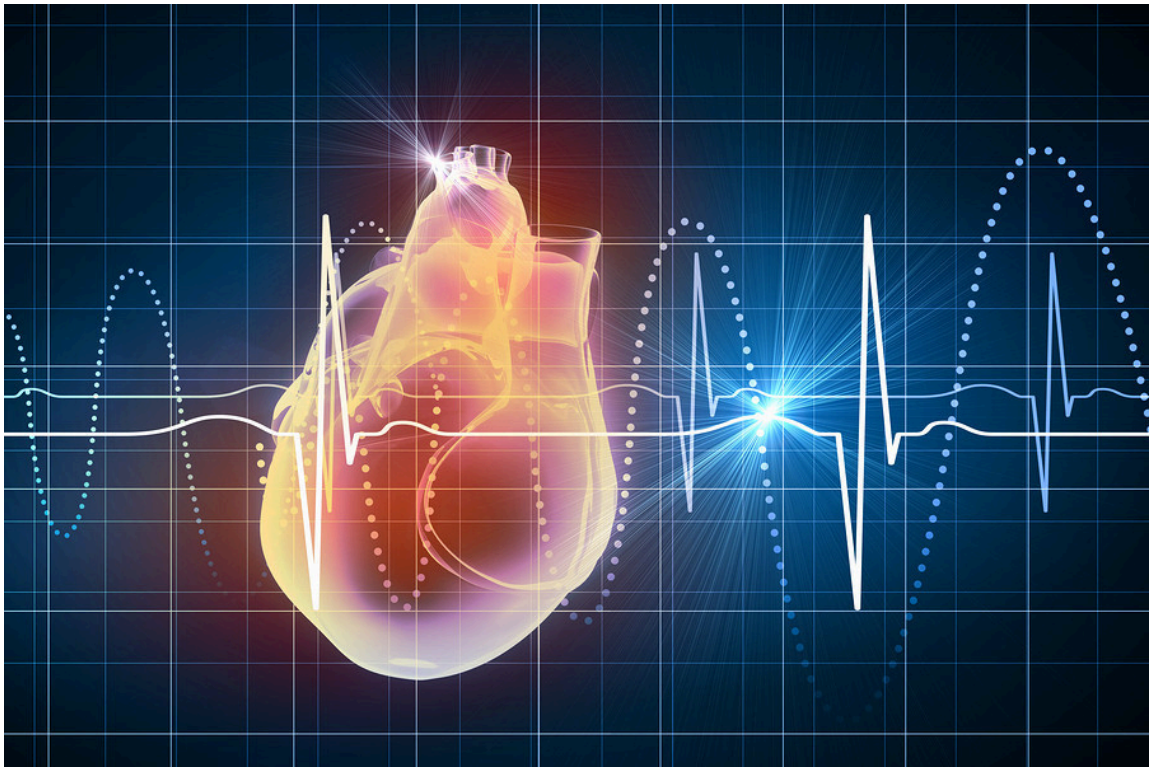
Modelling Heart Failure : Probabilistic Nearest Neighbors Algorithm



Probabilistic Nearest Neighbors Algorithm

Probabilistic Nearest Neighbors (PNN) is a variant of the traditional k-Nearest Neighbors (k-NN) algorithm. While k-NN classifies a data point based on the majority class of its k nearest neighbors, PNN assigns probabilities to each class based on the distribution of neighbors. This approach can provide a more nuanced understanding of class membership, which is particularly useful in cases where the data has overlapping class distributions.

Objectives

The primary objective of this project is to implement a Probabilistic Nearest Neighbors classifier, evaluate its performance on a heart failure patient dataset, and compare it with the traditional k-NN classifier. The project aims to:

- Understand the theoretical foundation of PNN.
- Implement the PNN algorithm in Python using PyTorch.
- Evaluate the algorithm on synthetic and real-world datasets.
- Evaluate the performance of PNN in terms of accuracy, precision, recall, and F1-score.

Theoretical Background

PNN extends the k-NN algorithm by introducing a probabilistic framework. The probability (P(C_i | x)) of a data point (x) belonging to class (C_i) is calculated based on the distances of (x) to its nearest neighbors. This probability can be computed using various methods, such as:

- **Distance-based weighting:** Closer neighbors have more influence.
- **Kernel density estimation:** Estimating the probability density function for each class based on the neighbors' distribution.

Methodology

1. **Data Preparation:**
    - Select or generate synthetic datasets with known class distributions.
    - Use heart failure dataset as a point ofstudy .
2. **Algorithm Implementation:**
    - Implement the traditional k-NN algorithm for baseline comparison.
    - Develop the PNN algorithm in Python using PyTorch, incorporating different probabilistic models.
3. **Model Training and Evaluation:**
    - Split the datasets into training and testing sets.
    - Train the PNN model on the training data.
    - Evaluate the model on the test data using performance metrics (accuracy, precision, recall, F1-score).
4. **Comparison and Analysis:**
    - Analyze the performance of PNN .
    - Analyze the results to understand the strengths and weaknesses of PNN.

In [6]:
```
# Import necessary libraries
import torch
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [13]:
```
df = pd.read_csv("C:\\Datasets\\Heart Failure prediction\\Heart_failure_clinical_re
```

In [14]:
```
df.head()
```

Out[14]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	plat
0	75.0	0	582	0	20	1	26500
1	55.0	0	7861	0	38	0	26335
2	65.0	0	146	0	20	0	16200
3	50.0	1	111	0	20	0	21000
4	65.0	1	160	1	20	0	32700

In [15]:
```
df.isnull().sum()
```

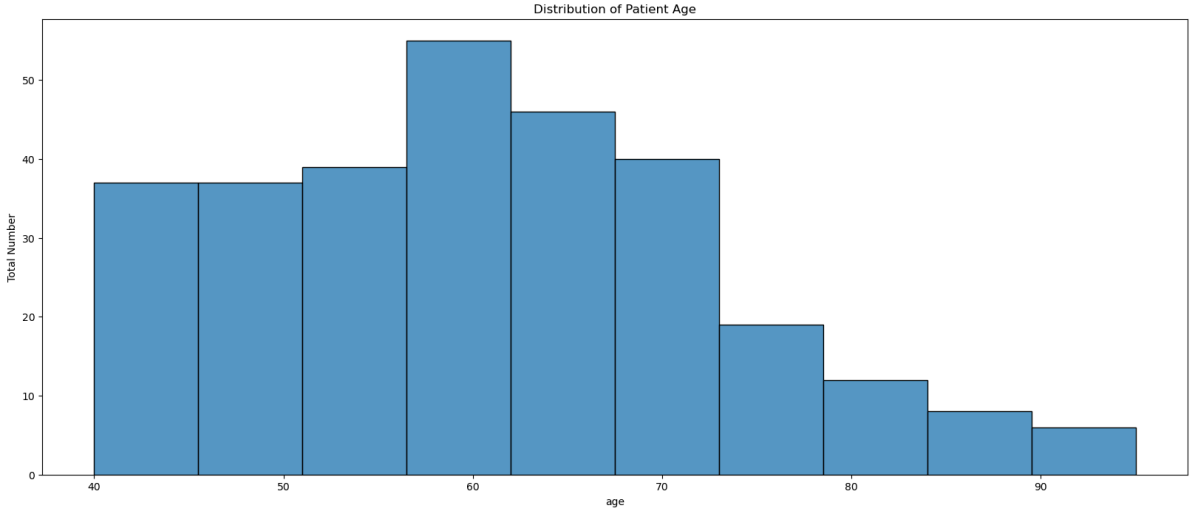
Out[15]:
```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

In [16]:
```
df.columns
```

Out[16]:
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

EDA

In [88]:
```
plt.figure(figsize=(20,8))
sns.histplot(x='age',data=df)
plt.ylabel('Total Number')
plt.title("Distribution of Patient Age")
plt.show()
```

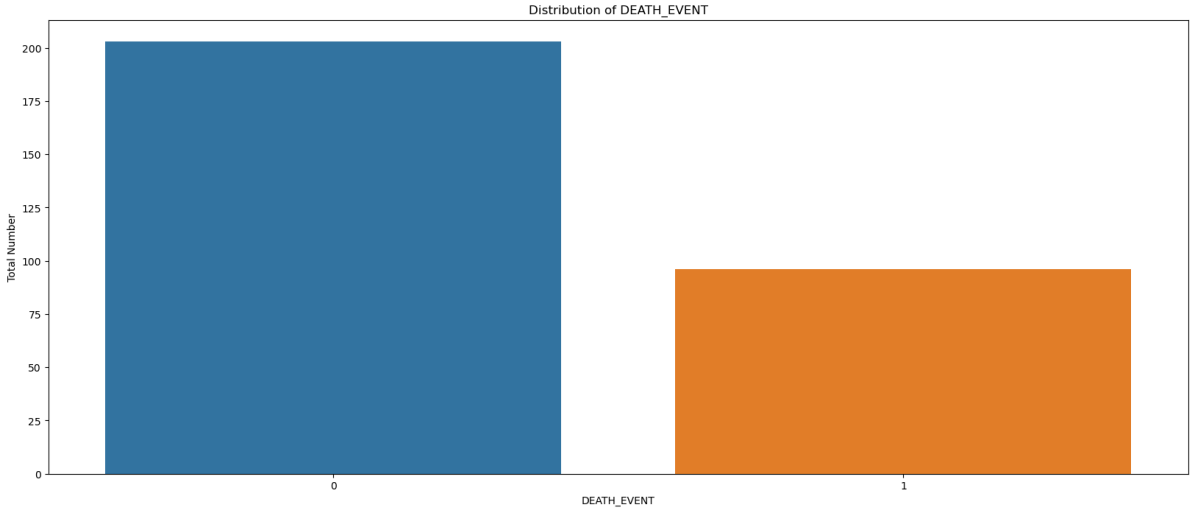


In [96]:
```
def plot_countplot(x,hue=None,df=df):
    plt.figure(figsize=(20,8))
    sns.countplot(x=x,data=df,hue=hue)
    plt.xlabel(f'{x}')
    plt.ylabel('Total Number')
    plt.title(f"Distribution of {x}")
    plt.show()
```

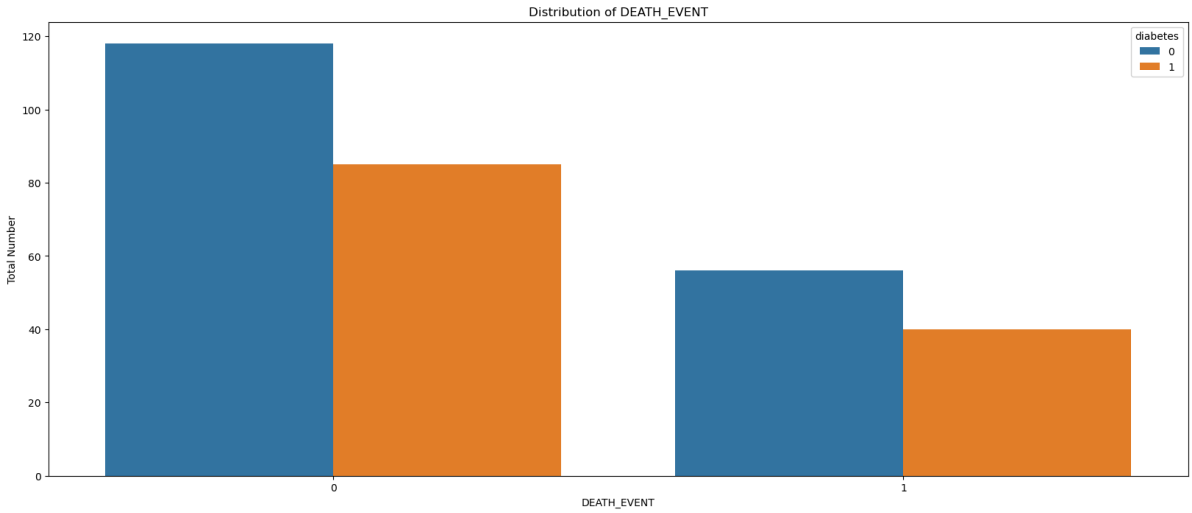
In [97]:
```
df.columns
```

Out[97]:
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

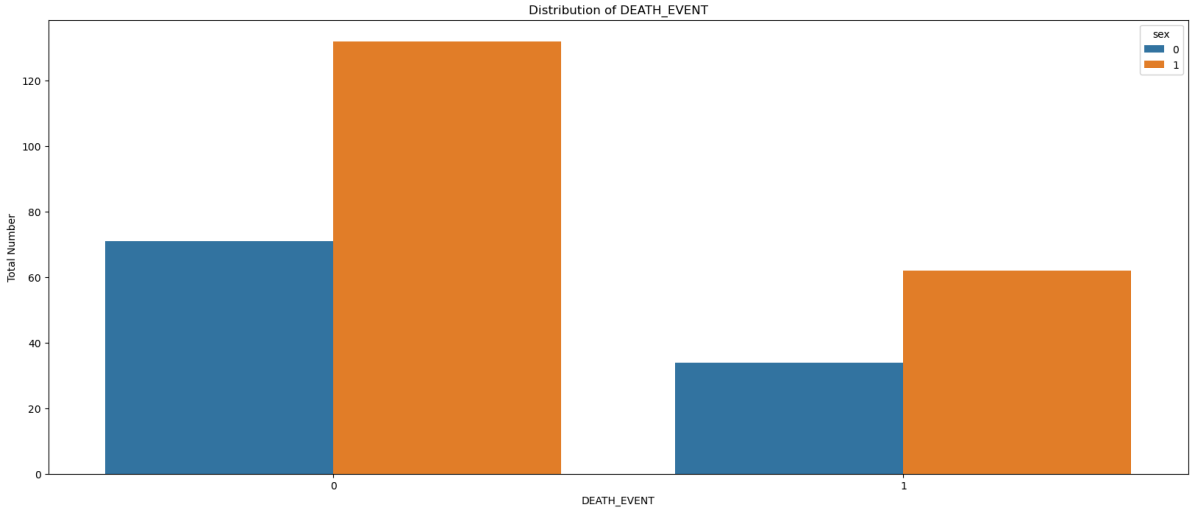
In [98]:
```
plot_countplot(x='DEATH_EVENT') # 0 represents alive 1 fatality # A majority of the
```



In [99]:
```
plot_countplot(x='DEATH_EVENT',hue='diabetes') # Diabetes may be a risk factor but
```

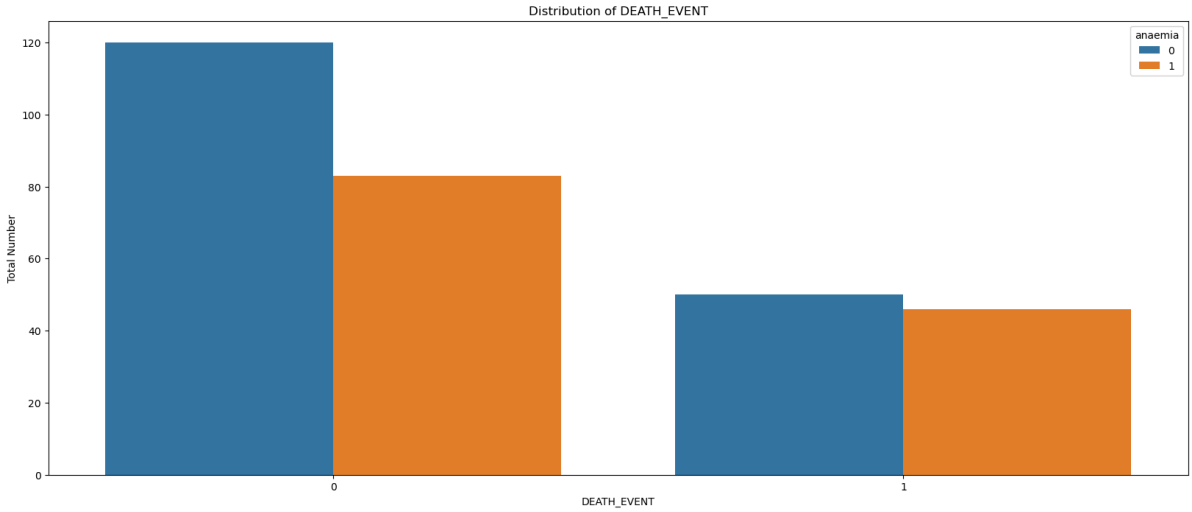


In [100…
```
plot_countplot(x='DEATH_EVENT',hue='sex') # The male gender is at high risk of havi
```



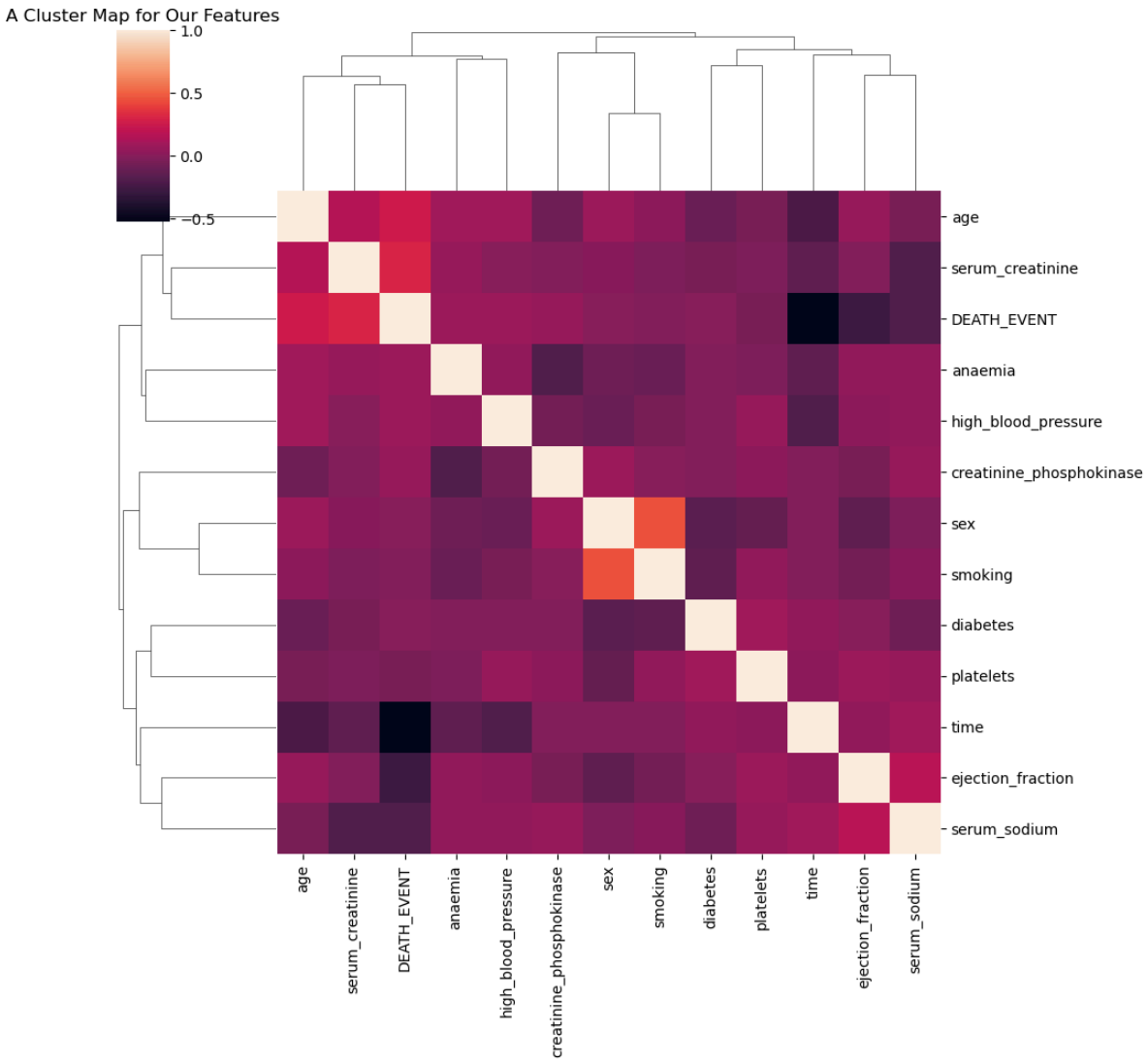
In [101…

```
plot_countplot(x='DEATH_EVENT',hue='anaemia') # Anaemia leads to heart failure in c
```



In [103…

```
sns.clustermap(df.corr())
plt.title('A Cluster Map for Our Features')
plt.show()
```



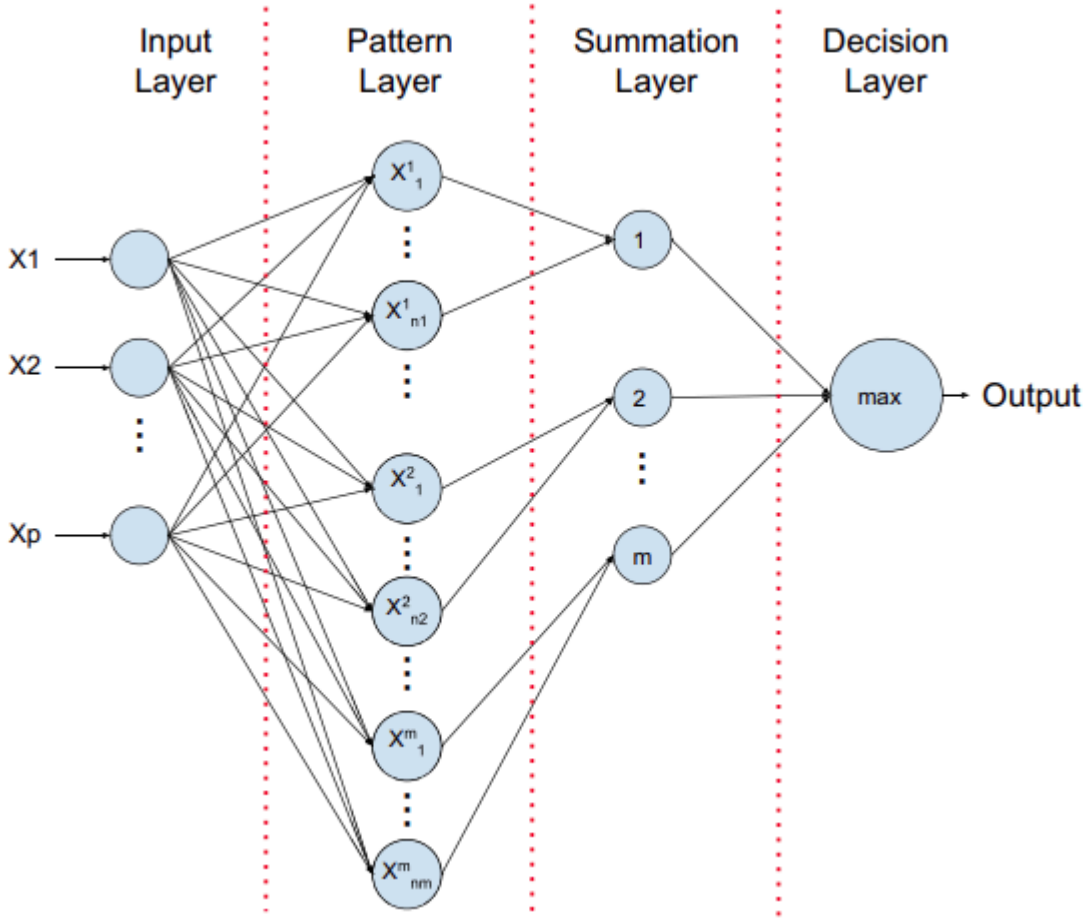
DATA PREPROCESSING

In [75]:

```
X= df.drop(['DEATH_EVENT'],axis=1)
y = df['DEATH_EVENT'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
X_knn_train, X_knn_test, y_knn_train, y_knn_test = train_test_split(X, y, test_size

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_knn_train = scaler.fit_transform(X_train)
X_knn_test = scaler.transform(X_test)
# Convert data to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

Probabilistic Nearest Neighbors(PNN)



In [52]:
```python
class ProbabilisticNearestNeighbors:
    def __init__(self, k=5):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        distances = self.compute_distances(X)
        return self.predict_labels(distances)

    def compute_distances(self, X):
        distances = torch.cdist(X, self.X_train)
        return distances

    def predict_labels(self, distances):
        knn_indices = distances.topk(self.k, largest=False).indices
        knn_labels = self.y_train[knn_indices]

        probas = []
        for knn in knn_labels:
            counts = torch.bincount(knn, minlength=len(torch.unique(self.y_train)))
            proba = counts / self.k
            probas.append(proba)

        return torch.stack(probas)
```

In [116…
```python
# Initialize and train the PNN model
k = 5
pnn = ProbabilisticNearestNeighbors(k=k)
pnn.fit(X_train, y_train)
```

```
# Predict probabilities on the test set
y_proba = pnn.predict(X_test)

# Convert probabilities to class predictions
y_pred = torch.argmax(y_proba, dim=1)

# Evaluate the model
accuracy = (y_pred == y_test).float().mean()
print(f'The PNN Accuracy stands at : {accuracy:.4f}')
```

```
The PNN Accuracy stands at : 0.6667
```

In [114…

```
X_train.shape,X_test.shape,y_train.shape
```

Out[114]:

```
(torch.Size([209, 12]), torch.Size([90, 12]), torch.Size([209]))
```

Conclusion

In this project, I implemented a Probabilistic Nearest Neighbors (PNN) classifier and evaluated its performance on a dataset with 12 features. Here are the key findings and insights from my analysis:

Model Performance

The PNN model achieved an accuracy of 66.67% on the test dataset. This accuracy indicates that the model correctly classified approximately two-thirds of the test samples. While this is a reasonable performance, further improvements could be explored.

Key Considerations

1. **Model Complexity:** The simplicity of the PNN classifier might limit its ability to capture complex patterns in the dataset. Exploring more sophisticated models or adjusting hyperparameters like the number of neighbors (k) could potentially improve performance.

2. **Feature Selection:** While the 2D decision boundary plot is insightful, the model's actual performance relies on all available features. Feature engineering or selection techniques could enhance the model's ability to generalize.

3. **Future Directions:** To enhance model performance, future work could focus on hyperparameter tuning, exploring ensemble methods, or improving data preprocessing techniques. Additionally, evaluating the model on a broader range of metrics and datasets could provide deeper insights into its strengths and limitations.

Recommendations

Based on the findings, I recommend:

- Further optimizing the PNN model by experimenting with different values of k and conducting cross-validation.
- Exploring more complex models or ensemble methods to potentially improve classification accuracy.

- Continuing to refine data preprocessing steps to ensure high-quality input for the model.

By addressing these areas, we can build upon the foundation laid by the PNN model and potentially achieve better performance in classification tasks.

In [ ]: