Active Burn Forecast

- User_ID: (used only for tracking individuals, non-predictive)
- Gender: (used to capture differences in calorie burn between males and females)
- Age: (age influences metabolism and calorie burn rates)
- Height: (taller individuals may burn more calories)
- Weight: (heavier individuals generally burn more calories during activity)
- Duration: (longer exercise duration typically results in more calories burned)
- Heart_Rate: (higher heart rates indicate more intense activity, leading to higher calorie burn)
- Body_Temp: (body temperature can correlate with the intensity of exercise and calorie burn)
- Calories: (the target variable representing the number of calories burned during the activity, which the model aims to predict)

Step 1 : import libraries & create dataframe

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

#from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
#from sklearn.linear_model import Ridge,Lasso
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from statsmodels.stats.outliers_influence import
variance_inflation_factor
import pickle

import warnings
from warnings import filterwarnings
filterwarnings("ignore")

sns.set()

#Load the Calories dataset
df1 = pd.read_csv("calories.csv")
df1.head()
```

```
   User_ID  Calories
0  14733363     231.0
1  14861698      66.0
2  11179863      26.0
3  16180408      71.0
4  17771927      35.0
```

```
df1.shape
```

```
(15000, 2)
```

```
#Load the Exercise Dataset
df2 = pd.read_csv("exercise.csv")
df2.head()
```

```
   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
```

```
df2.shape
```

```
(15000, 8)
```

Now Concatenate both the Dataframe i.e df1 and df2

```
df = pd.concat([df2,df1["Calories"]],axis=1)
```

```
df.head()
```

```
   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
```

```
   Calories
0     231.0
1      66.0
2      26.0
3      71.0
4      35.0
```

Step 2 : Data cleaning (handling the null values and droping unwanted columns)

```
df.info()
```

```
<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  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: 1.0+ MB
```

```
df.describe()
```

```
            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
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
```

```
df.isnull().sum()
```

```
User_ID       0
Gender        0
Age           0
Height        0
Weight        0
Duration      0
Heart_Rate    0
Body_Temp     0
Calories      0
dtype: int64
```

```
# drop User_ID column because this is not required from Main Dataframe
itself

df.drop(columns = ["User_ID"],axis=1,inplace =True)

df.head()
```

```
   Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
Calories
0    male   68   190.0    94.0      29.0       105.0       40.8
231.0
1  female   20   166.0    60.0      14.0        94.0       40.3
66.0
2    male   69   179.0    79.0       5.0        88.0       38.7
26.0
3  female   34   179.0    71.0      13.0       100.0       40.5
71.0
4  female   27   154.0    58.0      10.0        81.0       39.8
35.0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 8 columns):
```

```
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Gender      15000 non-null  object
 1   Age         15000 non-null  int64
 2   Height      15000 non-null  float64
 3   Weight      15000 non-null  float64
 4   Duration    15000 non-null  float64
 5   Heart_Rate  15000 non-null  float64
 6   Body_Temp   15000 non-null  float64
 7   Calories    15000 non-null  float64
dtypes: float64(6), int64(1), object(1)
memory usage: 937.6+ KB
```

step 3 : Encoding

Separate Categorical and Numerical Features

1. Categorical Feature

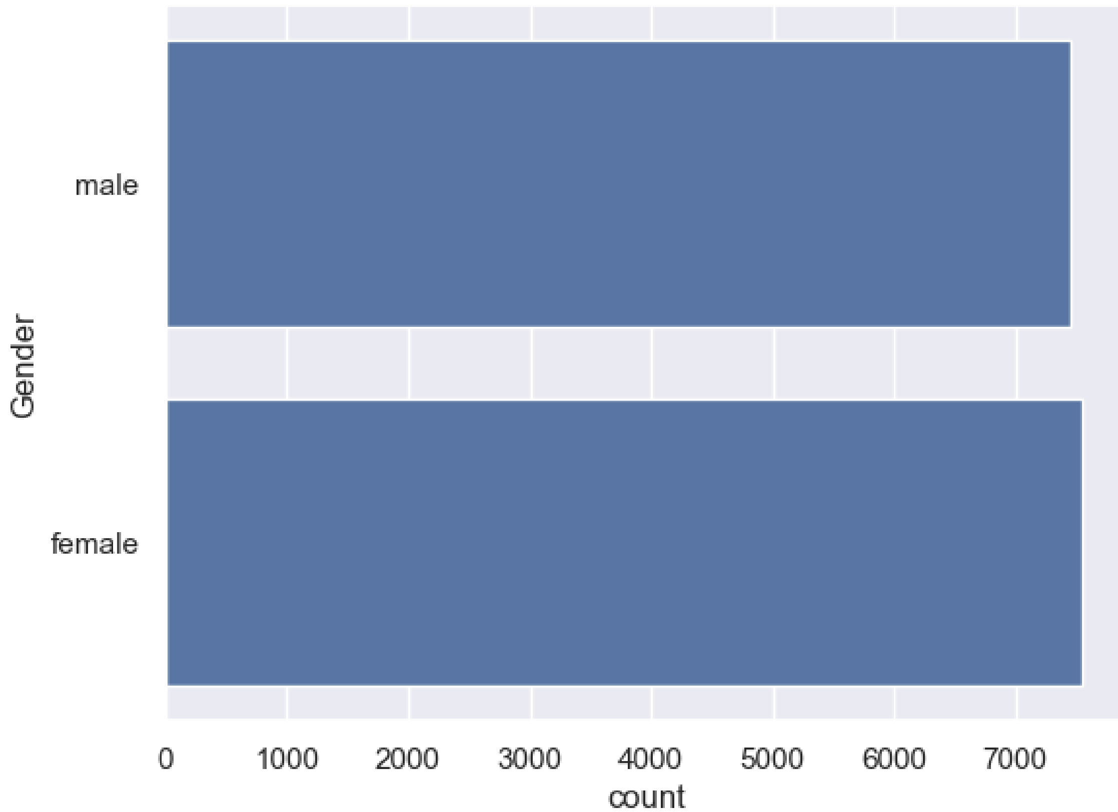
```
#Fatching Categorical Data
cat_col=[col for col in df.columns if df[col].dtype=='O'] #--
>Object-"o"
cat_col
```

```
['Gender']
```

```
df["Gender"].value_counts()
```

```
Gender
female    7553
male      7447
Name: count, dtype: int64
```

```
# plotting the gender column in count plot
sns.countplot(df['Gender'])
plt.show()
```




```
categorical = df[cat_col]
categorical.head()

   Gender
0    male
1  female
2    male
3  female
4  female

categorical = pd.get_dummies(categorical["Gender"],drop_first=True)

categorical

        male
0       True
1      False
2       True
3      False
4      False
...      ...
14995  False
14996  False
14997  False
14998   True
```

```
14999    True

[15000 rows x 1 columns]
```

2.Numerical Features

```
Num_col = [col for col in df.columns if df[col].dtype != "O"]
Num_col
```

```
['Age', 'Height', 'Weight', 'Duration', 'Heart_Rate', 'Body_Temp', 
'Calories']
```

```
df[Num_col].shape
```

```
(15000, 7)
```

```
Numerical = df[Num_col]
Numerical.head()
```

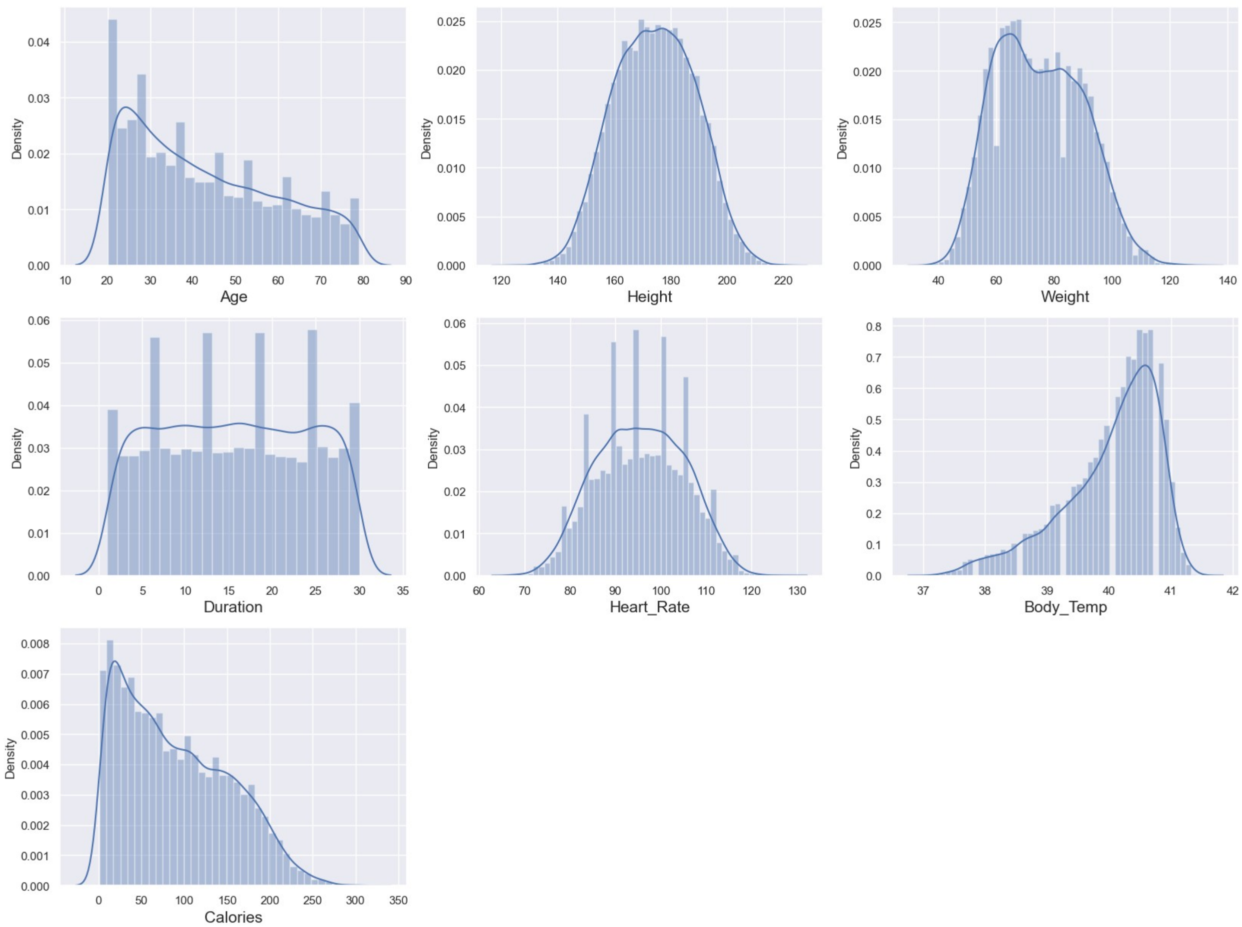
	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	68	190.0	94.0	29.0	105.0	40.8	231.0
1	20	166.0	60.0	14.0	94.0	40.3	66.0
2	69	179.0	79.0	5.0	88.0	38.7	26.0
3	34	179.0	71.0	13.0	100.0	40.5	71.0
4	27	154.0	58.0	10.0	81.0	39.8	35.0

Step 4 : EDA

```
Numerical.shape
```

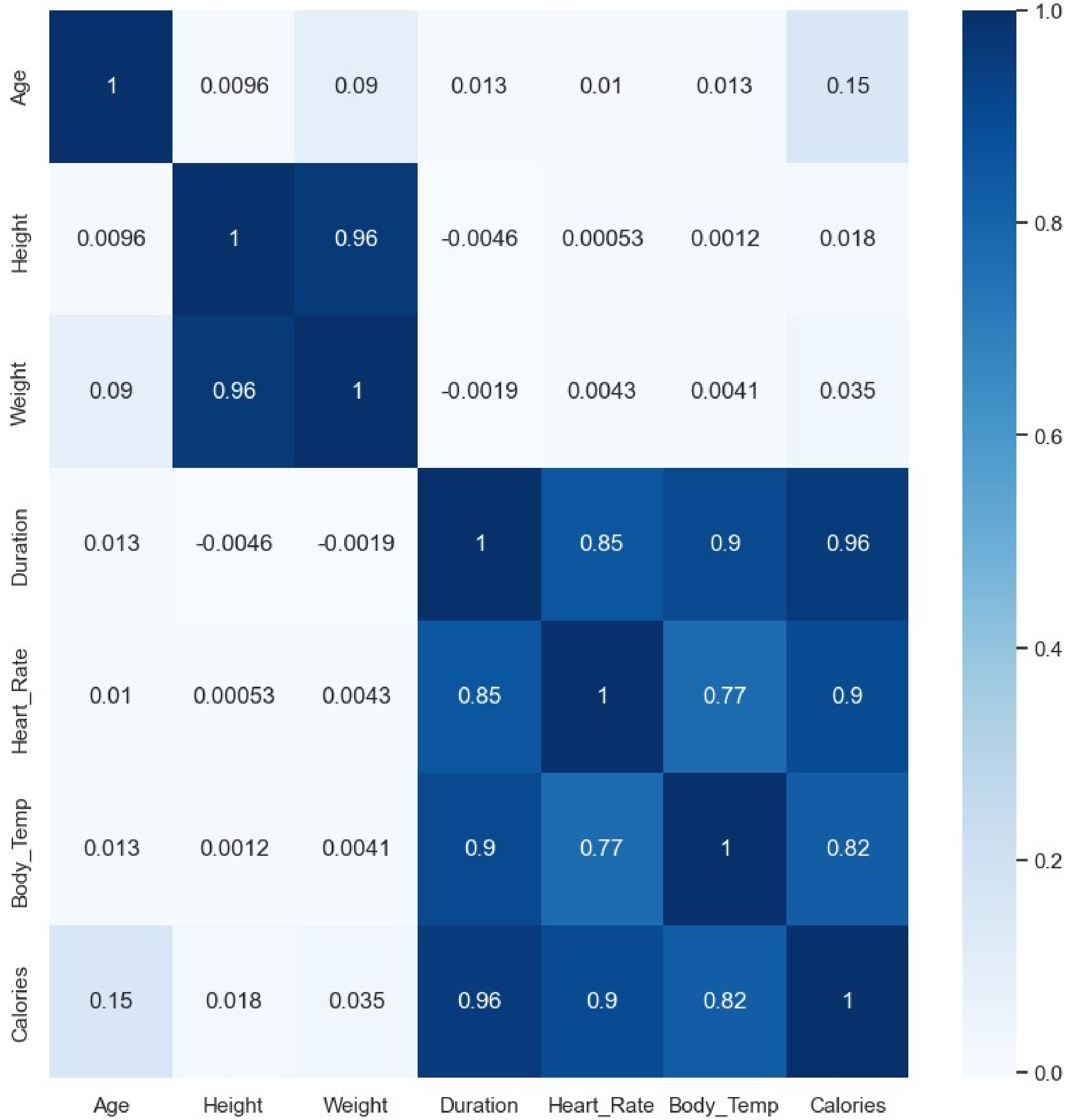
```
(15000, 7)
```

```
plt.figure(figsize=(20,15))
plotnumber = 1

for column in Numerical:
  if plotnumber <= 8:
    ax = plt.subplot(3,3,plotnumber)
    sns.distplot(Numerical[column])
    plt.xlabel(column,fontsize=15)
  plotnumber+=1
plt.show()
```




```python
# constructing a heatmap to understand the correlation

plt.figure(figsize=(10,10))
sns.heatmap(Numerical.corr(), cmap='Blues',annot = True)
```

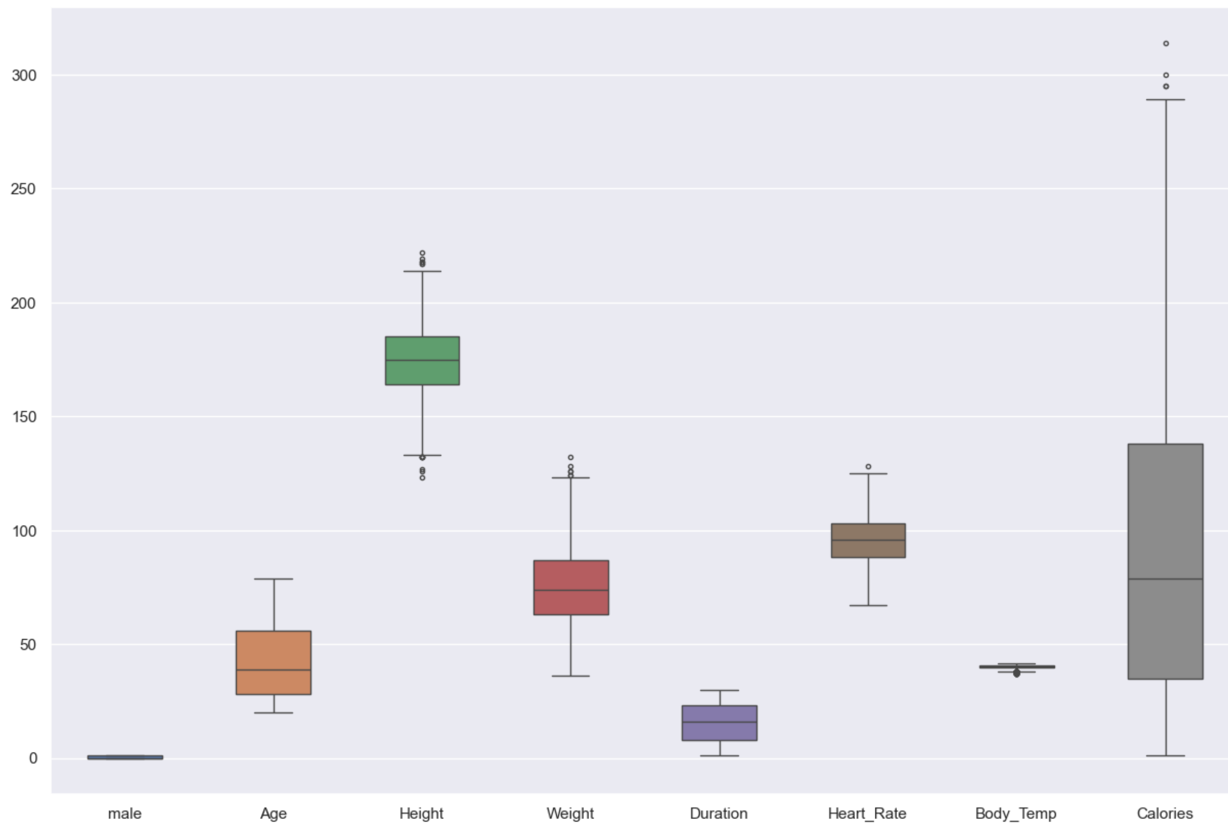
```
<Axes: >
```



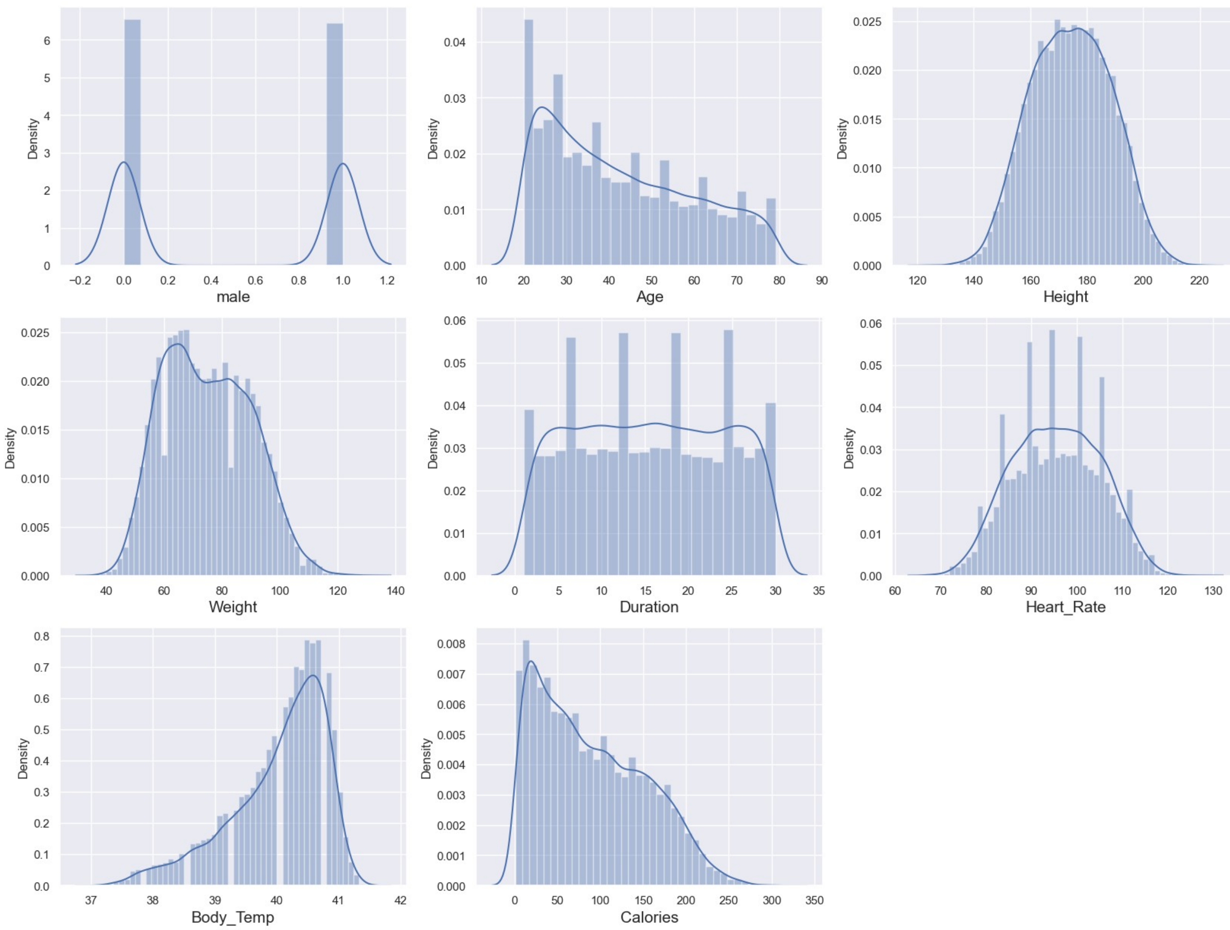
Concatenate Categorical and Numerical

