THYROID DISEASE CLASSIFICATION USING MACHINE LEARNING

TEAM ID:NM2023TMID32113

	Name of the Student	NM ID
TEAM LEADER	P.SWETHA	5C8B31E8228163EA03E4C36B80F928D7
TEAM MEMBERS	M.AARTHI	03014DA5DC22AA6A7A84DFA814565D93
	T.NISRINBHANU	6BFDAC52C428E0E89C645EE168DAC4C0
	S.RASIKA	7AB3EA0D911F3958C8D4A86CC715EDD0

GOVERNMENT ARTS & SCIENCE COLLEGE

KADAYANALLUR

INDEX

THYROID DISEASE CLASSIFICATION USING MACHINE LEARNING

1.INTRODUCTION

OVERVIEW

a) OVERVIEW OF EXISTING SYSTEM

1. Thyroid Disease Diagnosis Using Machine Learning Algorithms: A Review: This study reviewed various machine learning algorithms such as Support Vector Machines (SVM), Random Forest, and Artificial Neural Networks (ANN) used for the diagnosis of thyroid disease.
2. Thyroid Disease Diagnosis Using Deep Learning Techniques: In this study, researchers proposed a deep learning-based approach for the diagnosis of thyroid disease. They used Convolutional Neural Networks (CNN) to classify the disease.
3. Early Diagnosis of Thyroid Disease using Machine Learning: This research focused on developing an early diagnosis system for thyroid disease using machine learning algorithms such as Logistic Regression, SVM, and Decision Trees.

b) OVERVIEW OF PROPOSED SYSTEM

1. Thyroid Disease Detection Using Flask Framework: This project proposes the development of a web-based application for the detection of thyroid disease using the Flask framework. The application will use machine learning algorithms to classify the disease.

2. Thyroid Disease Diagnosis using Machine Learning and Genetic Algorithm: This study proposes the use of machine learning algorithms such as ANN and SVM along with the Genetic Algorithm to improve the accuracy of thyroid disease diagnosis.
3. Thyroid Nodule Detection and Diagnosis Using Machine Learning: This project proposes the use of machine learning algorithms such as CNN and SVM for the detection and diagnosis of thyroid nodules.

PURPOSE

The purpose of a thyroid disease classification using machine learning project is to develop an accurate and reliable system that can assist in the diagnosis and classification of thyroid diseases.

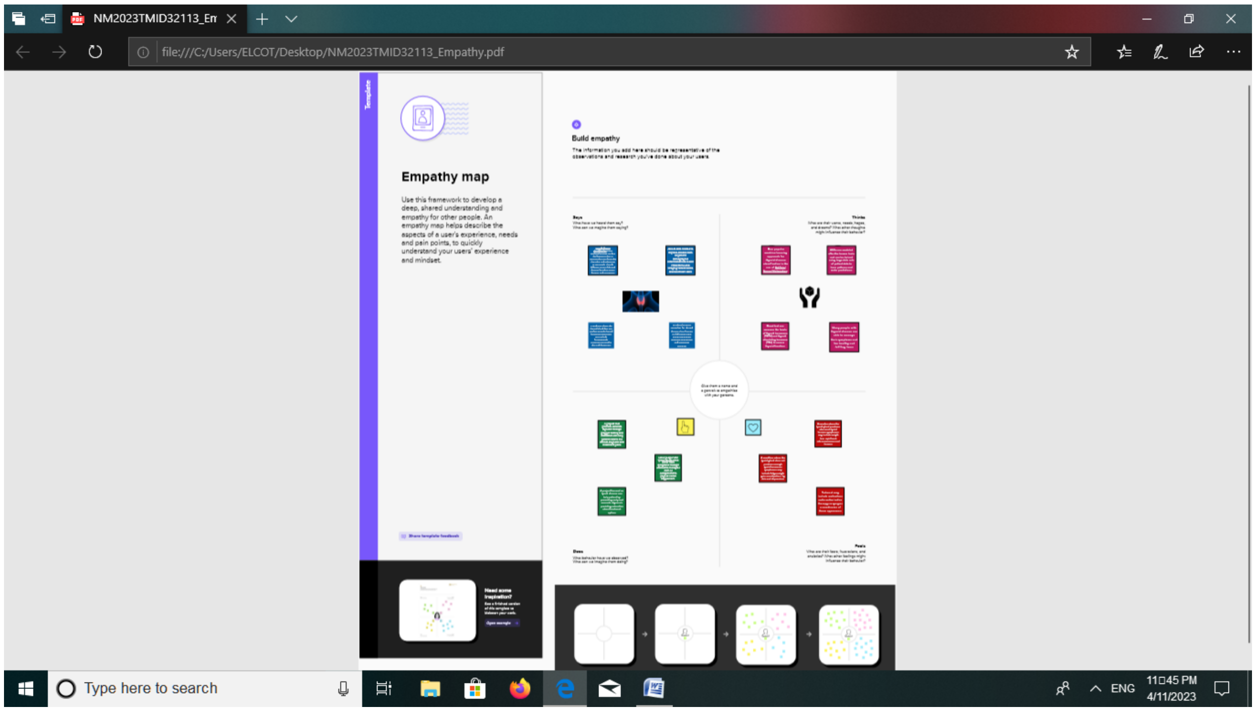
The thyroid gland is an important endocrine gland that produces hormones that regulate metabolism in the body. Thyroid diseases are common and can cause a range of symptoms, including weight gain or loss, fatigue, and mood changes. Accurate and timely diagnosis of thyroid diseases is critical for effective treatment.

Machine learning algorithms can be trained on large datasets of patient information, including clinical symptoms, laboratory tests, and imaging results, to develop models that can accurately predict the likelihood of a patient having a particular thyroid disease. These models can then be used by healthcare professionals to assist in the diagnosis and treatment of thyroid diseases.

The benefits of using machine learning for thyroid disease classification include faster and more accurate diagnosis, reduced healthcare costs, and improved patient outcomes. However, it is important to note that the performance of the machine learning models is highly dependent on the quality and quantity of the data used for training. Therefore, it is important to ensure that the data used is representative of the population being studied and that appropriate measures are taken to address any biases in the data.

2.PROBLEM DEFINITION & DESIGN THINKING

EMPATHY MAP



file:///C:/Users/ELCOT/Desktop/NM2023TMID32113_Empathy.pdf

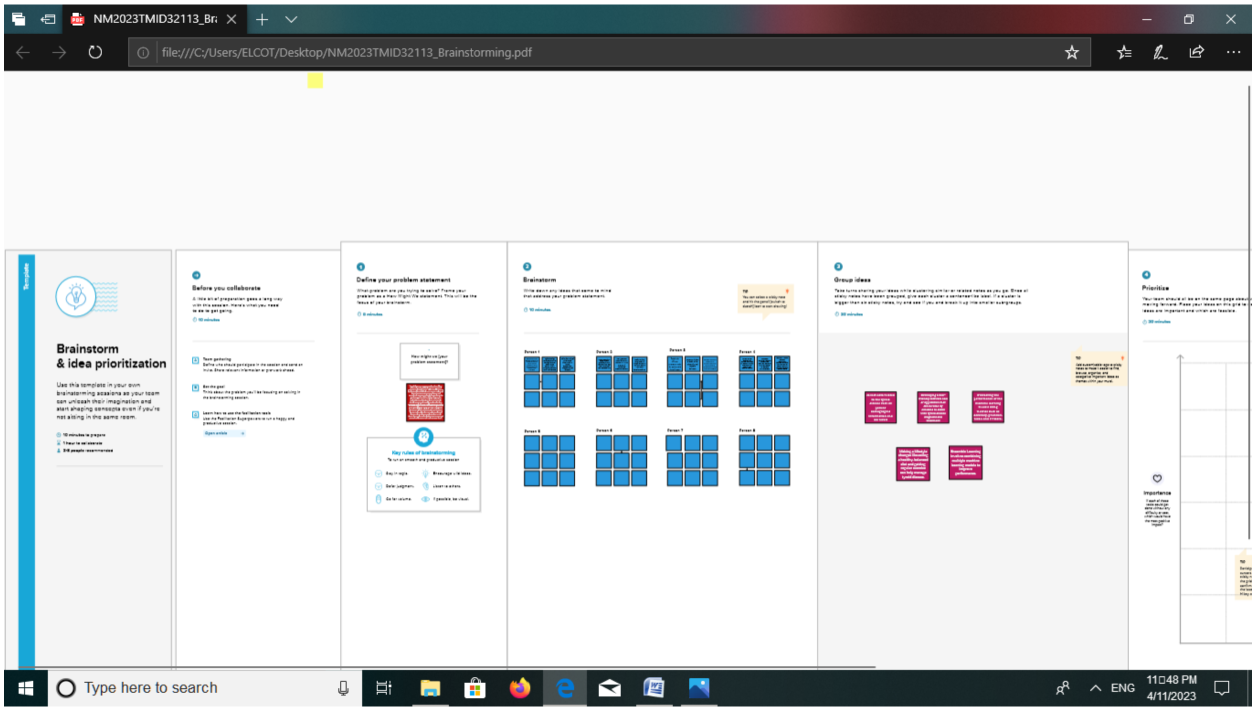
Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

IDERATION & BRAINSTORIMING MAP



file:///C:/Users/ELCOT/Desktop/NM2023TMID32113_Brainstorming.pdf

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Before you collaborate

Define your problem statement

Key rules of brainstorming

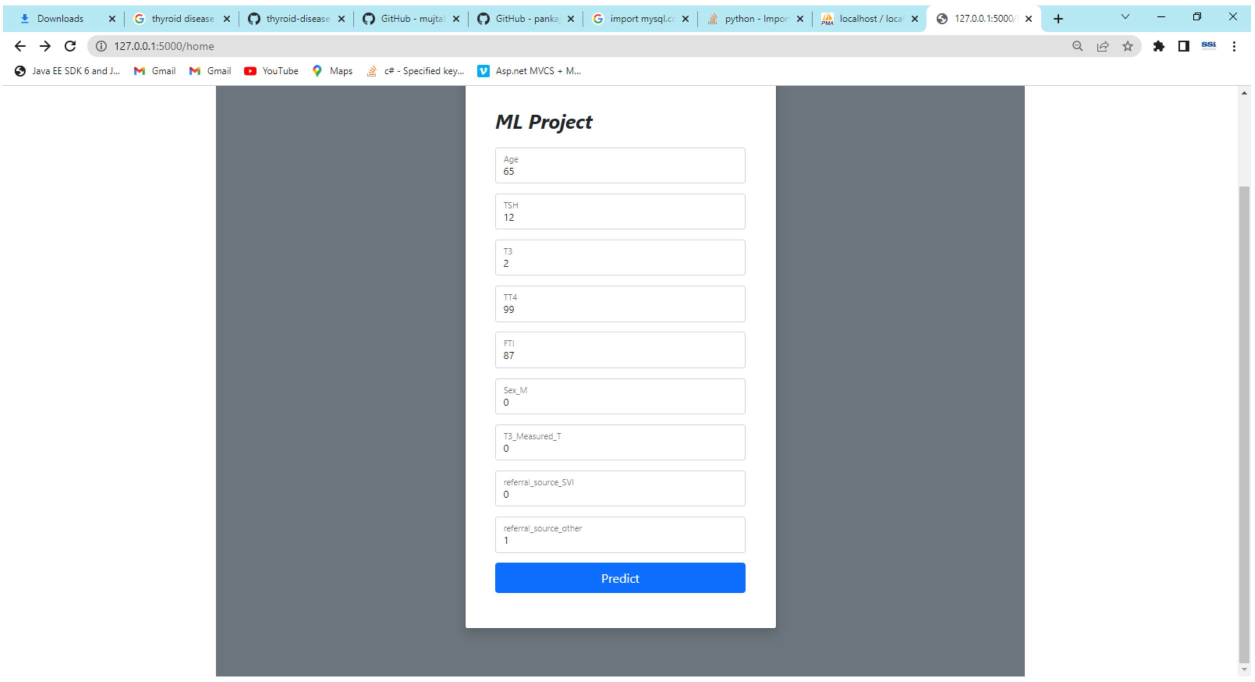
Brainstorm

Group ideas

Prioritize

3.RESULT

Positive result:


