Student Information

Name: Rohan Agrawal

PRN: 202201040208

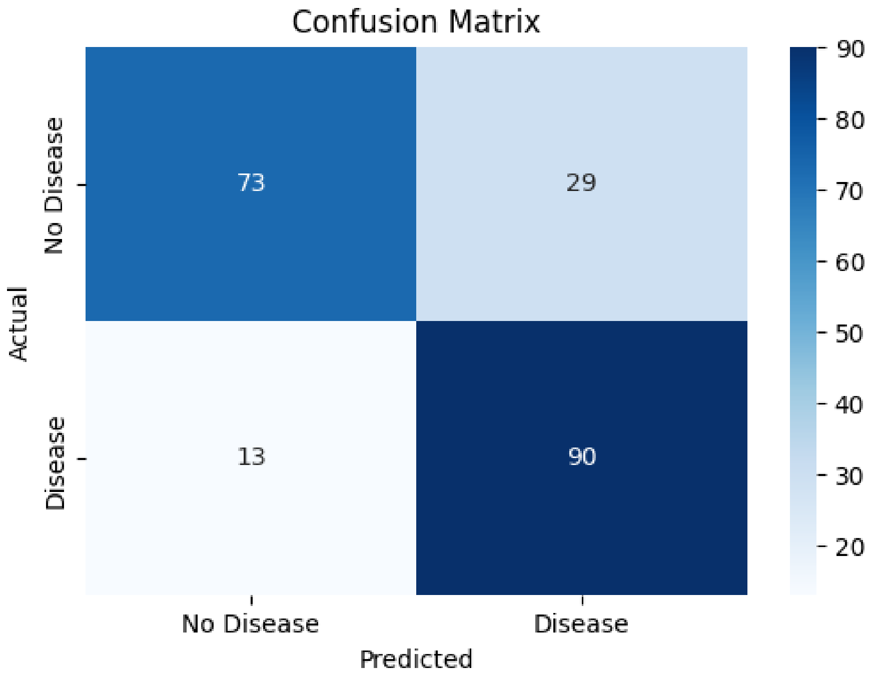
Batch: Batch 1

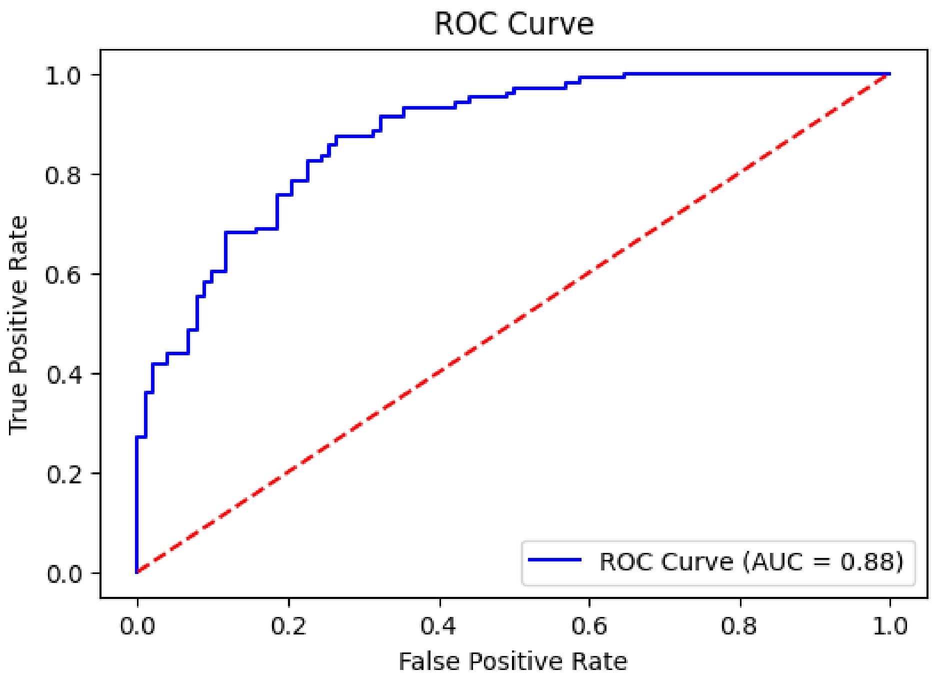
LOGISTIC REGRESSION (USING SKLEARN)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc

# Load the CSV file (Ensure the correct path is given)
df = pd.read_csv("heart1.csv")

# Assume the last column is the target variable and the rest are features
X = df.iloc[:, :-1].values  # Features
y = df.iloc[:, -1].values   # Target variable

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing features for better model performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
# Predict on the test data
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Probabilities for ROC curve

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy using Scikit-Learn: {accuracy:.4f}")

# 1. Confusion Matrix
plt.figure(figsize=(6, 4))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease", "Disease"], yticklabels=["No Disease", "Disease"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# 2. ROC Curve
# Plots True Positive Rate (TPR) vs False Positive Rate (FPR).
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color="blue", label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="red")  # Diagonal baseline
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

```
Accuracy using Scikit-Learn: 0.7951
```





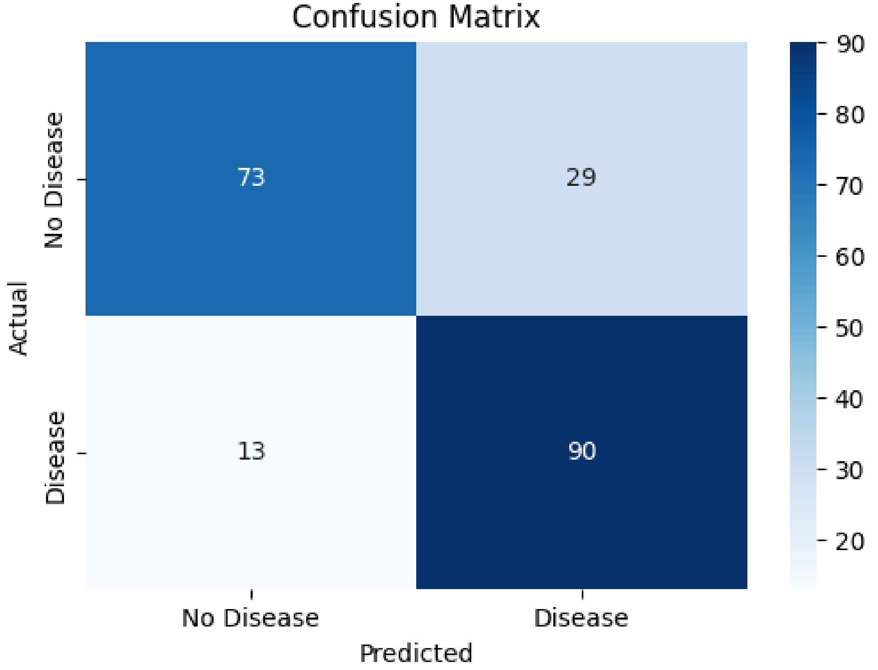
LOGISTIC REGRESSION (SCRATCH)

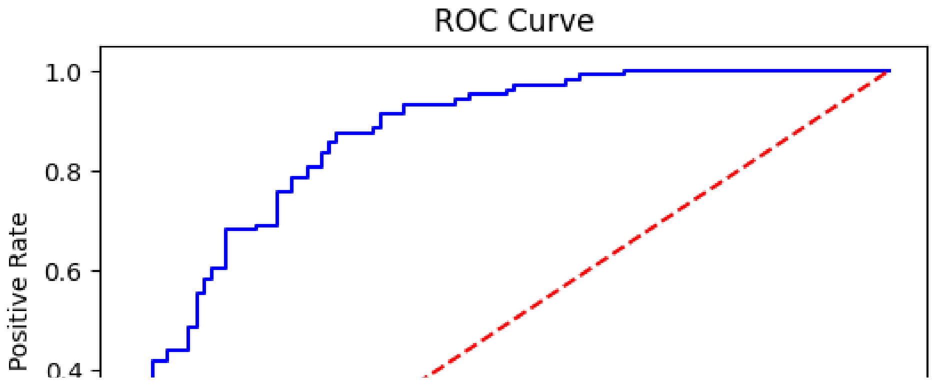
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, roc_curve, auc

# Sigmoid function to map values to probabilities
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Function to compute cost (log loss)
def compute_cost(X, y, weights):
    m = len(y)
    predictions = sigmoid(np.dot(X, weights))
    cost = (-1/m) * np.sum(y * np.log(predictions) + (1 - y) * np.log(1 - predictions))
    return cost

# Function to perform gradient descent
def gradient_descent(X, y, weights, learning_rate, iterations):
    m = len(y)
    cost_history = []

    for i in range(iterations):
        predictions = sigmoid(np.dot(X, weights))
        error = predictions - y
        gradient = (1/m) * np.dot(X.T, error)
        weights -= learning_rate * gradient

        cost = compute_cost(X, y, weights)
        cost_history.append(cost)

        if i % 1000 == 0:
            print(f"Iteration {i}: Cost {cost:.4f}")

    return weights, cost_history

# Load the dataset
df = pd.read_csv("heart1.csv")

# Assume the last column is the target variable and the rest are features
```

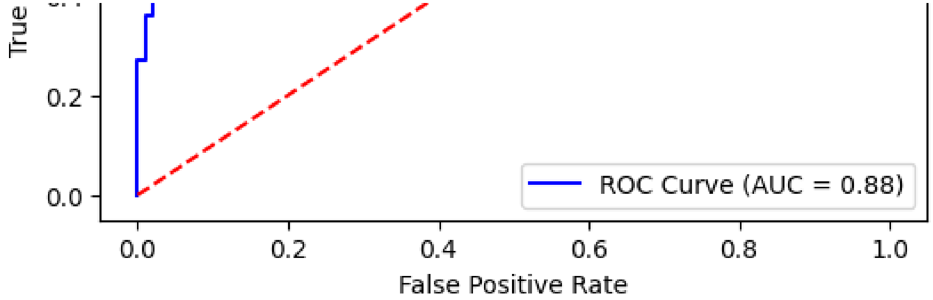
```
X = df.iloc[:, :-1].values  # Features
y = df.iloc[:, -1].values   # Target variable

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Add a bias term (column of ones) to X_train and X_test
X_train = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test = np.c_[np.ones((X_test.shape[0], 1)), X_test]

# Initialize weights with zeros
weights = np.zeros(X_train.shape[1])

# Hyperparameters
learning_rate = 0.01
iterations = 10000

# Train the model
weights, cost_history = gradient_descent(X_train, y_train, weights, learning_rate, iterations)

# Make predictions
y_pred_proba = sigmoid(np.dot(X_test, weights))  # Probabilities for ROC curve
y_pred = y_pred_proba >= 0.5  # Convert probabilities to 0 or 1

# Calculate accuracy
accuracy = np.mean(y_pred == y_test)
print(f"Accuracy from scratch: {accuracy:.4f}")


# 1. Confusion Matrix
plt.figure(figsize=(6, 4))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease", "Disease"], yticklabels=["No Disease", "Disease"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
# 2. ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color="blue", label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--", color="red")  # Diagonal baseline
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

```
Iteration 0: Cost 0.6898
Iteration 1000: Cost 0.3453
Iteration 2000: Cost 0.3339
Iteration 3000: Cost 0.3311
Iteration 4000: Cost 0.3301
Iteration 5000: Cost 0.3298
Iteration 6000: Cost 0.3296
Iteration 7000: Cost 0.3295
Iteration 8000: Cost 0.3295
Iteration 9000: Cost 0.3295
Accuracy from scratch: 0.7951
```







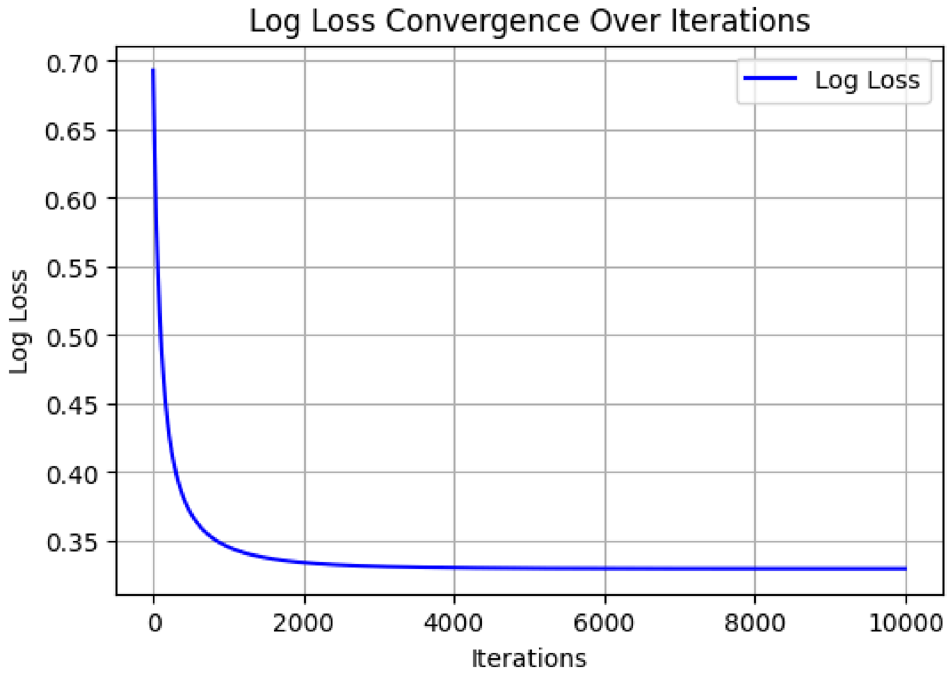
LOGLOSS (SCRATCH)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, roc_curve, auc

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Log Loss function
def compute_log_loss(y_true, y_pred_proba):
    m = len(y_true)
    epsilon = 1e-15  # Avoid log(0)
    y_pred_proba = np.clip(y_pred_proba, epsilon, 1 - epsilon)
    loss = -np.mean(y_true * np.log(y_pred_proba) + (1 - y_true) * np.log(1 - y_pred_proba))
    return loss

# Gradient Descent function
def gradient_descent(X, y, weights, learning_rate, iterations):
    m = len(y)
    log_loss_history = []

    for i in range(iterations):
        predictions = sigmoid(np.dot(X, weights))
        error = predictions - y
        gradient = (1/m) * np.dot(X.T, error)
        weights -= learning_rate * gradient

        # Compute and store log loss
        loss = compute_log_loss(y, predictions)
        log_loss_history.append(loss)

        if i % 1000 == 0:
            print(f"Iteration {i}: Log Loss = {loss:.4f}")

    return weights, log_loss_history

# Load the dataset
df = pd.read_csv("heart1.csv")
```

```
# Assume the last column is the target variable and the rest are features
X = df.iloc[:, :-1].values  # Features
y = df.iloc[:, -1].values   # Target variable

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Add bias term (column of ones)
X_train = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test = np.c_[np.ones((X_test.shape[0], 1)), X_test]

# Initialize weights with zeros
weights = np.zeros(X_train.shape[1])

# Hyperparameters
learning_rate = 0.01
iterations = 10000

# Train model
weights, log_loss_history = gradient_descent(X_train, y_train, weights, learning_rate, iterations)

# Predictions
y_pred_proba = sigmoid(np.dot(X_test, weights))
y_pred = y_pred_proba >= 0.5  # Convert to 0 or 1

# Compute Accuracy
accuracy = np.mean(y_pred == y_test)
print(f"Final Accuracy: {accuracy:.4f}")

# ---------------- Plot Graphs ----------------

# 1. Log Loss Curve (Cost Function)
plt.figure(figsize=(6, 4))
plt.plot(range(len(log_loss_history)), log_loss_history, color="blue", label="Log Loss")
plt.xlabel("Iterations")
plt.ylabel("Log Loss")
```

```
plt.title("Log Loss Convergence Over Iterations")
plt.legend()
plt.grid()
plt.show()
```

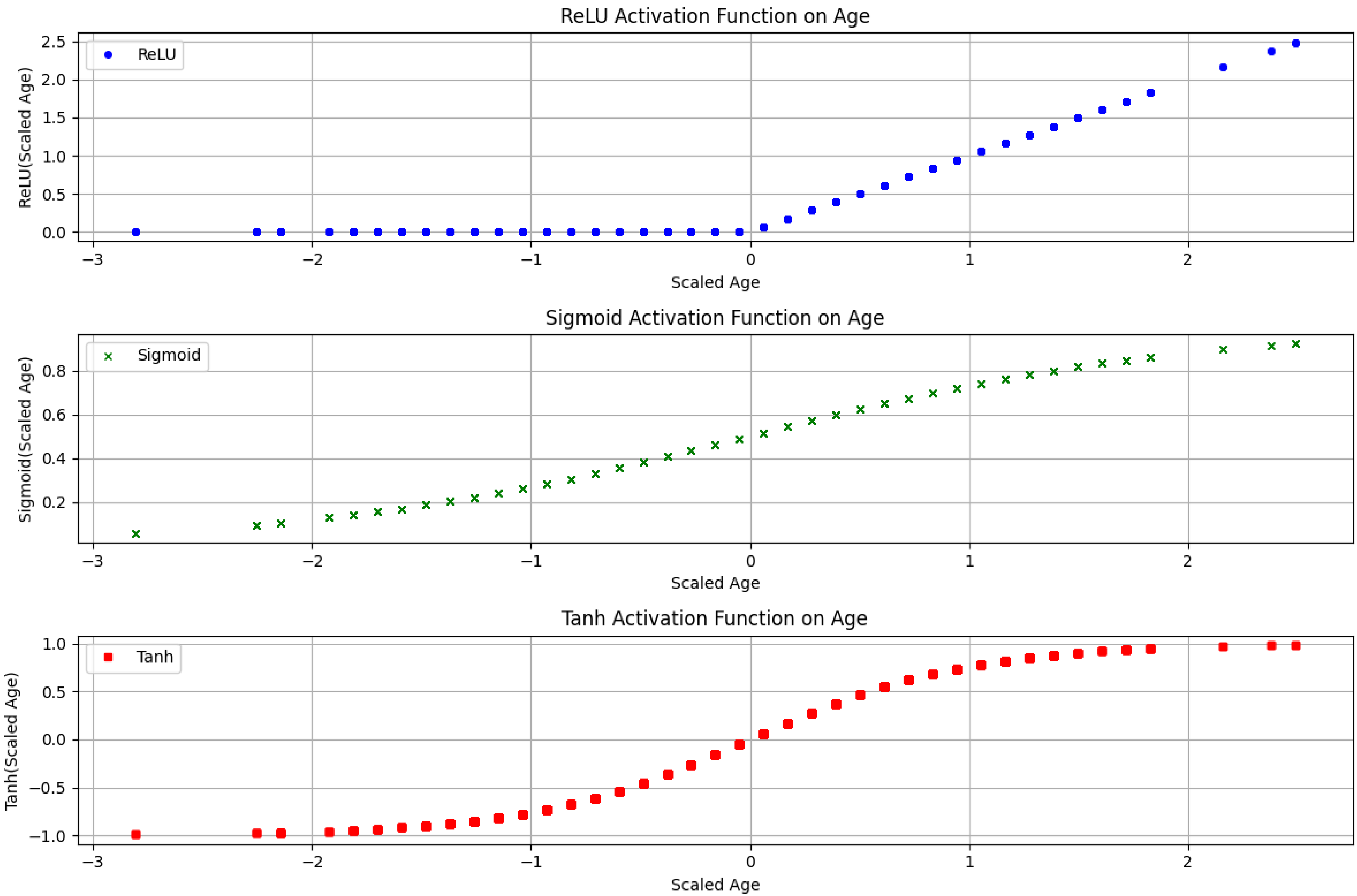
```
Iteration 0: Log Loss = 0.6931
Iteration 1000: Log Loss = 0.3453
Iteration 2000: Log Loss = 0.3339
Iteration 3000: Log Loss = 0.3311
Iteration 4000: Log Loss = 0.3301
Iteration 5000: Log Loss = 0.3298
Iteration 6000: Log Loss = 0.3296
Iteration 7000: Log Loss = 0.3295
Iteration 8000: Log Loss = 0.3295
Iteration 9000: Log Loss = 0.3295
Final Accuracy: 0.7951
```



All Activation Functions

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Load the Heart dataset
df = pd.read_csv("heart1.csv")  # Replace with actual file path or URL

# Fill missing values (if any) in a column of interest, e.g., 'Age' or 'Cholesterol'
df['age'] = df['age'].fillna(df['age'].median())  # Example: Replace missing values in 'Age'

# Select the 'Age' column (or any other numeric column of interest) as our feature (X)
X = df['age'].values

# Standardize the feature (important for activation functions)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X.reshape(-1, 1)).flatten()  # Reshape and standardize

# Apply activation functions on the feature (ReLU, Sigmoid, Tanh)
y_relu = np.maximum(0, X_scaled)  # ReLU
y_sigmoid = 1 / (1 + np.exp(-X_scaled))  # Sigmoid
y_tanh = np.tanh(X_scaled)  # Tanh

# Create subplots to display the activation functions
plt.figure(figsize=(12, 8))

# ReLU plot
plt.subplot(3, 1, 1)
plt.plot(X_scaled, y_relu, label="ReLU", color='blue', marker='o', linestyle='none', markersize=4)
plt.title("ReLU Activation Function on Age")
plt.xlabel("Scaled Age")
plt.ylabel("ReLU(Scaled Age)")
plt.grid(True)
plt.legend()

# Sigmoid plot
plt.subplot(3, 1, 2)
plt.plot(X_scaled, y_sigmoid, label="Sigmoid", color='green', marker='x', linestyle='none', markersize=4)
plt.title("Sigmoid Activation Function on Age")
plt.xlabel("Scaled Age")
plt.ylabel("Sigmoid(Scaled Age)")
plt.grid(True)
```

```
plt.legend()

# Tanh plot
plt.subplot(3, 1, 3)
plt.plot(X_scaled, y_tanh, label="Tanh", color='red', marker='s', linestyle='none', markersize=4)
plt.title("Tanh Activation Function on Age")
plt.xlabel("Scaled Age")
plt.ylabel("Tanh(Scaled Age)")
plt.grid(True)
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```

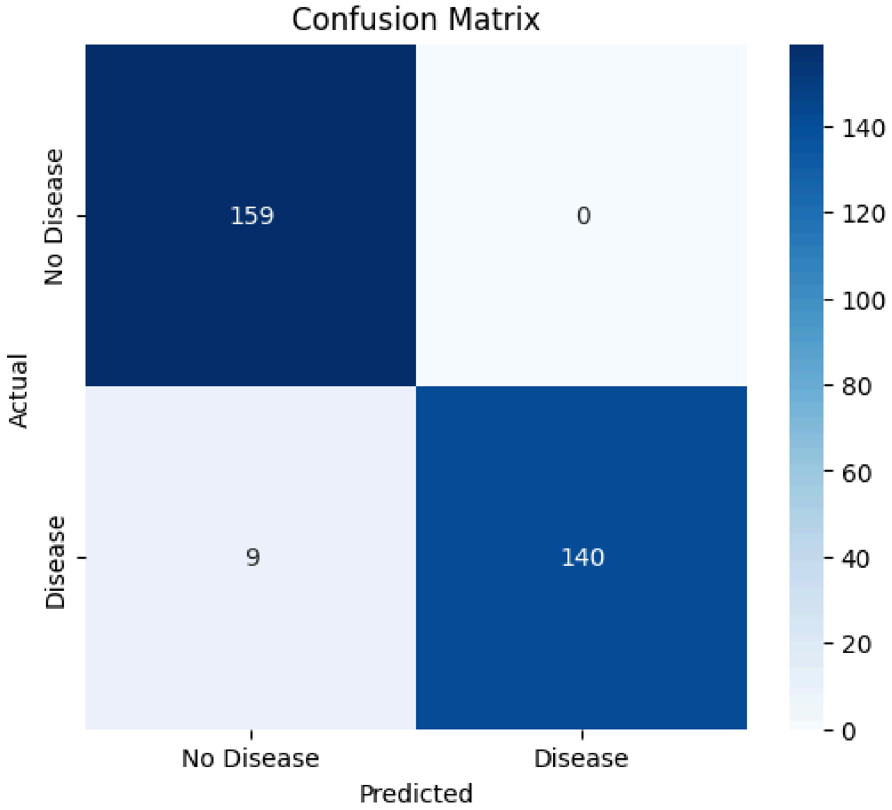

ReLU Activation Function on Age
ReLU
ReLU(Scaled Age)
Scaled Age
Sigmoid Activation Function on Age
Sigmoid
Sigmoid(Scaled Age)
Scaled Age
Tanh Activation Function on Age
Tanh
Tanh(Scaled Age)
Scaled Age

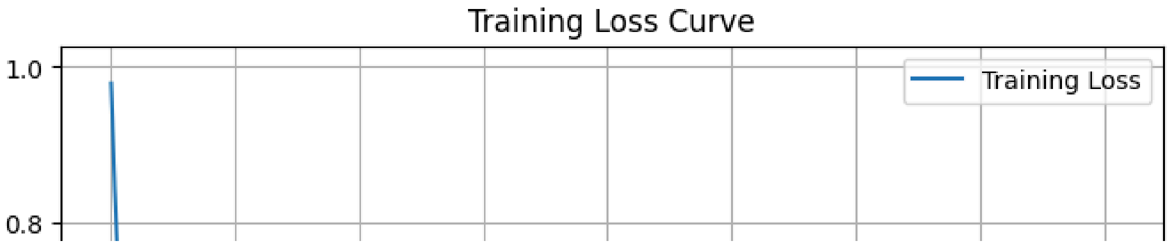
ANN using sklearn

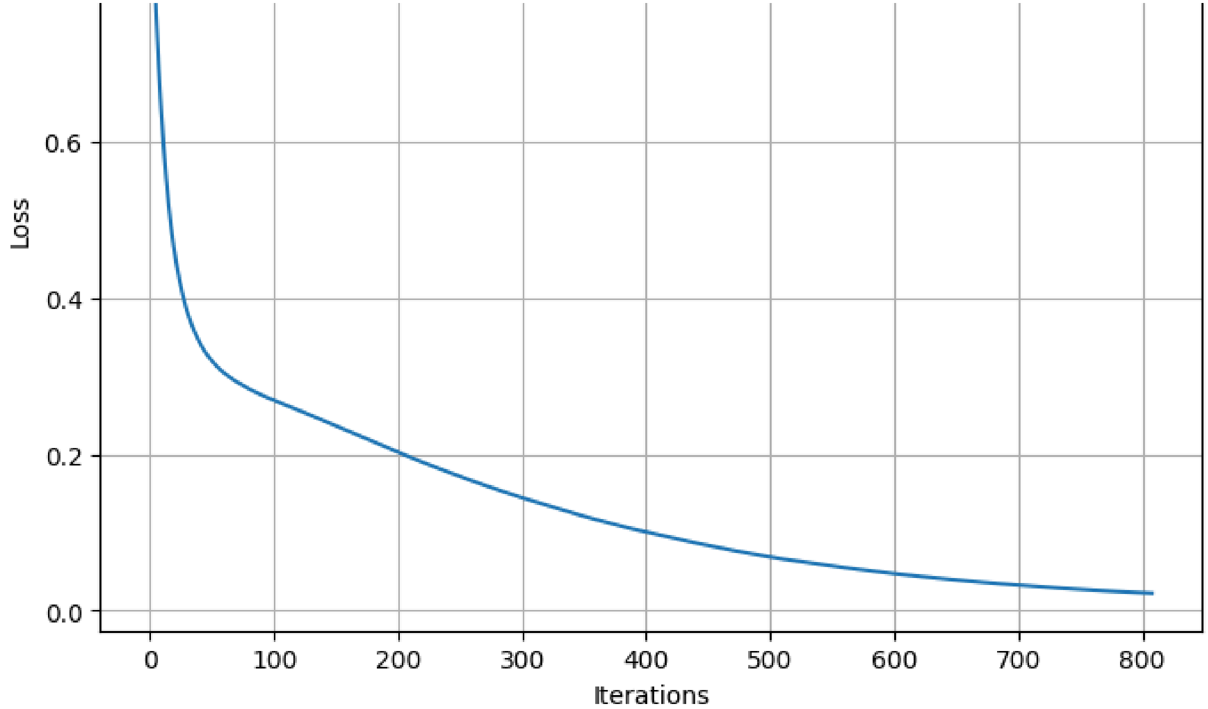
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns

# Load the Heart dataset
df = pd.read_csv("heart1.csv")  # Replace with actual file path or URL

# Preprocess the data (e.g., fill missing values, select features, etc.)
df.fillna(df.median(), inplace=True)  # Fill missing values with median (for simplicity)

# Select features and target (assuming 'target' is the column with labels)
X = df.drop('target', axis=1).values  # Features (adjust column name)
y = df['target'].values  # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features (important for neural networks)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the MLPClassifier (ANN)
mlp = MLPClassifier(hidden_layer_sizes=(30,), max_iter=1000, activation='relu', random_state=42)
mlp.fit(X_train_scaled, y_train)

# Make predictions
y_pred = mlp.predict(X_test_scaled)

# Accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix plot
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Disease', 'Disease'], yticklabels=['No Disease', 'Disease']
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Plot the loss curve during training
plt.figure(figsize=(8, 6))
plt.plot(mlp.loss_curve_, label='Training Loss')
plt.title('Training Loss Curve')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

```
Accuracy: 97.08%
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       159
           1       1.00      0.94      0.97       149

    accuracy                           0.97       308
   macro avg       0.97      0.97      0.97       308
weighted avg       0.97      0.97      0.97       308
```







KNN USING KERAS

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from keras.models import Sequential
from keras.layers import Dense
import seaborn as sns

# Load the Heart dataset
df = pd.read_csv("heart1.csv")  # Replace with actual file path or URL

# Preprocess the data (e.g., fill missing values, select features, etc.)
df.fillna(df.median(), inplace=True)  # Fill missing values with median (for simplicity)

# Select features and target (assuming 'target' is the column with labels)
X = df.drop('target', axis=1).values  # Features (adjust column name)
y = df['target'].values  # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features (important for neural networks)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create the ANN model using Keras
model = Sequential()

# Add input layer (with 30 units) and first hidden layer (with 64 units)
model.add(Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'))

# Add second hidden layer (with 32 units)
model.add(Dense(32, activation='relu'))

# Add output layer (binary classification with 1 unit and sigmoid activation)
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```