```
data = pd.concat([categorical,Numerical],axis=1)

data.head()

    male  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
Calories
0   True   68   190.0    94.0      29.0       105.0       40.8
231.0
1  False   20   166.0    60.0      14.0        94.0       40.3
```

```
66.0
2   True   69   179.0   79.0    5.0    88.0    38.7
26.0
3  False   34   179.0   71.0   13.0   100.0    40.5
71.0
4  False   27   154.0   58.0   10.0    81.0    39.8
35.0

fig,ax = plt.subplots(figsize = (15,10))
sns.boxplot(data=data,width = 0.5,fliersize = 3,ax=ax)

<Axes: >
```




```
plt.figure(figsize=(20,15))
plotnumber = 1

for column in data:
  if plotnumber <= 8:
    ax = plt.subplot(3,3,plotnumber)
    sns.distplot(data[column])
    plt.xlabel(column,fontsize=15)
  plotnumber+=1
plt.show()
```




```
data.columns
```

```
Index(['male', 'Age', 'Height', 'Weight', 'Duration', 'Heart_Rate',
       'Body_Temp', 'Calories'],
      dtype='object')
```

Step 5 : Splitting Data into X and Y

```
X = data.drop(columns = ["Calories"],axis = 1)
y = data["Calories"]

X.head()
```

	male	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
0	True	68	190.0	94.0	29.0	105.0	40.8
1	False	20	166.0	60.0	14.0	94.0	40.3
2	True	69	179.0	79.0	5.0	88.0	38.7
3	False	34	179.0	71.0	13.0	100.0	40.5
4	False	27	154.0	58.0	10.0	81.0	39.8

```
y.head()
```

```
0    231.0
1     66.0
2     26.0
3     71.0
4     35.0
Name: Calories, dtype: float64
```

```
# Split the Data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size =
0.2,random_state=1)
```

step 6 : Build a Model

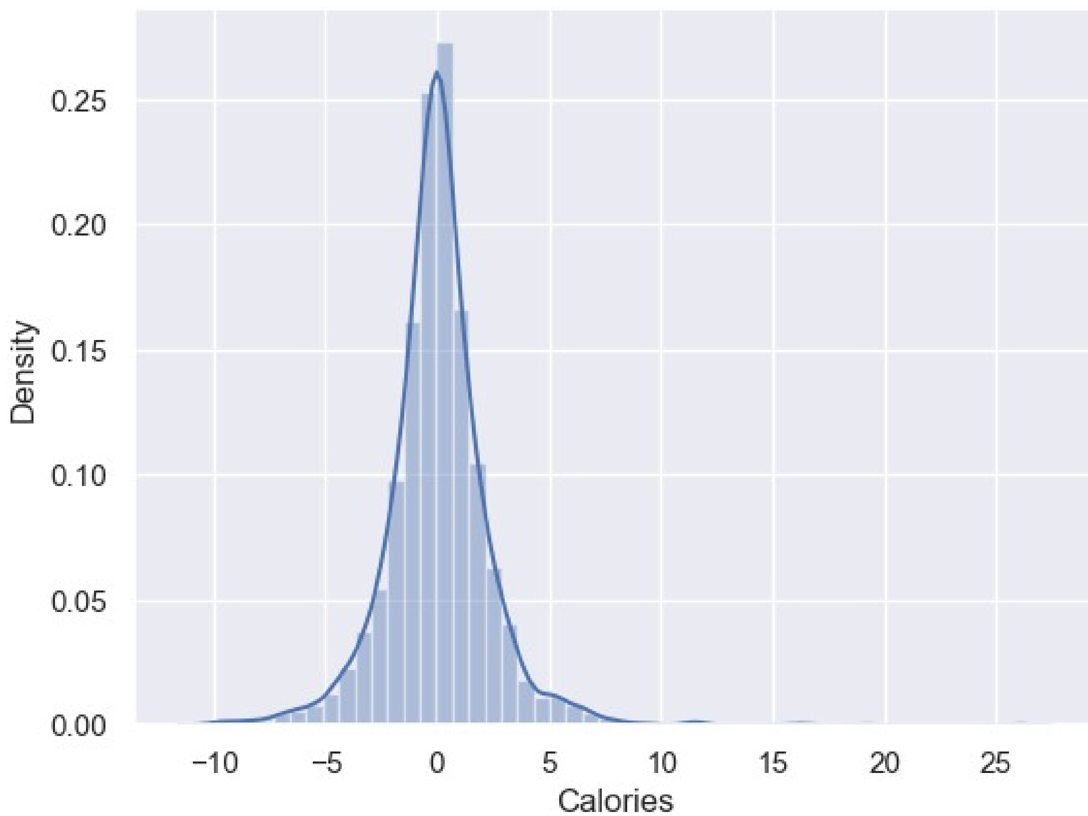
```
#from sklearn import metrics
def predict(ml_model):
    model=ml_model.fit(X_train,y_train)
    print('Score : {}'.format(model.score(X_train,y_train)))
    y_prediction=model.predict(X_test)
    print('predictions are: \n {}'.format(y_prediction))
    print('\n')

    r2_score=metrics.r2_score(y_test,y_prediction)
    print('r2 score: {}'.format(r2_score))

    print('MAE:',metrics.mean_absolute_error(y_test,y_prediction))
    print('MSE:',metrics.mean_squared_error(y_test,y_prediction))

print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,y_prediction))
)

    sns.distplot(y_test-y_prediction)
```

XGB Regressor

```
regression = predict(XGBRegressor())
regression
```

```
Score : 0.9995380557081355
predictions are:
 [197.06581   70.867226 196.99498  ...  29.043041 104.09284   14.61472
]


r2 score: 0.9986863132331905
MAE: 1.5521575984954834
```

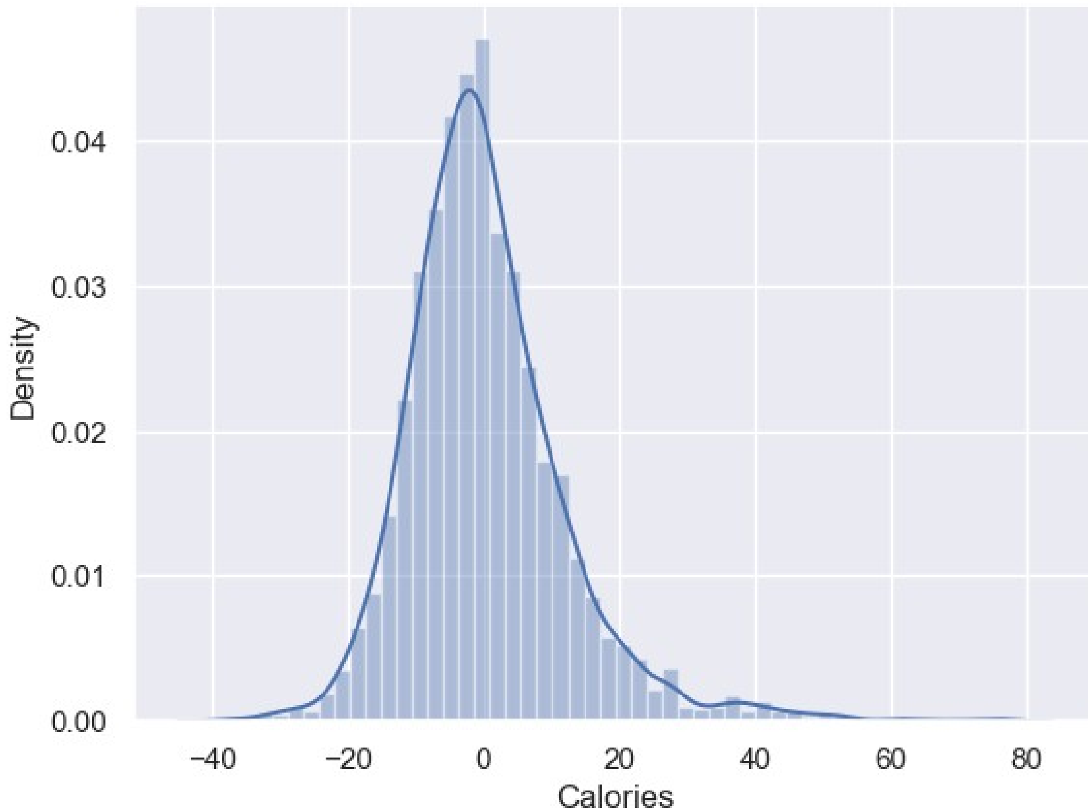
```
MSE: 5.2744122853837005
RMSE: 2.2966088664340956
```



Linear Regression

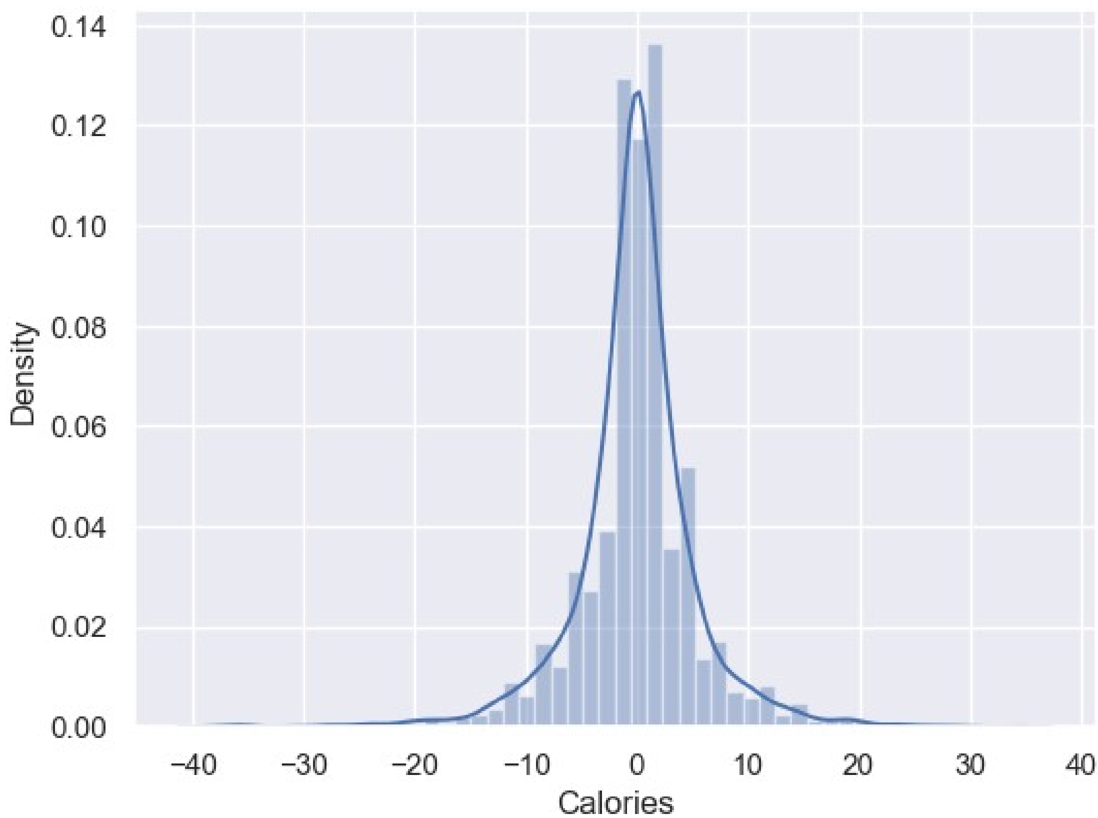
```
predict(LinearRegression())
```

```
Score : 0.9675925554735781
predictions are:
 [198.81182363  80.43555305 194.40940033 ...  22.14745631 118.63504926
 -11.98134672]


r2 score: 0.9655977245826503
MAE: 8.479071745987948
MSE: 138.12408611460904
RMSE: 11.752620393538159
```



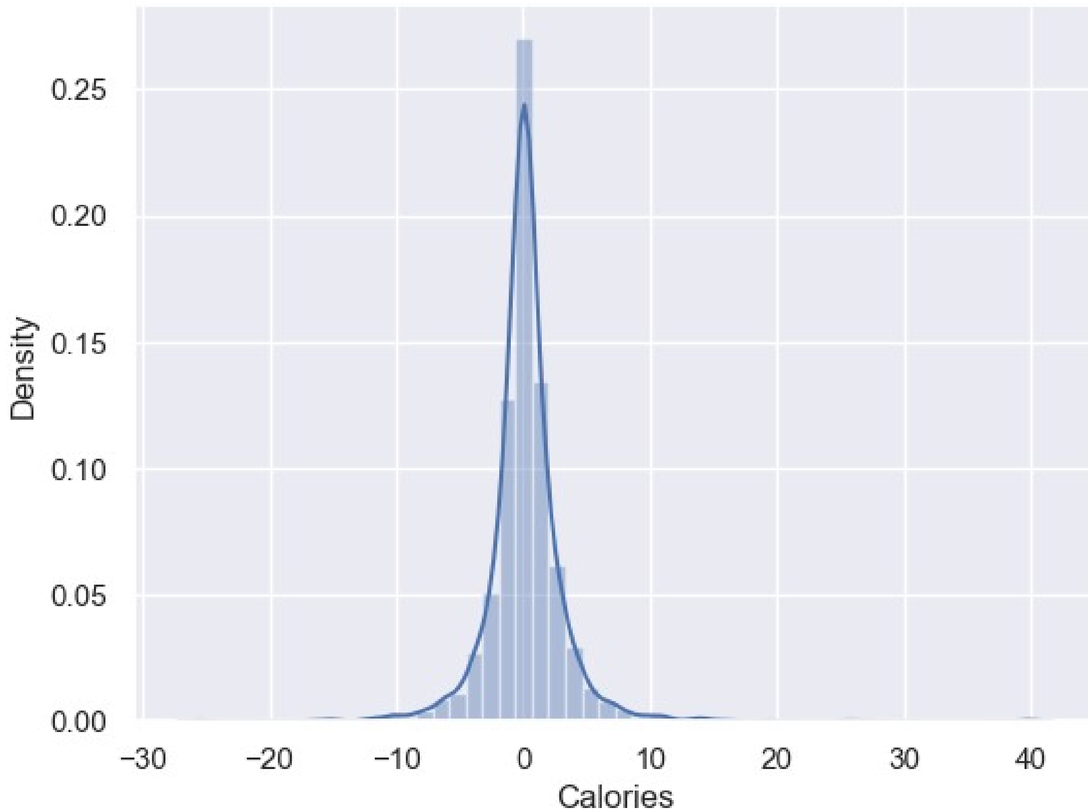
DecisionTree Regression

```
predict(DecisionTreeRegressor())

Score : 1.0
predictions are:
 [194.  75. 206. ...  30. 109.  13.]


r2 score: 0.9922832124228775
MAE: 3.544
MSE: 30.982666666666667
RMSE: 5.566207565898586
```

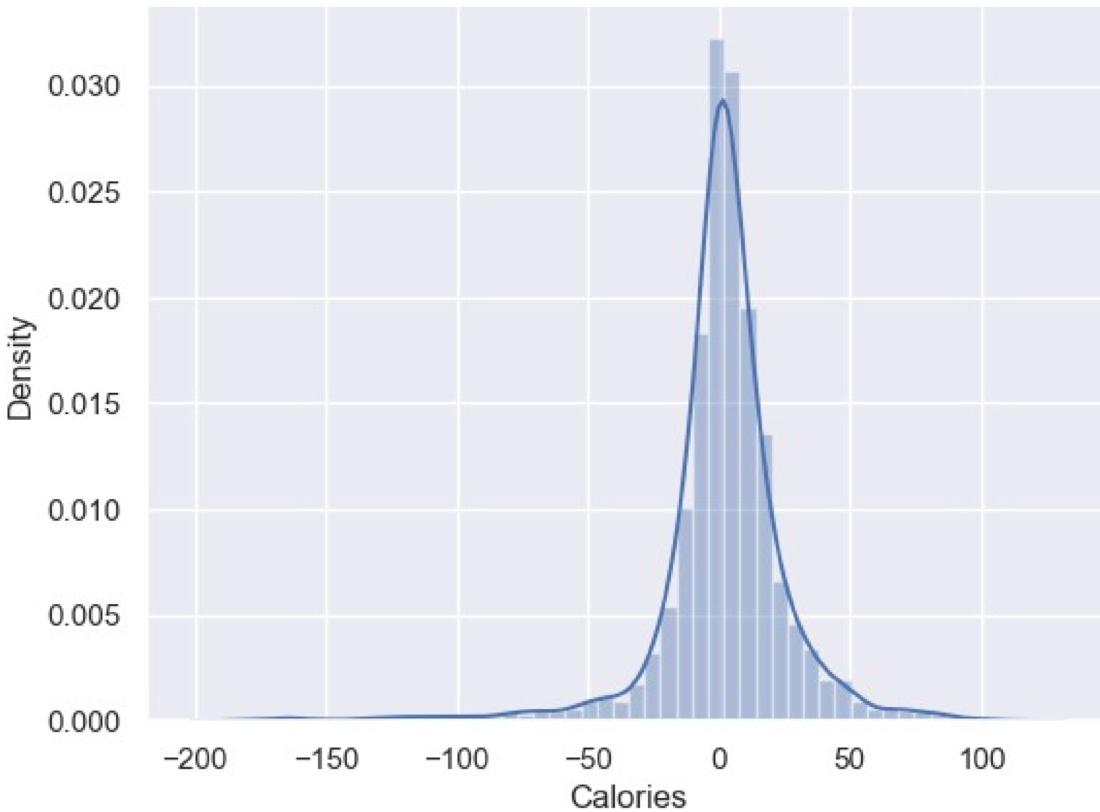


RandomForest Regression

```
predict(RandomForestRegressor())

Score : 0.9996811600697983
predictions are:
 [197.68  66.23 196.22 ...  27.31 111.5   13.96]


r2 score: 0.9976866738104277
MAE: 1.8274800000000002
MSE: 9.287934066666667
RMSE: 3.047611206611937
```




```
# Logestic Regression

predict(LogisticRegression())

Score : 0.05358333333333333
predictions are:
 [181.  60. 198. ...  43. 110.   3.]


r2 score: 0.836138812847468
MAE: 15.460333333333333
MSE: 657.8976666666666
RMSE: 25.6495159148602
```


0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
−200
−150
−100
−50
0
50
100
Calories