SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be **University** *u/s 3 of UGC Act, 1956)*

DEPARTMENT OF COMPUTER APPLICATION

COLLEGE OF SCIENCE AND HUMANITIES

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

BURN SMART

A PROJECT REPORT SUBMITTED TO SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR

THE AWARD OF THE DEGREE OF MASTER OF APPLIED DATA SCIENCE

BY

VARSHINI T	**RA2232014010024**
HAREL ALOY ISAAC C	**RA2232014010025**
PHEBE EVANGELINE R	**RA2232014010028**
ARUNKUMAR A	**RA2232014010056**

UNDER THE GUIDENCE OF

Dr. M JOHN BRITTO M.C.A., M.Phil., NET, Ph.D.

KATTANKULATHUR-603203

CHENGALPATTU, TAMILNADU

OCTOBER 2023

BONAFIDE CERTIFICATE

This is to certify that the project report titled **"BURN SMART"** is a bonafide work carried by **VARSHINI T (RA2232014010024), HAREL ALOY ISAAC C (RA2232014010025), PHEBE EVANGELINE (RA2232014010028), ARUNKUMAR A (RA2232014010056)** under my supervision for the Degree of Master of Applied Data Science. To my knowledge the work reported here in is the original work done by these students.

Dr. M JOHN BRITTO
Assistant Professor
Department of Computer Applications
(GUIDE)

Dr. S ALBERT ANTONY RAJ
Professor & Head
Department of Computer Applications

Internal Examiner

External Examiner

Declaration of Association of Research Project with SDG Goals

This is to certify that the research project entitled ----------

---

carried out by Mr / Miss ----------

under the supervision of Dr/Mr./Ms. ---------- (Designation) ---------- , of (Department)---------- in partial fulfilment of the requirement for the award of Under Graduation/Post Graduation/ Diploma/ Ph.D. program has been significantly or potentially associated with SDG Goal No ------- titled ----------

This study has clearly shown the extent to which its goals and objectives have been met in terms of filling the research gaps, identifying needs, resolving problems, and developing innovative solutions locally for achieving the above-mentioned SDG on a National and/or on an international level.

Signature of the Student **Guide and Supervisor**

Head of the Department

ACKNOWLEDGEMENT

With the profound gratitude to the **ALMIGHTY,** we take this chance to thank people who helped us to complete this project.

We take this as a right opportunity to say **THANKS** to our **Parents** who are there to stand with us always with the words **"YOU CAN"**

We are thankful to **Dr.T.R. Paarivendhar** Chancellor, SRM Institute of Science & Technology who gave us the platform to establish myself to reach Greater heights.

We earnestly thank **Dr. A. Duraisamy, Ph.D.,** Dean, College of Science and Humanities, SRM Institute of Science & Technology who always encourage us to do novel things.

We express our sincere thanks to **Dr. S. Albert Antony Raj, Ph.D.,** Professor and Head, Department of Computer Applications for his valuable guidance and support to execute all incline in learning.

It is our delight to thank our project guide, **Dr. M JOHN BRITTO, Ph.D.,** Associate Professor, Department of Computer Applications for his help, support, encouragement, suggestions and guidance throughout development phases of the project.

We convey our gratitude to all the **family members of the department** who extended their support through valuable comments and suggestions during the reviews.

A great note of gratitude to **friends** and people who are **known** and **unknown** to us who helped in carrying out this project work a successful one.

TABLE OF CONTENTS

ABSTRACT

This project aims to develop the machine learning models to predict calorie burn during workout. The project is motivated by the increasing emphasis on health and fitness, and the need for innovative tools and technologies to help people better understand their physical activity and its impact on their calorie expenditure.

Key components and objectives of the project:

Develop and train machine learning models to predict calorie burn based on a variety of factors, such as the heart rate, duration, body temperature, and individual characteristics such as age, weight, height and gender.

- This project compares the performance of different machine learning algorithms,
    - Linear Regression
        - Lasso Regression
        - Ridge Regression
    - Random Forest Regressor
    - XGB Regressor

- Optimize the machine learning models to improve their accuracy,
    - Hyperparameter Tuning
    - Feature selection
    - Data preprocessing

- A lower MAE (Mean Absolute Error) means the model is more accurate.
- Compare the performance of different machine learning algorithms and identify the best performing algorithm for predicting calorie burn.
- Develop a user-friendly interface to make the predictive models accessible to a wide range of users.

PROBLEM DESCRIPTION

Description:

- Predicting calories burned during exercise is a challenging task, as it depends on a variety of factors, including height, weight, gender, exercise duration, and body temperature. However, it can be difficult to accurately estimate calories burned without specialized equipment or training.

- Machine learning (ML) can be used to develop accurate and scalable models for predicting calories burned during exercise.

Goals:

- To develop an ML model that can accurately predict the number of calories burned during exercise using the following input attributes:

    - Height
    - Weight
    - Gender
    - Age
    - Heart rate
    - Workout duration
    - Body temperature

- The model should be implemented in a way that is scalable and efficient, so that it can be used to predict calories burned for a large number of users in real time.

CHAPTER I

INTRODUCTION

I INTRODUCTION

1.1OVERVIEW OF THE PROJECT

Estimating the number of calories burned during exercise is important for people who are trying to lose weight, improve their fitness, or track their overall health. However, it can be difficult to accurately estimate calories burned without specialized equipment or training. Machine learning can be used to develop accurate and scalable models for predicting calories burned during exercise. ML models can be trained on data from wearable devices, fitness trackers, and other sources to learn the relationship between factors such as exercise intensity, duration, and body composition and calories burned.

The model will be trained on a large and diverse dataset of exercise data, and it will be evaluated on its ability to generalize to new data accurately. The model will also be implemented in a way that is scalable and efficient, so that it can be used to predict calories burned for a large number of users in real time.

Overall, this project has the potential to develop an accurate and scalable ML model for predicting calories burned during exercise, which can be used in a variety of applications to help people improve their health and fitness.

1.2MODULE DESCRIPTION

- **Dependencies used:**

As we all know, in machine learning model development libraries are essential. The libraries in a programming language helps finishing the model easier. For this project Analysing and Visualizing plays the main role. So, Pandas, NumPy, Matplotlib, Seaborn, Scikit Learn and XGBoost, pickle, streamlit Libraries are used in this Project.

- **Dataset:**

A dataset with customers data is essential for this project. Data including customer's Age, Height, Weight, etc. For this Project we've chosen existing Dataset which contains Customer's details, their weight and their spending time for exercise.

- **Analysing Data:**

We are doing this project on Google Colab and Spyder Console platform. We are analysing the dataset by describing it, checking its shape, checking for missing data, viewing its properties, and cleaning the dataset for further Development.

- **Visualizing Data:**

Visualizing data includes all types of graph plots which are Count plot, Pair plot, Heatmap, Distribution plot and Scatter plot.

1.3 SYSTEM REQUIREMENT

1.3.1 Hardware Specification:

Processor	:	Intel core2/Ryzen3 or More
Memory	:	4 GB RAM or More
Hard disk Requirement	:	Free 500GB on installation drive

1.3.2 Software Specification:

- Browser with latest version Installed with Good Web Application Support
- **Language:** Python in Machine Learning
- **Platform:** Google Colab and Spider
- **Dataset Used:** Exercise and Calories Burn

1.3.3 Required Libraries:

- NumPy (pip install numpy)
- Pandas (pip install pandas)
- Matplotlib (pip install matplotlib)
- Seaborn (pip install seaborn)
- ScikitLearn (pip install sklearn)
- XGBoost (pip install XGboost)
- Pickle (pip install pickle)
- Streamlit (pip install streamlit)

1.4 SOFTWARE FEATURES

About Python:

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes. It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming. Python combines remarkable power with very clear syntax. It has interfaces to many systems calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants including Linux and macOS, and on Windows.

Python in Machine learning:

This Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. In the modern days, it is become very much easy and efficient compared to the olden days with various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reasons is its vast collection of libraries.

Python in Google Colab:

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Features of Python:

- Easy to learn and use
- Open-Source Libraries
- Support of Community
- Rapid development
- Productive
- Data Science
- Machine Learning

Python in Spyder Console:

Spyder's IPython Console serves as a powerful tool for deploying Python applications. By providing a comprehensive and interactive environment, it enables developers to seamlessly execute code, debug errors, and visualize results. The console's integration with Spyder's other features, such as the Variable Explorer and Editor, further streamlines the deployment process. Additionally, the ability to create and manage multiple consoles allows developers to isolate and test different versions of their code. As a result, Spyder's IPython Console is an invaluable asset for ensuring the successful deployment of Python applications.

1.5 LIBRARIES USED

Tools Employed:

- NumPy
- Pandas
- Seaborn
- Matplotlib
- Scikitlearn
- XGBoost
- Pickle
- Streamlit

NumPy:

- NumPy is a Python library for scientific computing.
- It provides a high-performance implementation of multidimensional arrays and matrices, as well as a large collection of mathematical functions for operating on those arrays.
- NumPy is widely used in data science, machine learning, and scientific computing.

Features:

- Multidimensional arrays: NumPy provides a high-performance implementation of multidimensional arrays, which are essential for many scientific and engineering applications.

- Mathematical functions: NumPy provides a large collection of mathematical functions for operating on multidimensional arrays, including linear algebra, statistical, and Fourier transform functions.
- Broadcasting: NumPy provides a powerful broadcasting mechanism that allows for efficient operations on arrays of different shapes.

Pandas:

- Pandas is a Python library for data analysis.
- It is built on top of the NumPy library and provides high-level data structures and operations for manipulating numerical data and time series.
- Pandas is widely used in data science, machine learning, and financial analysis.

Features:

- Data structures: Pandas provides two main data structures for storing and manipulating data: Series and DataFrame. A Series is a one-dimensional labeled array, similar to a Python list. A DataFrame is a two-dimensional labeled array, similar to a Python dictionary.
- Data manipulation: Pandas provides a wide range of functions for manipulating data, including sorting, filtering, grouping, and aggregating.
- Data visualization: Pandas integrates with Matplotlib, a Python library for data visualization, to provide easy-to-use plotting functions.
- Time series analysis: Pandas provides a variety of tools for working with time series data, such as date parsing, time shifting, and resampling.

Seaborn:

- Seaborn is a Python data visualization library built on top of Matplotlib.
- It provides a high-level interface for creating attractive and informative statistical graphics.
- Seaborn is widely used in data science, machine learning, and statistical analysis.

Features:

- Ease of use: Seaborn has a simple and intuitive interface, making it easy to learn and use.
- Attractive and informative visualizations: Seaborn produces high-quality visualizations with a consistent aesthetic.

- Statistical integration: Seaborn provides a variety of functions for statistical analysis, such as regression, correlation, and hypothesis testing.
- Deep integration with Pandas: Seaborn is tightly integrated with Pandas, making it easy to create visualizations from Pandas Data Frames.

Matplotlib:

- Matplotlib is a Python library for data visualization.
- It provides a wide range of plotting functions for creating static, animated, and interactive visualizations.
- Matplotlib is widely used in data science, machine learning, and scientific computing.

Features:

- Comprehensive set of plotting functions: Matplotlib provides a wide range of plotting functions for creating line plots, bar plots, scatter plots, histograms, and many other types of visualizations.
- Easy to use: Matplotlib has a simple and intuitive interface, making it easy to learn and use.
- Flexible and customizable: Matplotlib is highly customizable, allowing users to create visualizations that meet their specific needs.
- Well-documented: Matplotlib is well-documented, with a comprehensive user guide and tutorials.

Scikitlearn:

- Scikit-learn is a free software machine learning library for the Python programming language.
- It features various classification, regression, clustering and dimensionality reduction algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Scikit-learn is widely used in data science, machine learning, and natural language processing.

Features:

- Comprehensive set of algorithms: Scikit-learn provides a wide range of machine learning algorithms for classification, regression, clustering, and dimensionality reduction.
- Easy to use: Scikit-learn has a consistent and user-friendly interface, making it easy to learn and use.
- Efficient and scalable: Scikit-learn is written in a performance-critical fashion, making it efficient and scalable for large datasets.

XGBoost:

- XGBoost is an open-source machine learning library for boosting algorithms.
- It is a popular choice for machine learning tasks such as classification and regression.
- It is known for its speed and accuracy.

Features:

- Python-friendly interface: XGBoost provides a Python-friendly interface, making it easy to use for Python developers.
- Speed and accuracy: XGBoost is known for its speed and accuracy, making it a good choice for machine learning tasks where these qualities are important.
- Scalability: XGBoost is scalable, meaning that it can be used to train models on large datasets.
- Flexibility: XGBoost is a flexible library that supports a variety of features and parameters, giving users control over the training process.
- Classification: XGBoost can be used for classification tasks, such as predicting whether a customer will churn or not, or whether a medical image shows a tumor or not.
- Regression: XGBoost can be used for regression tasks, such as predicting the price of a house or the number of clicks an ad will receive.
- Ranking: XGBoost can be used for ranking tasks, such as ranking search results or product recommendations.

Pickle:

The pickle library is a built-in Python module that provides a mechanism for serializing and deserializing Python object structures. In other words, it allows Python objects to be converted into a byte stream that can be stored or transmitted, and then later reconstructed back into the original object.

Features:

- Serialization of most Python objects: The pickle library can serialize almost any Python object, including built-in types, user-defined classes, and instances of those classes.
- Recursive object support: The pickle library can handle recursive objects, which are objects that contain references to themselves.
- Object sharing: The pickle library can share objects between different serialization and deserialization operations. This means that if an object is pickled multiple times, only a single copy of the object will be stored.
- Portability: The pickle library is portable between different versions of Python. This means that a pickled object can be deserialized using a different version of Python than the one used to pickle it.

Streamlit:

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for data science and machine learning. With just a few lines of Python code, you can create interactive data visualizations, dashboards, and machine learning models that can be shared with others.

Features:

- Ease of use: Streamlit is designed to be easy to use, even for those with no prior experience in web development. Simply add Streamlit commands to your Python code, and Streamlit will automatically generate a web app based on your code.
- Interactivity: Streamlit apps are highly interactive, allowing users to explore data and experiment with machine learning models in real time. This makes Streamlit a great tool for communicating data science and machine learning results to others.

- Customization: Streamlit apps can be customized to meet your specific needs. You can add your own branding, layouts, and components to create a truly unique user experience.
- Sharing: Streamlit apps can be easily shared with others. Simply deploy your app to Streamlit Cloud or share the code with others so they can run it locally.

CHAPTER II

SYSTEM STUDY

II SYSTEM STUDY

2.1 EXISTING SYSTEM

This approach concentrates on the quantity of calories burned, leading to accurate predictions of calories burned. To have the correct understanding of weight reduction and management, calorie burn prediction is essential. Better understanding of the number of calories burned is crucial for leading a healthy life because the current system on calorie burnt forecast has significant disadvantages. People typically only consider eating or weight loss when they think of calories. However, a calorie is frequently a measurement of heat energy. The measurement could be applied to evaluate other energy-releasing mechanisms not connected to the human body. The quantity of Calories is viewed from the point of view of the human body as the energy required by the body to complete a task. Food has calories. As different foods have different calorie counts, each and every item has a unique amount of energy packed in it.

2.1.1 DISADVANTAGES OF EXISTING SYSTEM

- The existing system uses only one algorithm to make predictions, which can be limiting. If the data is not well-suited to that algorithm, or if the algorithm is not complex enough to capture all of the nuances of the data, the predictions will not be accurate.
- Additionally, the existing system does not optimize its predictions. This means that it does not take into account all of the factors that could affect the outcome of a prediction, and it may not produce the best possible prediction.

2.2 PROPOSED SYSTEM

The number of calories needed for activity and exercise varies from person to person. The number of calories a person burns while engaging in physical exercise depends on their age, gender, height, weight, body temperature, heart rate, and the length of the activity. The suggested method A comparison analysis of calorie burnt prediction that takes into account all the variables will yield superior outcomes. Each machine learning algorithm has advantages and disadvantages of its own. An algorithm that works best for one System might not work well for another. In order to find the most accurate model for predicting the number of calories burned, our system assesses the accuracy of various regression models like Random Forest, Ridge Regression, Lasso Regression, XGBoost and we have optimized it.

2.2.1 ADVANTAGES OF PROPOSED SYSTEM

Since we have used various algorithms, it helps the model in producing optimized solution. Moreover, we have developed a webpage for our system which helps in Prediction, Recommendation, Personalization and Anomaly detection. The significance of webpage in machine learning model are

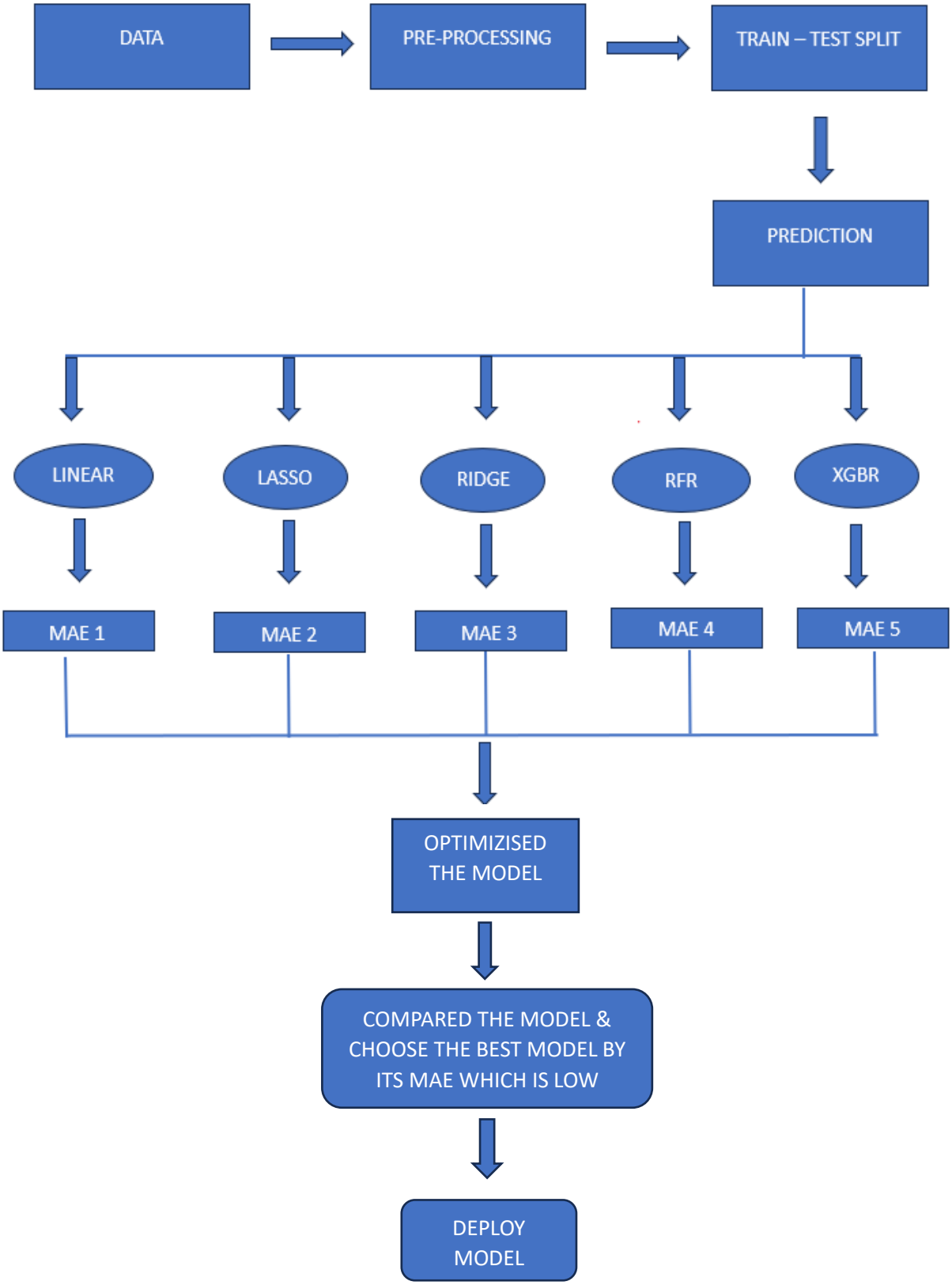
Model deployment: Webpages can be used to deploy machine learning models so that they can be used by users over the internet. This can be done by creating a web application that exposes the model as an API. The web application can then be integrated with other websites or applications to allow users to make predictions using the model.

CHAPTER III

SYSTEM DESIGN

III SYSTEM DESIGN

3.1 SYSTEM ARCHITECURE



3.2 DATASET

A data set is a collection of related, continuous items of related data that may be accessed individually or in combination or managed as a whole entity. A data set is organized into some type of data structure. Data sets are also used to store information needed by applications or the operating system itself, such as source programs, macro libraries, or system variables or parameters.

3.2.1 DATA SOURCE

We use "Kaggle" as our dataset store. A total of 15000 instances and 7 attributes of data are present throughout 2 CSV files. The "Kaggle" repository's dataset comprises information about a variety of people, including their height, weight, gender, age, workout intensity, heart rate, and body temperature. The training data is taken from the “exercise.csv” and “calories.csv” datasets. Additionally, the user id-mapped target class from the second calorie dataset comprises the calories burned by the person in exercise dataset.

Further, utilizing the Structured Query Language (SQL), we combine the data contained within the two tables 'exercise.csv' and 'calories.csv' by employing a join operation on the 'userid' column. The resulting dataset is subsequently assigned the name ' Exercise and Calories Burn.csv'.

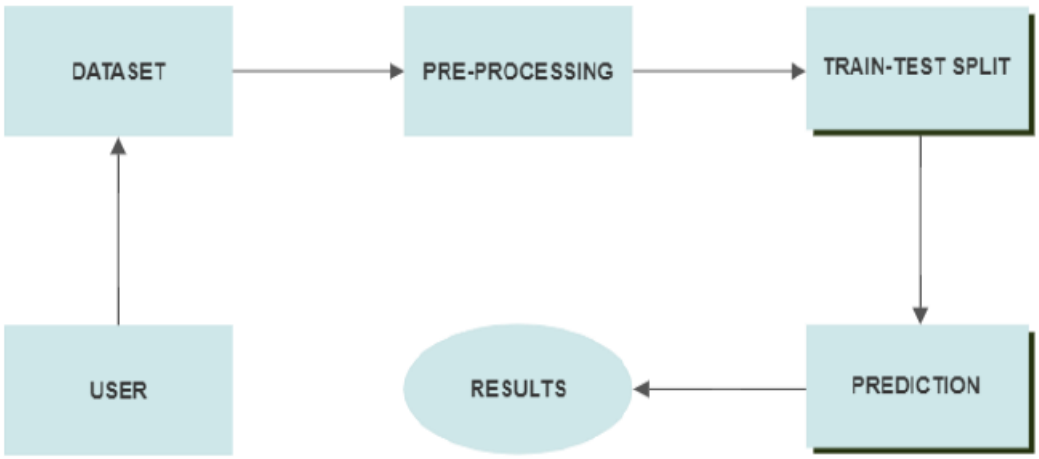
The new dataset csv files which should be uploaded to Google Colab which is used for processing. We use data frames for analysis and processing.

The following table shows the input features (columns) and its function.

Input Attributes	**Function**
gender	gender (male: 0, female: 1)
age	age mentioned in years
height	height of the person
weight	weight of the person
duration	The time taken to complete the exercising in minutes.
heart_rate	Average heart rate during the workout (more than normal rate 75 beats/min)
body_temp	Body temperature in the course of the workout (greater than 37 degree celsius)
Calories_burn	Total calories burned during the exercise.

3.3 IMPLEMENTATION PROCESS

Combining the implementation of multiple linear regression, Lasso regression, Ridge regression, Random Forest Regressor, and XGBoost Regressor in a predictive modelling task can be done using ensemble techniques to take advantage of the strengths of each method. Here are the steps for a combined implementation:



Data Preparation:

- Gather and preprocess your dataset.
- Handle missing values, outliers, and encode categorical variables as needed.

Data Split:

- Split the data into training and testing sets to evaluate the models' performance.

Feature Scaling:

- Standardize or normalize your features if necessary. This step is particularly important for linear regression models.

Models

Linear Regression Models

- Implement multiple linear regression, Lasso regression, and Ridge regression separately on the training data.
- Tune the hyperparameters, such as alpha for Lasso and Ridge, through cross-validation.

- Evaluate the performance of each model on the testing data using appropriate metrics (e.g., mean squared error or R-squared).

Random Forest Regressor:

- Tune hyperparameters, such as the number of trees, max depth, and minimum samples per leaf.
- Evaluate the performance on the testing data.

XGBoost Regressor:

- Implement the XGBoost Regressor on the training data.
- Tune hyperparameters, such as learning rate, max depth, and the number of boosting rounds.
- Evaluate the performance on the testing data.

Evaluate the Model:

- Assess the performance of the model on the testing data using appropriate metrics.

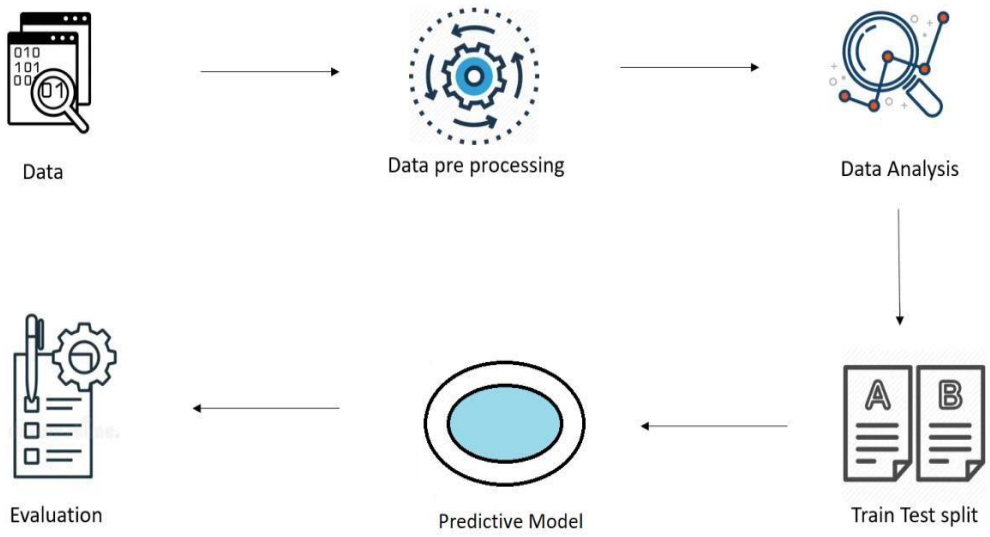
Fine-Tune and Refine:

- Iterate on the above steps by fine-tuning models and the ensemble approach based on the evaluation results.

Deployment:

- Once you are satisfied with the ensemble's performance, you can deploy it for making predictions on new, unseen data.
- Create a webpage using Streamlit library

3.4 WORKFLOW



The diagram also shows a magnifying glass and a gear, which represent the tools and techniques used in data processing. The clock represents the time required for data processing, and the magnifying glass represents the insights that can be extracted from the data.

It provides a good overview of the process of data processing. It shows the different steps involved in data processing, as well as the tools and techniques used. The image also highlights the importance of data processing for extracting insights from data and making predictions.

Exercise and Calories Burn Dataset:

- As we have already seen the ‘Exercise and Calories Burn.csv' dataset. we are using. The Dataset is the first step and requirement for Calories burn Prediction and visualization
- The dataset csv files which should be uploaded to Google Colab which is used for processing.
- First of all, we are loading the dataset for Analysis.

3.4.1 DATA PREPROCESSING

Import libraries:

Connecting Google Drive in Colab to Load the Dataset

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor


from sklearn import metrics
from sklearn.metrics import mean_absolute_error as mae
from sklearn import set_config

set_config(display='diagram')
from google.colab import drive
drive.mount('/content/drive')

import pickle
```

Load the dataset & Storing information in a Data Frame:

To load a CSV dataset in Python, first import the Pandas library. Then, specify the path to the CSV file. Next, use the **pandas.read_csv()** function to read the CSV file into a **DataFrame.** Using **shape** function to get the number of rows and columns in a dataframe.

```
[ ] #Load the dataset
    dataset = pd.read_csv('/content/drive/MyDrive/MiniProject -SRM Dataset/Exercise and Calories Burn.csv')
```

```
#shape of the dataset
print("Dataset Shape:",dataset.shape)
print("***"*20)
```

```
Dataset Shape: (15000, 9)
************************************************
```

Checking for null values:

Null values are missing values in a dataset. They can occur for a variety of reasons, such as data entry errors or incomplete surveys. Null values can cause problems for data analysis, so it is important to identify and handle them before proceeding with analysis. The **DataFrame.isnull().sum()** function was used to count the number of null values in each column of the DataFrame. This allowed the analyst to see which columns had missing values and how many missing values were present.

```
print("The dataset have the null values or not:\n",dataset.isnull().sum())
print("***"*20)
```

```
The dataset have the null values or not:
 User_ID          0
Gender            0
Age               0
Height            0
Weight            0
Duration          0
Heart_Rate        0
Body_Temp         0
Calories_Burn     0
dtype: int64
************************************************************
```

Checking the no. of unique values in each column:

The pandas.unique() function is used to return the unique values in a Pandas Series or DataFrame. The function returns an array of unique values in the order of appearance.

```
print("The unique values present in each of the column as follows:")
print(dataset.nunique(),"\n")
print("***"*20)
```

```
The unique values present in each of the column as follows:
User_ID          15000
Gender               2
Age                 60
Height              90
Weight              91
Duration            30
Heart_Rate          59
Body_Temp           45
Calories_Burn      277
dtype: int64

************************************************************
```

Summarize the dataset using statistical method:

- The **pandas.info()** method provides a concise summary of a DataFrame. This summary includes information of index dtype and columns, number of non-null values in each column,data types of each column,memory usage of the DataFrame
- **The pandas.describe()** method provides summary statistics for the numerical columns in a DataFrame. These statistics include count of non-null values,mean,standard deviation,minimum value, 25th percentile, 50th percentile (median),75th percentile,maximum value.

```
print("The information of data followed below:")
print(dataset.info())
print("***"*20)

print("The statistical measures about the dataset as follows:")
print(dataset.describe())
print("***"*20)
```

```
The information of data followed below:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   User_ID        15000 non-null  int64
 1   Gender         15000 non-null  object
 2   Age            15000 non-null  int64
 3   Height         15000 non-null  int64
 4   Weight         15000 non-null  int64
 5   Duration       15000 non-null  int64
 6   Heart_Rate     15000 non-null  int64
 7   Body_Temp      15000 non-null  float64
 8   Calories_Burn  15000 non-null  int64
dtypes: float64(1), int64(7), object(1)
memory usage: 1.0+ MB
None
************************************************************
The statistical measures about the dataset as follows:
            User_ID           Age        Height        Weight      Duration  \
count  1.500000e+04  15000.000000  15000.000000  15000.000000  15000.000000
mean   1.497736e+07     42.789800    174.465133     74.966867     15.530600
std    2.872851e+06     16.980264     14.258114     15.035657      8.319203
min    1.000116e+07     20.000000    123.000000     36.000000      1.000000
25%    1.247419e+07     28.000000    164.000000     63.000000      8.000000
50%    1.499728e+07     39.000000    175.000000     74.000000     16.000000
75%    1.744928e+07     56.000000    185.000000     87.000000     23.000000
max    1.999965e+07     79.000000    222.000000    132.000000     30.000000

          Heart_Rate     Body_Temp  Calories_Burn
count  15000.000000  15000.000000   15000.000000
mean      95.518533     40.025453      89.539533
std        9.583328      0.779230      62.456978
min       67.000000     37.100000       1.000000
25%       88.000000     39.600000      35.000000
50%       96.000000     40.200000      79.000000
75%      103.000000     40.600000     138.000000
max      128.000000     41.500000     314.000000
************************************************************
```

View the dataset:

```
print("The first 5 set of rows from the dataset shown below:")
print(dataset.head())
print("***"*20)
```

```
The first 5 set of rows from the dataset shown below:
    User_ID  Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp  \
0  14733363    male   68     190      94        29         105       40.8
1  14861698  female   20     166      60        14          94       40.3
2  11179863    male   69     179      79         5          88       38.7
3  16180408  female   34     179      71        13         100       40.5
4  17771927  female   27     154      58        10          81       39.8

   Calories_Burn
0            231
1             66
2             26
3             71
4             35
************************************************************
```

Handling categorical data:

- Using **sklearn.preprocessing.LabelEncoder()** is a Python class that can be used to convert categorical data into numerical data.
- This is useful for machine learning tasks, as most machine learning algorithms only accept numerical data.

```
lb=LabelEncoder()
dataset['Gender']=lb.fit_transform(dataset['Gender'])
print(lb.classes_)
print("Gender:")
print("      Female --> 0 \n      Male   --> 1")
print("***"*20)
```

```
[0 1]
Gender:
      Female --> 0
      Male   --> 1
************************************************************
```

Removed Unwanted Columns:

```
#Drop the unuse feature from the dataset
data = dataset
data = data.drop(columns=['User_ID'], axis=1)
```

```
print("Data Shape:",data.shape)
print(data.head())
```

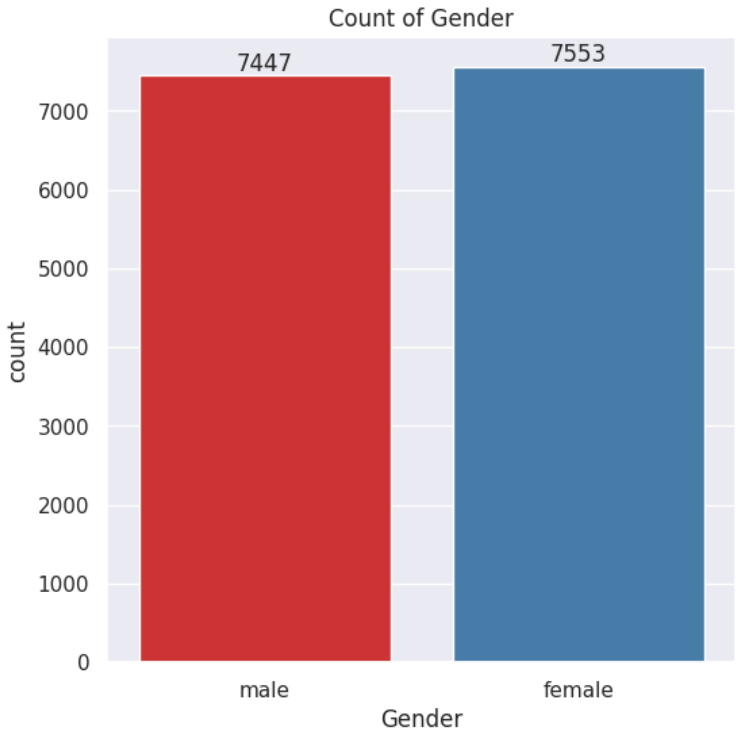
```
Data Shape: (15000, 8)
   Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp  Calories_Burn
0    male   68     190      94        29         105       40.8            231
1  female   20     166      60        14          94       40.3             66
2    male   69     179      79         5          88       38.7             26
3  female   34     179      71        13         100       40.5             71
4  female   27     154      58        10          81       39.8             35
```

3.5 DATA VISUALIZATION

Count Plot:

A countplot is a type of bar plot that is used to show the number of observations for each category in a categorical variable. Countplots are often used to visualize the distribution of categorical data.

```
#count of gender in the dataset
sns.set()
plt.figure(figsize=(6,6))
ax=sns.countplot(x=data.Gender,palette = "Set1")
ax.bar_label(ax.containers[0], fmt='{:.0f}')
plt.title("Count of Gender")
plt.show()
```



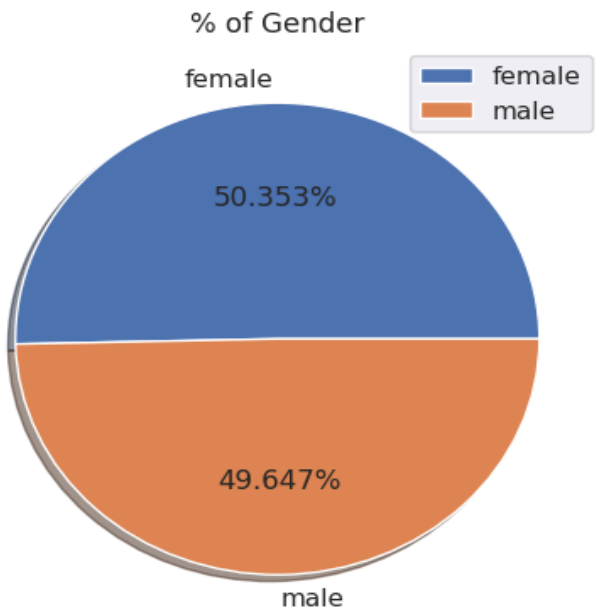
Observation:

Based on the given count plot for who can do exercise, it can be observed that female are more likely to do exercise and burn calories than male. There are 7553 female who can do exercise, while there are 7447 male who can do exercise. This means that there are 106 more female who can do exercise than male.

Pie Chart:

- A pie chart is a circular statistical graphic that is divided into slices to illustrate numerical proportion.
- It is a popular data visualization technique for representing categorical data.
- Pie charts are easy to read and understand, making them a good choice for communicating data to a general audience.

```
#pie chart of gender in %
plt.figure(figsize=(5,7))
plt.pie(data.Gender.value_counts(),labels=data.Gender.value_counts().index,autopct="%.3f%%",shadow=True)
plt.title("% of Gender")
plt.legend()
plt.show()
```



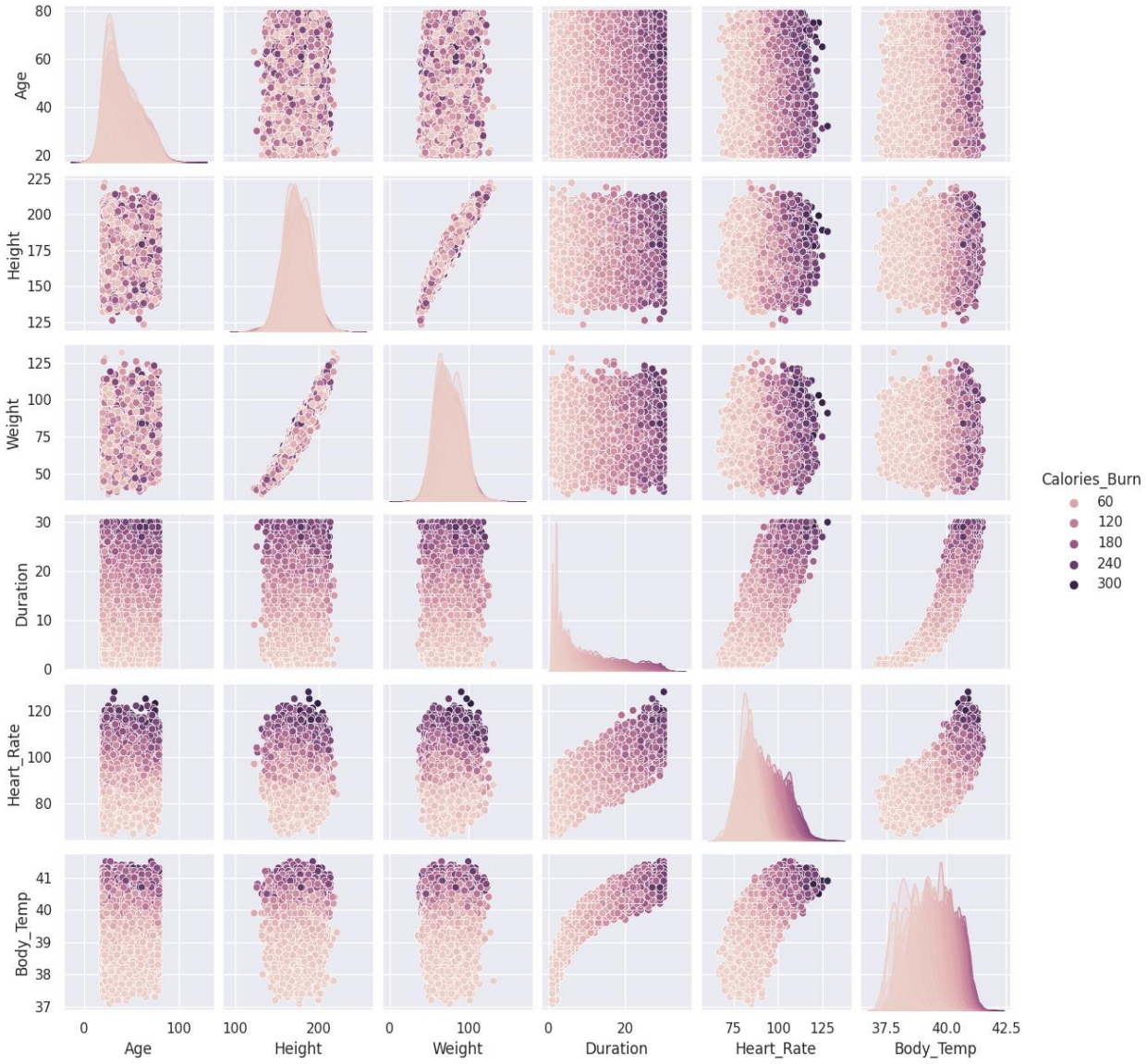
Observation:

The pie chart shows that 50.353% of people who can do exercise are females, while 49.647% are males. This means that there are slightly more females who can do exercise than males.

Pair plot:

- A pairplot is a visualization technique that creates a matrix of scatter plots, with each plot showing the relationship between two variables.
- Pairplots are a useful tool for exploring the relationships between multiple variables in a dataset.

```
#pairplot
sns.set()
plt.figure(figsize=(6,6))
sns.pairplot(data, hue='Calories_Burn', height=2)
plt.show()
```

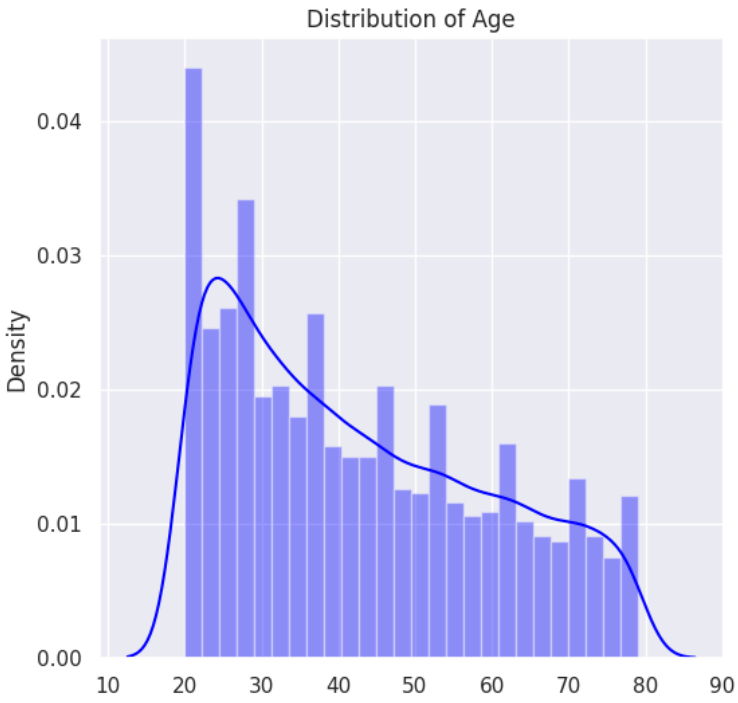


Distribution plot:

- A distribution plot is a type of data visualization that shows the distribution of a numerical variable.
- It can be used to identify patterns and trends in the data, as well as outliers and other anomalies.

- **Distribution of Age**

```
#Distribution of Age
plt.figure(figsize=(6,6))
sns.distplot(x=data.Age,color='blue')
plt.title("Distribution of Age")
plt.show()
```

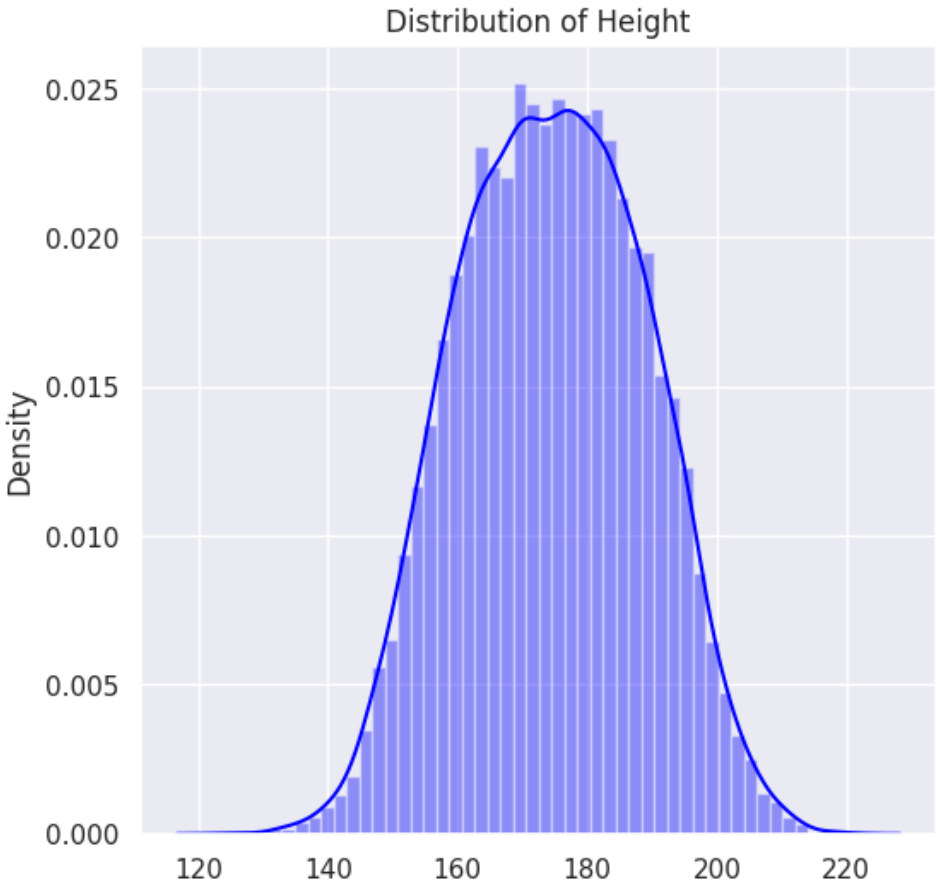


Observation:

The line distribution plot shows the distribution of age for those who can do exercise. The curve is skewed, with the highest density of people in their 20s and 30s. This means that the average age for those who can do exercise is between 20 and 30 years old. The number of people who can do exercise decreases with age.

- **Distribution of Height**

```
#Distribution of Height
plt.figure(figsize=(6,6))
sns.distplot(x=data.Height,color='blue')
plt.title("Distribution of Height")
plt.show()
```

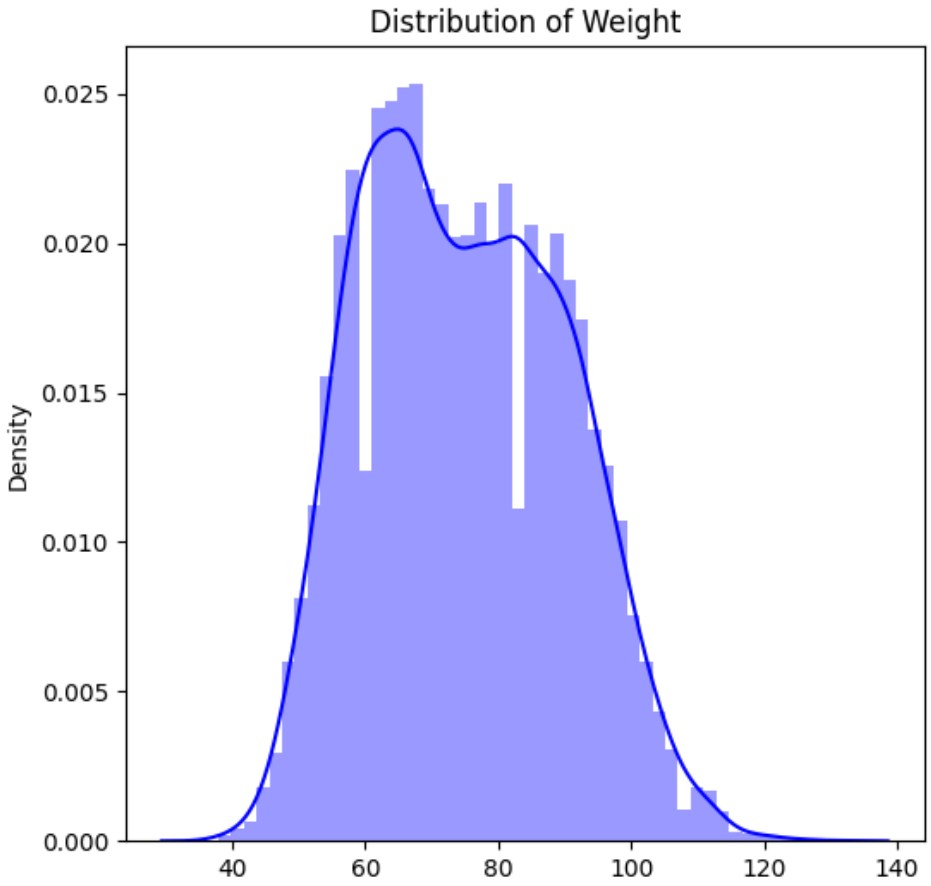


Observation:

- There are more people who are between 160 and 180 centimeters tall who can do exercise than people who are shorter or taller.
- The number of people who can do exercise decreases with height. This is likely due to a number of factors, such as the challenges of exercising with a shorter or taller body type.
- There is a small number of people who can do exercise who are very short or very tall. This is a positive sign that exercise is becoming more accessible to people of all heights

- **Distribution of Weight**

