# egh400-1

October 24, 2023

## 1 CAB420, Practical 8 - Question 1 Solution

[1]:
```python
import pandas
import numpy
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from scipy.spatial import distance
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from scipy.cluster.hierarchy import dendrogram
from matplotlib import cm
from datetime import datetime
```

### 1.1 Read the data and setup

[2]:
```python
data = pandas.read_csv('Data20kcsv.csv');
#data['Author Email Status'] = data['Author Email Status'].map({'Success': 1,␣
 ↪'Bad': 0, 'SMTP Error': 0, "Absent":0, 'DNS Error':0})
print(data)
```

```
                        Package  SoftwareMaturityScore  FreshnessScore  \
0                         0-0-1               0.999916        1.000000
1                     0lever-so               0.999193        0.999918
2                  0lever-utils               0.998450        0.999840
3       0x01-autocert-dns-aliyun               0.999937        1.000000
4                 0x01-cubic-sdk               0.999911        0.999993
…                             …                    …               …
19994                   asserty               0.998008        0.999769
19995                   assess               0.999807        0.999976
19996                    asset               0.997128        0.999655
19997         asset-allocation               0.998262        0.999852
19998           assetallocator               0.999785        0.999942

       Accessibility_Score  Community_Engagement_Score  Vulnerability_Score  \
0                      0.2                    0.770533             0.460727
1                      0.4                    0.770533             0.114956
2                      0.4                    0.724129             0.460365
```

```
3                      0.2                    0.770533              0.460727
4                      0.2                    0.770533              0.460546
…                    …                          …                     …
19994                  0.4                    0.724129              0.460546
19995                  0.4                    0.770533              0.461812
19996                  0.2                    0.724129              0.466876
19997                  0.2                    0.770533              0.122189
19998                  0.2                    0.724129              0.114956

       Popularity Score  FinalRiskScore
0              0.999986        0.831905
1              0.999984        0.710778
2              0.999997        0.836168
3              1.000000        0.831910
4              0.999997        0.831817
…                 …              …
19994          0.999997        0.836134
19995          0.999997        0.848123
19996          0.999997        0.822576
19997          0.999986        0.697521
19998          0.999995        0.684086

[19999 rows x 8 columns]
```

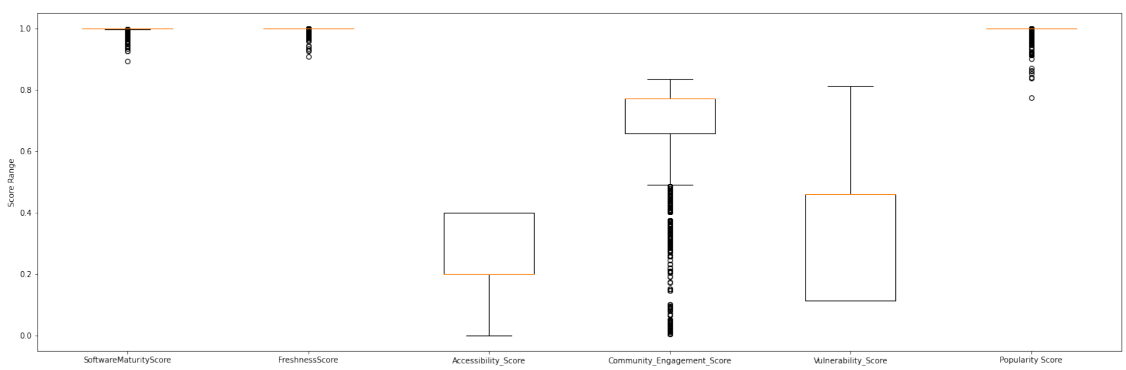
[3]:

```
#train = data.iloc[0:3000, [2,3,8,12,13,14,15,16,17,18,19]]
#test = data.iloc[3000:6000, [2,3,8,12,13,14,15,16,17,18,19]]

#train = data.iloc[0:3000, [2,3,8,12]]
#test = data.iloc[3000:6000, [2,3,8,12]]

#train = data.iloc[0:3000, [2,3,8,12,13,14,15]]
#test = data.iloc[3000:6000, [2,3,8,12,13,14,15]]

# Features to consider from new dataset

#SoftwareMaturityScore
#FreshnessScore
#Accessibility_Score
#Community_Engagement_Score
#Vulnerability_Score
#Popularity Score
#FinalRiskScore

#train = data.iloc[0:3000, [2,3,8,12,13,14,15,16,17,18,19,35]]
#test = data.iloc[3000:6000, [2,3,8,12,13,14,15,16,17,18,19,35]]

#train = data.iloc[0:50000, [2,3,8,12,13,14,15,16,17,18,19,35,38]]
```

```
#test = data.iloc[50000:100000, [2,3,8,12,13,14,15,16,17,18,19,35,38]]

#train = data.iloc[0:50000, [2,3,8]]
#test = data.iloc[50000:100000, [2,3,8]]

#train = data.iloc[0:8000, [1,2,3,4,5,6,7]]
#test = data.iloc[8000:16000, [1,2,3,4,5,6,7]]

train = data.iloc[0:8000, [1,2,3,4,5,6]]
test = data.iloc[8000:16000, [1,2,3,4,5,6]]

#train = data.iloc[0:8000, [3,4,5]]
#test = data.iloc[8000:16000, [3,4,5]]

#mu = numpy.mean(train)
#sigma = numpy.std(train)

#train = (train - mu) / sigma;
#test = (test - mu) / sigma;
```

[7]:
```
fig = plt.figure(figsize=[25, 8])
ax = fig.add_subplot(1, 1, 1)
xlabels = ['Software Maturity Score', 'Freshness Score', 'Accessibility Score', 'Community Engagement Score', 'Vulnerability Score', 'Popularity Score']
ylabel = 'Score Range'
ax.boxplot(train)
ax.set_xticklabels(labels)
ax.set_ylabel(ylabel)
```

[7]:
```
Text(0, 0.5, 'Score Range')
```



## 2 K-Means Algorithm

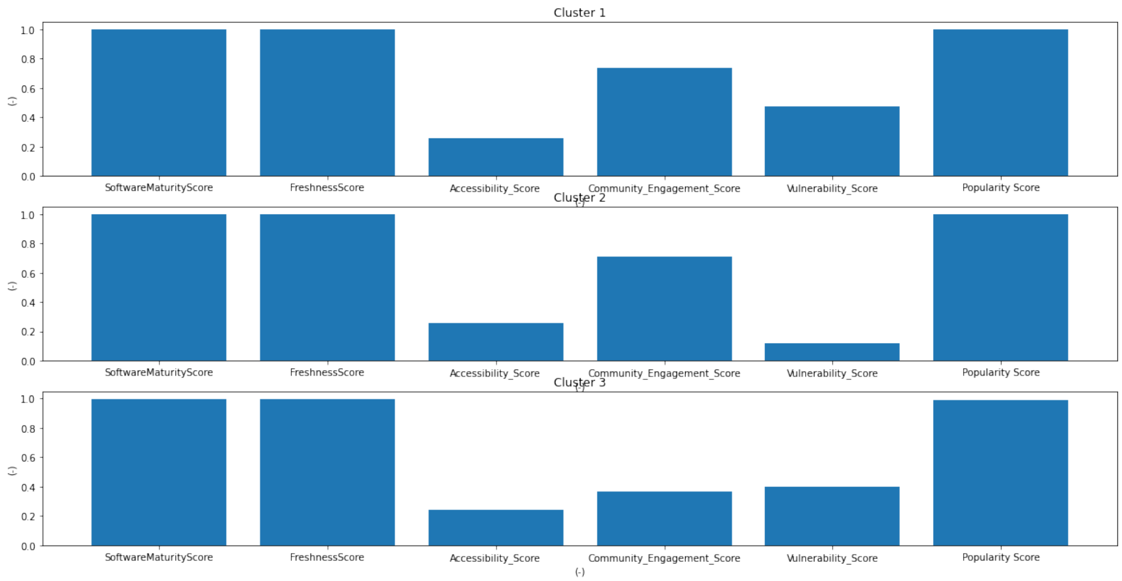
[16]:
```
#
# K-Means Analysis from CAB420, Prac 8, Q1
# Author: Simon Denman (s.denman@qut.edu.au)
#
def do_kmeans_analysis(n_clusters, random_state, train, test,␣
↪abnormal_threshold = 0.7):

    # train the k-means model
    kmeans = KMeans(n_clusters=n_clusters, random_state=random_state,␣
↪n_init='auto').fit(train)
    cluster_labels = kmeans.predict(test)

    # plot the cluster centres
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
↪'Closed_Issues', 'Open_Issues', 'Total_Branches']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones', 'Avg Package␣
↪Size']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones', 'Avg Package␣
↪Size', 'Author Email Status']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance']
    #x = ['SoftwareMaturityScore', 'FreshnessScore', 'Accessibility_Score',␣
↪'Community_Engagement_Score', 'Vulnerability_Score', 'Popularity Score',␣
↪'FinalRiskScore']
    x = ['SoftwareMaturityScore', 'FreshnessScore', 'Accessibility_Score',␣
↪'Community_Engagement_Score', 'Vulnerability_Score', 'Popularity Score']
    #x = ['Accessibility_Score', 'Community_Engagement_Score',␣
↪'Vulnerability_Score']
    fig = plt.figure(figsize=[20, 10])
    for i in range(n_clusters):
        ax = fig.add_subplot(n_clusters, 1, i + 1)
        ax.bar(x, kmeans.cluster_centers_[i,:])
        ax.set_title('Cluster %d' % (i+1))
        ax.set_xlabel('(-)')
        ax.set_ylabel('(-)')

    # plot the data, we can't plot 10-d data, so we'll run TSNE and show that
```

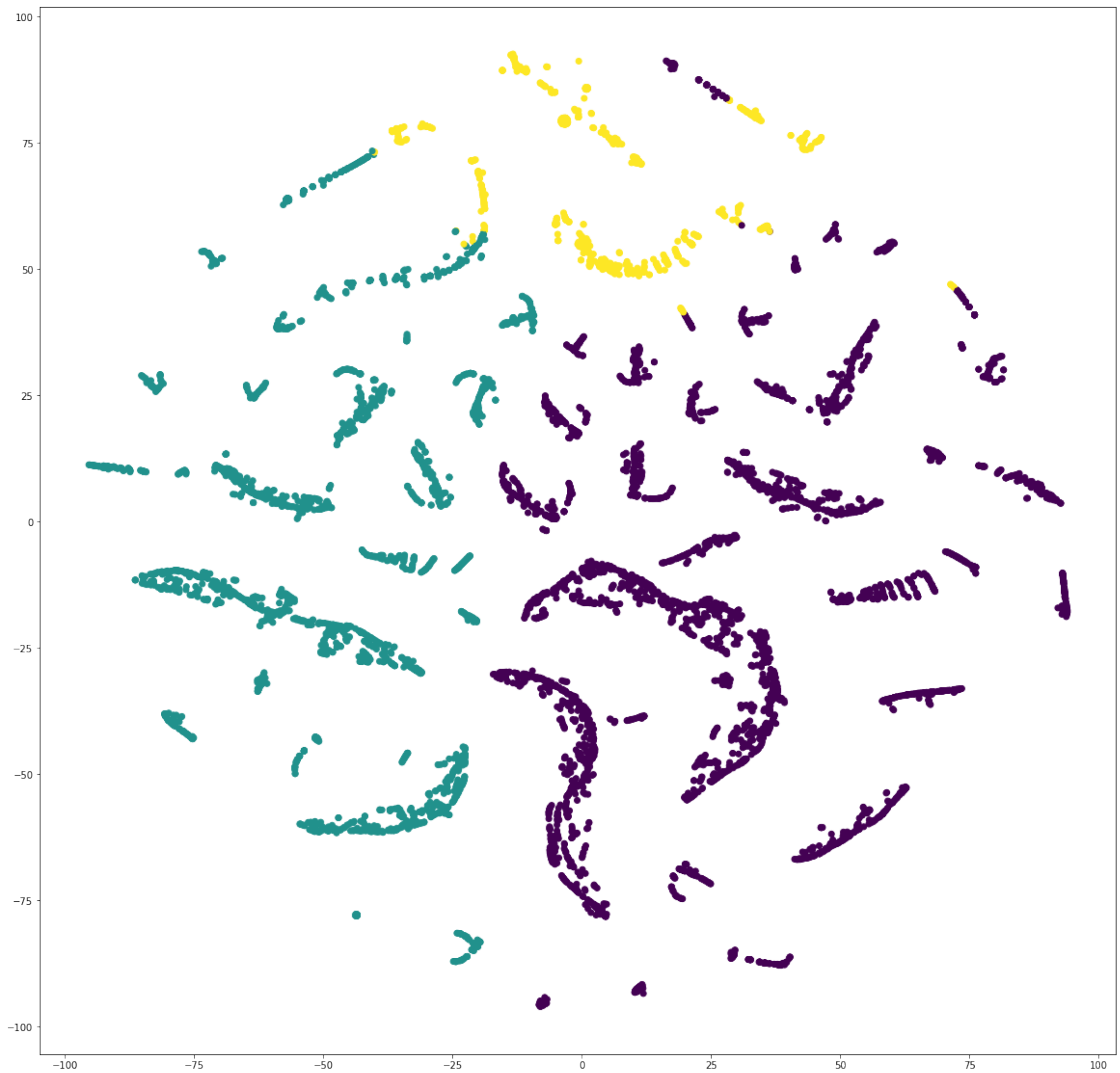
```
    fig = plt.figure(figsize=[20,20])
    ax = fig.add_subplot(1, 1, 1)
    tsne_embeddings = TSNE(random_state=4).fit_transform(numpy.vstack([train, kmeans.cluster_centers_]))
    ax.scatter(tsne_embeddings[:-n_clusters,0], tsne_embeddings[:-n_clusters,1], c = kmeans.labels_);
    #add cluster centres to the plot
    ax.scatter(tsne_embeddings[-n_clusters:,0], tsne_embeddings[-n_clusters:,1], s=200, marker='x')

    # find abnormal travellers. With k-means, we'll use distance to the cluster centre as our measure
    distances = kmeans.transform(test)
    distances = numpy.min(distances, axis=1)


    print('Risk Scores:')
    for count, (dist, cluster) in enumerate(zip(distances, cluster_labels)):
            #print(f'{cluster}')
        if cluster == 0:
        # If the package is in Cluster 0, multiply the distance by 5
            dist *= 5
        # If the new distance is greater than 1, set it to 1
            if dist > 1:
                dist = 1

        print(f'{dist}')

    print('Reliability Scores:')
    for count, (dist, cluster) in enumerate(zip(distances, cluster_labels)):
            #print(f'{cluster}')
        if cluster == 0:
        # If the package is in Cluster 0, multiply the distance by 5
            dist *= 5
        # If the new distance is greater than 1, set it to 1
            if dist > 1:
                dist = 1

        print(f'{1 - dist}')
```

