```
import numpy as np
import pandas as pd

x = np.array([[3,7],[4,6],[5,5],[7,5],[8,4],[5,1],[8,3],[9,3],[3,7],[3,5],[2,4],[5,9]])
y = np.array([[3,7],[4,6],[5,5],[7,5],[8,4],[5,1],[8,3],[9,3],[3,7],[3,5],[2,4],[5,9]])

dis = np.zeros((len(x),len(y)))

for i in range(len(x)):
  for j in range(len(y)):
    dis[i][j] = np.sqrt(np.sum((x[i] - y[j])**2))
row_names = ['P{}'.format(i + 1) for i in range(len(x))]
col_names = ['P{}'.format(i + 1) for i in range(len(y))]

dis_df = pd.DataFrame(dis, index=row_names, columns=col_names)

print("Euclidean distance Matrix:")
print(dis_df)
```

```
Euclidean distance Matrix:
           P1        P2        P3        P4        P5        P6        P7  \
P1   0.000000  1.414214  2.828427  4.472136  5.830952  6.324555  6.403124
P2   1.414214  0.000000  1.414214  3.162278  4.472136  5.099020  5.000000
P3   2.828427  1.414214  0.000000  2.000000  3.162278  4.000000  3.605551
P4   4.472136  3.162278  2.000000  0.000000  1.414214  4.472136  2.236068
P5   5.830952  4.472136  3.162278  1.414214  0.000000  4.242641  1.000000
P6   6.324555  5.099020  4.000000  4.472136  4.242641  0.000000  3.605551
P7   6.403124  5.000000  3.605551  2.236068  1.000000  3.605551  0.000000
P8   7.211103  5.830952  4.472136  2.828427  1.414214  4.472136  1.000000
P9   0.000000  1.414214  2.828427  4.472136  5.830952  6.324555  6.403124
P10  2.000000  1.414214  2.000000  4.000000  5.099020  4.472136  5.385165
P11  3.162278  2.828427  3.162278  5.099020  6.000000  4.242641  6.082763
P12  2.828427  3.162278  4.000000  4.472136  5.830952  8.000000  6.708204

           P8        P9       P10       P11       P12
P1   7.211103  0.000000  2.000000  3.162278  2.828427
P2   5.830952  1.414214  1.414214  2.828427  3.162278
P3   4.472136  2.828427  2.000000  3.162278  4.000000
P4   2.828427  4.472136  4.000000  5.099020  4.472136
P5   1.414214  5.830952  5.099020  6.000000  5.830952
P6   4.472136  6.324555  4.472136  4.242641  8.000000
P7   1.000000  6.403124  5.385165  6.082763  6.708204
P8   0.000000  7.211103  6.324555  7.071068  7.211103
P9   7.211103  0.000000  2.000000  3.162278  2.828427
P10  6.324555  2.000000  0.000000  1.414214  4.472136
P11  7.071068  3.162278  1.414214  0.000000  5.830952
P12  7.211103  2.828427  4.472136  5.830952  0.000000
```

```
threshold = 1.9

nearest_points = {}  # Store nearest points for each point

for point in dis_df.index:
    nearest = [col for col in dis_df.columns if col != point and dis_df.loc[point, col] > 0 and dis_df.loc[point, col] < threshold]
    nearest_points[point] = nearest

# Print nearest points for each point
for point, nearest in nearest_points.items():
    print(f"Nearest points to '{point}' are:", nearest)
```

```
Nearest points to 'P1' are: ['P2']
Nearest points to 'P2' are: ['P1', 'P3', 'P9', 'P10']
Nearest points to 'P3' are: ['P2']
Nearest points to 'P4' are: ['P5']
Nearest points to 'P5' are: ['P4', 'P7', 'P8']
Nearest points to 'P6' are: []
Nearest points to 'P7' are: ['P5', 'P8']
Nearest points to 'P8' are: ['P5', 'P7']
Nearest points to 'P9' are: ['P2']
Nearest points to 'P10' are: ['P2', 'P11']
Nearest points to 'P11' are: ['P10']
Nearest points to 'P12' are: []
```

```
nearest_points = {
    'P1': ['P2','P1'],
    'P2': ['P1', 'P3', 'P9', 'P10','P2'],
```

```
    'P3': ['P2','P3'],
    'P4': ['P5','P4'],
    'P5': ['P4', 'P7', 'P8','P5','P5'],
    'P6': ['P6'],
    'P7': ['P5', 'P8','P7'],
    'P8': ['P5', 'P7','P8'],
    'P9': ['P2','P9'],
    'P10': ['P2', 'P11','P10'],
    'P11': ['P10','P11'],
    'P12': ['P12']
}
```

```
# Define the minimum number of points required for a core point
min_points = 4

# Create a set to keep track of core points
core_points = set()

# Iterate through each point in the dictionary
for point, neighbors in nearest_points.items():
    if len(neighbors) >= min_points:
        core_points.add(point)

# Create a set to keep track of border points
border_points = set()

# Create a set to keep track of noise points
noise_points = set()

# Iterate through each point in the dictionary again to classify
for point, neighbors in nearest_points.items():
    if point in core_points:
        print(f"'{point}' is a core point")
    elif any(neigh in core_points for neigh in neighbors):
        border_points.add(point)
        print(f"'{point}' is a border point")
    else:
        noise_points.add(point)
        print(f"'{point}' is a noise point")

# Print the core points, border points, and noise points
print("Core Points:", core_points)
print("Border Points:", border_points)
print("Noise Points:", noise_points)
```

```
'P1' is a border point
'P2' is a core point
'P3' is a border point
'P4' is a border point
'P5' is a core point
'P6' is a noise point
'P7' is a border point
'P8' is a border point
'P9' is a border point
'P10' is a border point
'P11' is a noise point
'P12' is a noise point
Core Points: {'P2', 'P5'}
Border Points: {'P10', 'P9', 'P7', 'P3', 'P1', 'P4', 'P8'}
Noise Points: {'P6', 'P11', 'P12'}
```

```
# Define the points in two clusters and Outliers
cluster1 = [(3,7), (5,5), (3,7), (3,5), (4,6)]
cluster2 = [(7,5), (7,5), (8,3), (9,3), (8,4)]
cluster3=[(5,1),(2,4),(5,9),(2,4),(5,9)]
# Create a plot
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 6))

# Plot points in cluster 1
x1, y1 = zip(*cluster1)
plt.scatter(x1, y1, color='blue', label='Cluster 1')

# Plot points in cluster 2
x2, y2 = zip(*cluster2)
```

```
plt.scatter(x2, y2, color='red', label='Cluster 2')

x2, y2 = zip(*cluster3)
plt.scatter(x2, y2, color='yellow', label='Outliers')


plt.title('Graph of DBScan')
plt.legend()
plt.show()
```