PROJECT REPORT ON:

CALORIES BURNT

Submitted By:

GOPIKA A NAIR

Reg NO:200011024269

MSC DATA ANALYTICS

BASELIUS COLLEGE, KOTTAYAM

MAHATMA GANDHI UNIVERSITY

INDEX

CALORIES BURNT PREDICTION

ABSTRACT

All you have to do to maintain a healthy weight is ensure that the number of calories you ingest stays the same as the number of calories you expend. If you take in more calories, or energy, than you use, you gain weight; if the output is greater than the input, you lose it. This project is about calorie prediction with machine learning using python. We will predict calorie based on same features. Calories in the foods we eat provide energy in the form of heat so that our bodies can function. This means that we need to eat a certain amount of calories just to sustain life. But if we take in too many calories just sustain life. But if we take in too many calories, then we risk gaining weight. So, there is need to burn Calories, for burning calories we doing exercises and more for knowing how much calories we have burnt. Today we are going to build a machine learning model that predicts calories based on some data.

INTRODUCTION

In this fast and busy schedule life, people are not giving importance to the quality of food they are eating. They tend to neglect their eating patterns and habits. The fast-food consumption rate is alarmingly high and this consequently has led to the intake of unhealthy food. This leads to various health issues such as obesity, diabetes, an increase in blood pressure etc. Hence it has become very essential for people to have a good balanced nutritional healthy diet. There are many applications which are booming to help people so that they can have control over their diet and hence can reduce weight or they can help them to keep them fit and healthy. This project focuses on the calories burned in accordance with the age, gender, height, weight, duration provided, heart rate during the exercise period and body temperature. It introduces the topic of linear regression and its predicting capability with the effectiveness from the data provided. This research helps in providing the benefits of a machine learning algorithm over predicting the calories burned.

The project is sub-divided into following section. These are :

1.Loading necessary libraries.

2.Loading Dataset from a CSV file.

3.Summarization of Data to understand Dataset (Descriptive Statistics)

4.Visualization of Data to understand Dataset (Plots, Graphs etc.).

5.Data pre-processing and Data transformation.

6.Applying different learning algorithms on the training dataset.

7.Evaluating the performance of the fitted model using R2 score.

DATA

There are two dataset csv files which should be uploaded for processing. The dataset has 15000 rows and 9 columns. From this data, calories burnt should be predicted using some explanatory variables like age, gender, user id, height, weight, duration, heart rate, body temperature. The data is pre-processed before using it for training different models.

Dataset:

1. https://github.com/Kasra1377/calories-burned-prediction/blob/main/calories.csv
2. https://github.com/Kasra1377/calories-burned-prediction/blob/main/exercise.csv

The necessary libraries are imported and the data is read.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()
import statsmodels.api as sm
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
calories=pd.read_csv(r"C:\Users\chithu\Downloads\calories.csv")
calories
```

Out[2]:

	User_ID	Calories
0	14733363	231.0
1	14861698	66.0
2	11179863	26.0
3	16180408	71.0
4	17771927	35.0
...	...	...
14995	15644082	45.0
14996	17212577	23.0
14997	17271188	75.0
14998	18643037	11.0
14999	11751526	98.0

15000 rows × 2 columns

In [3]:
```
exercise=pd.read_csv(r"C:\Users\chithu\Downloads\exercise.csv")
exercise
```

Out[3]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
0	14733363	male	68	190.0	94.0	29.0	105.0	40.8
1	14861698	female	20	166.0	60.0	14.0	94.0	40.3
2	11179863	male	69	179.0	79.0	5.0	88.0	38.7
3	16180408	female	34	179.0	71.0	13.0	100.0	40.5
4	17771927	female	27	154.0	58.0	10.0	81.0	39.8
...	...	...	...	...	...	...	...	...
14995	15644082	female	20	193.0	86.0	11.0	92.0	40.4
14996	17212577	female	27	165.0	65.0	6.0	85.0	39.2
14997	17271188	female	43	159.0	58.0	16.0	90.0	40.1
14998	18643037	male	78	193.0	97.0	2.0	84.0	38.3
14999	11751526	male	63	173.0	79.0	18.0	92.0	40.5

15000 rows × 8 columns

In [4]:
```
df=pd.merge(exercise,calories)
df
```

Out[4]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	male	68	190.0	94.0	29.0	105.0	40.8	231.0
1	14861698	female	20	166.0	60.0	14.0	94.0	40.3	66.0
2	11179863	male	69	179.0	79.0	5.0	88.0	38.7	26.0
3	16180408	female	34	179.0	71.0	13.0	100.0	40.5	71.0
4	17771927	female	27	154.0	58.0	10.0	81.0	39.8	35.0
...	...	...	...	...	...	...	...	...	...
14995	15644082	female	20	193.0	86.0	11.0	92.0	40.4	45.0
14996	17212577	female	27	165.0	65.0	6.0	85.0	39.2	23.0
14997	17271188	female	43	159.0	58.0	16.0	90.0	40.1	75.0
14998	18643037	male	78	193.0	97.0	2.0	84.0	38.3	11.0
14999	11751526	male	63	173.0	79.0	18.0	92.0	40.5	98.0

15000 rows × 9 columns

The two datasets are merged into a dataset and this is a multiple linear regression problem.

EXPLORATORY DATA ANALYSIS (EDA)

Exploratory data analysis is an approach of analysing basic features of datasets to summarize their main characteristics often using statistical graphics and other data visualization methods.

This process includes descriptive statistical methods like describe( ), info( ), data visualization techniques, correlation etc.

```
In [5]: df.shape

Out[5]: (15000, 9)

In [6]: df.columns

Out[6]: Index(['User_ID', 'Gender', 'Age', 'Height', 'Weight', 'Duration',
               'Heart_Rate', 'Body_Temp', 'Calories'],
              dtype='object')

In [7]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 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  float64
 4   Weight      15000 non-null  float64
 5   Duration    15000 non-null  float64
 6   Heart_Rate  15000 non-null  float64
 7   Body_Temp   15000 non-null  float64
 8   Calories    15000 non-null  float64
dtypes: float64(6), int64(2), object(1)
memory usage: 1.1+ MB
```

In [65]:
```
df.describe()
```

Out[65]:

	User_ID	Duration	Heart_Rate	Body_Temp	Calories
count	1.500000e+04	15000.000000	15000.000000	15000.000000	15000.000000
mean	1.497736e+07	15.530600	95.516400	40.025453	89.480333
std	2.872851e+06	8.319203	9.579658	0.779230	62.361520
min	1.000116e+07	1.000000	67.000000	37.100000	1.000000
25%	1.247419e+07	8.000000	88.000000	39.600000	35.000000
50%	1.499728e+07	16.000000	96.000000	40.200000	79.000000
75%	1.744928e+07	23.000000	103.000000	40.600000	138.000000
max	1.999965e+07	30.000000	125.000000	41.500000	289.000000

In [60]:
```
df.head()
```

Out[60]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	male	60-69	181-195	40-41	29.0	105.0	40.8	231.0
1	14861698	female	20-29	166-180	40-41	14.0	94.0	40.3	66.0
2	11179863	male	60-69	166-180	40-41	5.0	88.0	38.7	26.0
3	16180408	female	30-39	166-180	40-41	13.0	100.0	40.5	71.0
4	17771927	female	20-29	151-165	40-41	10.0	81.0	39.8	35.0

Correlation:

In [69]:
```
df.corr()
```

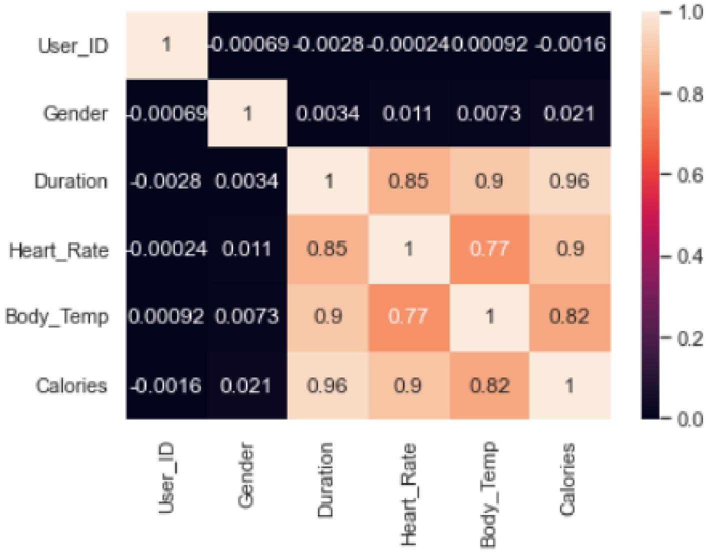
Out[69]:

	User_ID	Gender	Duration	Heart_Rate	Body_Temp	Calories
User_ID	1.000000	-0.000687	-0.002751	-0.000235	0.000923	-0.001601
Gender	-0.000687	1.000000	0.003440	0.011335	0.007264	0.021435
Duration	-0.002751	0.003440	1.000000	0.852808	0.903167	0.955432
Heart_Rate	-0.000235	0.011335	0.852808	1.000000	0.771574	0.896238
Body_Temp	0.000923	0.007264	0.903167	0.771574	1.000000	0.824852
Calories	-0.001601	0.021435	0.955432	0.896238	0.824852	1.000000

Correlation Heatmap:

In [70]:
```
sns.heatmap(df.corr(),annot=True)
```

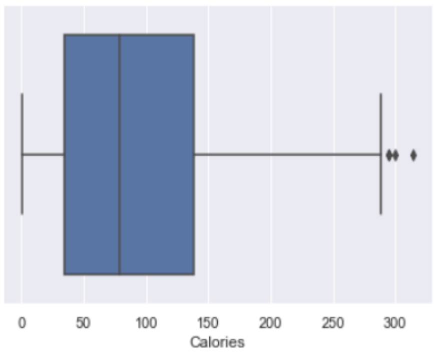
Out[70]: <AxesSubplot:>

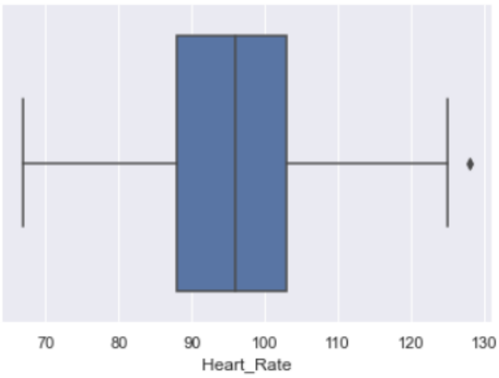


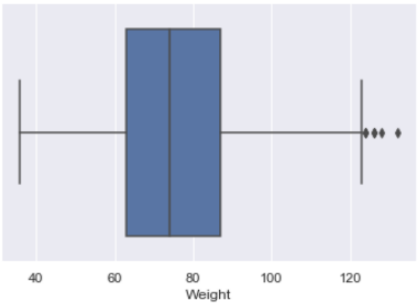
Box Plot :

Box plot is a type of chart often used to visually show the distribution of numerical data and skewness through displaying the data quartiles and averages.

```
In [11]:  data=df.drop('Gender',axis=1)
          for i in data:
              sns.boxplot(x=data[i])
              plt.show()
```







We have treated the outliers that was present in the variables.

Count Plot:

It is a method used to show the counts of observations in each categorical bin using bars.

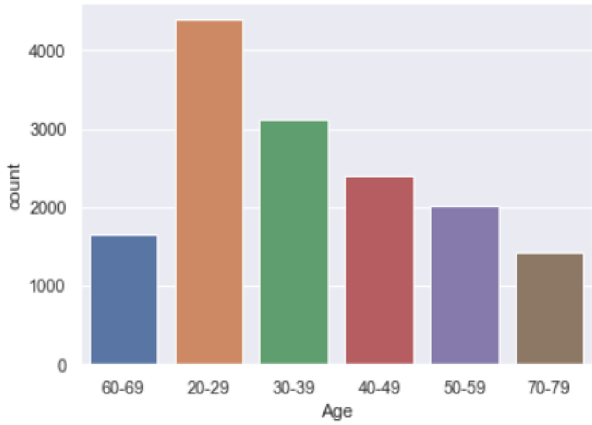
```
In [51]:
#dividing into classes
#Age
age=[]
for i in df.Age:
    if i<=29:
        age.append("20-29")
    elif i>29 and i<=39:
        age.append("30-39")
    elif i>39 and i<=49:
        age.append("40-49")
    elif i>49 and i<=59:
        age.append("50-59")
    elif i>59 and i<=69:
        age.append("60-69")
    else:
        age.append("70-79")
```

