SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY

Year 4 Semester 1

2024

Machine Learning - IT4060

Group Member Details

Student ID number	Name	Specialization
IT21036620 (Leader)	Kariyawasam K.P.W.D.V.	SE
IT21055294	Kahandagamage P.N	SE
IT20613440	Samarawijaya W.G.M.P	IT
IT21067242	Fernando M.R.R.	IT

Git Repository: https://github.com/IT21036620/ML_Assignment_Heart_Disease

DESCRIPTION OF THE PROBLEM ADDRESSED

In the healthcare industry, one of the major challenges for patients is to identify diseases early, before their health risks increase. Using predictive models or forecasting methods has become a captivating and necessary tool for healthcare professionals. However, forecasting heart disease presents unique complexities compared to other medical predictive attempts. Heart disease also known as cardiovascular disease (CVD) also affected by many other factors such as blood vessels, including coronary artery disease, myocarditis, vascular disease, and more. Furthermore, CVD claims the lives of 80% of individuals affected, with three-quarters of these fatalities occurring before the age of 70. CVD can also be affected by the lifestyle of a person Numerous factors can considered for the risk of developing cardiovascular disease, including gender, smoking, age, family history, dietary habits, cholesterol levels, physical activity levels, high blood pressure, weight management, and alcohol consumption.

An efficient and accurate heart disease forecasting model will help patients and healthcare providers take early action before a patient's health gets into a critical stage. Machine Learning techniques used to predict heart disease include the analysis of historical patient data to predict the possibility of having heart disease. This can be done by analyzing data to identify patterns of past information and finding the most suitable predictive model.

DATASET

I. Discription

The Dataset originated from the CDC's Behavioral Risk Factor Surveillance System, which annually conducts a telephone survey to collect data from US adults to collect data of their health status. This survey focuses on the major risk factors identified by the CDC such as high blood pressure, high cholesterol, and smoking, along with other significant indicators like diabetes, obesity, lack of physical activity, and excessive alcohol consumption. This refined dataset can be used for applying machine learning classification models such as logistic regression, Random Forest, Decision Tree, and SVM algorithms to predict heart disease.

Dataset Name	2022 annual CDC survey data of 400k+ adults related to their health status
Stroed Location	Kaggle
Dataset Link	https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease/data
Number of columns	18
Number of rows	319795
Columns	HeartDisease, BMI, Smoking, AlcoholDrinking, Stroke, PhysicalHealth, MentalHealth, DiffWalking, Sex, AgeCategory, Race, Diabetic, PhysicalActivity, GenHealth, SleepTime, Asthma, KidneyDisease, SkinCancer

II. Sample Images of the Dataset

```
dataset
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHe
0	0	16.60	1	0	0	3.0	30.0	0	0	7	5	2	1	
1	0	20.34	0	0	1	0.0	0.0	0	0	12	5	0	1	
2	0	26.58	1	0	0	20.0	30.0	0	1	9	5	2	1	
3	0	24.21	0	0	0	0.0	0.0	0	0	11	5	0	0	
4	0	23.71	0	0	0	28.0	0.0	1	0	4	5	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
319790	1	27.41	1	0	0	7.0	0.0	1	1	8	3	2	0	
319791	0	29.84	1	0	0	0.0	0.0	0	1	3	3	0	1	
319792	0	24.24	0	0	0	0.0	0.0	0	0	5	3	0	1	
319793	0	32.81	0	0	0	0.0	0.0	0	0	1	3	0	0	
319794	0	46.56	0	0	0	0.0	0.0	0	0	12	3	0	1	

301717 rows × 18 columns

SELECTED MACHINE LEARNING ALGORITHMS

For a dataset concerning heart disease, where the task is typically to predict whether an individual is at risk based on various medical and lifestyle factors, several types of machine learning algorithms can be effectively applied. The selected algorithms for this project are,

I. **Random Forests Classification -** Since a Random Forest Classifier (RFC) can handle various data types and complex, non-linear associations without requiring considerable preprocessing, it is a great fit for a dataset on heart disease. Because RFC uses an ensemble technique and averages several decision trees to improve generalization, it is prone to overfitting. It efficiently addresses data imbalances that are frequently found in medical datasets and offers useful information about the importance of features, which helps identify key risks. Because of these qualities, RFC is a great option for medical forecasts, where dependability, accuracy, and interpretability are essential for making effective clinical decisions.

II. **Logistic Regression** - For the binary classification problem of heart disease prediction, where precise and comprehensible results are essential, logistic regression is perfect. With its capacity to provide probability ratings and easily interpreted coefficients that illustrate the relative contributions of different risk variables to heart disease, this model is particularly useful for clinical decision-making. With regularization to avoid overfitting and guarantee strong predictions, it performs well on medium-sized datasets commonly used in medical research. Overall, logistic regression is a very useful tool for medical diagnostics, especially when evaluating the risk and early diagnosis of cardiac disease, due to its robustness, efficiency, and interpretability.

III. **Decision Tree** - The decision tree algorithm is suitable for heart disease prediction due to its ability to handle mixed data types, capture complex relationships, and provide transparent decision making. With its simplicity and interpretability, it facilitates understanding and trust in medical predictions. Additionally, feature importance analysis helps in identifying key risk factors critical to accurate predictions, making it a reliable choice for healthcare applications.

IV. **SVM -** SVM is appropriate for the prediction of heart disease, as SVMs will capture complex decision boundaries, hyperplanes for multidimensional data space. It is ideal for nonlinear relationships while accurate diagnosis is imperative for making accurate risk assessments of the heart. Generally, these statistical learning methods provide stable solutions in heart disease risk and heart disease diagnosis, and due to these properties, they will be exploited in the field of medicine.

METHODOLOGY

1. Random Forest Algorithm

I. Libraries

```
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

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

II. Data Visualization

Load The Data

```
#Load the Heart disease dataset from the Excel file
dataset = pd.read_csv('heart_2020_cleaned.csv')
```

View Dataset Details

```
#Display the number of rows & columns
print("Rows and Columns:",dataset.shape)
```

```
Rows and Columns: (319795, 18)
```

```
#Display data types of the columns
print("Column List:\n",dataset.columns)
```

```
Column List:
 Index(['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime',
       'Asthma', 'KidneyDisease', 'SkinCancer'],
      dtype='object')
```

```
#Display data types of the columns
print("Data Types:\n",dataset.dtypes)
```

```
Data Types:
 HeartDisease         object
BMI                 float64
Smoking              object
AlcoholDrinking      object
Stroke               object
PhysicalHealth      float64
MentalHealth        float64
DiffWalking          object
Sex                  object
AgeCategory          object
Race                 object
Diabetic             object
PhysicalActivity     object
GenHealth            object
SleepTime           float64
Asthma               object
KidneyDisease        object
SkinCancer           object
dtype: object
```

```
# Display first 5 records
dataset.head()
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealt
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Yes	Very goo
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Yes	Very goo
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Yes	Fa
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	No	Goo
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Yes	Very goo

Checking if there is any null vaues in the dataset before proceding futher

```
#display the number of null values in the dataset
print(dataset.isnull().sum())
```

```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

Analize the number of unique values contained in each column to understand the details of each columns to choose proper preprocessing methods

```
#Checking the number of unique values in each column
print(dataset.nunique())
```

```
HeartDisease           2
BMI                 3604
Smoking                2
AlcoholDrinking        2
Stroke                 2
PhysicalHealth        31
MentalHealth          31
DiffWalking            2
Sex                    2
AgeCategory           13
Race                   6
Diabetic               4
PhysicalActivity       2
GenHealth              5
SleepTime             24
Asthma                 2
KidneyDisease          2
SkinCancer             2
dtype: int64
```

III. Preprocessing

Duplicate data can lead the model to give biased or inaccurate predictions.This step removes duplicate rows from the dataset to ensure that the model training data includes only unique instances.

```
#Removing duplicates
dataset.drop_duplicates(inplace= True)
print("Dataset shape removing duplicates:", dataset.shape)
```

```
Dataset shape removing duplicates: (301717, 18)
```

```
#Creating an object list including object datatype
obj_list = dataset.select_dtypes(include='object').columns
```

```
#Transform the objects in the columns into numeric values
le = LabelEncoder()
for obj in obj_list:
        dataset[obj] = le.fit_transform(dataset[obj].astype(str))
```

above step is used to convert categorical data into a numerical format that can be processed by machine learning algorithms, which typically require numerical input

After applying the preprocessing dataset can be displayed as below,

```
#view the processed dataset
dataset
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHe
0	0	16.60	1	0	0	3.0	30.0	0	0	7	5	2	1	
1	0	20.34	0	0	1	0.0	0.0	0	0	12	5	0	1	
2	0	26.58	1	0	0	20.0	30.0	0	1	9	5	2	1	
3	0	24.21	0	0	0	0.0	0.0	0	0	11	5	0	0	
4	0	23.71	0	0	0	28.0	0.0	1	0	4	5	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
319790	1	27.41	1	0	0	7.0	0.0	1	1	8	3	2	0	
319791	0	29.84	1	0	0	0.0	0.0	0	1	3	3	0	1	
319792	0	24.24	0	0	0	0.0	0.0	0	0	5	3	0	1	
319793	0	32.81	0	0	0	0.0	0.0	0	0	1	3	0	0	
319794	0	46.56	0	0	0	0.0	0.0	0	0	12	3	0	1	

301717 rows × 18 columns

IV. Model Training

- Define X and Y

```
# Split the data into features and target
X = dataset.drop('HeartDisease', axis=1)
y = dataset['HeartDisease']
```

- Split Data

From the whole dataset 70% of the dataset will be used to train the model and 30% will be used to testing

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- Model Training

With the use of RandomForestClassifier the model will be trained.

```
# Training the Random Forest model
rfc_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rfc_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

V. Predicted and Actual Values Comparison

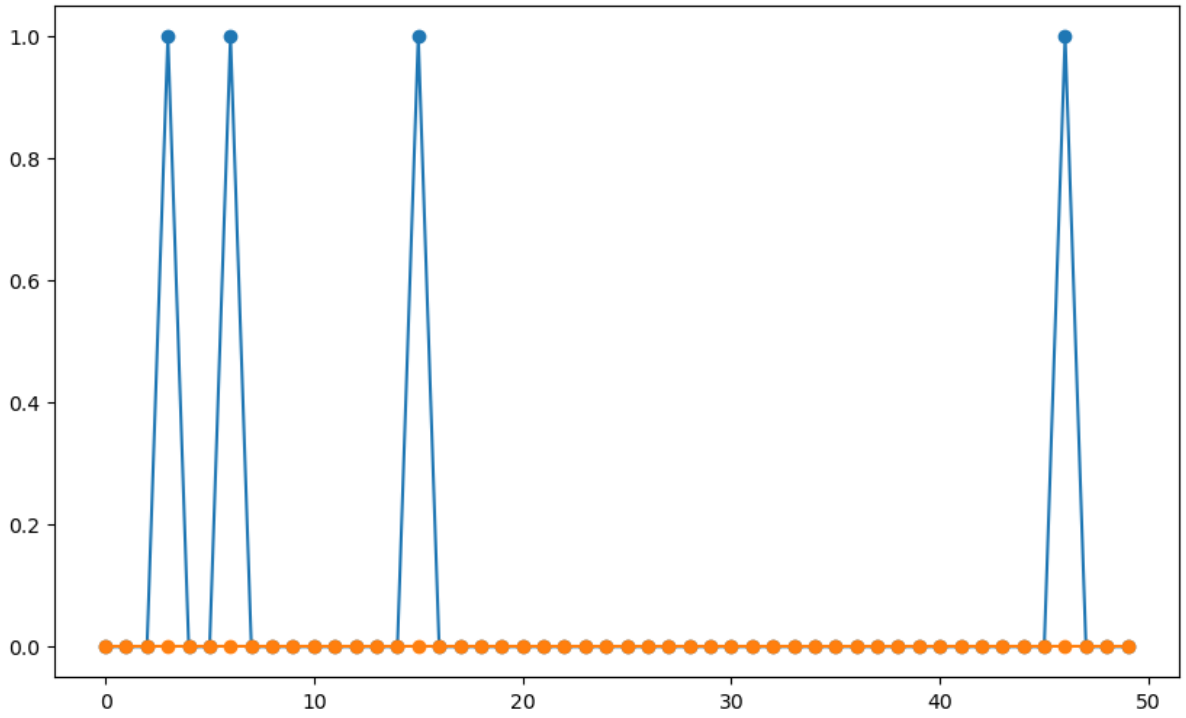
```
# Predicting the Test set results
y_pred = rfc_classifier.predict(X_test)
```

```
#compare the accuracy of predicted data with the actual data
print(f'Training Score: {rfc_classifier.score(X_train, y_train)}')
print(f'Testing Score: {rfc_classifier.score(X_test, y_test)}')
```

```
Training Score: 0.9974810725328005
Testing Score: 0.8992443324937027
```

```
#Create a Dataframe of actual values and predicted values
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Display the first 10 of the DataFrame
print("Comparison of Actual and Predicted values:")
print(df.head(10))

# Generating a plot for a visual comparison
plt.figure(figsize=(10, 6))
plt.plot(df[:50].reset_index(drop=True), marker='o')
```

```
Comparison of Actual and Predicted values:
        Actual  Predicted
284798       0          0
206219       0          0
298442       0          0
119751       1          0
103309       0          0
232511       0          0
45977        1          0
168666       0          0
189716       0          0
146390       0          0
[<matplotlib.lines.Line2D at 0x194cadc11d0>,
 <matplotlib.lines.Line2D at 0x194cadbe290>]
```

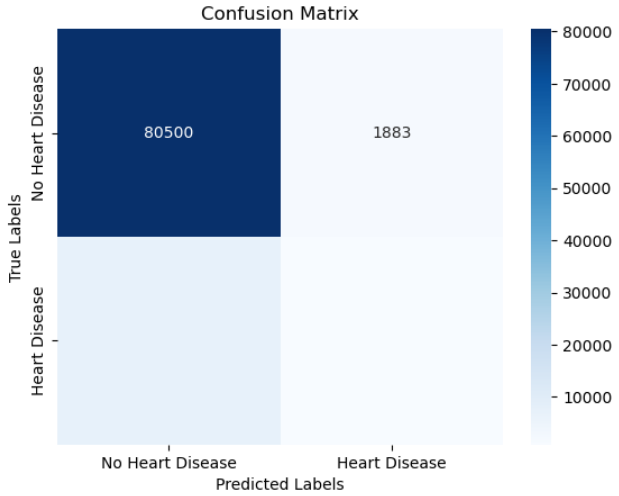


VI. Accuracy and Loss Function Values of the Model

```
print('Precision score:', precision_score(y_test, y_pred))
print('Recall score:', recall_score(y_test, y_pred))
print('Accuracy score:', accuracy_score(y_test, y_pred))
print('F1 score:', f1_score(y_test, y_pred))
print('Confusion Matrix:', confusion_matrix(y_test, y_pred))
```

```
Precision score: 0.3224181360201511
Recall score: 0.11016844952661994
Accuracy score: 0.8992443324937027
F1 score: 0.16422287390029322
Confusion Matrix: [[80500  1883]
 [ 7237   896]]
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['No Heart Disease', 'Heart Disease'], yticklabels=['No Heart Disease', 'Heart Disease'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



2. Logistic Regression

I. Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, log_loss
```

II. Data Visualization

```
# Load the dataset from excel sheet for processing
data_path = "heart_2020_cleaned.csv"
dataset = pd.read_csv(data_path)
```

```
# Check number of rows & columns in the dataset
print("Initial data shape:", dataset.shape)
```

```
Initial data shape: (319795, 18)
```

Prints the imported dataset's dimensions, giving a summary of the number of rows and columns. Uses the head() method to show the first five rows of the data frame.

```
# Check first five rows of dataset
print("First five rows of the dataset:\n", dataset.head())
```

```
First five rows of the dataset:
   HeartDisease    BMI Smoking AlcoholDrinking Stroke  PhysicalHealth  \
0           No  16.60     Yes              No     No             3.0
1           No  20.34      No              No    Yes             0.0
2           No  26.58     Yes              No     No            20.0
3           No  24.21      No              No     No             0.0
4           No  23.71      No              No     No            28.0

   MentalHealth DiffWalking     Sex  AgeCategory   Race Diabetic  \
0          30.0          No  Female        55-59  White      Yes
1           0.0          No  Female  80 or older  White       No
2          30.0          No    Male        65-69  White      Yes
3           0.0          No  Female        75-79  White       No
4           0.0         Yes  Female        40-44  White       No

  PhysicalActivity  GenHealth  SleepTime Asthma KidneyDisease SkinCancer
0              Yes  Very good        5.0    Yes            No        Yes
1              Yes  Very good        7.0     No            No         No
2              Yes       Fair        8.0    Yes            No         No
3               No       Good        6.0     No            No        Yes
4              Yes  Very good        8.0     No            No         No
```

```
#overview of the dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   HeartDisease      319795 non-null  object
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object
 3   AlcoholDrinking   319795 non-null  object
 4   Stroke            319795 non-null  object
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object
 8   Sex               319795 non-null  object
 9   AgeCategory       319795 non-null  object
 10  Race              319795 non-null  object
 11  Diabetic          319795 non-null  object
 12  PhysicalActivity  319795 non-null  object
 13  GenHealth         319795 non-null  object
 14  SleepTime         319795 non-null  float64
 15  Asthma            319795 non-null  object
 16  KidneyDisease     319795 non-null  object
 17  SkinCancer        319795 non-null  object
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

III. Preprocessing

To set up a dataset for machine learning modeling, Using dataset.drop_duplicates(inplace=True), it first eliminates duplicate rows to prevent duplication that can distort the model's findings. Then, depending on the quality of the data, it determines which columns may need to be excluded or undergo additional processing, such as imputation, by calculating and reporting the total number of missing values in each column using dataset.isnull().sum(). Finally, to generate binary variables for each category and prevent collinearity, the script analyzes categorical data using pd.get_dummies() with the parameter drop_first=True. Using this technique, category data is transformed into a numerical representation that machine learning algorithms can use. These procedures yield a dataset that is ready for modeling, as seen by the first few rows of the encoded data frame being shown.

```python
# Removes duplicate rows from the dataset
dataset.drop_duplicates(inplace=True)
```

```python
# Checking number of rows & columns after dropping duplicates
print("Dataset shape after dropping duplicate Values:", dataset.shape)
```

```
Dataset shape after dropping duplicate Values: (301717, 18)
```

```python
# Calculates and prints the number of missing values in each column.
missing_values = dataset.isnull().sum()
print("Missing Values in Each Column:\n", missing_values)
```

```
Missing Values in Each Column:
 HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

```python
# Preprocessing, Encoding categorical variables
categorical_cols = dataset.select_dtypes(include=['object']).columns
data_encoded = pd.get_dummies(dataset, columns=categorical_cols, drop_first=True)
```

```
# Display the first few rows of the new dataframe to verify encoding
print("Encoded Dataframe First rows:\n", data_encoded.head())
```

```
Encoded Dataframe First rows:
      BMI  PhysicalHealth  MentalHealth  SleepTime  HeartDisease_Yes  \
0  16.60             3.0          30.0        5.0             False
1  20.34             0.0           0.0        7.0             False
2  26.58            20.0          30.0        8.0             False
3  24.21             0.0           0.0        6.0             False
4  23.71            28.0           0.0        8.0             False

   Smoking_Yes  AlcoholDrinking_Yes  Stroke_Yes  DiffWalking_Yes  Sex_Male  \
0         True                False       False            False     False
1        False                False        True            False     False
2         True                False       False            False      True
3        False                False       False            False     False
4        False                False       False             True     False

   ...  Diabetic_Yes  Diabetic_Yes (during pregnancy)  PhysicalActivity_Yes  \
0  ...          True                            False                  True
1  ...         False                            False                  True
2  ...          True                            False                  True
3  ...         False                            False                 False
4  ...         False                            False                  True

   GenHealth_Fair  GenHealth_Good  GenHealth_Poor  GenHealth_Very good  \
0           False           False           False                 True
1           False           False           False                 True
2            True           False           False                False
3           False            True           False                False
4           False           False           False                 True

   Asthma_Yes  KidneyDisease_Yes  SkinCancer_Yes
0        True              False            True
1       False              False           False
2        True              False           False
3       False              False            True
4       False              False           False

[5 rows x 38 columns]
```

IV. Model Training

- Define X and Y

  The data is divided into features (X) and the target variable (y).

  ```
  X = data_encoded.drop('HeartDisease_Yes', axis=1)
  y = data_encoded['HeartDisease_Yes']
  ```

- Split Data

  Splitting the dataset into training and testing sets using train_test_split, with 80% of the data allocated for training and 20% for testing. This division is critical for evaluating the model's performance on unseen data, ensuring that it generalizes well beyond the training dataset.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Model Training

  The logistic regression model is initialized with a higher number of iterations (max_iter=1000) to allow the algorithm to converge properly.

```
# Initializes and trains the logistic regression model with a higher number of iterations
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train_scaled, y_train)
```

```
LogisticRegression(max_iter=1000)
```

V. Comparison of Actual Values and Predictions Values

```
# Predictions on the test set
y_pred = logistic_model.predict(X_test_scaled)
y_pred_proba = logistic_model.predict_proba(X_test_scaled)[:, 1]
```

```
# Training and Testing scores
train_score = logistic_model.score(X_train_scaled, y_train)
test_score = logistic_model.score(X_test_scaled, y_test)

print("Training Score:", train_score)
print("Testing Score:", test_score)
```

```
Training Score: 0.9113902549166643
Testing Score: 0.9107616333023996
```

VI. Accuracy and Loss Function Values of the Model

```python
# Calculates and prints various performance metrics
print("Precision Score:", precision_score(y_test, y_pred))
print("Recall Score:", recall_score(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Precision Score: 0.5303454715219421
Recall Score: 0.10422018348623853
Accuracy Score: 0.9107616333023996
F1 Score: 0.17420641005980678
```

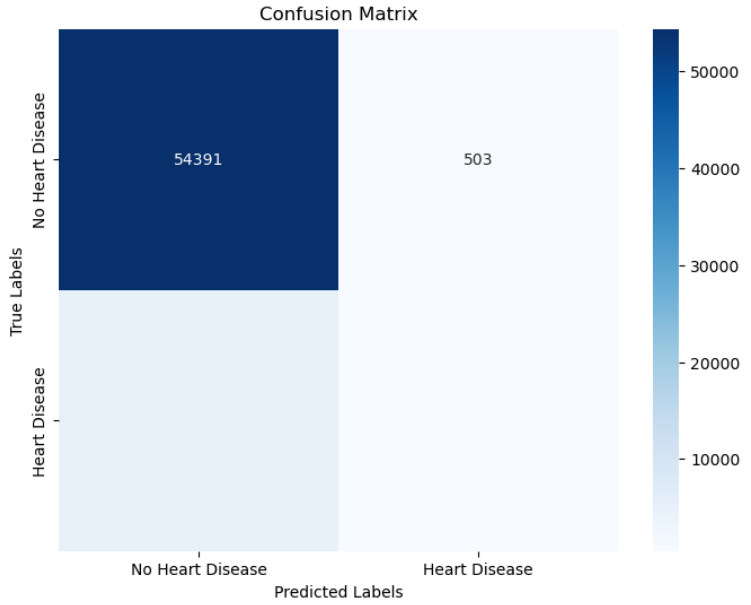
```python
# Log Loss metrics
log_loss_val = log_loss(y_test, y_pred_proba)
print("Log Loss:", log_loss_val)
```

```
Log Loss: 0.24021010552570374
```

```python
# Displays a detailed classification report and confusion matrix
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Classification Report:\n", report)
print("Confusion Matrix:\n", conf_matrix)
```

```
Classification Report:
               precision    recall  f1-score   support

       False       0.92      0.99      0.95     54894
        True       0.53      0.10      0.17      5450

    accuracy                           0.91     60344
   macro avg       0.72      0.55      0.56     60344
weighted avg       0.88      0.91      0.88     60344

Confusion Matrix:
 [[54391   503]
 [ 4882   568]]
```

```
# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No Heart Disease', 'Heart Disease'], yticklabels=['No Heart Disease', 'Heart D
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



3. Decision Tree

I. Libraries

The code starts by importing necessary libraries such as Pandas for data manipulation, NumPy for numerical operations, Matplotlib and Seaborn for visualization, and scikit-learn for machine learning algorithms.

```
#importing libraries
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.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
```

II. Data Visualization

The dataset is loaded using pd.read_csv(). The head() function displays the first few rows of the Data Set to understand its structure.

```
#Load the dataset
df = pd.read_csv('heart_2020_cleaned.csv')
```

```
#displays the first few rows of a DataFrame
df.head()
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenH
0	0	16.60	1	0	0	3.0	30.0	0	0	55-59	White	1	1	Very
1	0	20.34	0	0	1	0.0	0.0	0	0	80 or older	White	0	1	Very
2	0	26.58	1	0	0	20.0	30.0	0	1	65-69	White	1	1	
3	0	24.21	0	0	0	0.0	0.0	0	0	75-79	White	0	0	
4	0	23.71	0	0	0	28.0	0.0	1	0	40-44	White	0	1	Very

This is used to get summary statistics of the numerical columns in the data frame such as count, mean, std, min, max etc.

```
#get summary statistics
df.describe()
```

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.000000	319795.00000	319795.000000	319795.000000
mean	28.325399	3.37171	3.898366	7.097075
std	6.356100	7.95085	7.955235	1.436007
min	12.020000	0.00000	0.000000	1.000000
25%	24.030000	0.00000	0.000000	6.000000
50%	27.340000	0.00000	0.000000	7.000000
75%	31.420000	2.00000	3.000000	8.000000
max	94.850000	30.00000	30.000000	24.000000

This is used to find the unique values in each Colum.

```
#find unique values
df.nunique()
```

```
HeartDisease           2
BMI                 3604
Smoking                2
AlcoholDrinking        2
Stroke                 2
PhysicalHealth        31
MentalHealth          31
DiffWalking            2
Sex                    2
AgeCategory           13
Race                   6
Diabetic               2
PhysicalActivity       2
GenHealth              5
SleepTime             24
Asthma                 2
KidneyDisease          2
SkinCancer             2
dtype: int64
```

This is used to check for missing values in dataset.

```
#check for missing values
df.isnull().sum()
```

```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

III. Preprocessing

Preprocessing is a crucial step in preparing the data for machine learning models. In this code, certain categorical values are transformed into binary representations to ensure compatibility with the decision tree classifier and improve the model's performance.

Preprocessing involves identifying categorical variables and mapping their values to binary representations. This simplifies modeling for algorithms like decision trees. Binary encoding replaces 'Yes' with 1 and 'No' with 0.

```
#preprocessing
df = df[df.columns].replace({'Yes':1, 'No':0, 'Male':1, 'Female':0, 'No, borderline diabetes':0,'Yes (during pregnancy)':1, 'Y':1
df.head()
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenH
0	0	16.60	1	0	0	3.0	30.0	0	0	55-59	White	1	1	Very
1	0	20.34	0	0	1	0.0	0.0	0	0	80 or older	White	0	1	Very
2	0	26.58	1	0	0	20.0	30.0	0	1	65-69	White	1	1	
3	0	24.21	0	0	0	0.0	0.0	0	0	75-79	White	0	0	
4	0	23.71	0	0	0	28.0	0.0	1	0	40-44	White	0	1	Very

IV. Model Training

- Define X and Y

The dataset is split into features (X) and target variable (y). Features are selected by dropping columns related to the target variable, age category, race, and general health.

```
#Split dataset for training and testing
y=df['HeartDisease']
X=df.drop(['HeartDisease', 'AgeCategory', 'Race', 'GenHealth'], axis=1)
```

- Split Data

The train_test_split() function from sklearn is used to split the data into training and testing sets.

```
#split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

- Model Training

Here used decision tree classifier is instantiated and trained using the training data.

```
#Train the decision tree model
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

V. Comparison of Actual Values and Predictions Values

This part assesses the model's performance by calculating its accuracy on both training and testing data and provides a direct comparison between actual and predicted values to evaluate the model's predictive capability.

```
# predict the test set resut
predictions_tree = clf.predict(X_test)

# Accuracy of predicted data with the actual data
train = clf.score(X_train, y_train)
test = clf.score(X_test, y_test)

print(f"training score = {train}")
print(f"testing score = {test}")
```

```
training score = 0.9827612393681653
testing score = 0.8691772897361866
```

```
# Create DataFrame with actual and predicted values
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': predictions_tree})

# Print the DataFrame (optional)
print(comparison_df.head(10))
```

```
        Actual  Predicted
271884       0          0
270361       0          0
219060       0          0
24010        0          0
181930       0          0
24149        1          0
185683       0          0
316656       0          1
305719       0          0
56786        0          1
```

VI. Accuracy and Loss Function Values of the Model

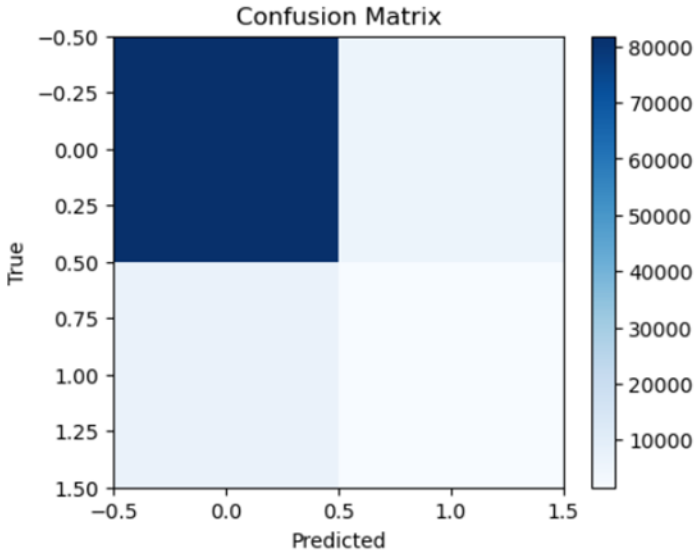
This code snippet calculates and prints performance metrics for decision tree model.

```
#Calculate and print performance metrics
predictions_tree = clf.predict(X_test)
accuracy_tree = clf.score(X_test, y_test)
confusion_matrix_tree = confusion_matrix(y_test, predictions_tree)
precision = precision_score(y_test, predictions_tree)
recall = recall_score(y_test, predictions_tree)
f1 = f1_score(y_test, predictions_tree)

print("Confusion matrix for Decision Tree")
print(confusion_matrix_tree)
print(f"Accuracy for Decision Tree = {accuracy_tree*100}%")
print(f"Precision for Decision Tree = {precision:.4f}")
print(f"Recall for Decision Tree = {recall:.4f}")
print(f"F1-score for Decision Tree = {f1:.4f}")
```

```
Confusion matrix for Decision Tree
[[81831  5818]
 [ 6733  1557]]
Accuracy for Decision Tree = 86.91772897361865%
Precision for Decision Tree = 0.2111
Recall for Decision Tree = 0.1878
F1-score for Decision Tree = 0.1988
```

The confusion matrix is visualized using Matplotlib to provide a graphical representation of the model's performance on the test set.

```
#visualize confusion matrix
plt.figure(figsize=(5, 4))
plt.imshow(confusion_matrix_tree, cmap=plt.cm.Blues)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.colorbar()
plt.show()
```



4. SVM

1. Necessary imports

In [21]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

2. Load data from csv

In [22]:
```
heartD_df = pd.read_csv('heart_2020_cleaned.csv')
heartD_df.head()
#heartD_df.size
heartD_df.shape
```

Out[22]:
```
(319795, 18)
```

3. Analyze dataset before preprocessing

3.1 Checking data types

In [24]:
```
heartD_df.dtypes
```

Out[24]:
```
HeartDisease         object
BMI                 float64
Smoking              object
AlcoholDrinking      object
Stroke               object
PhysicalHealth      float64
MentalHealth        float64
DiffWalking          object
Sex                  object
AgeCategory          object
Race                 object
Diabetic             object
PhysicalActivity     object
GenHealth            object
SleepTime           float64
Asthma               object
KidneyDisease        object
SkinCancer           object
dtype: object
```

3.2 Checking Categories Across Multiple Columns

In [25]:
```
# List of columns contains unique categories
```

3.2 Checking Categories Across Multiple Columns

In [25]:
```
# List of columns contains unique categories
columns_to_check = ['HeartDisease', 'Smoking', 'AlcoholDrinking',
                    'Stroke', 'DiffWalking', 'Sex', 'Race',
                    'PhysicalActivity', 'GenHealth', 'Asthma', 'KidneyDisease', 'SkinCancer', 'AgeCategory']

for column in columns_to_check:
    print(f"Categories in column '{column}':")
    print(heartD_df[column].value_counts())
    print()
```

```
Categories in column 'HeartDisease':
HeartDisease
No     292422
Yes     27373
Name: count, dtype: int64

Categories in column 'Smoking':
Smoking
No     187887
Yes    131908
Name: count, dtype: int64

Categories in column 'AlcoholDrinking':
AlcoholDrinking
No     298018
Yes     21777
Name: count, dtype: int64

Categories in column 'Stroke':
Stroke
No     307726
Yes     12069
Name: count, dtype: int64

Categories in column 'DiffWalking':
DiffWalking
No     275385
Yes     44410
Name: count, dtype: int64

Categories in column 'Sex':
Sex
Female    167805
Male      151990
Name: count, dtype: int64

Categories in column 'Race':
Race
White                             245212
Hispanic                           27446
Black                              22939
Other                              10928
Asian                               8068
American Indian/Alaskan Native      5202
Name: count, dtype: int64

Categories in column 'PhysicalActivity':
PhysicalActivity
```

3.3 Check for missing values in columns

In [26]:
```
# Check for missing values in columns
missing_values = heartD_df.isnull().sum()

# Print columns with missing values (if any)
columns_with_missing_values = missing_values[missing_values > 0]
if not columns_with_missing_values.empty:
    print("Columns with missing values:")
    print(columns_with_missing_values)
else:
    print("No missing values found in any column.")
```

```
No missing values found in any column.
```

4. Preprocessing

Identified Preprocessing steps:

- Extract only "Yes" and "No" values from the "Diabetic" column.
- Apply one-hot encoding to object type columns and Convert them into numeric representations.

In [27]:

```
# Create a new DataFrame for the converted columns
converted_df = heartD_df.copy()

# Extract only "Yes" and "No" values from the "Diabetic" column
converted_df['Diabetic'] = converted_df['Diabetic'].apply(lambda x: 'Yes' if x == 'Yes' else 'No')

# Define dictionaries for conversion
sex_cat_dct = {'Female': 0, 'Male': 1}
yes_no_dct = {'Yes': 1, 'No': 0}
genHealth_dct= {'Poor':1, 'Fair':2, 'Good':3, 'Very good':4, 'Excellent':5}

ageCategory_dct= {'18-24':0, '25-29':1, '30-34':2, '35-39':3, '40-44':4, '45-49':5,
   '50-54':6, '55-59':7, '60-64':8, '65-69':9, '70-74':10, '75-79':11,
   '80 or older':12}

# Define columns to be converted and their corresponding dictionaries
columns_to_convert = {
    'Sex': sex_cat_dct,
    'GenHealth': genHealth_dct,
    'AgeCategory': ageCategory_dct,
    'Diabetic':yes_no_dct,
    'HeartDisease': yes_no_dct,
    'Smoking': yes_no_dct,
    'AlcoholDrinking': yes_no_dct,
    'Stroke': yes_no_dct,
    'DiffWalking': yes_no_dct,
    'PhysicalActivity': yes_no_dct,
    'Asthma': yes_no_dct,
    'KidneyDisease': yes_no_dct,
    'SkinCancer': yes_no_dct,
}

# Convert columns using dictionaries
for column, dct in columns_to_convert.items():
    converted_df[column] = converted_df[column].map(dct)

# Apply one-hot encoding to the 'Race' column and concatenate with converted_df
race_encoded = pd.get_dummies(converted_df['Race'], dtype=int)
converted_df = pd.concat([converted_df, race_encoded], axis=1)
converted_df.drop('Race', axis=1, inplace=True)

# Remove duplicate rows considering all columns
df_no_duplicates = converted_df.drop_duplicates()

# Display the new DataFrame with converted columns
df_no_duplicates.head(20)
```

Out[27]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	...	SleepTime	Asthma	KidneyDisease	SkinCancer	American Indian/Alaskan Native	Asian	Black	Hispanic	Other	White
0	0	16.60	1	0	0	3.0	30.0	0	0	7	...	5.0	1	0	1	0	0	0	0	0	1
1	0	20.34	0	0	1	0.0	0.0	0	0	12	...	7.0	0	0	0	0	0	0	0	0	1
2	0	26.58	1	0	0	20.0	30.0	0	1	9	...	8.0	1	0	0	0	0	0	0	0	1
3	0	24.21	0	0	0	0.0	0.0	0	0	11	...	6.0	0	0	1	0	0	0	0	0	1
4	0	23.71	0	0	0	28.0	0.0	1	0	4	...	8.0	0	0	0	0	0	0	0	0	1
5	1	28.87	1	0	0	6.0	0.0	1	0	11	...	12.0	0	0	0	0	0	1	0	0	0

5. Set Independed Variables and Dependent Variables

In [28]:
```
df_no_duplicates.columns
```

Out[28]:
```
Index(['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma',
       'KidneyDisease', 'SkinCancer', 'American Indian/Alaskan Native',
       'Asian', 'Black', 'Hispanic', 'Other', 'White'],
      dtype='object')
```

In [29]:
```
# picked columns
feature_df = df_no_duplicates[['BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma',
       'KidneyDisease', 'SkinCancer', 'American Indian/Alaskan Native',
       'Asian', 'Black', 'Hispanic', 'Other', 'White']]

# Independed Variables
X = np.asarray(feature_df)

# Dependent variable
y = np.asarray(df_no_duplicates['HeartDisease'])
y[0:5]
```

Out[29]:
```
array([0, 0, 0, 0, 0])
```

6. Divide the data set as Train and Test data set

In [30]:
```
'''
heartD_df(319795 rows) ---> Train (80%), Test (20%)
X is 2D array
y is 1D array
'''

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=4)

# 31979 * 22
X_train.shape

# 287816 * 22
X_train.shape
df_no_duplicates.shape
```

Out[30]:
```
(301166, 23)
```

7. Modeling (SVM with Scikit-learn)

In [31]:
```
from sklearn import svm

classifier = svm.SVC(kernel='poly', degree=2)
classifier.fit(X_train[:10000], y_train[:10000])

y_predict = classifier.predict(X_test)
y_predict
```

Out[31]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

8. Evaluation

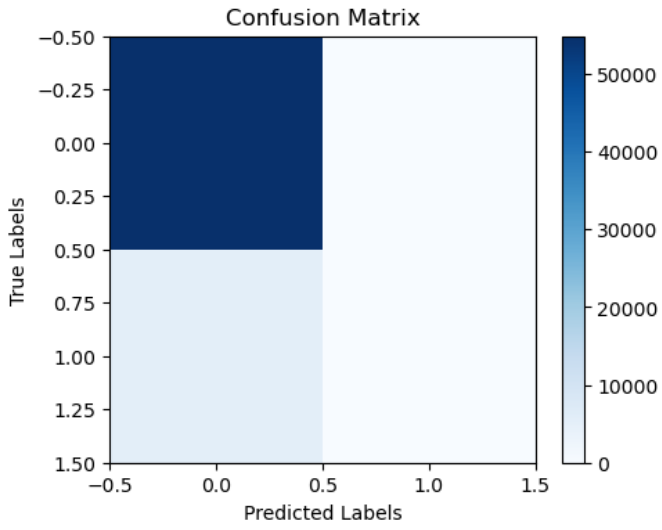
In [32]:
```
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

print("Accuracy score")
print(metrics.accuracy_score(y_test,y_predict))
```

```
Accuracy score
0.910449247933061
```

In [33]:
```
# Accuracy of predicted data with the actual data

train = classifier.score(X_train, y_train)

test = classifier.score(X_test, y_test)



print(f"training score = {train}")

print(f"testing score = {test}")



# Create DataFrame with actual and predicted values

comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_predict})


#Calculate and print performance metrics

predictions_tree = classifier.predict(X_test)

accuracy_tree = classifier.score(X_test, y_test)

confusion_matrix_tree = confusion_matrix(y_test, y_predict)

precision = precision_score(y_test, y_predict)

recall = recall_score(y_test, y_predict)

f1 = f1_score(y_test, y_predict)
```

```
#visualize confusion matrix

plt.figure(figsize=(5, 4))

plt.imshow(confusion_matrix_tree, cmap=plt.cm.Blues)

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.title("Confusion Matrix")

plt.colorbar()

plt.show()
```

```
training score = 0.9092565537164014
testing score = 0.910449247933061
Confusion matrix for SVM
[[54840     0]
 [ 5394     0]]
Accuracy for SVM = 91.04492479330611%
Precision for SVM = 0.0000
Recall for SVM = 0.0000
F1-score for SVM = 0.0000
/home/fernando-mrr/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classific
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



In [ ]:

RESULTS AND DISCUSSION

I. Accuracy Comparison of the models

The performance of a model depends on its loss function values and other scores,

Precision Score: The accuracy of positive predictions is measured by the Precision Score. It is defined as the percentage of accurately anticipated positive observations to all positive predictions. A low percentage of false positives is indicated by high precision.

Recall Score: A model's recall score indicates its capacity to identify every significant example in a dataset. It is the proportion of all observations made during the actual class to the correctly predicted positive observations.

F1 Score: Precision and Recall are weighted averages that make up the F1 Score. This score helps find a balance between precision and recall because it accounts for both erroneous positives and false negatives.

Confusion Matrix: A table known as a confusion matrix is frequently used to explain how well a classification model performs when applied to a collection of data for which the true values are known. The counts of true positives, true negatives, false positives, and false negatives are displayed in particular.

Log Loss: Log Loss restricts incorrect classifications to quantify a classifier's accuracy. A better model is indicated by a lower log loss; ideal models have a log loss of zero.

Accuracy Score: The ratio of accurately predicted observations to total observations is known as the accuracy score. It provides an easy-to-understand indicator of how many predictions a model correctly predicted.

Model	Precision	Recall	F1 Score	Confusion Matrix	Log Loss	Accuracy
Random Forest Classification	0.3224	0.1101	0.1642	[[80500 1883] [7237 896]]	-	0.8992
Logistic Regression	0.5303	0.1042	0.1742	[[54391 503] [4882 568]]	0.2402	0.9107
Decision Tree	0.2111	0.1878	0.1988	[[81831 5818] [6733 1557]]	-	0.8691
SVM	0.0000	0.0000	0.0000	[[54840 0] [5394 0]]	-	0.9104

II. Challenges

- The dataset is collected from the residence of the US population because of that the dataset is bias to that population leading to models to not perform well across diverse groups.
- Since the data is collected through a telephone survey data could be incomplete and inconsistent.

- Unexpected global and local situations like COVID-19 will lead to less accuracy of the models.
- Heart disease consists of a variety of conditions with different etiologies, risk factors and other factors that can be specific to a certain population, making it difficult to develop a universally accurate model.

III. Future Works (Areas of possible improvements)

- This dataset mainly focuses on behavioral factors of the patients, adding more important information such as electronic health records, wearable device data, and real-time monitoring systems can improve the forecast accuracy.
- Developing a model using an un-biased dataset that can be applied to any population as universally accurate model.
- Using hybrid forecast models can improve accuracy.
- Using more advanced modeling techniques such as deep learning and federate learning can improve the accuracy further.

Individual Contributions

IT21036620- Kariyawasam K.P.W.D.V.	Random Forest Algorithm
IT21055294- Kahandagamage P.N.	Logistic Regression
IT20613440- Samarawijaya W.G.M.P	Decision Tree
IT21067242- Fernando M.R.R.	Support Vector Machines

APPENDIX (Source Code)

I. Random Forest Algorithm –

```
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score,
f1_score,confusion_matrix

#Load the Heart disease dataset from the Excel file
dataset = pd.read_csv('heart_2020_cleaned.csv')

#Display the number of rows & columns
print("Rows and Columns:",dataset.shape)

#Display data types of the columns
print("Column List:\n",dataset.columns)

#Display data types of the columns
print("Data Types:\n",dataset.dtypes)

# Display first 5 records
dataset.head()

#display the number of null values in the dataset
print(dataset.isnull().sum())

#Checking the number of unique values in each column
print(dataset.nunique())

#Removing duplicates
dataset.drop_duplicates(inplace= True)
print("Dataset shape removing duplicates:", dataset.shape)

#Creating an object list including object datatype
obj_list = dataset.select_dtypes(include='object').columns

#Transform the objects in the columns into numeric values
le = LabelEncoder()
```

```
for obj in obj_list:
    dataset[obj] = le.fit_transform(dataset[obj].astype(str))

#view the processed dataset
Dataset

# Split the data into features and target
X = dataset.drop('HeartDisease', axis=1)
y = dataset['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Training the Random Forest model
rfc_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rfc_classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = rfc_classifier.predict(X_test)

#compare the accuracy of predicted data with the actual data
print(f'Training Score: {rfc_classifier.score(X_train, y_train)}')
print(f'Testing Score: {rfc_classifier.score(X_test, y_test)}')

#Create a Dataframe of actual values and predicted values
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Display the first 10 of the DataFrame
print("Comparison of Actual and Predicted values:")
print(df.head(10))

# Generating a plot for a visual comparison
plt.figure(figsize=(10, 6))
plt.plot(df[:50].reset_index(drop=True), marker='o')

print('Precision score:', precision_score(y_test, y_pred))
```

```
print('Recall score:', recall_score(y_test, y_pred))
print('Accuracy score:', accuracy_score(y_test, y_pred))
print('F1 score:', f1_score(y_test, y_pred))
print('Confusion Matrix:', confusion_matrix(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['No Heart Disease', 'Heart Disease'], yticklabels=['No Heart Disease', 'Heart Disease'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

II. Logistic Regression –

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, log_loss

# Load the dataset from excel sheet for processing
data_path = "heart_2020_cleaned.csv"
dataset = pd.read_csv(data_path)

# Check number of rows & columns in the dataset
```

```
print("Initial data shape:", dataset.shape)

# Check first five rows of dataset
print("First five rows of the dataset:\n", dataset.head())

#overview of the dataset
dataset.info()

# Removes duplicate rows from the dataset
dataset.drop_duplicates(inplace=True)

# Checking number of rows & columns after dropping duplicates
print("Dataset shape after dropping duplicate Values:", dataset.shape)

# Calculates and prints the number of missing values in each column.
missing_values = dataset.isnull().sum()
print("Missing Values in Each Column:\n", missing_values)

# Preprocessing, Encoding categorical variables
categorical_cols = dataset.select_dtypes(include=['object']).columns
data_encoded = pd.get_dummies(dataset, columns=categorical_cols, drop_first=True)

# Display the first few rows of the new dataframe to verify encoding
print("Encoded Dataframe First rows:\n", data_encoded.head())

#  Splits the data into training and testing sets
X = data_encoded.drop('HeartDisease_Yes', axis=1)  # Adjust the column name if necessary
y = data_encoded['HeartDisease_Yes']  # Adjust the column name if necessary
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizes the features to have zero mean and unit variance
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initializes and trains the logistic regression model with a higher number of iterations
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train_scaled, y_train)

# Predictions on the test set
y_pred = logistic_model.predict(X_test_scaled)
y_pred_proba = logistic_model.predict_proba(X_test_scaled)[:, 1]

# Training and Testing scores
train_score = logistic_model.score(X_train_scaled, y_train)
test_score = logistic_model.score(X_test_scaled, y_test)

print("Training Score:", train_score)
print("Testing Score:", test_score)

# Calculates and prints various performance metrics
print("Precision Score:", precision_score(y_test, y_pred))
print("Recall Score:", recall_score(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))

# Log Loss metrics
log_loss_val = log_loss(y_test, y_pred_proba)
print("Log Loss:", log_loss_val)

# Displays a detailed classification report and confusion matrix
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Classification Report:\n", report)
```

```
print("Confusion Matrix:\n", conf_matrix)


# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No Heart Disease',
'Heart Disease'], yticklabels=['No Heart Disease', 'Heart Disease'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()


# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)


plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

III. Decision Tree Algorithm-

```
#importing libraries
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.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score



#load the dataset
df = pd.read_csv('heart_2020_cleaned.csv')

#displays the first few rows of a DataFrame
df.head()

#check for missing values
df.isnull().sum()

#get summary statistics
df.describe()

#find unique values
df.nunique()

#preprocessing
df = df[df.columns].replace({'Yes':1, 'No':0, 'Male':1, 'Female':0, 'No, borderline diabetes':0,'Yes (during pregnancy)':1, 'Y':1, 'N':0})
df.head()

#Split dataset for training and testing
y=df['HeartDisease']
X=df.drop(['HeartDisease', 'AgeCategory', 'Race', 'GenHealth'], axis=1)
```

```
#split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

#Train the decision tree model
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# predict the test set resut
predictions_tree = clf.predict(X_test)

# Accuracy of predicted data with the actual data
train = clf.score(X_train, y_train)
test = clf.score(X_test, y_test)

print(f"training score = {train}")
print(f"testing score = {test}")

# Create DataFrame with actual and predicted values
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': predictions_tree})

# Print the DataFrame (optional)
print(comparison_df.head(10))

#Calculate and print performance metrics
predictions_tree = clf.predict(X_test)
accuracy_tree = clf.score(X_test, y_test)
confusion_matrix_tree = confusion_matrix(y_test, predictions_tree)
precision = precision_score(y_test, predictions_tree)
recall = recall_score(y_test, predictions_tree)
f1 = f1_score(y_test, predictions_tree)
```

```
print("Confusion matrix for Decision Tree")

print(confusion_matrix_tree)

print(f"Accuracy for Decision Tree = {accuracy_tree*100}%")

print(f"Precision for Decision Tree = {precision:.4f}")

print(f"Recall for Decision Tree = {recall:.4f}")

print(f"F1-score for Decision Tree = {f1:.4f}")



#visualize confusion matrix

plt.figure(figsize=(5, 4))

plt.imshow(confusion_matrix_tree, cmap=plt.cm.Blues)

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.title("Confusion Matrix")

plt.colorbar()

plt.show()
```

IV. SVM –

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


heartD_df = pd.read_csv('heart_2020_cleaned.csv')
heartD_df.head()
#heartD_df.size
heartD_df.shape

heartD_df.dtypes

3.2 Checking Categories Across Multiple Columns


# List of columns contains unique categories
columns_to_check = ['HeartDisease', 'Smoking', 'AlcoholDrinking',
                    'Stroke', 'DiffWalking', 'Sex', 'Race',
```

'PhysicalActivity', 'GenHealth', 'Asthma', 'KidneyDisease', 'SkinCancer', 'AgeCategory']

```
for column in columns_to_check:
    print(f"Categories in column '{column}':")
    print(heartD_df[column].value_counts())
    print()
```

3.3 Check for missing values in columns

In [26]:

```
# Check for missing values in columns
missing_values = heartD_df.isnull().sum()

# Print columns with missing values (if any)
columns_with_missing_values = missing_values[missing_values > 0]
if not columns_with_missing_values.empty:
    print("Columns with missing values:")
    print(columns_with_missing_values)
else:
    print("No missing values found in any column.")
```

4. Preprocessing

Identified Preprocessing steps:

- Extract only "Yes" and "No" values from the "Diabetic" column.
- Apply one-hot encoding to object type columns and Convert them into numeric representations.

In [27]:

```
# Create a new DataFrame for the converted columns
converted_df = heartD_df.copy()

# Extract only "Yes" and "No" values from the "Diabetic" column
converted_df['Diabetic'] = converted_df['Diabetic'].apply(lambda x: 'Yes' if x == 'Yes' else 'No')

# Define dictionaries for conversion
sex_cat_dct = {'Female': 0, 'Male': 1}
yes_no_dct = {'Yes': 1, 'No': 0}
genHealth_dct= {'Poor':1, 'Fair':2, 'Good':3, 'Very good':4, 'Excellent':5}

ageCategory_dct= {'18-24':0, '25-29':1, '30-34':2, '35-39':3, '40-44':4, '45-49':5,
  '50-54':6, '55-59':7, '60-64':8, '65-69':9, '70-74':10, '75-79':11,
  '80 or older':12}

# Define columns to be converted and their corresponding dictionaries
columns_to_convert = {
    'Sex': sex_cat_dct,
    'GenHealth': genHealth_dct,
    'AgeCategory': ageCategory_dct,
    'Diabetic':yes_no_dct,
    'HeartDisease': yes_no_dct,
    'Smoking': yes_no_dct,
    'AlcoholDrinking': yes_no_dct,
    'Stroke': yes_no_dct,
    'DiffWalking': yes_no_dct,
```

```
    'PhysicalActivity': yes_no_dct,
    'Asthma': yes_no_dct,
    'KidneyDisease': yes_no_dct,
    'SkinCancer': yes_no_dct,
}

# Convert columns using dictionaries
for column, dct in columns_to_convert.items():
    converted_df[column] = converted_df[column].map(dct)

# Apply one-hot encoding to the 'Race' column and concatenate with converted_df
race_encoded = pd.get_dummies(converted_df['Race'], dtype=int)
converted_df = pd.concat([converted_df, race_encoded], axis=1)
converted_df.drop('Race', axis=1, inplace=True)

# Remove duplicate rows considering all columns
df_no_duplicates = converted_df.drop_duplicates()

# Display the new DataFrame with converted columns
df_no_duplicates.head(20)
```

5. Set Independed Variables and Dependent Variables

In [28]:

```
df_no_duplicates.columns
```

Out[28]:

```
Index(['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma',
       'KidneyDisease', 'SkinCancer', 'American Indian/Alaskan Native',
       'Asian', 'Black', 'Hispanic', 'Other', 'White'],
      dtype='object')
```

In [29]:

```
# picked columns
feature_df = df_no_duplicates[['BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime', 'Asthma',
       'KidneyDisease', 'SkinCancer', 'American Indian/Alaskan Native',
       'Asian', 'Black', 'Hispanic', 'Other', 'White']]

# Independed Variables
X = np.asarray(feature_df)

# Dependent variable
y = np.asarray(df_no_duplicates['HeartDisease'])
y[0:5]
```

6. Divide the data set as Train and Test data set

In [30]:

```
'''
heartD_df(319795 rows) ---> Train (80%), Test (20%)
X is 2D array
y is 1D array
'''

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=4)

# 31979 * 22
X_train.shape

# 287816 * 22
X_train.shape
df_no_duplicates.shape
```

7. Modeling (SVM with Scikit-learn)

In [31]:

```
from sklearn import svm

classifier = svm.SVC(kernel='poly', degree=2)
classifier.fit(X_train[:10000], y_train[:10000])

y_predict = classifier.predict(X_test)
y_predict
```

8. Evaluation

In [32]:

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

print("Accuracy score")
print(metrics.accuracy_score(y_test,y_predict))
```

```
Accuracy score
0.910449247933061
```

In [33]:

```
# Accuracy of predicted data with the actual data

train = classifier.score(X_train, y_train)
```

```
test = classifier.score(X_test, y_test)


print(f"training score = {train}")

print(f"testing score = {test}")


# Create DataFrame with actual and predicted values

comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_predict})


#Calculate and print performance metrics

predictions_tree = classifier.predict(X_test)

accuracy_tree = classifier.score(X_test, y_test)

confusion_matrix_tree = confusion_matrix(y_test, y_predict)

precision = precision_score(y_test, y_predict)

recall = recall_score(y_test, y_predict)

f1 = f1_score(y_test, y_predict)


print("Confusion matrix for SVM")

print(confusion_matrix_tree)

print(f"Accuracy for SVM = {accuracy_tree*100}%")

print(f"Precision for SVM = {precision:.4f}")

print(f"Recall for SVM = {recall:.4f}")

print(f"F1-score for SVM = {f1:.4f}")
```

```
#visualize confusion matrix

plt.figure(figsize=(5, 4))

plt.imshow(confusion_matrix_tree, cmap=plt.cm.Blues)

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.title("Confusion Matrix")

plt.colorbar()

plt.show()
```