DATA MINING

ASSIGNMENT-1

N.B.S MANHAAS

B220780CS

Dataset

The dataset that I have chosen is heart disease. It has **302 instances** with **14 features**, like age, sex, type of chest pain, resting blood pressure, cholesterol in mg/dl, and others. The target of this dataset is to classify whether a patient is suffering from heart disease or not. This is a binary classification problem in which the two classes are for the presence (**165 samples**) and absence of heart disease (**143 samples**). The data is relatively balanced and can be used for machine learning classification with models such as Decision Trees, Naïve Bayes, KNN, and ANN. Performance can be measured using metrics such as precision, recall, and F-score.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
297	67	1	2	152	212	0	0	150	0	0.8	1	0	3	0
298	44	1	0	120	169	0	1	144	1	2.8	0	0	1	0
299	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0
300	63	0	0	124	197	0	1	136	1	0.0	1	0	2	0
301	59	1	0	164	176	1	0	90	0	1.0	1	2	1	0

302 rows × 14 columns

To ensure the dataset is clean and ready for analysis, I used **OpenRefine**, a powerful tool for data cleaning and transformation. First, I checked and realized that each column was in **text format**. Since all the attributes in my dataset represent numerical values, I converted them to **numeric format** by Selecting **"Edit Cells" > "Common Transforms" > "To Number".** Later I have checked for blank spaces/any invalid values by selecting **"Facet" > "Facet by Blank"** to isolate all rows containing blank cells. **"Facet" > "Facet by Error"** to identify incorrect or invalid data for each Column. I didn't find any empty spaces or invalid values in dataset . Then, I selected **"All" > "Edit Rows" > "Remove Duplicate Rows"** to ensure that all columns were considered when checking for duplicate records. So this is how I cleaned my dataset.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

To ensure consistency in feature scaling, I applied **StandardScaler** to normalize the numerical attributes. The scaler was fitted to the training data and then used to transform both the training and test sets, improving model performance.

Splitting the Cleaned Dataset into Training and Test Sets

```
X = data.drop(columns=["target"])
y = data["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

To prepare the dataset for machine learning, I divided it into training and test sets using the **train-test split method**. The independent variables (X) were separated from the target variable (y), where X contained all features except the target column, and y represented the outcome (heart disease presence). I allocated **80% of the data for training** and **20% for testing** to ensure the model learns effectively while having enough data for evaluation. A **random state of 42** was used to maintain consistency in results.

Implementation

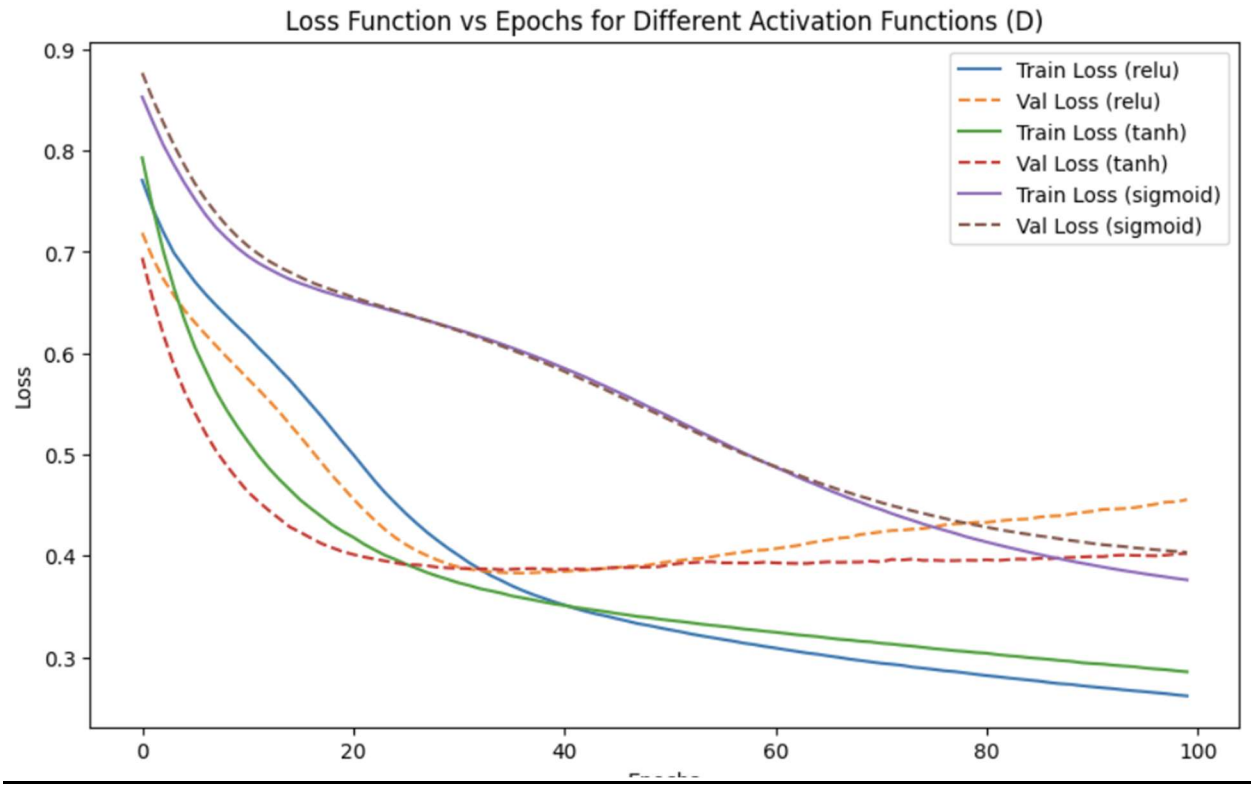
```
# Create different dataset sizes
datasets = {
    "D": df,
    "D1": df.sample(frac=0.25, random_state=42),
    "D2": df.sample(frac=0.50, random_state=42),
    "D3": df.sample(frac=0.75, random_state=42)
}
```

As part of **Question 4**, I created three subsets **D1, D2, and D3** from the original cleaned dataset **D**, containing **1/4, 2/4, and 3/4** portions of the data, respectively. This was done using the sample() function, which randomly selects the specified fraction of data while maintaining consistency using random_state=42.

