Clustering

1. DBSCAN

Using DBSCAN iterate (for-loop) through different values of `min_samples` (1 to 10) and `epsilon` (.05 to .5, in steps of .01) to find clusters in the road-data used in the Lesson and calculate the Silohouette Coeff for `min_samples` and `epsilon`. Plot ***one*** line plot with the multiple lines generated from the min_samples and epsilon values. Use a 2D array to store the SilCoeff values, one dimension represents `min_samples`, the other represents epsilon.

Expecting a plot of `epsilon` vs `sil_score`.

In [97]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn import metrics
plt.rcParams['font.size'] = 14
plt.rcParams['figure.figsize'] = (20.0, 10.0)
```

In [113…

```
X = pd.read_csv('../data/3D_spatial_network.txt.gz', header=None, names=['osm', 'lat','l
X = X.drop(['osm'], axis=1).sample(10000)
X.head()
```

Out[113]:

	lat	lon	alt
114939	10.400343	57.582848	1.507983
264146	10.515504	57.445018	20.212532
132821	10.167036	56.783060	16.480972
231108	9.928222	57.043775	2.791583
393317	10.103037	57.239930	25.387568

In [114…

```
XX = X.copy()
XX['alt'] = (X.alt - X.alt.mean())/X.alt.std()
XX['lat'] = (X.lat - X.lat.mean())/X.lat.std()
XX['lon'] = (X.lon - X.lon.mean())/X.lon.std()
XX.head()
```

Out[114]:

	lat	lon	alt
114939	1.061388	1.742729	-1.110135
264146	1.244744	1.263748	-0.124896
132821	0.689920	-1.036656	-0.321451
231108	0.309687	-0.130632	-1.042523
393317	0.588024	0.551036	0.147693

In [115…

```
min_samples = np.arange(1, 11, 1)
epsilons = np.arange(.05, .51, .01)
```

In [116…
```
min_samples
```

Out[116]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [117…
```
epsilons
```

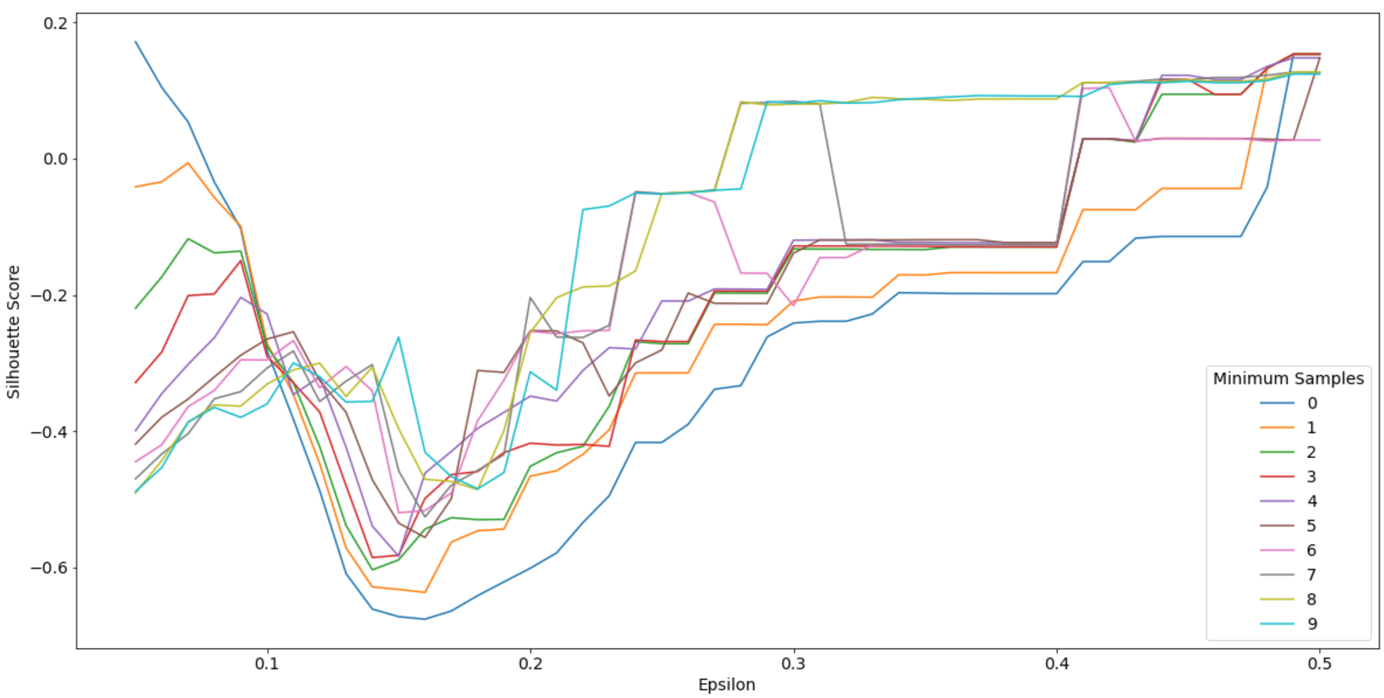
Out[117]:
```
array([0.05, 0.06, 0.07, 0.08, 0.09, 0.1 , 0.11, 0.12, 0.13, 0.14, 0.15,
       0.16, 0.17, 0.18, 0.19, 0.2 , 0.21, 0.22, 0.23, 0.24, 0.25, 0.26,
       0.27, 0.28, 0.29, 0.3 , 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37,
       0.38, 0.39, 0.4 , 0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48,
       0.49, 0.5 ])
```

In [123…
```
all_scores = []
for min_sample in min_samples:
    scores = []
    for epsilon in epsilons:
        dbscan=DBSCAN(eps=epsilon, min_samples=min_sample)
        labels = dbscan.fit_predict(XX[['lat','lon', 'alt']])
        # calculate silouette score here
        score = metrics.silhouette_score(XX[['lat', 'lon', 'alt']], labels)

        scores.append(score)

    all_scores.append(scores)
```

In [124…
```
len(all_scores)
```

Out[124]:
```
10
```

In [127…
```
plt.figure()
for m in range(len(min_samples)):
    plt.plot(epsilons,all_scores[m],label=m)
plt.legend(title='Minimum Samples',loc='lower right')
plt.xlabel('Epsilon')
plt.ylabel('Silhouette Score')
plt.show()
```



2. Clustering your own data

Using your own data, find relevant clusters/groups within your data (repeat the above). If your data is labeled with a class that you are attempting to predict, be sure to not use it in training and clustering.

You may use the labels to compare with predictions to show how well the clustering performed using one of the clustering metrics (http://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation).

If you don't have labels, use the silhouette coefficient to show performance. Find the optimal fit for your data but you don't need to be as exhaustive as above.

Additionally, show the clusters in 2D or 3D plots.

As a bonus, try using PCA first to condense your data from N columns to less than N.

Two items are expected:

- Metric Evaluation Plot (like in 1.)
- Plots of the clustered data

In [128…

```
wine = pd.read_csv('../add_data/wine/wine.data',
                   names = ['class','alcohol','malic acid','ash','alc of ash',
                            'mag','phenols','flavan','nonflav phenols','proanth',
                            'color inten','hue','OD280','proline'])
wine['class'].value_counts()
```

Out[128]:

```
2    71
1    59
3    48
Name: class, dtype: int64
```

In [129…

```
labels_test = wine['class']
xx = wine.copy().drop('class',axis=1)
len(xx)
```

Out[129]:

```
178
```

In [130…

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
xx_s = StandardScaler().fit_transform(xx)
xx_s
```

Out[130]:

```
array([[ 1.51861254, -0.5622498 ,  0.23205254, ...,  0.36217728,
         1.84791957,  1.01300893],
       [ 0.24628963, -0.49941338, -0.82799632, ...,  0.40605066,
         1.1134493 ,  0.96524152],
       [ 0.19687903,  0.02123125,  1.10933436, ...,  0.31830389,
         0.78858745,  1.39514818],
       ...,
       [ 0.33275817,  1.74474449, -0.38935541, ..., -1.61212515,
        -1.48544548,  0.28057537],
       [ 0.20923168,  0.22769377,  0.01273209, ..., -1.56825176,
        -1.40069891,  0.29649784],
       [ 1.39508604,  1.58316512,  1.36520822, ..., -1.52437837,
        -1.42894777, -0.59516041]])
```

In [131…

```
pca = PCA(n_components=2)
xx_r = pca.fit_transform(xx_s)
xx_r
```

Out[131]:

```
array([[ 3.31675081, -1.44346263],
       [ 2.20946492,  0.33339289],
       [ 2.51674015, -1.0311513 ],
```

```
[ 3.75706561, -2.75637191],
[ 1.00890849, -0.86983082],
[ 3.05025392, -2.12240111],
[ 2.44908967, -1.17485013],
[ 2.05943687, -1.60896307],
[ 2.5108743 , -0.91807096],
[ 2.75362819, -0.78943767],
[ 3.47973668, -1.30233324],
[ 1.7547529 , -0.61197723],
[ 2.11346234, -0.67570634],
[ 3.45815682, -1.13062988],
[ 4.31278391, -2.09597558],
[ 2.3051882 , -1.66255173],
[ 2.17195527, -2.32730534],
[ 1.89897118, -1.63136888],
[ 3.54198508, -2.51834367],
[ 2.0845222 , -1.06113799],
[ 3.12440254, -0.78689711],
[ 1.08657007, -0.24174355],
[ 2.53522408,  0.09184062],
[ 1.64498834,  0.51627893],
[ 1.76157587,  0.31714893],
[ 0.9900791 , -0.94066734],
[ 1.77527763, -0.68617513],
[ 1.23542396,  0.08980704],
[ 2.18840633, -0.68956962],
[ 2.25610898, -0.19146194],
[ 2.50022003, -1.24083383],
[ 2.67741105, -1.47187365],
[ 1.62857912, -0.05270445],
[ 1.90269086, -1.63306043],
[ 1.41038853, -0.69793432],
[ 1.90382623, -0.17671095],
[ 1.38486223, -0.65863985],
[ 1.12220741, -0.11410976],
[ 1.5021945 ,  0.76943201],
[ 2.52980109, -1.80300198],
[ 2.58809543, -0.7796163 ],
[ 0.66848199, -0.16996094],
[ 3.07080699, -1.15591896],
[ 0.46220914, -0.33074213],
[ 2.10135193,  0.07100892],
[ 1.13616618, -1.77710739],
[ 2.72660096, -1.19133469],
[ 2.82133927, -0.6462586 ],
[ 2.00985085, -1.24702946],
[ 2.7074913 , -1.75196741],
[ 3.21491747, -0.16699199],
[ 2.85895983, -0.7452788 ],
[ 3.50560436, -1.61273386],
[ 2.22479138, -1.875168  ],
[ 2.14698782, -1.01675154],
[ 2.46932948, -1.32900831],
[ 2.74151791, -1.43654878],
[ 2.17374092, -1.21219984],
[ 3.13938015, -1.73157912],
[-0.92858197,  3.07348616],
[-1.54248014,  1.38144351],
[-1.83624976,  0.82998412],
[ 0.03060683,  1.26278614],
[ 2.05026161,  1.9250326 ],
[-0.60968083,  1.90805881],
[ 0.90022784,  0.76391147],
[ 2.24850719,  1.88459248],
[ 0.18338403,  2.42714611],
[-0.81280503,  0.22051399],
```

```
[ 1.9756205 ,  1.40328323],
[-1.57221622,  0.88498314],
[ 1.65768181,  0.9567122 ],
[-0.72537239,  1.0636454 ],
[ 2.56222717, -0.26019855],
[ 1.83256757,  1.2878782 ],
[-0.8679929 ,  2.44410119],
[ 0.3700144 ,  2.15390698],
[-1.45737704,  1.38335177],
[ 1.26293085,  0.77084953],
[ 0.37615037,  1.0270434 ],
[ 0.7620639 ,  3.37505381],
[ 1.03457797,  1.45070974],
[-0.49487676,  2.38124353],
[-2.53897708,  0.08744336],
[ 0.83532015,  1.47367055],
[ 0.78790461,  2.02662652],
[-0.80683216,  2.23383039],
[-0.55804262,  2.37298543],
[-1.11511104,  1.80224719],
[-0.55572283,  2.65754004],
[-1.34928528,  2.11800147],
[-1.56448261,  1.85221452],
[-1.93255561,  1.55949546],
[ 0.74666594,  2.31293171],
[ 0.95745536,  2.22352843],
[ 2.54386518, -0.16927402],
[-0.54395259,  0.36892655],
[ 1.03104975,  2.56556935],
[ 2.25190942,  1.43274138],
[ 1.41021602,  2.16619177],
[ 0.79771979,  2.3769488 ],
[-0.54953173,  2.29312864],
[-0.16117374,  1.16448332],
[-0.65979494,  2.67996119],
[ 0.39235441,  2.09873171],
[-1.77249908,  1.71728847],
[-0.36626736,  2.1693533 ],
[-1.62067257,  1.35558339],
[ 0.08253578,  2.30623459],
[ 1.57827507,  1.46203429],
[ 1.42056925,  1.41820664],
[-0.27870275,  1.93056809],
[-1.30314497,  0.76317231],
[-0.45707187,  2.26941561],
[-0.49418585,  1.93904505],
[ 0.48207441,  3.87178385],
[-0.25288888,  2.82149237],
[-0.10722764,  1.92892204],
[-2.4330126 ,  1.25714104],
[-0.55108954,  2.22216155],
[ 0.73962193,  1.40895667],
[ 1.33632173, -0.25333693],
[-1.177087  ,  0.66396684],
[-0.46233501,  0.61828818],
[ 0.97847408,  1.4455705 ],
[-0.09680973,  2.10999799],
[ 0.03848715,  1.26676211],
[-1.5971585 ,  1.20814357],
[-0.47956492,  1.93884066],
[-1.79283347,  1.1502881 ],
[-1.32710166, -0.17038923],
[-2.38450083, -0.37458261],
[-2.9369401 , -0.26386183],
[-2.14681113, -0.36825495],
[-2.36986949,  0.45963481],
```

```
       [-3.06384157, -0.35341284],
       [-3.91575378, -0.15458252],
       [-3.93646339, -0.65968723],
       [-3.09427612, -0.34884276],
       [-2.37447163, -0.29198035],
       [-2.77881295, -0.28680487],
       [-2.28656128, -0.37250784],
       [-2.98563349, -0.48921791],
       [-2.3751947 , -0.48233372],
       [-2.20986553, -1.1600525 ],
       [-2.625621  , -0.56316076],
       [-4.28063878, -0.64967096],
       [-3.58264137, -1.27270275],
       [-2.80706372, -1.57053379],
       [-2.89965933, -2.04105701],
       [-2.32073698, -2.35636608],
       [-2.54983095, -2.04528309],
       [-1.81254128, -1.52764595],
       [-2.76014464, -2.13893235],
       [-2.7371505 , -0.40988627],
       [-3.60486887, -1.80238422],
       [-2.889826  , -1.92521861],
       [-3.39215608, -1.31187639],
       [-1.0481819 , -3.51508969],
       [-1.60991228, -2.40663816],
       [-3.14313097, -0.73816104],
       [-2.2401569 , -1.17546529],
       [-2.84767378, -0.55604397],
       [-2.59749706, -0.69796554],
       [-2.94929937, -1.55530896],
       [-3.53003227, -0.8825268 ],
       [-2.40611054, -2.59235618],
       [-2.92908473, -1.27444695],
       [-2.18141278, -2.07753731],
       [-2.38092779, -2.58866743],
       [-3.21161722,  0.2512491 ],
       [-3.67791872, -0.84774784],
       [-2.4655558 , -2.1937983 ],
       [-3.37052415, -2.21628914],
       [-2.60195585, -1.75722935],
       [-2.67783946, -2.76089913],
       [-2.38701709, -2.29734668],
       [-3.20875816, -2.76891957]])
```

In [132…

```
xx_df = pd.DataFrame(xx_r,columns=['pca1','pca2'])
xx_df
```

Out[132]:

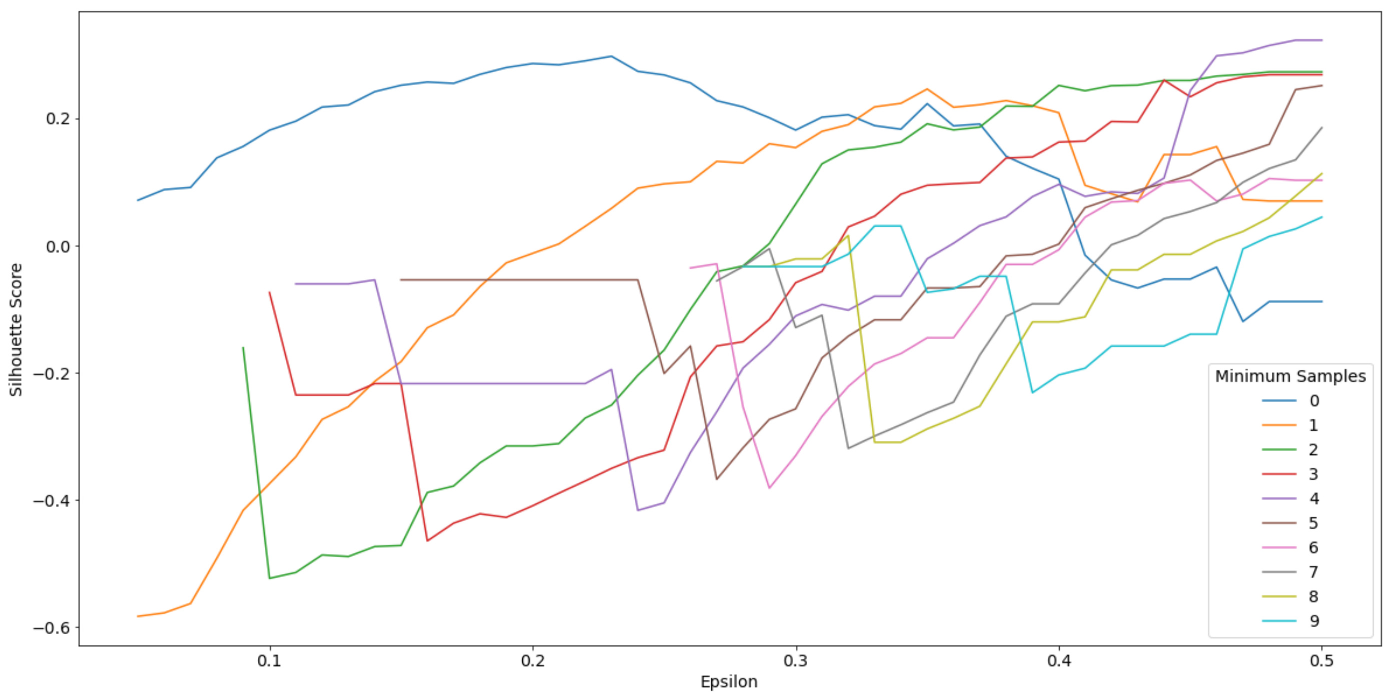
	pca1	pca2
0	3.316751	-1.443463
1	2.209465	0.333393
2	2.516740	-1.031151
3	3.757066	-2.756372
4	1.008908	-0.869831
...	...	...
173	-3.370524	-2.216289
174	-2.601956	-1.757229
175	-2.677839	-2.760899
176	-2.387017	-2.297347

177 -3.208758 -2.768920

178 rows × 2 columns

In [133…
```
all_scores = []

for min_sample in min_samples:
    scores = []

    for epsilon in epsilons:

        try:
            predict=DBSCAN(eps=epsilon, min_samples=min_sample).fit_predict(xx_df)
            # calculate silouette score here
            score = metrics.silhouette_score(xx_df, predict)

            scores.append(score)

        except:
            scores.append(np.nan)

    all_scores.append(scores)
```

In [135…
```
plt.figure()
for m in range(len(min_samples)):
    plt.plot(epsilons,all_scores[m],label=m)
plt.legend(title='Minimum Samples',loc='lower right')
plt.xlabel('Epsilon')
plt.ylabel('Silhouette Score')
plt.show()
```



In [136…
```
labels_pred = DBSCAN(eps=0.5,min_samples=4).fit_predict(xx_df)
```

In [137…
```
metrics.rand_score(labels_test, labels_pred)
```

Out[137]: 0.7700120611946931

In [138…
```
metrics.adjusted_rand_score(labels_test, labels_pred)
```

Out[138]: 0.4492086520143072

In [140…

```
xx_df['labels_pred'] = labels_pred
xx_df
```

Out[140]:

	pca1	pca2	labels_pred
0	3.316751	-1.443463	0
1	2.209465	0.333393	0
2	2.516740	-1.031151	0
3	3.757066	-2.756372	-1
4	1.008908	-0.869831	0
...	...	...	...
173	-3.370524	-2.216289	-1
174	-2.601956	-1.757229	4
175	-2.677839	-2.760899	4
176	-2.387017	-2.297347	4
177	-3.208758	-2.768920	-1

178 rows × 3 columns

In [150…

```
np.unique(labels_pred)
```

Out[150]:

```
array([-1,  0,  1,  2,  3,  4])
```

In [154…

```
colors = np.array(['red','orange','gold','green','blue','purple'])
```

In [156…

```
plt.figure()
plt.scatter(xx_df['pca1'],xx_df['pca2'],c=colors[labels_pred])
plt.title('DBSCAN Clustering of Wine Dataset w/ PCA')
plt.show()
```