```
In [52]:
df.Age=age
```

```
In [53]:
df.Age.value_counts()
```

```
Out[53]:
20-29    4387
30-39    3115
40-49    2394
50-59    2011
60-69    1664
70-79    1429
Name: Age, dtype: int64
```

```
In [62]:
sns.countplot(df['Age'])
plt.show()
```



From the count plot of age, we came to know that more calories are burnt for people who belong to the age category of 20-29 and the least is burnt among senior citizens who belong to the age category of 70-79.

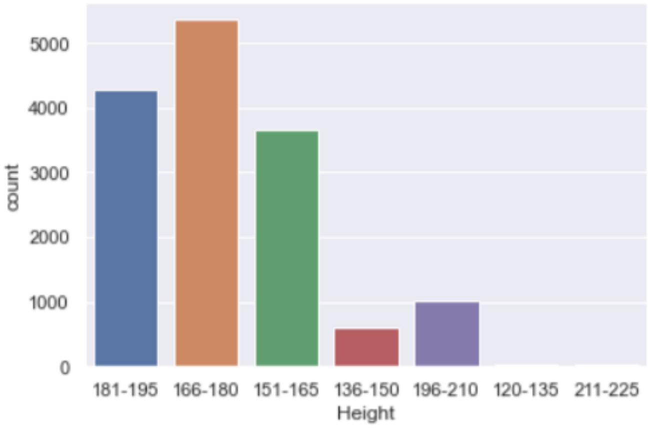
In [55]:
```
#Height
height=[]
for i in df.Height:
    if i<=135:
        height.append("120-135")
    elif i>135 and i<=150:
        height.append("136-150")
    elif i>150 and i<=165:
        height.append("151-165")
    elif i>165 and i<=180:
        height.append("166-180")
    elif i>180 and i<=195:
        height.append("181-195")
    elif i>195 and i<=210:
        height.append("196-210")
    else:
        height.append("211-225")
```

In [56]:
```
df.Height=height
```

In [57]:
```
df.Height.value_counts()
```

Out[57]:
```
166-180    5366
181-195    4292
151-165    3669
196-210    1013
136-150     607
211-225      33
120-135      20
Name: Height, dtype: int64
```

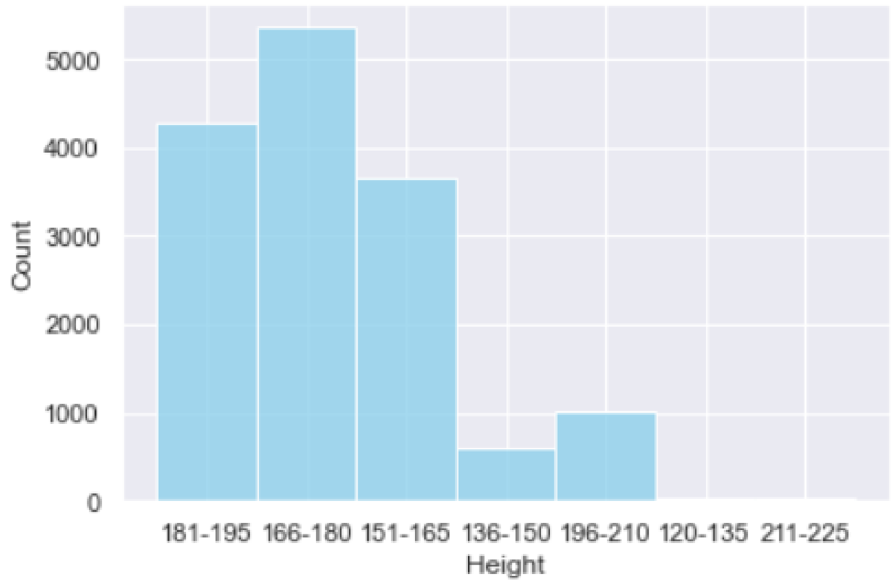
In [100]:
```
##plt.subplots(figsize=(30,30))
sns.countplot(df['Height'])
plt.show()
```



From the count plot of height we came to know that more calories are burnt for people who have height between 166-180 and the least is burnt among the height of 136-150.

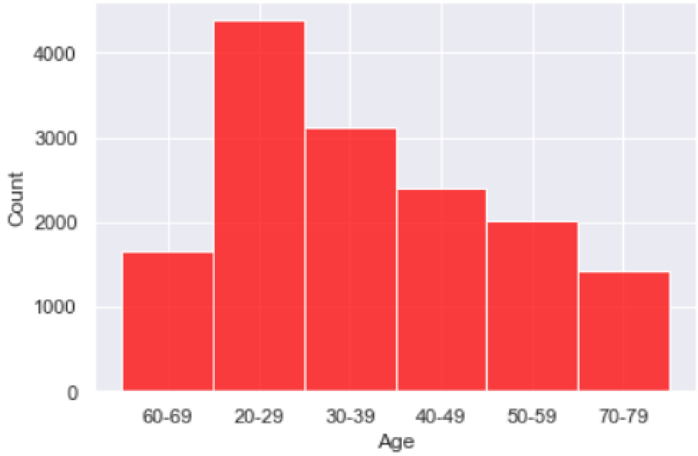
Hist plot:

In [119]:
```
sns.histplot(df['Height'],color='skyblue');
```



The Height data is normally distributed.

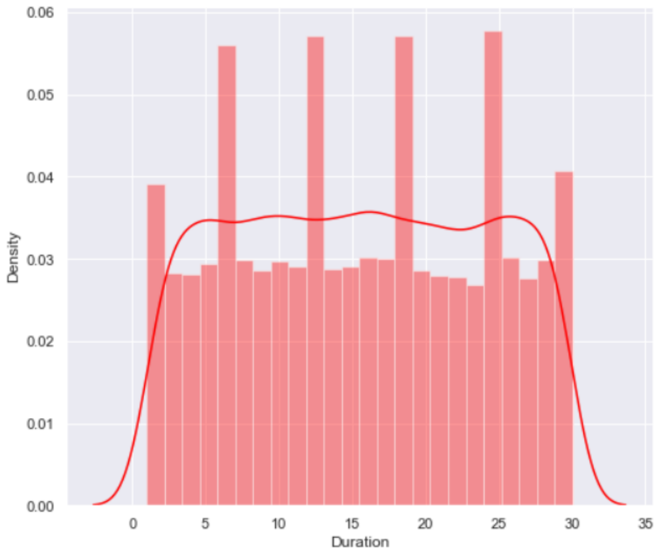
In [120]:
```
sns.histplot(df['Age'],color='red');
```



For age, the data is positively skewed.

Dist Plot:

In [112]:
```
plt.figure(figsize=[8,7])
sns.distplot(df['Duration'],color='red');
plt.show()
```



The duration of data is low peaked.

DATA PRE-PROCESSING

In this step we are treating our outliers, label encode the object column, data splitting etc.

❖ Outlier Treatment

In [13]:
```
#WEIGHT
iqr=stats.iqr(df.Weight,interpolation='midpoint')
iqr
```
Out[13]: 24.0

In [14]:
```
q1=df.Weight.quantile(0.25)
q3=df.Weight.quantile(0.75)
min_lim=q1-1.5*iqr
max_lim=q3+1.5*iqr
```

In [15]:
```
min_lim
```
Out[15]: 27.0

In [16]:
```
max_lim
```
Out[16]: 123.0

In [17]:
```
df.loc[df['Weight']>max_lim]
```
Out[17]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
1909	14839747	male	67	212.0	124.0	16.0	87.0	40.1	94.0
6711	12623721	male	40	218.0	132.0	1.0	88.0	37.8	5.0
10362	17352972	male	22	222.0	128.0	6.0	84.0	39.3	20.0
12189	10775847	male	21	219.0	126.0	15.0	97.0	40.4	78.0
13276	13066276	male	55	217.0	126.0	17.0	91.0	40.4	101.0
13806	10934514	male	43	214.0	124.0	17.0	103.0	40.7	121.0

In [18]:
```
df.loc[df['Weight']>max_lim,'Weight']=np.median(df.Weight)
```

In [19]:
```
df.loc[df['Weight']>max_lim]
```
Out[19]:

User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories

In [20]:
```
#HEART_RATE
iqr=stats.iqr(df.Heart_Rate,interpolation='midpoint')
iqr
```

Out[20]: 15.0

In [21]:
```
q1=df.Heart_Rate.quantile(0.25)
q3=df.Heart_Rate.quantile(0.75)
min_lim=q1-1.5*iqr
max_lim=q3+1.5*iqr
```

In [22]:
```
min_lim
```

Out[22]: 65.5

In [23]:
```
max_lim
```

Out[23]: 125.5

In [24]:
```
df.loc[df['Heart_Rate']>max_lim]
```

Out[24]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
9981	12114332	male	32	188.0	91.0	30.0	128.0	40.9	289.0

In [25]:
```
df.loc[df['Heart_Rate']>max_lim,'Heart_Rate']=np.median(df.Heart_Rate)
```

In [26]:
```
df.loc[df['Heart_Rate']>max_lim]
```

Out[26]:

User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories

In [34]:
```
#CALORIES
iqr=stats.iqr(df.Calories,interpolation='midpoint')
iqr
```
Out[34]: 103.0

In [35]:
```
q1=df.Calories.quantile(0.25)
q3=df.Calories.quantile(0.75)
min_lim=q1-1.5*iqr
max_lim=q3+1.5*iqr
```
In [36]:
```
min_lim
```
Out[36]: -119.5

In [37]:
```
max_lim
```
Out[37]: 292.5

In [38]:
```
df.loc[df['Calories']>max_lim]
```
Out[38]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
428	13079051	male	75	199.0	103.0	28.0	123.0	40.5	314.0
3357	17825244	male	65	189.0	98.0	27.0	125.0	40.9	295.0
6240	17545969	male	69	193.0	90.0	29.0	121.0	41.1	300.0
13871	10784322	male	75	178.0	76.0	29.0	120.0	40.8	295.0

In [39]:
```
df.loc[df['Calories']>max_lim,'Calories']=np.median(df.Calories)
```
In [40]:
```
df.loc[df['Calories']>max_lim]
```
Out[40]:

User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories

❖ Label Encoding

In [66]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
le.fit(["male","female"])
gender=le.transform(df.Gender)
gender
```

Out[66]:
```
array([1, 0, 1, ..., 0, 1, 1])
```

In [67]:
```
df.Gender=gender
df
```

Out[67]:

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	1	60-69	181-195	40-41	29.0	105.0	40.8	231.0
1	14861698	0	20-29	166-180	40-41	14.0	94.0	40.3	66.0
2	11179863	1	60-69	166-180	40-41	5.0	88.0	38.7	26.0
3	16180408	0	30-39	166-180	40-41	13.0	100.0	40.5	71.0
4	17771927	0	20-29	151-165	40-41	10.0	81.0	39.8	35.0
...	...	...	...	...	...	...	...	...	...
14995	15644082	0	20-29	181-195	40-41	11.0	92.0	40.4	45.0
14996	17212577	0	20-29	151-165	40-41	6.0	85.0	39.2	23.0
14997	17271188	0	40-49	151-165	40-41	16.0	90.0	40.1	75.0
14998	18643037	1	70-79	181-195	40-41	2.0	84.0	38.3	11.0
14999	11751526	1	60-69	166-180	40-41	18.0	92.0	40.5	98.0

15000 rows × 9 columns

In [71]:

```
X=df.iloc[:,5:8]
y=df.Calories
X
```

Out[71]:

	Duration	Heart_Rate	Body_Temp
0	29.0	105.0	40.8
1	14.0	94.0	40.3
2	5.0	88.0	38.7
3	13.0	100.0	40.5
4	10.0	81.0	39.8
...	...	...	...
14995	11.0	92.0	40.4
14996	6.0	85.0	39.2
14997	16.0	90.0	40.1
14998	2.0	84.0	38.3
14999	18.0	92.0	40.5

15000 rows × 3 columns

In [72]:

```
y
```

Out[72]:

```
0        231.0
1         66.0
2         26.0
3         71.0
4         35.0
         ...
14995     45.0
14996     23.0
14997     75.0
14998     11.0
14999     98.0
Name: Calories, Length: 15000, dtype: float64
```

In [100…

```
import statsmodels.api as sm
model=sm.OLS(y,X).fit()
model.summary()
```

Out[100…

OLS Regression Results

Dep. Variable:	Calories	**R-squared (uncentered):**	0.981
Model:	OLS	**Adj. R-squared (uncentered):**	0.981
Method:	Least Squares	**F-statistic:**	2.544e+05
Date:	Thu, 19 May 2022	**Prob (F-statistic):**	0.00
Time:	08:31:30	**Log-Likelihood:**	-62044.
No. Observations:	15000	**AIC:**	1.241e+05
Df Residuals:	14997	**BIC:**	1.241e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Duration	5.5591	0.025	221.855	0.000	5.510	5.608
Heart_Rate	2.0561	0.024	83.926	0.000	2.008	2.104
Body_Temp	-4.8289	0.051	-94.264	0.000	-4.929	-4.728

Omnibus:	3054.266	**Durbin-Watson:**	1.989
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	14475.454
Skew:	0.915	**Prob(JB):**	0.00
Kurtosis:	7.451	**Cond. No.**	50.9

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

❖ Data Splitting

In [74]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
model=lr.fit(xtrain,ytrain)
```

APPLYING DIFFERENT REGRESSION ALGORITHMS

- **Linear Regression**

  Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you are using to predict is called dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

```
In [74]: from sklearn.linear_model import LinearRegression
         lr=LinearRegression()
         model=lr.fit(xtrain,ytrain)

In [75]: model.score(xtrain,ytrain)

Out[75]: 0.9443885200620931

In [76]: pred_train=model.predict(xtrain)
         pred_train

Out[76]: array([  7.06643992,  66.32229605, 122.57254319, ..., 143.50955142,
                 71.81001673, -12.77477445])

In [77]: from sklearn.metrics import r2_score

In [78]: accuracy_train=r2_score(pred_train,ytrain)
         accuracy_train

Out[78]: 0.9411137696440333

In [79]: pred_test=model.predict(xtest)
         pred_test

Out[79]: array([ 46.86560167,  11.74959885, 118.64563944, ..., 101.83418735,
                180.97885627,  26.91973093])

In [80]: accuracy_test=r2_score(pred_test,ytest)
         accuracy_test

Out[80]: 0.9450202581504005

In [81]: print("Training set accuracy = " ,accuracy_train)
         print("Testing set accuracy = " ,accuracy_test)

         Training set accuracy =  0.9411137696440333
         Testing set accuracy =  0.9450202581504005
```

- **Random Forest Regressor**

  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. The main principle behind the ensemble methods is that weak learners can form strong learners. Random forest operates by constructing multiple decision trees at training time. These decision trees are independently trained on bootstrapped datasets. The final predicted value is calculated by taking the mean of predictions by all the individual trees.

```
In [82]: from sklearn.ensemble import RandomForestRegressor
         Regressor=RandomForestRegressor(n_estimators=10,random_state=0)
         model_reg=Regressor.fit(xtrain,ytrain)

In [83]: ypred_reg=Regressor.predict(xtest)
         ypred_reg

Out[83]: array([ 40.25     ,  21.08     , 115.425    , ...,  92.7697619,
                176.5     ,  24.8      ])

In [84]: from sklearn.metrics import r2_score
         acc=r2_score(ypred_reg,ytest)
         acc

Out[84]: 0.9499447452703316

In [85]: print("Accuracy = ",acc)

         Accuracy =  0.9499447452703316
```

- **ANN Regressor**

  Regression ANNs predict an output variable as a function of the inputs. The input features (independent variables) can be categorical or numeric types, however, for regression ANNs, we require a numeric dependent variable.

```
In [86]: import tensorflow as tf
```

```
In [87]: nn=tf.keras.models.Sequential()
```

```
In [88]: nn.add(tf.keras.layers.Dense(units=100,activation="relu"))
```

```
In [89]: nn.add(tf.keras.layers.Dense(units=100,activation="relu"))
```

```
In [90]: nn.add(tf.keras.layers.Dense(units=1,activation="linear"))
```

```
In [91]: nn.compile(optimizer="adam",loss="mean_squared_error",metrics=tf.keras.metrics.RootMeanSquaredError())
```

```
In [92]: nn.fit(xtrain,ytrain,batch_size=32,epochs=25)
```

```
Epoch 1/25
352/352 [==============================] - 2s 2ms/step - loss: 1142.8251 - root_mean_squared_error: 33.8057
Epoch 2/25
352/352 [==============================] - 1s 2ms/step - loss: 208.2551 - root_mean_squared_error: 14.4310
Epoch 3/25
352/352 [==============================] - 1s 2ms/step - loss: 194.3219 - root_mean_squared_error: 13.9399
Epoch 4/25
352/352 [==============================] - 1s 2ms/step - loss: 184.7072 - root_mean_squared_error: 13.5907
Epoch 5/25
352/352 [==============================] - 1s 2ms/step - loss: 181.8635 - root_mean_squared_error: 13.4857
Epoch 6/25
352/352 [==============================] - 1s 2ms/step - loss: 179.7817 - root_mean_squared_error: 13.4083
Epoch 7/25
352/352 [==============================] - 1s 2ms/step - loss: 181.2779 - root_mean_squared_error: 13.4639
Epoch 8/25
352/352 [==============================] - 1s 2ms/step - loss: 177.6073 - root_mean_squared_error: 13.3269
Epoch 9/25
352/352 [==============================] - 1s 2ms/step - loss: 179.1458 - root_mean_squared_error: 13.3845
Epoch 10/25
352/352 [==============================] - 1s 2ms/step - loss: 175.2813 - root_mean_squared_error: 13.2394
Epoch 11/25
352/352 [==============================] - 1s 2ms/step - loss: 173.4569 - root_mean_squared_error: 13.1703
```

```
Epoch 12/25
352/352 [==============================] - 1s 2ms/step - loss: 176.5900 - root_mean_squared_error: 13.2887
Epoch 13/25
352/352 [==============================] - 1s 2ms/step - loss: 176.2236 - root_mean_squared_error: 13.2749
Epoch 14/25
352/352 [==============================] - 1s 2ms/step - loss: 171.8179 - root_mean_squared_error: 13.1079
Epoch 15/25
352/352 [==============================] - 1s 2ms/step - loss: 173.3864 - root_mean_squared_error: 13.1676
Epoch 16/25
352/352 [==============================] - 1s 2ms/step - loss: 173.8104 - root_mean_squared_error: 13.1837
Epoch 17/25
352/352 [==============================] - 1s 2ms/step - loss: 170.5985 - root_mean_squared_error: 13.0613
Epoch 18/25
352/352 [==============================] - 1s 2ms/step - loss: 173.0049 - root_mean_squared_error: 13.1531
Epoch 19/25
352/352 [==============================] - 1s 2ms/step - loss: 171.5513 - root_mean_squared_error: 13.0978
Epoch 20/25
352/352 [==============================] - 1s 2ms/step - loss: 168.4760 - root_mean_squared_error: 12.9798
Epoch 21/25
352/352 [==============================] - 1s 2ms/step - loss: 174.1380 - root_mean_squared_error: 13.1961
Epoch 22/25
352/352 [==============================] - 1s 2ms/step - loss: 168.6684 - root_mean_squared_error: 12.9872
Epoch 23/25
352/352 [==============================] - 1s 2ms/step - loss: 171.0532 - root_mean_squared_error: 13.0787
Epoch 24/25
352/352 [==============================] - 1s 2ms/step - loss: 172.8069 - root_mean_squared_error: 13.1456
Epoch 25/25
352/352 [==============================] - 1s 2ms/step - loss: 168.4773 - root_mean_squared_error: 12.9799
```

Out[92]:

```
<keras.callbacks.History at 0x208b3e72520>
```

In [93]:

```
pred_ann=nn.predict(xtest)
pred_ann
```

Out[93]:

```
array([[ 46.330772],
       [ 21.717344],
       [114.76123 ],
       ...,
       [ 99.055756],
       [191.31848 ],
       [ 27.330671]], dtype=float32)
```

In [94]:

```
acc_ann=r2_score(pred_ann,ytest)
print("Accuracy = ",acc_ann*100)
```

```
Accuracy =  96.01072626681564
```

ACCURACY:

NO	ALGORITHM	TEST ACCURACY
1	Linear Regression	94.50202581504004
2	Random Forest	94.99447452703316
3	ANN	96.01072626681564

CONCLUSION

The calories burnt dataset has been analysed by different machine learning techniques. The overall accuracies obtained were in the range 94.50-96.01%. We trained our models with 3 features. Among all the models ANN Regressor has the highest accuracy of 96.01%.

REFERENCE

[1]https://www.kaggle.com/fmendes/fmendesdat263xdemos

[2]https://machinelearningmastery.com/linear-regression-for-machine-learning

/[3]https://machinelearningmastery.com/xgboost-for-regression/

[4]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5496172

[5]https://devhadvani.github.io/calorie.htm