We'll run three versions of K-means, with different random seeds.

[ ]:
```
do_kmeans_analysis(2, 59, train, test)
```

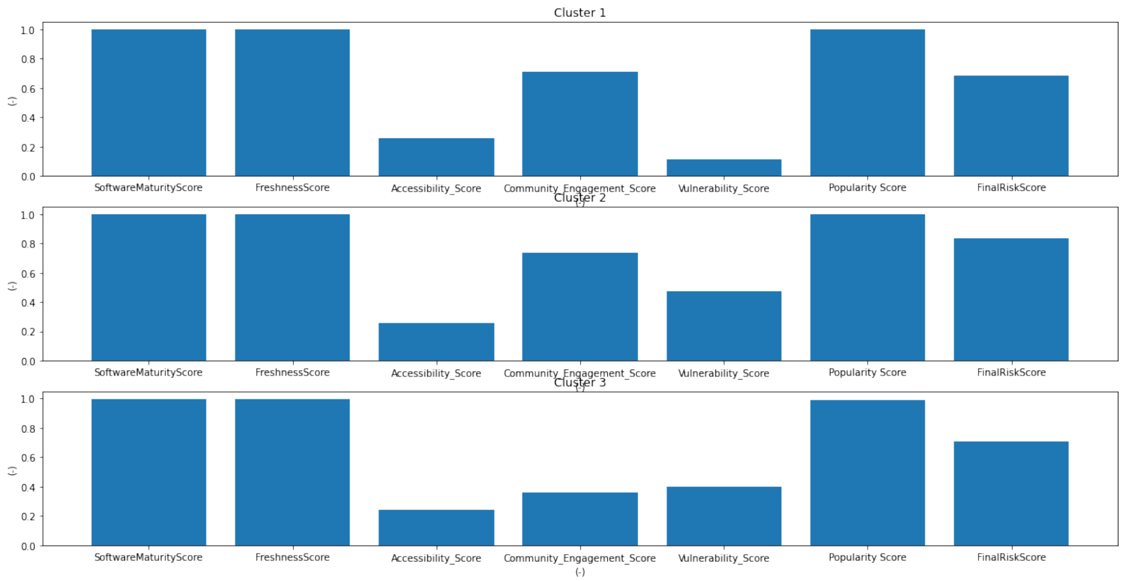
[11]:
```
do_kmeans_analysis(3, 314, train, test)
```

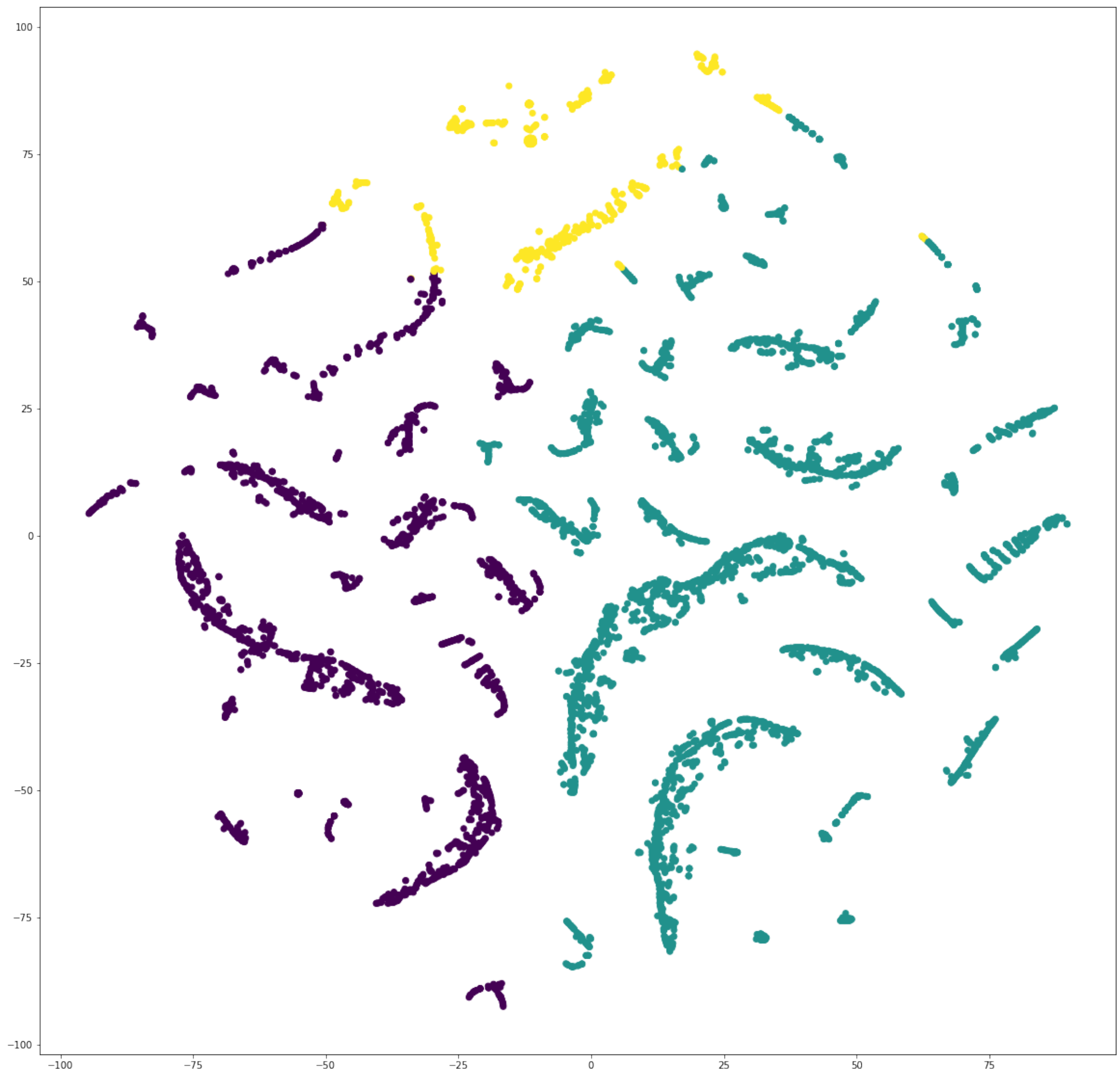

Cluster 1
Cluster 2
Cluster 3
(-)
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
0.0
0.2
0.4
0.6
0.8
1.0




```
<Figure size 1800x1440 with 0 Axes>
```

[30]:
```
do_kmeans_analysis(3, 200, train, test)
```


Cluster 1
Cluster 2
Cluster 3
(-)
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
FinalRiskScore
0.0
0.2
0.4
0.6
0.8
1.0



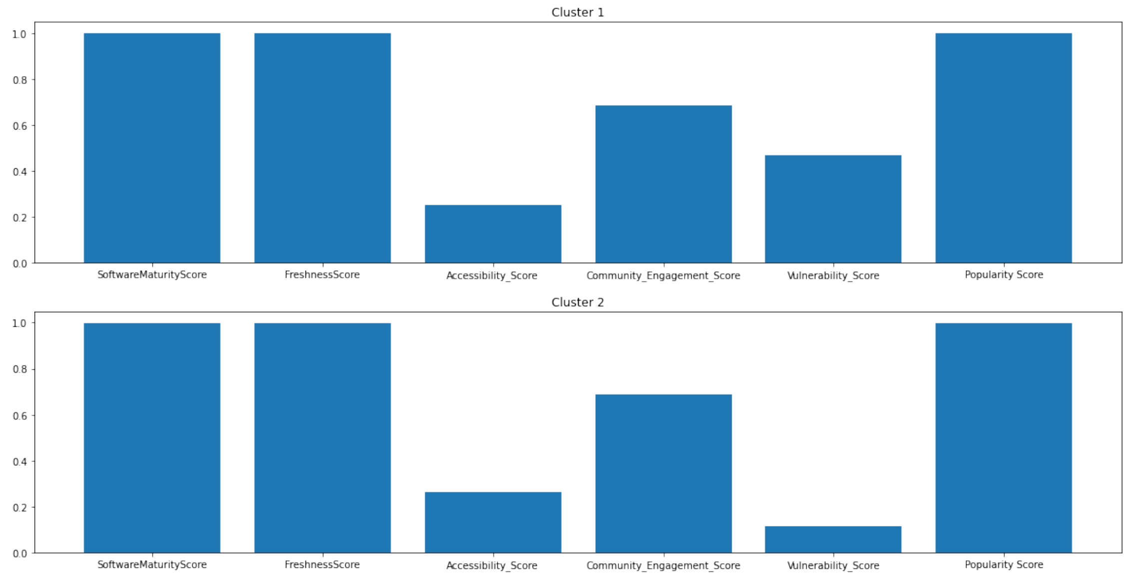

<Figure size 1800x1440 with 0 Axes>

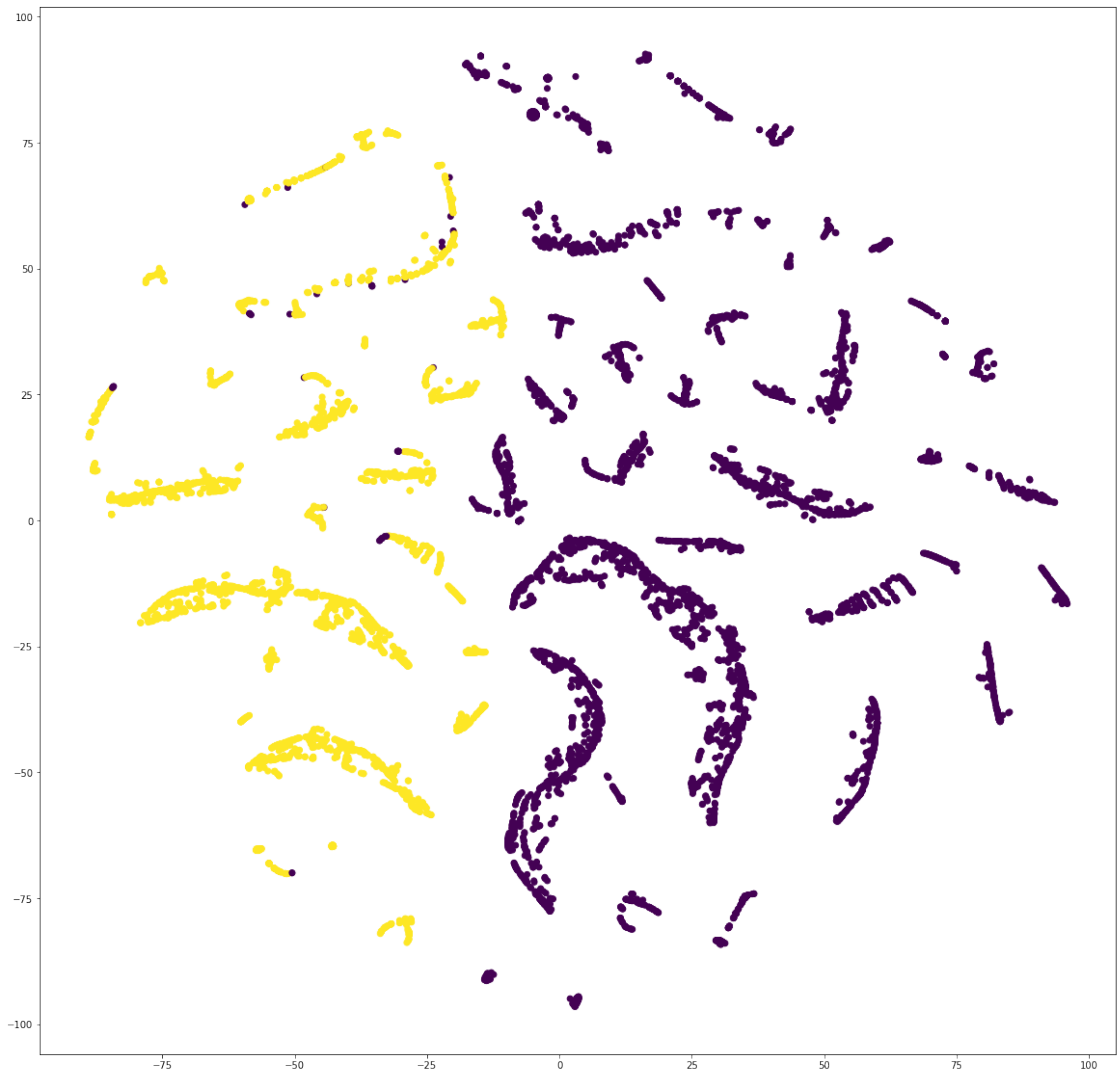
#### 2.0.1 GMM

[18]:
```
#
# GMM Analysis from CAB420, Prac 8, Q1
# Author: Simon Denman (s.denman@qut.edu.au)
#
def do_gmm_analysis(n_components, random_state, train, test):

    # train the GMM
    gmm = GaussianMixture(n_components=n_components, random_state=random_state).
↪fit(train)
```

```
    # plot the component means
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones']
    x = ['SoftwareMaturityScore', 'FreshnessScore', 'Accessibility_Score',␣
↪'Community_Engagement_Score', 'Vulnerability_Score', 'Popularity Score']
    fig = plt.figure(figsize=[20, 10])
    for i in range(n_components):
        ax = fig.add_subplot(n_components, 1, i + 1)
        ax.bar(x, gmm.means_[i,:])
        ax.set_title('Cluster %d' % (i+1))

    # TSNE plot of the data
    fig = plt.figure(figsize=[20,20])
    ax = fig.add_subplot(1, 1, 1)
    tsne_embeddings = TSNE(random_state=4).fit_transform(numpy.vstack([train,␣
↪gmm.means_]))
    train_labels = gmm.predict(train)
    ax.scatter(tsne_embeddings[:-n_components,0], tsne_embeddings[:
↪-n_components,1], c = train_labels);

    # add component means to the scatter plot
    #ax.scatter(tsne_embeddings[-n_components:,0],␣
↪tsne_embeddings[-n_components:,1], s=200, marker='x')

    # compute likelihood for travellers. With a GMM, we can get the likelihood␣
↪that such a traveller exists
    # given the learned distribution
    distances = gmm.score_samples(test)
```

[19]:
```
do_gmm_analysis(2, 59, train, test)
```


Cluster 1
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
Cluster 2
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score


100
75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75
100

[ ]: