In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score
import seaborn as sns
```

In [ ]:
```
# load the dataset
df = pd.read_csv("binary_class_2d.csv", header=None, names=['X', 'Y', 'Class']
```

In [ ]:
```
df.head()
```

Out[ ]:

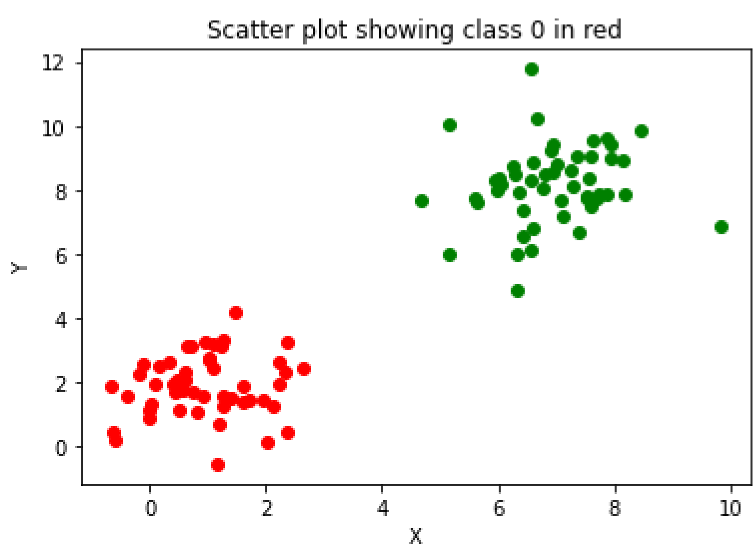
	X	Y	Class
0	-0.590912	0.221098	0
1	-0.366340	1.578768	0
2	1.111379	3.185019	0
3	0.329676	2.633543	0
4	1.259236	3.327122	0

In [ ]:
```
df.shape
```

Out[ ]:
```
(100, 3)
```

In [ ]:
```
# scatter plot with red for class 0 and green for class 1
plt.scatter(df.loc[df['Class'] == 0, 'X'].values, df.loc[df['Class'] == 0, 'Y'
plt.scatter(df.loc[df['Class'] == 1, 'X'].values, df.loc[df['Class'] == 1, 'Y'
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Scatter plot showing class 0 in red")
```

Out[ ]:
```
Text(0.5, 1.0, 'Scatter plot showing class 0 in red')
```



In [ ]:
```
# Part 2 - Implementing GDA from scratch
class GDA():
    def __init__(self):
        self.pi = None
```

```
        self.mu0 = None
        self.mu1 = None
        self.sigma = None

    def train(self, x, y):
        self.pi = np.mean(y)
        self.mu0 = np.mean(x[y[:,0]==0], axis=0)
        # centroid of class 0
        self.mu1 = np.mean(x[y[:,0]==1], axis=0)
        # centroid of class 1
        n_x = x[y[:,0] == 0] - self.mu0
        p_x = x[y[:,0] == 1] - self.mu1
        # calculate sigma and inverse of sigma
        self.sigma = ((n_x.T).dot(n_x) + (p_x.T).dot(p_x))/x.shape[0]
        self.sigma_inv = np.linalg.inv(self.sigma)

    def predict(self, x):
        # maximizing the log likelihood to predict the class
        # p1 >= p0 means 1 else 0
        p0 = -np.sum(np.dot((x-self.mu0),self.sigma_inv)*(x - self.mu0),axis=1
        p1 = -np.sum(np.dot((x-self.mu1),self.sigma_inv)*(x - self.mu1),axis=1
        return p1 >= p0
```

In [ ]:
```
gda = GDA()
```

In [ ]:
```
# separate features and class label
x = df.loc[:, df.columns != 'Class']
y = df['Class'].values.reshape(-1,1)
```

In [ ]:
```
# train GDA
gda.train(x.values, y)
```

In [ ]:
```
# store predictions
df['predictions'] = gda.predict(x)
```

In [ ]:
```
# calculate accuracy score
accuracy_score(df['Class'], df['predictions'])
```

Out[ ]: 1.0

In [ ]:
```
x.head()
```

Out[ ]:

	X	Y
0	-0.590912	0.221098
1	-0.366340	1.578768
2	1.111379	3.185019
3	0.329676	2.633543
4	1.259236	3.327122

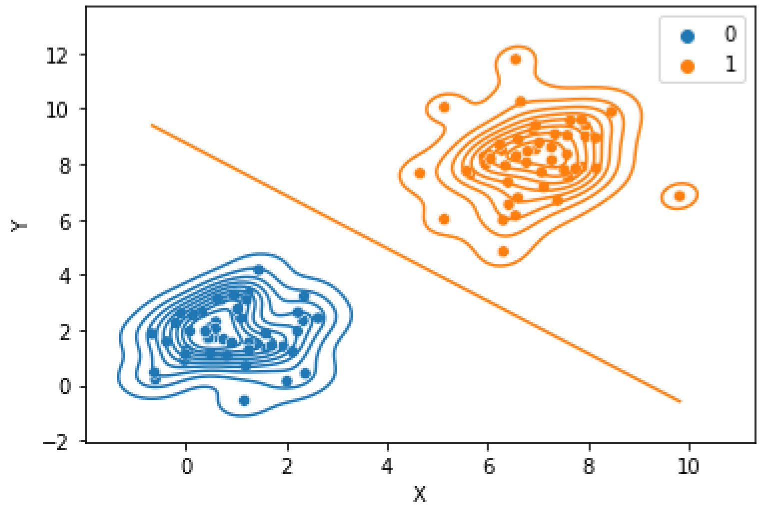
In [ ]:
```
# equation for linear decision boundary
mu1 = gda.mu0
mu2 = gda.mu1
```

```
line_slope = (mu2[1] - mu1[1])/ (mu2[0] - mu1[0])
slope = -1/line_slope
intercept = (mu1[1] + mu2[1])/2 - slope * (mu1[0] + mu2[0])/2

x['line_plot'] = slope*x['X'] + intercept
```

In [ ]:

```
sns.kdeplot(x=x['X'], y=x['Y'], hue=df['Class'])
sns.scatterplot(x=x['X'], y=x['Y'], hue=df['Class'])
sns.lineplot(x=x['X'], y=x['line_plot'])
plt.show()
```



In [ ]: