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 [1]:
```
import numpy as np
from sklearn.preprocessing import OrdinalEncoder
import random
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn import metrics
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (20, 12)
plt.rcParams['font.size'] = 14
import seaborn as sns
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
X = pd.read_csv('../data/3D_spatial_network.txt.gz', header=None, names=['osm', 'lat','lon','alt'])
# X = X.drop(['osm'], axis=1).sample(10000)
X = X.drop(['osm'], axis=1).sample(10000)
X.head()

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()
```

In [3]:
```
XX
```

Out[3]:

	lat	lon	alt
146410	0.257351	-0.204735	1.604755
291655	0.114990	-0.543178	0.831559
406955	0.919020	1.599851	-0.566212
25913	1.106220	1.660781	-1.140015
228435	1.038135	1.820251	-0.423854
...	...	...	...
400563	0.251867	-0.340925	-0.823734
393209	-0.472379	-0.498089	0.214753
385459	-1.431094	-1.023583	-0.641439
340303	1.028803	0.377678	0.210510
74116	1.213256	0.526365	0.130349

10000 rows × 3 columns

In [4]:

```
min_samples = np.arange(1,11,1)
epsilons = np.arange(.05,0.51,.01)
```

In [5]:

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

        scores.append(score)

    all_scores.append(scores)
```

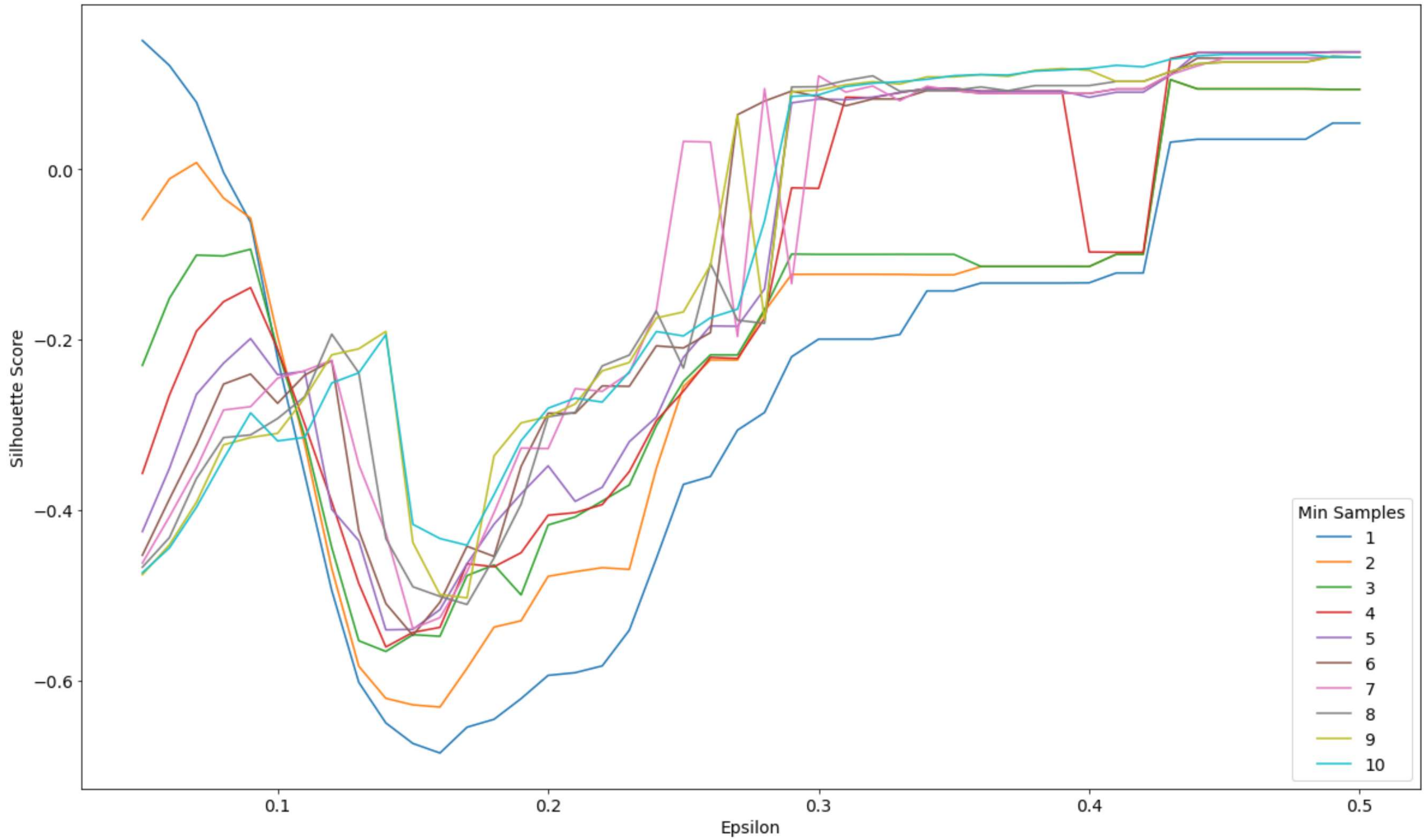
In [6]:

```
plt.figure()
for i in range(len(min_samples)):
    plt.plot(epsilons, all_scores[i], label = f"{i + 1}")
plt.legend(title = "Min Samples",loc = "lower right")
```

```
plt.xlabel("Epsilon")
plt.ylabel("Silhouette Score")
```

Out[6]:

```
Text(0, 0.5, 'Silhouette Score')
```



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 [7]:
```
beer_df = pd.read_csv('beer_reviews.csv')
```

In [8]:
```
beer = beer_df.copy()
beer = beer.dropna()
beer = beer.drop(
    ['brewery_id',
     'brewery_name',
     'review_time',
     'review_profilename',
     'beer_beerid',
     'beer_name'], axis = 1)

beer_styles = list(beer.beer_style.value_counts().nlargest(50).index)
beer_styles = random.choices(beer_styles, k = 10)
beer = beer[beer.beer_style.isin(beer_styles)]
beer = beer.groupby('beer_style').apply(lambda x: x.sample(2000))
# beer = beer.sample(5000)
beer = beer.reset_index(drop = True)
```

In [9]:
```
features = ['review_overall',
 'review_aroma',
 'review_appearance',
 'review_palate',
 'review_taste',
 'beer_abv']

X = beer[features].values
```

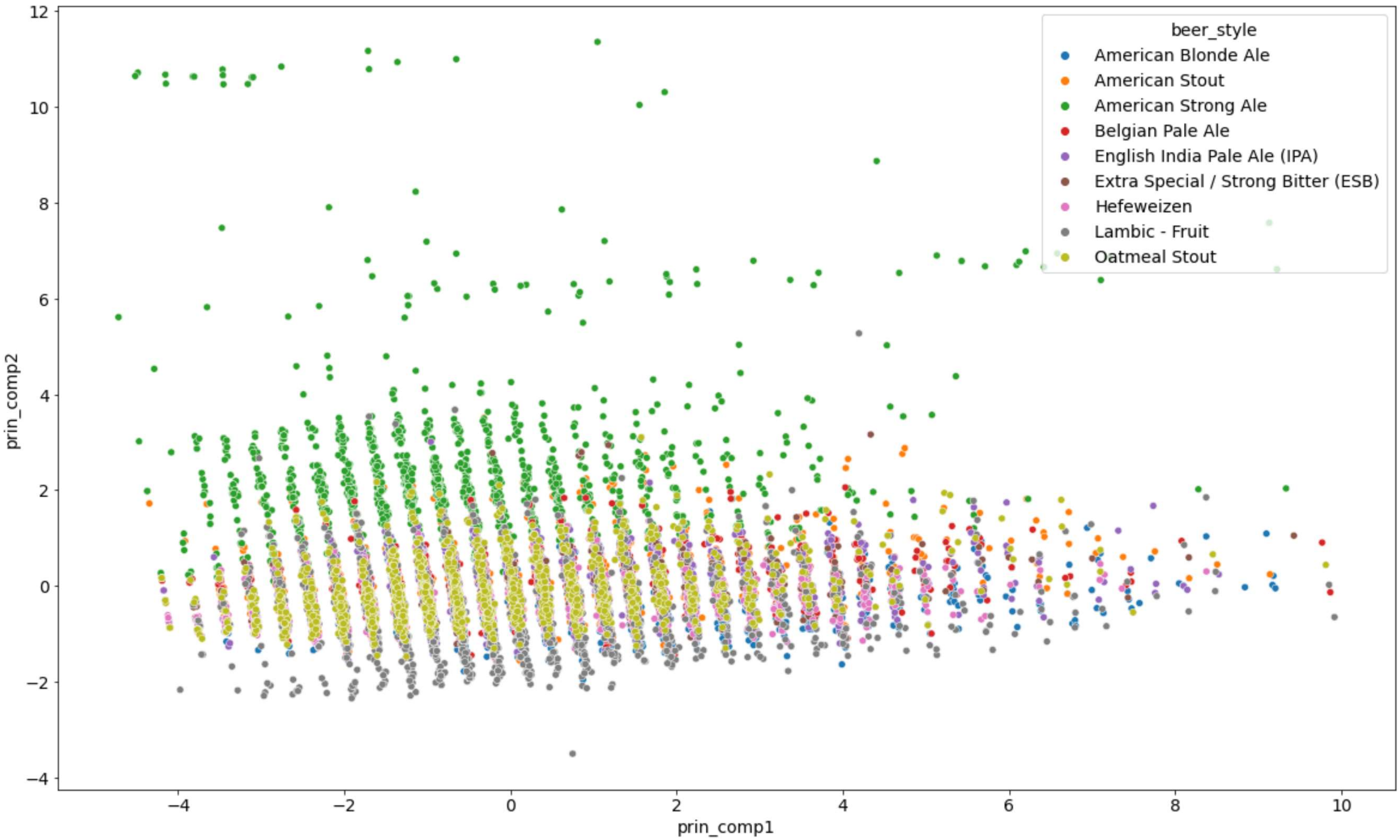
```
X = StandardScaler().fit_transform(X)

pca = PCA(n_components = 2)

prin_comp = pca.fit_transform(X)

principal_df = pd.DataFrame(data = prin_comp,
                            columns = ['prin_comp1', 'prin_comp2'])

finaldf = pd.concat([principal_df, beer[['beer_style']]], axis = 1)

enc = OrdinalEncoder()
labels_true = enc.fit_transform(finaldf[['beer_style']]).reshape(-1)
```

In [10]:
```
print(pca.explained_variance_ratio_)
print(sum(pca.explained_variance_ratio_))
```

```
[0.55935707 0.16698164]
0.726338706694336
```

In [11]:
```
sns.scatterplot(data = finaldf,
                x = 'prin_comp1',
                y = 'prin_comp2', hue = 'beer_style')
```

Out[11]:
```
<AxesSubplot:xlabel='prin_comp1', ylabel='prin_comp2'>
```



In [12]:
```
principal_df
```

Out[12]:

	prin_comp1	prin_comp2
0	-1.104219	0.174459
1	1.498374	-0.658964
2	2.883536	-0.182620
3	1.501711	-0.865520
4	-0.235015	-0.453291
...	...	...
17995	-2.008453	-0.789357
17996	7.494684	-0.552115
17997	-2.043071	-0.426385
17998	-3.117720	-0.237616
17999	-1.017328	0.196983

18000 rows × 2 columns

In [13]:

```
db = DBSCAN(eps = 0.05, min_samples = 1)

labels_pred = db.fit_predict(principal_df)


print(len(set(labels_pred)))
```

```
1436
```

In [14]:

```
min_samples = np.arange(1,11,1)
epsilons = np.arange(.5,1.01,.05)
```

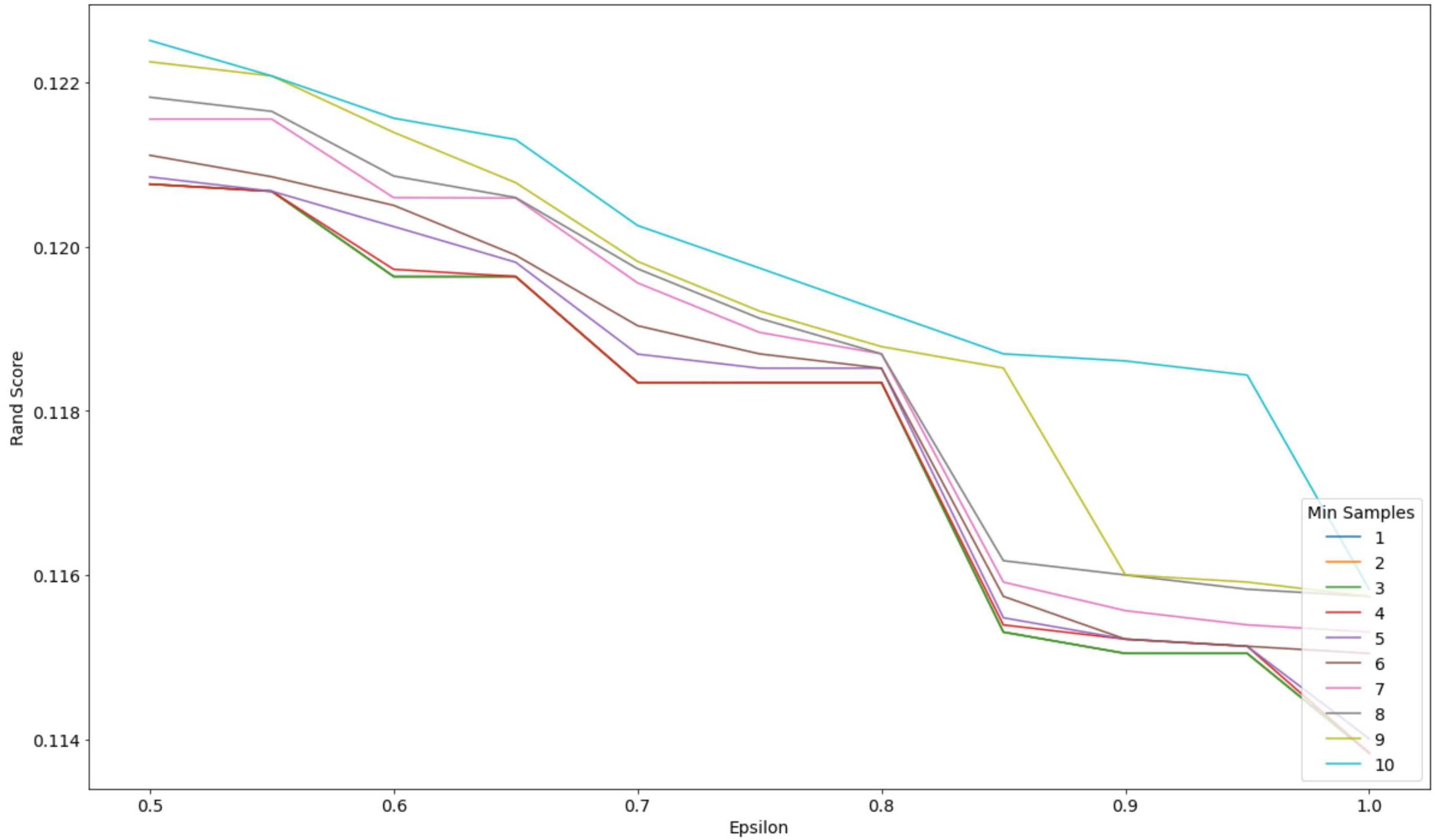
In [16]:

```
all_scores = []
for min_sample in min_samples:
    scores = []
    for epsilon in epsilons:
        db = DBSCAN(eps = epsilon, min_samples=min_sample)
        labels_pred = db.fit_predict(principal_df)
        # calculate silouette score here
        randscore = metrics.rand_score(labels_true, labels_pred)
        adjrandscore = metrics.adjusted_rand_score(labels_true, labels_pred)
```

```
        scores.append(randscore)

    all_scores.append(scores)
```

In [17]:
```
plt.figure()
for i in range(len(min_samples)):
    plt.plot(epsilons, all_scores[i], label = f"{i + 1}")
plt.legend(title = "Min Samples",loc = "lower right")
plt.xlabel("Epsilon")
plt.ylabel("Rand Score")
```

Out[17]:
```
Text(0, 0.5, 'Rand Score')
```



In [18]:
```
db = DBSCAN(eps = 0.5, min_samples = 3)
labels_pred = db.fit_predict(principal_df)
```

In [19]:
```
set(labels_pred)
```

Out[19]:
```
{-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

In [20]:
```
preddf = pd.concat([principal_df, pd.DataFrame(data = labels_pred, columns = ["pred"])], axis = 1)
```

In [21]:
```
preddf
```

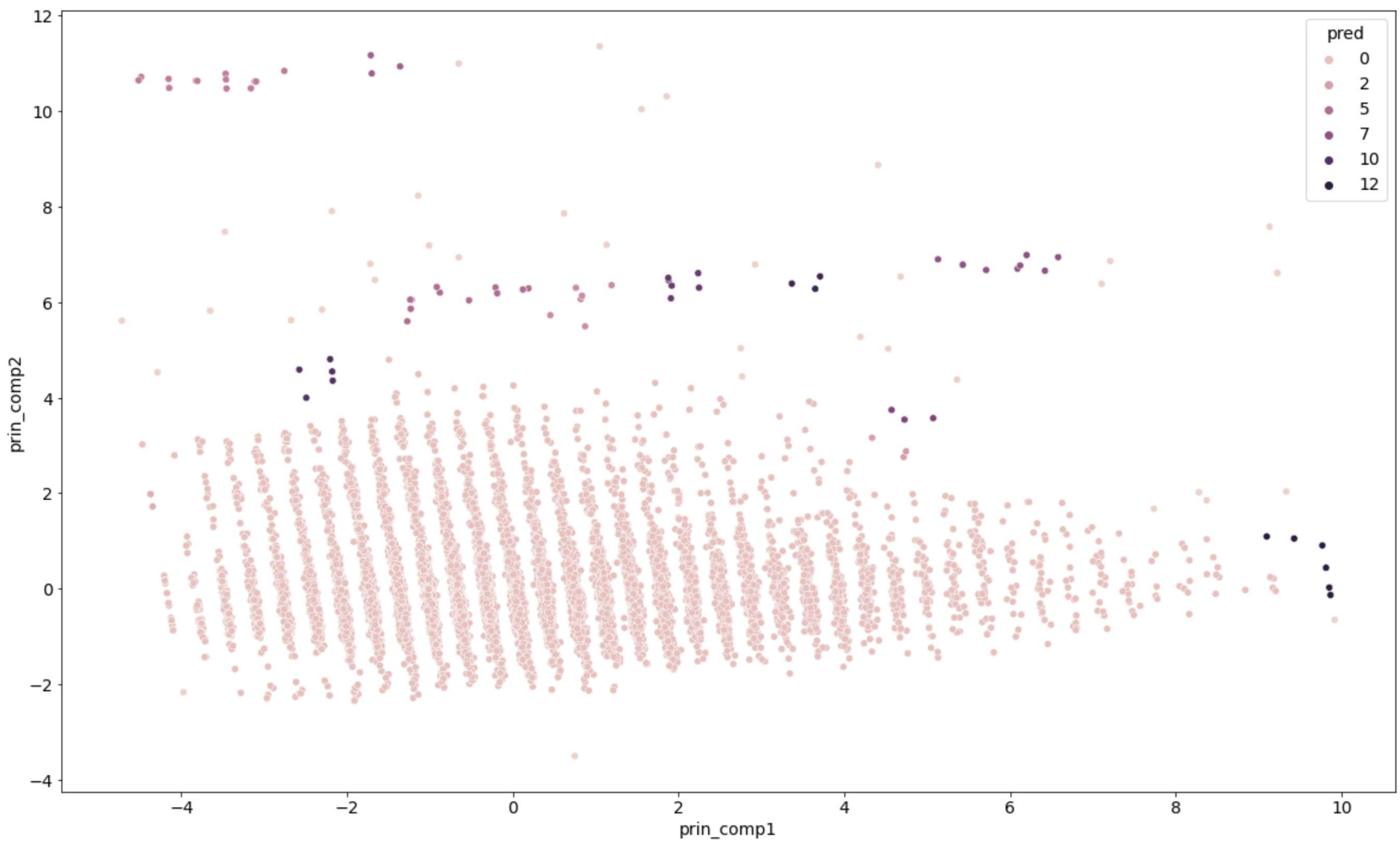
Out[21]:

	prin_comp1	prin_comp2	pred
0	-1.104219	0.174459	0
1	1.498374	-0.658964	0
2	2.883536	-0.182620	0
3	1.501711	-0.865520	0
4	-0.235015	-0.453291	0
...	...	...	...
17995	-2.008453	-0.789357	0
17996	7.494684	-0.552115	0
17997	-2.043071	-0.426385	0
17998	-3.117720	-0.237616	0
17999	-1.017328	0.196983	0

18000 rows × 3 columns

In [22]:
```
sns.scatterplot(
    data = preddf,
    x = "prin_comp1",
    y = "prin_comp2",
    hue = "pred"
)
```

Out[22]: <AxesSubplot:xlabel='prin_comp1', ylabel='prin_comp2'>



In [ ]: