Task 2

Disease PREDICTION

Ibrahim Tarek Abdelazeem

Table of
CONTENTS

INITIALIZING

Libraries importing

```
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.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

Loading the Data

```
# Load the dataset
data = pd.read_csv('data.csv')
testdata = pd.read_csv('test_data.csv')
# Display the first few rows of the dataset
print(data.head())
```

DATASET

- Glucose,
- Cholesterol,
- Hemoglobin,
- Platelets,
- White Blood Cells,
- Red Blood Cells,
- Hematocrit,
- Mean Corpuscular Volume,
- Mean Corpuscular Hemoglobin,
- Mean Corpuscular Hemoglobin Concentration,
- Insulin,
- BMI,
- Systolic Blood Pressure,
- Diastolic Blood Pressure,
- Triglycerides,
- HbA1c,
- LDL Cholesterol,
- HDL Cholesterol,
- ALT,
- AST,
- Heart Rate,
- Creatinine,
- Troponin,
- C-reactive Protein,
- Disease

	Red Blood Cells	Hematocrit	Mean Corpuscular Volume	\
0	0.529895	0.290006	0.631045	
1	0.403033	0.164216	0.307553	
2	0.382021	0.625267	0.295122	
3	0.166214	0.073293	0.668719	
4	0.439851	0.894991	0.442159	

	Mean Corpuscular Hemoglobin	Mean Corpuscular Hemoglobin Concentration	\
0	0.001328	0.795829	
1	0.207938	0.505562	
2	0.868369	0.026808	
3	0.125447	0.501051	
4	0.257288	0.805987	

	...	HbA1c	LDL Cholesterol	HDL Cholesterol	ALT	AST	\
0	...	0.502665	0.215560	0.512941	0.064187	0.610827	
1	...	0.856810	0.652465	0.106961	0.942549	0.344261	
2	...	0.466795	0.387332	0.421763	0.007186	0.506918	
3	...	0.016256	0.040137	0.826721	0.265415	0.594148	
4	...	0.429431	0.146294	0.221574	0.015280	0.567115	

	Heart Rate	Creatinine	Troponin	C-reactive Protein	Disease
0	0.939485	0.095512	0.465957	0.769230	Healthy
1	0.666368	0.659060	0.816982	0.401166	Diabetes
2	0.431704	0.417295	0.799074	0.779208	Thalasse
3	0.225756	NaN	0.637061	0.354094	Anemia
4	NaN	0.153350	0.794008	0.094970	Thalasse

[5 rows x 25 columns]

DATA PREPROCESSING

Data Preprocessing

```
# Check for missing values
print(data.isnull().sum())
# Fill missing values if any (for simplicity, using mean of the columns here)
data = data.fillna(data.mean())
# Separate features and target variable
X_train = data.drop('Disease', axis=1)
y_train = data['Disease']

X_test = testdata.drop('Disease', axis=1)
y_test = testdata['Disease']

disease_classes = y_train.unique()  # Assuming all disease classes are present in the training set
# Define the mapping dictionary
mapping = {disease_class: 1 for disease_class in disease_classes}

# Apply the mapping to both the training and test target variables
y_train = y_train.apply(lambda x: 0 if x == 'Healthy' else 1)
y_test = y_test.apply(lambda x: 0 if x == 'Healthy' else 1)
```

```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
#  will have a mean of 0 and a standard deviation of 1 after transformation.
```

OUR MODEL

Building and Training the Model

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize the SVM base estimator
base_estimator = SVC(kernel='linear', probability=True)
# Initialize the AdaBoost classifier with SVM as the base estimator
adaboost_svm_classifier = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, random_state=42)
# Train the AdaBoost classifier
adaboost_svm_classifier.fit(X_train, y_train)
```

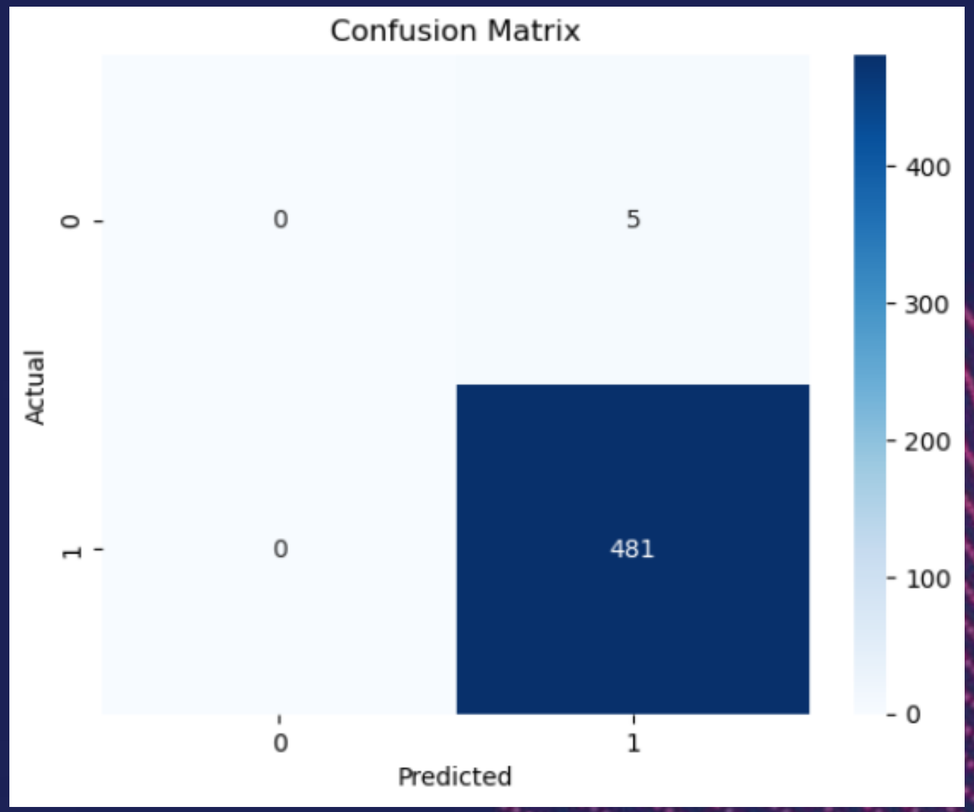
PREDICTION

Making Predictions and Evaluating the Model

```python
# Make predictions on the test set
y_pred = adaboost_svm_classifier.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Display the metrics
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'Precision: {precision* 100:.2f}%')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

ACCURACY

```
Accuracy: 98.97%
Precision: 97.95%
Recall: 98.97%
F1 Score: 98.46%
```




THANK YOU