**1. Introduction**

1.1 Project Overview

The Calorie Burn Prediction System is a web application that provides users with accurate estimations of calorie expenditure based on their physical activities. Users input data manually, including information on activity type, duration, and personal details such as age, weight, and height. The system employs machine learning algorithms to analyze this input and predict calorie burn. The user interface features a user-friendly dashboard displaying real-time and historical calorie burn data, along with personalized recommendations for optimizing fitness goals.  Basically the Calorie Burn Prediction System a valuable tool for individuals seeking insight into their fitness activities within the simplicity of a web app.

1.2 Purpose

The Calorie Prediction feature is designed to provide users with personalized insights into their calorie expenditure during workouts. By considering individual characteristics such as gender, age, height, exercise duration, heart rate, and body temperature, this feature aims to estimate the calories burnt during a workout session. This information not only helps users track their progress but also allows for better optimization of their fitness routines. Here are some of the key advantages: 1. Personalized Fitness Guidance: By predicting calories burnt during different activities, machine learning models can provide personalized fitness recommendations. Users can receive tailored exercise plans and activity suggestions based on their goals, fitness levels, and preferences. 2. Optimized Workouts: Calorie prediction can help individuals optimize their workouts by selecting activities that align with their desired calorie burn. This can lead to more efficient and effective exercise routines, preventing wasted time on activities that might not align with their goals.

**2. LITERATURE SURVEY**

2.1 Existing Problem

The existing problem in the realm of fitness and calorie management lies in the lack of personalized and accurate methods to anticipate caloric expenditure during different activities. Traditional approaches often rely on generalized estimations, which might not account for individual variations in metabolism, fitness levels, and activity preferences. Moreover, the manual tracking of calories burned can be cumbersome and prone to inaccuracies. This gap hinders the ability to provide tailored fitness guidance, optimize workouts, and effectively manage weight based on real-time and personalized data.

2.2 References

* [Challenges faced by the modern fitness industry in India](https://timesofindia.indiatimes.com/blogs/voices/the-challenges-faced-by-the-modern-fitness-industry-in-india-and-the-side-effects/) - TOI
* [Exercise, metabolism, and weight: New research from The Biggest Loser](https://www.health.harvard.edu/blog/exercise-metabolism-and-weight-new-research-from-the-biggest-loser-202201272676) - Harvard Health

2.3 Problem Statement Definition

The problem at hand is to develop an end-to-end machine learning solution capable of predicting the number of calories burnt during various activities. This involves the extraction of relevant features from input data,with factors such as ***input relevant features here***  The objective is to create a robust model that considers the intricacies of individual metabolism and fitness levels.

The problem statement includes:

- Designing and implementing machine learning algorithms to extract pertinent features from input data.

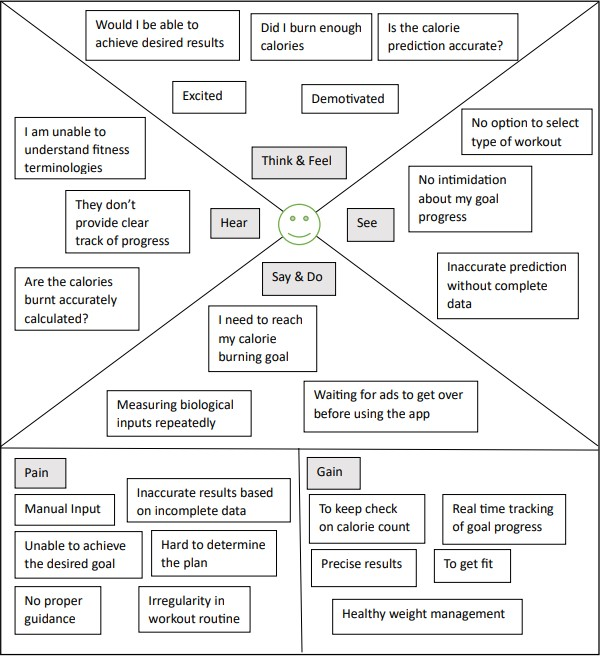
- Developing a model capable of accurately predicting calories burned based on the extracted features.

- Creating an intuitive web application that displays these predictions for the user while also saving the static user details in the database for seamless input of data, also saving the records of each of the user’s activities in the database

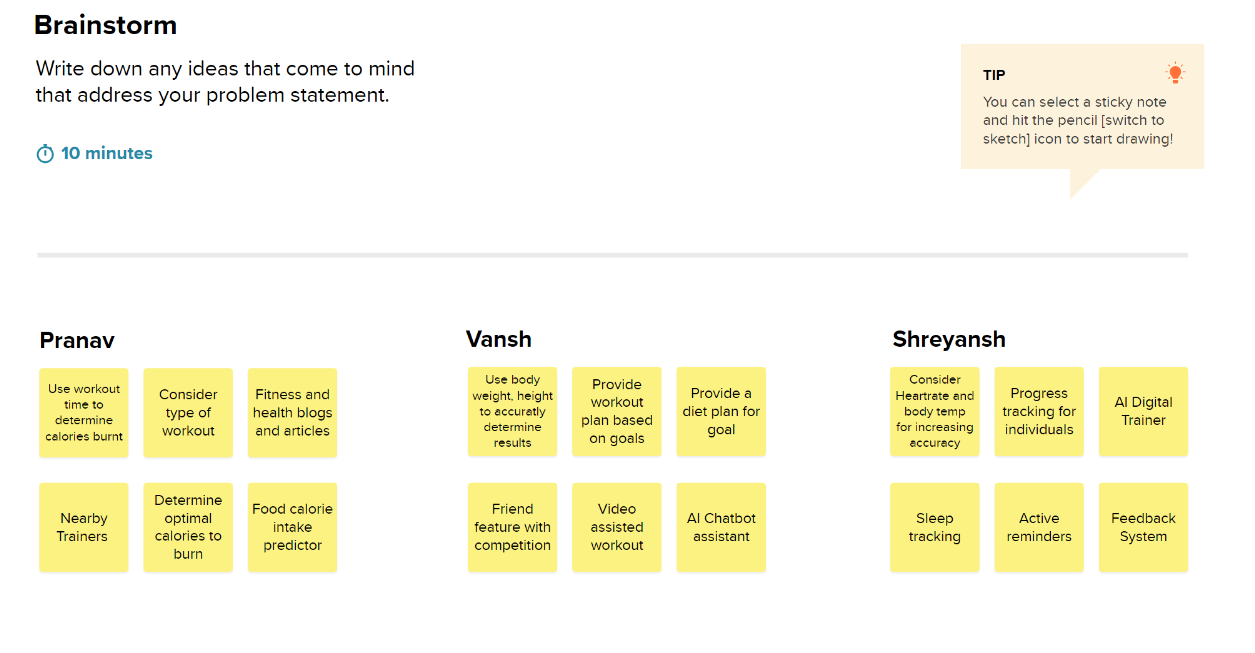
By achieving these objectives, the project aims to improve the way individuals approach fitness by providing a holistic and personalized approach to calorie management through machine learning.

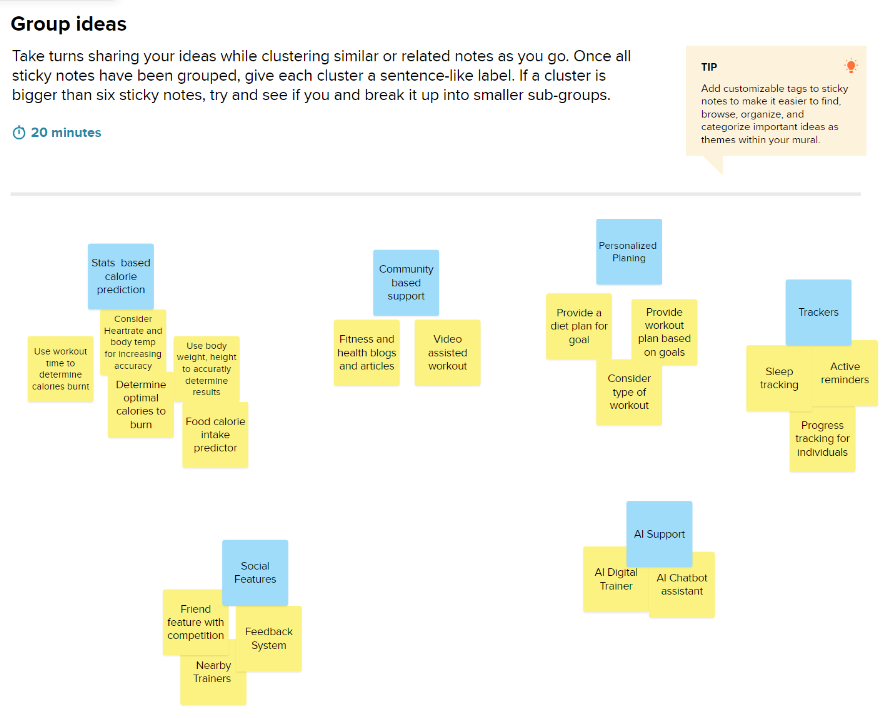
**3. IDEATION AND PROPOSED SOLUTION**

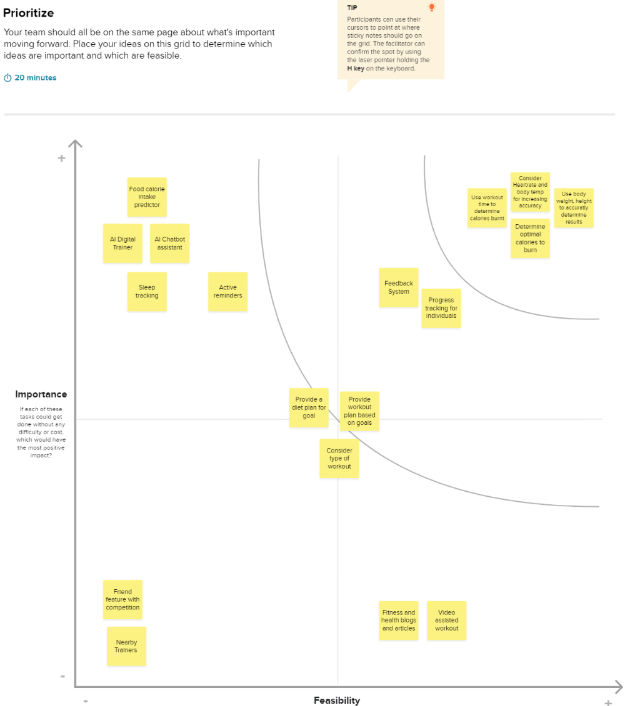
3.1 Empathy Map Canvas



3.2 Ideation and Brainstorming







**4. REQUIREMENT ANALYSIS**

4.1 Functional Requirements

1. User Authentication and Authorization

=> Users should be able to create accounts and log in securely.

=>Different user roles (e.g. guest user) with appropriate access levels.

1. Input Data Handling

=> Users should be able to input data through means such as manual entry.

=> System should accept input data related to height, bmi, weight etc. of the user.

1. Calorie Prediction Algorithm

=> An accurate calorie prediction algorithm based on information provided by the user.

=> The algorithm should consider factors like the user's height, weight, bmi etc.

1. Activity Tracking Database

=> A database keeping track of workouts performed by the user, calories burned etc. should be maintained/

=> The database should be regularly updated to include new workouts performed by the user, user’s current weight etc.

1. Integration with wearable devices

=> If applicable, integrate with fitness trackers or wearable devices to enhance accuracy.

4.2 Non-Functional Requirements

1. Performance

=> The system should respond to user inputs promptly.

=> It should be capable of handling a reasonable number of simultaneous users.

1. Accuracy

=> The calorie prediction algorithm should have a high level of accuracy.

=> Minimize error in database entries.

1. Scalability

=> The system should be scalable to accommodate an increasing number of users and data.

1. Security

=> Implement secure user authentication and data encryption.

=> Protect user data from unauthorized access.

1. Reliability

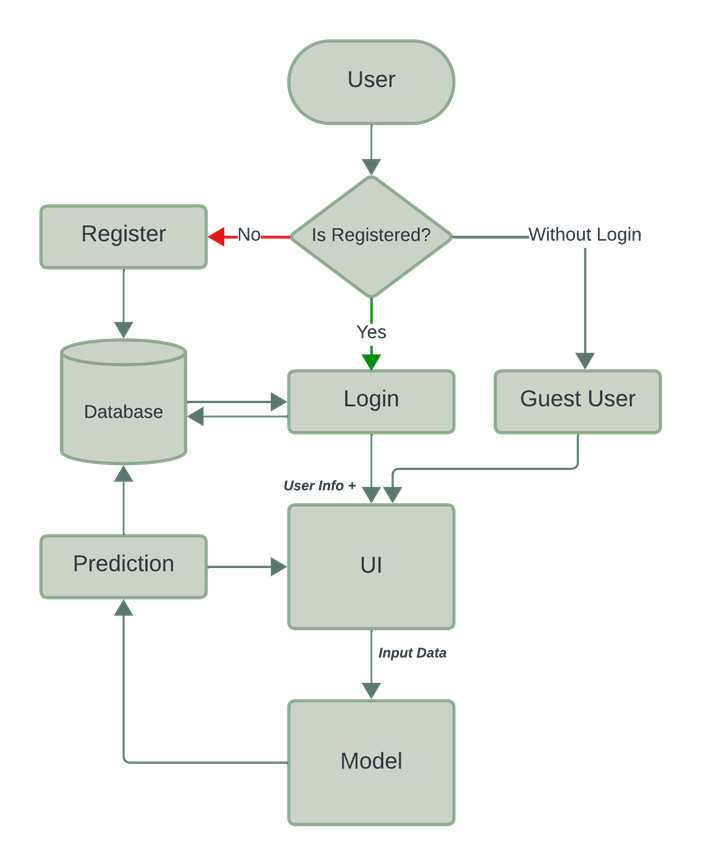
=> The system should be reliable and available for users when needed.

=> Implement backup and recovery mechanisms.

**5. PROJECT DESIGN**

5.1 Data Flow Diagrams and User Stories

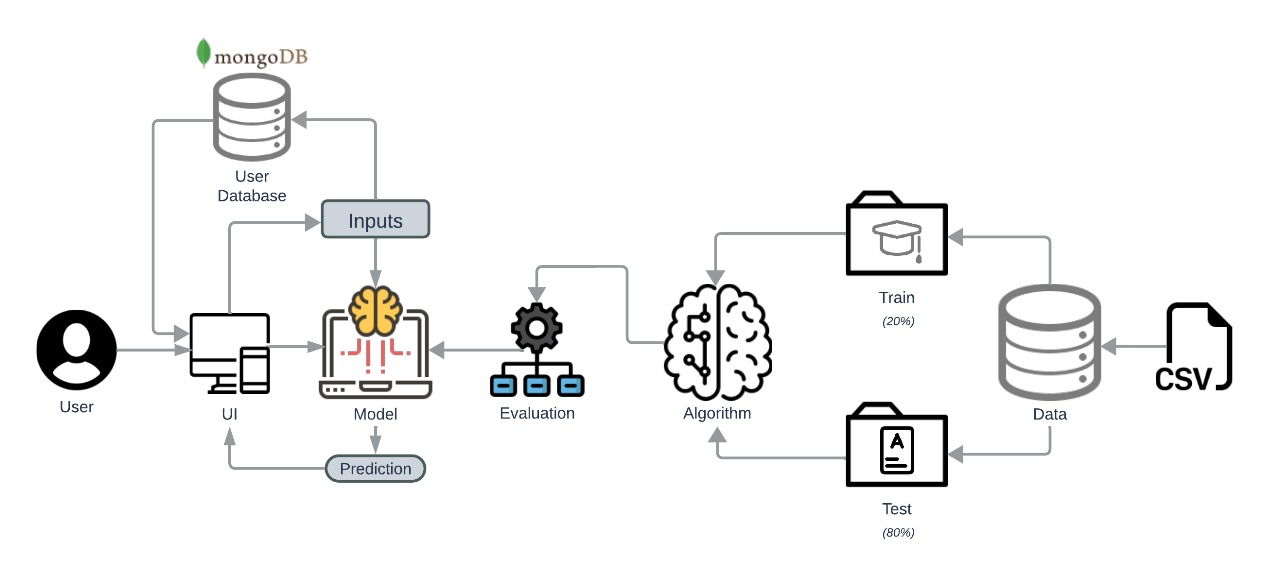
=> A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right  amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is  stored.



**User Stories**

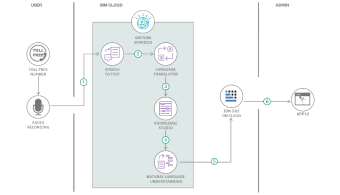
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Type** | **Functional**  **Requirement**  **(Epic)** | **User Story  Number** | **User Story / Task** | **Acceptance criteria** | **Priority** | **Release** |
| Customer | Registration | USN-1 | As a user, I can register for the website by  entering my email-id, password, and confirming  my password. | I can access my account /  dashboard | High | Sprint-1 |
|  | Login | USN-2 | As a user, I can log into the website by  entering my email-id and password |  | High | Sprint-1 |
|  | Dashboard | USN 3 | As a user, I can track my goal progress | I can see  my previous workout activities | Medium | Sprint-2 |
|  |  | USN-4 | As a user, I can input and update my health data | I can update my user data anytime | High | Sprint-2 |
|  | Calculator | USN-5 | As a user, I can enter my body/exercise details | I can get the calories burnt | High | Sprint-3 |

5.2 Solution Architecture



**6. PROJECT PLANNING AND SCHEDULING**

6.1 Technical Architecture



**Table-1 : Components & Technologies:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Component** | **Description** | **Technology** |
| 1. | User Interface | How user interacts with application | HTML, CSS, JavaScript |
| 2. | Application Logic-1 | Logic for a process in the application - Login/Register | Python(Flask) |
| 3. | Database | User info, Calorie info etc. | SQL, CSV |
| 4. | Machine Learning Model | Prediction of the calories burnt | XGB Regressor etc. |
| 5. | Infrastructure (Server / Cloud) | Application Deployment on Cloud Server /  Cloud Server Configuration : |  |

6.2 Sprint Planning & Estimation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional**  **Requirement (Epic)** | **User Story**  **Number** | **User Story / Task** | **Story Points** | **Priority** | **Team**  **Members** |
| Sprint-1 | Registration | USN-1 | As a user, I can register for the application by  entering my email, password, and confirming  my password. | 2 | High | 3 |
| Sprint-1 | Login | USN-2 | As a user, I can log into the application by  entering email & password | 1 | High | 3 |
| Sprint-2 | Dashboard | USN-3 | As a user, I can track my progress | 2 | Medium | 3 |
| Sprint-2 |  | USN-4 | As a user, I can input and update my health data | 2 | High | 3 |
| Sprint-3 | Calculator | USN-5 | As a user, I can enter the body/exercise details | 1 | High | 3 |

6.3 Sprint Delivery Schedule

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Total Story**  **Points** | **Duration** | **Sprint Start Date** | **Sprint End Date**  **(Planned)** | **Story Points**  **Completed (as on  Planned End Date)** | **Sprint Release Date**  **(Actual)** |
| Sprint-1 | 20 | 6 Days | 1 Nov 2023 | 6 Nov 2023 | 3 | 6 Nov 2023 |
| Sprint-2 | 20 | 6 Days | 7 Nov 2023 | 12 Nov 2023 | 7 | 12 Nov 2023 |
| Sprint-3 | 20 | 6  Days | 13 Nov 2023 | 18 Nov 2023 | 8 | 16 Nov 2023 |

**7. CODING AND SOLUTIONING**

7.1 & 7.2 Features

**Introduction**

The integration of User Registration and Personalized Dashboard functionality, coupled with the innovative Calorie Prediction feature, elevates the user experience within the application. These features collectively provide a seamless and personalized journey for users, fostering engagement and motivation in their fitness pursuits, registration offers an enhanced, customized experience tailored to their fitness journey.

**Key Attributes**

**1.User Registration and Profile Management**

**1.1 Sign-Up Process**

Users can easily register by providing essential details, ensuring a quick and hassle-free onboarding experience.

**1.2 Profile Management**

Seamless profile editing allows users to update information such as weight, height, and other relevant details effortlessly.

Changes made in the profile settings are promptly reflected in the personalized dashboard.

**1.3 Engagement and Commitment**

Personalized dashboards encourage consistent engagement, instilling a sense of ownership in users regarding their fitness journey.

**2. Personalized Dashboard**

**2.1 Overview**

Post-registration and login, users are directed to a dedicated dashboard offering a central hub for tracking and visualizing workout-related data.

**2.2 Workout Statistics**

Users gain insights into their workout history, including exercise duration, total calories burned, average BPM, and more.

Graphical representations provide an intuitive understanding of progress over time.

**3. Calorie Prediction Feature**

**3.1 User Input Parameters**

Users input fundamental details like gender, age, height, exercise duration, heart rate, and body temperature for accurate calorie predictions.

**3.2 Calorie Prediction Algorithm**

An advanced algorithm processes user-input data, considering physiological factors and exercise intensity for precise calorie estimations.

**3.3 Personalized Calorie Insights**

Users receive a detailed breakdown of estimated calories burnt, with graphical representations showcasing trends over time.

Goal alignment highlights whether users are meeting, exceeding, or falling short of targeted calorie burn.

**3.4 Benefits**

Precision in Fitness Planning: Users can plan workouts effectively by understanding the calorie impact of different exercises and durations.

Motivation Through Data: Real-time feedback on calorie expenditure motivates users to push themselves during workouts.

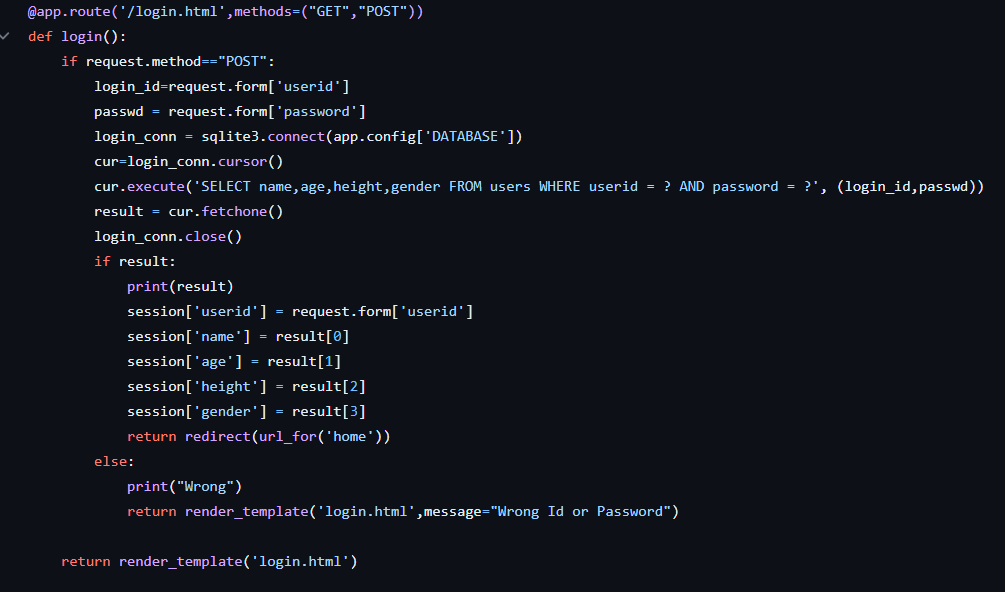
Personalized Optimization: Users can fine-tune workouts based on personalized data, optimizing calorie-burning potential.

**Conclusion**

The combined synergy of User Registration, Personalized Dashboards, and the innovative Calorie Prediction feature amplifies the user experience. By providing tailored insights, robust tracking capabilities, and personalized predictions, these features empower users on their fitness journey and foster sustained engagement with the application.

**Code:**

login:



The login() function begins by checking if the request method is POST. This implies that the user has submitted the input. The function then extracts the 'userid' and 'password' from the input data using request.form. These values come from the user entering their login credentials in the login page

The application establishes a connection to a SQLite database to manage user data. The database connection is assumed to be stored in the Flask application's configuration under the key 'DATABASE'. This connection is vital for retrieving user information and ensuring secure authentication. The authentication process begins with the creation of a cursor (cur) to interact with the database. A SQL SELECT query is executed to retrieve essential user information, including name, age, height, and gender, based on the provided 'userid' and 'password'.The app connects to the SQLite database using the configuration stored under 'DATABASE'.A SQL SELECT query is executed to fetch user information based on the provided 'userid' and 'password'. The result of the query is obtained using cur.fetchone().If a matching user is found in the database, the retrieved information is stored in the Flask session object. Subsequently, the user is redirected to the 'home' page.If no matching user is found, an error message is displayed on the login page, indicating an incorrect ID or password.

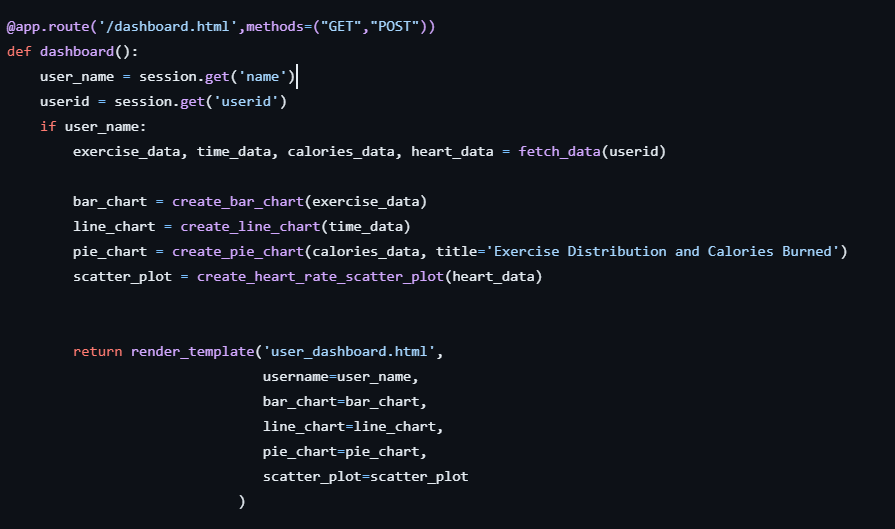
Register:



The function initiates by verifying the incoming request method to determine if it is a POST request ie. the app has received the necessary inputs.In the case of a POST request, the function retrieves form data from the request using request.form. This includes fields such as 'userid', 'password', 'name', 'age', 'height', and 'gender'.To interact with user data, the function establishes a connection to a SQLite database, just like in login function. It assumes that the database connection is stored in the Flask application's configuration under the key 'DATABASE'.The function executes a SELECT query to verify whether the provided 'userid' already exists in the database. If a matching user is found, an error message is returned, indicating that the user already exists.

In the scenario where the user does not exist, the function proceeds to insert a new user into the 'users' table, incorporating the provided information (userid, password, name, age, height, gender). Following the insertion, it commits the changes to the database.Upon successful registration, the user's information is stored in the Flask session object, and the user is redirected to the 'home' route using redirect(url\_for('home')).If the request method is not POST (i.e., it is a GET request), the function renders the 'register.html' template. This is typically done when the user initially accesses the registration page to complete the registration form.

Dashboard:

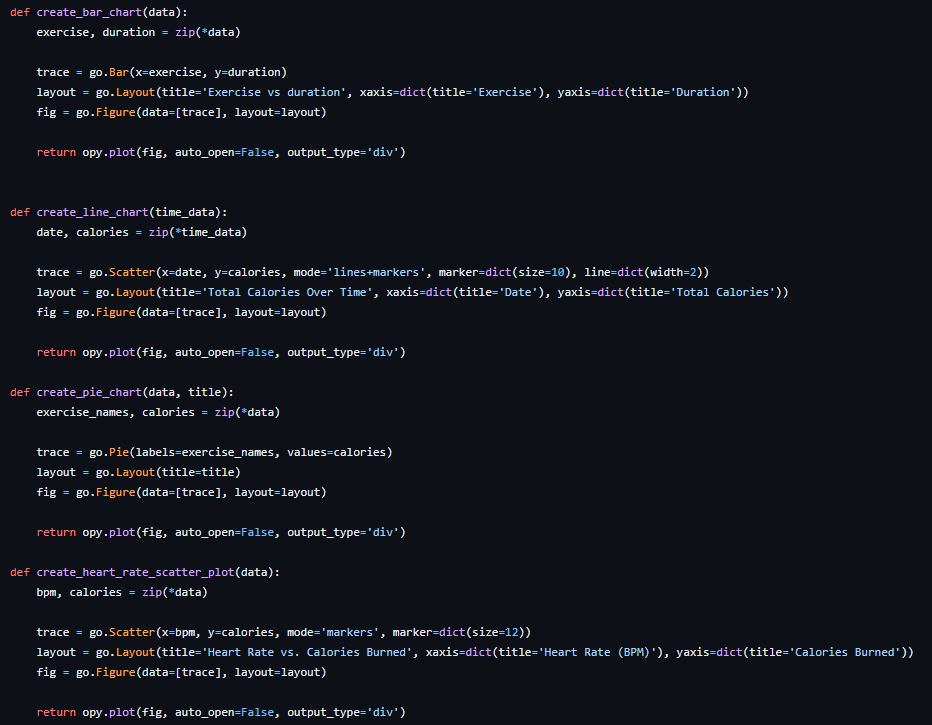


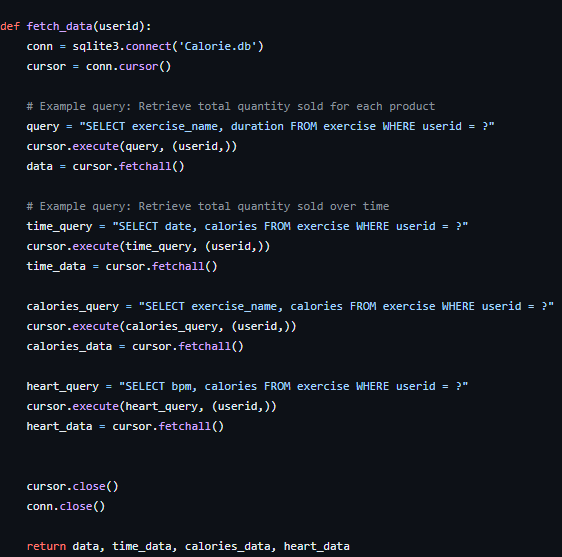
To populate the user dashboard, the function retrieves essential user information from the session object. This is accomplished using session.get('name') and session.get('userid'). It is important to note that this is accessible only if the user is a registered user. The function calls the fetch\_data(userid) function to obtain exercise data, time data, calories data, and heart rate data. The next step involves the creation of diverse charts to visually represent the user's fitness-related information.

Upon retrieving user information and creating charts, the function renders the 'user\_dashboard.html' template. This template includes HTML and Jinja2 code to dynamically display the user's name, along with the charts and other dashboard-related information.

This associated template serves as the frontend counterpart, responsible for visually presenting the user dashboard. It incorporates dynamic rendering using Jinja2 to display the user's name and the generated charts.

The functions used in the above code are also defined, and are imported in our main flask app.





Calorie prediction:



The prediction function is designed to handle a POST request, taking a request (req) as its parameter, which contains input data. The relevant features, including gender, age, height, duration, heart rate, and body temperature, are extracted from the request.

The function loads a pre-trained machine learning model from a file named 'final\_model.pkl' using the pickle.load method. The extracted feature values are organized into a list (data), which is then used to make a prediction using the loaded model (model.predict(data)). The result is rounded to two decimal places and returned. The main route, which handles both GET and POST requests at the root URL ('/'), is responsible for orchestrating the prediction process.

If the request method is POST, the route calls the prediction function with the form data (request.form), obtains the prediction result, and renders the 'index.html' template with the result displayed.

If the request method is GET, the route renders the 'index.html' template without displaying any prediction result.



This function is used to display the calories burnt by the user. Like other functions, it checks whether the incoming request method is POST. If it is, it assumes that a form has been submitted.

If it's a POST request, the function calls the prediction function with the form data (request.form) to obtain a prediction result. The result is stored in the session under the key 'calories'.

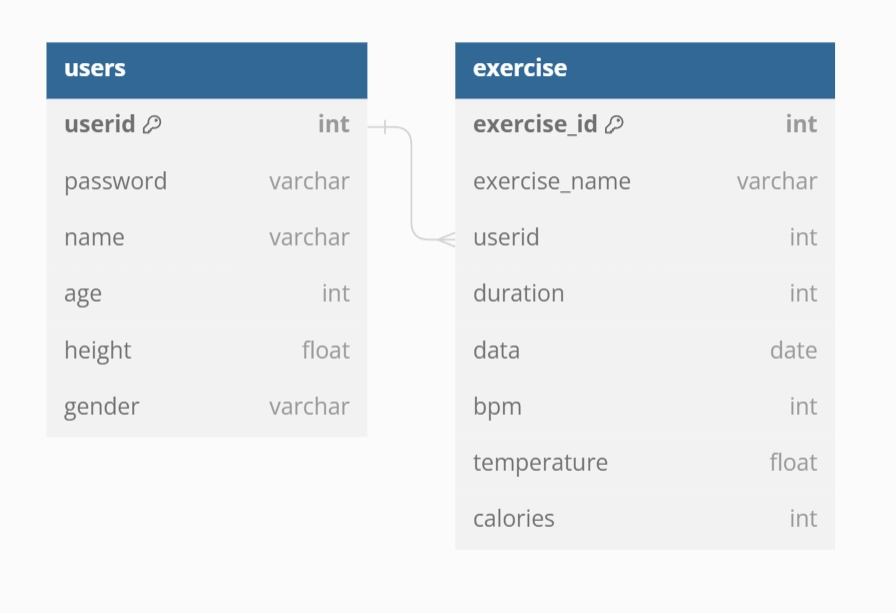
It establishes a connection to a SQLite database and inserts exercise-related data into the 'exercise' table. The data includes exercise name, userid, duration, date, heart rate (bpm), temperature, and calories burned.

The database changes are committed, and the connection is closed.

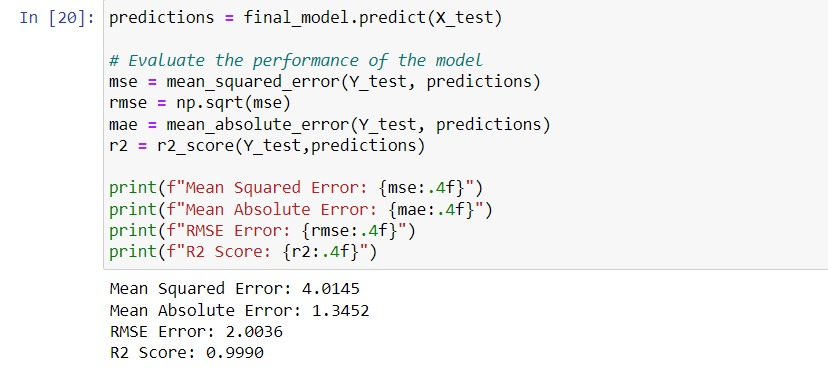
The function renders the 'home.html' template, which is used to display the predicted value of the calories burnt, passing the calculated calories as a variable if it's a POST request.

If it's a GET request, the function renders the 'home.html' template, displaying user information such as name, age, height, and gender.

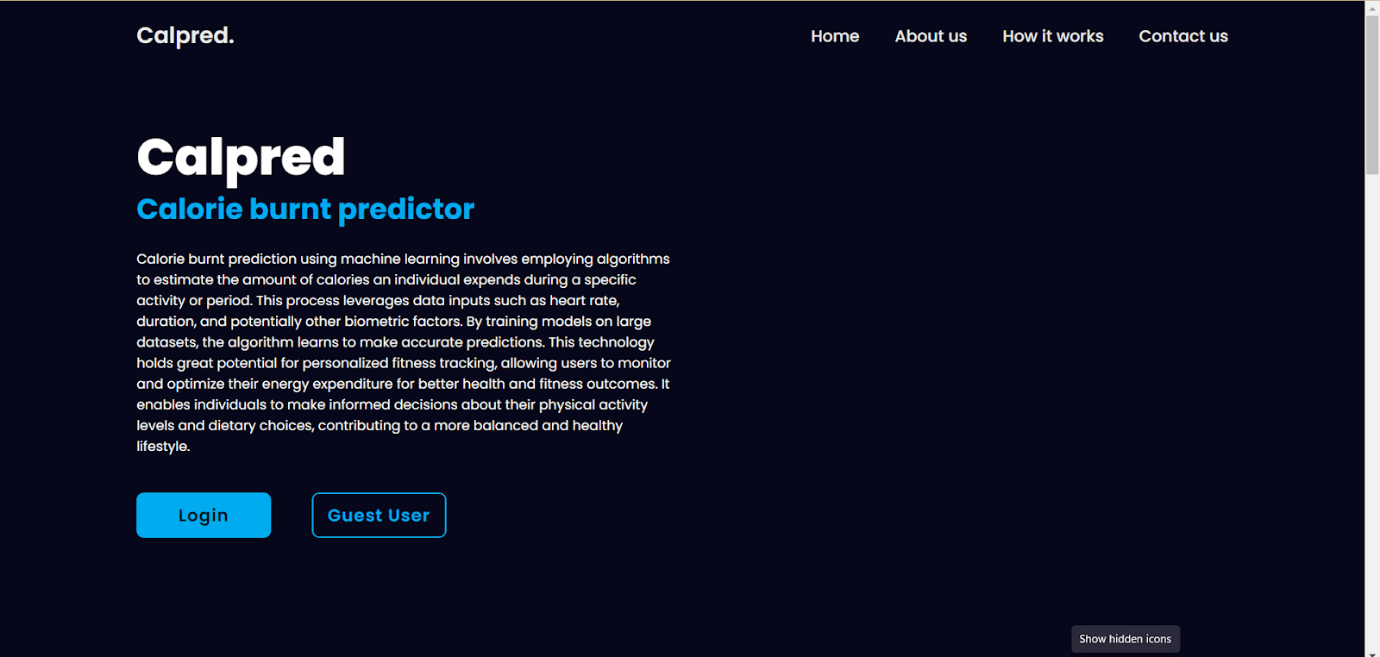
7.3 Database Schema

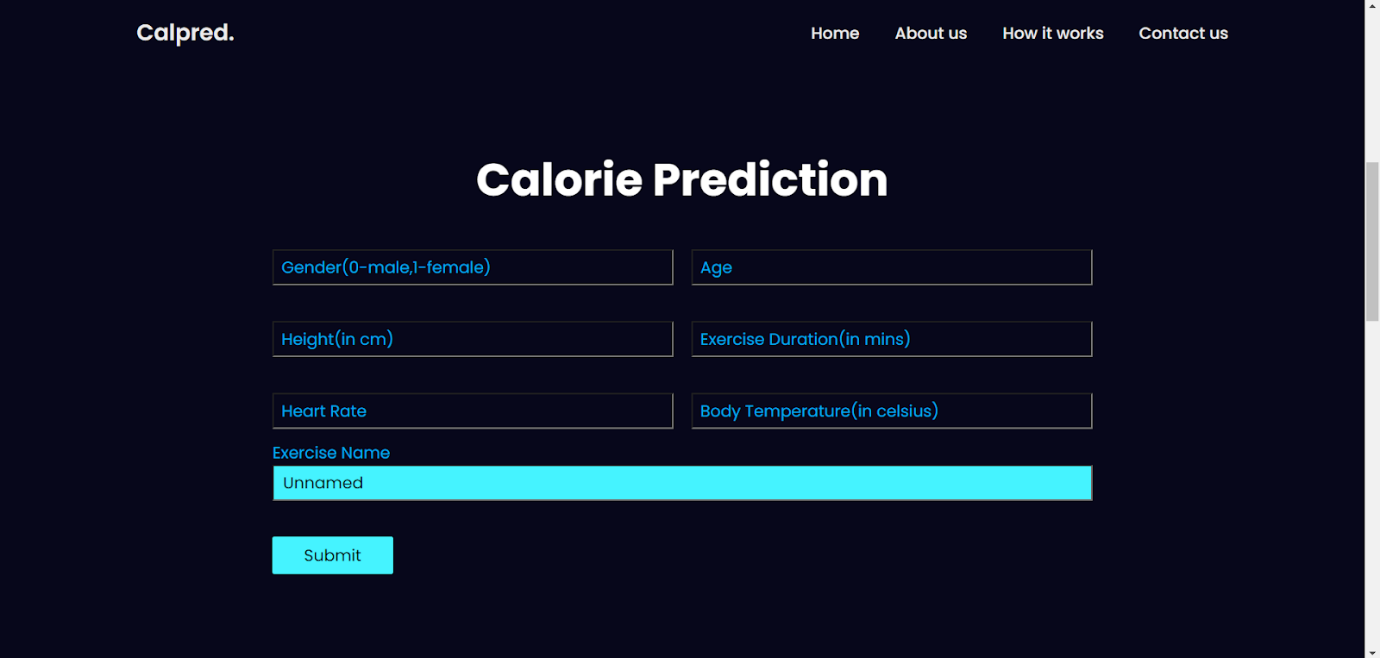


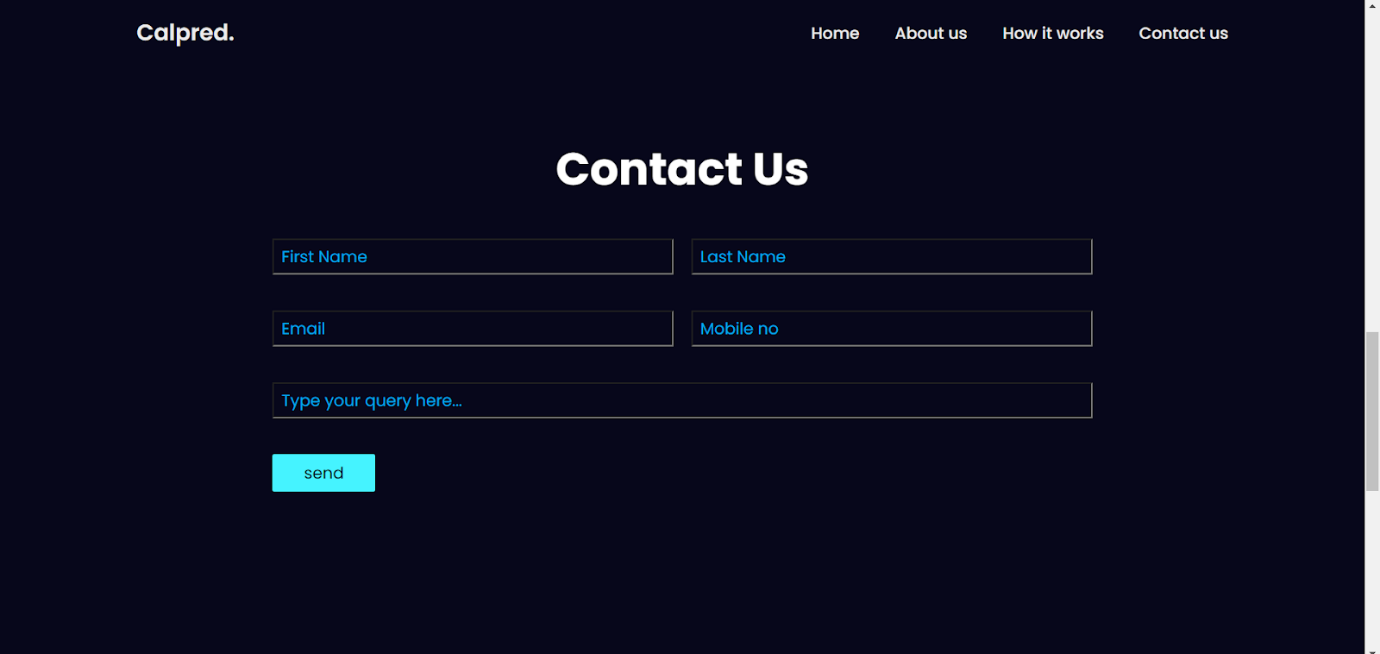
**8. PERFORMANCE TESTING**

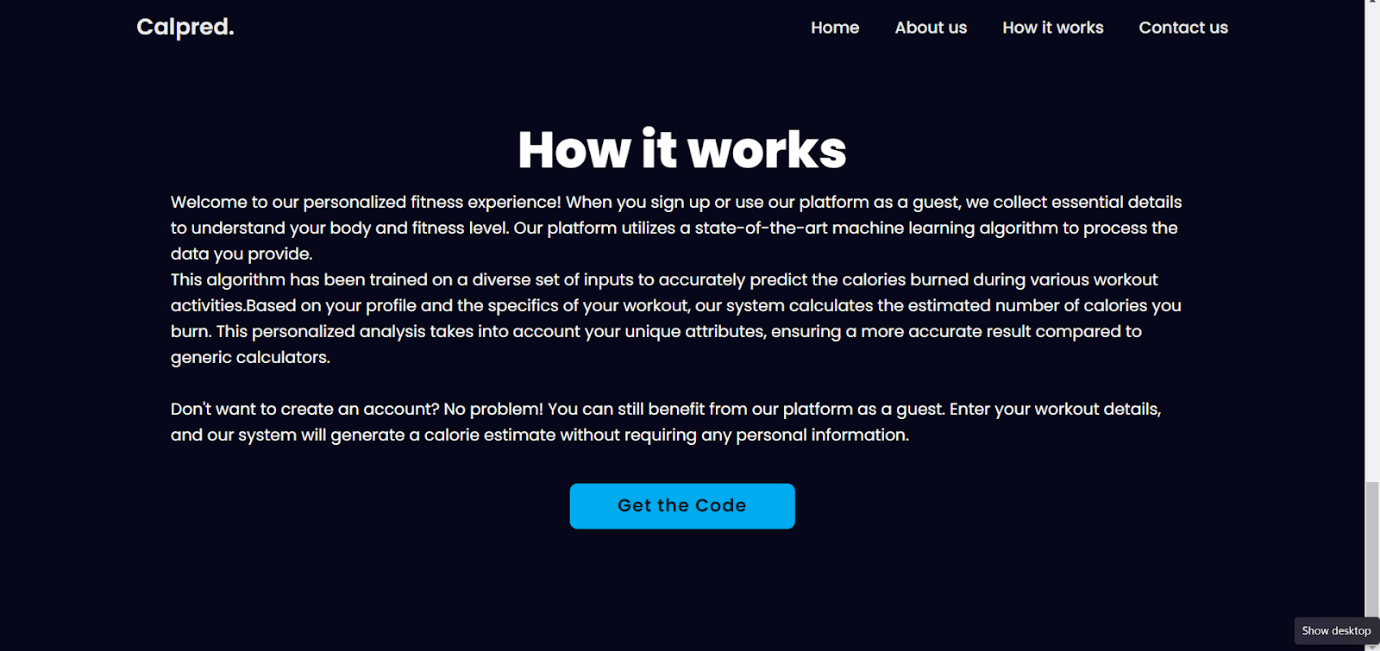


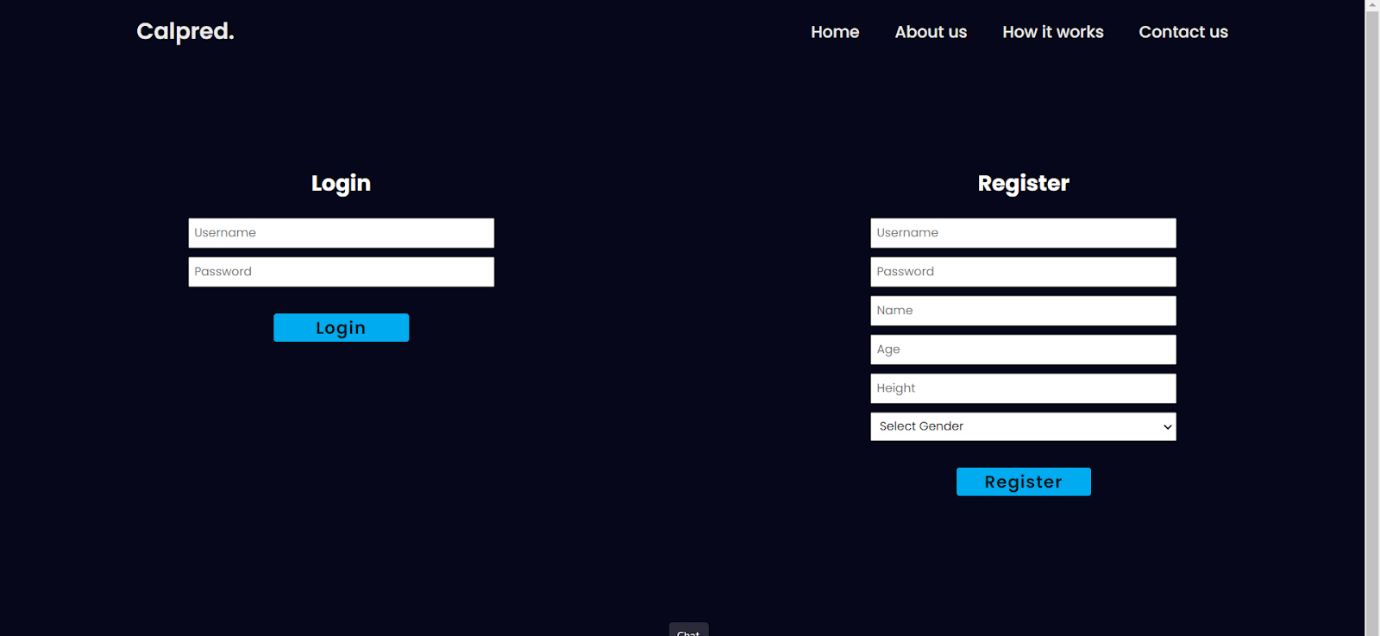
**9. RESULTS**

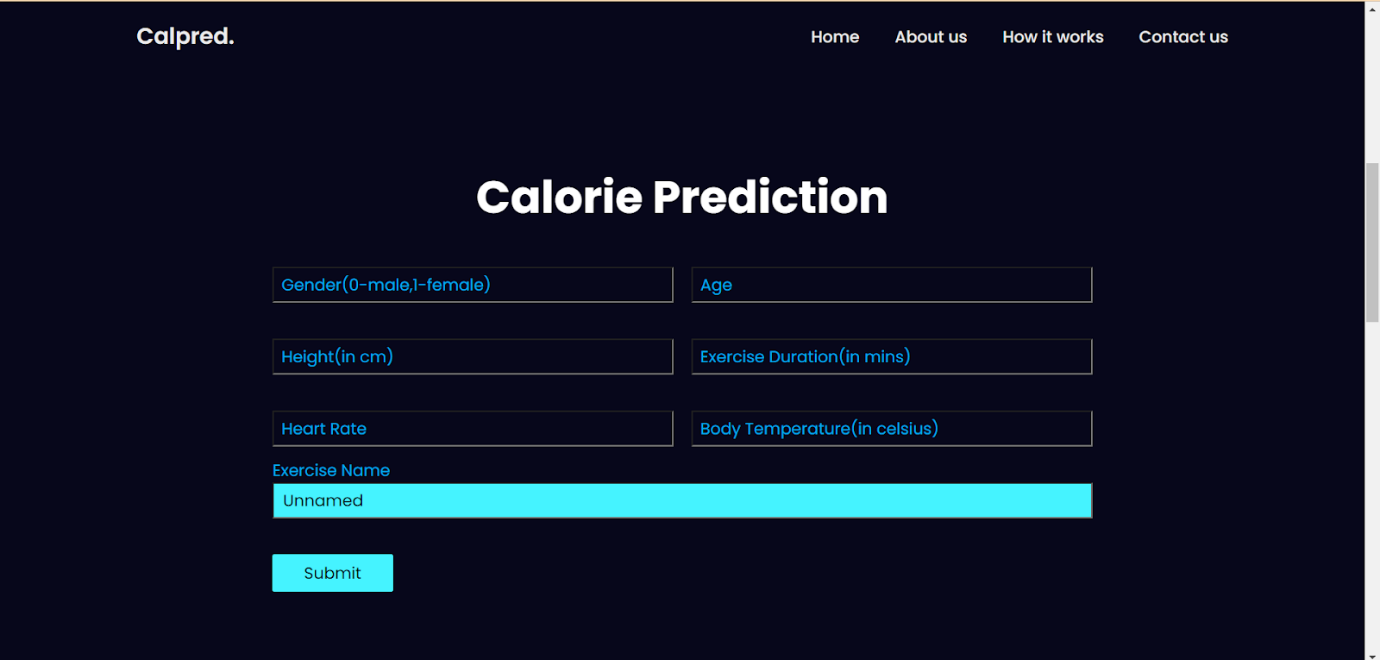


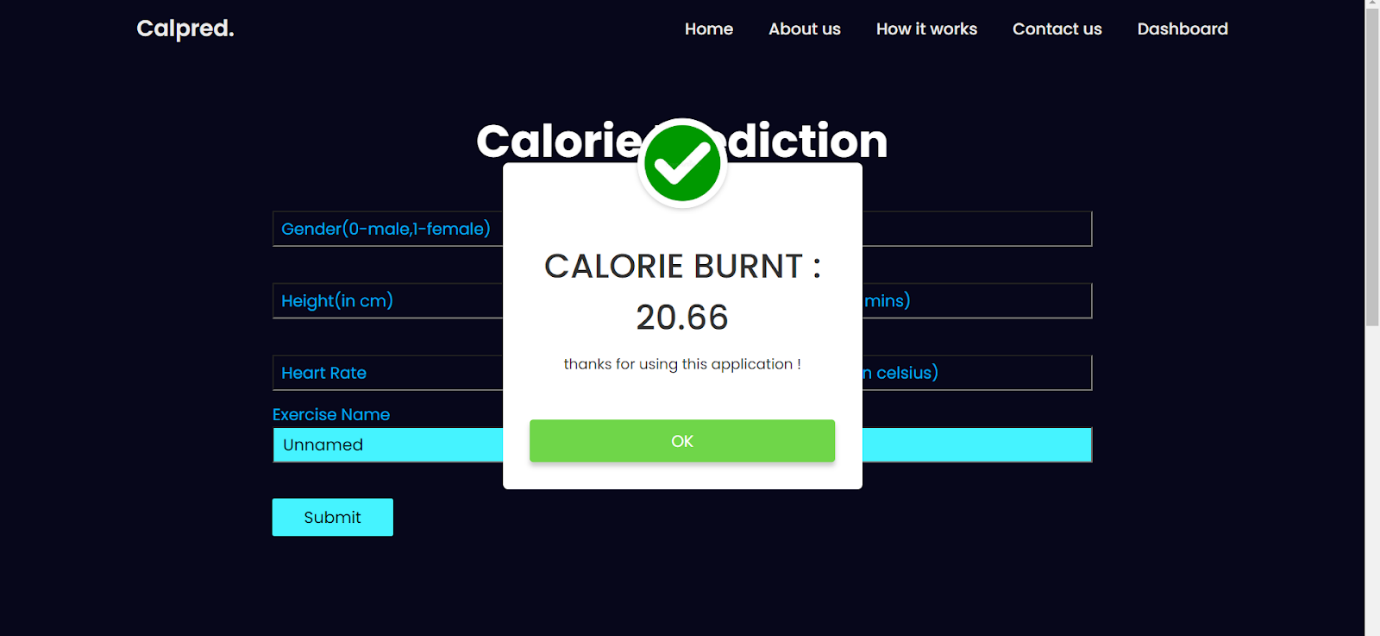


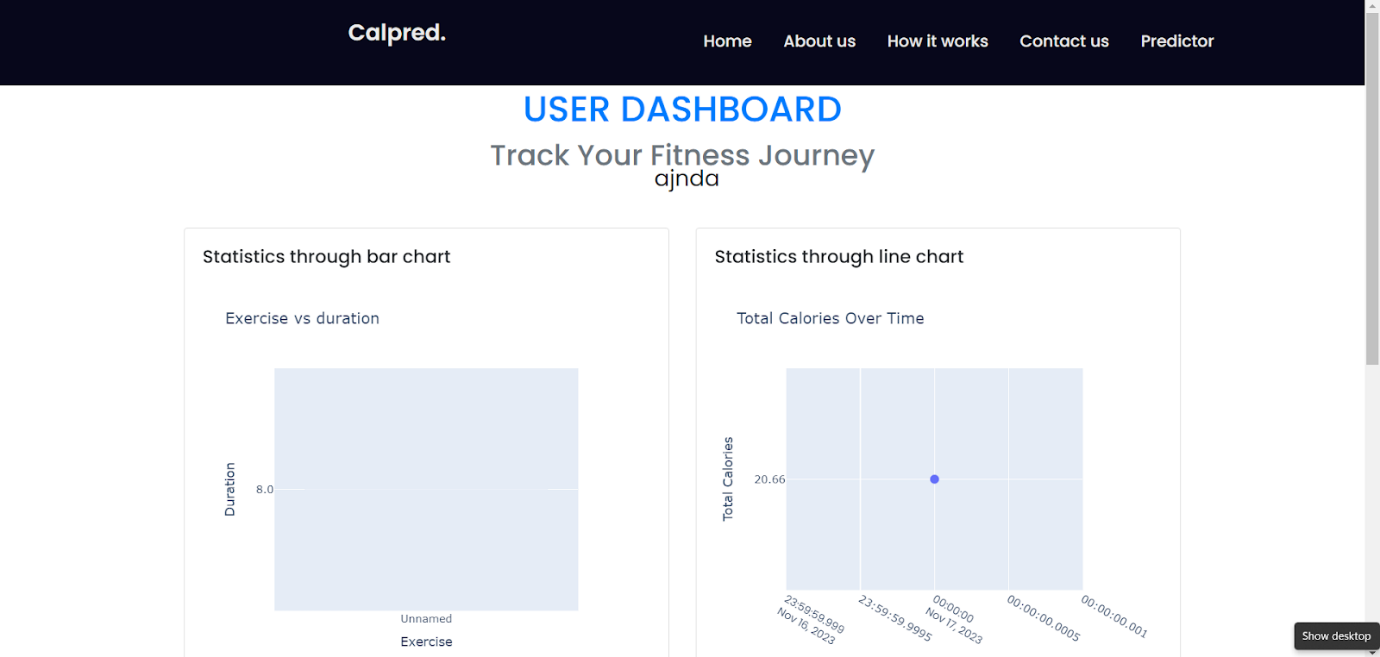












**10. ADVANTAGES & DISADVANTAGES**

* Advantages:

1. High Accuracy: XGBoost is known for its high predictive accuracy. It can capture complex relationships in the data and provide more accurate predictions compared to traditional regression models.
2. Parallel Processing: XGBoost is designed for parallel and distributed computing, making it efficient for training on large datasets.
3. Versatility: XGBoost can be applied to a wide range of regression problems, and its versatility makes it suitable for different types of data.
4. Real-time Feedback: Machine learning models can provide real-time feedback on caloric expenditure, helping individuals make instant adjustments to their activities or dietary habits. This can enhance the effectiveness of weight management and fitness programs.

* Disadvantages:

1. Risk of Overfitting: Without proper hyperparameter tuning and regularization, XGBoost models can be prone to overfitting, especially when the dataset is small.
2. Need for Sufficient Data: XGBoost tends to perform well with a large amount of data. If the dataset is small, there is a risk that the model may not generalize well to unseen data.
3. Interpretability Challenges: Despite providing feature importance scores, interpreting the exact relationships and decision-making processes within an XGBoost model can be challenging, which might be a concern in applications where interpretability is crucial.
4. Data Privacy Concerns: Caloric expenditure prediction often requires collecting and analyzing personal health and activity data. This raises concerns about privacy, especially if the data falls into the wrong hands or is not adequately secured.

**11. CONCLUSION**

This study set out to find out how many calories our bodies burn, a function of several factors including age, gender, height, weight, duration, and heart rate. Understanding our calorie intake is essential to maintaining our health and fitness. The number of calories burned can be predicted using regression techniques such as Linear Regression, XG Boost Regression, and Random Forest. Out of all of these algorithms, Extreme Gradient Boosting, or XG Boost, regression yields the most accurate result. The Mean Absolute Error (MAE) value of 1.68 for the XG Boost suggests that there were comparatively few errors. Therefore, the XGB Regressor algorithm is currently the most effective way to predict the number of calories burned. With modifications, the suggested technique's adaptability was enhanced. This enables us to create a website that is fully functional and has a user interface (UI) that works well. This is required so that users can enter their values and get results that show how many calories they have burned in addition to other features like progress tracking.

**12. FUTURE SCOPE**

Key features from the feature selection and evaluation process were also related to the problem domain using the XGBRegressor Model, offering implications and insights for possible applications. Although, the small dataset size and potential for overfitting were two of the study's shortcomings. Subsequent investigations may tackle these constraints and enhance the efficacy of the models and feature selection methodologies.

**13. APPENDIX**

Source Code:

init.py

import sqlite3

conn = sqlite3.connect('Calorie.db')

cur = conn.cursor()

cur.execute('''create table if not exists users (

            userid text primary key,

            password text,

            name text,

            age text,

            height text,

            gender text)''')

cur.execute('''create table if not exists exercise (

            exercise\_id text primary key,

            exercise\_name text,

            userid text,

            duration text,

            date text,

            bpm text,

            temperature text,

            calories text)''')

conn.commit()

conn.close()

app.py

from flask import Flask, render\_template, request, redirect, url\_for, flash, session

import sqlite3

import os

from dashboard import fetch\_data, create\_bar\_chart, create\_line\_chart, create\_pie\_chart, create\_heart\_rate\_scatter\_plot

import pickle

from datetime import datetime

app = Flask(\_\_name\_\_)

app.config['DATABASE'] = 'Calorie.db'

app.secret\_key = 'TH15\_1S\_@\_S3CR3T\_K3Y'

def prediction(req):

    print(req)

    Gender=int(req['gender'])

    Age = float(req['age'])

    Height = float(req['height'])

    Duration=float(req['duration'])

    Heart\_Rate = float(req['heart\_rate'])

    Body\_Temp = float(req['temperature'])

    model=pickle.load(open('final\_model.pkl','rb'))

    data=[[Gender,Age,Height,Duration,Heart\_Rate,Body\_Temp]]

    print(data)

    result = model.predict(data)

    result = round(float(result),2)

    return result

@app.route('/',methods=("GET","POST"))

def main():

    if request.method=="POST":

        result=prediction(request.form)

        return render\_template('index.html',calories = result)

    return render\_template('index.html')

@app.route('/login.html',methods=("GET","POST"))

def login():

    if request.method=="POST":

        login\_id=request.form['userid']

        passwd = request.form['password']

        login\_conn = sqlite3.connect(app.config['DATABASE'])

        cur=login\_conn.cursor()

        cur.execute('SELECT name,age,height,gender FROM users WHERE userid = ? AND password = ?', (login\_id,passwd))

        result = cur.fetchone()

        login\_conn.close()

        if result:

            print(result)

            session['userid'] = request.form['userid']

            session['name'] = result[0]

            session['age'] = result[1]

            session['height'] = result[2]

            session['gender'] = result[3]

            return redirect(url\_for('home'))

        else:

            print("Wrong")

            return render\_template('login.html',message="Wrong Id or Password")

    return render\_template('login.html')

@app.route('/register.html',methods=("GET","POST"))

def register():

    if request.method=="POST":

        user\_id=request.form['userid']

        passwd = request.form['password']

        name = request.form['name']

        age = request.form['age']

        height = request.form['height']

        gender = request.form['gender']

        register\_conn = sqlite3.connect(app.config['DATABASE'])

        register\_cur=register\_conn.cursor()

        register\_cur.execute('SELECT userid FROM users WHERE userid = ?', (user\_id,))

        result = register\_cur.fetchone()

        if result:

            register\_conn.close()

            return render\_template('register.html',message="User already exists")

        else:

            register\_cur.execute('insert into users values(?,?,?,?,?,?)',(user\_id,passwd,name,age,height,gender))

            register\_conn.commit()

            register\_conn.close()

            session['userid'] = request.form['userid']

            session['name'] = request.form['name']

            session['age'] = request.form['age']

            session['height'] = request.form['height']

            session['gender'] = request.form['gender']

            return redirect(url\_for('home'))

    return render\_template('register.html')

@app.route('/home.html',methods=("GET","POST"))

def home():

    if request.method=="POST":

        result = prediction(request.form)

        session['calories'] = result

        exercise\_conn = sqlite3.connect(app.config['DATABASE'])

        exercise\_cur=exercise\_conn.cursor()

        exercise\_cur.execute('insert into exercise(exercise\_name,userid,duration,date,bpm,temperature,calories) values(?,?,?,?,?,?,?)',

                             (request.form['exercise\_name'],

                              session['userid'],

                              float(request.form['duration']),

                              datetime.today().date(),

                              float(request.form['heart\_rate']),

                              float(request.form['temperature']),

                              result))

        exercise\_conn.commit()

        exercise\_conn.close()

        return render\_template('home.html',calories = result)

    return render\_template('home.html',name = session['name'], age = session['age'], height = session['height'], gender = 0 if session['gender']=='Male' else 1)

@app.route('/dashboard.html',methods=("GET","POST"))

def dashboard():

    user\_name = session.get('name')

    userid = session.get('userid')

    if user\_name:

        exercise\_data, time\_data, calories\_data, heart\_data = fetch\_data(userid)

        bar\_chart = create\_bar\_chart(exercise\_data)

        line\_chart = create\_line\_chart(time\_data)

        pie\_chart = create\_pie\_chart(calories\_data, title='Exercise Distribution and Calories Burned')

        scatter\_plot = create\_heart\_rate\_scatter\_plot(heart\_data)

        return render\_template('user\_dashboard.html',

                               username=user\_name,

                               bar\_chart=bar\_chart,

                               line\_chart=line\_chart,

                               pie\_chart=pie\_chart,

                               scatter\_plot=scatter\_plot

                            )

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True,host='0.0.0.0')

dashboard.py

import plotly.graph\_objs as go

import plotly.offline as opy

import sqlite3

import plotly.express as px

def fetch\_data(userid):

    conn = sqlite3.connect('Calorie.db')

    cursor = conn.cursor()

    # Example query: Retrieve total quantity sold for each product

    query = "SELECT exercise\_name, duration FROM exercise WHERE userid = ?"

    cursor.execute(query, (userid,))

    data = cursor.fetchall()

    # Example query: Retrieve total quantity sold over time

    time\_query = "SELECT date, calories FROM exercise WHERE userid = ?"

    cursor.execute(time\_query, (userid,))

    time\_data = cursor.fetchall()

    calories\_query = "SELECT exercise\_name, calories FROM exercise WHERE userid = ?"

    cursor.execute(calories\_query, (userid,))

    calories\_data = cursor.fetchall()

    heart\_query = "SELECT bpm, calories FROM exercise WHERE userid = ?"

    cursor.execute(heart\_query, (userid,))

    heart\_data = cursor.fetchall()

    cursor.close()

    conn.close()

    return data, time\_data, calories\_data, heart\_data

def create\_bar\_chart(data):

    exercise, duration = zip(\*data)

    trace = go.Bar(x=exercise, y=duration)

    layout = go.Layout(title='Exercise vs duration', xaxis=dict(title='Exercise'), yaxis=dict(title='Duration'))

    fig = go.Figure(data=[trace], layout=layout)

    return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_line\_chart(time\_data):

    date, calories = zip(\*time\_data)

    trace = go.Scatter(x=date, y=calories, mode='lines+markers', marker=dict(size=10), line=dict(width=2))

    layout = go.Layout(title='Total Calories Over Time', xaxis=dict(title='Date'), yaxis=dict(title='Total Calories'))

    fig = go.Figure(data=[trace], layout=layout)

    return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_pie\_chart(data, title):

    exercise\_names, calories = zip(\*data)

    trace = go.Pie(labels=exercise\_names, values=calories)

    layout = go.Layout(title=title)

    fig = go.Figure(data=[trace], layout=layout)

    return opy.plot(fig, auto\_open=False, output\_type='div')

def create\_heart\_rate\_scatter\_plot(data):

    bpm, calories = zip(\*data)

    trace = go.Scatter(x=bpm, y=calories, mode='markers', marker=dict(size=12))

    layout = go.Layout(title='Heart Rate vs. Calories Burned', xaxis=dict(title='Heart Rate (BPM)'), yaxis=dict(title='Calories Burned'))

    fig = go.Figure(data=[trace], layout=layout)

    return opy.plot(fig, auto\_open=False, output\_type='div')

Github and project demo link:

<https://github.com/smartinternz02/SI-GuidedProject-612450-1699198590>

<https://drive.google.com/file/d/1sOQXSaZ2dGT8nY8aryCNQb3-7z33NzCW/view?usp=sharing>