj.get\_medoids()**

***#print("Index of clusters' points :\n", clst)***

***#print("\nIndex of the best medoids : ", med)***

**labels = pd.DataFrame(clst).T.melt(var\_name='clusters').dropna()**

**labels['value'] = labels.value.astype(int)**

**labels = labels.sort\_values(['value']).set\_index('value').values.flatten()**

**vis = cluster\_visualizer\_multidim()**

**vis.append\_clusters(clst, cl\_data, marker="\*", markersize=5)**

**vis.show(max\_row\_size=3)**

**print('Silhouette Score(euclidean):', metrics.silhouette\_score(cl\_data, labels, metric='euclidean'), " (", k, "-clusters)")**

**print('Silhouette Score(manhattan):', metrics.silhouette\_score(cl\_data, labels, metric='manhattan'))**

**elif model == 'DBSCAN':**

**eps = hyperparams['DBSCAN\_params']['eps']**

**minsam = hyperparams['DBSCAN\_params']['min\_samples']**

**for i in eps:**

**for j in minsam:**

**db = DBSCAN(eps=i, min\_samples=j)**

**cluster = db.fit(df)**

**cluster\_id = pd.DataFrame(cluster.labels\_)**

**d2 = pd.DataFrame()**

**d2 = pd.concat([df, cluster\_id], axis=1)**

**d2.columns = [0, 1, "cluster"]**

**sns.scatterplot(d2[0], d2[1], hue=d2['cluster'], legend="full")**

**plt.title('DBSCAN with eps {}'.format(i))**

**plt.show()**

**print('Silhouette Score(euclidean):', metrics.silhouette\_score(d2.iloc[:, :-1], d2['cluster'], metric='euclidean'), " (eps=", i, ")", " (min\_samples=", j, ")")**

**print('Silhouette Score(manhattan):', metrics.silhouette\_score(d2.iloc[:, :-1], d2['cluster'], metric='manhattan'))**

**elif model == 'MeanShift':**

**n = hyperparams['MeanShift\_params']['n']**

**for i in n:**

**bandwidth = estimate\_bandwidth(df, quantile=0.2, n\_samples=i)**

**ms = MeanShift(bandwidth=bandwidth)**

**cluster = ms.fit(df)**

**cluster\_id = pd.DataFrame(cluster.labels\_)**

**d6 = pd.DataFrame()**

**d6 = pd.concat([df, cluster\_id], axis=1)**

**d6.columns = [0, 1, "cluster"]**

**sns.scatterplot(d6[0], d6[1], hue=d6['cluster'], legend="full")**

**plt.title('Mean Shift with {} samples'.format(i))**

**plt.show()**

**print('n\_samples(estimate\_bandwidth) = {}'.format(i))**

**print('Silhouette Coefficient(euclidean): ',metrics.silhouette\_score(d6.iloc[:, :-1], d6['cluster'], metric='euclidean'))**

**print('Silhouette Coefficient(manhattan): ',metrics.silhouette\_score(d6.iloc[:, :-1], d6['cluster'], metric='manhattan'))**

**combi = []**

**combi.append(['longitude', 'latitude'])**

***#combi.append(['longitude', 'latitude', 'population'])***

**combi.append(['total\_rooms', 'total\_bedrooms'])**

**combi.append(['population','households'])**

**quantiles = list(df['median\_house\_value'].quantile([0.25, 0.5, 0.75, 1.0]))**

**df.loc[df['median\_house\_value'] >= quantiles[0], 'quantiles'] = 1**

**df.loc[df['median\_house\_value'] >= quantiles[1], 'quantiles'] = 2**

**df.loc[df['median\_house\_value'] >= quantiles[2], 'quantiles'] = 3**

**df.loc[df['median\_house\_value'] >= quantiles[3], 'quantiles'] = 4**

**y = df['quantiles'].astype("category")**

**main(df, y, scalers, models, hyperparams, combi)**