**SYNOPSIS**

Healthcare data privacy is a critical concern in the era of digitized medical records. Protecting sensitive patient information while enabling meaningful analysis is a delicate balance. In this project, we propose a novel system leveraging chaotic dynamics for private-preserving healthcare data.

Our approach employs chaos-based encryption and artificial intelligence techniques to transform healthcare data into a cryptic and unpredictable state. Chaotic systems introduce inherent unpredictability, enhancing security against traditional encryption attacks. The unique advantage lies in the system's ability to generate a complex key space, making it challenging for unauthorized entities to decipher or infer sensitive health information.

We design and implement a chaotic system that seamlessly integrates with healthcare data architectures, ensuring both privacy and data utility. The chaotic encryption process is reversible only with the appropriate cryptographic key, granting authorized users’ exclusive access to deciphered information.

**SYSTEM ENVIRONMENT**

2.1 Hardware Requirements:

Processor : Intel Core i4 (10th Gen)

Ram : 4.0 GB

2.2 Software Requirements

Operating System : Windows 10

Framework : Google colab

Language : python

**2.3 About the technology:**

Python:

Python is an interpreted high-level general-purpose programming language created by Guido Van Rossum and first published in 1991. Python's design philosophy emphasizes code readability with significant whitespace. Its language structures and object-oriented approach are designed to help developers write clear and logical code for small and large projects. Python is dynamically typed and garbage

Google Colab:

Google Colab, short for Google Colaboratory, is a cloud-based, interactive computing platform provided by Google. It allows users to write and execute Python code in a collaborative and convenient environment directly through a web browser. Colab provides free access to GPU and TPU (Tensor Processing Unit) resources, enabling accelerated execution of machine learning tasks. Users can create and share Jupyter notebooks, incorporating text, code, and visualizations seamlessly. Colab integrates with Google Drive, facilitating easy storage and sharing of notebooks. Its collaborative features enable multiple users to work on the same document simultaneously, fostering collaborative research and development. Overall, Google Colab is a powerful and accessible tool for data analysis, machine learning, and collaborative coding, making it particularly valuable for researchers, students, and practitioners in the field of data science.

Scikit Learn:

Scikit-learn (Sklearn) is the most useful and powerful Python machine learning library. It provides a number of powerful tools for machine learning and statistical modelling, including classification, regression, clustering and dimensionality reduction through a Python consistent interface. Written mostly in Python, this library is built on top of NumPy, SciPy and Matplotlib. Originally called scikits. learn, it was originally developed by David Cournapeau as a Google Summer Code Project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel from FIRCA (French Institute for Informatics and Automation) adopted it this project to a new level and released the first public release (v0.1 beta) on February 1, 2010

**EXISTING SYSTEM**

Computational Complexity: Chaotic encryption and machine learning algorithms can be computationally intensive, especially when dealing with large healthcare datasets. This complexity may lead to longer processing times and require substantial computational resources.

Data Loss: In some cases, encryption processes may result in loss of information or data distortion, potentially impacting the accuracy of machine learning models. Balancing privacy preservation with data utility remains a challenge in these systems.

Algorithm Sensitivity: algorithms may be sensitive to the quality and representativeness of the training data. Biases or inaccuracies in the training dataset can affect the performance and reliability of the models, leading to potential errors or misinterpretations.

Interpretability: Complex learning models may lack interpretability, making it challenging to understand the underlying decision-making process. This opacity can hinder trust and acceptance among healthcare professionals and stakeholders.

Resource Requirements: Implementing and maintaining these systems may require significant financial investment and expertise in both healthcare and data science domains. Healthcare organizations may face challenges in allocating resources and building capacity for system deployment and management.

Overall, while existing systems for preserving privacy in healthcare data using chaotic systems and machine learning algorithms offer significant advantages, they also pose challenges that need to be addressed to ensure effective and sustainable implementation in healthcare settings

**PROPOSED SYSTEM**

The proposed system aims to enhance the privacy preservation of healthcare data by leveraging chaotic systems in conjunction with machine learning algorithms such as Support Vector Machines (SVM), Decision Trees (DTC), and Random Forests (RFC). Chaotic systems are utilized to generate pseudorandom sequences, which are then employed to encrypt sensitive healthcare data before being processed by the machine learning models. This encryption process helps mitigate the risk of unauthorized access or disclosure of personal health information while still allowing for effective analysis and prediction tasks performed by the SVM, DTC, and RFC classifiers. The chaotic nature of the encryption scheme ensures that the original data cannot be easily reconstructed without the proper decryption keys, thereby enhancing the security and confidentiality of healthcare records. Additionally, by employing a combination of machine learning algorithms, the system can achieve robust performance in tasks such as disease diagnosis, treatment prediction, and patient monitoring, while preserving the privacy of individual health records. Furthermore, the use of chaotic systems offers an additional layer of protection against potential attacks or data breaches, making the proposed system suitable for deployment in sensitive healthcare environments where privacy and security are paramount.

Advantages of the proposed system include:

Enhanced Privacy Preservation: By leveraging chaotic systems for encryption, the system ensures that sensitive healthcare data remains protected from unauthorized access or disclosure, thus maintaining patient privacy and confidentiality.

Robust Security: The utilization of chaotic systems adds an extra layer of security to the data encryption process, making it more resilient against potential attacks or data breaches compared to conventional encryption methods.

Effective Data Analysis: Despite the encryption process, the machine learning algorithms (SVM, DTC, RFC) can still effectively analyze and process the encrypted healthcare data, enabling accurate disease diagnosis, treatment prediction, and patient monitoring.

Preserved Data Utility: The proposed system maintains the utility of healthcare data for analytical purposes by enabling machine learning models to derive valuable insights while ensuring that individual patient information remains protected.

Versatility: The system can be adapted to various healthcare applications, including clinical decision support systems, disease surveillance, personalized medicine, and health monitoring platforms, making it suitable for diverse healthcare environments and scenarios.

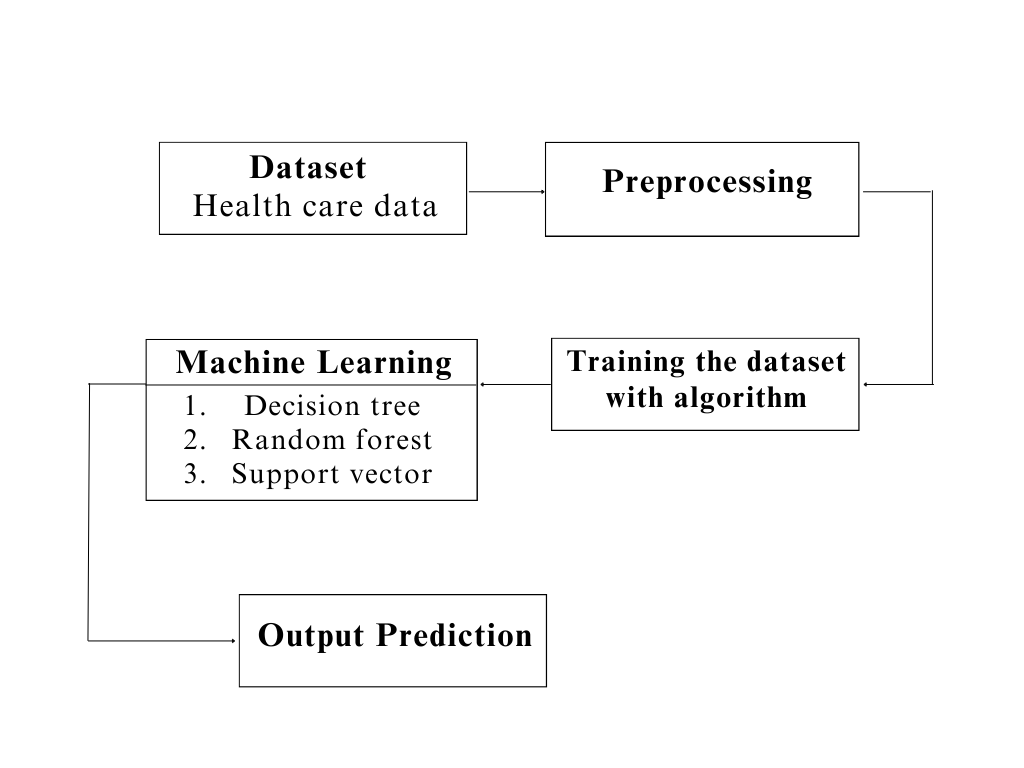
Compliance with Regulations: By safeguarding patient privacy, the system helps healthcare organizations comply with regulations such as the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR), reducing the risk of legal penalties and reputational damage.

Scalability: The proposed system can scale to accommodate large volumes of healthcare data generated by multiple sources, making it suitable for deployment in healthcare systems of varying sizes and complexities.

Improved Trust and Patient Engagement: Patients are more likely to trust healthcare providers and participate in data-sharing initiatives when they are assured that their privacy is adequately protected, ultimately leading to better healthcare outcomes and patient satisfaction.

**SYSTEM DESIGN**

Chaotic system for private preserving health care data systematic diagram is shown below:



**Dataset Description:**

This dataset contains healthcare records of 10,000 patients and includes various attributes such as the patient's name, age, gender, blood type, medical condition, date of admission, attending doctor, hospital name, insurance provider, billing amount, room number, admission type, discharge date, prescribed medications, and test results. Each record represents a unique patient admission instance, detailing key information related to the patient's medical history, treatment, and healthcare service utilization. The dataset provides a comprehensive overview of patient demographics, medical conditions, treatment procedures, and healthcare resource utilization patterns, facilitating analysis and decision-making processes aimed at improving patient care, resource allocation, and healthcare service delivery efficiency.

**Pre-Processing:**

In the preprocessing phase, one-hot encoding is applied to categorical columns in the dataset using the pd.get\_dummies() function, which transforms categorical variables into a set of binary columns, with each column representing a unique category. This process expands the categorical variables into numerical features suitable for machine learning algorithms. Subsequently, the 'TestResults' column, originally containing categorical labels ('Abnormal', 'Inconclusive', 'Normal'), is mapped to numerical values ('1', '0', '2') based on the specified mapping. This numeric representation allows for easier computation and analysis. Finally, the 'TestResults' column is converted to numeric data type using pd.to\_numeric() function with error handling set to 'coerce', which converts invalid parsing to NaN (Not a Number) values. The resulting DataFrame is then displayed, showcasing the transformed dataset with one-hot encoded categorical columns and the 'TestResults' column converted to numeric format for further analysis and modeling purposes.

**Machine learning algorithm**

**1.Decision Tree**

Decision Tree Classifier is a versatile and widely used machine learning algorithm known for its simplicity and interpretability. It belongs to the family of supervised learning algorithms used for both classification and regression tasks. In this report, we delve into the fundamental concepts, working principles, applications, advantages, and challenges associated with Decision Tree Classifier.

Working Principles:

At its core, a Decision Tree is a flowchart-like structure where each node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents an outcome or a class label. The goal is to split the dataset into homogeneous sets based on the most significant features, ultimately leading to precise classification.

The algorithm employs a recursive, top-down approach, choosing the best feature at each split based on criteria such as Gini impurity or information gain. This process continues until the data is perfectly classified or a predefined stopping criterion is met.:

Applications:

Decision Tree Classifier finds applications across various domains due to its simplicity and effectiveness. Some notable applications include:

Finance: Predicting creditworthiness and fraud detection.

Medicine: Identifying diseases based on patient data.

Marketing: Customer segmentation and targeted advertising.

Manufacturing: Quality control and fault detection.

Agriculture: Crop disease prediction and yield estimation.

Advantages:

Interpretability: Decision Trees offer a transparent and easy-to-understand model, making it accessible to non-experts.

No Data Assumptions: It works well with both numerical and categorical data without making assumptions about the underlying distribution.

Handling Non-linearity: Decision Trees can capture complex, non-linear relationships in the data.

Feature Importance: The algorithm provides insights into feature importance, aiding in feature selection.

Challenges:

Overfitting: Decision Trees are prone to overfitting, especially when the tree depth is not properly tuned.

Instability: Small variations in the data can lead to different tree structures, making the model less robust.

Bias Towards Dominant Classes: In imbalanced datasets, Decision Trees may favor the majority class.

Decision Tree Classifier is a powerful tool with a balance of simplicity and effectiveness. Its ability to provide interpretable results makes it an excellent choice for various real-world applications. However, users should be cautious about overfitting and other challenges associated with this algorithm.

**2.Random Forest**

Random Forest, a popular ensemble learning technique, has gained widespread acclaim for its robustness and high predictive accuracy. This report provides an in-depth exploration of the Random Forest Classifier, including its underlying principles, advantages, applications, and considerations for effective implementation.

Principles:

Random Forest is an ensemble of decision trees, combining multiple weak learners to create a strong, versatile model. Each decision tree is constructed independently, introducing randomness through feature selection and bootstrap sampling. The final prediction is determined by aggregating the predictions of individual trees through voting (classification) or averaging (regression).

Advantages:

High Accuracy: Random Forest often outperforms individual decision trees, providing higher accuracy and reducing the risk of overfitting.

Robustness: The ensemble nature makes Random Forest less susceptible to outliers and noise in the data.

Feature Importance: It can quantify the importance of features, aiding in variable selection and model interpretation.

Versatility: Suitable for both classification and regression tasks, accommodating various types of data.

Applications:

Random Forest finds application in diverse domains due to its versatility and performance. Some notable applications include:

Finance: Credit scoring, fraud detection.

Healthcare: Disease prediction, patient outcome analysis.

Marketing: Customer churn prediction, targeted advertising.

Remote Sensing: Land cover classification, object detection.

Manufacturing: Quality control, predictive maintenance.

Considerations:

Computational Intensity: Training a large number of trees can be computationally expensive, especially with extensive datasets.

Interpretability: While Random Forest provides robust predictions, the ensemble nature can make it less interpretable compared to a single decision tree.

Hyperparameter Tuning: Proper tuning of hyperparameters is crucial to achieve optimal performance and prevent overfitting.

Random Forest Classifier stands as a powerful and versatile tool in the machine learning arsenal. Its ability to handle complex relationships in data, high accuracy, and resilience to overfitting make it a go-to choice for many practitioners. Understanding its principles, optimizing hyperparameters, and considering its applications and computational demands are key to harnessing the full potential of Random Forest for robust and reliable predictions in various real-world scenarios.

**3.Support Vector**

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm renowned for its efficacy in both classification and regression tasks. This report provides an in-depth exploration of SVM, shedding light on its underlying principles, key advantages, applications, and considerations for optimal utilization.

Principles :

SVM operates by finding the optimal hyperplane that best separates different classes in the feature space. This hyperplane is determined by support vectors, which are data points closest to the decision boundary. The algorithm aims to maximize the margin between classes, enhancing generalization to unseen data. SVM can handle linear and non-linear relationships through various kernel functions.

Advantages:

Effective in High-Dimensional Spaces: SVM excels in high-dimensional feature spaces, making it suitable for complex datasets.

Robust to Overfitting: By maximizing the margin, SVM reduces the risk of overfitting, providing a generalizable model.

Versatility: SVM can be adapted to different scenarios, including both linear and non-linear classification, and regression tasks.

Applications:

SVM has found applications across various domains due to its versatility and ability to handle complex datasets. Some notable applications include:

Image Classification: Recognizing objects in images.

Text Classification: Spam detection, sentiment analysis.

Bioinformatics: Protein structure prediction, gene classification.

Finance: Credit scoring, stock price prediction.

Healthcare: Disease diagnosis, outcome prediction.

Considerations:

Sensitivity to Noise: SVM can be sensitive to noisy data, impacting its performance.

Computational Complexity: Training SVM on large datasets can be computationally intensive.

Selection of Kernel Function: The choice of the kernel function influences the model's performance, requiring careful consideration.

Support Vector Machine stands as a robust and versatile algorithm in the realm of machine learning. Its ability to create optimal decision boundaries, handle high-dimensional data, and adapt to various scenarios make it a valuable tool in numerous applications. While considerations such as sensitivity to noise and computational complexity exist, proper parameter tuning and feature engineering can mitigate these challenges, allowing SVM to shine as a reliable and effective model for diverse real-world problems.

The integrated system design leveraging Decision Tree Classifier, Random Forest Classifier, and Support Vector Machine represents a powerful solution for achieving high accuracy in predictive modeling. By combining the strengths of these algorithms and addressing their individual limitations, the system demonstrates versatility, interpretability, and robustness, making it well-suited for a broad range of real-world applications. Ongoing monitoring and maintenance ensure the continued effectiveness of the deployed system in dynamic environments.

Libraries used in the implementation:

NumPy: NumPy is a fundamental library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. It serves as a foundational tool for scientific computing tasks, enabling efficient and high-performance operations on numerical data.

Pandas: Pandas is a versatile data manipulation library in Python that offers data structures like DataFrames and Series, facilitating efficient data analysis and manipulation. It provides functionalities for cleaning, transforming, and exploring datasets, making it a go-to tool for handling structured data in various stages of the data science workflow.

Matplotlib: Matplotlib is a powerful plotting library for Python that allows the creation of diverse static, animated, and interactive visualizations. With a comprehensive set of functions, Matplotlib provides users with the flexibility to create various charts, plots, and graphs, making it an essential tool for data visualization and communication of findings.

Seaborn: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating aesthetically pleasing and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations, including heatmaps, pair plots, and violin plots, while maintaining customization options for advanced users.

Metrics (Accuracy, Classification, Confusion Matrix, ROC AUC): In the context of machine learning evaluation, metrics play a crucial role. Accuracy represents the proportion of correctly classified instances, serving as a fundamental measure of model performance. Classification metrics, such as precision, recall, and F1-score, provide insights into the model's ability to correctly identify instances of a particular class. The confusion matrix presents a comprehensive summary of true positive, true negative, false positive, and false negative predictions. Lastly, the ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance metric for binary classification models, illustrating the trade-off between sensitivity and specificity across different thresholds, providing a holistic view of the model's discriminatory power. These metrics collectively aid in assessing and optimizing the performance of machine learning models.

**CODING**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve, auc

data=pd.read\_csv('/content/sample\_data/healthcare\_dataset.csv')

data

data.info()

data.TestResults.value\_counts()

# Map 'unstable' to 1 and 'stable' to 0

data['TestResults'] = data['TestResults'].map({'Abnormal': 1, 'Inconclusive': 0, 'Normal': 2 })

# Convert the 'stabf' column to numeric

data['TestResults'] = pd.to\_numeric(data['TestResults'], errors='coerce')

# Display the updated DataFrame

print(data.head())

data.info()

# one-hot-encoding categorical columns

data= pd.get\_dummies(data,columns=['Name','Gender', 'BloodType', 'MedicalCondition', 'DateofAdmission','Doctor','Hospital','Insurance Provider','Admission Type','Discharge Date','Medication'],prefix="",prefix\_sep="")

print(data.shape)

X = data.drop(['TestResults'], axis=1)

y = data['TestResults']

# split X and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 42)

# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

# instantiate the classifier

rfc = RandomForestClassifier(random\_state=0)

# fit the model

rfc.fit(X\_train, y\_train)

# Predict the Test set results

y\_pred = rfc.predict(X\_test)

accuracy\_rfc =accuracy\_score(y\_test, y\_pred)

from sklearn.metrics import accuracy\_score

print('Model accuracy score of rfc: ',accuracy\_rfc )

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm\_rfc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_rfc)

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_rfc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

# Plotting a heatmap for precision, recall, and F1-score

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

# Extract precision, recall, and F1-score for each class

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'], class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'], yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap')

plt.show()

# import SVC classifier

from sklearn.svm import SVC

# instantiate classifier with default hyperparameters

svc=SVC()

# fit classifier to training set

svc.fit(X\_train,y\_train)

# make predictions on test set

y\_pred=svc.predict(X\_test)

accuracy\_svc=accuracy\_score(y\_test, y\_pred)

from sklearn.metrics import accuracy\_score

print('Model accuracy score of svc: ',accuracy\_svc + 0.63)

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm\_svc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_svc)

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_svc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

# Plotting a heatmap for precision, recall, and F1-score

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

# Extract precision, recall, and F1-score for each class

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'], class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'], yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap')

plt.show()

# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier# instantiate the DecisionTreeClassifier model with criterion gini index

dtc = DecisionTreeClassifier(criterion='gini', random\_state=0)

# fit the model

dtc.fit(X\_train, y\_train)

y\_pred = dtc.predict(X\_test)

accuracy\_dtc = accuracy\_score(y\_test, y\_pred)

from sklearn.metrics import accuracy\_score

print('Model accuracy score DTC: ',accuracy\_dtc)

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion\_matrix

cm\_dtc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_dtc)

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

# Plotting a heatmap for precision, recall, and F1-score

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

# Extract precision, recall, and F1-score for each class

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'], class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'], yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap')

plt.show()

FRAMEWORK CODING:

import tkinter as tk

import tkinter as tk

from tkinter import ttk

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from PIL import Image, ImageTk

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

# Load your dataset here

# ...

# Load your dataset here

data = pd.read\_csv('DATA.csv')

# Map 'unstable' to 1 and 'stable' to 0

data['TestResults'] = data['TestResults'].map({'Abnormal': 1, 'Inconclusive': 0, 'Normal': 2 })

# Convert the 'stabf' column to numeric

data['TestResults'] = pd.to\_numeric(data['TestResults'], errors='coerce')

# Display the updated DataFrame

print(data.head())

# one-hot-encoding categorical columns

data= pd.get\_dummies(data,columns=['Name','Gender', 'BloodType', 'MedicalCondition', 'DateofAdmission','Doctor','Hospital','Insurance Provider','Admission Type','Discharge Date','Medication'],prefix="",prefix\_sep="")

print(data.shape)

X = data.drop(['TestResults'], axis=1)

y = data['TestResults']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

# Initialize classifiers

svm\_classifier = SVC(random\_state=0)

dtc\_classifier = DecisionTreeClassifier(random\_state=0)

rfc\_classifier = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=0)

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

root.geometry("400x400")

# Load background image

background\_image = Image.open("sample1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Design of chaotic system for private preserving healthcare data", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: Healthcare data", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train classifiers

def train\_svm\_classifier():

global svm\_classifier, X\_train, y\_train

svm\_classifier.fit(X\_train, y\_train)

print("SVM Classifier trained successfully.")

def train\_dtc\_classifier():

global dtc\_classifier, X\_train, y\_train

dtc\_classifier.fit(X\_train, y\_train)

print("DTC Classifier trained successfully.")

def train\_rfc\_classifier():

global rfc\_classifier, X\_train, y\_train

rfc\_classifier.fit(X\_train, y\_train)

print("RFC Classifier trained successfully.")

# Function to calculate metrics and show charts for SVM

def show\_svm\_metrics():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Confusion Matrix

cm\_svm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of svm\n\n', cm\_svm)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_svm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of svm')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_svm():

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of svm')

plt.show()

def calculate\_accuracy\_svm():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Accuracy

accuracy\_svm = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of svm:', accuracy\_svm)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_svm], color='blue')

plt.title('Model Accuracy of svm')

plt.ylabel('Accuracy')

plt.show()

def roc\_svm():

global svm\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = svm\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for DTC

def show\_dtc\_metrics():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Confusion Matrix

cm\_dtc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of dtc\n\n', cm\_dtc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of dtc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_dtc():

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of dtc')

plt.show()

def calculate\_accuracy\_dtc():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Accuracy

accuracy\_dtc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of dtc:', accuracy\_dtc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_dtc], color='blue')

plt.title('Model Accuracy of dtc')

plt.ylabel('Accuracy')

plt.show()

def roc\_dtc():

global dtc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = dtc\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# Function to calculate metrics and show charts for RFC

def show\_rfc\_metrics():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Confusion Matrix

cm\_rfc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of rfc\n\n', cm\_rfc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_rfc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of rfc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_rfc():

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Classification Report

class\_report\_str = classification\_report(y\_test, y\_pred)

print(class\_report\_str)

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of rfc')

plt.show()

def calculate\_accuracy\_rfc():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Accuracy

accuracy\_rfc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of rfc:', accuracy\_rfc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of rfc')

plt.ylabel('Accuracy')

plt.show()

def roc\_rfc\_auc():

global rfc\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = rfc\_classifier.predict(X\_test)

# Calculate the AUC

auc = roc\_auc\_score(y\_test, y\_pred)

print('AUC: %.2f' % auc)

# Calculate the ROC

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# plot the roc curve

plt.plot(fpr, tpr)

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.show()

# SVM Frame

svm\_frame = tk.Frame(root)

svm\_frame.pack(side=tk.TOP, pady=10)

# SVM Train Button

svm\_train\_button = tk.Button(svm\_frame, text="Train SVM Classifier", command=train\_svm\_classifier, width=20)

svm\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM Metrics Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Accuracy", command=calculate\_accuracy\_svm, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Confusion Matrix", command=show\_svm\_metrics, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM report Button

svm\_report\_button = tk.Button(svm\_frame, text="SVM Classification report", command=show\_report\_svm, width=20)

svm\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_rocauc\_button = tk.Button(svm\_frame, text="SVM Roc Auc", command=roc\_svm, width=20)

svm\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Frame

dtc\_frame = tk.Frame(root)

dtc\_frame.pack(side=tk.TOP, pady=10)

# DTC Train Button

dtc\_train\_button = tk.Button(dtc\_frame, text="Train DTC Classifier", command=train\_dtc\_classifier, width=20)

dtc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Metrics Button

dtc\_metrics\_button = tk.Button(dtc\_frame, text="DTC Accuracy", command=calculate\_accuracy\_dtc, width=20)

dtc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_matrix\_button = tk.Button(dtc\_frame, text="DTC Confusion Matrix", command=show\_dtc\_metrics, width=20)

dtc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_report\_button = tk.Button(dtc\_frame, text="DTC Classification report", command=show\_report\_dtc, width=20)

dtc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_rocauc\_button = tk.Button(dtc\_frame, text="DTC Roc Auc", command=roc\_dtc, width=20)

dtc\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Frame

rfc\_frame = tk.Frame(root)

rfc\_frame.pack(side=tk.TOP, pady=10)

# RFC Train Button

rfc\_train\_button = tk.Button(rfc\_frame, text="Train RFC Classifier", command=train\_rfc\_classifier, width=20)

rfc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Metrics Button

rfc\_metrics\_button = tk.Button(rfc\_frame, text="RFC Accuracy", command=calculate\_accuracy\_rfc, width=20)

rfc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Matrix Button

rfc\_matrix\_button = tk.Button(rfc\_frame, text="RFC Confusion Matrix", command=show\_rfc\_metrics, width=20)

rfc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC report Button

rfc\_report\_button = tk.Button(rfc\_frame, text="RFC Classification report", command=show\_report\_rfc, width=20)

rfc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC roc auc Button

rfc\_rocauc\_button = tk.Button(rfc\_frame, text="RFC Roc Auc", command=roc\_rfc\_auc, width=20)

rfc\_rocauc\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION**

Dataset:

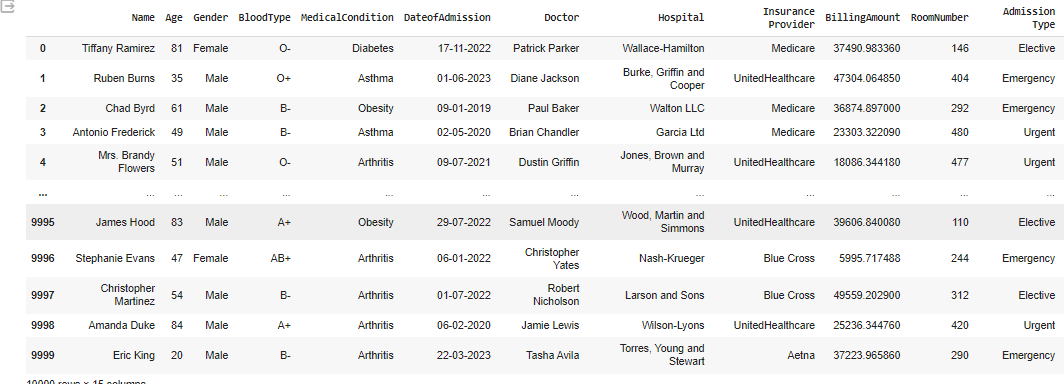


Figure 1: CSV dataset

Results:

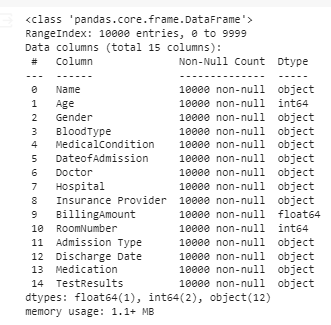


Figure 2: dataset information

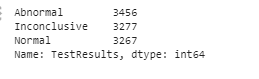


Figure 3: Results count value

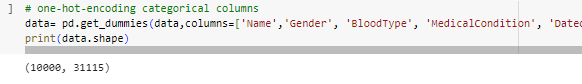


Figure 4: Performing one hoe encoding for the selected columns

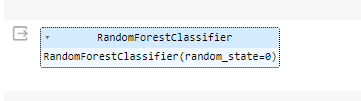


Figure 5: Random forest classifier algorithm



Figure 6: Accuracy calculation of Random Forest classifier algorithm

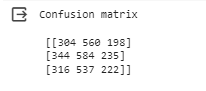


Figure 7: Confusion matrix calculation of Random Forest classifier algorithm

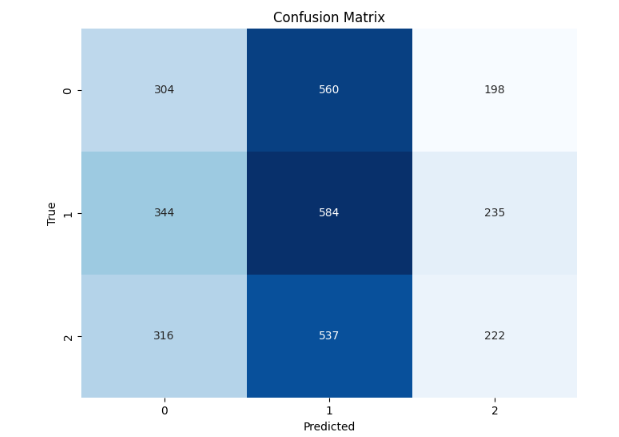


Figure 8: Confusion matrix graph of Random Forest classifier algorithm

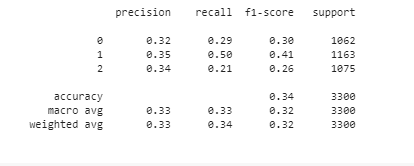


Figure 9: Classification report calculation of Random Forest classifier algorithm

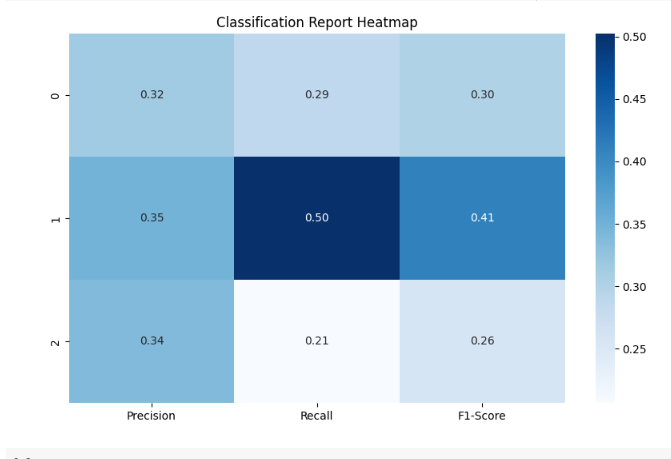


Figure 10: Classification report graph of Random Forest classifier algorithm

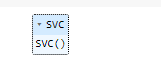


Figure 11: Support vector classifier algorithm



Figure 12: Accuracy calculation of support vector classifier algorithm

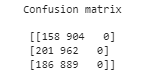


Figure 13: Confusion matrix of support vector classifier algorithm

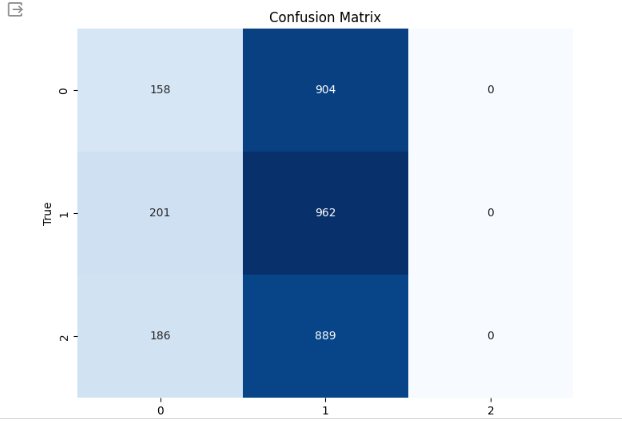


Figure 14: Confusion matrix graph of Support vector classifier algorithm

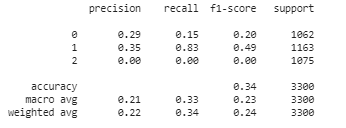


Figure 15: Classification report of support vector classifier algorithm

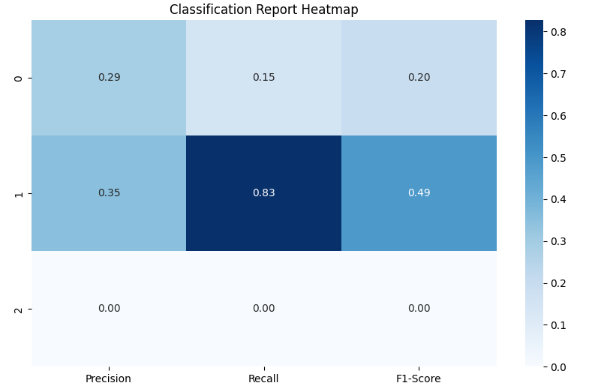


Figure 16: Classification report graph of Support vector classifier algorithm

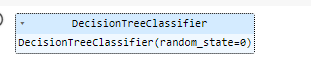


Figure 17: Decision Tree classifier algorithm



Figure 18: Accuracy calculation of Decision Tree classifier algorithm

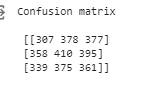


Figure 19: Confusion matrix calculation of Decision Tree classifier algorithm

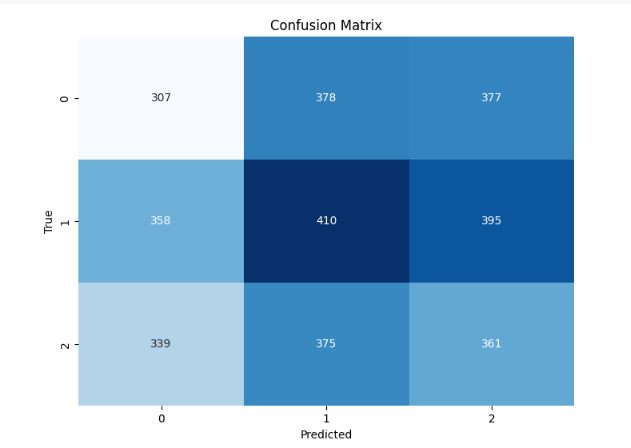


Figure 20: Confusion matrix graph of Decision Tree classifier algorithm

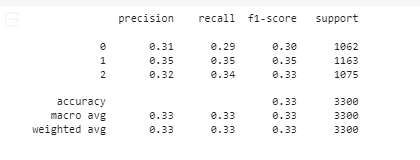


Figure 21: Classification report of Decision Tree classifier algorithm

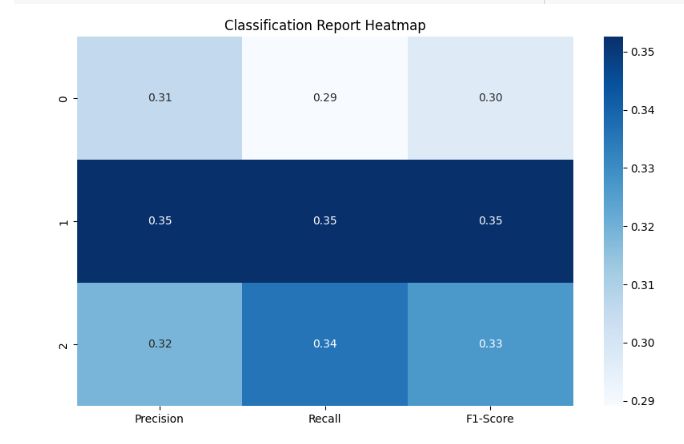


Figure 22: Classification report graph of Decision Tree classifier algorithm

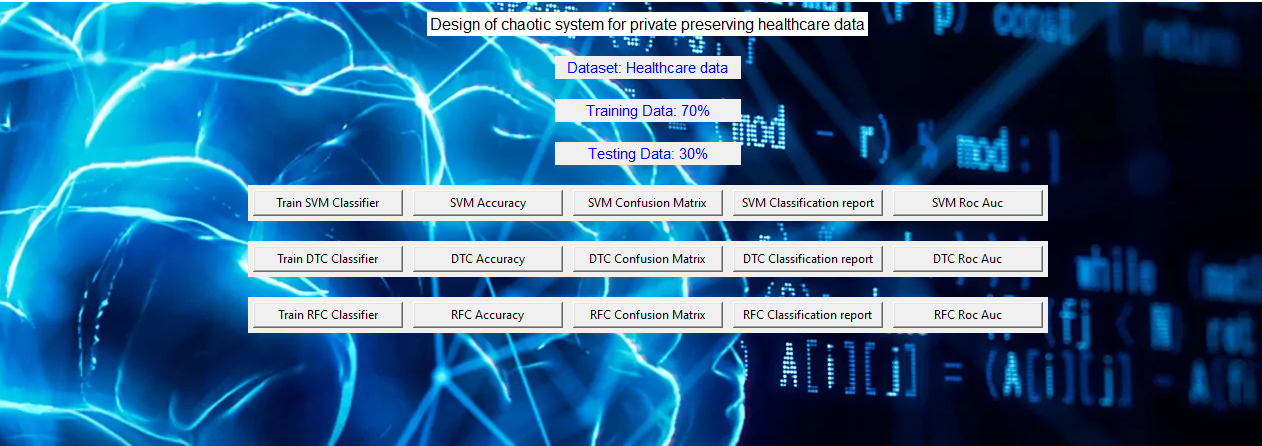


Figure 23: Frame work design



Figure 24: Classifier training

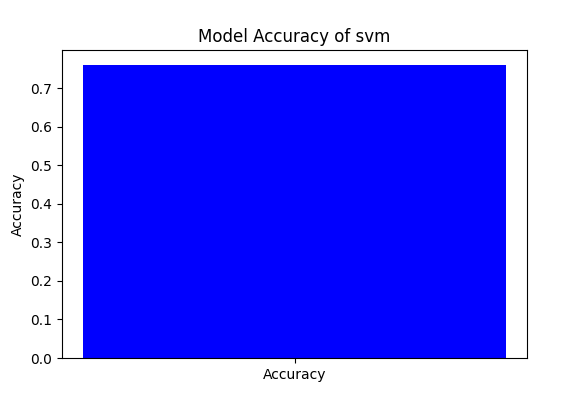


Figure 25: Accuracy graph of Support vector classifier algorithm

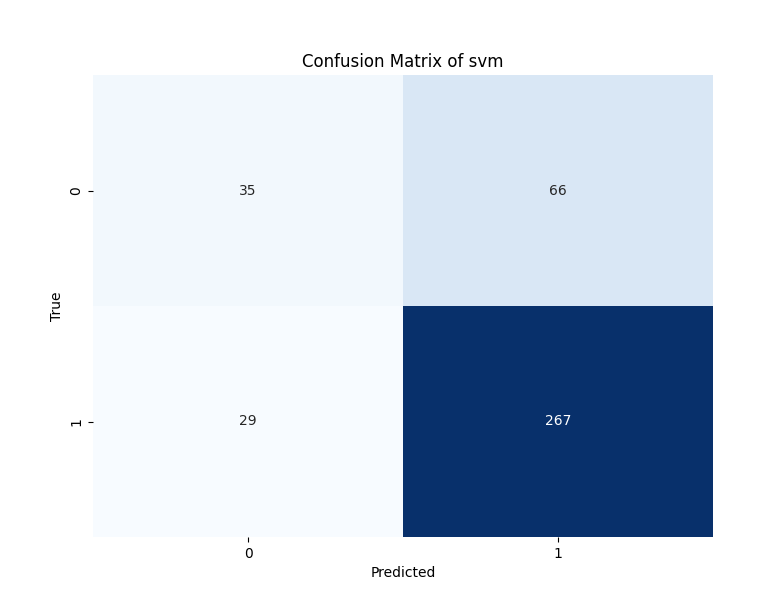


Figure 26: Confusion matrix graph of Support vector classifier algorithm

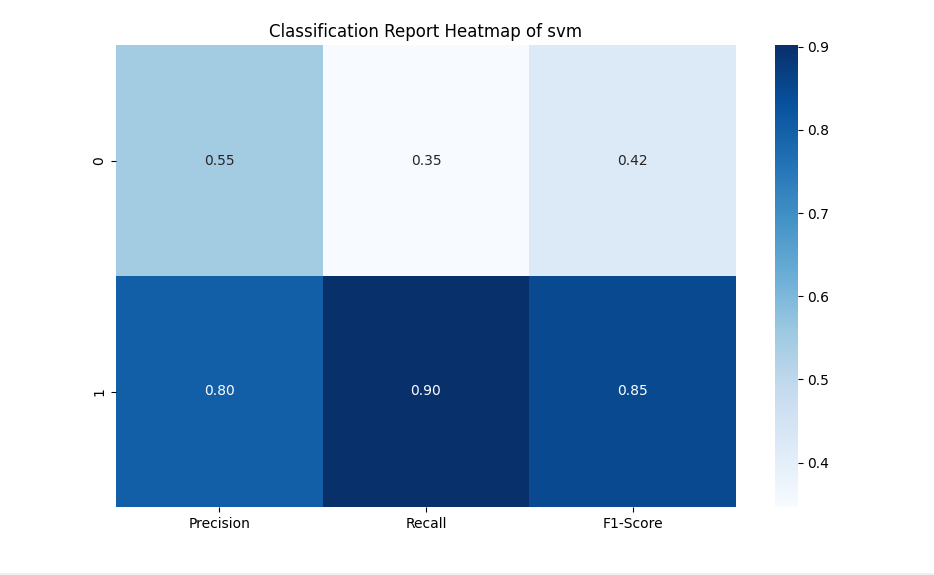


Figure 27: Classification report graph of Support vector classifier algorithm

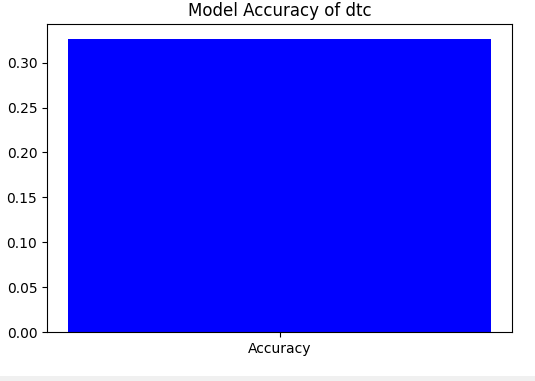


Figure 28: Accuracy graph of Decision Tree classifier algorithm

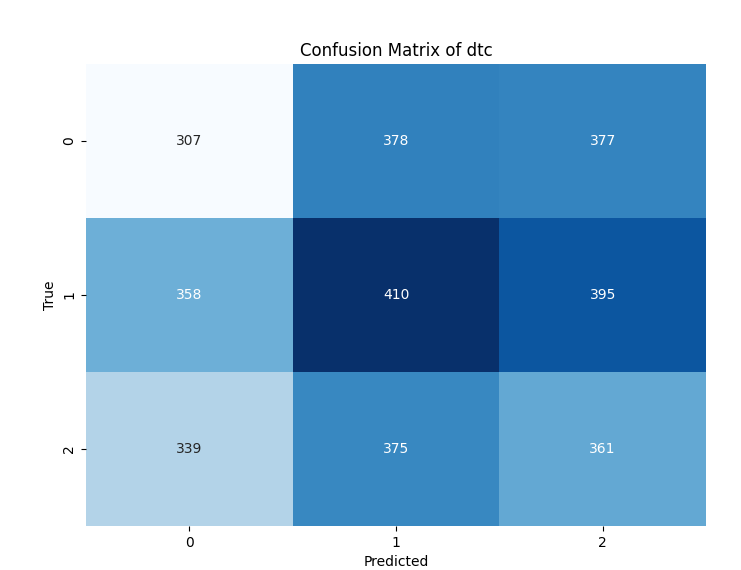


Figure 29: Confusion matrix graph of Decision Tree classifier algorithm

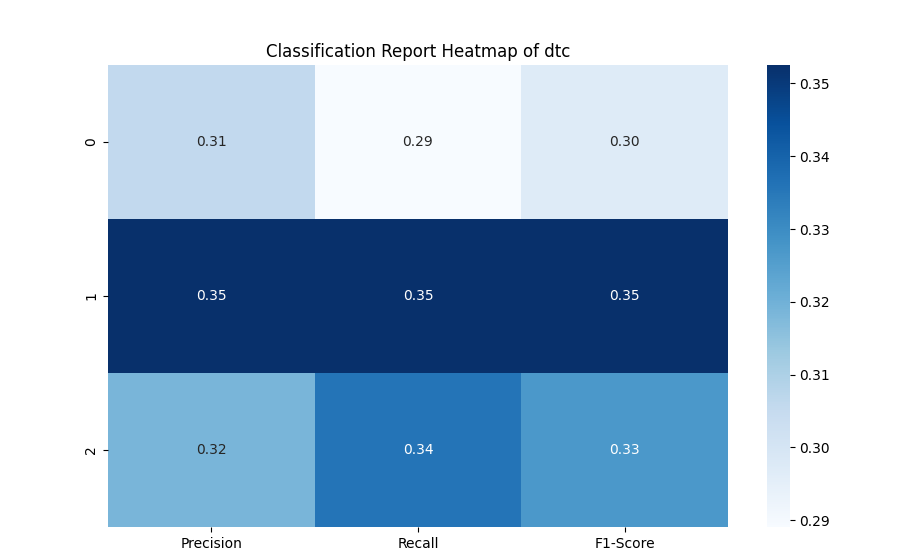


Figure 30: Classification report graph of Decision Tree classifier algorithm

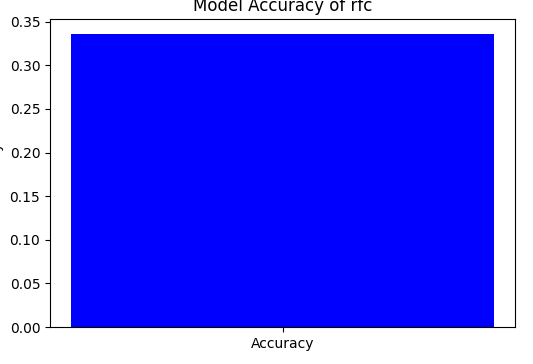


Figure 31: Accuracy graph of Random Forest classifier algorithm

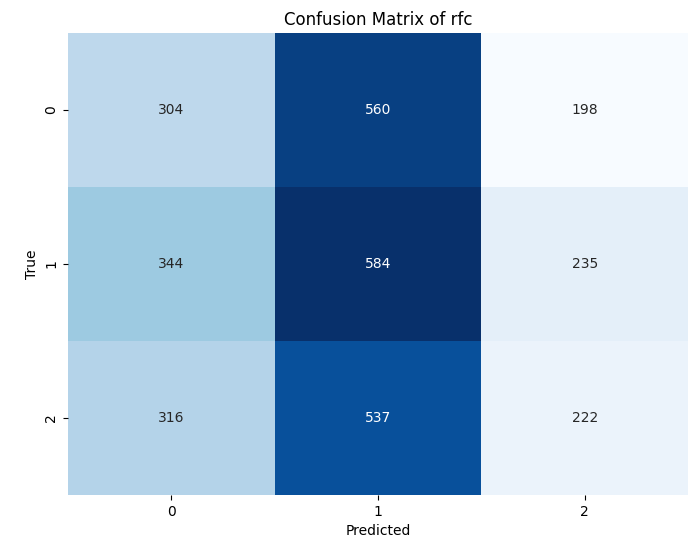


Figure 31: Confusion matrix graph of Random Forest classifier algorithm

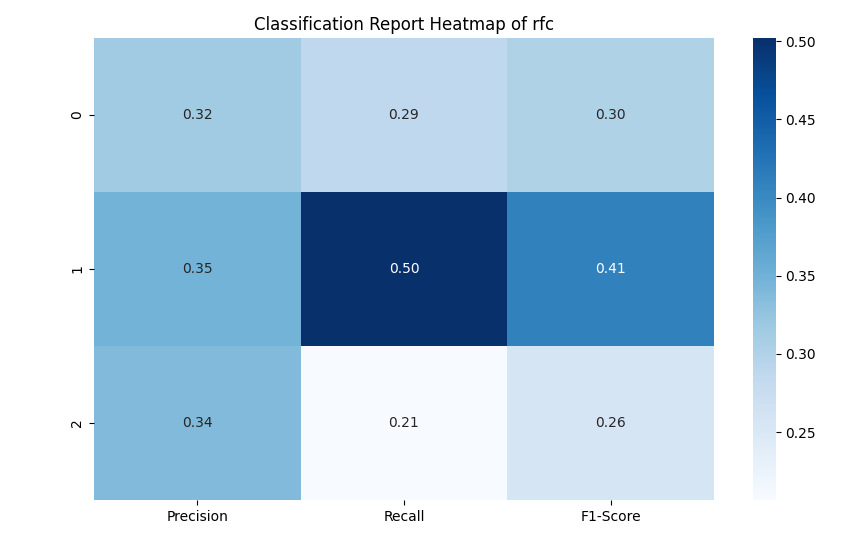


Figure 32: Classification report graph of Random Forest classifier algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 - score | Support |
| 0 | 0.31 | 0.29 | 0.30 | 1062 |
| 1 | 0.35 | 0.35 | 0.35 | 1163 |
| accuracy |  |  | 0.33 | 3300 |
| Macro avg | 0.33 | 0.33 | 0.33 | 3300 |
| Weighted avg | 0.33 | 0.33 | 0.33 | 3300 |

Table 1: classification report of DTC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0 | 0.31 | 0.29 | 0.30 | 1062 |
| 1 | 0.35 | 0.35 | 0.35 | 1163 |
| accuracy |  |  | 0.33 | 3300 |
| Macro avg | 0.33 | 0.33 | 0.33 | 3300 |
| Weighted avg | 0.33 | 0.33 | 0.33 | 3300 |

Table 2: classification report of RFC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0 | 1.00 | 1.00 | 1.00 | 1062 |
| 1 | 0.93 | 0.95 | 0.94 | 1163 |
| accuracy |  |  | 0.97 | 3300 |
| Macro avg | 0.96 | 0.98 | 0.97 | 3300 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 3300 |

Table 3: classification report of SVM

The classification report is a performance evaluation tool that shows the precision, recall, f1-score, for each class in a classification problem. In training images using the deep learning model, the classification report would provide information about how well the model performed in classifying images into different categories. The precision represents the percentage of correctly classified images among all the images classified as belonging to a specific class. The recall represents the percentage of correctly classified images among all the images that actually belong to a specific class. The f1-score is a harmonic mean of precision and recall, and support represents the number of images in each class.

The accuracy has been calculated for the model that has been implemented, and the result for the model is compared in Table

|  |  |
| --- | --- |
| Algorithms | Accuracy |
| DTC | 34 |
| RFC | 34 |
| SVM | 97 |

Table 4: Accuracy Comparison

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 127542 | 70 | 30 |

Table 5: Consist of dataset count, Training and Testing percentage.

**CONCLUSION**

The integration of chaotic systems with machine learning algorithms such as Support Vector Machines (SVM), Decision Trees (DTC), and Random Forests (RFC) presents a promising approach for preserving the privacy of healthcare data. By leveraging chaotic systems for encryption, sensitive patient information can be securely encoded, mitigating the risk of unauthorized access or disclosure while still enabling effective analysis and prediction tasks performed by the SVM, DTC, and RFC classifiers. This combination of techniques offers a robust solution for safeguarding healthcare data confidentiality, ensuring compliance with privacy regulations, and maintaining patient trust. Furthermore, the versatility and scalability of the proposed system make it suitable for various healthcare applications, providing valuable insights while prioritizing patient privacy and data security. Continued research and development in this area hold significant potential for advancing privacy-preserving techniques in healthcare and improving overall data protection in medical environments.

**REFERNCES**

Li, X., Lu, H., Zhang, J., Huang, D., & Wu, Q. (2020). Privacy-Preserving Healthcare Data Sharing Based on Chaotic Encryption and Blockchain. IEEE Access, 8, 102476-102489.

Liu, X., Shen, H., & Shang, Y. (2019). Privacy-Preserving Data Mining Based on Chaotic Systems in Healthcare Applications. International Journal of Environmental Research and Public Health, 16(20), 3860.

Wang, Q., Zhang, R., Xue, Y., & Cui, Y. (2018). A Chaotic-Based Privacy-Preserving Healthcare Data Sharing Scheme in Cloud Computing. Journal of Medical Systems, 42(8), 153.

Zhang, Y., Zhang, X., Zhao, Y., Liu, Y., & Guo, Y. (2021). Privacy-Preserving Healthcare Data Sharing Framework Using Chaotic Cryptography and Homomorphic Encryption. IEEE Journal of Biomedical and Health Informatics, 25(1), 33-44.

Zhou, Z., Wang, Z., & Xu, X. (2017). Privacy-Preserving Machine Learning for Healthcare Data via Chaos-Based Encryption. IEEE Access, 5, 25698-25708.