AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

408/1, Kuratoli, Khilkhet, Dhaka 1229, Bangladesh

Assignment Title: Final Term project Report

Assignment No: Final Report | Date of Submission: 18.01.25

Course Title: PROGRAMMING IN PYTHON

Course Code: **00789** | Section: A

Semester: 11 | Session: 24-25 | Course Teacher: **DR. ABDUS SALAM**

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student's work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaborationhas been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the Faculty for review and comparison, including review by external examiners.

Group No : 09

No	Name	ID	Program	Signature
1	MD. Shanzid hasan	21-44983-2	CSE	
2	Nahar Islam	21-44971-2	CSE	

1. Dataset Overview

The dataset contains 4,998 rows and 20 features, with no missing values. It includes demographic, lifestyle, clinical, and diagnostic attributes such as age, gender, BMI, smoking, hypertension, cholesterol levels, and more. The target variable, "Heart_Attack_Risk," is categorical, providing labels like "High" for classification tasks. The dataset's diverse range of features makes it ideal for studying heart attack risk prediction, feature engineering, and machine learning model development.

2. Code Description

2.1 Task Mount your google drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

To access files stored on Google Drive in Google Colab, the drive.mount() function is used. The following code snippet mounts the Google Drive to a directory (/content/drive), making the files available for use in the Colab environment. This is essential for loading and saving data directly from Google Drive while working in Colab.

2.2 Task Import all necessary libraries.

```
import pandas as pd import numpy as np from sklearn import
tree from sklearn import metrics import math from
sklearn.utils import shuffle import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder from
sklearn.model_selection import train_test_split from
sklearn.svm import SVC from sklearn.metrics import
accuracy_score
from sklearn.metrics import confusion_matrix,
classification_report
```

This task involves importing necessary libraries for data processing, machine learning, and evaluation. The following libraries are imported:

1. `pandas` and `numpy` for handling and manipulating data.
2. `tree` and `SVC` from `sklearn` for building decision tree and support vector machine models, respectively.
3. `metrics` for evaluating model performance.
4. `math` for mathematical functions.
5. `shuffle` from `sklearn.utils` to shuffle the dataset.
6. `matplotlib.pyplot` for visualizing the results.

7. `LabelEncoder` for encoding categorical labels.

8. `train_test_split` for splitting the dataset into training and test sets.

9. `accuracy_score`, `confusion_matrix`, and `classification_report` for model evaluation metrics.

2.3 Task 1 Read/Load the dataset file in your program. Use Pandas library to complete this task.

```
np.random.seed(123)
df = pd.read_csv('/content/drive/My
Drive/heart_attack_risk_dataset.csv') df = shuffle(df)
print(df)
```

In this task, the dataset is loaded from a CSV file located on Google Drive using pandas.read_csv(). The dataset is then shuffled using shuffle() from sklearn.utils to randomize the order of the data, which helps in avoiding biases during training. The np.random.seed(123) is used to ensure reproducibility of the shuffling process. The print(df) statement outputs the shuffled dataset for inspection.

2.4 Task 2 Apply appropriate data cleaning techniques to the dataset. In this step, replace bad data using proper methods and do not delete any record except duplicate records. Use Pandas library to complete this task.

```
print(df.info()) df.drop_duplicates(inplace
= True) print(df.info())
```

This task involves inspecting the dataset's structure and cleaning it by removing duplicate entries. The df.info() function is used to display information about the dataset, such as the number of entries, columns, and data types. After inspecting the dataset, duplicates are removed using df.drop_duplicates(inplace=True), which ensures that only unique rows remain. The inplace=True argument modifies the dataset directly without needing to assign it back to df. Finally, df.info() is called again to verify the removal of duplicates.

2.5 Task 3 Draw graphs to analyze the frequency distributions of the features. Use Matplotlib library to complete this task. Draw all the plots in a single figure so that all plots can be seen in one diagram (use subplot() function).

```
feature_columns = ['Age', 'Gender', 'Smoking',
'Alcohol_Consumption','Physical_Activity_Level','BMI','Diabetes','Hyper
tension','Cholesterol_Level','Resting_BP','Heart_Rate','Family_History'
,'Stress_Level','Chest_Pain_Type','Thalassemia','Fasting_Blood_Sugar','
ECG_Results','Exercise_Induced_Angina','Max_Heart_Rate_Achieved']
 plt.figure(figsize=(18, 15)) for i, column in
enumerate(feature_columns, 1):
    plt.subplot(5, 4, i)     plt.hist(df[column], bins=30,
color='teal', edgecolor='black')     plt.title(f'{column}
Distribution')     plt.xlabel(column)     plt.ylabel('Frequency')
 plt.tight_layout()
plt.show()
```

This task involves visualizing the distributions of the selected features in the dataset. A list of feature columns, feature_columns, is defined, and a histogram is plotted for each feature using matplotlib.pyplot. The plt.subplot(5, 4, i) creates a grid of subplots (5 rows and 4 columns) to display all the histograms. Each subplot shows the distribution of a feature using plt.hist(), with 30 bins and a teal color scheme. Titles and labels are added to each subplot for clarity. The plt.tight_layout() ensures that the subplots are spaced appropriately, and plt.show() displays the final visualization.

2.6 Task 4 Perform scaling to the features of the dataset. Remember that you will need to apply data conversion before performing scaling whenever necessary.

```
categorical_columns = [
    'Gender', 'Physical_Activity_Level', 'Stress_Level',
    'Chest_Pain_Type', 'Thalassemia', 'ECG_Results',
'Heart_Attack_Risk'
]  encoded_values
= {}
 for column in
categorical_columns:
    encoder = LabelEncoder()
encoder.fit(df[column])

    encoded_values[column] =
dict(zip(encoder.classes_,
encoder.transform(encoder.classes_)))     df[column] =
encoder.transform(df[column])
 for column, mapping in
encoded_values.items():
    print(f'{column}: {mapping}')
print(df.head())
```

This task involves encoding categorical variables in the dataset using LabelEncoder from sklearn.preprocessing. A list of categorical columns, categorical_columns, is defined. For each column in this list, a LabelEncoder is instantiated, and its fit() method is called to learn the unique values in the column. The transform() method is then applied to convert the categorical values into numerical labels. The mappings between the original categories and their corresponding numerical labels are stored in the encoded_values dictionary. After encoding, the dataset's columns are updated with the transformed numerical values. The mappings for each encoded column are printed to ensure proper encoding, followed by the first few rows of the updated dataset using df.head().

2.7 Task 5 Split your data into two parts: Training dataset and Testing dataset. You must use the function train_test_split() to complete this task and use value 3241 as the value of the random_state parameter of this function.

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=3241) print(x_train.shape)
print(x_test.shape) print(y_train.shape)
print(y_test.shape)
```

The dataset is split into training and test sets using train_test_split() from sklearn.model_selection. The feature variables (x) and target variable (y) are separated, and then 80% of the data is used for training (x_train, y_train) and 20% for testing (x_test, y_test). The test_size=0.2 argument specifies the 80/20 split, and random_state=3241 ensures reproducibility of the split. The shapes of the resulting datasets are printed to confirm the split.

2.8 Task 6 Apply Support Vector Machine (SVM) Classifier to the dataset. Build (train) your prediction model in this step.

```
svm_classifier          =          SVC(kernel='linear')
svm_classifier.fit(x_train,  y_train)  y_pred_train  =
svm_classifier.predict(x_train)      y_pred_test      =
svm_classifier.predict(x_test)        accuracy        =
accuracy_score(y_train, y_pred_train)
print(f'Accuracy of the SVM model: {accuracy:.4f}')
```

Support Vector Machine (SVM) classifier is trained using the SVC class from sklearn.svm with a linear kernel. The classifier is fitted to the training data (x_train, y_train) using the fit() method. Predictions are then made for both the training set (y_pred_train) and the test set (y_pred_test) using the predict() method. The accuracy of the model is calculated using accuracy_score() by comparing the predicted values

(y_pred_train) with the actual training labels (y_train). The resulting accuracy score is printed to assess the performance of the model.

2.9 Task 7 Calculate the confusion matrix for your model. Interpret it in detail in the report.

```
confusion_matrix = metrics.confusion_matrix(y_test, y_pred_test)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix) cm_display.plot()
plt.show()
```

The performance of the trained SVM model is further evaluated using a confusion matrix. The metrics.confusion_matrix() function from sklearn.metrics is used to generate the confusion matrix by comparing the true labels (y_test) with the predicted labels (y_pred_test). The confusion matrix is then visualized using metrics.ConfusionMatrixDisplay(). The plot() method is used to display the confusion matrix as a graphical plot, and plt.show() renders the plot. This allows for a detailed assessment of the model's classification performance, including true positives, true negatives, false positives, and false negatives.

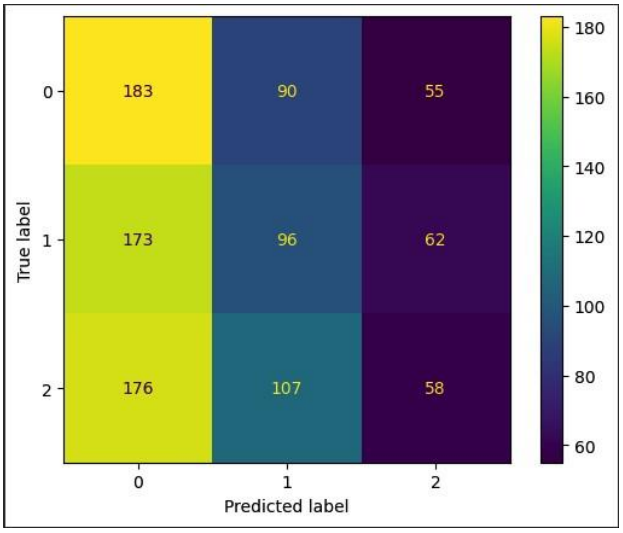

Fig: Confusion matrix

The confusion matrix shows the model's performance across three classes (0, 1, and 2). It performs best for Class 0, with 183 correct predictions but struggles with Class 1 (96 correct) and Class 2 (58 correct). There is significant misclassification, especially between adjacent classes. The overall accuracy is approximately 30.64% (337 correct predictions out of 1100). The model has difficulty distinguishing between the classes, particularly Class 2, and would benefit from improvements like dataset balancing or feature optimization.

2.10 Task 8 Calculate the train and test accuracy of your model and compare them.

```
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy  =  accuracy_score(y_test,  y_pred_test)
print(f'Training   Accuracy:   {train_accuracy:.4f}')
print(f'Testing Accuracy: {test_accuracy:.4f}')
```

The accuracy of the SVM model is calculated separately for both the training and test sets. The accuracy_score() function is used to compute the accuracy by comparing the predicted values (y_pred_train and y_pred_test) with the actual values (y_train and y_test). The training accuracy is calculated using the training data, while the testing accuracy is calculated using the test data. Both accuracy scores are then printed to provide insights into the model's performance on both the training and test datasets.