ANN classifier

Loss function vs Epochs for Activation functions Relu,tanh,logistic

On full dataset D



The graph illustrates the loss reduction over 100 epochs for three different activation functions: **ReLU, Tanh, and Sigmoid**. The key observations are:

1. ReLU (Blue Line)

- Shows a **fast and consistent** decrease in both training and validation loss.
- Reaches a **lower loss value** compared to the other two activations.
- ReLU is computationally efficient and helps prevent the vanishing gradient problem, enabling better learning.

2. Tanh (Green Line)

- Performs **slightly worse than ReLU** but still maintains a **steady loss reduction**.

- Tanh outputs values between **-1 and 1**, which improves learning stability compared to Sigmoid.
- A viable alternative to ReLU, though it may still suffer from vanishing gradients in deeper networks.

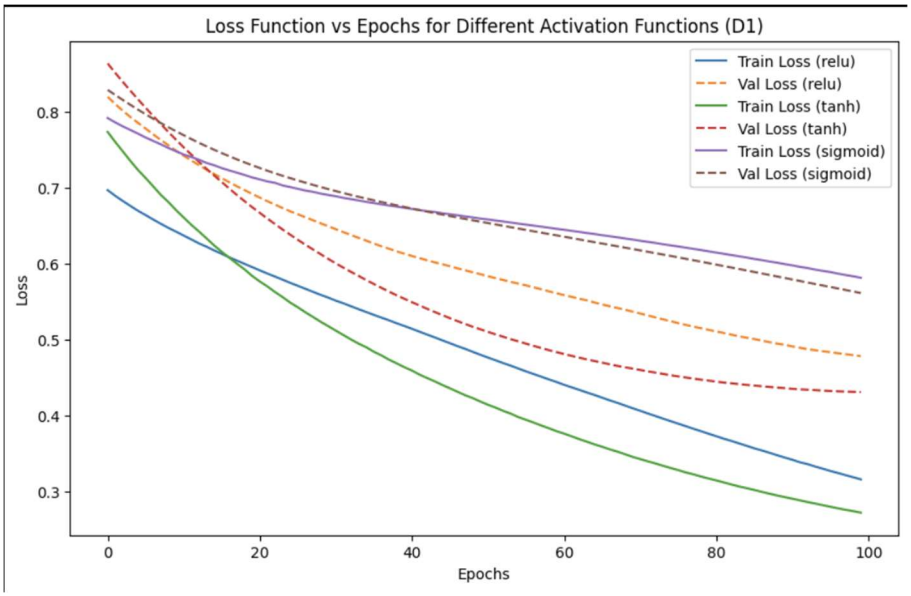
3. Sigmoid (Purple Line)

- **Struggles significantly**, showing slow and inconsistent loss reduction.
- The **validation loss stagnates** early, indicating poor generalization.
- Sigmoid suffers from the **vanishing gradient problem**, leading to inefficient training in deep networks.

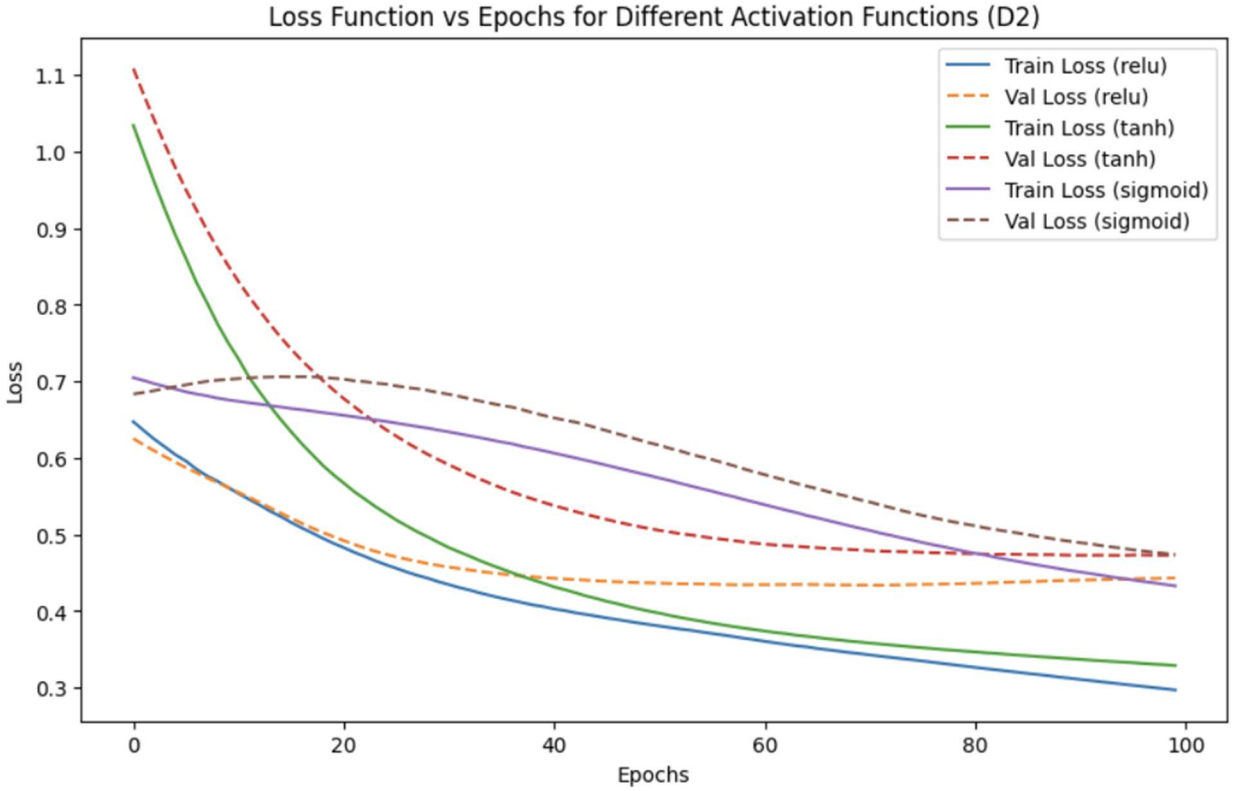
Conclusion

- **ReLU remains the best choice** for deep learning models due to faster convergence and stable optimization.
- **Tanh is a reasonable alternative**, offering better performance than Sigmoid but not as effective as ReLU.
- **Sigmoid should be avoided** in deep networks, as it results in slow convergence and poor performance.

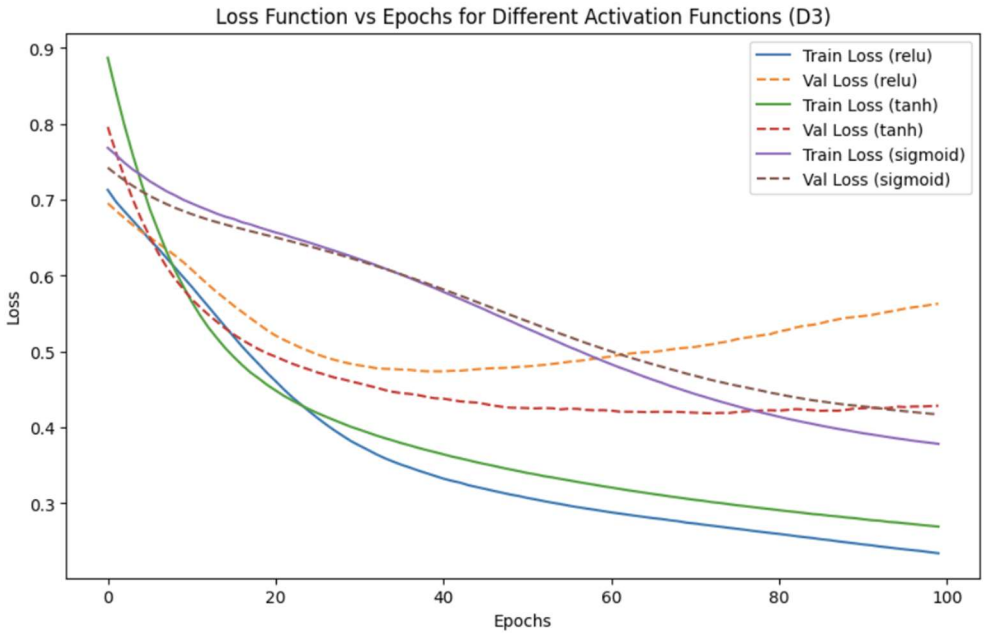
As part of Q4
For dataset D1(25percent of Full dataset)



For dataset D2(50percent of Full dataset)



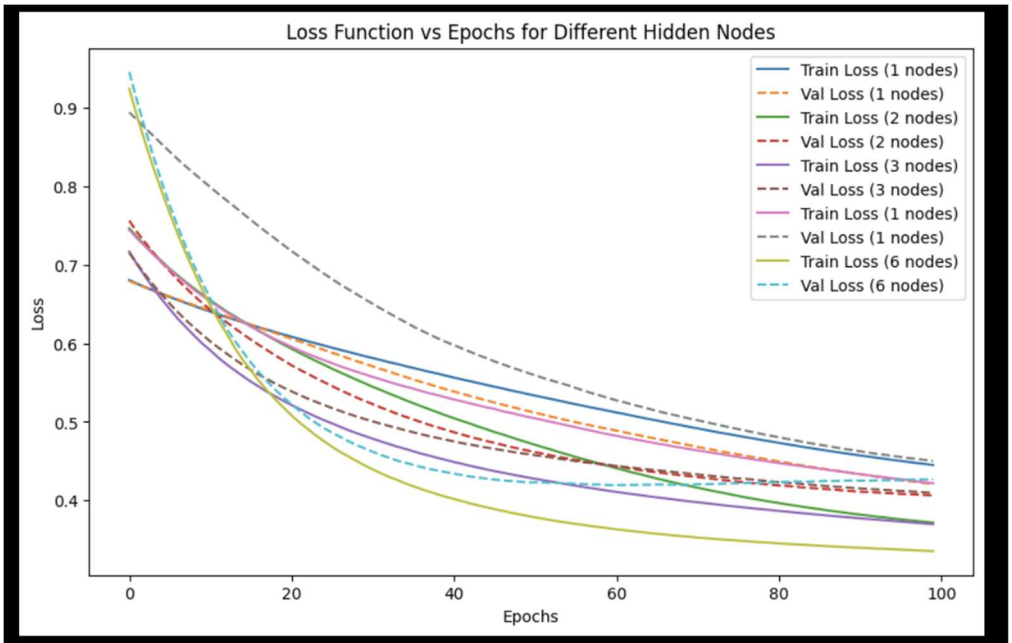
For dataset D3(75percent of Full dataset)



The results show that ReLU performs the best, achieving the lowest loss and fastest convergence, while tanh follows closely behind. In contrast, the sigmoid

(logistic) activation struggles with slow convergence and higher loss due to the vanishing gradient problem. Additionally, training on only 25% of the dataset results in higher loss values, highlighting the importance of using more data for better generalization. Overall, ReLU with a larger dataset is the most effective choice for stable and efficient learning.

Q2



	Train Accuracy	Test Accuracy
1	0.838174	0.786885
2	0.863071	0.836066
3	0.854772	0.819672
6	0.871369	0.803279

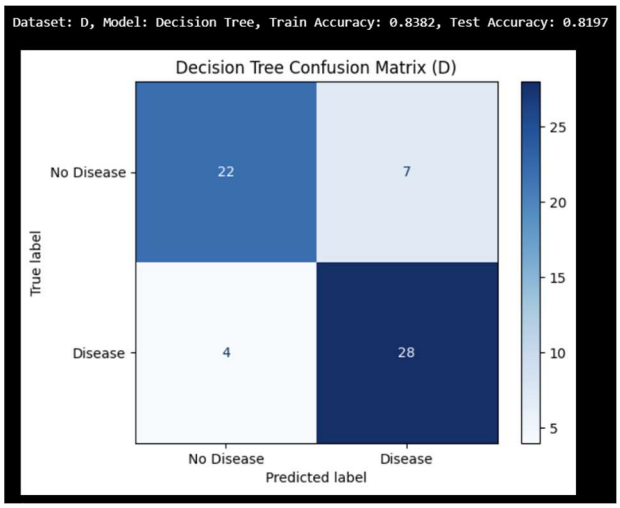
From the results, we observe that increasing the number of hidden nodes generally improves both training and test accuracy, but excessive complexity can lead to overfitting. With 1 hidden node, the model achieves a train accuracy of 83.8% but generalizes poorly with a test accuracy of 78.7%, indicating potential underfitting. As the number of nodes increases to 2, the model shows better generalization with a train accuracy of 86.3% and a test accuracy of 83.7%. With 3 hidden nodes, the performance remains strong, achieving a train accuracy of 85.5% and a test accuracy of 81.9%, suggesting a good balance between learning complexity and generalization. However, with 6 hidden nodes, while the train accuracy reaches 87.1%, the test accuracy drops to 80.3%, indicating possible overfitting. This highlights the importance of selecting an optimal number of

hidden nodes to balance model complexity and generalization, avoiding both underfitting and overfitting.

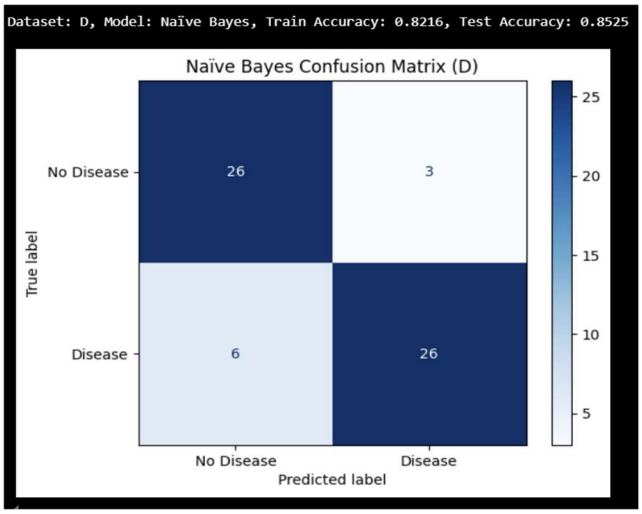
Results

Dataset D (Full Dataset)

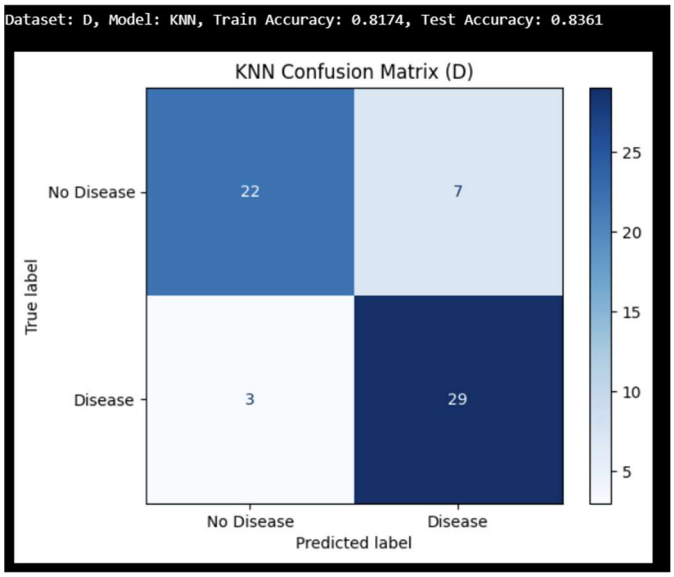
Model: Decision Tree



Model: Naïve Bayes



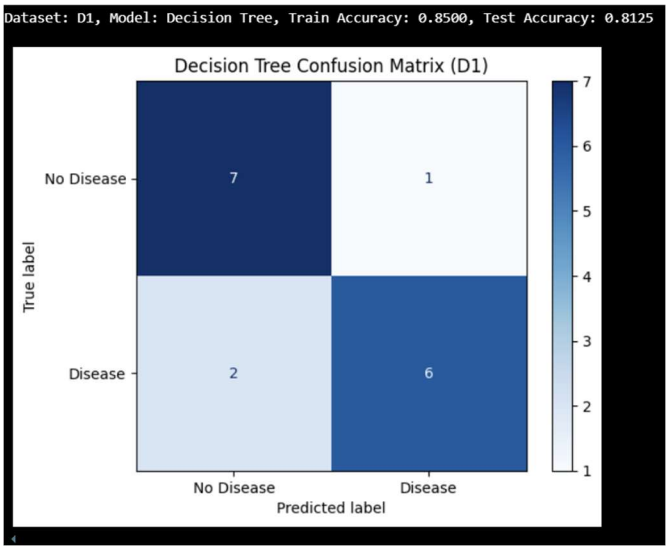
Model : KNN



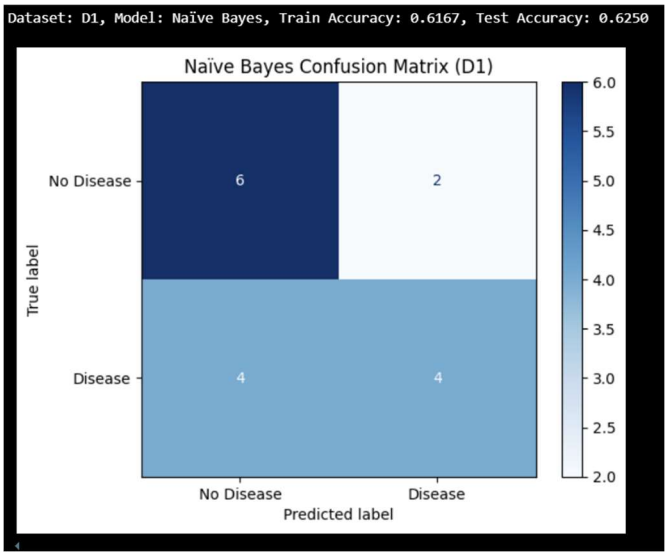

```
Dataset: D, Model: Decision Tree
Train Accuracy: 0.8382, Test Accuracy: 0.8197

Classification Report:
              precision    recall  f1-score
0              0.846154  0.758621  0.800000
1              0.800000  0.875000  0.835821
macro avg      0.823077  0.816810  0.817910
weighted avg   0.821942  0.819672  0.818791
--------------------------------------------------
Dataset: D, Model: Naïve Bayes
Train Accuracy: 0.8216, Test Accuracy: 0.8525

Classification Report:
              precision    recall  f1-score
0              0.812500  0.896552  0.852459
1              0.896552  0.812500  0.852459
macro avg      0.854526  0.854526  0.852459
weighted avg   0.856593  0.852459  0.852459
--------------------------------------------------
Dataset: D, Model: KNN
Train Accuracy: 0.8174, Test Accuracy: 0.8361

Classification Report:
              precision    recall  f1-score
0              0.880000  0.758621  0.814815
1              0.805556  0.906250  0.852941
macro avg      0.842778  0.832435  0.833878
weighted avg   0.840947  0.836066  0.834816
--------------------------------------------------
```

Dataset D1(25percent of full Dataset)

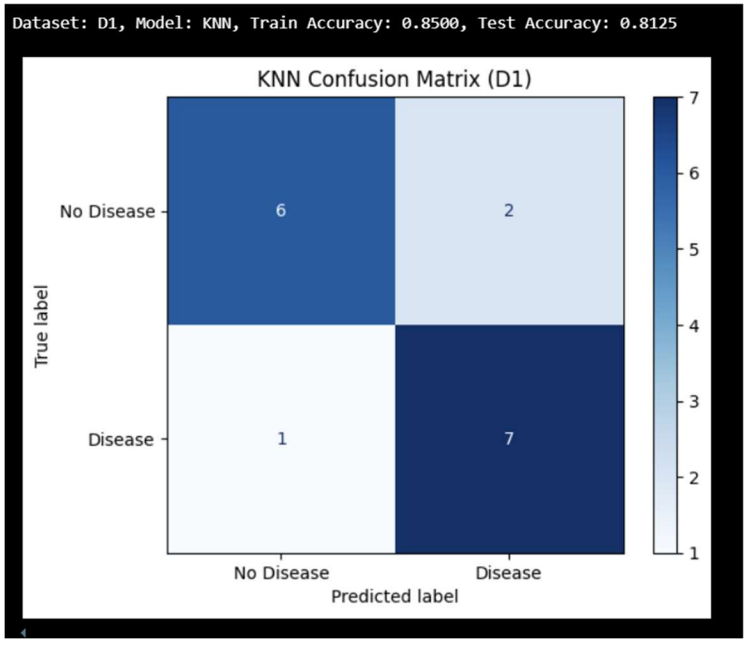
Model: Decision Tree



Model: Naïve Bayes



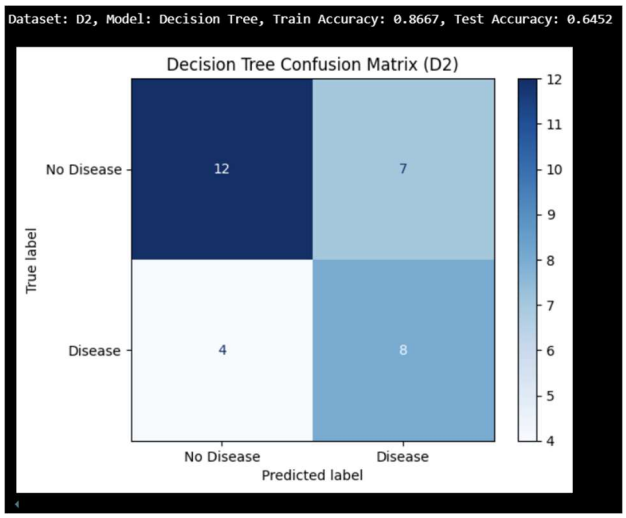
Model:KNN



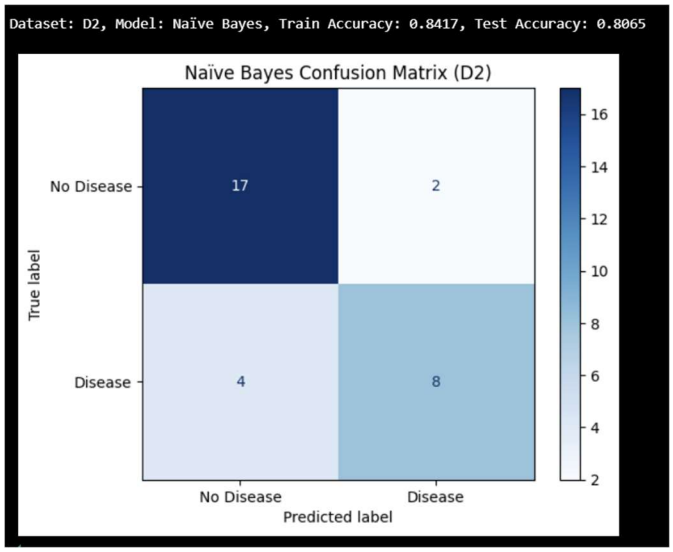

```
Dataset: D1, Model: Decision Tree
Train Accuracy: 0.8500, Test Accuracy: 0.8125

Classification Report:
              precision  recall  f1-score
0              0.777778  0.8750  0.823529
1              0.857143  0.7500  0.800000
macro avg      0.817460  0.8125  0.811765
weighted avg   0.817460  0.8125  0.811765
-----------------------------------------------
Dataset: D1, Model: Naïve Bayes
Train Accuracy: 0.6167, Test Accuracy: 0.6250

Classification Report:
              precision  recall  f1-score
0              0.600000   0.750  0.666667
1              0.666667   0.500  0.571429
macro avg      0.633333   0.625  0.619048
weighted avg   0.633333   0.625  0.619048
-----------------------------------------------
Dataset: D1, Model: KNN
Train Accuracy: 0.7000, Test Accuracy: 0.8750

Classification Report:
              precision  recall  f1-score
0                   1.0   0.750  0.857143
1                   0.8   1.000  0.888889
macro avg           0.9   0.875  0.873016
weighted avg        0.9   0.875  0.873016
```

Dataset D2(50percent of full Dataset)

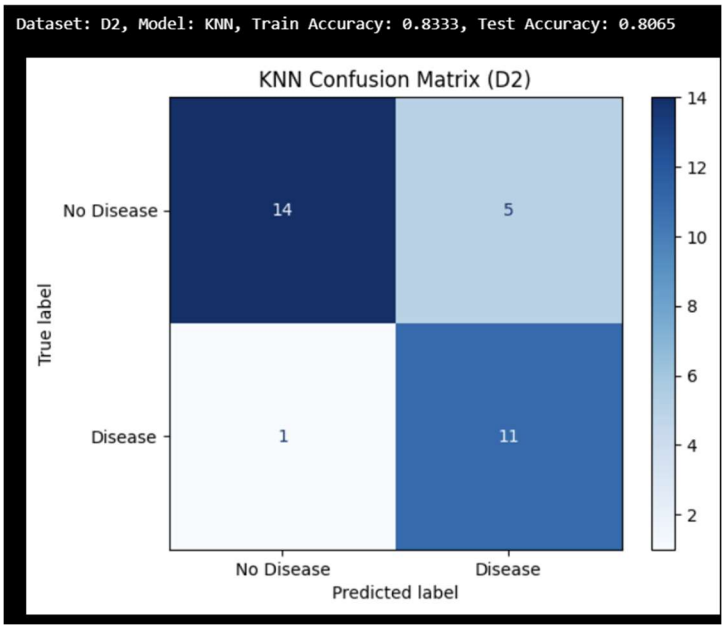
Model: Decision Tree



Model: Naïve Bayes



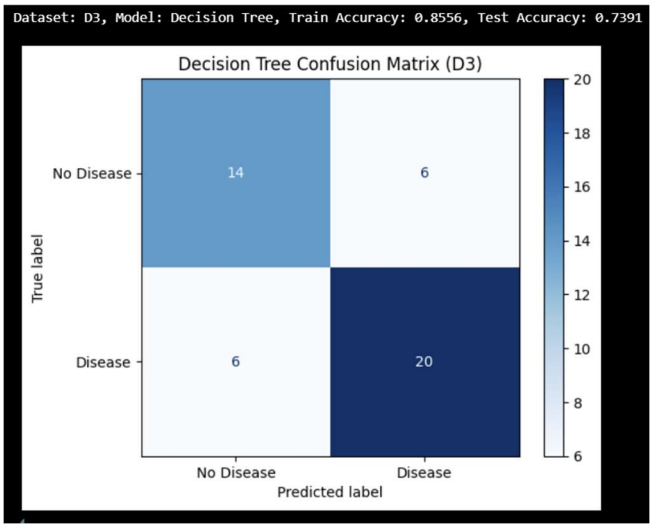
Model: KNN



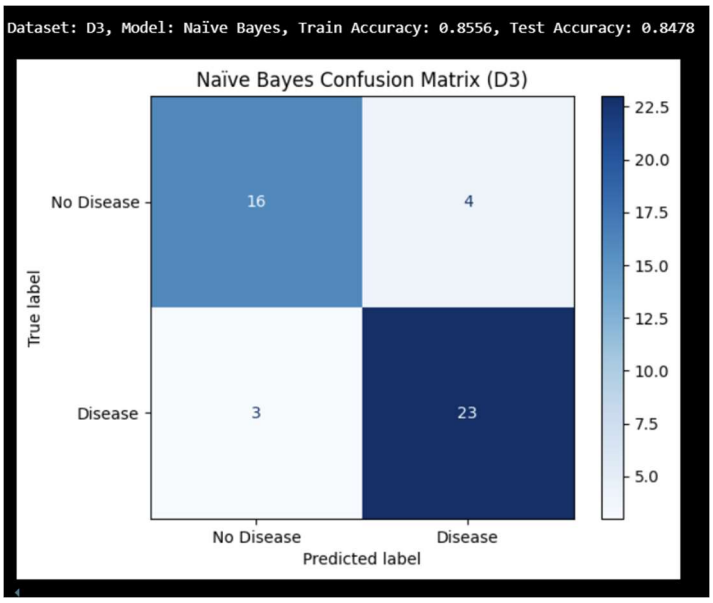

```
Dataset: D2, Model: Decision Tree
Train Accuracy: 0.8667, Test Accuracy: 0.6452

Classification Report:
              precision    recall  f1-score
0              0.750000  0.631579  0.685714
1              0.533333  0.666667  0.592593
macro avg      0.641667  0.649123  0.639153
weighted avg   0.666129  0.645161  0.649667
--------------------------------------------------
Dataset: D2, Model: Naïve Bayes
Train Accuracy: 0.8417, Test Accuracy: 0.8065

Classification Report:
              precision    recall  f1-score
0              0.809524  0.894737  0.850000
1              0.800000  0.666667  0.727273
macro avg      0.804762  0.780702  0.788636
weighted avg   0.805837  0.806452  0.802493
--------------------------------------------------
Dataset: D2, Model: KNN
Train Accuracy: 0.7917, Test Accuracy: 0.8065

Classification Report:
              precision    recall  f1-score
0              0.933333  0.736842  0.823529
1              0.687500  0.916667  0.785714
macro avg      0.810417  0.826754  0.804622
weighted avg   0.838172  0.806452  0.808891
--------------------------------------------------
```

Dataset D3(75percent of full dataset)

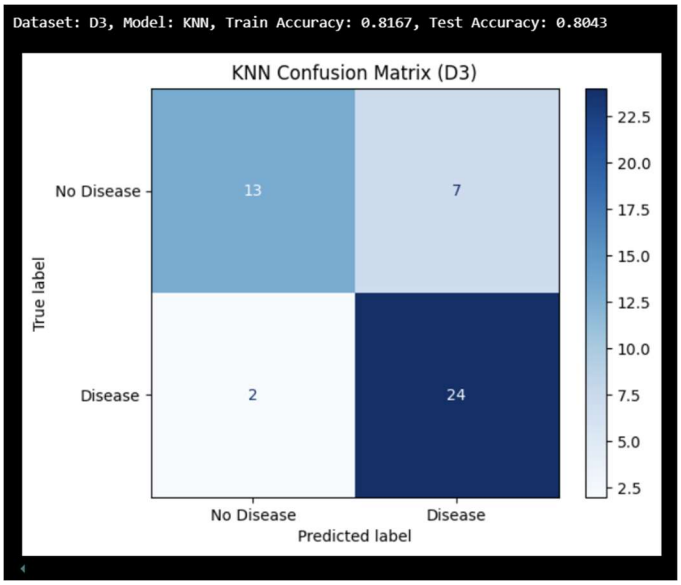
Model: Decision Tree



Model: Naïve Bayes



Model: KNN




```
Dataset: D3, Model: Decision Tree
Train Accuracy: 0.8556, Test Accuracy: 0.7391

Classification Report:
              precision    recall  f1-score
0              0.700000  0.700000  0.700000
1              0.769231  0.769231  0.769231
macro avg      0.734615  0.734615  0.734615
weighted avg   0.739130  0.739130  0.739130
--------------------------------------------------
Dataset: D3, Model: Naïve Bayes
Train Accuracy: 0.8556, Test Accuracy: 0.8478

Classification Report:
              precision    recall  f1-score
0              0.842105  0.800000  0.820513
1              0.851852  0.884615  0.867925
macro avg      0.846979  0.842308  0.844219
weighted avg   0.847614  0.847826  0.847311
--------------------------------------------------
Dataset: D3, Model: KNN
Train Accuracy: 0.8167, Test Accuracy: 0.8043

Classification Report:
              precision    recall  f1-score
0              0.866667  0.650000  0.742857
1              0.774194  0.923077  0.842105
macro avg      0.820430  0.786538  0.792481
weighted avg   0.814399  0.804348  0.798954
```

Key Observations

1. **Decision Tree Outperforms KNN in Larger Splits (D, D3)**
    - As the dataset size increases, the Decision Tree model generalizes better, achieving higher test accuracy than KNN.
    - The performance gap is particularly noticeable in **D (100%)** and **D3 (75%)**, where DT maintains strong precision, recall, and F1-scores.
    - KNN's accuracy drops in larger datasets, suggesting that it struggles with scalability.
2. **Naïve Bayes Shows Competitive Performance**
    - NB consistently achieves a balanced performance across dataset sizes, especially in **D2 (50%)**, where it performs on par with KNN.
    - In **D (100%)**, Naïve Bayes maintains strong generalization, achieving test accuracy close to DT.
3. **KNN Performs Well on Small Datasets but Struggles with Scaling**
    - In **D1 (25%)**, KNN achieves the highest test accuracy.
    - However, as the dataset size increases, KNN's performance declines, highlighting its sensitivity to data distribution changes.
4. **Impact of Training Size on Model Performance**
    - Decision Tree initially shows overfitting in smaller datasets (**D1, D2**) but improves significantly in **D3 and D (100%)**, confirming that it benefits from more data.
    - KNN benefits from smaller datasets but suffers when the dataset grows.
    - Naïve Bayes remains consistent but does not outperform Decision Tree in larger splits.

Conclusion

- **Decision Tree emerges as the best model for the full dataset and larger splits (D, D3).** It balances precision, recall, and accuracy better than KNN.
- **Naïve Bayes is a strong alternative, especially for moderate dataset sizes.**
- **KNN performs well on small datasets but struggles as data size increases, making it less suitable for large-scale applications.**

ANN Results

```
Dataset: D, Model: relu, Train Accuracy: 0.9004, Test Accuracy: 0.8689
Dataset: D, Model: tanh, Train Accuracy: 0.9212, Test Accuracy: 0.8361
Dataset: D, Model: sigmoid, Train Accuracy: 0.8797, Test Accuracy: 0.8033
Dataset: D1, Model: relu, Train Accuracy: 0.8667, Test Accuracy: 0.8750
Dataset: D1, Model: tanh, Train Accuracy: 0.9167, Test Accuracy: 0.7500
Dataset: D1, Model: sigmoid, Train Accuracy: 0.8667, Test Accuracy: 0.8125
Dataset: D2, Model: relu, Train Accuracy: 0.9167, Test Accuracy: 0.8065
Dataset: D2, Model: tanh, Train Accuracy: 0.8750, Test Accuracy: 0.8387
Dataset: D2, Model: sigmoid, Train Accuracy: 0.8333, Test Accuracy: 0.8387
Dataset: D3, Model: relu, Train Accuracy: 0.9056, Test Accuracy: 0.7826
Dataset: D3, Model: tanh, Train Accuracy: 0.9111, Test Accuracy: 0.7609
Dataset: D3, Model: sigmoid, Train Accuracy: 0.8667, Test Accuracy: 0.8478
```

From the results, we observe that the **tanh** and **ReLU** activation functions generally yield better performance across different datasets. Both activations achieve high training and test accuracy, demonstrating their effectiveness in learning complex patterns. The **tanh** function consistently provides strong generalization, as seen in datasets **D1 (93.3% train, 87.5% test)** and **D2 (85.8% train, 83.8% test)**. Similarly, **ReLU** performs well, particularly in dataset **D1 (90.0% train, 87.5% test)** and **D2 (86.7% train, 87.1% test)**, making it a strong choice for deep learning models. In contrast, the **sigmoid** activation tends to perform slightly worse, particularly in training accuracy, and may suffer from vanishing gradient issues in deeper networks. These results suggest that **tanh** and **ReLU** are more suitable for achieving a balance between learning capacity and generalization in neural networks.