Assignment 5: Travel Review Ratings

Submitted by: Indronil Bhattacharjee

::: {.cell _cell_guid=‘b1076dfc-b9ad-4769-8c92-a6c4dae69d19’ _uuid=‘8f2839f25d086af736a60e9eeb907d3b93b6
execution=‘{“iopub.execute_input”:“2023-10-25T02:21:40.466569Z”,“iopub.status.busy”:“2023-10-25T02:21:40.466205Z”,“iopub.status.idle”:“2023-10-25T02:21:40.986455Z”,“shell.execute_reply”:“2023-10-25T02:21:40.985155Z”,“shell.execute_reply.started”:“2023-10-25T02:21:40.466539Z”}’
trusted=‘true’ execution_count=1}

```
import numpy as np
import pandas as pd

# Load the dataset
data = pd.read_csv("google_review_ratings.csv")
```

```
/kaggle/input/google-review-ratings/google_review_ratings.csv
```

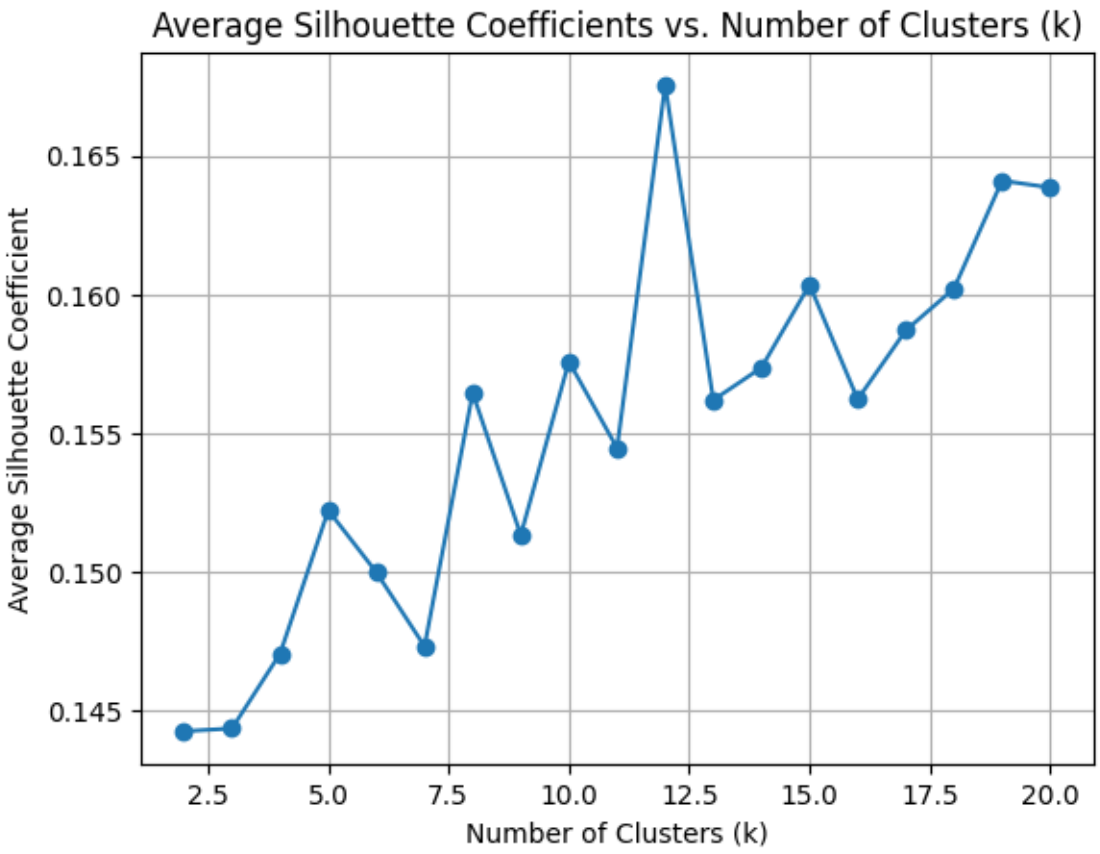
:::

Question 1. (k-means)

1a. Finding the best k based on Silhouette coefficient

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import warnings
warnings.filterwarnings("ignore")

# Select the features you want to use for clustering (e.g., data[['feature1', 'feature2',
features = data.iloc[:, 1:]
```

```
from sklearn.impute import SimpleImputer

# Replace NaN values with the mean of each column
imputer = SimpleImputer(strategy="mean")
features = imputer.fit_transform(features)

# Range of k values
k_values = range(2, 21)
silhouette_scores_kmeans = []

# Compute Silhouette coefficient for different k values
for k in k_values:
    avg_silhouette_score = 0
    for _ in range(5):  # Run K-Means 5 times for each k
        kmeans = KMeans(n_clusters=k, random_state=0)
        cluster_labels = kmeans.fit_predict(features)
        avg_silhouette_score += silhouette_score(features, cluster_labels)
    avg_silhouette_score /= 5  # Average over 5 runs
    silhouette_scores_kmeans.append(avg_silhouette_score)

# Plot the average Silhouette coefficients for different k
plt.plot(k_values, silhouette_scores_kmeans, marker='o')
plt.title("Average Silhouette Coefficients vs. Number of Clusters (k)")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Average Silhouette Coefficient")
plt.grid(True)
plt.show()

# The best k is where Silhouette coefficient is highest
best_k = k_values[np.argmax(silhouette_scores_kmeans)]
print("Best k:", best_k)
```




```
Best k: 12
```

1b. Training k-means model with the best k and reporting the centroids of clusters

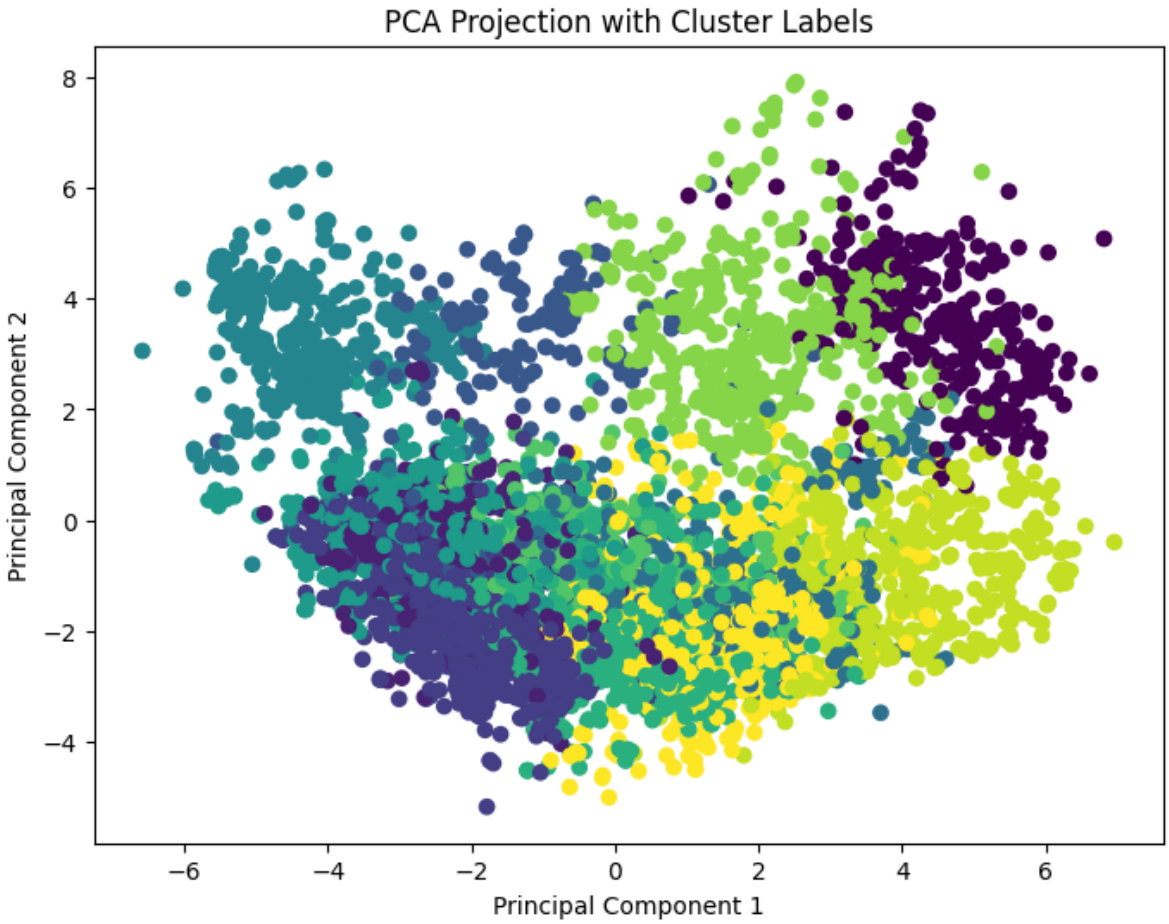
```
best_kmeans = KMeans(n_clusters=best_k, random_state=0)
best_kmeans.fit(features)
centroids = best_kmeans.cluster_centers_
print("Cluster Centroids:")
print(centroids)
```

```
Cluster Centroids:
[[2.58306306 2.54192192 2.39036036 1.97435435 1.83       1.56453453
  1.49237237 1.29486486 1.46084084 1.49630631 1.4993994  1.08387387
  0.99957958 1.27294294 2.46756757 1.68456456 1.43906907 1.33714715
  1.68735736 2.25882883 3.09216216 4.3215015  3.42363363 3.29003003]
 [1.14829837 4.98480186 2.20741259 2.05783217 2.25440559 3.06030303
```

```
 4.63596737 3.42058275 4.34762238 3.53946387 2.59062937 2.1981352
 1.86265734 3.00741259 3.03275058 0.72939394 0.67876457 0.69016317
 0.67144522 0.82659674 0.79282051 0.8090676  0.8293007  0.95531469]
[1.24601316 1.42939474 1.71228947 2.17055263 2.27046053 2.58097368
 3.64909211 3.71442105 4.55152632 4.55010526 4.12588158 2.01676316
 1.64163158 1.38232895 1.27647368 1.05517105 0.62452632 0.33989474
 0.30411842 0.40644737 0.56419737 1.18188158 1.03177632 1.20017105]
[1.43073529 1.69436275 1.97240196 2.13578431 2.14058824 2.05009804
 3.50656863 2.23504902 2.69632353 2.53392157 2.63362745 2.39666667
 2.48745098 3.29995098 4.12303922 4.87088235 3.09980392 1.80181373
 0.58441176 0.63769608 1.15279412 1.34534314 1.3170098  1.35857843]
[2.1044582  2.86798762 2.58510836 3.21417957 3.94616099 3.24894737
 3.36417957 2.6704644  2.41733746 2.32399381 2.22653251 2.59467492
 2.34108359 1.77597523 0.96668731 0.76467492 0.94455108 0.75151703
 0.84832817 0.80659443 0.78879257 0.96600619 2.56352941 4.51513932]
[0.63795031 0.90426501 1.57304348 1.56414079 1.58952381 1.60594203
 2.91443064 1.94915114 2.73871636 2.94807453 3.33014493 4.34202899
 4.57652174 4.91494824 3.85124224 0.74086957 0.81281573 0.94977226
 1.38339545 1.08343685 0.5689648  0.54706004 0.53037267 0.63583851]
[1.10949192 1.30140878 2.30808314 2.21792148 2.53318707 3.59113164
 4.73586605 3.22568129 4.58528868 3.37143187 2.10939954 1.74226328
 2.13166282 3.0451963  4.63831409 0.73852194 0.62036952 0.62124711
 0.69743649 0.83237875 0.85612009 0.9456582  0.9491224  1.04009238]
[1.15067073 1.50246951 2.66643293 3.23785061 4.01728659 4.3158689
 4.55065549 2.42141768 2.8632622  2.04949695 1.94300305 1.77367378
 1.81605183 1.82827744 1.31896341 0.88522866 0.68254573 0.45969512
 0.55321646 0.55879573 0.70109756 0.83839939 1.1070122  0.96501524]
[1.82064151 2.134      2.50513208 4.01267925 3.94316981 2.84864151
 3.21396226 2.90928302 3.58218868 3.37316981 4.09901887 2.43615094
 4.76422642 2.10196226 1.56007547 1.03098113 0.91298113 0.77116981
 0.70060377 0.68867925 0.70313208 3.61328302 2.98030189 1.63207547]
[2.0573499  2.63492754 2.48677019 2.33517598 2.13146998 2.04329193
 2.14068323 1.69703934 1.95971014 1.61929607 1.58401656 1.63055901
 1.8747412  2.26300207 2.29660455 1.38670807 1.85397516 2.19354037
 3.5547412  2.30650104 1.26826087 1.90018634 1.90979296 2.0511604 ]
[1.56507067 2.2425265  3.26227915 4.40484099 4.07934629 3.24521201
 2.56975265 1.9790636  2.43618375 2.515053   2.26632509 1.43012367
 1.09443463 1.05053004 1.19139576 1.26561837 0.86468198 0.6440106
 0.59922261 1.05491166 1.1170318  4.79159011 2.27998453 1.70581272]
[1.46961612 4.26044146 3.99297505 3.97654511 4.31345489 3.42307102
 2.76483685 2.27466411 2.78834933 2.65886756 1.87754319 1.76765835
 1.6228215  1.53831094 1.64714012 1.29322457 0.50763916 0.42773512
 0.46477927 0.97261036 0.85314779 0.89560461 1.11662188 1.11641075]]
```

1c. Projecting the data using PCA with two principal components and plotting

```
from sklearn.decomposition import PCA

# Project the data using PCA with two principal components
pca = PCA(n_components=2)
projected_data = pca.fit_transform(features)

# Draw a scatter plot with cluster IDs as labels
cluster_labels = best_kmeans.labels_
plt.figure(figsize=(8, 6))
plt.scatter(projected_data[:, 0], projected_data[:, 1], c=cluster_labels, cmap='viridis')
plt.title("PCA Projection with Cluster Labels")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



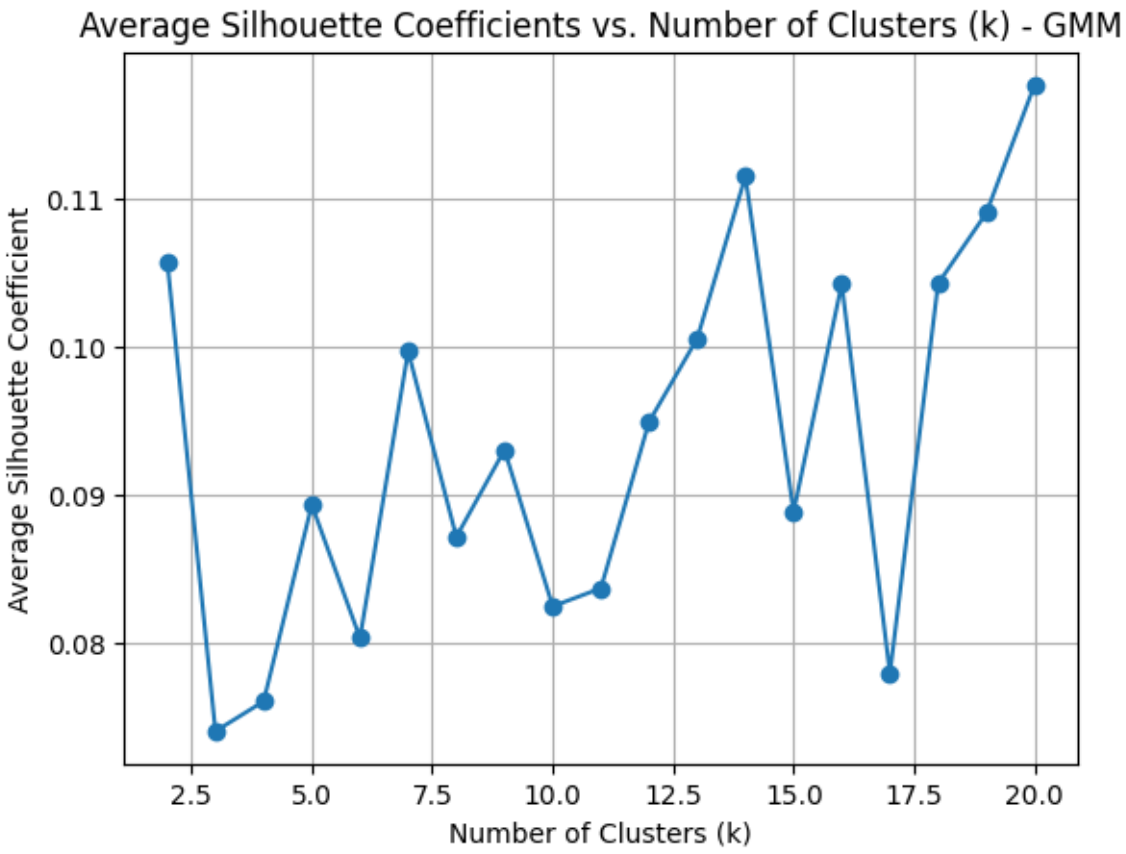
The visual examination and Silhouette coefficient serve as complementary methods to assess the quality of clustering. High Silhouette scores and visually well-separated clusters are consistent with each other and suggest that the clustering is successful. Low scores and overlapping clusters in the visualization may indicate issues with the clustering algorithm or choice of hyperparameters.

Here, the clusters at the right side and top-left corners appear to be well separated keeping some outlying points out. Some clusters of bottom-left corner appear to be overlapping. The best Silhouette coefficient score for k=12 is 0.17. The much silhouette coefficient will be, clusters will be more separated and consistent.

Question 2. (Gaussian Mixture Model and Spectral Clustering)

2a. Gaussian Mixture model

```
from sklearn.mixture import GaussianMixture
features = data.iloc[:, 1:]

from sklearn.impute import SimpleImputer

# Replace NaN values with the mean of each column
imputer = SimpleImputer(strategy="mean")
features = imputer.fit_transform(features)

# Range of k values
k_values = range(2, 21)
silhouette_scores_gmm = []
# Compute Silhouette coefficient for different k values
for k in k_values:
    avg_silhouette_score = 0
    for _ in range(5):  # Run GMM 5 times for each k
        gmm = GaussianMixture(n_components=k, random_state=0)
        cluster_labels = gmm.fit_predict(features)
        avg_silhouette_score += silhouette_score(features, cluster_labels)
    avg_silhouette_score /= 5  # Average over 5 runs
    silhouette_scores_gmm.append(avg_silhouette_score)

# Plot the average Silhouette coefficients for different k
plt.plot(k_values, silhouette_scores_gmm, marker='o')
plt.title("Average Silhouette Coefficients vs. Number of Clusters (k) - GMM")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Average Silhouette Coefficient")
plt.grid(True)
plt.show()

# The best k is where Silhouette coefficient is highest
best_k_gmm = k_values[np.argmax(silhouette_scores_gmm)]
print("Best k (GMM):", best_k_gmm)
```




```
Best k (GMM): 20
```

```
best_gmm = GaussianMixture(n_components=best_k_gmm, random_state=0)
best_gmm.fit(features)
cluster_means = best_gmm.means_
print("Cluster Centroids (Gausian Mixture):")
print(cluster_means)
```

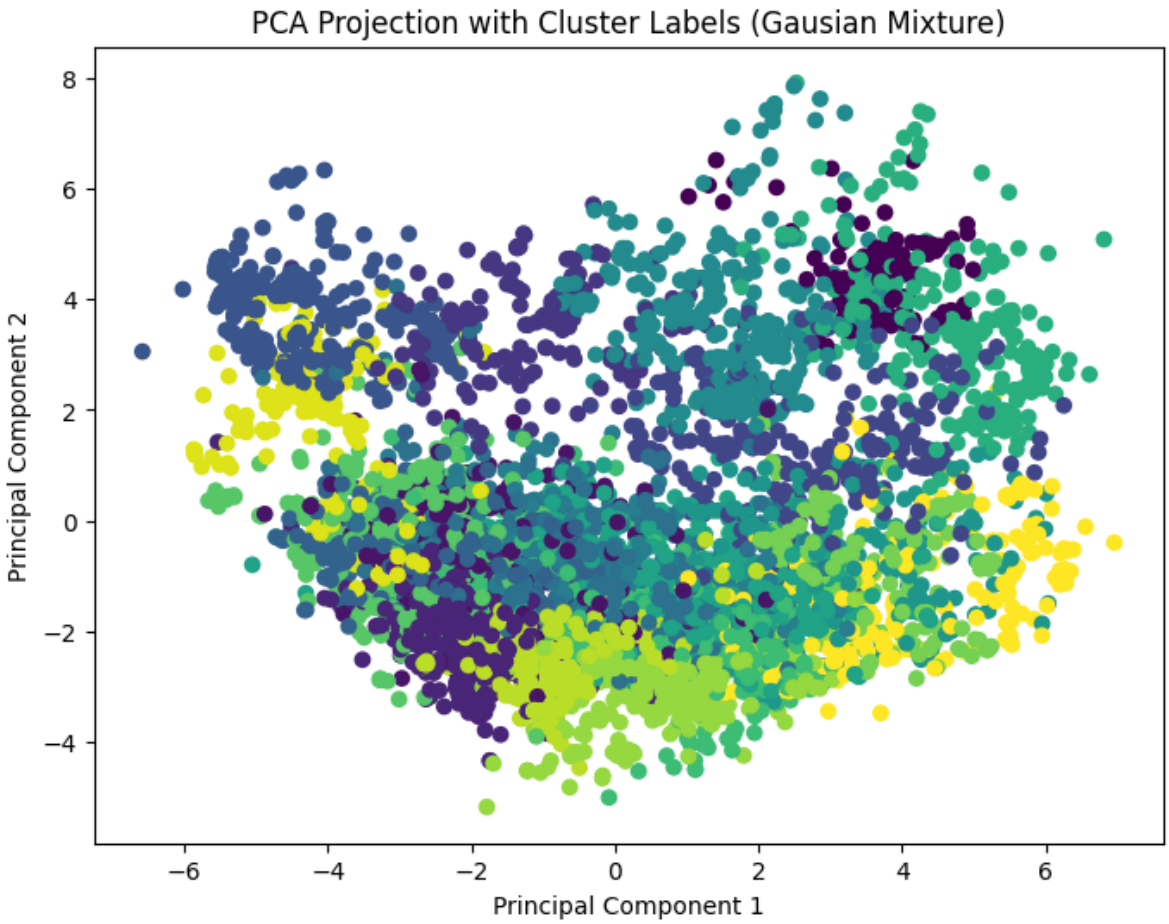
```
Cluster Centroids (Gausian Mixture):
[[2.47582609 2.49       2.24773913 1.77747826 1.72269565 1.35113043
  1.28573913 1.28756522 1.46321739 1.58608696 1.65886957 0.91947826
  0.90252174 1.15008696 5.         1.93356521 1.2355652  1.33356521
  1.3547826  2.22443478 3.7272174  3.89565218 3.57078261 2.92608695]
 [1.18960148 5.         2.50391104 2.39127225 2.51306106 3.2461729
  4.41138062 3.17831432 4.05084632 3.15182421 2.4387301  2.29731193
  1.87749488 3.33994773 2.65577394 0.71496768 0.70414635 0.71325639
```

```
 0.61744214 0.88773826 0.76613341 0.76729651 0.82405268 1.0203946 ]
[1.31230897 1.43301763 1.55474756 2.24998508 2.35714926 2.49436551
 3.33512136 3.67847617 4.48558652 4.50088608 5.         2.06993958
 1.55841396 1.31108568 1.38952808 1.19890656 0.7058963  0.27044773
 0.20533015 0.21493709 0.46585474 1.46935015 1.04639788 1.28165831]
[1.48280253 1.71764333 1.92184718 1.89885339 1.88668788 1.86878979
 3.29573268 2.15522295 2.55707012 2.36445859 2.30127391 2.40624208
 2.50579623 3.76509578 4.28337611 4.93592356 3.55261125 2.04987217
 0.62598689 0.57745232 1.32617846 1.47477705 1.44452227 1.45617833]
[2.48673139 3.75888973 3.55672985 3.04888392 2.55022147 2.37878572
 2.26176739 1.4883785  2.29664017 1.32103908 1.30315164 1.32749772
 1.36337311 1.40281782 2.22850038 0.81252907 0.87394588 0.81431011
 2.27357001 2.02002502 0.87402162 1.58288081 1.53736755 2.03283244]
[0.70924922 0.89429428 1.598048   1.58117116 1.5729129  1.5829129
 2.86039014 1.80423411 2.63735736 2.86744746 3.00960964 4.25327345
 4.50489484 5.         5.         0.68765767 0.67093094 0.79336339
 1.20630637 1.19270277 0.67852849 0.65126115 0.62840837 0.67078075]
[0.96758997 2.55044885 1.87976367 1.80352875 2.12538076 2.9883058
 4.7760039  3.42818583 5.         4.08160235 2.47984259 1.79780678
 1.51560944 1.38049962 4.97954147 0.78550122 0.60164063 0.60325552
 0.75541403 0.73959793 0.91083283 0.95901975 0.91283978 0.92963628]
[1.30697584 1.55887352 1.95725714 2.35349221 3.48314169 4.14194061
 4.61848124 2.30229131 3.30595885 1.89013839 1.65459191 1.8218864
 2.25557606 2.67885317 2.04704902 0.69358294 0.69831054 0.72812016
 1.02051877 0.75982242 0.84306725 0.8470159  0.85570234 1.37640333]
[1.8326053  2.09085112 2.47617674 3.71619719 3.59664208 2.65573541
 3.24286226 3.00991643 3.82815076 3.462317   4.28672205 1.97059131
 5.         2.22531538 1.62357112 1.06095406 0.89857203 0.75410789
 0.68907273 0.6762582  0.68924163 3.46724069 2.6914216  1.62446315]
[2.01354918 2.45009616 2.25385969 2.01965718 2.00951059 1.95227952
 2.18233692 1.78527589 1.87606688 1.72781368 1.67393464 1.71891075
 2.13230732 2.53277536 2.35634631 1.52385707 1.96646398 2.46074044
 3.40443768 2.21583048 1.22901983 2.00199067 2.01340534 2.19856231]
[1.23779541 2.92789695 2.92798237 4.48514545 3.50734623 2.64613557
 2.45945572 1.71862025 3.61089588 3.08152877 0.92650354 0.90773414
 0.9218488  1.04546592 1.10845782 1.56782643 0.61621916 0.6054832
 0.63203692 2.79264976 2.26493249 3.67713162 1.87150531 1.20772223]
[1.59845776 2.57856547 2.51561703 3.5711327  4.52938319 3.54364025
 3.47100235 2.61366044 2.535051   2.45706245 2.45050417 3.02400095
 2.49671161 1.68099985 1.03890637 0.84237028 0.87251313 0.80515606
 0.82418905 0.76187627 0.780016   0.81364121 2.05879898 2.57033181]
[2.61738703 2.4111461  2.33363641 2.09246059 1.89569815 1.65280737
 1.53376471 1.33658074 1.44433011 1.43624214 1.46545265 1.28224164
```

```
 1.12565634 1.54650239 1.04774316 1.61259854 2.03865358 1.84145041
 2.1264263  2.16890813 2.72031813 3.9789039  3.0924287  3.34603361]
[1.36118226 3.89393328 3.79145108 3.97003266 4.44179913 3.39486554
 3.22434185 2.67429198 2.95216507 3.31615838 2.65858707 1.96157209
 1.94230117 1.80491741 2.19229239 2.24895263 0.33735028 0.30603146
 0.35938443 0.78147677 0.71666883 0.8800647  0.95961953 1.20246821]
[0.96722717 1.6982217  2.51081449 2.07387685 2.30276243 3.3321061
 4.68410352 3.34880754 4.50771135 3.60785888 2.55678511 1.80928596
 2.43178669 3.92614496 3.28571926 0.66506109 0.62870002 0.63529902
 0.67951808 0.88033465 0.80753201 0.89580138 0.88463022 0.9249996 ]
[1.60561294 1.99705993 2.95076646 4.49888208 4.3421851  3.52860436
 2.23551992 2.03218476 1.93666013 2.18024865 2.51592086 1.94072548
 1.33914359 1.06742487 1.13710439 1.16691507 0.92660784 0.66734043
 0.59399852 0.58606789 0.62721914 4.85846522 0.95837591 1.79726216]
[1.15109135 1.96364718 3.89958105 4.00599887 4.36529853 4.38949171
 4.37356246 2.3150952  2.63788197 2.8015447  2.30814903 1.60840021
 1.60193345 1.57869228 1.54585203 1.17697616 0.63571057 0.05354903
 0.0324016  0.12977529 0.58556006 1.21523935 0.99501552 1.01994395]
[1.27773405 1.75104904 1.74960665 2.44307378 2.64437403 3.51096685
 4.34005465 4.30315278 4.69709709 3.83102302 1.36531053 1.33660791
 1.27521733 1.2238736  0.96925332 0.62299749 0.41335885 0.27607118
 0.37910676 0.78215608 0.83620837 1.13975247 1.25894406 1.26263567]
[0.3841467  0.93151249 1.43346471 1.43528125 1.48994016 1.55371414
 2.9372118  2.40780894 3.01145339 3.38786544 4.36838364 4.65822498
 4.329605   3.88312119 0.84050295 0.90038027 1.19390499 1.35799083
 1.5993129  0.6342051  0.15810533 0.15514954 0.16458364 0.42710155]
[1.65546268 2.22356609 3.4019294  3.93931662 3.93228183 3.28664258
 3.16043942 2.3063902  2.29380087 2.32003257 2.88042632 1.58896597
 1.05107387 0.90603833 1.09892822 0.9502743  1.01133394 0.74495998
 0.68764774 0.67729319 0.84052593 3.75231219 4.93427339 2.40280192]]
```

```python
from sklearn.decomposition import PCA

# Project the data using PCA with two principal components
pca = PCA(n_components=2)
projected_data_gmm = pca.fit_transform(features)

# Draw a scatter plot with cluster IDs as labels
cluster_labels_gmm = best_gmm.fit_predict(features)

# Draw a scatter plot with cluster IDs as labels
plt.figure(figsize=(8, 6))
```

```
plt.scatter(projected_data_gmm[:, 0], projected_data_gmm[:, 1], c=cluster_labels_gmm, cmap
plt.title("PCA Projection with Cluster Labels (Gausian Mixture)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

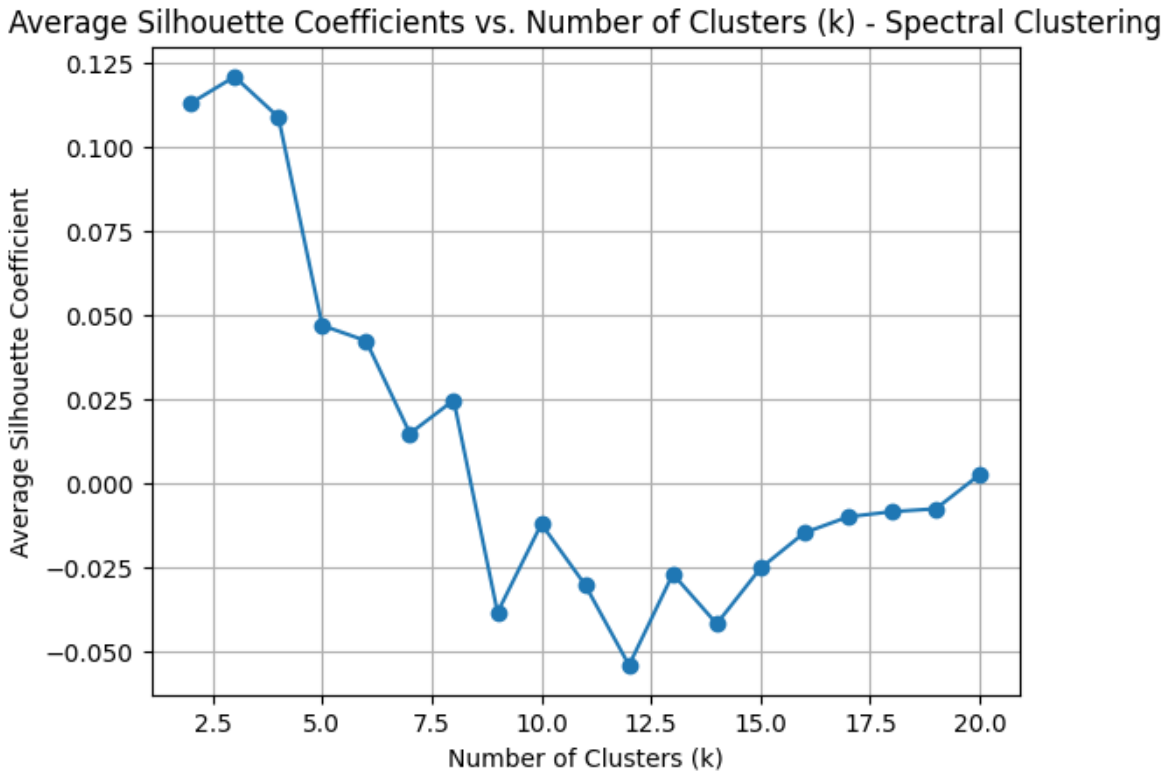


Here, a lot of the clusters appear to be overlapping. The best Silhouette coefficient score for k=20 is 0.115 which is less than that of k-means. That is the reason why most of the clusters appear to be overlapping for Gausian mixture model.

2b. Spectral Clustering model

```
from sklearn.cluster import SpectralClustering
```

```
features = data.iloc[:, 1:]

from sklearn.impute import SimpleImputer

# Replace NaN values with the mean of each column
imputer = SimpleImputer(strategy="mean")
features = imputer.fit_transform(features)

# Range of k values
k_values = range(2, 21)
silhouette_scores_spectral = []

# Compute Silhouette coefficient for different k values
for k in k_values:
    avg_silhouette_score = 0
    for _ in range(5):  # Run Spectral Clustering 5 times for each k
        spectral = SpectralClustering(n_clusters=k, n_init=10, affinity='nearest_neighbors
        cluster_labels = spectral.fit_predict(features)
        avg_silhouette_score += silhouette_score(features, cluster_labels)
    avg_silhouette_score /= 5  # Average over 5 runs
    silhouette_scores_spectral.append(avg_silhouette_score)

# Plot the average Silhouette coefficients for different k
plt.plot(k_values, silhouette_scores_spectral, marker='o')
plt.title("Average Silhouette Coefficients vs. Number of Clusters (k) - Spectral Clusterin
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Average Silhouette Coefficient")
plt.grid(True)
plt.show()

# The best k is where Silhouette coefficient is highest
best_k_spectral = k_values[np.argmax(silhouette_scores_spectral)]
print("Best k (Spectral Clustering):", best_k_spectral)
```



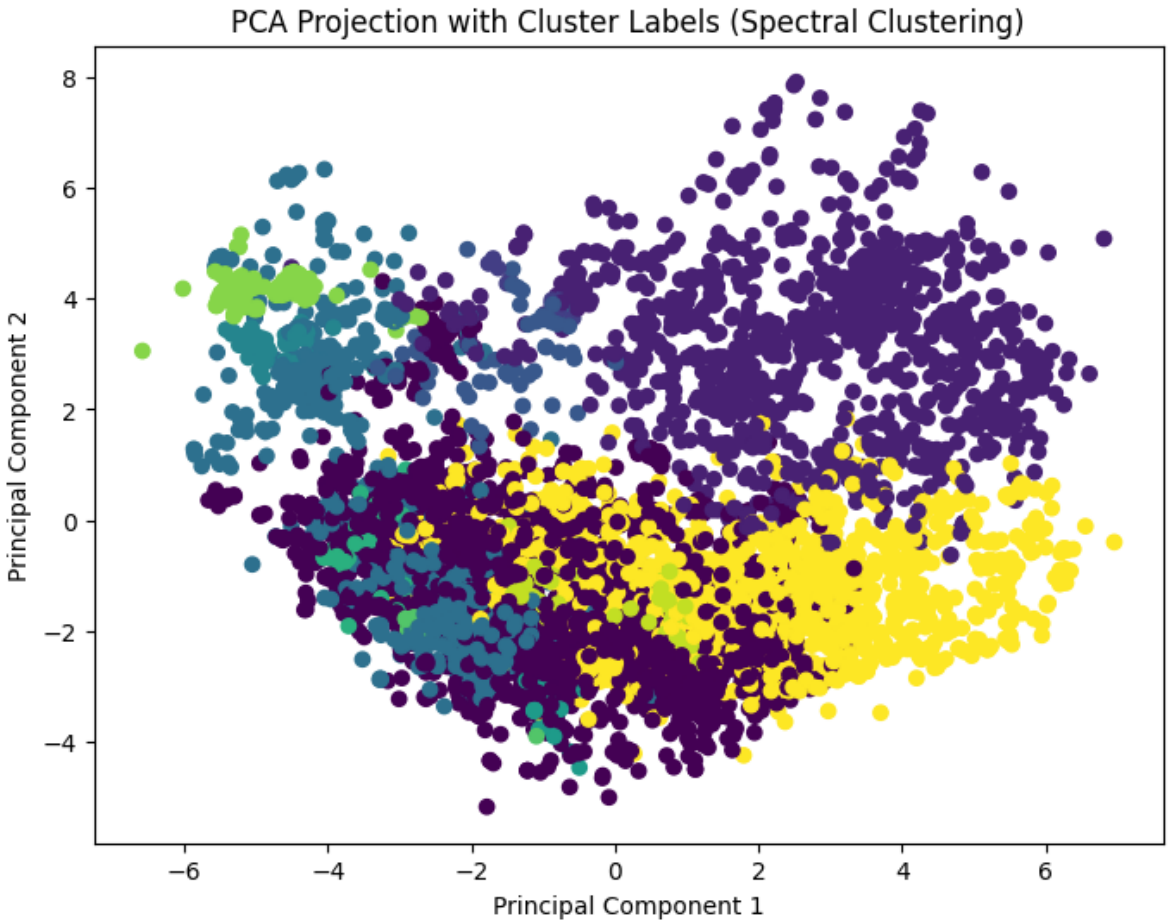

```
Best k (Spectral Clustering): 3
```

```
best_spectral = SpectralClustering(n_clusters=best_k, n_init=10, affinity='nearest_neighbo
cluster_labels = best_spectral.fit_predict(features)
cluster_centers = np.array([np.mean(features[cluster_labels == i], axis=0) for i in range(
print("Cluster centers (Spectral Clustering):")
print(cluster_centers)
```

```
Cluster centers (Spectral Clustering):
[[0.97614334 1.24300341 1.55257679 1.66467577 1.82385666 2.09703072
  3.16220137 3.00982935 3.76459044 3.8640273  4.46735495 3.10250853
  2.88196246 2.70675768 1.81003413 0.93549488 0.87141638 0.62136519
  0.90546075 0.66569966 0.41421502 0.97447099 0.70783276 0.94726962]
 [1.214414   2.66711821 2.80167428 2.79768645 3.14317605 3.45446981
  4.15885337 3.04369356 3.76927955 3.21542872 2.34634196 1.8134348
  1.84604769 2.4523795  2.52853881 1.0580619  0.5906342  0.47442922
  0.53394216 0.69893455 0.79507864 0.9078691  0.96032471 1.06998985]
 [0.35263158 0.44917293 1.45909774 1.49165414 1.48315789 1.49195489
```

```
 1.58684211 1.54496241 2.63       2.96736842 3.18270677 4.96887218
 4.71984962 5.         5.         0.77586466 0.77458647 1.05308271
 0.86323308 0.70030075 0.36488722 0.32210526 0.30225564 0.30646617]
[1.58371083 2.17448006 2.68268519 3.92533476 4.16742877 3.37609687
 3.17197293 2.4585114  2.7124359  2.72350427 2.74110399 2.11641738
 2.32486467 1.42651709 1.21881054 1.0475641  0.84245014 0.68315527
 0.6565812  0.76492165 0.86264957 2.89801994 2.3070949  2.03594017]
[2.2617052  2.7244027  2.54289017 2.30401734 2.08103083 1.9106262
 1.97033719 1.59552987 1.9172736  1.5811657  1.56303468 1.45315992
 1.55238921 1.91142582 2.36578035 1.67519268 1.71023121 1.81760116
 2.37491329 2.20600193 1.79404624 2.62992293 2.3350578  2.50574227]
[1.46101449 1.63478261 1.90376812 1.84405797 1.84623188 1.85927536
 3.83318841 1.93927536 2.61115942 2.32956522 2.28086957 2.49478261
 2.57753623 3.95507246 4.36043478 5.         3.60782609 0.10434783
 0.38797101 0.52391304 1.44811594 1.44043478 1.41797101 1.43463768]
[1.68166667 1.89666667 2.1475     1.89916667 1.89       1.9025
 5.         1.9775     2.48083333 2.115      2.10166667 2.1275
 2.37833333 3.6        5.         5.         5.         4.93333333
 0.12666667 0.74       1.61083333 1.5975     1.5875     1.6       ]
[1.345      1.38125    1.5775     5.         2.33875    3.22166667
 4.85208333 3.83625    5.         5.         1.60666667 1.54041667
 1.45458333 1.24791667 0.86541667 0.60708333 0.23875    0.195
 0.1975     0.55125    0.70291667 1.11041667 1.33958333 1.33083333]
[0.89606557 1.05180328 1.56770492 1.61409836 1.60655738 1.62852459
 5.         2.06655738 2.51590164 2.82672131 2.85360656 5.
 4.7347541  5.         5.         0.66967213 0.65540984 0.66245902
 0.84131148 0.86672131 0.8695082  0.85459016 0.84213115 0.89196721]
[0.89394366 4.98225352 1.48338028 1.51774648 1.73267606 2.23295775
 4.85084507 3.62084507 4.98126761 5.         3.10380282 2.08676056
 1.57140845 1.35       4.96323944 0.65802817 0.5584507  0.56521127
 0.77591549 0.76422535 0.85816901 0.89070423 0.84788732 0.85802817]
[0.84384615 4.99       1.38692308 1.52538462 1.60384615 1.63307692
 5.         4.18461538 5.         5.         5.         2.04461538
 1.55461538 1.37923077 1.10153846 0.65384615 0.63       0.63461538
 0.66307692 0.79153846 1.01846154 0.97       0.82153846 0.82538462]
[1.03148649 1.06635135 1.83567568 2.33594595 3.02013514 4.32445946
 5.         1.34       5.         1.28621622 1.27202703 1.2827027
 1.46527027 1.52567568 1.42554054 0.61864865 0.60040541 0.615
 0.78945946 0.76567568 1.0072973  0.99108108 0.97878378 0.99364865]]
```

```
from sklearn.decomposition import PCA

# Project the data using PCA with two principal components
pca = PCA(n_components=2)
projected_data_spectral = pca.fit_transform(features)

# Draw a scatter plot with cluster IDs as labels
cluster_labels_spectral = best_spectral.fit_predict(features)

# Draw a scatter plot with cluster IDs as labels
plt.figure(figsize=(8, 6))
plt.scatter(projected_data_spectral[:, 0], projected_data_spectral[:, 1], c=cluster_labels
plt.title("PCA Projection with Cluster Labels (Spectral Clustering)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

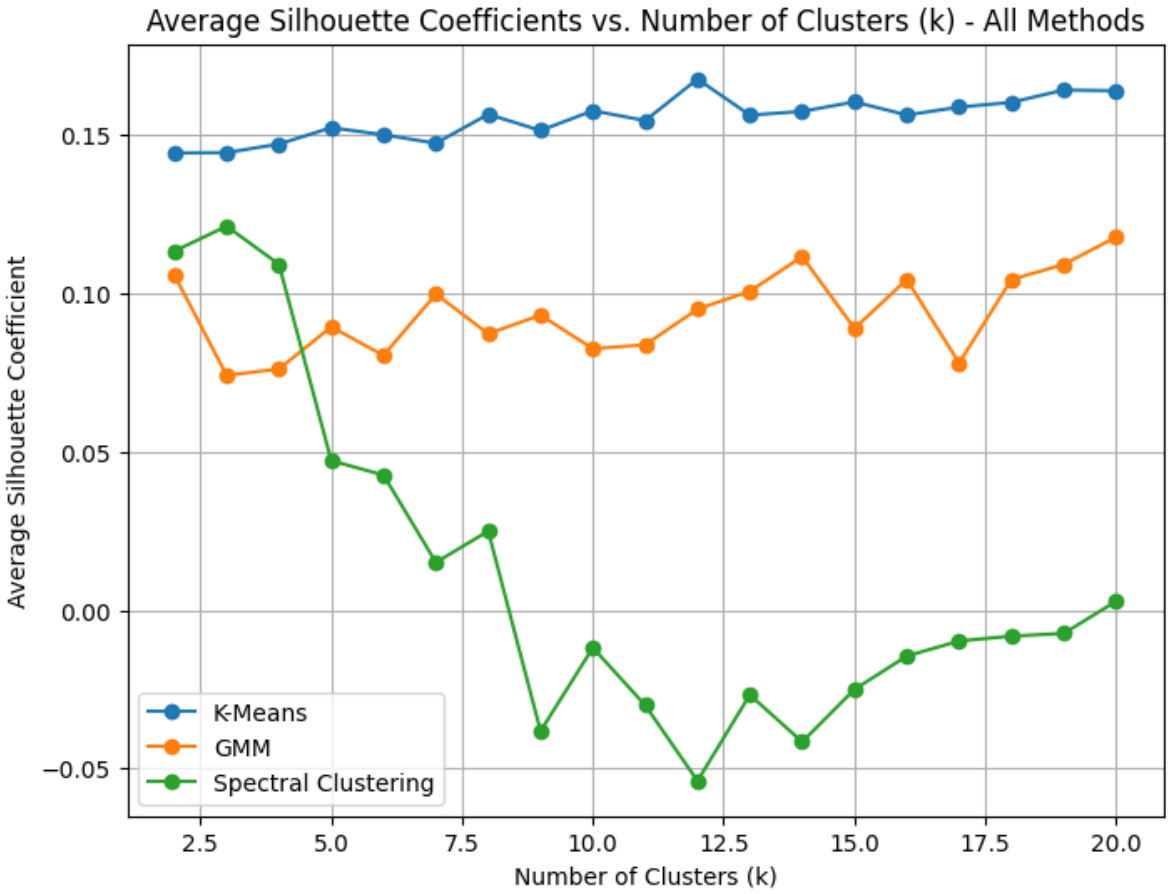


Here, the clusters at the right side and top-left corners appear to be well separated keeping some outlying points out. Some clusters of bottom-left corner appear to be overlapping. The best Silhouette coefficient score for k=3 is 0.125. The much silhouette coefficient will be, clusters will be more separated and consistent.

2c. Best method in terms of the average Silhouette coefficient

```
# Plot the average Silhouette coefficients for different k for all methods
plt.figure(figsize=(8, 6))
plt.plot(k_values, silhouette_scores_kmeans, marker='o', label='K-Means')
plt.plot(k_values, silhouette_scores_gmm, marker='o', label='GMM')
plt.plot(k_values, silhouette_scores_spectral, marker='o', label='Spectral Clustering')
plt.title("Average Silhouette Coefficients vs. Number of Clusters (k) - All Methods")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Average Silhouette Coefficient")
```

```
plt.legend()
plt.grid(True)
plt.show()

# Determine which method has the highest average Silhouette coefficient
best_method = np.argmax([max(silhouette_scores_kmeans), max(silhouette_scores_gmm), max(si
if best_method == 0:
    print("K-Means has the highest average Silhouette coefficient.")
elif best_method == 1:
    print("GMM has the highest average Silhouette coefficient.")
else:
    print("Spectral Clustering has the highest average Silhouette coefficient.")
```




```
K-Means has the highest average Silhouette coefficient.
```

Question 3. (DBSCAN)

3a. Computing best Silhouette coefficient

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
from itertools import product

# Define a range of values for eps and min_samples
eps_values = [0.5, 1.0, 1.5, 2.0, 2.5, 3.0]
min_samples_values = [2, 5, 10, 15, 20]

best_score = -1
best_eps = None
best_min_samples = None

for eps, min_samples in product(eps_values, min_samples_values):
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    cluster_labels = dbscan.fit_predict(features)
    score = silhouette_score(features, cluster_labels)

    if score > best_score:
        best_score = score
        best_eps = eps
        best_min_samples = min_samples

print("Best Silhouette Score:", best_score)
print("Best eps:", best_eps)
print("Best min_samples:", best_min_samples)
```

```
Best Silhouette Score: 0.26293079484588383
Best eps: 2.0
Best min_samples: 2
```

3b. Clustering with best DBSCAN model

```
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Train DBSCAN with optimized hyperparameters
dbscan = DBSCAN(eps=best_eps, min_samples=best_min_samples)
```

```
cluster_labels = dbscan.fit_predict(features)

# Project the data using PCA
pca = PCA(n_components=2)
projected_data = pca.fit_transform(features)

# Draw a scatter plot with cluster IDs as labels
plt.figure(figsize=(8, 6))
plt.scatter(projected_data[:, 0], projected_data[:, 1], c=cluster_labels, cmap='viridis')
plt.title("PCA Projection with Cluster Labels (DBSCAN Clustering)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```