# **4. Program/Code**

## **4.1 Code for the Home App:**

import streamlit as st

from res.multiapp import MultiApp

from Apps import Hypertension\_App, Stroke\_App, Heart\_Disease, Diabetes, Breast\_Cancer, \

    Kidney\_App  # import your app modules here

from PIL import Image

import json

from res import Header as hd

from streamlit\_lottie import st\_lottie

from streamlit\_extras.switch\_page\_button import switch\_page

from st\_pages import show\_pages, Page

st.set\_page\_config(

    page\_title="TRUTH LABORATORIES",

    page\_icon=Image.open("images/medical-team.png"),

    layout="wide"

)

image = Image.open("images/Logo.png")

st.sidebar.image(image, use\_column\_width=True)

show\_pages(

    [

        Page("Home.py", "DASHBOARD", "🏠"),

        Page("pages/Dataset.py", "RECORDS", ":books:"),

        Page("pages/Diagonizer.py", "EXPERT OPINION", "🏣"),

        Page("pages/Contact.py", "KEEP IN TOUCH", "✉️"),

    ]

)

def load\_lottiefile(filepath: str):

    with open(filepath, "r") as f:

        return json.load(f)

lottie\_coding = load\_lottiefile("res/Logo\_Final.json")

app = MultiApp()

st.markdown(

    """

    <style>

    .markdown-section {

        margin-left: 10px;

    }

    </style>

    """,

    unsafe\_allow\_html=True

)

col1, col2 = st.columns([1, 1], gap="small")

with col1:

    st\_lottie(

        lottie\_coding,

        speed=1,

        reverse=False,

        loop=True,

        quality="medium",

        height=None,

        width=None,

        key=None,

    )

    col1.empty()

with col2:

    col2.empty()

    st.title("TRUTH LABORATORIES")

    st.markdown("""

    \*\*TRUTH LABORATORIES WELCOMES YOU ALL\*\*,

    Where every heartbeat matters, and every smile tells a story of healing. Step into a realm where expertise meets empathy,

    and where hope is the cornerstone of our practice. Explore the corridors of knowledge, where the latest advancements in medicine merge with timeless compassion. Join us on a journey of health and wellness, guided by dedication and driven by a commitment to excellence. Together, let's embark on a path towards brighter tomorrows, where each visit brings reassurance and each interaction fosters trust. Welcome to our medical community, where your well-being is our priority, and where healing begins with a warm embrace.

    \_The parameters could include\_ `age, gender, lifestyle habits, genetic factors, and existing health conditions` \_, among others.\_

    """)

    st.markdown("""Check-out our Data Records""")

    page\_switch = st.button("Check out Records")

    if page\_switch:

        switch\_page("Dataset")

hd.colored\_header(

    label="Select your disease",

    color\_name="violet-70",

)

# Add all your application here

app.add\_app("Breast Cancer Detector", Breast\_Cancer.app)

app.add\_app("Diabetes Detector", Diabetes.app)

app.add\_app("Heart Disease Detector", Heart\_Disease.app)

app.add\_app("Hypertension Detector", Hypertension\_App.app)

app.add\_app("Kidney Disease Detector", Kidney\_App.app)

app.add\_app("Stroke Detector", Stroke\_App.app)

# The main app

app.run()

## **4.2 Code for the Classifier Model of Breast Cancer Disease:**

import numpy as np

import streamlit as st

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

import pickle

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

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

url = "res/dataset/breast-cancer.csv"

bc = pd.read\_csv(url)

# Ordinal feature encoding

df = bc.copy()

# Separating X and y

X = df.drop('diagnosis', axis=1)

Y = df['diagnosis']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

"""\_\_\_\_\_\_\_\_Naive Bayes Algorithm\_\_\_\_\_\_\_\_"""

# Train the Naive Bayes classifier

nb\_classifier = GaussianNB(var\_smoothing=1e-9)

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

# Predict using the Naive Bayes classifier

nb\_predictions = nb\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Naive Bayes classifier

nb\_cm = confusion\_matrix(y\_test, nb\_predictions)

nb\_accuracy = accuracy\_score(y\_test, nb\_predictions)

nb\_classifier\_report = classification\_report(y\_test, nb\_predictions)

nb\_classifier\_report\_dict = classification\_report(y\_test, nb\_predictions, output\_dict=True)

def plt\_NB():

    def classifier\_report():

        report\_df = pd.DataFrame(nb\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Naive Bayes Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for Naive Bayes classifier

    plt.figure()

    plt.imshow(nb\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Naive Bayes')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = nb\_cm.max() / 2

    for i, j in np.ndindex(nb\_cm.shape):

        plt.text(j, i, format(nb\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if nb\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_NB():

        report\_df = pd.DataFrame(nb\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Naive Bayes Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Decision Tree\_\_\_\_\_\_\_\_"""

# Train the Decision Tree classifier

dt\_classifier = DecisionTreeClassifier(max\_depth=None)

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

# Predict using the Decision Tree classifier

dt\_predictions = dt\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Decision Tree classifier

dt\_cm = confusion\_matrix(y\_test, dt\_predictions)

dt\_accuracy = accuracy\_score(y\_test, dt\_predictions)

dt\_classifier\_report = classification\_report(y\_test, dt\_predictions)

dt\_classifier\_report\_dict = classification\_report(y\_test, dt\_predictions, output\_dict=True)

def plt\_DT():

    def classifier\_report():

        report\_df = pd.DataFrame(dt\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Decision Tree Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for Naive Bayes classifier

    plt.figure()

    plt.imshow(dt\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Decision Tree')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = dt\_cm.max() / 2

    for i, j in np.ndindex(dt\_cm.shape):

        plt.text(j, i, format(dt\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if dt\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_DT():

    report\_df = pd.DataFrame(dt\_classifier\_report\_dict).transpose()

    # Display the classification report as a table using st.write()

    st.write("Decision Tree Classifier Report")

    st.write(report\_df)

    st.write()

"""\_\_\_\_\_\_\_\_Logistic Regression Algorithm\_\_\_\_\_\_\_\_"""

# Train the Logistic Regression classifier

lr\_classifier = LogisticRegression(max\_iter=1000)

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

# Predict using the Logistic Regression classifier

lr\_predictions = lr\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Logistic Regression classifier

lr\_cm = confusion\_matrix(y\_test, lr\_predictions)

lr\_accuracy = accuracy\_score(y\_test, lr\_predictions)

lr\_classifier\_report = classification\_report(y\_test, lr\_predictions)

lr\_classifier\_report\_dict = classification\_report(y\_test, lr\_predictions, output\_dict=True)

def plt\_LR():

    def classifier\_report():

        report\_df = pd.DataFrame(lr\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Logistic Regression Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for classifier

    plt.figure()

    plt.imshow(lr\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Logistic Regression')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = lr\_cm.max() / 2

    for i, j in np.ndindex(lr\_cm.shape):

        plt.text(j, i, format(lr\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if lr\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_LR():

        report\_df = pd.DataFrame(lr\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Logistic Regression Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Random Forest Algorithm\_\_\_\_\_\_\_\_"""

# Train the Random Forest classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100)

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

# Predict using the Random Forest classifier

rf\_predictions = rf\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Random Forest classifier

rf\_cm = confusion\_matrix(y\_test, rf\_predictions)

rf\_accuracy = accuracy\_score(y\_test, rf\_predictions)

rf\_classifier\_report = classification\_report(y\_test, rf\_predictions)

rf\_classifier\_report\_dict = classification\_report(y\_test, rf\_predictions, output\_dict=True)

def plt\_RF():

    def classifier\_report():

        report\_df = pd.DataFrame(rf\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Random Forest Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for  classifier

    plt.figure()

    plt.imshow(rf\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Random Forest')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = rf\_cm.max() / 2

    for i, j in np.ndindex(rf\_cm.shape):

        plt.text(j, i, format(rf\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if rf\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_RF():

        report\_df = pd.DataFrame(rf\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Random Forest Classifier Report")

        st.write(report\_df)

        st.write()

# Compare the 5 models and select the best algorithm

models = {

    'Naive Bayes': nb\_classifier\_report,

    #'K-Nearest Neighbors (KNN)': knn\_classifier\_report,

    'Decision Tree': dt\_classifier\_report,

    'Logistic Regression': lr\_classifier\_report,

    'Random Forest': rf\_classifier\_report

}

best\_model = max(models, key=models.get)

# Saving the model

pickle.dump(nb\_classifier, open('res/pickle/breast-cancer\_disease\_classifier\_NB.pkl', 'wb'))

#pickle.dump(knn\_classifier, open('res/pickle/breast-cancer\_disease\_classifier\_KNN.pkl', 'wb'))

pickle.dump(dt\_classifier, open('res/pickle/breast-cancer\_disease\_classifier\_DT.pkl', 'wb'))

pickle.dump(lr\_classifier, open('res/pickle/breast-cancer\_disease\_classifier\_LR.pkl', 'wb'))

pickle.dump(rf\_classifier, open('res/pickle/breast-cancer\_disease\_classifier\_RF.pkl', 'wb'))

## **4.3 Code for the Breast Cancer App for the Streamlit:**

import time

import streamlit as st

import pandas as pd

import Classifier\_Models.Classifier\_model\_builder\_breast\_cancer as cmb

import pickle

import numpy as np

from streamlit\_toggle import st\_toggle\_switch

import json

from streamlit\_lottie import st\_lottie

def app():

    def load\_lottiefile(filepath: str):

        with open(filepath, "r") as f:

            return json.load(f)

    lottie\_coding = load\_lottiefile("res/Yoga\_Feet.json")

    st.title("Breast Cancer Predictor")

    st.info("This Model predicts whether a Female is suffering from Breast Cancer or not")

    st.markdown("""

    \*\*Note\*\* - :red[THIS PREDICTION MODEL IS ONLY FOR FEMALES.]

    """)

    st.sidebar.header('Report Uploader')

    uploaded\_file = st.sidebar.file\_uploader("Upload your parameters of your Report through a CSV File", type=["csv"])

    if uploaded\_file is not None:

        input\_df = pd.read\_csv(uploaded\_file)

    else:

        def patient\_details():

            radius\_mean = st.sidebar.number\_input('Radius of Lobes', 6.98, 28.10, step=0.01)

            texture\_mean = st.sidebar.number\_input('Mean of Surface Texture', 9.71, 39.30, step=0.01)

            perimeter\_mean = st.sidebar.number\_input('Outer Perimeter of Lobes', 43.8, 189.0, step=0.1)

            area\_mean = st.sidebar.number\_input('Mean Area of Lobes', 144, 2510)

            smoothness\_mean = st.sidebar.number\_input('Mean of Smoothness Levels', 0.05, 0.16, step=0.01)

            compactness\_mean = st.sidebar.number\_input('Mean of Compactness', 0.02, 0.35, step=0.01)

            concavity\_mean = st.sidebar.number\_input('Mean of Concavity', 0.00, 0.43, step=0.01)

            concave\_points\_mean = st.sidebar.number\_input('Mean of Cocave Points', 0.00, 0.20, step=0.01)

            symmetry\_mean = st.sidebar.number\_input('Mean of Symmetry', 0.11, 0.30, step=0.01)

            fractal\_dimension\_mean = st.sidebar.number\_input('Mean of Fractal Dimension', 0.05, 0.10, step=0.01)

            radius\_se = st.sidebar.number\_input('SE of Radius', 0.11, 2.87, step=0.01)

            texture\_se = st.sidebar.number\_input('SE of Texture', 0.36, 4.88, step=0.01)

            perimeter\_se = st.sidebar.number\_input('Perimeter of SE', 0.76, 22.00, step=0.01)

            area\_se = st.sidebar.number\_input('Area of SE', 6.8, 542.0, step=0.01)

            smoothness\_se = st.sidebar.number\_input('SE of Smoothness', 0.00, 0.03, step=0.01)

            compactness\_se = st.sidebar.number\_input('SE of compactness', 0.00, 0.14, step=0.01)

            concavity\_se = st.sidebar.number\_input('SE of concavity', 0.00, 0.40, step=0.01)

            concave\_points\_se = st.sidebar.number\_input('SE of concave points', 0.00, 0.05, step=0.01)

            symmetry\_se = st.sidebar.number\_input('SE of symmetry', 0.01, 0.08, step=0.01)

            fractal\_dimension\_se = st.sidebar.number\_input('SE of Fractal Dimension', 0.00, 0.03, step=0.01)

            radius\_worst = st.sidebar.number\_input('Worst Radius', 7.93, 36.00, step=0.01)

            texture\_worst = st.sidebar.number\_input('Worst Texture', 12.0, 49.5, step=0.1)

            perimeter\_worst = st.sidebar.number\_input('Worst Permimeter', 50.40, 251.20, step=0.01)

            area\_worst = st.sidebar.number\_input('Worst Area', 185.20, 4250.00, step=0.01)

            smoothness\_worst = st.sidebar.number\_input('Worst Smoothness', 0.07, 0.22, step=0.01)

            compactness\_worst = st.sidebar.number\_input('Worst Compactness', 0.03, 1.06, step=0.01)

            concavity\_worst = st.sidebar.number\_input('Worst Concavity', 0.00, 1.25, step=0.01)

            concave\_points\_worst= st.sidebar.number\_input('Worst Concave Points', 0.00, 0.29, step=0.01)

            symmetry\_worst = st.sidebar.number\_input('Worst Symmetry', 0.16, 0.66, step=0.01)

            fractal\_dimension\_worst = st.sidebar.number\_input('Worst Fractal Dimension', 0.06, 0.21, step=0.01)

            data = {'radius\_mean': radius\_mean,

                    'texture\_mean': texture\_mean,

                    'perimeter\_mean': perimeter\_mean,

                    'area\_mean': area\_mean,

                    'smoothness\_mean': smoothness\_mean,

                    'compactness\_mean': compactness\_mean,

                    'concavity\_mean': concavity\_mean,

                    'concave points\_mean': concave\_points\_mean,

                    'symmetry\_mean': symmetry\_mean,

                    'fractal\_dimension\_mean': fractal\_dimension\_mean,

                    'radius\_se': radius\_se,

                    'texture\_se': texture\_se,

                    'perimeter\_se': perimeter\_se,

                    'area\_se': area\_se,

                    'smoothness\_se': smoothness\_se,

                    'compactness\_se': compactness\_se,

                    'concavity\_se': concavity\_se,

                    'concave points\_se': concave\_points\_se,

                    'symmetry\_se': symmetry\_se,

                    'fractal\_dimension\_se': fractal\_dimension\_se,

                    'radius\_worst': radius\_worst,

                    'texture\_worst': texture\_worst,

                    'perimeter\_worst': perimeter\_worst,

                    'area\_worst': area\_worst,

                    'smoothness\_worst': smoothness\_worst,

                    'compactness\_worst': compactness\_worst,

                    'concavity\_worst': concavity\_worst,

                    'concave points\_worst': concave\_points\_worst,

                    'symmetry\_worst': symmetry\_worst,

                    'fractal\_dimension\_worst': fractal\_dimension\_worst, }

            features = pd.DataFrame(data, index=[0])

            return features

        input\_df = patient\_details()

    heart = cmb.X

    df = pd.concat([input\_df, heart], axis=0)

    df = df[:1]  # Selects only the first row (the user input data)

    df.loc[:, ~df.columns.duplicated()]

    if uploaded\_file is not None:

        st.write(df)

    else:

        st.write('Waiting for the Report to be Uploaded... Currently displaying the Parameters given manually')

        df = df.loc[:, ~df.columns.duplicated()]

        st.write(df)

    # Load the classification models

    load\_clf\_NB = pickle.load(open('res/pickle/breast-cancer\_disease\_classifier\_NB.pkl', 'rb'))

    #load\_clf\_KNN = pickle.load(open('res/pickle/breast-cancer\_disease\_classifier\_KNN.pkl', 'rb'))

    load\_clf\_DT = pickle.load(open('res/pickle/breast-cancer\_disease\_classifier\_DT.pkl', 'rb'))

    load\_clf\_LR = pickle.load(open('res/pickle/breast-cancer\_disease\_classifier\_LR.pkl', 'rb'))

    load\_clf\_RF = pickle.load(open('res/pickle/breast-cancer\_disease\_classifier\_RF.pkl', 'rb'))

    # Apply models to make predictions

    prediction\_NB = load\_clf\_NB.predict(df)

    prediction\_proba\_NB = load\_clf\_NB.predict\_proba(df)

    #prediction\_KNN = load\_clf\_KNN.predict(df)

    #prediction\_proba\_KNN = load\_clf\_KNN.predict\_proba(df)

    prediction\_DT = load\_clf\_DT.predict(df)

    prediction\_proba\_DT = load\_clf\_DT.predict\_proba(df)

    prediction\_LR = load\_clf\_LR.predict(df)

    prediction\_proba\_LR = load\_clf\_LR.predict\_proba(df)

    prediction\_RF = load\_clf\_RF.predict(df)

    prediction\_proba\_RF = load\_clf\_RF.predict\_proba(df)

    def NB():

        st.subheader('Prediction of Naïve Bayes Classifier')

        NB\_prediction = np.array([0, 1])

        if NB\_prediction[prediction\_NB] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorry!! You are suffering from Breast Cancer 😰</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### Opps!! You got a Malignant Tumor!! Consult our Expert Opinion Now 😣

                        """)

        else:

            st.write("<p style='font-size:20px;color: green'><b>Hopefully you are safe!! You got a Benign Tumors.😊</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `A benign tumor is a non-cancerous growth of cells that does not invade nearby tissues or spread to other parts of the body `

                        """)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Naïve Bayes Classifier')

            st.write(prediction\_proba\_NB)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_NB(

    def DT():

        st.subheader('Prediction of Decision Tree Classifier')

        DT\_prediction = np.array([0, 1])

        if DT\_prediction[prediction\_DT] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorry!! You are suffering from Breast Cancer 😰</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `Opps!! You got a Malignant Tumor!! Consult our Expert Opinion Now 😣`

                        """)

        else:

            st.write("<p style='font-size:20px;color: green'><b>Hopefully you are safe!! You got a Benign Tumors.😊</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `A benign tumor is a non-cancerous growth of cells that does not invade nearby tissues or spread to other parts of the body`

                        """)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Decision Tree Classifier')

            st.write(prediction\_proba\_DT)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_DT()

    def LR():

        st.subheader('Prediction of Logistic Regression')

        LR\_prediction = np.array([0, 1])

        if LR\_prediction[prediction\_LR] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorry!! You are suffering from Breast Cancer 😰</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `Opps!! You got a Malignant Tumor!! Consult our Expert Opinion Now 😣`

                        """)

        else:

            st.write("<p style='font-size:20px;color: green'><b>Hopefully you are safe!! You got a Benign Tumors.😊</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `A benign tumor is a non-cancerous growth of cells that does not invade nearby tissues or spread to other parts of the body`

                        """)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Logistic Regression')

            st.write(prediction\_proba\_LR)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_LR()

    def RF():

        st.subheader('Prediction by Random Forest Algorithm')

        RF\_prediction = np.array([0, 1])

        if RF\_prediction[prediction\_RF] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorry!! You are suffering from Breast Cancer 😰</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `Opps!! You got a Malignant Tumor!! Consult our Expert Opinion Now 😣`

                        """)

        else:

            st.write("<p style='font-size:20px;color: green'><b>Hopefully you are safe!! You got a Benign Tumors.😊</b></p>",

                     unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `A benign tumor is a non-cancerous growth of cells that does not invade nearby tissues or spread to other parts of the body`

                        """)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Random Forest Classifier')

            st.write(prediction\_proba\_RF)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_RF()

    def predict\_best\_algorithm():

        if cmb.best\_model == 'Naive Bayes':

            NB()

        # elif cmb.best\_model == 'K-Nearest Neighbors (KNN)':

        #     KNN()

        elif cmb.best\_model == 'Decision Tree':

            DT()

        elif cmb.best\_model == 'Logistic Regression':

            LR()

        elif cmb.best\_model == 'Random Forest':

            RF()

        else:

            st.write("<p style='font-size:20px;color: green'><b>Hopefully you are safe!! You got a Benign Tumors.😊</b></p>",

                    unsafe\_allow\_html=True)

            st.markdown("""

                        ##### `A benign tumor is a non-cancerous growth of cells that does not invade nearby tissues or spread to other parts of the body`

                        """)

    st.markdown("👈 Provide your input data in the sidebar")

    # Displays the user input features

    with st.expander("Prediction Results", expanded=False):

        # Display the input dataframe

        st.write("Your input values are shown below:")

        st.dataframe(input\_df)

        # Call the predict\_best\_algorithm() function

        st.text('Showing you the Best Report Generated by our Service', help='This Report shows Approximate Prediction')

        predict\_best\_algorithm()

        # Tips, Diagnosis, Treatment, and Recommendations.

        st.subheader("Opinions provided by Our Consultancy on Breast Cancer 👨‍⚕️ ")

        tab1, tab2, tab3 = st.tabs(["Advices", "Diagnosis", "Therapy"])

        with tab1:

            st.subheader("Advices of Our Consultancy:")

            prevention\_tips = [

                "1. Stay physically active by engaging in regular exercise, which helps to maintain a healthy weight and reduces the risk of breast cancer.",

                "2. Maintain a healthy weight, especially after menopause, as being overweight increases the risk of breast cancer.",

                "3. Follow a healthy diet rich in fruits, vegetables, whole grains, and lean proteins, and limit the intake of processed and red meats.",

                "4. Limit or avoid alcohol consumption, as excessive alcohol intake is linked to an increased risk of breast cancer.",

                "5. Minimize hormone replacement therapy (HRT) if possible, as it may increase the risk of breast cancer.",

                "6. Breastfeed your children, as breastfeeding may lower the risk of breast cancer.",

                "7. Conduct regular self-exams and mammograms as recommended by healthcare professionals for early detection and treatment of any abnormalities."

            ]

            for tip in prevention\_tips:

                st.write(f"- {tip}")

        with tab2:

            st.subheader("Diagnosis Methods provided by Our Service:")

            c1, c2, c3 = st.columns([1, 1, 1], gap="small")

            with c1:

                diagnosis\_methods = [

                    "1. \*\*Clinical Breast Examination (CBE)\*\* ",

                    "2. \*\*Mammography\*\* ",

                    "3. \*\*Breast Ultrasound\*\* ",

                    "4. \*\*Breast MRI\*\* ",

                    "5. \*\*Biopsy\*\* "

                ]

                for method in diagnosis\_methods:

                    st.write(f"- {method}")

            with c3:

                st\_lottie(

                    lottie\_coding,

                    speed=1,

                    reverse=False,

                    loop=True,

                    quality="medium",

                    height=None,

                    width=None,

                    key=None,

                )

        with tab3:

            st.subheader("Treatment Category Available Here:")

            treatment\_options = [

                "1. \*\*Surgery:\*\* Lumpectomy (removal of the tumor and surrounding tissue) or mastectomy (removal of the breast tissue)",

                "2. \*\*Radiation Therapy:\*\* Used after surgery to destroy any remaining cancer cells and reduce the risk of recurrence",

                "3. \*\*Chemotherapy:\*\* Administered before or after surgery to shrink tumors, kill cancer cells, and prevent metastasis.",

                "4. \*\*Hormone Therapy:\*\* Blocks hormones that fuel certain types of breast cancer, often used in hormone receptor-positive breast cancers",

                "5. \*\*Targeted Therapy:\*\* Targets specific molecules involved in cancer cell growth and survival, such as HER2-targeted drugs for HER2-positive breast cancer",

                "6. \*\*Immunotherapy:\*\* Boosts the body's immune system to fight cancer cells, although its use in breast cancer is still evolving "

            ]

            for option in treatment\_options:

                st.write(f"- {option}")

            st.subheader("Note:")

            st.write("Treatment decisions are based on factors like cancer stage, grade, overall health, and menopause status.")

            st.write("You can discuss your treatment options with your healthcare team and ask questions at any time.")

    with st.expander("Comparison Study", expanded=False):

        col1, col2 = st.columns(2)

        with col1:

            st.header("Naïve Bayes Classifier")

            cmb.CR\_NB()

        with col2:

            st.header("Decision Tree Classifier")

            cmb.CR\_DT()

        col3, col4 = st.columns(2)

        with col3:

            st.header("Logistic Regression Algorithm")

            cmb.CR\_LR()

        with col4:

            st.header("Random Forest Algorithm")

            cmb.CR\_RF()

    # Create a multiselect for all the plot options

    selected\_plots = st.multiselect("You can see all the Detailed Reports Here 👇",

                                    ["Naïve Bayes", "Decision Tree", "Logistic Regression", "Random Forest"], default=[],key="ms\_B")

    if "ms\_B" not in st.session\_state:

        st.session\_state.selected\_plots = []

    # Check the selected plots and call the corresponding plot functions

    if selected\_plots:

        col1, col2 = st.columns(2)

        with col1:

            st.text('Understanding the Report',

                    help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

        with col2:

            st.text('Understanding Confusion Matrix',

                    help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

    placeholder = st.empty()

    # Check the selected plots and call the corresponding plot functions

    if "Naïve Bayes" in selected\_plots:

        with st.spinner("Generating Report of Naïve Bayes Classifier ...."):

            cmb.plt\_NB()

            time.sleep(1)

    # if "K-Nearest Neighbors" in selected\_plots:

    #     with st.spinner("Generating KNN...."):

    #         cmb.plt\_KNN()

    #         time.sleep(1)

    if "Decision Tree" in selected\_plots:

        with st.spinner("Generating Report of Decision Tree Classifier...."):

            cmb.plt\_DT()

            time.sleep(1)

    if "Logistic Regression" in selected\_plots:

        with st.spinner("Generating Report of Logistic Regression...."):

            cmb.plt\_LR()

            time.sleep(1)

    if "Random Forest" in selected\_plots:

        with st.spinner("Generating Report of Random Forest Algorithm...."):

            cmb.plt\_RF()

            time.sleep(1)

    # Remove the placeholder to display the list options

    placeholder.empty()

## **4.4 Code for the Classifier Model of the PIMA Indian Diabetes:**

import numpy as np

import streamlit as st

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

import pickle

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

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

url = "res/dataset/diabetes.csv"

dia = pd.read\_csv(url)

# Ordinal feature encoding

df = dia.copy()

# Separating X and y

X = df.drop('Outcome', axis=1)

Y = df['Outcome']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

"""\_\_\_\_\_\_\_\_Naive Bayes Algorithm\_\_\_\_\_\_\_\_"""

# Train the Naive Bayes classifier

nb\_classifier = GaussianNB(var\_smoothing=1e-9)

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

# Predict using the Naive Bayes classifier

nb\_predictions = nb\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Naive Bayes classifier

nb\_cm = confusion\_matrix(y\_test, nb\_predictions)

nb\_accuracy = accuracy\_score(y\_test, nb\_predictions)

nb\_classifier\_report = classification\_report(y\_test, nb\_predictions)

nb\_classifier\_report\_dict = classification\_report(y\_test, nb\_predictions, output\_dict=True)

def plt\_NB():

    def classifier\_report():

        report\_df = pd.DataFrame(nb\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Naive Bayes Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for Naive Bayes classifier

    plt.figure()

    plt.imshow(nb\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Naive Bayes')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = nb\_cm.max() / 2

    for i, j in np.ndindex(nb\_cm.shape):

        plt.text(j, i, format(nb\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if nb\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_NB():

        report\_df = pd.DataFrame(nb\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Naive Bayes Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Decision Tree\_\_\_\_\_\_\_\_"""

# Train the Decision Tree classifier

dt\_classifier = DecisionTreeClassifier(max\_depth=None)

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

# Predict using the Decision Tree classifier

dt\_predictions = dt\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Decision Tree classifier

dt\_cm = confusion\_matrix(y\_test, dt\_predictions)

dt\_accuracy = accuracy\_score(y\_test, dt\_predictions)

dt\_classifier\_report = classification\_report(y\_test, dt\_predictions)

dt\_classifier\_report\_dict = classification\_report(y\_test, dt\_predictions, output\_dict=True)

def plt\_DT():

    def classifier\_report():

        report\_df = pd.DataFrame(dt\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Decision Tree Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for Naive Bayes classifier

    plt.figure()

    plt.imshow(dt\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Decision Tree')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = dt\_cm.max() / 2

    for i, j in np.ndindex(dt\_cm.shape):

        plt.text(j, i, format(dt\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if dt\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_DT():

        report\_df = pd.DataFrame(dt\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Decision Tree Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Logistic Regression Algorithm\_\_\_\_\_\_\_\_"""

# Train the Logistic Regression classifier

lr\_classifier = LogisticRegression(max\_iter=1000)

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

# Predict using the Logistic Regression classifier

lr\_predictions = lr\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Logistic Regression classifier

lr\_cm = confusion\_matrix(y\_test, lr\_predictions)

lr\_accuracy = accuracy\_score(y\_test, lr\_predictions)

lr\_classifier\_report = classification\_report(y\_test, lr\_predictions)

lr\_classifier\_report\_dict = classification\_report(y\_test, lr\_predictions, output\_dict=True)

def plt\_LR():

    def classifier\_report():

        report\_df = pd.DataFrame(lr\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Logistic Regression Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for classifier

    plt.figure()

    plt.imshow(lr\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Logistic Regression')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = lr\_cm.max() / 2

    for i, j in np.ndindex(lr\_cm.shape):

        plt.text(j, i, format(lr\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if lr\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_LR():

        report\_df = pd.DataFrame(lr\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Logistic Regression Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Random Forest Algorithm\_\_\_\_\_\_\_\_"""

# Train the Random Forest classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100)

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

# Predict using the Random Forest classifier

rf\_predictions = rf\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Random Forest classifier

rf\_cm = confusion\_matrix(y\_test, rf\_predictions)

rf\_accuracy = accuracy\_score(y\_test, rf\_predictions)

rf\_classifier\_report = classification\_report(y\_test, rf\_predictions)

rf\_classifier\_report\_dict = classification\_report(y\_test, rf\_predictions, output\_dict=True)

def plt\_RF():

    def classifier\_report():

        report\_df = pd.DataFrame(rf\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Random Forest Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for  classifier

    plt.figure()

    plt.imshow(rf\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Random Forest')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = rf\_cm.max() / 2

    for i, j in np.ndindex(rf\_cm.shape):

        plt.text(j, i, format(rf\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if rf\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_RF():

        report\_df = pd.DataFrame(rf\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Random Forest Classifier Report")

        st.write(report\_df)

        st.write()

# Compare the 5 models and select the best algorithm

models = {

    'Naive Bayes': nb\_classifier\_report,

    #'K-Nearest Neighbors (KNN)': knn\_classifier\_report,

    'Decision Tree': dt\_classifier\_report,

    'Logistic Regression': lr\_classifier\_report,

    'Random Forest': rf\_classifier\_report

}

best\_model = max(models, key=models.get)

# Saving the model

pickle.dump(nb\_classifier, open('res/pickle/diabetes\_disease\_classifier\_NB.pkl', 'wb'))

#pickle.dump(knn\_classifier, open('res/pickle/diabetes\_disease\_classifier\_KNN.pkl', 'wb'))

pickle.dump(dt\_classifier, open('res/pickle/diabetes\_disease\_classifier\_DT.pkl', 'wb'))

pickle.dump(lr\_classifier, open('res/pickle/diabetes\_disease\_classifier\_LR.pkl', 'wb'))

pickle.dump(rf\_classifier, open('res/pickle/diabetes\_disease\_classifier\_RF.pkl', 'wb'))

## **4.5 Code for the PIMA Indian Diabetes App for Streamlit:**

import time

import streamlit as st

import pandas as pd

import Classifier\_Models.Classifier\_model\_builder\_diabetes as cmb

import pickle

import numpy as np

from streamlit\_toggle import st\_toggle\_switch

import json

from streamlit\_lottie import st\_lottie

def app():

    def load\_lottiefile(filepath: str):

        with open(filepath, "r") as f:

            return json.load(f)

    lottie\_coding = load\_lottiefile("res/Yoga\_Feet.json")

    st.title("PIMA Indian Diabetes Predictor")

    st.info("This Model predicts whether a Female is suffering from PIMA Indian Diabetes or not")

    st.markdown("""

    \*\*Note\*\* - :red[THIS PREDICTION MODEL IS ONLY FOR FEMALES.]