```
#Distribution of Weight
plt.figure(figsize=(6,6))
sns.distplot(x=data.Weight,color='blue')
plt.title("Distribution of Weight")
plt.show()
```

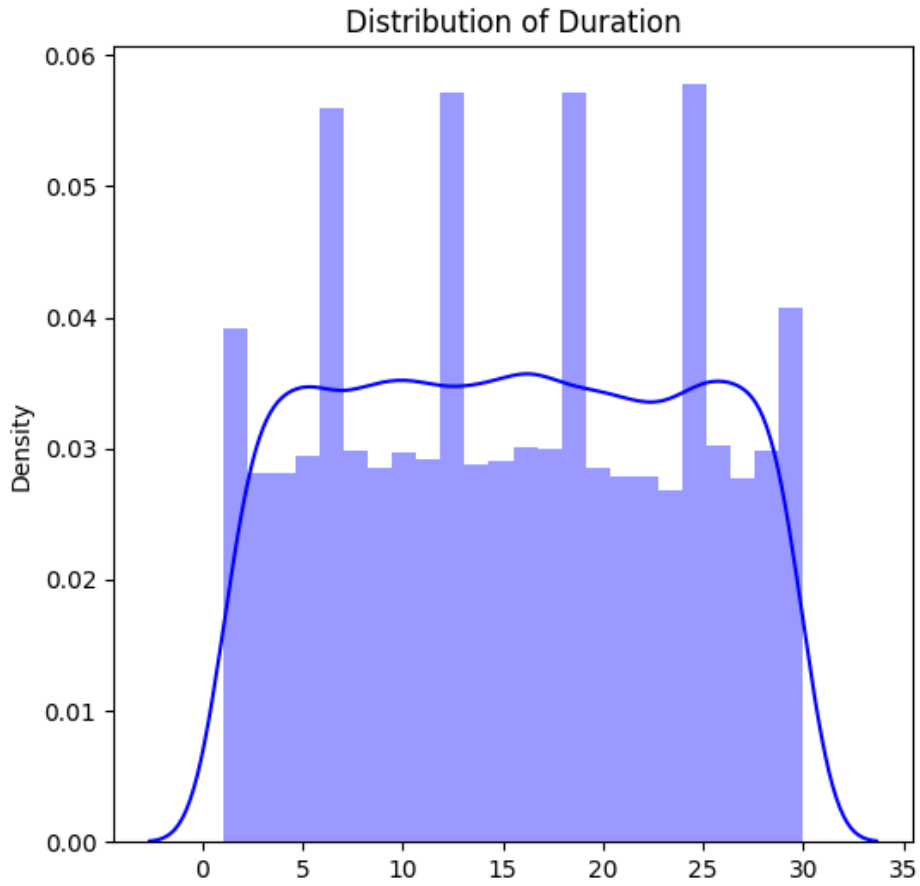


Observation:

The distribution of weight for people who can do exercise is skewed, with the highest density of people in the 60-80 kilogram range. This means that the average weight for people who can do exercise is between 60 and 80 kilograms.

- **Distribution of Duration**

```
#Distribution of Duration
plt.figure(figsize=(6,6))
sns.distplot(x=data.Duration,color='blue')
plt.title("Distribution of Duration")
plt.show()
```

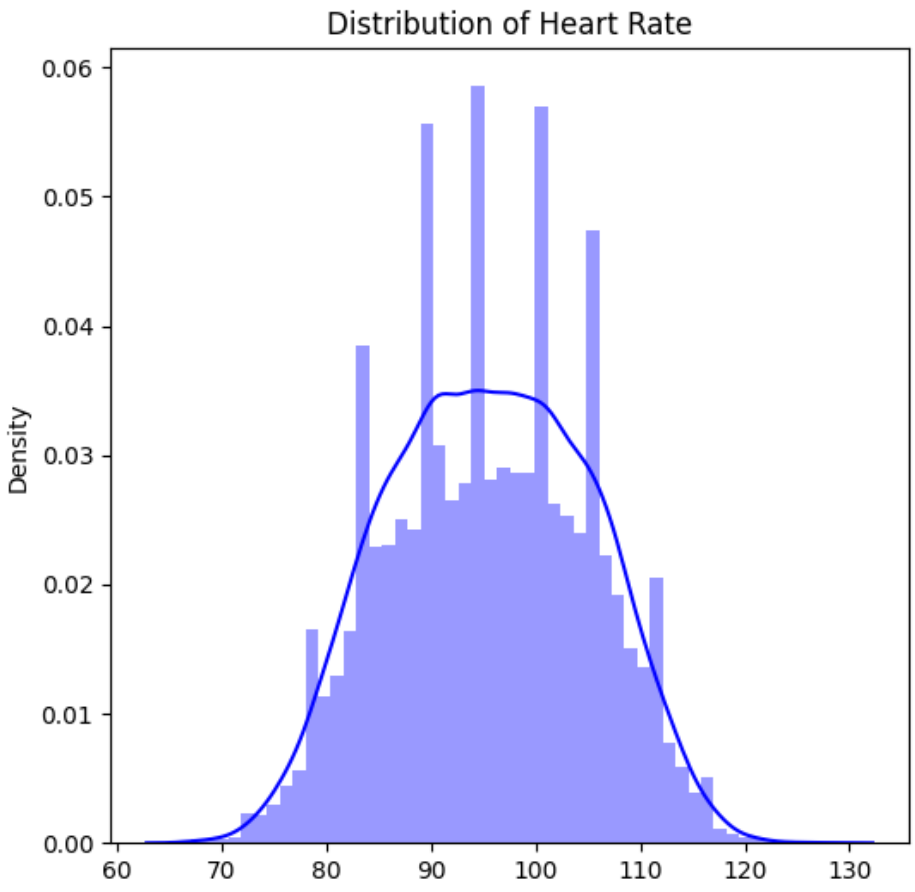


Observation:

The distribution of duration of exercise for people who can do exercise is not skewed, with the highest density of people exercising for 20-35 minutes. This means that the average duration of exercise for people who can do exercise is between 20 and 35 minutes.

- **Distribution of Heart Rate**

```
#Distribution of Heart Rate
plt.figure(figsize=(6,6))
sns.distplot(x=data.Heart_Rate,color='blue')
plt.title("Distribution of Heart Rate")
plt.show()
```

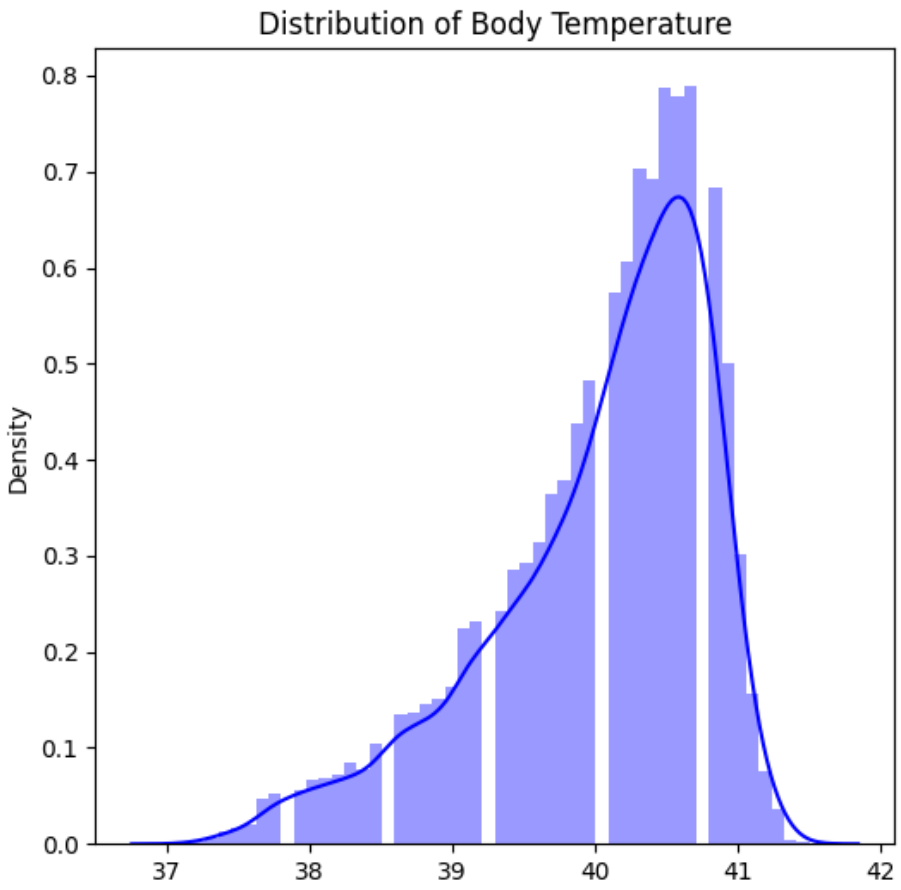


Observation:

The distribution of average heart rate for people who can do exercise is approximately normal, with a mean of 120 beats per minute and a standard deviation of 10 beats per minute. This means that the majority of people who can do exercise have an average heart rate between 110 and 130 beats per minute.

- **Distribution of Body Temperature**

```
#Distribution of Body Temperature
plt.figure(figsize=(6,6))
sns.distplot(x=data.Body_Temp,color='blue')
plt.title("Distribution of Body Temperature")
plt.show()
```



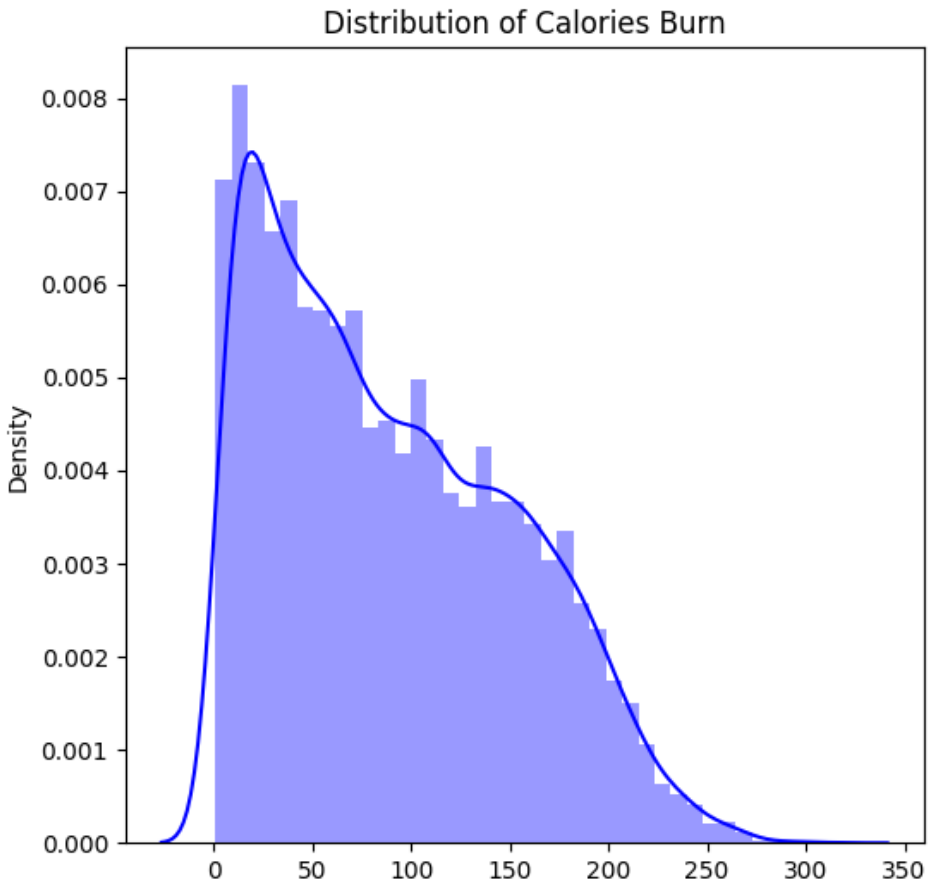
Observation:

The distribution of average body temperature for people who can do exercise is approximately normal, with a mean of 37.5 degrees Celsius and a standard deviation of 0.5 degrees Celsius. This means that the majority of people who can do exercise have an average body temperature between 37 degrees Celsius and 38 degrees Celsius.

However, there is a wide range of average body temperatures of people who can do exercise. It shows that there are people who can do exercise with an average body temperature as low as 36.5 degrees Celsius and as high as 38.5 degrees Celsius. This means that average body temperature is not a perfect indicator of fitness level or exercise ability.

- **Distribution of Calories Burn**

```
#Distribution of Calories Burn
plt.figure(figsize=(6,6))
sns.distplot(x=data.Calories_Burn,color='blue')
plt.title("Distribution of Calories Burn")
plt.show()
```



Observation:

- The distribution of calories burned for people who can do exercise is skewed, with the highest density of people burning between 150 and 250 calories per session. This means that the average calorie burn for people who can do exercise is between 150 and 250 calories per session.
- However, there is a wide range of calories burned by people who can do exercise. It shows that there are people who can do exercise and burn less than 100 calories per session and more than 500 calories per session.

Heatmap:

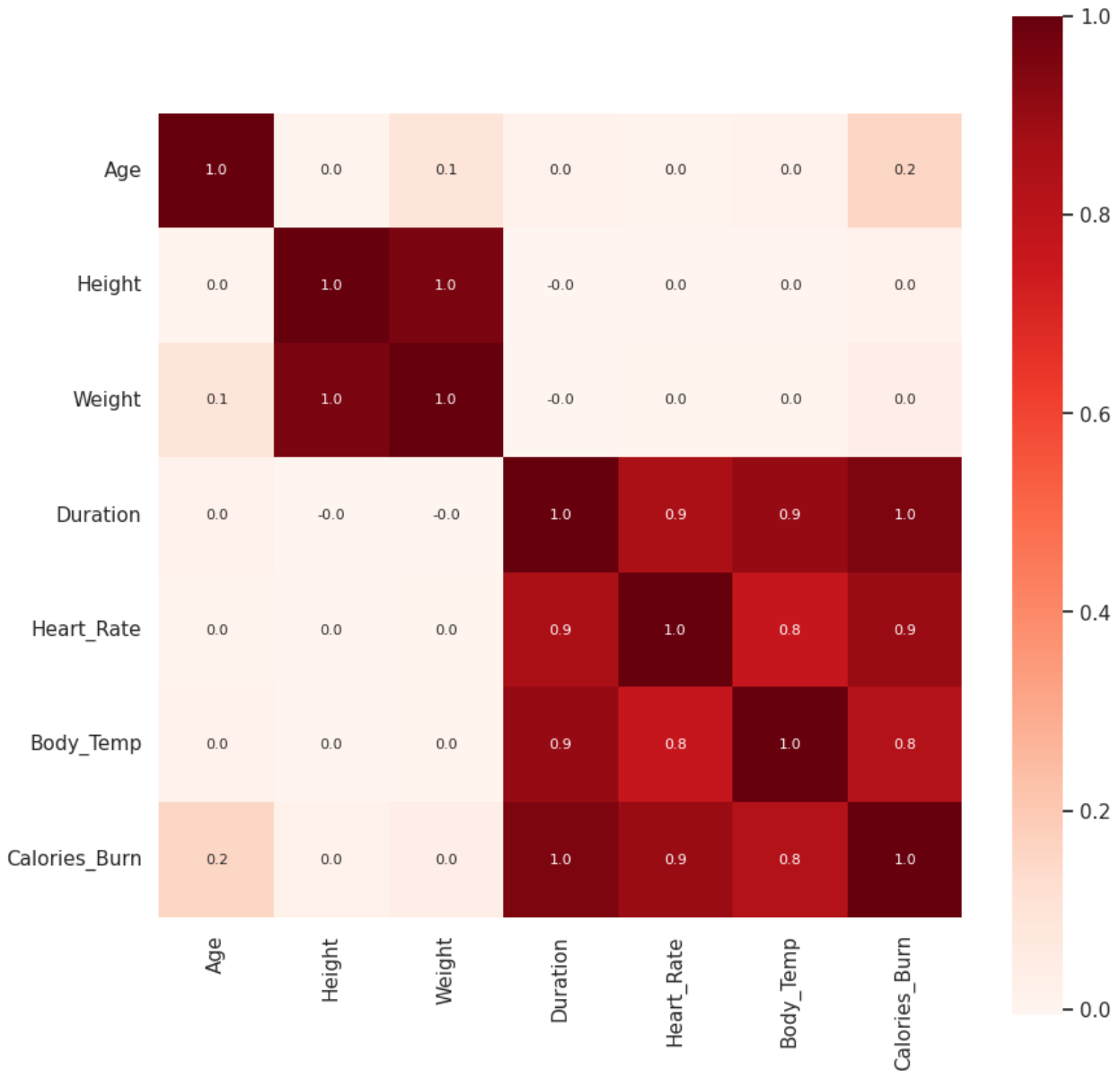
- A heatmap is a graphical representation of data where the individual values that are contained in a matrix are represented as colours.
- The colour of each cell in the heatmap represents the magnitude of the value in that cell.
- Heatmaps are often used to visualize large and complex datasets, as they can provide a quick and easy way to identify patterns and trends.

```
#correlation between the each column of the dataset
print("Correlation among the columns follows below:")
print(data.corr())
```

```
Correlation among the columns follows below:
                    Age    Height    Weight  Duration  Heart_Rate  Body_Temp  \
Age            1.000000  0.009554  0.090094  0.013247    0.010482   0.013175
Height         0.009554  1.000000  0.958451 -0.004625    0.000528   0.001200
Weight         0.090094  0.958451  1.000000 -0.001884    0.004311   0.004095
Duration       0.013247 -0.004625 -0.001884  1.000000    0.852869   0.903167
Heart_Rate     0.010482  0.000528  0.004311  0.852869    1.000000   0.771529
Body_Temp      0.013175  0.001200  0.004095  0.903167    0.771529   1.000000
Calories_Burn  0.154395  0.017537  0.035481  0.955421    0.897882   0.824558

               Calories_Burn
Age                 0.154395
Height              0.017537
Weight              0.035481
Duration            0.955421
Heart_Rate          0.897882
Body_Temp           0.824558
Calories_Burn       1.000000
```

```
#corr value round to -1 to 1
#heatmap
correlation = data.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap='Reds')
```



Observation:

The heatmap shows the correlation between the different features in the exercise dataset. The darker the colour, the stronger the correlation. The correlation coefficient is a measure of the strength of the linear relationship between two variables. It ranges from -1 to 1, with a value of 1 indicating a perfect positive correlation, a value of -1 indicating a perfect negative correlation, and a value of 0 indicating no correlation.

- Age and height: There is a weak positive correlation between age and height. This means that people who are taller tend to be older.
- Weight and height: There is a strong positive correlation between weight and height. This means that people who are taller tend to weigh more.
- Height and duration of exercise: There is a weak negative correlation between height and duration of exercise. This means that taller people tend to exercise for a shorter duration.
- Weight and duration of exercise: There is a moderate negative correlation between weight and duration of exercise. This means that heavier people tend to exercise for a shorter duration.
- Heart rate and calories burned: There is a strong positive correlation between heart rate and calories burned. This means that people with a higher heart rate burn more calories.
- Body temperature and calories burned: There is a moderate positive correlation between body temperature and calories burned. This means that people with a higher body temperature burn more calories.

Scatter Plot:

- A scatter plot is a type of data visualization that shows the relationship between two variables.
- Each point in the scatter plot represents a single data point, and the color and size of the point can be used to represent additional information.
- Scatter plots are a useful tool for exploring the relationship between two variables and identifying patterns and trends.

- **Relation between Age and Calories Burn**

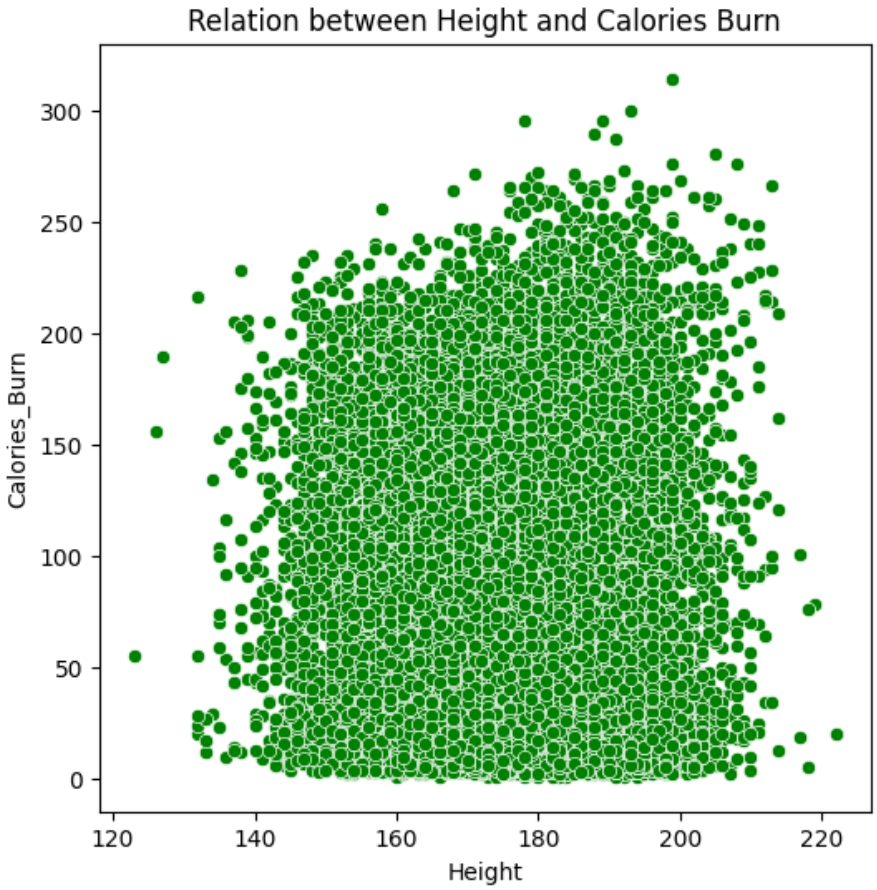
```
#Relation between Age and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Age,y=data.Calories_Burn,color='green')
plt.title("Relation between Age and Calories Burn")
plt.show()
```

Observation:

The scatter plot shows the relationship between age and calories burned during exercise. The scatter plot shows that there is a negative correlation between age and calories burneduring exercise. This means that as people get older, they tend to burn fewer calories during exercise.

- **Relation between Height and Calories Burn**

```
#Relation between Height and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Height,y=data.Calories_Burn,color='green')
plt.title("Relation between Height and Calories Burn")
plt.show()
```



Observation:

The scatter plot shows the relationship between height and calories burned during exercise. The scatter plot shows that there is a positive correlation between height and calories burned during exercise. This means that as people get taller, they tend to burn more calories during exercise. it is also possible that taller people have a higher resting metabolic rate (RMR). RMR is the number of calories that the body burns at rest. People with a higher RMR burn more calories throughout the day, including during exercise.

- **Relation between Weight and Calories Burn**

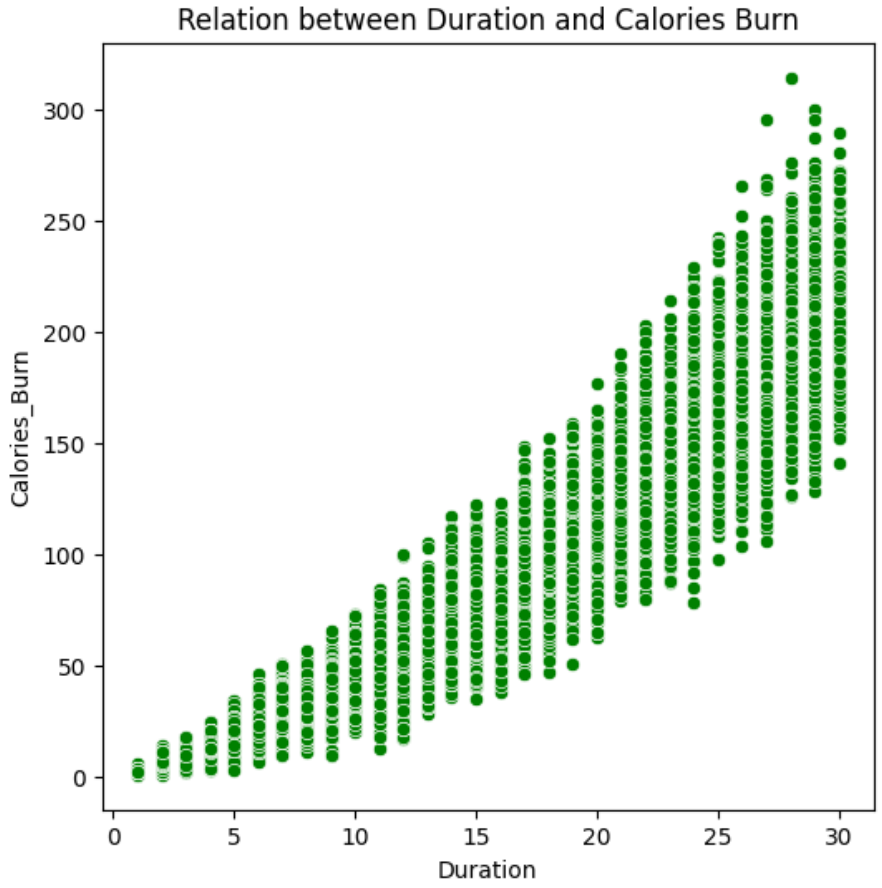
```
#Relation between Weight and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Weight,y=data.Calories_Burn,color='green')
plt.title("Relation between Weight and Calories Burn")
plt.show()
```

Observation:

- The scatter plot shows the relationship between weight and calories burned during exercise. The correlation between weight and calories burned during exercise is moderate. This means that there is a relationship between the two variables, but it is not perfect.
- There is a wide range of calories burned during exercise for people of all weights. This means that weight is not the only factor that determines how many calories a person burns during exercise. Other factors, such as fitness level, muscle mass, and intensity of exercise, also play a role.

- **Relation between Duration and Calories Burn**

```
#Relation between Duration and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Duration,y=data.Calories_Burn,color='green')
plt.title("Relation between Duration and Calories Burn")
plt.show()
```

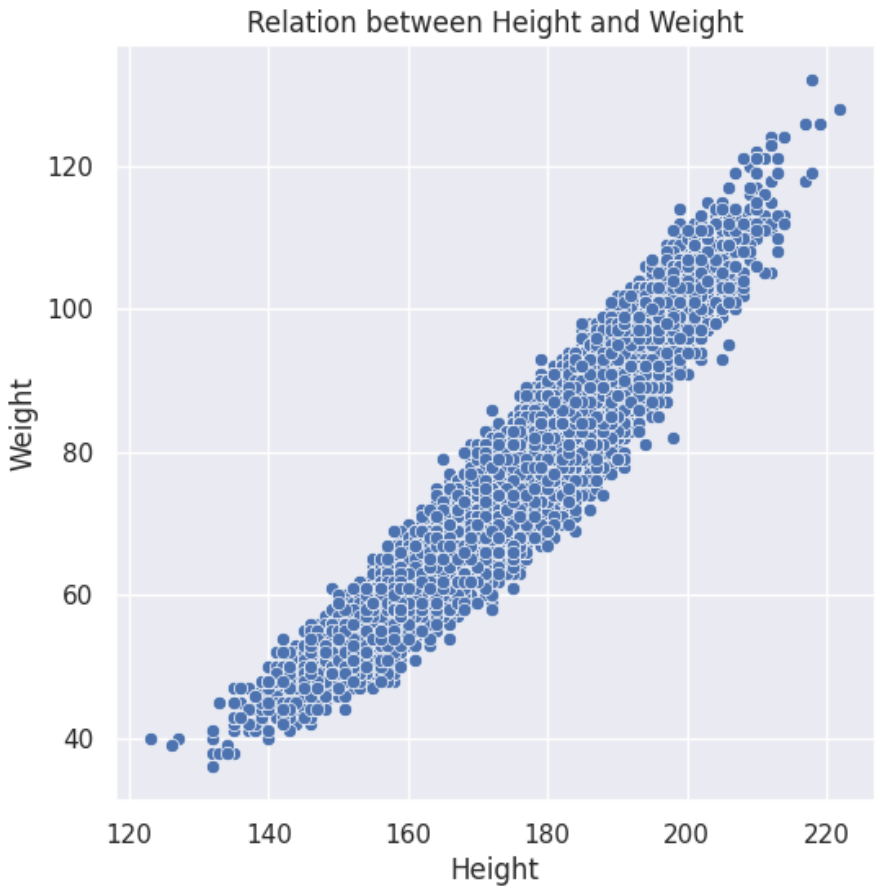


Observation:

- The correlation between duration of exercise and calories burned during exercise is strong. This means that there is a strong relationship between the two variables.
- It is important to note that the scatter plot only shows the correlation between duration of exercise and calories burned during exercise. It does not show whether duration of exercise causes people to burn more calories. It is possible that there are other factors that are causing the positive correlation, such as intensity of exercise or fitness level.

- **Relation between Height and Weight**

```
#Relation between Height and Weight
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Height,y=data.Weight)
plt.title("Relation between Height and Weight")
plt.show()
```



Observation:

- The correlation between height and weight is moderate. This means that there is a relationship between the two variables
- There are a few possible explanations for this positive correlation. One possibility is that taller people have a larger muscle mass. Muscle mass is more dense than fat, so taller people with more muscle mass may weigh more than shorter people with less muscle mass.

Overall Observations:

- As expected, higher is the duration of the workout higher will be the calories burnt. But except for that, we cannot observe any such relation between calories burnt and height or weight features.
- The average height of the male is higher than female. Also, the weight of the girls is lower than that of the boys.
- For the same average duration of workout calories burnt by men is higher than that of women.
- So, we have a kind of linear relationship between these two features which is quite obvious.

CHAPTER IV

ALGORITHM IMPLEMENTATION

IV ALGORITHM IMPLEMENTATION

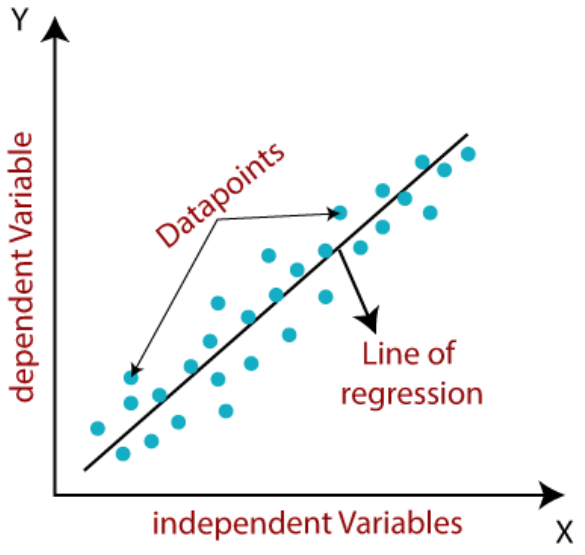
4.1 ALGORITHMS USED

- Linear Regression (Multi-linear)
- Lasso Regression
- Ridge Regression
- Random Forest Regressor
- XGB Regressor

Linear Regression

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression.

The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s)



The Linear equations is

$$y= a0+a1x+ \varepsilon$$

where:

Y=Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

Types of Linear Regression:

Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression

Assumptions of Linear Regression:

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- Linear relationship between the features and target:
- Small or no multicollinearity between the features:
- Homoscedasticity Assumption:
- Normal distribution of error terms

Multiple Linear Regression

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

The multiple linear regression equation is

$$y = b0 + b1x1 + b2x2 + ... + bnxn + \varepsilon$$

where:

y is the dependent variable

b0 is the y-intercept

b1, b2, ..., bn are the regression coefficients for the independent variables x1, x2, ..., xn

ε is the error term

Assumptions for Multiple Linear Regression:

- A linear relationship should exist between the Target and predictor variables.
- The regression residuals must be normally distributed.
- MLR assumes little or no multicollinearity (correlation between the independent variable) in data

Advantages of linear regression:

Simple to understand and implement: Linear regression is a relatively simple algorithm to understand and implement. This makes it a good choice for beginners in machine learning.

Computationally efficient: Linear regression is a computationally efficient algorithm. This means that it can be trained and run quickly, even on large datasets.

Robust to noise: Linear regression is relatively robust to noise in the data. This means that it can still produce accurate predictions even if there is some noise in the data.

Disadvantages of linear regression:

Assumes linearity: Linear regression assumes that there is a linear relationship between the dependent variable and the independent variables. If the relationship is non-linear, then linear regression will not produce accurate predictions.

Prone to overfitting: Linear regression is prone to overfitting. Overfitting occurs when the model learns the training data too well and is unable to generalize to new data.

Lasso Regression:

Lasso regression, also known as L1 regularization, is a type of linear regression that performs both variable selection and regularization. It encourages sparse solutions where some coefficients are forced to be exactly zero. This makes lasso regression well-suited for datasets with many features, as it can help to identify the most important features and to reduce overfitting.

Lasso regression works by adding a penalty term to the traditional linear regression cost function. The penalty term is the sum of the absolute values of the regression coefficients. As the value of the coefficients increases from zero, the penalty term penalizes the model, causing it to decrease the value of the coefficients in order to reduce loss.

The strength of the regularization is controlled by a hyperparameter called lambda. A larger value of lambda will force more coefficients to zero, resulting in a sparser model. A smaller value of lambda will allow more coefficients to be non-zero, resulting in a more complex model.

$$L(x, y)= \text{Min}\left(\sum_{i=1}^{n} (y_i - w_i x_i)^2 + \lambda \sum_{i=1}^{n} |w_i| \right)$$

Advantages of lasso regression:

Feature selection: Lasso regression can be used to identify the most important features in a dataset. This can be useful for reducing the dimensionality of the dataset and for improving the performance of the model.

Overfitting prevention: Lasso regression can be used to prevent overfitting in linear regression. This is because the regularization term penalizes the model for having large coefficients.

Disadvantages of lasso regression:

Sensitive to outliers: Lasso regression is sensitive to outliers in the data. Outliers can distort the regression coefficients and lead to inaccurate predictions.

Biased coefficients: The coefficients produced by lasso regression are biased. This is because the regularization term artificially shrinks the coefficients closer to zero.

Ridge Regression:

Ridge regression, also known as L2 regularization, is a type of linear regression that performs regularization. It adds a penalty term to the traditional linear regression cost function, which penalizes the model for having large coefficients. This helps to reduce overfitting and to improve the generalization performance of the model.

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}{\beta_j}^2$$

The strength of the regularization is controlled by a hyperparameter called lambda. A larger value of lambda will result in a stronger penalty and a sparser model. A smaller value of lambda will result in a weaker penalty and a more complex model.

Ridge regression is often used in conjunction with other regularization techniques, such as lasso regression and elastic net regularization. This can help to improve the performance of the model on complex problems.

Advantages:

- Reduces overfitting: Ridge regression adds a penalty term to the traditional linear regression cost function, which penalizes the model for having large coefficients. This helps to reduce overfitting and to improve the generalization performance of the model.

- Computationally efficient: Ridge regression is a computationally efficient algorithm. This means that it can be trained and run quickly, even on large datasets.

Disadvantages:

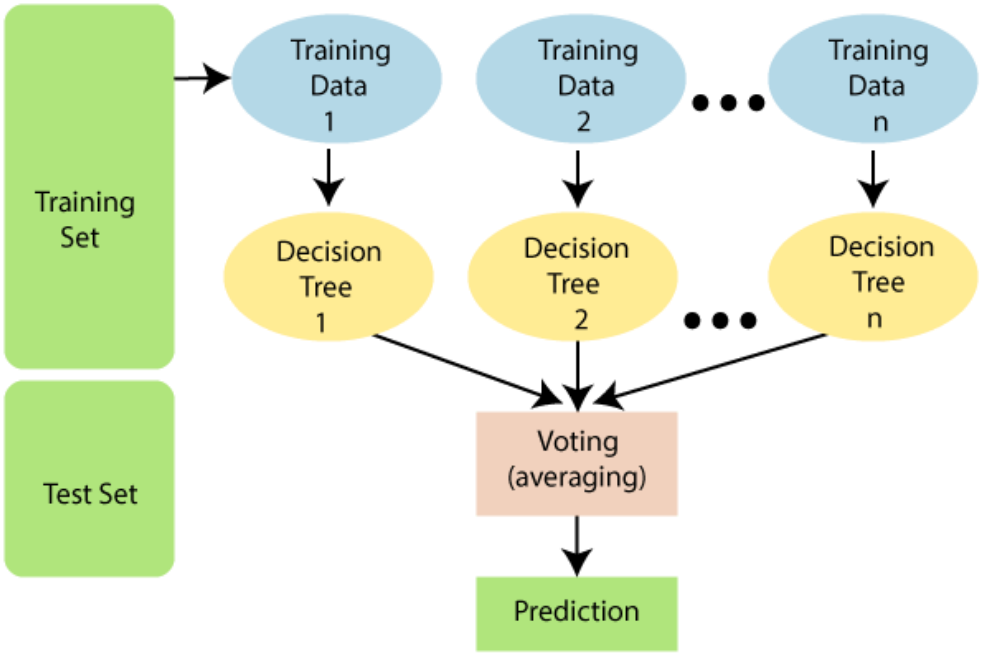
- Biased coefficients: The coefficients produced by ridge regression are biased. This is because the regularization term artificially shrinks the coefficients closer to zero.
- Sensitive to outliers: Ridge regression is sensitive to outliers in the data. Outliers can distort the regression coefficients and lead to inaccurate predictions.

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.The below diagram explains the working of the Random Forest algorithm:



Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations

Random Forest Regressor:

RandomForestRegressor is a supervised machine learning algorithm that can be used for regression tasks. It is an ensemble learning algorithm, which means that it combines the predictions of multiple individual models to produce a more accurate prediction.

RandomForestRegressor works by building a collection of decision trees. Each decision tree is trained on a random subset of the data, and a random subset of the features are considered at each split in the tree. This helps to reduce the overfitting of the model to the training data.

Once the decision trees are trained, RandomForestRegressor makes a prediction by averaging the predictions of all of the individual trees. This averaging process helps to reduce the variance of the model and to improve its overall accuracy.

RandomForestRegressor is a powerful and versatile algorithm that can be used for a variety of regression tasks. It is especially well-suited for problems with high-dimensional data and non-linear relationships between the variables.

Advantages:

Accurate: RandomForestRegressor is a very accurate algorithm that can be used to produce accurate predictions on a variety of regression tasks.

Robust to overfitting: RandomForestRegressor is less prone to overfitting than other regression algorithms, such as linear regression. This is because it uses a technique called bagging, which involves training multiple models on random subsets of the data.

Versatile: RandomForestRegressor can be used for a variety of regression tasks, including both linear and non-linear problems. It can also be used for problems with high-dimensional data.

Disadvantages:

Can be computationally expensive: RandomForestRegressor can be computationally expensive to train, especially on large datasets.

Can be difficult to interpret: RandomForestRegressor models can be difficult to interpret, as they are made up of a collection of decision trees. This can make it difficult to understand how the model is making its predictions.

XGBRegressor:

XGBoost is a machine learning algorithm that uses an ensemble of decision trees and gradient boosting to make predictions. It is widely used in data science and has won several machine learning competitions.

XGBRegressor is a supervised machine learning algorithm that can be used for regression tasks. It is based on the gradient boosting algorithm, but it makes a number of improvements that make it more efficient and accurate. XGBRegressor works by building a collection of decision trees. Each decision tree is trained on a residual of the previous tree, which is the difference between the predicted and actual values. This helps to improve the accuracy of the model.

XGBRegressor is a powerful and versatile algorithm that can be used for a variety of regression tasks. It is especially well-suited for problems with high-dimensional data and non-linear relationships between the variables.

XGBoost Benefits and Attributes:

1. High accuracy: XGBoost is known for its accuracy and has been shown to outperform other machine learning algorithms in many predictive modeling tasks.
2. Scalability: XGBoost is highly scalable and can handle large datasets with millions of rows and columns.
3. Efficiency: XGBoost is designed to be computationally efficient and can quickly train models on large datasets.
4. Flexibility: XGBoost supports a variety of data types and objectives, including regression, classification, and ranking problems.
5. Regularization: XGBoost incorporates regularization techniques to avoid overfitting and improve generalization performance.
6. Interpretability: XGBoost provides feature importance scores that can help users understand which features are most important for making predictions.

Disadvantages:

Can be computationally expensive: XGBRegressor can be computationally expensive to train, especially on large datasets.

Can be difficult to tune: XGBRegressor has a number of hyperparameters that can be tuned to improve the performance of the model. However, tuning these hyperparameters can be difficult and time-consuming

4.2 SPLITTING PROCESS

4.2.1 Splitting input and output features (X and y):

```
#Relation between Height and Weight
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Height,y=data.Weight)
plt.title("Relation between Height and Weight")
plt.show()
```

```
Input Features of the data:
       Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
0        male   68     190      94        29         105       40.8
1      female   20     166      60        14          94       40.3
2        male   69     179      79         5          88       38.7
3      female   34     179      71        13         100       40.5
4      female   27     154      58        10          81       39.8
...       ...  ...     ...     ...       ...         ...        ...
14995  female   20     193      86        11          92       40.4
14996  female   27     165      65         6          85       39.2
14997  female   43     159      58        16          90       40.1
14998    male   78     193      97         2          84       38.3
14999    male   63     173      79        18          92       40.5

[15000 rows x 7 columns]
**********************************************************************
Output Feature of the data:
0        231
1         66
2         26
3         71
4         35
        ...
14995     45
14996     23
14997     75
14998     11
14999     98
Name: Calories_Burn, Length: 15000, dtype: int64
```

4.2.2Splitting the X and y into training and testing set.

Using sklearn.model_selection.train_test_split is a function provided by the scikit-learn library (often abbreviated as sklearn) in Python. It is a very useful utility for splitting a dataset into training and testing sets for machine learning purposes.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
print("Input Shape:",X.shape)
print("Training Set Shape:",X_train.shape)
print("Testing Set Shape:",X_test.shape)
```

```
Input Shape: (15000, 7)
Training Set Shape: (12000, 7)
Testing Set Shape: (3000, 7)
```

4.2.3 Rescale the features of 'X'

StandardScaler:

- StandardScaler is a Python class that can be used to standardize numerical data.
- This involves subtracting the mean from each feature and then dividing by the standard deviation.
- Standardization is a common preprocessing step for machine learning tasks, as it can help to improve the performance of machine learning algorithms.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

4.3 MODEL SELECTION

4.3.1 Model Selection and Comparison the models

```
from sklearn.metrics import mean_absolute_error as mae
models = [LinearRegression(), XGBRegressor(),
        Lasso(), RandomForestRegressor(), Ridge()]

for i in range(5):
    models[i].fit(X_train, y_train)

    print(f'{models[i]} : ')

    train_preds = models[i].predict(X_train)
    print('Training Error : ', mae(y_train, train_preds))

    y_pred = models[i].predict(X_test)
    print('Validation Error : ', mae(y_test, y_pred))
    print()
```

```
LinearRegression() :
Training Error :  8.332985229896742
Validation Error :  8.385188053147179

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...) :
Training Error :  0.9322033420062313
Validation Error :  1.4833678883314132

Lasso() :
Training Error :  9.049658999072935
Validation Error :  8.989469141792506

RandomForestRegressor() :
Training Error :  0.6865499999999999
Validation Error :  1.6871166666666664

Ridge() :
Training Error :  8.332695704014442
Validation Error :  8.38482196600372
```

4.3.2 Optimization of model

For Ridge Regression:

```
#initialising Ridge() function
ridge = Ridge()
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
                    1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,10.0, 20, 50, 100, 500, 1000 ]}

# defining cross validation folds as 5
folds =5
grid_cv_model = GridSearchCV(estimator=ridge,
                        param_grid=params,
                        scoring='neg_mean_absolute_error',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)

# fiting GridSearchCV() with X_train and y_train
l2 = grid_cv_model.fit(X_train,y_train)
print("Best score for Ridge :",l2.best_score_)
print("The best alpha for Ridge is:",l2.best_params_)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
Best score for Ridge : -8.314399852597974
The best alpha for Ridge is: {'alpha': 10.0}
```

For Lasso Regression:

```
#initialising lasso() function
lasso = Lasso()
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
                    0.8, 0.9, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,
                    10.0, 20, 50, 100, 500, 1000 ]}

# defining cross validation folds as 5
folds =5
grid_cv_model = GridSearchCV(estimator=lasso,
                        param_grid=params,
                        scoring='neg_mean_absolute_error',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)

# fiting GridSearchCV() with X_train and y_train
l1 = grid_cv_model.fit(X_train,y_train)
print("Best score for Lasso:",l1.best_score_)
print("The best alpha for Lasso is:",l1.best_params_)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
Best score for Lasso: -8.312317779467548
The best alpha for Lasso is: {'alpha': 0.05}
```

For Random Forest Regressor:

```
rfr = RandomForestRegressor(random_state=0)
parameters = {'n_estimators': [100, 150, 200, 250, 300],'max_depth': [1,2,3,4],}
grid_cv_model = GridSearchCV(rfr, parameters)

rfr_model = grid_cv_model.fit(X_train, y_train)
print("Best score for RandomForestRegressor",rfr_model.best_score_)
print("The best alpha for RandomForestRegressor is:",rfr_model.best_params_)
```

```
Best score for RandomForestRegressor 0.9573196740144668
The best alpha for RandomForestRegressor is: {'max_depth': 4, 'n_estimators': 150}
```

For XGB Regressor

```
xgb = XGBRegressor()
parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
              'objective':['reg:linear'],
              'learning_rate': [.03, 0.05, .07], #so called `eta` value
              'max_depth': [5, 6, 7],
              'min_child_weight': [4],
              'subsample': [0.7],
              'colsample_bytree': [0.7],
              'n_estimators': [500]}

grid_cv_model = GridSearchCV(xgb,
                        parameters,
                        cv = 2,
                        n_jobs = 5,
                        verbose=True)

xgb_model = grid_cv_model.fit(X_train, y_train)
print("Best score for RandomForestRegressor",xgb_model.best_score_)
print("The best alpha for RandomForestRegressor is:",xgb_model.best_params_)
```

4.4 BEST MODEL

Using sklearn.metrics.mean_absolute_error is a function in the scikit-learn (sklearn) library for Python that is used to calculate the mean absolute error (MAE) between actual and predicted values in a regression problem. The mean absolute error is a common metric for evaluating the performance of regression models.

The mean absolute error is defined as the average of the absolute differences between the actual (observed) values and the predicted values. Mathematically, it can be expressed as:

$$MAE = (1/n) * \Sigma |y_i - \hat{y}_i|$$

Where,

MAE: Mean Absolute Error

n: The number of data points (samples)

y_i: The actual (observed) value for the i-th data point

ŷ_i: The predicted value for the i-th data point

Σ: The summation symbol, indicating that you should sum up the absolute differences for all data points.

4.4.1 Validate the model

```
from sklearn.metrics import mean_absolute_error as mae
models = [LinearRegression(), Ridge(alpha=10), Lasso(alpha=0.05),RandomForestRegressor(),
          XGBRegressor(colsample_bytree=0.7,learning_rate=0.07,max_depth=5,min_child_weight=4,
                       n_estimators=500,nthread=4,objective='reg:linear',subsample=0.7)]

for i in range(5):
    models[i].fit(X_train, y_train)

    print(f'{models[i]} : ')

    train_preds = models[i].predict(X_train)
    print('Training Error : ', mae(y_train, train_preds))

    y_pred = models[i].predict(X_test)
    print('Validation Error : ', mae(y_test, y_pred))
    print()
```

```
LinearRegression() :
Training Error :  8.306790197742497
Validation Error :  8.441513553849703

Ridge(alpha=10) :
Training Error :  8.304994376664778
Validation Error :  8.438993797246134

Lasso(alpha=0.05) :
Training Error :  8.302465721352533
Validation Error :  8.442647976630964

RandomForestRegressor() :
Training Error :  0.6698925
Validation Error :  1.7103733333333335

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [16:26:49]
  warnings.warn(smsg, UserWarning)
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.7, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.07, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=None,
             min_child_weight=4, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=None, nthread=4,
             num_parallel_tree=None, ...) :
Training Error :  0.7825378867114584
Validation Error :  1.005842310667038
```

Best Model:

Out of all the above models, we have trained RandomForestRegressor and the XGB model's performance is the same as their MAE for the validation data is same.

The XGBRegressor model has a lower validation error than the RandomForestRegressor model, which suggests that it is less likely to overfit the data and perform better on unseen data. Therefore, the XGBRegressor model is the better choice for deployment.

```
#bestmodel
xgb = XGBRegressor(colsample_bytree=0.7,learning_rate=0.07,max_depth=5,min_child_weight=4,
                   n_estimators=500,nthread=4,objective='reg:linear',subsample=0.7)
model = xgb.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [16:26:58] WARNIN
  warnings.warn(smsg, UserWarning)
```

4.5 PREDICTION AND DEPLOYMENT MODEL

Prediction Model:

- Getting Inputs from the user one by one.

```
#unseen data
print("Chosse Your Gender according to the Refrence Given Below.")
print("Female --> 0\nMale   --> 1")
gender = int(input("Enter Your Gender: "))
print('*'*60)

print("Choose The Age From 17.")
age = int(input("Enter Your Age: "))
print('*'*60)

height = float(input("Enter Your Height (in cm) : "))
print('*'*60)

weight = float(input("Enter Your Weight (in kg) : "))
print('*'*60)

duration = int(input("Enter Your duration (in mins) : "))
print('*'*60)

heart_rate = int(input("Enter Your Heart Rate from 60 to 130 : " ))
print('*'*60)

body_temp = int(input("Enter Your Body temp from 36 to 42 (in celsius ) : " ))
print('*'*60)
```

```
Chosse Your Gender according to the Refrence Given Below.
Female --> 0
Male   --> 1
Enter Your Gender: 1
************************************************************
Choose The Age From 17.
Enter Your Age: 22
************************************************************
Enter Your Height (in cm) : 170
************************************************************
Enter Your Weight (in kg) : 118
************************************************************
Enter Your duration (in mins) : 33
************************************************************
Enter Your Heart Rate from 60 to 130 : 118
************************************************************
Enter Your Body temp from 36 to 42 (in celsius ) : 37
************************************************************
```

- Put all the inputs in one 2D array name as “new_entry”.
- Then, call the predict function to the best model by passing the new_entry as the argument
- It gives the calories burn to the given inputs.

```
#predict calories burn
new_entry = [[gender,age,height,weight,duration,heart_rate,body_temp]]
Predict_lr = model.predict(new_entry)
print("Calories burn by Your above details:",Predict_lr)
```

```
Calories burn by Your above details: [272.74466]
```

Deploy the Model:

```
#deploy model
filename = "best_model_cb_pred.sav"
pickle.dump(xgb, open(filename, 'wb'))
```

Steps follow to implement the code using streamlit library to create a webpage

- From the left side of colab click file icon on this Download the "best_model_cb_pred.sav".
- Further, implementation will be in the spider console to create a webpage using the above downloaded file.

CHAPTER V

SYSTEM TESTING AND IMPLEMENTATION

V SYSTEM TESTING AND IMPLEMENTATION

5.1 SYSTEM TESTING

Testing is carried out after the development of the proposed system. The principal activity of system development is preparing the source code. In this system the source code is developed for each module separately. The source code is prepared for master files and they are compiled and corrected. Then the source code for the transaction files is prepared, compiled and corrected. Then the modules are combined and corrected as a whole module.

A strategy for software testing must accommodate low-level tests that are necessary to verify that all small source code segments has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. Testing is a process of executing program with the intent of finding error. A good test case is one that has high probability of finding an undiscovered error. If testing is conducted successfully, it uncovers the errors in the software. Testing cannot show the absence of defects, it can only show that software defects present. Test configuration includes test plan and test cases and test tools.

Testing Objectives:

Software Testing has different goals and objectives. The major objectives of Software testing are as follows:

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality and to prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and
- SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product

Testing Methodologies:

Testing methodologies are the strategies and approaches used to test a particular product to ensure it is fit for purpose. Testing methodologies usually involve testing that the product works in accordance with its specification, has no undesirable side effects when used in ways outside of its design parameters and worst case will fail-safely.

- **Unit Testing:**

Unit testing is essential for the verification of the code produced during the coding phase and hence the goal is to test the internal logic of the modules. Using the detailed design description as a guide, important paths are tested to uncover errors within the boundary of the modules. These tests were carried out during the programming stage itself.

- **Integration Testing**

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover error associated within the interface. The objective is to take unit tested modules and build a program structure that has been dictated by design. All modules are combined in this step. The entire program is tested as whole. And chaos in interfaces may usually result. A set of errors is encountered in such a case.

- **Validation Testing**

Here in the validation testing we want to check whether the given conditions to the text box are working correctly. Because in the name place we want to enter the characters and the special symbols only we should not enter the numbers in the name field. Here while on runtime we entered numeric values in the string specified columns of product inwards. It raises error. In this phase each module has been tested by wrong inputs, for example Employee Name should be a character as well as their age should be in numbers.

- **Functional Testing**

The functional testing part of a testing methodology is typically broken down into four components - unit testing, integration testing, system testing and acceptance testing – usually executed in this order. Entire system is working properly or not will be tested here, and specified path connection is correct or not, and giving output or not are tested here these verifications and validations are done by giving input values to the system and by comparing with expected output.

5.2 SYSTEM IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system and is giving confidence on the new system for the users that it will work efficiently and effectively. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the changeover, an evaluation of change over methods. Apart from planning major task of preparing the implementation are education and training of users. The implementation process begins with preparing a plan for the implementation of the system.

According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system. In network backup system no additional resources are needed. Implementation is the final and the most important phase. The most critical stage in achieving a successful new system is giving the users confidence that the new system will work and be effective. The system can be implemented only after thorough testing is done and if it is found to be working according to the specification. This method also offers the greatest security since the old system can take over if the errors are found or inability to handle certain type of transactions while using the new system. As the part of system testing, we execute the program with the intent of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied. The ultimate aim is quality assurance

CHAPTER VI

DEPLOY MODEL

VI DEPLOY MODEL

Steps follows to deploy the predicted model:

1) In the spyder run the model loading code snippet and save the file as a 'py' extension

2) Next open Anaconda terminal and change the working directory to the location where the python file is stored

3) Run the following command <streamlit run NAME_OF_THE_PYTHON_FILE

A webpage will be created in your default browser where you can enter the details required

Below Screenshot shows the code and output of code and webpage:

```
import numpy as np
import streamlit as st
import pickle

loaded_model = pickle.load(open('C:/Users/arunk/Desktop/Final Mini/best_model_cb_pred.sav','rb'))


def prediction(new_entry):
    Predict_lr = loaded_model.predict(new_entry)
    outp = float(Predict_lr)
    return (outp)

def main():
    st.title("Calories burn prediction")
    gender = st.number_input("Enter Your Gender as (Female:0,Male:1) ",step=0,min_value=0, max_value=1)
    age = st.number_input("Enter Your Age: ",step=0,min_value=15, max_value=100)
    height = st.number_input("Enter Your Height (in cm) : ",step=0,min_value=100, max_value=200)
    weight = st.number_input("Enter Your Weight (in kg) : ",step=0,min_value=40, max_value=200)
    duration = st.number_input("Enter Your duration (in mins) : ",step=0,min_value=1, max_value=180)
    heart_rate = st.number_input("Enter Your Heart Rate from 60 to 130 : ",step=0,min_value=60, max_value=130 )
    body_temp = st.number_input("Enter Your Body temp from 36 to 42 (in celsius ) : ",step=0,min_value=36, max_value=42 )


    new_entry = [[gender,age,height,weight,duration,heart_rate,body_temp]]
    # ctreating a button for prediction
    if st.button('Calories burned'):
        predicted_lr = prediction(new_entry)
        st.write("Calories burned:")
        st.success(predicted_lr)

if __name__=='__main__':
    main()
```

OUTPUT:

```
In [8]: import numpy as np
   ...: import streamlit as st
   ...: import pickle
   ...:
   ...: loaded_model = pickle.load(open('C:/Users/arunk/Desktop/
Final Mini/best_model_cb_pred.sav','rb'))
```

Terminal Execution:

- After running the code in spyder console. Now move on to the Anaconda Navigator Terminal.
- Enter the path of the python script that is use for deploy.
- Then enter the following comment after entering the path
- “streamlit run deploy.py”

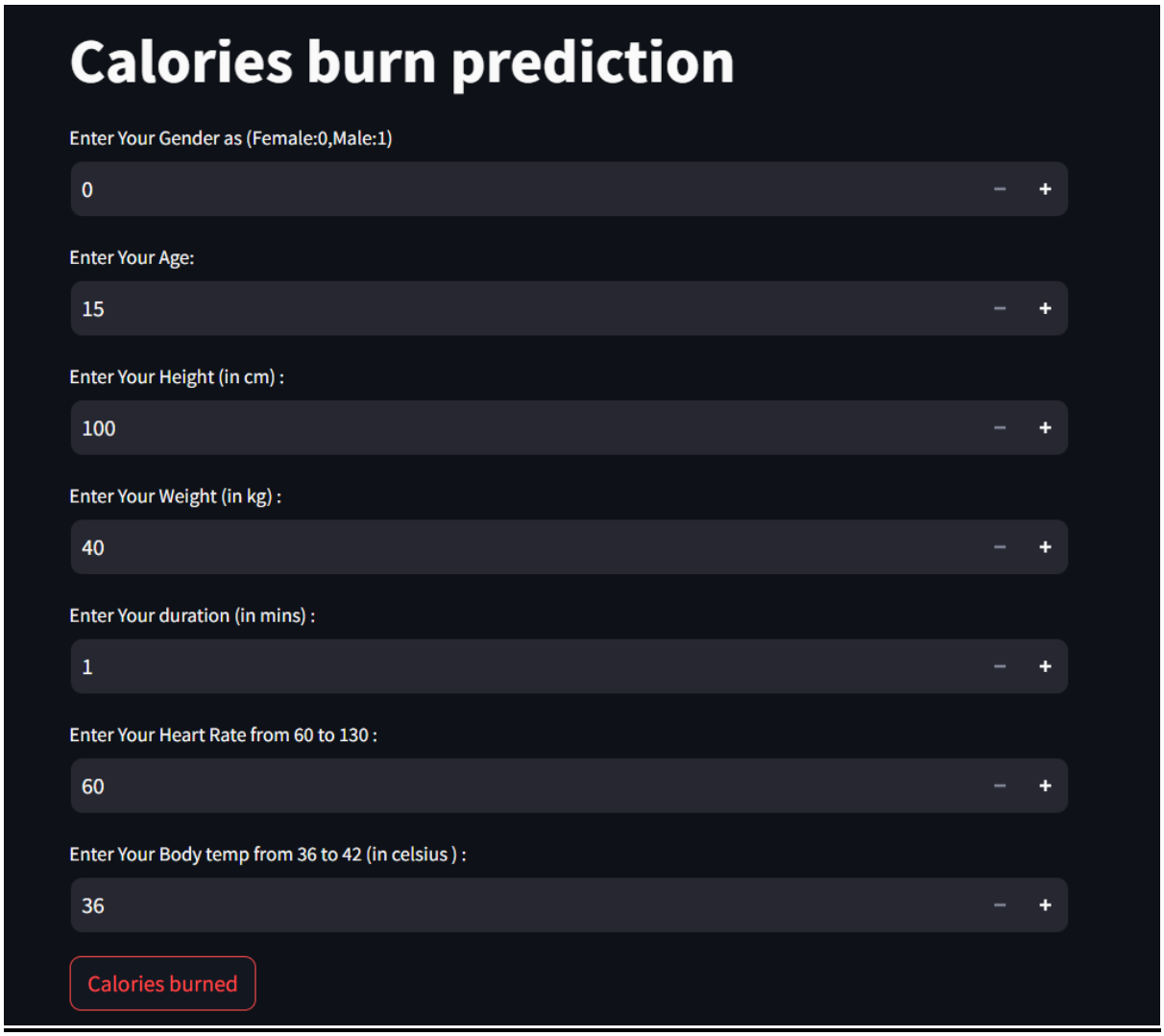
```
(base) C:\Users\arunk>cd C:\Users\arunk\Desktop\Final Mini

(base) C:\Users\arunk\Desktop\Final Mini>streamlit run deploy.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.19:8501
```

Webpage for Predicting Calories Burn:



CHAPTER VII

CONCLUSION

VII CONCLUSION

We have proposed a system for predicting the number of calories burned during physical activity that takes into account all of the relevant variables, including age, gender, height, weight, body temperature, heart rate, and the length of the activity. We have compared the accuracy of various regression models, including Random Forest, Ridge Regression, Lasso Regression, and XGBoost. Our system achieves the highest accuracy by using a combination of these models.

In addition to being accurate, our system is also user-friendly. We have developed a webpage for our system that allows users to easily make predictions, receive recommendations, and personalize their experience. The webpage also includes anomaly detection capabilities to identify unusual activity patterns.

Advantages:

- It uses a combination of machine learning models to achieve the highest accuracy in predicting calorie burn.
- It is user-friendly, with a webpage that allows users to easily make predictions, receive recommendations, and personalize their experience.
- It includes anomaly detection capabilities to identify unusual activity patterns.

We believe that our system is a valuable tool for people who are trying to track their calorie burn and make informed decisions about their health and fitness.

FURTHER ENHANCEMENTS

- This project will be further enhanced by adding the feature to get the food intake and calculate the calories of that and predict what exercise to do and how long to do it so as to maintain good health.

- Also, a mobile application will be developed invoking this project so that a large number of people can get benefitted from this and they can lead a healthy and happy life.

APPENDIX

USER DOCUMENTATION SOFTWARE DETAILS:

- The software used is Google Collaboratory.
- To Access Google Colab visit ‘colab.research.google.com’.
- Use your google account to sign in.
- To create a new notebook, click on “File” and then “New Notebook”.
- Next add your code, text in the respective dialogue boxes and run them.
- Your work is autosaved in google drive of the signed in account.
- To access the files in the google drive we can use the following code snippet
    - **‘drive.mount()’.**
- To upload files, we can use the following code snippet
    - **‘files.upload()’.**
- To install libraries, we can use
    - **‘!pip install’.**
- You can also share or download the colab file.
- To create a webpage using streamlit library in the spyder console.

USER MODULE CODE:

The users have to enter the following information.

- Their gender
- Their age
- Their height
- Their weight
- How long did they do the workout?
- Their average heartrate
- Their body temperature

BIBLIOGRAPHY

BOOK REFERENCES

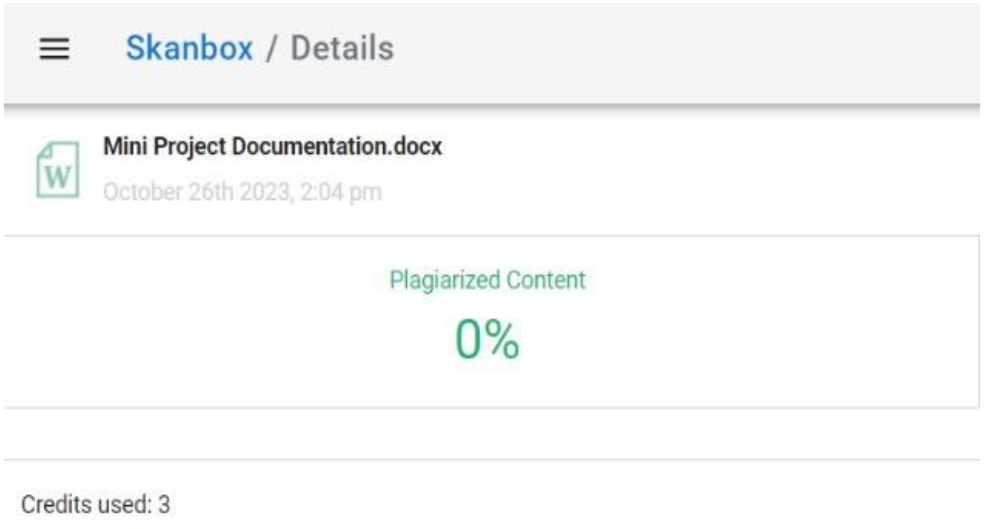
1. Introduction to Machine Learning with Python: A Guide for Data Scientists - Book by Andreas C. Müller and Sarah Guido
2. Python Machine Learning by Example - Book by Liu Yuxi
3. Machine Learning for Absolute Beginners: A Plain English - Book by Oliver Theobald
4. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython - Book by Wes McKinney

REFERENCES WEBSITE

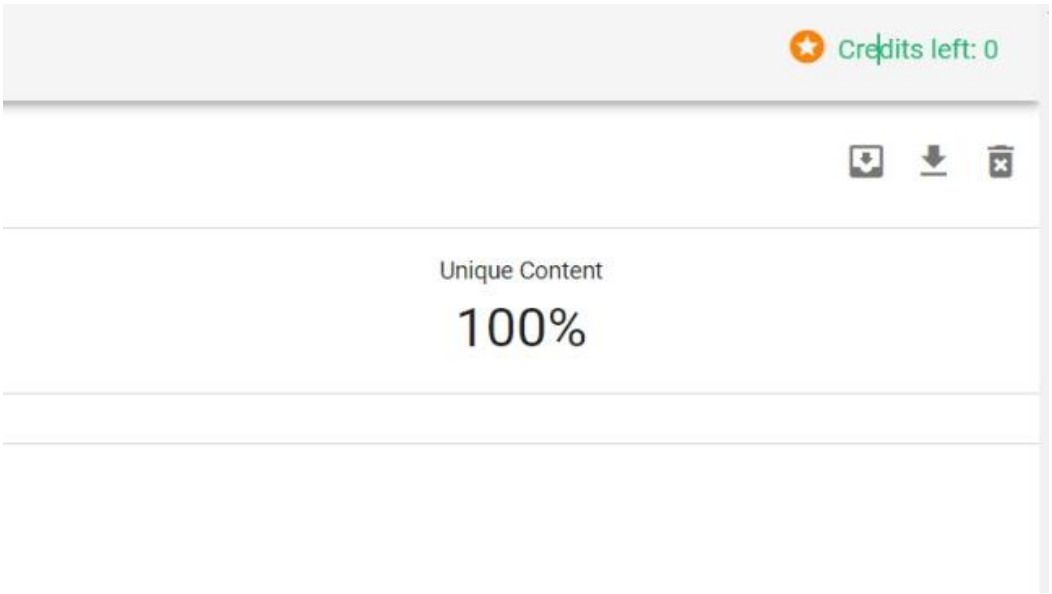
1. www.w3schools.com - Python Machine Learning
2. www.javatpoint.com - Machine Learning Tutorial
3. www.tutorialspoint.com - Machine Learning with Python Tutorial
4. https://www.geeksforgeeks.org/calories-burnt-prediction-using-machine-learning/ - Existing Research
5. https://www.geeksforgeeks.org/xgboost-for-regression/ - XGB Regressor Algorithm
6. https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/ - Streamlit Library

PLAGIARISM

Plagiarism Score:



Unique Score:



SOURCE CODE

Google Colab Console:

1.Importing the Dependencies

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Lasso,
Ridge
from sklearn.ensemble import RandomForestRegressor


from sklearn import metrics
from sklearn.metrics import mean_absolute_error as mae
from sklearn import set_config

set_config(display='diagram')
from google.colab import drive
drive.mount('/content/drive')

import pickle
```

2.Data Collection & Processing:

1.Load the dataset

```
#Load the dataset
dataset = pd.read_csv('/content/drive/MyDrive/MiniProject -SRM
Dataset/Exercise and Calories Burn.csv')
```

2.Summarize the dataset

```
#shape of the dataset
print("Dataset Shape:",dataset.shape)
print("***"*20)
#shape of the dataset
print("Dataset Shape:",dataset.shape)
print("***"*20)
```

```
#checking null values is present or not
print("The dataset have the null values or
not:\n",dataset.isnull().sum())
print("***"*20)
```

```
#check the unique values
print("The unique values present in each of the column as
follows:")
print(dataset.nunique(),"\n")
print("***"*20)
```

```
#information of data
print("The information of data followed below:")
print(dataset.info())
print("***"*20)
```

```
#get some statistical measures about the data
print("The statistical measures about the dataset as
follows:")
print(dataset.describe())
print("***"*20)
```

```
#view
print("The first 5 set of rows from the dataset shown below:")
print(dataset.head())
print("***"*20)
```

```
#Drop the unanted feature from the dataset
data = dataset
data = data.drop(columns=['User_ID'], axis=1)
print("Data Shape:",data.shape)
print(data.head())
```

3.Data Visualization and Distribution:

==> *Count Plot*

```
#count of gender in the dataset
sns.set()
plt.figure(figsize=(6,6))
ax=sns.countplot(x=data.Gender,palette = "Set1")
ax.bar_label(ax.containers[0], fmt='{:.0f}')
```

```
plt.title("Count of Gender")
plt.show()
```

==> **Pie Chart**

```
#pie chart of gender in %
plt.figure(figsize=(5,7))
plt.pie(data.Gender.value_counts(),labels=data.Gender.value_co
unts().index,autopct="%.3f%%",shadow=True)
plt.title("% of Gender")
plt.legend()
plt.show()
```

==> **Pair Plot**

```
#pairplot
sns.set()
plt.figure(figsize=(6,6))
sns.pairplot(data, hue='Calories_Burn', height=2)
plt.show()
```

==> **Distribution Plot**

```
#Distribution of Age
plt.figure(figsize=(6,6))
sns.distplot(x=data.Age,color='blue')
plt.title("Distribution of Age")
plt.show()
#Distribution of Height
plt.figure(figsize=(6,6))
sns.distplot(x=data.Height,color='blue')
plt.title("Distribution of Height")
plt.show()
```

```
#Distribution of Weight
plt.figure(figsize=(6,6))
sns.distplot(x=data.Weight,color='blue')
plt.title("Distribution of Weight")
plt.show()
```

```
#Distribution of Duration
plt.figure(figsize=(6,6))
sns.distplot(x=data.Duration,color='blue')
```

```
plt.title("Distribution of Duration")
plt.show()
```

```
#Distribution of Heart Rate
plt.figure(figsize=(6,6))
sns.distplot(x=data.Heart_Rate,color='blue')
plt.title("Distribution of Heart Rate")
plt.show()
```

```
#Distribution of Body Temperature
plt.figure(figsize=(6,6))
sns.distplot(x=data.Body_Temp,color='blue')
plt.title("Distribution of Body Temperature")
plt.show()
```

```
#Distribution of Calories Burn
plt.figure(figsize=(6,6))
sns.distplot(x=data.Calories_Burn,color='blue')
plt.title("Distribution of Calories Burn")
plt.show()
```

==> Heat Map

```
#correlation between the each column of the dataset
print("Correlation among the columns follows below:")
print(data.corr())

#heatmap
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True)
plt.title("Correlation between Columns")
```

```
#corr value round to -1 to 1
#heatmap
correlation = data.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f',
annot=True, annot_kws={'size':8}, cmap='Reds')
```

==> Scatter Plot

```
#Relation between Age and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Age,y=data.Calories_Burn,color='green')
plt.title("Relation between Age and Calories Burn")
plt.show()
```

```
#Relation between Height and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Height,y=data.Calories_Burn,color='gree
n')
plt.title("Relation between Height and Calories Burn")
plt.show()
```

```
#Relation between Weight and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Weight,y=data.Calories_Burn,color='gree
n')
plt.title("Relation between Weight and Calories Burn")
plt.show()
```

```
#Relation between Duration and Calories Burn
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Duration,y=data.Calories_Burn,color='gr
een')
plt.title("Relation between Duration and Calories Burn")
plt.show()
```

```
#Relation between Height and Weight
plt.figure(figsize=(6,6))
sns.scatterplot(x=data.Height,y=data.Weight)
plt.title("Relation between Height and Weight")
plt.show()
```

4.Model Selection

Label Encoding

```
#encoding
lb=LabelEncoder()
data['Gender']=lb.fit_transform(data['Gender'])
print(lb.classes_)
```

```
print("Gender:")
print("      Female --> 0 \n      Male   --> 1")
print("***"*20)
print(data.head())
print("***"*20)
```

Splitting Input and Output features from the dataset

```
#split the X and y values
X=data.iloc[:,:-1]
y=data.iloc[:,-1]
print("Input Features of the data:")
print(X)
print("***"*25)
print("Output Feature of the data:")
print(y)
print("***"*25)
```

Splitting the X & y into training data and Testing data

```
#split train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print("Input Shape:",X.shape)
print("Training Set Shape:",X_train.shape)
print("Testing Set Shape:",X_test.shape)
print("***"*25)
```

Rescale the features of X

```
#rescale feature
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5.Evaluate the model

```
from sklearn.metrics import mean_absolute_error as mae
models = [LinearRegression(), Ridge(), Lasso(),
RandomForestRegressor(), XGBRegressor() ]

for i in range(5):
    models[i].fit(X_train, y_train)
```

```
    print(f'{models[i]} : ')

    train_preds = models[i].predict(X_train)
    print('Training Error : ', mae(y_train, train_preds))

    y_pred = models[i].predict(X_test)
    print('Validation Error : ', mae(y_test, y_pred))
    print()
```

Optimisation

For Ridge Regression

```
#initialising Ridge() function
ridge = Ridge()
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0,
7.0, 8.0, 9.0,
                    10.0, 20, 50, 100, 500, 1000 ]}

# defining cross validation folds as 5
folds =5
grid_cv_model = GridSearchCV(estimator=ridge,
                        param_grid=params,
                        scoring='neg_mean_absolute_error',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)

# fiting GridSearchCV() with X_train and y_train
l2 = grid_cv_model.fit(X_train,y_train)
print("Best score for Ridge :",l2.best_score_)
print("The best alpha for Ridge is:",l2.best_params_)
```

For Lasso Regression

```
#initialising lasso() function
lasso = Lasso()
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0,
7.0, 8.0, 9.0,
                    10.0, 20, 50, 100, 500, 1000 ]}

# defining cross validation folds as 5
folds =5
grid_cv_model = GridSearchCV(estimator=lasso,
```

```
                        param_grid=params,
                        scoring='neg_mean_absolute_error',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)

# fiting GridSearchCV() with X_train and y_train
l1 = grid_cv_model.fit(X_train,y_train)
print("Best score for Lasso:",l1.best_score_)
print("The best alpha for Lasso is:",l1.best_params_)
```

For Random Forest Regressor

```
rfr = RandomForestRegressor(random_state=0)
parameters = {'n_estimators': [100, 150, 200, 250,
300],'max_depth': [1,2,3,4],}
grid_cv_model = GridSearchCV(rfr, parameters)

rfr_model = grid_cv_model.fit(X_train, y_train)
print("Best score for
RandomForestRegressor",rfr_model.best_score_)
print("The best alpha for RandomForestRegressor
is:",rfr_model.best_params_)
```

For XGB Regressor

```
xgb = XGBRegressor()
parameters = {'nthread':[4], #when use hyperthread, xgboost
may become slower
              'objective':['reg:linear'],
              'learning_rate': [.03, 0.05, .07], #so called
`eta` value
              'max_depth': [5, 6, 7],
              'min_child_weight': [4],
              'subsample': [0.7],
              'colsample_bytree': [0.7],
              'n_estimators': [500]}

grid_cv_model = GridSearchCV(xgb,
                        parameters,
                        cv = 2,
                        n_jobs = 5,
                        verbose=True)

xgb_model = grid_cv_model.fit(X_train, y_train)
```

```
print("Best score for
RandomForestRegressor",xgb_model.best_score_)
print("The best alpha for RandomForestRegressor
is:",xgb_model.best_params_)
```

Validate the model

```
from sklearn.metrics import mean_absolute_error as mae
models = [LinearRegression(), Ridge(alpha=10),
Lasso(alpha=0.05),RandomForestRegressor(),
          XGBRegressor(colsample_bytree=0.7,learning_rate=0.07
,max_depth=5,min_child_weight=4,n_estimators=500,nthread=4,obj
ective='reg:linear',subsample=0.7)]

for i in range(5):
    models[i].fit(X_train, y_train)

    print(f'{models[i]} : ')

    train_preds = models[i].predict(X_train)
    print('Training Error : ', mae(y_train, train_preds))

    y_pred = models[i].predict(X_test)
    print('Validation Error : ', mae(y_test, y_pred))
    print()
```

Best Model

```
#bestmodel
xgb =
XGBRegressor(colsample_bytree=0.7,learning_rate=0.07,max_depth
=5,min_child_weight=4,n_estimators=500,nthread=4,objective='re
g:linear',subsample=0.7)
model = xgb.fit(X_train, y_train)
```

To predict the unseen data

```
#unseen data
print("Chosse Your Gender according to the Refrence Given
Below.")
print("Female --> 0\nMale   --> 1")
gender = int(input("Enter Your Gender: "))
print('*'*60)

print("Choose The Age From 17.")
```

```
age = int(input("Enter Your Age: "))
print('*'*60)

height = float(input("Enter Your Height (in cm) : "))
print('*'*60)

weight = float(input("Enter Your Weight (in kg) : "))
print('*'*60)

duration = int(input("Enter Your duration (in mins) : "))
print('*'*60)

heart_rate = int(input("Enter Your Heart Rate from 60 to 130 :
" ))
print('*'*60)

body_temp = int(input("Enter Your Body temp from 36 to 42 (in
celsius ) : " ))
print('*'*60)
```

```
#predict calories burn
new_entry =
[[gender,age,height,weight,duration,heart_rate,body_temp]]
Predict_lr = model.predict(new_entry)
print("Calories burn by Your above details:",Predict_lr)
```

```
#deploy model
filename = "best_model_cb_pred.sav"
pickle.dump(xgb, open(filename, 'wb'))
```

Spyder Console:

```
import numpy as np
import streamlit as st
import pickle
loaded_model = pickle.load(open('C:/Users/arunk/Desktop/Final
Mini/best_model_cb_pred.sav','rb'))
def prediction(new_entry):
    Predict_lr = loaded_model.predict(new_entry)
    outp = float(Predict_lr)
    return (outp)

def main():
    st.title("Calories burn prediction")
    gender = st.number_input("Enter Your Gender as (Female:0,Male:1)
",step=0,min_value=0, max_value=1)
    age = st.number_input("Enter Your Age: ",step=0,min_value=15,
max_value=100)
    height = st.number_input("Enter Your Height (in cm) :
",step=0,min_value=100, max_value=200)
    weight = st.number_input("Enter Your Weight (in kg) :
",step=0,min_value=40, max_value=200)
    duration = st.number_input("Enter Your duration (in mins) :
",step=0,min_value=1, max_value=180)
    heart_rate = st.number_input("Enter Your Heart Rate from 60 to 130
: ",step=0,min_value=60, max_value=130 )
    body_temp = st.number_input("Enter Your Body temp from 36 to 42 (in
celsius ) : ",step=0,min_value=36, max_value=42 )


    new_entry =
[[gender,age,height,weight,duration,heart_rate,body_temp]]
    # ctreating a button for prediction
    if st.button('Calories burned'):
        predicted_lr = prediction(new_entry)
        st.write("Calories burned:")
        st.success(predicted_lr)

if __name__=='__main__':
 main()
```

---------- **END** ----------