127.0.0.1:5000/home

ML Project

Age
65

TSH
12

T3
2

TT4
99

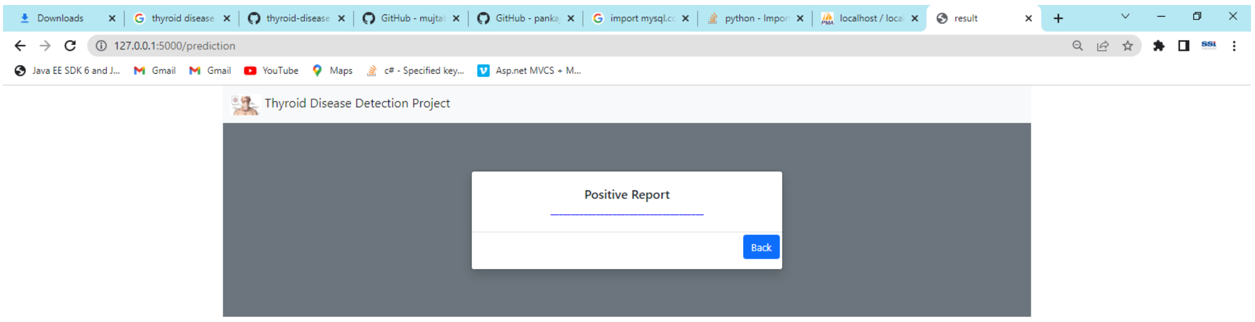
FTI
87

Sex_M
0

T3_Measured_T
0

referral_source_SVI
0

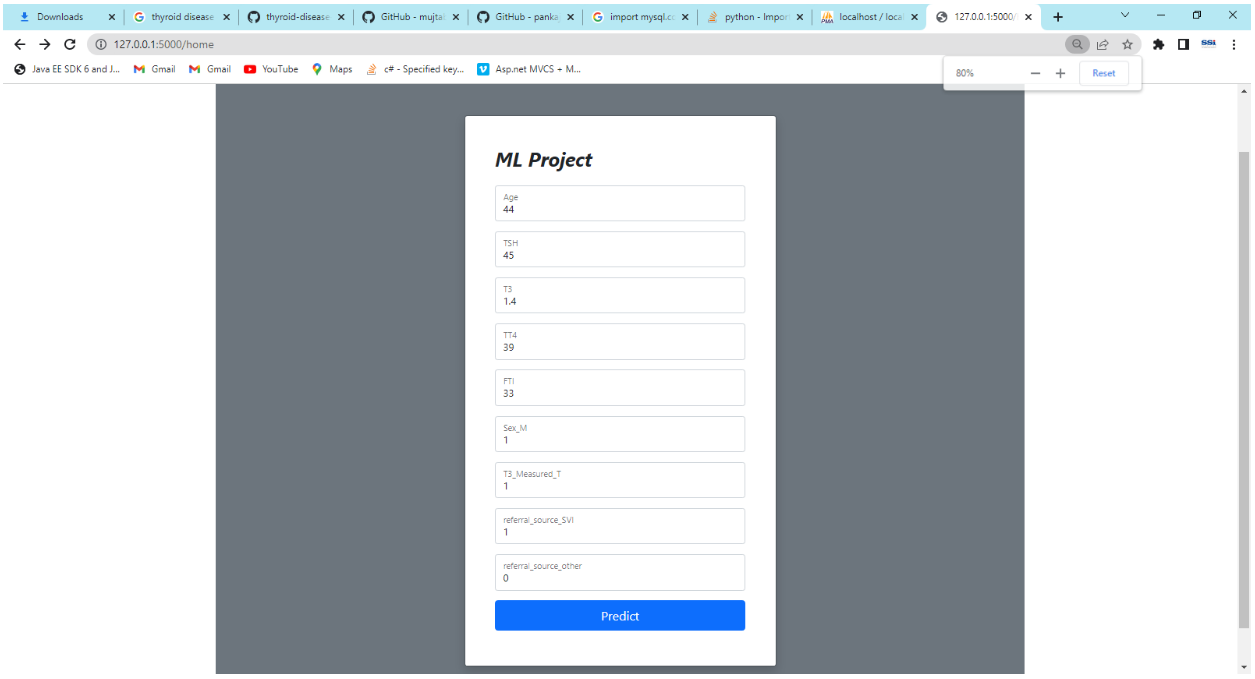
referral_source_other
1


127.0.0.1:5000/prediction

Thyroid Disease Detection Project

Positive Report

Negative result:



ML Project

Age
44

TSH
45

T3
1.4

TT4
39

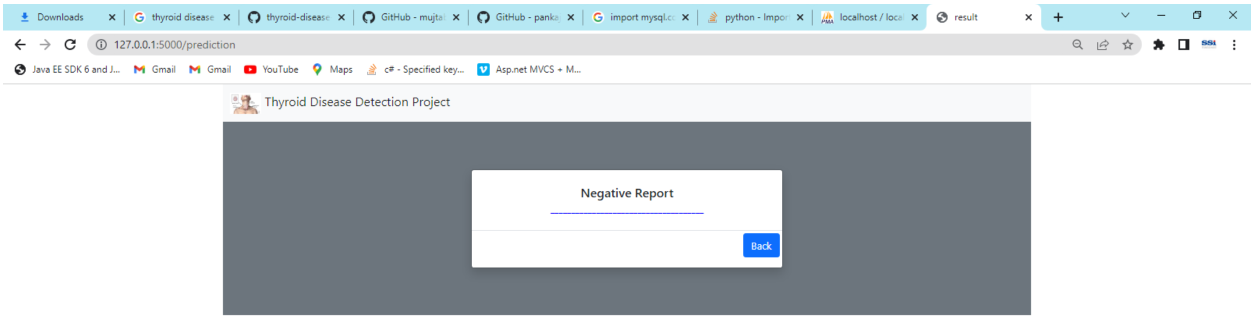
FTI
33

Sex_M
1

T3_Measured_T
1

referral_source_SVI
1

referral_source_other
0



Negative Report

4.ADVANTAGES AND DISADVANTAGES

ADVANTAGES

Accurate diagnosis: Machine learning algorithms can accurately classify thyroid disease types, which can help doctors make accurate diagnose.

Time-saving: Machine learning algorithms can analyze large amounts of data in a short amount of time, which can save healthcare professionals' time and allow them to treat more patients

.

Early detection: Machine learning algorithms can detect thyroid diseases at an early stage, which can help prevent serious complications and improve patient outcomes.

Personalized treatment: Machine learning algorithms can analyze patient data and recommend personalized treatment options based on a patient's specific thyroid condition.

DISADVANTAGES

Data quality: The accuracy of machine learning algorithms is highly dependent on the quality and quantity of data used for training. If the data is biased or incomplete, the algorithm may not be accurate.

Overfitting: Machine learning algorithms can sometimes overfit to the data used for training, leading to inaccurate predictions on new data.

Interpretability: Machine learning algorithms can be difficult to interpret, making it hard to understand how they arrived at their conclusions. This can be a concern for healthcare professionals who need to explain diagnoses and treatments to patients.

Cost: Developing and implementing machine learning algorithms can be expensive, and may not be feasible for all healthcare organizations.

5.APPLICATION

The application of Thyroid Disease Classification using Machine Learning can be beneficial in various ways, including:

Early Detection: ML-based thyroid disease classification can aid in the early detection of thyroid disease, enabling healthcare providers to initiate timely treatment.

Personalized Treatment: Machine learning algorithms can analyze patient data and provide personalized treatment plans, which can improve patient outcomes.

Improved Diagnosis Accuracy: Machine Learning algorithms can identify patterns and detect subtle differences that may be difficult for human doctors to identify, leading to more accurate diagnoses.

Reduced Healthcare Costs: Early detection and personalized treatment can reduce healthcare costs by preventing complications associated with thyroid disease and reducing the number of unnecessary tests or exams.

Telemedicine: Thyroid Disease Classification using Machine Learning can be implemented in telemedicine, enabling patients to receive accurate diagnoses and personalized treatment plans remotely.

Healthcare Resource Allocation: ML-based thyroid disease classification can assist in allocating healthcare resources effectively by prioritizing patients with more severe thyroid disease symptoms.

Research: Machine learning algorithms can be used to analyze large datasets of thyroid disease cases, providing insights that may help in developing new treatments or therapies.

Overall, the application of Thyroid Disease Classification using Machine Learning can improve the accuracy of diagnosis, enable personalized treatment, reduce healthcare costs, and enhance the overall quality of patient

6. CONCLUSION

Thyroid disease is one of the diseases that afflict the world's population, and the number of cases of this disease is increasing. Because of medical reports that show serious imbalances in thyroid diseases, our study deals with the classification of thyroid disease between hyperthyroidism and hypothyroidism. This disease was classified using algorithms. Thyroid Detection using Machine Learning is a project idea that aims a smart and precise way to predict thyroid disease. We have made use of CNN and SVM algorithms to train our dataset and to predict thyroid disease with more accuracy. Here the machine is trained to detect whether the person normal, hyper hypothyroidism based on the user's input. So when user enters data in web app the data will be processed in backend (model) and the result will be displayed on the screen. Our objective was to give society an efficient and precise way of machine learning which can be used in applications aiming to perform disease detection.

7. FUTURE SCOPE

Further development can be done by using image processing of ultrasonic scanning of thyroid images to predict thyroid nodules and cancer, which cannot be recognized in blood test report. By combining both the results, thyroid disease prediction can cover all thyroid related diseases.

The future scope of Thyroid Disease Classification using Machine Learning is significant and holds promise for various applications, including:

Improved Precision: Machine learning algorithms can be further developed to improve precision in the diagnosis of thyroid disease, resulting in even more accurate and reliable diagnoses.

Increased Efficiency: The use of machine learning algorithms can enhance the efficiency of healthcare systems by reducing the workload on healthcare providers and improving resource allocation.

Personalized Medicine: Machine learning algorithms can be used to develop personalized medicine plans for patients based on their unique medical history and symptoms, leading to better patient outcomes.

Integration with Wearable Devices: The integration of machine learning algorithms with wearable devices can enable real-time monitoring of patient data, leading to early detection and timely intervention in thyroid disease cases.

Better Data Quality: The development of robust data collection techniques and the availability of high-quality medical datasets can enhance the accuracy and reliability of machine learning algorithms used in thyroid disease classification.

Research and Development: Machine learning algorithms can be used to identify new patterns and insights in large datasets of thyroid disease cases, leading to new treatments or therapies for thyroid disease.

Global Accessibility: The integration of machine learning algorithms with telemedicine can improve access to thyroid disease diagnosis and treatment in underserved areas, reducing the global burden of thyroid disease.

In conclusion, the future scope of Thyroid Disease Classification using Machine Learning is vast and holds significant promise for improving patient outcomes, enhancing the efficiency of healthcare systems, and advancing research in the field of thyroid disease.

8.APPENDIX

CODING:

```
from train_build import building_model
import pickle
from flask import Response
import numpy as np
from Logs import logger
from flask import Flask,request,render_template,redirect,url_for
from train_build import building_model
import training_validation
import js2py
from training_validation import train_validation
from DBOperations import operations
app=Flask(__name__,template_folder='templates')
@app.route("/",methods=['GET','POST'])
def login_page():
    return render_template('admin_login.html')
@app.route("/login",methods=['POST','GET'])
def admin_login():
    loger = logger.App_Logger()
    f=open("Prediction_logs/app_logs.txt",'a+')
    loger.log(f,"admin_login is started")
    try:
        values=[value for value in request.form.values()]
        obj=operations.Operation_DB()
        conn=obj.connection("thyroid_peoject")
        mycursor=conn.cursor()
        mycursor.execute("select * from `login_table` where user_id='{}' and
admin_password='{}'".format(values[0],values[1]))
        result=mycursor.fetchall()
        if result:
            return redirect('home') # for route and redirect(url_for('function_name')) for function
calling
            # return redirect(url_for('home_page'))
        else:
            return Response("Failed to login")
    except:
        loger.log(f,"Error in admin_login function ")
        f.close()
        return Response("Failed to connect")
    f.close()
@app.route("/home", methods=['POST','GET'])
def home_page():
    return  render_template('index.html')
@app.route("/training",methods=['GET','POST'])
def training_model():
    loger = logger.App_Logger()
    f = open("Prediction_logs/app_logs.txt", 'a+')
    loger.log(f,"training function is started!!")
    try:
        file_path="Datasets/hypothyroid.csv"
        object=building_model.Build(file_path)
        object.make_a_model(file_path)
        loger.log(f,"training done successfully")
        # return redirect("home")
        f.close()
    except Exception as e:
        loger.log(f, "Error Occurred in training "+e)
        f.close()
        return Response ("Error Occurred %s " % e)
    f.close()
    # return Response("Training done successful")
    return redirect("home")
@app.route('/prediction',methods=['POST','GET'])
def prediction_function():
    loger = logger.App_Logger()
    f = open("Prediction_logs/app_logs.txt", 'a+')
    loger.log(f,"prediction function is started !!")
    try:
        obj = operations.Operation_DB()
        db_name = "thyroid_peoject"
        loger=logger.App_Logger()
        loger.log(f,"prediction function is starter!!")
        model=pickle.load(open("Models/logistic_model.pkl",'rb'))
        values=[float(value) for value in request.form.values()]
        final=np.array([[values]])
        result=model.predict(final[0])
        result=int(result)
        values.append(result)
        obj.insert_on_table(db_name,values)
        if result==0:
            var = "Negative Report"
        else:
            var="Positive Report"
```

```
          loger.log(f, "prediction function is done successfully")
          f.close()
          return render_template("result.html",value=var)
        except Exception as e:
          loger.log(f, "Error Occurred : "+e)
          f.close()
          return Response ("Error Occurred %s " % e)
    f.close()
    return Response ("Prediction Successfully Done")
if __name__=="__main__":
    app.run(debug=True)
```

Training.py

```
from Training_validation import row_data_validations as rdw
from Logs import logger
from Training_validation import data_transformation as dtf
from Training_validation import feature_selection as fs
class train_validation:
    def __init__(self,file_path):
        self.file_path=file_path
        self.row_data=rdw.row_data_validation(file_path)
        self.data_transform=dtf.transformation()
        self.file_object=open("training_logs/training_main.txt",'a+')
        self.logger=logger.App_Logger()
        self.feature_selection=fs.Selection()

    def train_validation_data(self):
        self.logger.log(self.file_object,"train_validation_data function is started!!")
        try:
            data1=self.row_data.replace_by_null()
            self.logger.log(self.file_object, "replace_by_null is called")
            data2=self.row_data.drop_more_than_95_missing(data1)
            self.logger.log(self.file_object, "drop_more_than_95 function is called")
            data3=self.data_transform.miss_value_imputation(data2)
            self.logger.log(self.file_object,"Missing Value imputation function is called")
            data3.to_csv("Datasets/validated_data.csv")
            cols=['age','TSH','T3','TT4','T4U','FTI']
            data4=self.data_transform.object_type_into_float(data3,cols)
            data5=self.data_transform.Encoding(data4)
            data5.to_csv("Datasets/Encoded_data1.csv",header=True,index=None)
            thresh=0.112
            X=data5.drop(columns=['binaryClass_P'],axis=1)
            y=data5['binaryClass_P']
            final_data=self.feature_selection.feature_selection(X,thresh)
            final_data['Result'] = y
            final_data.to_csv("Datasets/final_data.csv",header=True,index=None)
            return final_data
        except:
            self.logger.log(self.file_object, "Error Occurred in train validation_data function!!")
        self.file_object.close()
```

validation.py

```
import numpy as np
from  training_validation import *
from Logs import logger
import pandas as pd
import numpy as np
class row_data_validation:
    def __init__(self,path):
        self.path=path
        self.logger=logger.App_Logger()
        # self.file_object=open("training_logs/row_data_validation.txt",'a+')

    def replace_by_null(self):
        f=open("Validation_logs/row_data_validation.txt",'a+')
        self.logger.log(f,"replace_by_null is started!!")
        try:
            self.data=pd.read_csv(self.path)
            print(self.data.head())
            self.df=self.data.replace('?',np.nan)
            self.logger.log(f,"Dataset's ? is replaced by Nan Successfully !!")
            self.df.to_csv("Datasets/Missing_replaced.csv",index=None,header=True)
            return self.df
        except Exception as e:
            self.logger.log(f,"Error is Occurred in replace_by_null function : "+e)
            f.close()
        f.close()

    def drop_more_than_95_missing(self,data):
        f = open("Validation_logs/row_data_validation.txt", 'a+')
        try:
            self.col_to_drop=[col for col in data.columns if data[col].isnull().mean()>0.95]
            self.new_data=data.drop(columns=self.col_to_drop,axis=1)
            self.logger.log(f, "drop_more_than_95_missing function done successfully!!")
            return self.new_data
        except Exception as e:
            self.logger.log(f,"Some Error Occurred in drop_more_than_95_missing function")
            f.close()
        f.close()
```

Result.html

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
<title>result</title>
  </head>
  <body>
  <div class="container">
<!--  navbar  -->
<nav class="navbar navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="images.png">
      <img src="static/thyroid_pic.jpg" alt="" width="50" height="34">
Thyroid Disease Detection Project
    </a>
  </div>
</nav>
<!--    end navbar-->
 <form action="/home" method="POST">
      <div class="modal modal-alert position-static d-block bg-secondary py-5" tabindex="-1"
role="dialog" id="modalChoice">
  <div class="modal-dialog" role="document">
    <div class="modal-content rounded-4 shadow">
      <div class="modal-body p-4 text-center">
        <h5 class="mb-0">{{value}}</h5>
<!--          <p>Accuracy Score : 95.11%</p>-->
        <p class="mb-0" style="color:blue;">_______________________________________</p>
      </div>
      <div class="modal-footer flex-nowrap p-0">
```

```
          <button type="submit" class="btn btn-primary mb-3">Back</button>
        </div>
      </div>
    </div>
  </div>
   </form>

    </div>



      <!-- Optional JavaScript; choose one of the two! -->

      <!-- Option 1: Bootstrap Bundle with Popper -->
      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

      <!-- Option 2: Separate Popper and Bootstrap JS -->
      <!--
      <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
integrity="sha384-
7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
crossorigin="anonymous"></script>
      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
integrity="sha384-
QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
crossorigin="anonymous"></script>
      -->
    </body>
</html>
```

Building_model.py

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from training_validation import train_validation
from Logs import logger
import pickle
class Build:
    def __init__(self,file_path):
        # file_path = "Datasets/hypothyroid.csv"
        self.train_validations=train_validation(file_path)
        self.logger_object=logger.App_Logger()
        self.path=file_path
        self.make_a_model(file_path)

    def X_y(self,data,label_column):
        f = open("model_building_logs/model_building_main.txt", 'a+')
        self.logger_object.log(f, "X_y function is started!!")
        try:
            self.X=data.drop(columns=label_column,axis=1)
            self.y=data[label_column]
            self.logger_object.log(f, "X_y function done successfully")
            return self.X,self.y
        except Exception as e:
            self.logger_object.log(f,"Error Occurred : "+e)
            f.close()
        f.close()
    def train_test_split(self,X,y):
        f=open("model_building_logs/model_building_main.txt",'a+')
        self.logger_object.log(f,"train_test_split function is started!!")
        try:
            self.x_train,self.x_test,self.y_train,self.y_test = 
train_test_split(X,y,test_size=0.3,random_state=100)
            self.logger_object.log(f,"train_test_Split function is done successfully")
            return self.x_train,self.x_test,self.y_train,self.y_test
        except Exception as e:
            self.logger_object.log(f,"Error Occurred "+e)
            f.close()
        f.close()
    def make_a_model(self,file_path):
        f = open("model_building_logs/model_building_main.txt", 'a+')
        self.logger_object.log(f, "make_a_model function is started!!")
        try:
            data=self.train_validations.train_validation_data()
            X,y=self.X_y(data,'Result')
            x_train,x_test,y_train,y_test=self.train_test_split(X,y)
            lr=LogisticRegression()
            lr.fit(x_train,y_train)
            train_time = lr.score(x_train,y_train)
            test_time = lr.score(x_test,y_test)
            pickle.dump(lr,open("Models/logistic_model.pkl",'wb'))
            self.logger_object.log(f,f"model build successfully!! training_time : {train_time} 
Testing_time : {test_time}")
            # return train_time,test_time
        except Exception as e:
            self.logger_object.log(f,"Error Occurred "+e)
            f.close()
```