    """)

    st.sidebar.header('Report Uploader')

    uploaded\_file = st.sidebar.file\_uploader("Upload your parameters of your Report through a CSV File", type=["csv"])

    if uploaded\_file is not None:

        input\_df = pd.read\_csv(uploaded\_file)

    else:

        def patient\_details():

            Pregnancies = st.sidebar.number\_input('Pregnancies', 0, 17)

            Glucose = st.sidebar.number\_input('Glucose', 0, 199)

            BloodPressure = st.sidebar.number\_input('BloodPressure', 0, 122)

            SkinThickness = st.sidebar.number\_input('SkinThickness', 0, 99)

            Insulin = st.sidebar.number\_input('Insulin', 0, 846)

            BMI = st.sidebar.number\_input('BMI', 0.0, 67.1, step=0.1)

            DiabetesPedigreeFunction = st.sidebar.number\_input('DiabetesPedigreeFunction', 0.08, 2.42, step=0.01)

            Age = st.sidebar.number\_input('Age', 21, 81)

            data = {'Pregnancies': Pregnancies,

                    'Glucose': Glucose,

                    'BloodPressure': BloodPressure,

                    'SkinThickness': SkinThickness,

                    'Insulin': Insulin,

                    'BMI': BMI,

                    'DiabetesPedigreeFunction': DiabetesPedigreeFunction,

                    'Age': Age, }

            features = pd.DataFrame(data, index=[0])

            return features

        input\_df = patient\_details()

    heart = cmb.X

    df = pd.concat([input\_df, heart], axis=0)

    df = df[:1]  # Selects only the first row (the user input data)

    df.loc[:, ~df.columns.duplicated()]

    if uploaded\_file is not None:

        st.write(df)

    else:

        st.write('Waiting for the Report to be Uploaded... Currently displaying the Parameters given manually')

        df = df.loc[:, ~df.columns.duplicated()]

        st.write(df)

    # Load the classification models

    load\_clf\_NB = pickle.load(open('res/pickle/diabetes\_disease\_classifier\_NB.pkl', 'rb'))

    #load\_clf\_KNN = pickle.load(open('res/pickle/diabetes\_disease\_classifier\_KNN.pkl', 'rb'))

    load\_clf\_DT = pickle.load(open('res/pickle/diabetes\_disease\_classifier\_DT.pkl', 'rb'))

    load\_clf\_LR = pickle.load(open('res/pickle/diabetes\_disease\_classifier\_LR.pkl', 'rb'))

    load\_clf\_RF = pickle.load(open('res/pickle/diabetes\_disease\_classifier\_RF.pkl', 'rb'))

    # Apply models to make predictions

    prediction\_NB = load\_clf\_NB.predict(df)

    prediction\_proba\_NB = load\_clf\_NB.predict\_proba(df)

    #prediction\_KNN = load\_clf\_KNN.predict(df)

    #prediction\_proba\_KNN = load\_clf\_KNN.predict\_proba(df)

    prediction\_DT = load\_clf\_DT.predict(df)

    prediction\_proba\_DT = load\_clf\_DT.predict\_proba(df)

    prediction\_LR = load\_clf\_LR.predict(df)

    prediction\_proba\_LR = load\_clf\_LR.predict\_proba(df)

    prediction\_RF = load\_clf\_RF.predict(df)

    prediction\_proba\_RF = load\_clf\_RF.predict\_proba(df)

    def NB():

        st.subheader('Prediction of Naïve Bayes Classifier')

        NB\_prediction = np.array([0, 1])

        if NB\_prediction[prediction\_NB] == 1:

            st.write("<p style='font-size:20px;color: red'><b>I am Sorryy!! You are Suffering from PIMA Indian Diabetes 😰</b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Naïve Bayes Classifier ')

            st.write(prediction\_proba\_NB)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_NB()

    def DT():

        st.subheader('Prediction of Decision Tree Classifier')

        DT\_prediction = np.array([0, 1])

        if DT\_prediction[prediction\_DT] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorryy!! You are Suffering from PIMA Indian Diabetes 😰</b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Decision Tree Classifier')

            st.write(prediction\_proba\_DT)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_DT()

    def LR():

        st.subheader('Prediction of Logistic Regression')

        LR\_prediction = np.array([0, 1])

        if LR\_prediction[prediction\_LR] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorryy!! You are Suffering from PIMA Indian Diabetes 😰<b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Logistic Regression')

            st.write(prediction\_proba\_LR)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_LR()

    def RF():

        st.subheader('Prediction of Random Forest')

        RF\_prediction = np.array([0, 1])

        if RF\_prediction[prediction\_RF] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am Sorryy!! You are Suffering from PIMA Indian Diabetes 😰</b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Random Forest')

            st.write(prediction\_proba\_RF)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_RF()

    def predict\_best\_algorithm():

        if cmb.best\_model == 'Naive Bayes':

            NB()

        # elif cmb.best\_model == 'K-Nearest Neighbors (KNN)':

        #     KNN()

        elif cmb.best\_model == 'Decision Tree':

            DT()

        elif cmb.best\_model == 'Logistic Regression':

            LR()

        elif cmb.best\_model == 'Random Forest':

            RF()

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

    st.markdown("👈 Provide your input data in the sidebar")    # Displays the user input features

    with st.expander("Prediction Results",expanded=False):

        # Display the input dataframe

        st.write("Your input values are shown below:")

        st.dataframe(input\_df)

        # Call the predict\_best\_algorithm() function

        st.text('Showing you the Best Report Generated by our Service', help='This Report shows Approximate Prediction')

        predict\_best\_algorithm()

        # Tips, Diagnosis, Treatment, and Recommendations.

        st.subheader("Opinions provided by Our Consultancy on PIMA Indian Diabetes 👨‍⚕️")

        tab1, tab2, tab3 = st.tabs(["Advises", "Work-Out", "Diet"])

        with tab1:

            st.subheader("Advices of our Consultancy")

            management\_tips = [

                "1. \*\*Maintain a Healthy Weight:\*\* Losing extra weight and maintaining a healthy body weight can significantly reduce the risk of developing type 2 diabetes",

                "2. \*\*Stay Physically Active:\*\* Engage in regular physical activity to help control weight, lower blood sugar levels, and boost sensitivity to insulin",

                "3. \*\*Adopt a Plant-Based Diet:\*\* Incorporate more healthy plant foods such as fruits, vegetables, whole grains, nuts, and legumes into your diet ",

                "4. \*\*Choose Healthy Fats:\*\* Opt for healthy fats found in foods like avocados, nuts, seeds, and olive oil, while minimizing saturated and trans fats",

                "5. \*\*Limit Processed Foods:\*\* Reduce intake of processed foods, sugary beverages, and refined carbohydrates to lower the risk of insulin resistance and type 2 diabetes",

                "6. \*\*Control Portion Sizes:\*\* Be mindful of portion sizes to prevent overeating and maintain a balanced diet"

            ]

            for tip in management\_tips:

                st.write(f"- {tip}")

        with tab2:

            st.subheader("Exercise Recommended by our Experts:")

            c1, c2, c3 = st.columns([1, 1, 1], gap="small")

            with c1:

                exercise\_recommendations = [

                    "1. \*\*Walking\*\* ",

                    "2. \*\*Cycling\*\* ",

                    "3. \*\*Aerobic Dance\*\* ",

                    "4. \*\*Weightlifting\*\* ",

                    "5. \*\*Swimming\*\* "

                ]

                for exercise in exercise\_recommendations:

                    st.write(f"- {exercise}")

            with c3:

                st\_lottie(

                    lottie\_coding,

                    speed=1,

                    reverse=False,

                    loop=True,

                    quality="medium",

                    height=None,

                    width=None,

                    key=None,

                )

        with tab3:

            st.subheader("Diet Control  Measures Recommended by our Experts:")

            dietary\_recommendations = [

                "1. \*\*Limit Carbohydrates:\*\* Restrict intake of carbohydrates with added sugars or refined grains like white bread and white rice. Opt for carbohydrates from fruits, vegetables, and whole grains",

                "2. \*\*Healthy Eating Plan:\*\* Embrace a diet rich in nutrients, low in fat and calories, with a focus on fruits, vegetables, and whole grains",

                "3. \*\*Balanced Diet:\*\* Ensure meals include a balance of carbohydrates, proteins, and healthy fats to manage blood sugar levels effectively",

                "4. \*\*Portion Control:\*\* Watch portion sizes to avoid overeating and maintain a healthy weight",

                "5. \*\*Consistent Meal Timing:\*\* Maintain regular meal times and spacing throughout the day to help regulate blood sugar levels"

            ]

            for food in dietary\_recommendations:

                st.write(f"- {food}")

    with st.expander("Comparison Study", expanded=False):

        col1, col2 = st.columns(2)

        with col1:

            st.header("Naïve Bayes Classifier")

            cmb.CR\_NB()

        with col2:

            st.header("Decision Tree Classifier")

            cmb.CR\_DT()

        col3, col4 = st.columns(2)

        with col3:

            st.header("Logistic Regression Algorithm")

            cmb.CR\_LR()

        with col4:

            st.header("Random Forest Algorithm")

            cmb.CR\_RF()

    # Create a multiselect for all the plot options

    selected\_plots = st.multiselect("You can see all the Detailed Reports Here 👇",

                                    ["Naïve Bayes", "Decision Tree", "Logistic Regression",

                                     "Random Forest"], default=[],key="ms\_D")

    if "ms\_D" not in st.session\_state:

        st.session\_state.selected\_plots = []

    # Check the selected plots and call the corresponding plot functions

    if selected\_plots:

        col1, col2 = st.columns(2)

        with col1:

            st.text('Understanding the Report',

                    help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

        with col2:

            st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

    placeholder = st.empty()

    # Check the selected plots and call the corresponding plot functions

    if "Naïve Bayes" in selected\_plots:

        with st.spinner("Generating Report of Naïve Bayes Classifier ...."):

            cmb.plt\_NB()

            time.sleep(1)

    # if "K-Nearest Neighbors" in selected\_plots:

    #     with st.spinner("Generating KNN...."):

    #         cmb.plt\_KNN()

    #         time.sleep(1)

    if "Decision Tree" in selected\_plots:

        with st.spinner("Generating Report of Decision Tree Classifier...."):

            cmb.plt\_DT()

            time.sleep(1)

    if "Logistic Regression" in selected\_plots:

        with st.spinner("Generating Report of Logistic Regression...."):

            cmb.plt\_LR()

            time.sleep(1)

    if "Random Forest" in selected\_plots:

        with st.spinner("Generating Report of Random Forest Algorithm...."):

            cmb.plt\_RF()

            time.sleep(1)

    # Remove the placeholder to display the list options

    placeholder.empty()

## **4.6 Code for the Classifier Model of the Heart Disease:**

import numpy as np

import streamlit as st

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

import pickle

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

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

url = "res/dataset/heart.csv"

heart = pd.read\_csv(url)

# Ordinal feature encoding

df = heart.copy()

encode = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST\_Slope']

for col in encode:

    dummy = pd.get\_dummies(df[col], prefix=col)

    df = pd.concat([df, dummy], axis=1)

    del df[col]

    del dummy

# Separating X and y

X = df.drop('HeartDisease', axis=1)

Y = df['HeartDisease']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

"""\_\_\_\_\_\_\_\_Naive Bayes Algorithm\_\_\_\_\_\_\_\_"""

# Train the Naive Bayes classifier

nb\_classifier = GaussianNB(var\_smoothing=1e-9)

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

# Predict using the Naive Bayes classifier

nb\_predictions = nb\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Naive Bayes classifier

nb\_cm = confusion\_matrix(y\_test, nb\_predictions)

nb\_accuracy = accuracy\_score(y\_test, nb\_predictions)

nb\_classifier\_report = classification\_report(y\_test, nb\_predictions)

nb\_classifier\_report\_dict = classification\_report(y\_test, nb\_predictions, output\_dict=True)

def plt\_NB():

    def classifier\_report():

        report\_df = pd.DataFrame(nb\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Naive Bayes Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for Naive Bayes classifier

    plt.figure()

    plt.imshow(nb\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Naive Bayes')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = nb\_cm.max() / 2

    for i, j in np.ndindex(nb\_cm.shape):

        plt.text(j, i, format(nb\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if nb\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_NB():

        report\_df = pd.DataFrame(nb\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Naive Bayes Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Decision Tree\_\_\_\_\_\_\_\_"""

# Train the Decision Tree classifier

dt\_classifier = DecisionTreeClassifier(max\_depth=None)

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

# Predict using the Decision Tree classifier

dt\_predictions = dt\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Decision Tree classifier

dt\_cm = confusion\_matrix(y\_test, dt\_predictions)

dt\_accuracy = accuracy\_score(y\_test, dt\_predictions)

dt\_classifier\_report = classification\_report(y\_test, dt\_predictions)

dt\_classifier\_report\_dict = classification\_report(y\_test, dt\_predictions, output\_dict=True)

def plt\_DT():

    def classifier\_report():

        report\_df = pd.DataFrame(dt\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Decision Tree Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for Naive Bayes classifier

    plt.figure()

    plt.imshow(dt\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Decision Tree')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = dt\_cm.max() / 2

    for i, j in np.ndindex(dt\_cm.shape):

        plt.text(j, i, format(dt\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if dt\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_DT():

        report\_df = pd.DataFrame(dt\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Decision Tree Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Logistic Regression Algorithm\_\_\_\_\_\_\_\_"""

# Train the Logistic Regression classifier

lr\_classifier = LogisticRegression(max\_iter=1000)

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

# Predict using the Logistic Regression classifier

lr\_predictions = lr\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Logistic Regression classifier

lr\_cm = confusion\_matrix(y\_test, lr\_predictions)

lr\_accuracy = accuracy\_score(y\_test, lr\_predictions)

lr\_classifier\_report = classification\_report(y\_test, lr\_predictions)

lr\_classifier\_report\_dict = classification\_report(y\_test, lr\_predictions, output\_dict=True)

def plt\_LR():

    def classifier\_report():

        report\_df = pd.DataFrame(lr\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Logistic Regression Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for classifier

    plt.figure()

    plt.imshow(lr\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Logistic Regression')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = lr\_cm.max() / 2

    for i, j in np.ndindex(lr\_cm.shape):

        plt.text(j, i, format(lr\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if lr\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_LR():

        report\_df = pd.DataFrame(lr\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Logistic Regression Classifier Report")

        st.write(report\_df)

        st.write()

"""\_\_\_\_\_\_\_\_Random Forest Algorithm\_\_\_\_\_\_\_\_"""

# Train the Random Forest classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100)

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

# Predict using the Random Forest classifier

rf\_predictions = rf\_classifier.predict(X\_test)

# Calculate confusion matrix and accuracy for Random Forest classifier

rf\_cm = confusion\_matrix(y\_test, rf\_predictions)

rf\_accuracy = accuracy\_score(y\_test, rf\_predictions)

rf\_classifier\_report = classification\_report(y\_test, rf\_predictions)

rf\_classifier\_report\_dict = classification\_report(y\_test, rf\_predictions, output\_dict=True)

def plt\_RF():

    def classifier\_report():

        report\_df = pd.DataFrame(rf\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Random Forest Classifier Report")

        st.write(report\_df)

        st.write()

    # Plot confusion matrix for  classifier

    plt.figure()

    plt.imshow(rf\_cm, interpolation='nearest', cmap=plt.cm.Reds)

    plt.title('Confusion Matrix - Random Forest')

    plt.colorbar()

    plt.xticks([0, 1], ['No Disease', 'Disease'])

    plt.yticks([0, 1], ['No Disease', 'Disease'])

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    thresh = rf\_cm.max() / 2

    for i, j in np.ndindex(rf\_cm.shape):

        plt.text(j, i, format(rf\_cm[i, j], 'd'), ha='center', va='center',

                 color='white' if rf\_cm[i, j] > thresh else 'black')

    # Display the confusion matrix on Streamlit

    st.set\_option('deprecation.showPyplotGlobalUse', False)

    col1, col2 = st.columns(2)

    with col1:

        classifier\_report()

    with col2:

        st.pyplot()

def CR\_RF():

        report\_df = pd.DataFrame(rf\_classifier\_report\_dict).transpose()

        # Display the classification report as a table using st.write()

        st.write("Random Forest Classifier Report")

        st.write(report\_df)

        st.write()

# Selecting the best suitable algorithm based on classifier\_report

models = {

    'Naive Bayes': nb\_classifier\_report,

    #'K-Nearest Neighbors (KNN)': knn\_classifier\_report,

    'Decision Tree': dt\_classifier\_report,

    'Logistic Regression': lr\_classifier\_report,

    'Random Forest': rf\_classifier\_report

}

best\_model = max(models, key=models.get)

# Saving the model

pickle.dump(nb\_classifier, open('res/pickle/heart\_disease\_classifier\_NB.pkl', 'wb'))

#pickle.dump(knn\_classifier, open('res/pickle/heart\_disease\_classifier\_KNN.pkl', 'wb'))

pickle.dump(dt\_classifier, open('res/pickle/heart\_disease\_classifier\_DT.pkl', 'wb'))

pickle.dump(lr\_classifier, open('res/pickle/heart\_disease\_classifier\_LR.pkl', 'wb'))

pickle.dump(rf\_classifier, open('res/pickle/heart\_disease\_classifier\_RF.pkl', 'wb'))

## **4.7 Code for the Heart Disease App for Streamlit:**

import time

import streamlit as st

import pandas as pd

from Classifier\_Models import Classifier\_model\_builder\_heart as cmb

import pickle

import numpy as np

from streamlit\_toggle import st\_toggle\_switch

import json

from streamlit\_lottie import st\_lottie

def app():

    def load\_lottiefile(filepath: str):

        with open(filepath, "r") as f:

            return json.load(f)

    lottie\_coding = load\_lottiefile("res/Yoga\_Vrikshasana.json")

    st.title("Heart Disease Detector")

    st.info("This Model predicts whether a person is suffering from Heart Disease or not")

    st.sidebar.header('Report Uploader')

    uploaded\_file = st.sidebar.file\_uploader("Upload your parameters of your Report through a CSV File", type=["csv"])

    if uploaded\_file is not None:

        input\_df = pd.read\_csv(uploaded\_file)

    else:

        def patient\_details():

            sex = st.sidebar.selectbox('Sex', ('M', 'F'))

            ChestPainType = st.sidebar.selectbox('Chest Pain Type', ('TA', 'ASY', 'NAP'))

            RestingECG = st.sidebar.selectbox('Resting Electrocardiogram', ('Normal', 'ST', 'LVH'))

            ExerciseAngina = st.sidebar.selectbox('ExerciseAngina', ('Y', 'N'))

            ST\_Slope = st.sidebar.selectbox('ST Slope', ('Up', 'Flat', 'Down'))

            Age = st.sidebar.number\_input('Age', 28, 77)

            RestingBP = st.sidebar.number\_input('Resting Blood Pressure', 0, 200)

            Cholesterol = st.sidebar.number\_input('Cholesterol', 0, 603)

            MaxHR = st.sidebar.number\_input('Maximum Heart Rate', 60, 202)

            Oldpeak = st.sidebar.number\_input('Old peak', -2, 6)

            FastingBS = st.sidebar.number\_input('Fasting Blood Sugar', 0, 1)

            data = {'Age': Age,

                    'Sex': sex,

                    'ChestPainType': ChestPainType,

                    'RestingBP': RestingBP,

                    'Cholesterol': Cholesterol,

                    'FastingBS': FastingBS,

                    'RestingECG': RestingECG,

                    'MaxHR': MaxHR,

                    'ExerciseAngina': ExerciseAngina,

                    'Oldpeak': Oldpeak,

                    'ST\_Slope': ST\_Slope, }

            features = pd.DataFrame(data, index=[0])

            return features

        input\_df = patient\_details()

    heart\_disease\_raw = pd.read\_csv('res/dataset/heart.csv')

    heart = heart\_disease\_raw.drop(columns=['HeartDisease'])

    df = pd.concat([input\_df, heart], axis=0)

    # Encoding of ordinal features

    encode = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST\_Slope']

    for col in encode:

        dummy = pd.get\_dummies(df[col], prefix=col)

        df = pd.concat([df, dummy], axis=1)

        del df[col]

    df = df[:1]  # Selects only the first row (the user input data)

    df.loc[:, ~df.columns.duplicated()]

    if uploaded\_file is not None:

        st.write(df)

    else:

        st.write('Waiting for the Report to be Uploaded... Currently displaying the Parameters given manually')

        df = df.loc[:, ~df.columns.duplicated()]

        st.write(df)

    # Load the classification models

    load\_clf\_NB = pickle.load(open('res/pickle/heart\_disease\_classifier\_NB.pkl', 'rb'))

    #load\_clf\_KNN = pickle.load(open('res/pickle/heart\_disease\_classifier\_KNN.pkl', 'rb'))

    load\_clf\_DT = pickle.load(open('res/pickle/heart\_disease\_classifier\_DT.pkl', 'rb'))

    load\_clf\_LR = pickle.load(open('res/pickle/heart\_disease\_classifier\_LR.pkl', 'rb'))

    load\_clf\_RF = pickle.load(open('res/pickle/heart\_disease\_classifier\_RF.pkl', 'rb'))

    # Apply models to make predictions

    prediction\_NB = load\_clf\_NB.predict(df)

    prediction\_proba\_NB = load\_clf\_NB.predict\_proba(df)

    #prediction\_KNN = load\_clf\_KNN.predict(df)

    #prediction\_proba\_KNN = load\_clf\_KNN.predict\_proba(df)

    prediction\_DT = load\_clf\_DT.predict(df)

    prediction\_proba\_DT = load\_clf\_DT.predict\_proba(df)

    prediction\_LR = load\_clf\_LR.predict(df)

    prediction\_proba\_LR = load\_clf\_LR.predict\_proba(df)

    prediction\_RF = load\_clf\_RF.predict(df)

    prediction\_proba\_RF = load\_clf\_RF.predict\_proba(df)

    def NB():

        st.subheader('Prediction of Naïve Bayes Classifier')

        NB\_prediction = np.array([0, 1])

        if NB\_prediction[prediction\_NB] == 1:

            st.write("<p style='font-size:20px;color: red'><b>I am sorryy!! You are suffering from Heart Disease 😰</b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Naïve Bayes Classifier')

            st.write(prediction\_proba\_NB)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_NB()

    def DT():

        st.subheader('Prediction of Decision Tree Classifier')

        DT\_prediction = np.array([0, 1])

        if DT\_prediction[prediction\_DT] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am sorryy!! You are suffering from Heart Disease 😰</b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        enabled = st\_toggle\_switch("See detailed prediction")

        #Toggle Switch

        if enabled:

            st.subheader('Report Generated by Decision Tree Classifier ')

            st.write(prediction\_proba\_DT)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_DT()

    def LR():

        st.subheader('Prediction of Logistic Regression')

        LR\_prediction = np.array([0, 1])

        if LR\_prediction[prediction\_LR] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am sorryy!! You are suffering from Heart Disease 😰<b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Logistic Regression')

            st.write(prediction\_proba\_LR)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_LR()

    def RF():

        st.subheader('Prediction of Random Forest')

        RF\_prediction = np.array([0, 1])

        if RF\_prediction[prediction\_RF] == 1:

            st.write("<p style='font-size:20px; color: red'><b>I am sorryy!! You are suffering from Heart Disease 😰</b></p>",

                     unsafe\_allow\_html=True)

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

        #Toggle Switch

        enabled = st\_toggle\_switch("Show Detailed Report")

        if enabled:

            st.subheader('Report Generated by Random Forest')

            st.write(prediction\_proba\_RF)

            col1, col2 = st.columns(2)

            with col1:

                st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

            cmb.plt\_RF()

    def predict\_best\_algorithm():

        if cmb.best\_model == 'Naive Bayes':

            NB()

        # elif cmb.best\_model == 'K-Nearest Neighbors (KNN)':

        #     KNN()

        elif cmb.best\_model == 'Decision Tree':

            DT()

        elif cmb.best\_model == 'Logistic Regression':

            LR()

        elif cmb.best\_model == 'Random Forest':

            RF()

        else:

            st.write("<p style='font-size:20px;color: green'><b>You are absolutely Fit 'n Fine 👍</b></p>", unsafe\_allow\_html=True)

    st.markdown("👈 Provide your input data in the sidebar")

    # Displays the user input features

    with st.expander("Prediction Results",expanded=False):

        # Display the input dataframe

        st.write("Your input values are shown below:")

        st.dataframe(input\_df)

        # Call the predict\_best\_algorithm() function

        st.text('Showing you the Best Report Generated by our Service', help='This Report shows Approximate Prediction')

        predict\_best\_algorithm()

        # Tips, Diagnosis, Treatment, and Recommendations.

        st.subheader("Opinions provided by Our Consultancy on Heart Disease 👨‍⚕️")

        tab1, tab2, tab3 = st.tabs(["Advices", "Work-Out", "Diet"])

        with tab1:

            st.subheader("Advices of Our Consultancy:")

            prevention\_tips = [

                "1. \*\*Monitor Cholesterol and Blood Pressure:\*\* Regularly check cholesterol and blood pressure levels and follow medical advice to keep them within healthy ranges.",

                "2. \*\*Adopt a Heart-Healthy Diet:\*\* Consume a balanced diet rich in fruits, vegetables, whole grains, lean proteins, and healthy fats to maintain heart health and manage weight.",

                "3. \*\*Quit Smoking:\*\* Avoid all forms of tobacco to reduce the risk of heart disease and improve overall health.",

                "4. \*\*Regular Physical Activity:\*\* Aim for at least 30 to 60 minutes of moderate-intensity exercise daily to strengthen the heart and lower blood pressure.",

                "5. \*\*Limit Alcohol Intake:\*\* Consume alcohol in moderation, as excessive alcohol consumption can contribute to heart disease.",

                "6. \*\*Maintain a Healthy Weight:\*\* Achieve and maintain a healthy weight through a combination of healthy eating and regular physical activity to reduce strain on the heart."

            ]

            for tip in prevention\_tips:

                st.write(f"- {tip}")

        with tab2:

            st.subheader("Exercise Recommended by our Experts:")

            c1, c2, c3 = st.columns([1, 1, 1], gap="small")

            with c1:

                exercises = [

                    "1. \*\*Trikonasana (Triangle Pose)\*\*",

                    "2. \*\*Standing Forward Bend\*\*",

                    "3. \*\*Extended Triangle Pose\*\*",

                    "4. \*\*Bridge Pose\*\*",

                    "5. \*\*Chair Pose\*\*"

                ]

                for exercise in exercises:

                    st.write(f"- {exercise}")

            with c3:

                st\_lottie(

                    lottie\_coding,

                    speed=1,

                    reverse=False,

                    loop=True,

                    quality="medium",

                    height=None,

                    width=None,

                    key=None,

                )

        with tab3:

            st.subheader("Diet Control  Measures Recommended by our Experts:")

            diet\_tips = [

                "1. \*\*Control Portion Size:\*\* Manage portion sizes to prevent overeating and maintain a healthy weight, which is crucial for heart health. ",

                "2. \*\*Increase Fruits and Vegetables:\*\* Incorporate plenty of fruits and vegetables into your diet for essential vitamins, minerals, and antioxidants that support heart health. Aim for a variety of colors and types.",

                "3. \*\*Choose Whole Grains:\*\* Opt for whole grains such as brown rice, whole wheat bread, and oats instead of refined grains to increase fiber intake and promote heart health",

                "4. \*\*Limit Saturated and Trans Fats:\*\* Reduce intake of saturated fats found in fatty meats, butter, and cheese, as well as trans fats found in processed and fried foods, to lower cholesterol levels and decrease the risk of heart disease.",

                "5. \*\*Choose Healthy Fats:\*\* Include sources of healthy fats such as nuts, seeds, avocados, and olive oil, which can improve cholesterol levels and protect heart health.",

                "6. \*\*Limit Sodium Intake:\*\* Reduce consumption of high-sodium foods like processed foods, canned soups, and salty snacks to manage blood pressure and reduce the risk of heart disease."

            ]

            for tip in diet\_tips:

                st.write(f"- {tip}")

    with st.expander("Comparison Study", expanded=False):

        col1, col2 = st.columns(2)

        with col1:

            st.header("Naïve Bayes Classifier")

            cmb.CR\_NB()

        with col2:

            st.header("Decision Tree Classifier")

            cmb.CR\_DT()

        col3, col4 = st.columns(2)

        with col3:

            st.header("Logistic Regression Algorithm")

            cmb.CR\_LR()

        with col4:

            st.header("Random Forest Algorithm")

            cmb.CR\_RF()

    # Create a multiselect for all the plot options

    selected\_plots = st.multiselect("You can see all the Detailed Reports Here 👇",

                                    ["Naïve Bayes", "Decision Tree", "Logistic Regression",

                                     "Random Forest"], default=[], key="ms\_H")

    if "ms\_H" not in st.session\_state:

        st.session\_state.selected\_plots = []

    # Check the selected plots and call the corresponding plot functions

    if selected\_plots:

        col1, col2 = st.columns(2)

        with col1:

            st.text('Understanding the Report',

                        help="It helps assess the model's ability to correctly identify classes and its overall performance in classifying data.")

            with col2:

                st.text('Understanding Confusion Matrix',

                        help="A confusion matrix is a performance evaluation tool in machine learning that provides a concise summary of the performance of a classification model. It presents a tabular representation of the model's predictions compared to the actual outcomes.")

    placeholder = st.empty()

    # Check the selected plots and call the corresponding plot functions

    if "Naïve Bayes" in selected\_plots:

        with st.spinner("Generating Report of Naïve Bayes Classifier...."):

            cmb.plt\_NB()

            time.sleep(1)

    # if "K-Nearest Neighbors" in selected\_plots:

    #     with st.spinner("Generating KNN...."):

    #         cmb.plt\_KNN()

    #         time.sleep(1)

    if "Decision Tree" in selected\_plots:

        with st.spinner("Generating Report of Decision Tree Classifier...."):

            cmb.plt\_DT()

            time.sleep(1)

    if "Logistic Regression" in selected\_plots:

        with st.spinner("Generating Report of Logistic Regression...."):

            cmb.plt\_LR()

            time.sleep(1)

    if "Random Forest" in selected\_plots:

        with st.spinner("Generating Report of Random Forest Algorithm...."):

            cmb.plt\_RF()

            time.sleep(1)

    # Remove the placeholder to display the list options

    placeholder.empty()