ASSIGNMENT - 3

Preparing the Data:

- Importing the required libraries such as numpy, pandas, sklearn etc..
- Loading the dataset into a dataframe.

	Elevation	Aspect	Slope	Hillshade_9am	Hillshade_Noon	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology
0	elevation_medium	aspect_medium	slope_low	hillshade_9am_max	hillnoon_max	0	1
1	elevation_high	aspect_medium	slope_low	hillshade_9am_max	hillnoon_max	1	1
2	elevation_medium	aspect_low	slope_low	hillshade_9am_max	hillnoon_max	1	1
3	elevation_high	aspect_ultra	slope_medium	hillshade_9am_max	hillnoon_max	2	1
4	elevation_high	aspect_high	slope_low	hillshade_9am_max	hillnoon_max	2	1
...	...	...	...	...	...	...	...
406703	elevation_ultra	aspect_medium	slope_low	hillshade_9am_max	hillnoon_max	1	1
406704	elevation_medium	aspect_low	slope_medium	hillshade_9am_max	hillnoon_max	0	1
406705	elevation_medium	aspect_medium	slope_low	hillshade_9am_max	hillnoon_max	0	1
406706	elevation_high	aspect_high	slope_low	hillshade_9am_max	hillnoon_max	2	2
406707	elevation_medium	aspect_ultra	slope_low	hillshade_9am_max	hillnoon_max	0	1

406708 rows × 11 columns

- Finding the data type, not null and count of each feature in the dataframe

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 406708 entries, 0 to 406707
Data columns (total 11 columns):
 #   Column                              Non-Null Count   Dtype 
---  ------                              --------------   ----- 
 0   Elevation                           406708 non-null  object
 1   Aspect                              406708 non-null  object
 2   Slope                               406708 non-null  object
 3   Hillshade_9am                       406708 non-null  object
 4   Hillshade_Noon                      406708 non-null  object
 5   Horizontal_Distance_To_Hydrology    406708 non-null  int64 
 6   Vertical_Distance_To_Hydrology      406708 non-null  int64 
 7   Horizontal_Distance_To_Fire_Points  406708 non-null  object
 8   Soil_Type                           406708 non-null  int64 
 9   Wilderness                          406708 non-null  int64 
 10  target                              406708 non-null  int64 
dtypes: int64(5), object(6)
memory usage: 34.1+ MB
```

- Checking the count of each target label in the dataframe

```
target
1    148288
2    198310
3     25028
4      1923
5      6645
6     12157
7     14357
dtype: int64
```

- Checking if there are any null values in the dataframe.

```
Elevation                             0
Aspect                                0
Slope                                 0
Hillshade_9am                         0
Hillshade_Noon                        0
Horizontal_Distance_To_Hydrology      0
Vertical_Distance_To_Hydrology        0
Horizontal_Distance_To_Fire_Points    0
Soil_Type                             0
Wilderness                            0
target                                0
dtype: int64
```

- Dropping the duplicate entries in the dataframe significantly reduced size of the dataframe from 406708 to 12495 rows.

	Elevation	Aspect	Slope	Hillshade_9am	Hillshade_Noon	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology
0	elevation_medium	aspect_medium	slope_low	hillshade_9am_max	hillnoon_max	0	1
1	elevation_high	aspect_medium	slope_low	hillshade_9am_max	hillnoon_max	1	1
2	elevation_medium	aspect_low	slope_low	hillshade_9am_max	hillnoon_max	1	1
3	elevation_high	aspect_ultra	slope_medium	hillshade_9am_max	hillnoon_max	2	1
4	elevation_high	aspect_high	slope_low	hillshade_9am_max	hillnoon_max	2	1
...	...	...	...	...	...	...	...
405630	elevation_high	aspect_high	slope_low	hillshade_9am_max	hillnoon_max	2	2
405688	elevation_medium	aspect_low	slope_medium	hillshade_9am_max	hillnoon_max	1	1
405828	elevation_high	aspect_ultra	slope_low	hillshade_9am_max	hillnoon_max	3	0
405883	elevation_low	aspect_ultra	slope_low	hillshade_9am_max	hillnoon_max	0	1
406025	elevation_high	aspect_low	slope_medium	hillshade_9am_max	hillnoon_max	0	1

12495 rows × 11 columns

- separating target column to use clustering algorithms.
  xx = df.drop(['target'], axis=1)
  y = df['target']

- Encoding the categorical values into numeric values using one hot encoder.
- One hot encoder will encode categorical features to one-hot numeric array.
- It creates binary column for each category and will return a sparse matrix.

```
data = pd.get_dummies(xx)
data.head()
```

	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Soil_Type	Wilderness	Elevation_elevation_high	Elevation_elevation_low	Elevation_elevatic
0	0	1	22	0	0	0	
1	1	1	32	2	1	0	
2	1	1	10	2	0	0	
3	2	1	23	2	1	0	
4	2	1	28	0	1	0	

5 rows × 23 columns

- Dimensionality reduction using tsne into two components tsne1 and tsne2.
- TSNE stands for T-distributed Stochastic Neighbor Embedding.It is a tool to visualize high dimensional data.
- We used tsne over pca because it preserves the local stricture or cluster of the data and also it can handle outliers (pca gets highly affected by outliers).
  tsne = TSNE(n_components=2)
  x_emb = tsne.fit_transform(data)

```
x_emb
```

```
array([[ 34.43368  ,  21.655138 ],
       [ 36.26295  , -18.390411 ],
       [ -4.4189544, -78.11279  ],
       ...,
       [ -5.707396 ,  47.086857 ],
       [-37.36911  ,  51.412697 ],
       [-50.749264 ,   8.6139965]], dtype=float32)
```

Q1) (1) Representative object of each cluster

K-means Clustering:

```
km = KMeans(n_clusters=7, random_state=0)
km.fit_predict(data)
y_km = km.labels_
```

```
#attribute1: coordinate of cluster centres
print(km.cluster_centers_)
```

```
[[ 1.84917770e+00  1.63056558e+00  3.14384276e+01  1.80144404e+00
   1.66533454e-16  1.54432411e-01  1.31167268e-01  2.71159246e-01
   2.39069394e-01  2.76774970e-01  5.18652226e-01  4.02326514e-01
   3.61010830e-03  5.13437625e-02  9.62695548e-03  6.85118331e-01
   3.14480546e-01]
 [ 1.25775194e+00  1.47093023e+00  1.03861434e+01  1.81637597e+00
   1.26937984e-01  5.24709302e-01 -1.80411242e-16  2.88759690e-01
   2.24321705e-01  2.91182171e-01  4.89341085e-01  3.90503876e-01
   3.87596899e-03  8.13953488e-02  1.55038760e-02  7.83914729e-01
   1.47286822e-01]
 [ 1.41123370e+00  1.26328987e+00  2.21850552e+01  1.09879639e+00
   1.52655666e-16  1.72016048e-01  1.70511535e-02  3.09929789e-01
   2.33701103e-01  2.77331996e-01  5.69709127e-01  3.87662989e-01
   4.01203611e-03  3.20962889e-02  3.51053159e-03  6.08324975e-01
   3.40020060e-01]
 [ 1.74327752e+00  1.52663623e+00  3.79766616e+01  9.75646880e-01
   1.24900090e-16 -4.99600361e-16  5.18011162e-01  3.06443430e-01
   2.63318113e-01  2.68899036e-01  5.35768645e-01  3.89649924e-01
   2.02942669e-03  3.09487570e-02  8.11770675e-03  6.31659056e-01
   3.51090817e-01]
 [ 1.53017751e+00  1.46153846e+00  2.81923077e+01  2.44970414e-01
   1.11022302e-16  3.02366864e-01  1.53846154e-02  2.89349112e-01
   2.62721893e-01  2.55621302e-01  4.19526627e-01  4.31952663e-01
   4.73372781e-03  6.50887574e-02  1.59763314e-02  5.29585799e-01
   3.15976331e-01]
 [ 9.23988842e-01  1.46443515e+00  2.55299861e+00  2.60111576e+00
   3.89818689e-01  5.45327755e-01  1.24900090e-16  2.55927476e-01
   3.04044630e-01  1.91073919e-01  4.40027894e-01  4.30962343e-01
   6.97350070e-04  4.18410042e-02  3.48675035e-03  9.53974895e-01
   4.18410042e-02]
 [ 8.22143698e-01  1.03062426e+00  1.73345112e+01  9.41107185e-01
   6.36042403e-02  3.75736160e-01  8.24499411e-03  2.93286219e-01
   2.54416961e-01  2.48527680e-01  7.53828033e-01  2.29681979e-01
  -4.33680869e-19  9.42285041e-03 -3.46944695e-18  5.34746761e-01
   2.76796231e-01]]
```

```
#attribute2: cluster label of each point
print(km.labels_)
```

```
[2 0 1 ... 4 5 3]
```

- Representative object: cluster centers represent coordinates of cluster centers.
- Labels represent the labels of each point.

Hierarchical Clustering: Agglomerative Clustering:

```
#compute agglomerative clustering
am = AgglomerativeClustering(n_clusters=7,compute_distances=True)
am.fit_predict(data)
y_am = am.labels_
```

```
#attribute1: cluster label of each point
print(am.labels_)
```

```
[6 2 0 ... 5 1 3]
```

```
#attribute2: distance between nodes in corresponding place in children(of non-leaf node)
print(am.distances_)
```

```
[   0.            0.            0.          ...  416.02339628  736.13165651
 1543.22871934]
```

```
#attribute3: number of leaves in hierarchical tree
print(am.n_leaves_)
#children of each non-leaf node
am.children_
```

```
12495

array([[    0,  2643],
       [    1,  1045],
       [    2,  1381],
       ...,
       [24977, 24984],
       [24983, 24986],
       [24985, 24987]], dtype=int64)
```

- Representative object: Labels represent the labels of each point.
- N_leaves represents the numbr of leaves in hierarchical tree.
- Distances shows distance between nodes in corresponding place in children(of non-leaf node)

BIRCH Clustering:

```
#compute BIRCH clustering
br = Birch(n_clusters=7)
br.fit_predict(data)
y_br = br.labels_
```

```
##attribute1: cluster label of each point
print(br.labels_)
```

```
[6 0 1 ... 2 4 3]
```

```
#attribute2: centroid of all subclusters
print(br.subcluster_centers_)
```

```
[[ 1.6  2.  26.  ...  0.   1.   0. ]
 [ 3.   1.  25.  ...  0.   1.   0. ]
 [ 2.   2.  25.  ...  0.   1.   0. ]
 ...
 [ 1.   1.   3.  ...  0.   1.   0. ]
 [ 2.   0.   2.  ...  0.   1.   0. ]
 [ 2.   0.   2.  ...  0.   1.   0. ]]
```

- Representative object: Labels represent the labels of each point.
- Subcluster centers shows centroid of all subclusters

Gaussian Mixture Clustering:

```
gmm = GaussianMixture(n_components = 7, random_state=0)
y_gmm = gmm.fit_predict(data)
```

```
#attribute1: mean of each mixture component
print(gmm.means_)
```

```
[[1.84182652e+00 1.49928428e+00 2.80377904e+01 1.18880615e+00
  0.00000000e+00 0.00000000e+00 1.88233618e-01 2.80131694e-01
  2.62381908e-01 2.51503007e-01 5.87317504e-01 4.12682496e-01
  0.00000000e+00 0.00000000e+00 0.00000000e+00 6.39278543e-01
  3.60721457e-01]
 [3.88276291e-01 1.26712462e+00 1.50317747e+01 1.47390958e+00
  2.80019109e-01 2.94761598e-01 0.00000000e+00 4.41915254e-01
  4.42242598e-02 4.65987177e-01 0.00000000e+00 1.32432888e-01
  0.00000000e+00 5.44185974e-01 5.21238257e-01 5.21238257e-01
  2.58060400e-01]
 [1.00720739e+00 1.30853136e+00 1.60282591e+01 1.41233770e+00
  0.00000000e+00 8.65051057e-01 0.00000000e+00 3.15744887e-01
  2.67301944e-01 2.32121752e-01 5.91121534e-01 4.08878466e-01
  0.00000000e+00 0.00000000e+00 0.00000000e+00 6.49940290e-01
  1.58017093e-01]
 [8.19414908e-01 1.51769984e+00 1.12632025e+01 2.53026812e+00
  5.84171049e-01 2.14311883e-01 0.00000000e+00 2.38789079e-01
  1.75868663e-01 3.53880904e-01 2.15653893e-01 3.48566194e-01
  0.00000000e+00 2.67246869e-01 0.00000000e+00 1.00000000e+00
  0.00000000e+00]
 [9.72130742e-01 1.46933759e+00 3.56607355e+01 7.80095752e-01
  0.00000000e+00 0.00000000e+00 4.47529731e-01 2.68014032e-01
  1.98191928e-01 4.01809108e-01 2.95074622e-02 2.16952715e-01
  0.00000000e+00 4.20276782e-01 0.00000000e+00 4.02133568e-01
  5.48687328e-01]
 [8.61612792e-01 1.86371274e+00 2.88523165e+01 1.72926548e+00
  0.00000000e+00 2.84269052e-01 1.21829486e-01 4.55479717e-01
  1.89502610e-01 2.88789285e-01 0.00000000e+00 3.01706651e-02
  4.87316681e-01 4.09164003e-01 7.39756341e-01 8.07439775e-01
  1.92560225e-01]
 [9.39374722e-01 1.36879716e+00 2.84458268e+01 0.00000000e+00
  0.00000000e+00 2.94773058e-01 0.00000000e+00 3.42403729e-01
  2.81824780e-01 3.28665464e-01 0.00000000e+00 1.12413416e-01
  8.67182498e-03 2.37553106e-01 9.09846326e-02 6.02307397e-01
  3.28209402e-01]]
```

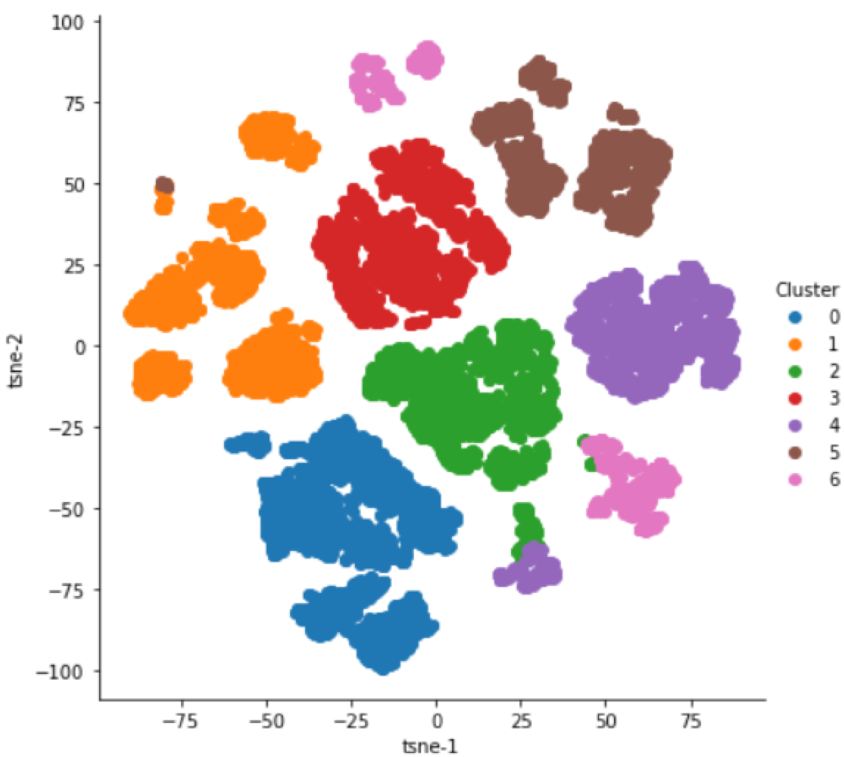
```
#attribute2: weights of each mixture component
print(gmm.weights_)
```

```
[0.55910362 0.00543037 0.27754936 0.11727285 0.01627358 0.00591228
 0.01845794]
```

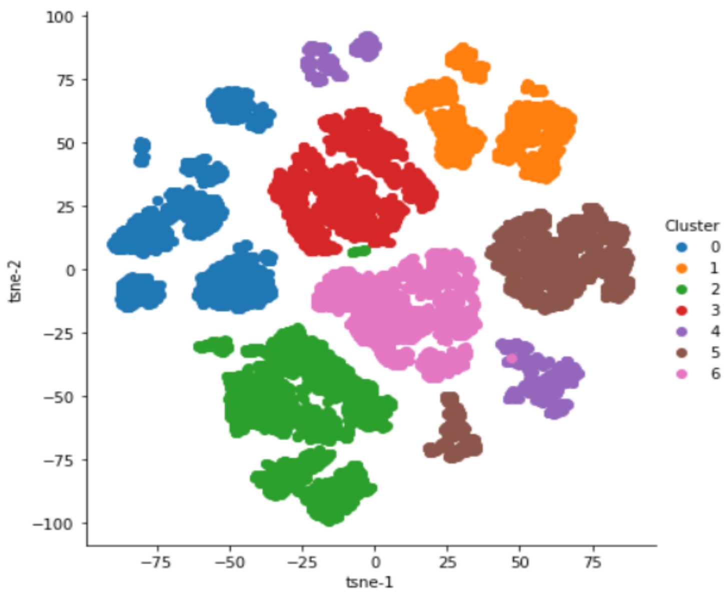
- Representative object: weights shows the weight of each mixture component
- Means shows the mean of each mixture component.

Q1 (2) Visualization of clusters

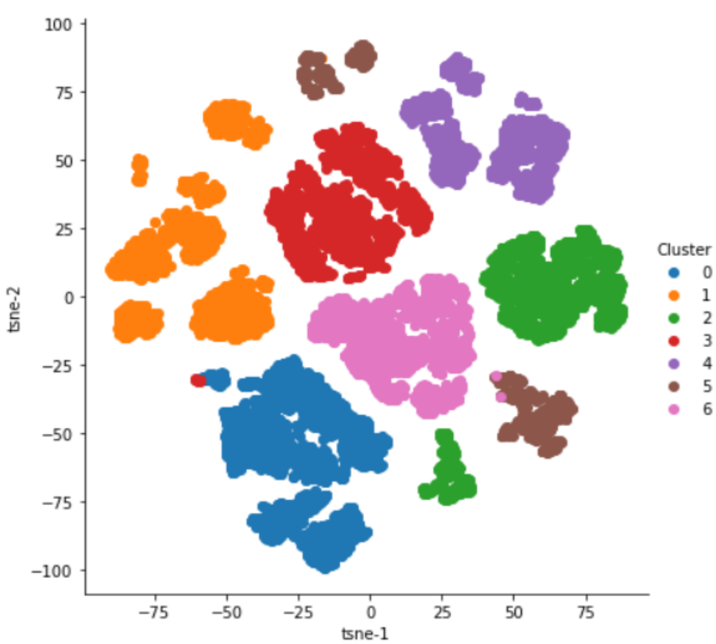
K-means Clustering:



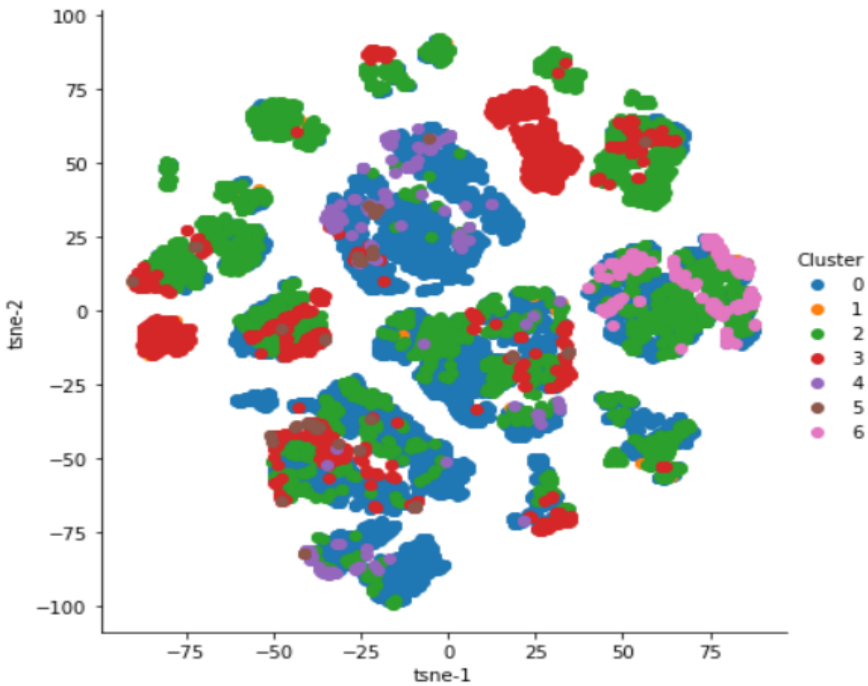
Hierarchical Clustering: Agglomerative Clustering:



BIRCH Clustering:



Gaussian Mixture Clustering:



Q1 (3) Cluster distribution with true label count

- Finding the true label count of target column

```
#value count of each label present in target column
y.value_counts()
```

```
2    4711
1    4112
7    1158
3    1056
6     686
5     597
4     175
Name: target, dtype: int64
```

- True label counts for K-means clustering

```python
#K-means clustering distribution with true label count
tl_count(y_km,y)
```

```
Cluster:0
2    1125
1    1018
7     180
5     113
6      43
3      14
Name: target, dtype: int64
Cluster:1
2    973
3    329
1    318
6    250
5    156
4     35
7      3
Name: target, dtype: int64
Cluster:2
2    372
1    304
5     64
3     48
6     46
4     13
7      2
Name: target, dtype: int64
Cluster:3
3    665
6    327
2    237
4    127
5     62
1     11
7      5
Name: target, dtype: int64
Cluster:4
1    1002
7     814
2     155
Name: target, dtype: int64
Cluster:5
2    959
1    881
7     81
5     53
6     20
Name: target, dtype: int64
Cluster:6
2    890
1    578
5    149
7     73
Name: target, dtype: int64
```

Datapoints present in each cluster:

```
#check the count of datapoints present in each cluster: K-means
dataframe1 = pd.DataFrame()
dataframe1['kmean'] = km.labels_
dataframe1['kmean'].value_counts()
```

```
0    2493
1    2064
5    1994
4    1971
6    1690
3    1434
2     849
Name: kmean, dtype: int64
```

- True label counts for Hierarchical clustering (agglomerative clustering)

```python
#Agglomerative clustering distribution with true label count
tl_count(y_am,y)
```

```
Cluster:0
2    984
3    329
1    318
6    251
5    156
4     35
7      3
Name: target, dtype: int64
Cluster:1
3    665
6    327
2    226
4    127
5     62
1     11
7      5
Name: target, dtype: int64
Cluster:2
2    1133
1    1023
7     180
5     113
6      43
3      14
Name: target, dtype: int64
Cluster:3
1    997
7    814
2    147
Name: target, dtype: int64
Cluster:4
2    371
1    304
5     64
3     48
6     45
4     13
7      2
Name: target, dtype: int64
Cluster:5
2    973
1    592
5    161
7     73
Name: target, dtype: int64
Cluster:6
2    877
1    867
7     81
5     41
6     20
Name: target, dtype: int64
```

Datapoints in each clusters:

```
#check the count of datapoints present in each cluster: Agglomerative
dataframe2 = pd.DataFrame()
dataframe2['agglomerative'] = am.labels_
dataframe2['agglomerative'].value_counts()
```

```
2    2506
0    2076
3    1958
6    1886
5    1799
1    1423
4     847
Name: agglomerative, dtype: int64
```

- True label counts for BIRCH clustering

```
#BIRCH clustering distribution with true label count
tl_count(y_br,y)
```

```
Cluster:0
2    984
3    329
1    318
6    251
5    156
4     35
7      3
Name: target, dtype: int64
Cluster:1
1    1006
7     820
2     147
Name: target, dtype: int64
Cluster:2
2    1133
1    1014
7     174
5     113
6      43
3      14
Name: target, dtype: int64
Cluster:3
2    376
1    316
5     64
3     48
6     45
4     13
7      4
Name: target, dtype: int64
Cluster:4
3    665
6    327
2    226
4    127
5     62
1     11
7      5
Name: target, dtype: int64
Cluster:5
2    979
1    593
5    161
7     73
Name: target, dtype: int64
Cluster:6
2    866
1    854
7     79
5     41
6     20
Name: target, dtype: int64
```

Datapoints in each cluster:

```
#check the count of datapoints present in each cluster: Birch
dataframe3 = pd.DataFrame()
dataframe3['BIRCH'] = br.labels_
dataframe3['BIRCH'].value_counts()
```

```
2    2491
0    2076
1    1973
6    1860
5    1806
4    1423
3     866
Name: BIRCH, dtype: int64
```

- True label counts for Gaussian mixture model clustering

```
#Gaussian mixture clustering distribution with true label count
tl_count(y_gmm,y)
```

```
Cluster:0
2    27
3    18
1    12
6     5
Name: target, dtype: int64
Cluster:1
1    54
2    48
7    20
5     3
Name: target, dtype: int64
Cluster:2
1    24
2    21
Name: target, dtype: int64
Cluster:3
3    507
2    373
6    276
4    143
1    118
7     17
5      8
Name: target, dtype: int64
Cluster:4
2    4042
1    3077
5     560
3     531
7     492
6     405
4      32
Name: target, dtype: int64
Cluster:5
1    750
7    623
2    101
Name: target, dtype: int64
Cluster:6
2    99
1    77
5    26
7     6
Name: target, dtype: int64
```

Datapoints in each cluster:

```
#check the count of datapoints present in each cluster: Gaussian Mixture
dataframe4 = pd.DataFrame()
dataframe4['GMM'] = y_gmm
dataframe4['GMM'].value_counts()
```

```
4    9139
5    1474
3    1442
6     208
1     125
0      62
2      45
Name: GMM, dtype: int64
```

Q1 (4) <u>Comparison of cluster formation of gaussian based model</u>

- We can see in the clusters formed above that the Gaussian model is considering a certain number of distributions as clusters. So, with the plot, we can see that the grouping is done for data points based on the distribution of clusters. Visualizing the Gaussian mixture model in 3D will give us show us better visualization of distributions.
- Gaussian Mixture Models are probabilistic models and use the soft clustering approach for distributing the points in different clusters. Uses mean as well as the variance of the data to update the centroid.
- **With K-means clustering:** In the above-visualized plots, we can see that the clusters formed are almost circular in shape. This is because the centroids of the clusters are updated iteratively using the mean value. But as the distribution of points is not circular so it fails to identify the right clusters. Hence, instead of distance-based, a distribution-based model like Gaussian gives better clusters.
- **With agglomerative clustering:** We can see that agglomerative clustering doesn't handle the outliers well. Also, as it is creating equal size clusters so it didn't predict the data points in the right clusters as well.
- **With BIRCH** (Balanced Iterative Reducing and Clustering using Hierarchies)**:** Birch performs better than the above two clustering algorithms in terms of scaling the data but the Gaussian mixture model performs better in terms of distribution.
- The Rand index computes the similarity measure between two clustering by considering all pairs of samples and counting pairs assigned in the same or different clusters in true and predicted clustering.
- The raw RI score is:

  RI = (number of agreeing pairs) / (number of pairs)

```
#rand score for K-means
rand_score(y,km.labels_)
```

0.6901680089356056

```
#rand score for agglomerative
rand_score(y,am.labels_)
```

0.6906230652978336

```
#rand score for BIRCH
rand_score(y,br.labels_)
```

0.6908459045535935

```
#rand score for gaussian mixture
rand_score(y,y_gmm)
```

0.6016009989717033

Q2 Prediction of clusters

```
#compute Gaussian mixture clustering (Gaussian Model)
gmm = GaussianMixture(n_components = 7)
gmm.fit(data)
y_gmm = gmm.predict(data)
```

```
def map_cl(y_target,y_label):
    cluster_map={}
    for i in range(7):
        add=np.where(y_label==i)[0].tolist()
        y=y_target.iloc[add]
        check_y=check_label(y)
        cluster_map[i]=check_y
    return cluster_map
```

```
#checking after mapping with GMM
cluster_map=map_cl(y,y_gmm)
cluster_map
```

```
{0: 1, 1: 7, 2: 6, 3: 2, 4: 7, 5: 5, 6: 3}
```

```
#function to assign clusters
def assign(cluster_map,y_label):
    yp=np.zeros(y_label.shape)
    for i in cluster_map:
        add=np.where(y_label==i)[0].tolist()
        yp[add]=cluster_map[i]
    return yp
```

```
yp=assign(cluster_map,y_gmm)
f1_score(y,yp,average='weighted')
```

```
0.3466280015913269
```

```
gmm.cluster_map=cluster_map
```

Conclusion: So, we are able to get 6 out of 7 clusters mapped and predicting based on the GMM which is giving best score.

For inference:

```
def predict(test_set) :
    prediction=[]
    model= pickle.load(open('final_model','rb'))
    testpd=pd.read_csv(test_set)
    testpd=pd.get_dummies(testpd)
    prediction=model.predict(testpd)
    yp=assign(model.cluster_map,np.array(prediction))
    return yp
```

- In the above predict function, we pass the .csv file.
- We do the pre processing steps inside the function itself like encoding.
- Then we load the dumped model into model and do the prediction on it.
- Finally, we return list of output labels.

References:

- https://scikit-learn.org/stable/modules/clustering.html
- https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/
- https://www.geeksforgeeks.org/clustering-in-machine-learning/
- https://medium.com/@mygreatlearning/clustering-algorithms-d7b3ae040a95