In [1]:

```
# import required libraries

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.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()
```

In [2]:

```
#Load the Calories dataset
df1 = pd.read_csv("C:\\Users\\RAHUL KUMAR UPADHYAY\\Downloads\\calories.csv")
df1.head()
```

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

In [3]:

```
df1.shape
```

Out[3]:

```
(15000, 2)
```

In [4]:

```
#Load the Exercise Dataset
df2 = pd.read_csv("C:\\Users\\RAHUL KUMAR UPADHYAY\\Downloads\\exercise.csv")
df2.head()
```

Out[4]:

	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

In [5]:
```
df2.shape
```

Out[5]:
```
(15000, 8)
```

Now Concatenate both the Dataframe i.e df1 and df2

In [9]:
```
df = pd.concat([df2,df1["Calories"]],axis=1)
```

In [10]:
```
df.head()
```

Out[10]:

	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

In [11]:
```
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
```

In [13]:
```
df.describe()
```

Out[13]:

	User_ID	Age	Height	Weight	Duration	Heart_Rate	
count	1.500000e+04	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15
mean	1.497736e+07	42.789800	174.465133	74.966867	15.530600	95.518533	
std	2.872851e+06	16.980264	14.258114	15.035657	8.319203	9.583328	
min	1.000116e+07	20.000000	123.000000	36.000000	1.000000	67.000000	
25%	1.247419e+07	28.000000	164.000000	63.000000	8.000000	88.000000	
50%	1.499728e+07	39.000000	175.000000	74.000000	16.000000	96.000000	
75%	1.744928e+07	56.000000	185.000000	87.000000	23.000000	103.000000	
max	1.999965e+07	79.000000	222.000000	132.000000	30.000000	128.000000	

In [14]:

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

Out[14]:

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

In [15]:

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

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

In [16]:

```
df.head()
```

Out[16]:

	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

In [17]:

```
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
```

Separate Categorical and Numerical Features

1. Categorical Feature

In [27]:

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

Out[27]:

```
['Gender']
```

In [28]:

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

Out[28]:

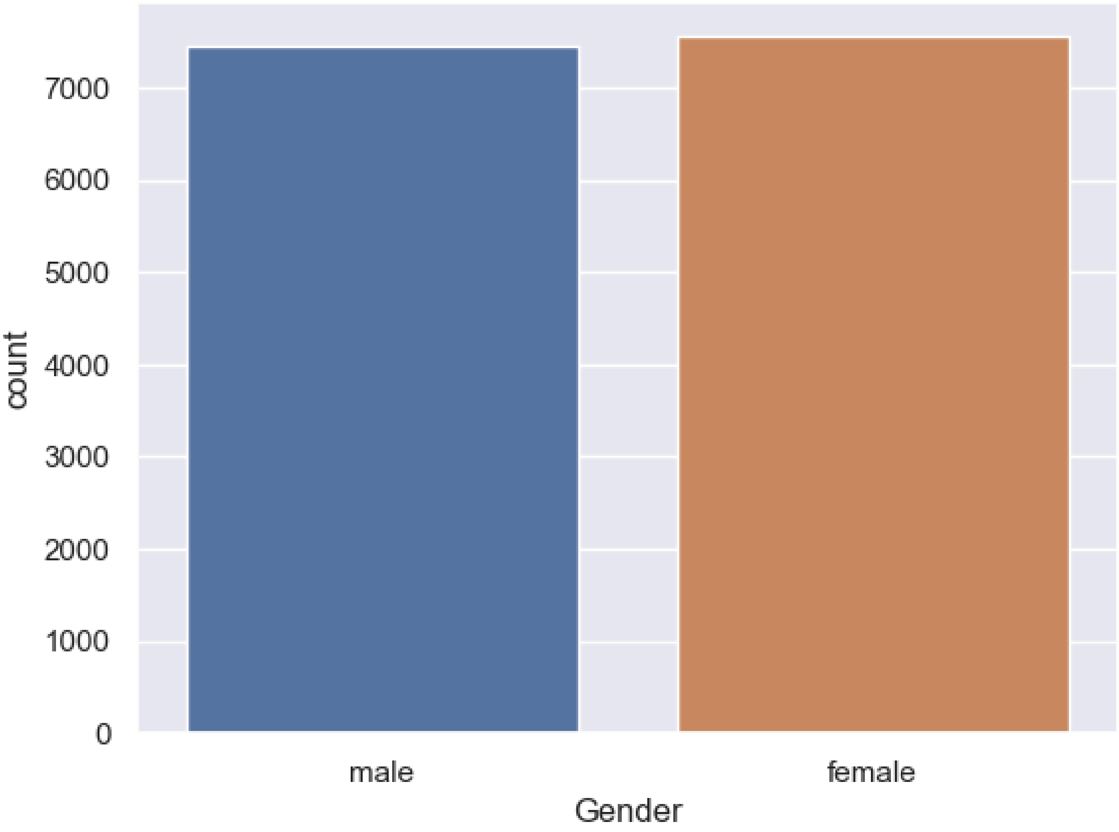
```
Gender
female    7553
male      7447
Name: count, dtype: int64
```

In [32]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting the 'Gender' column
sns.countplot(x="Gender", data=df)

# Display the plot
plt.show()
```



In [34]:
```
pd.get_dummies(df["Gender"],drop_first=True)
```

Out[34]:

	male
0	True
1	False
2	True
3	False
4	False
...	...
14995	False
14996	False
14997	False
14998	True
14999	True

15000 rows × 1 columns

In [35]:
```
categorical = df[cat_col]
categorical.head()
```

Out[35]:

	Gender
0	male
1	female
2	male
3	female
4	female

In [36]:

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

In [37]:

```
categorical
```

Out[37]:

	male
0	True
1	False
2	True
3	False
4	False
...	...
14995	False
14996	False
14997	False
14998	True
14999	True

15000 rows × 1 columns

2.Numerical Features

In [38]:

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

Out[38]:

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

In [39]:

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

Out[39]:

```
(15000, 7)
```

In [40]:

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

Out[40]:

	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

In [41]:

```
Numerical.shape
```

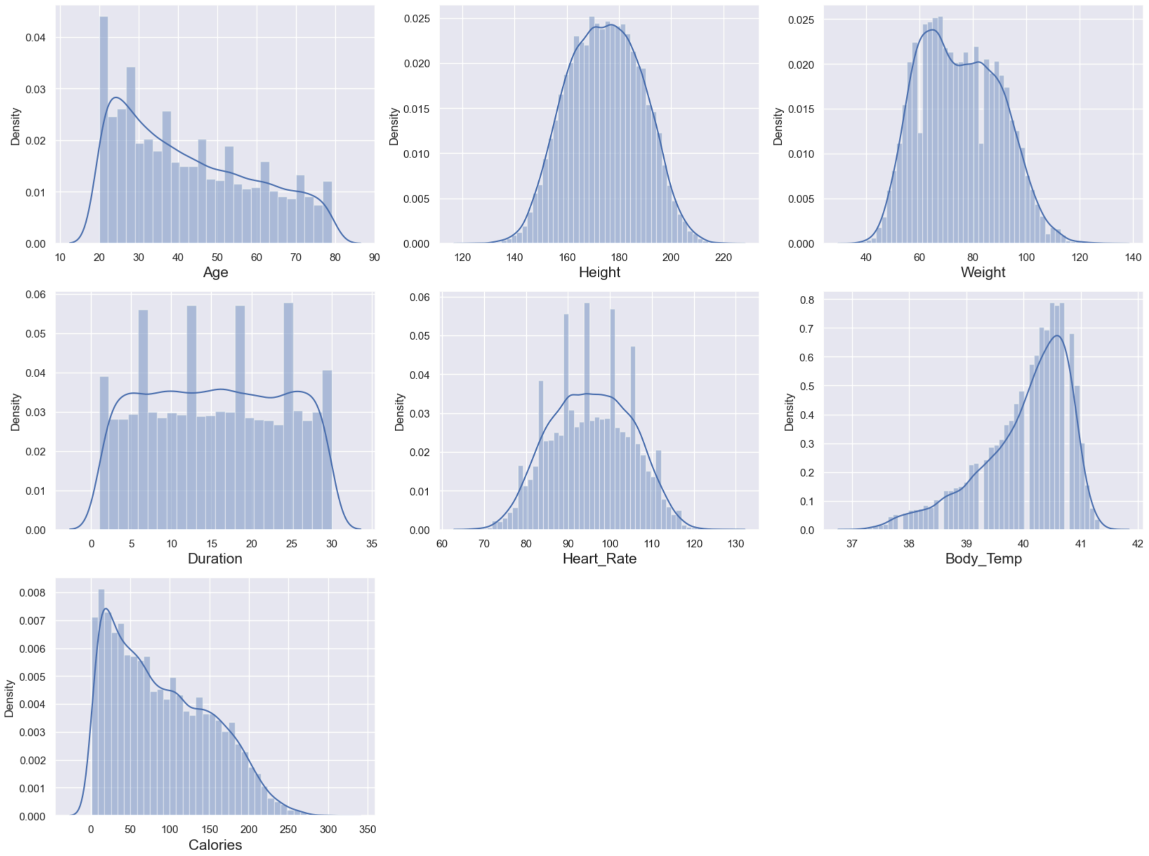
Out[41]:

```
(15000, 7)
```

In [42]:

```
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()
```



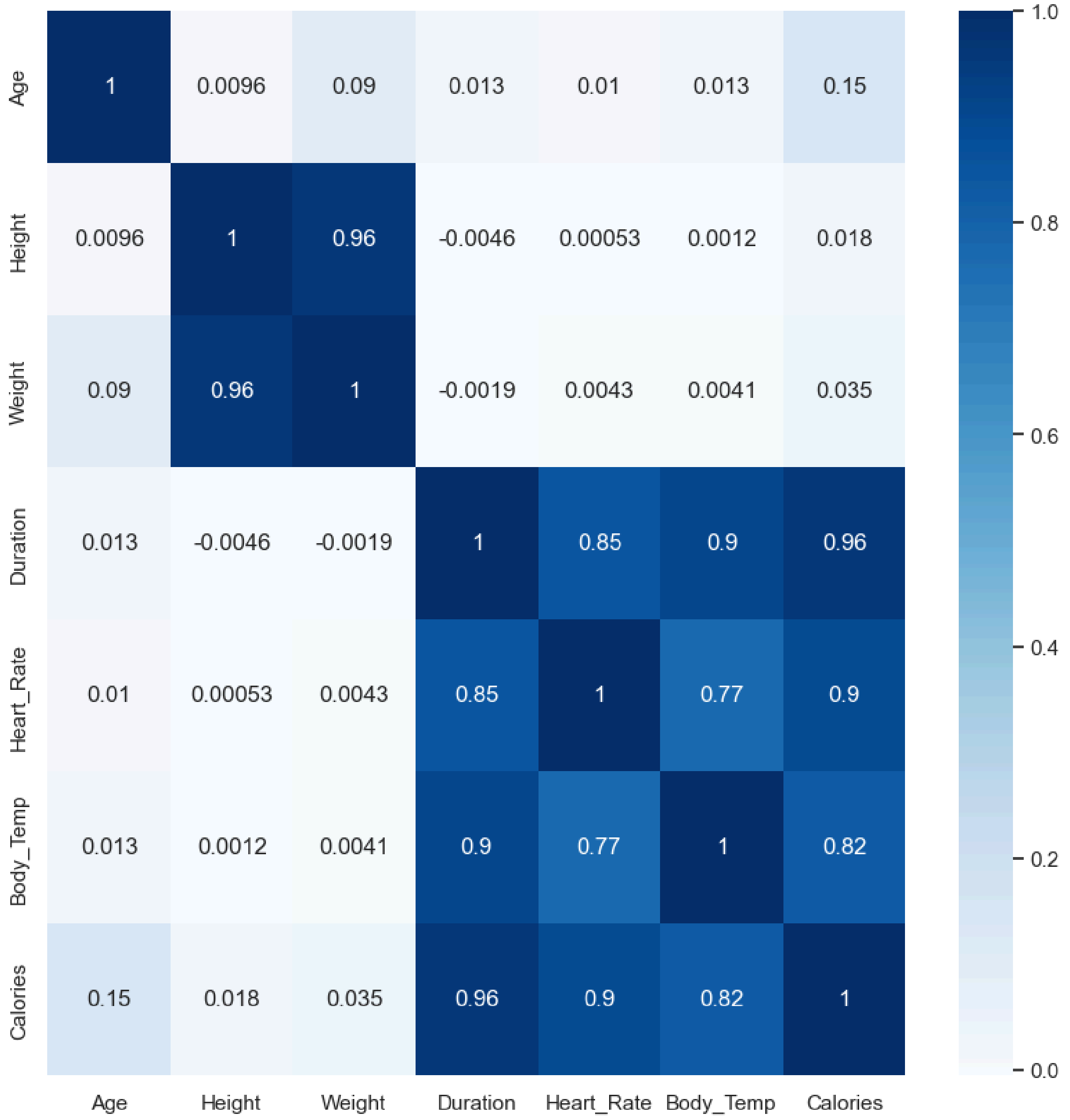
In [43]:

```
# constructing a heatmap to understand the correlation

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

Out[43]:

```
<Axes: >
```



Concatenate Categorical and Numerical

In [45]:

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

In [46]:

```
data.head()
```

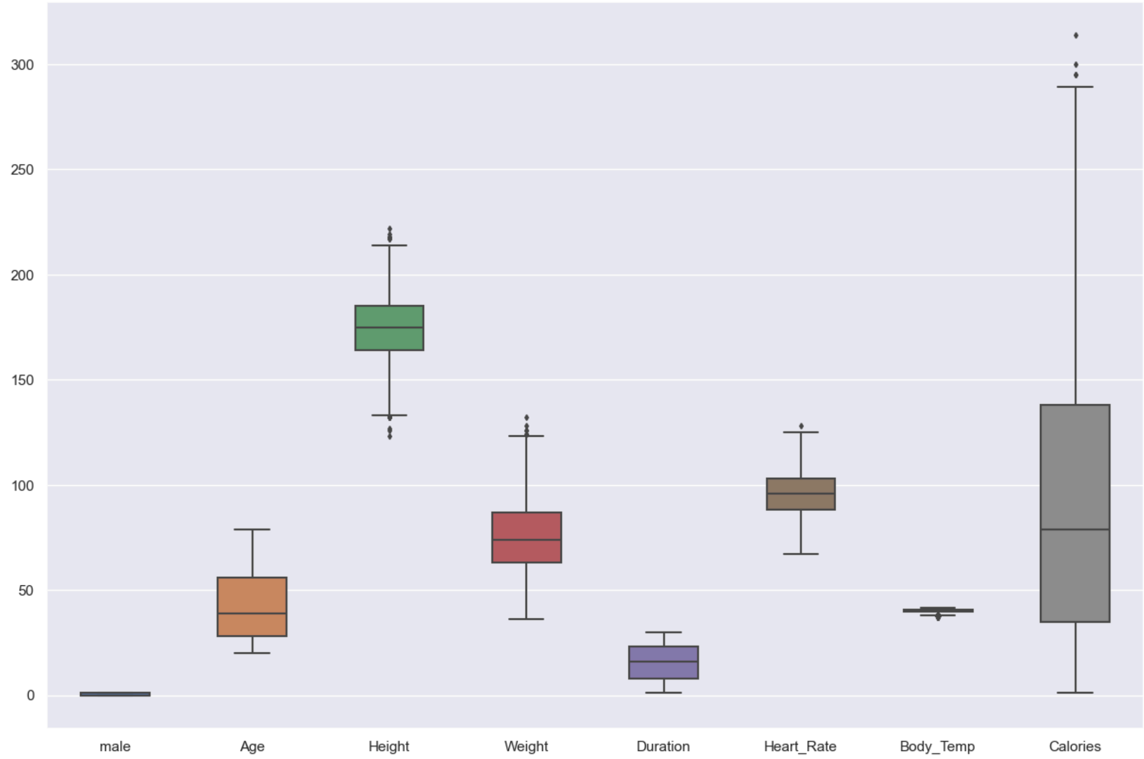
Out[46]:

	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

In [47]:

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

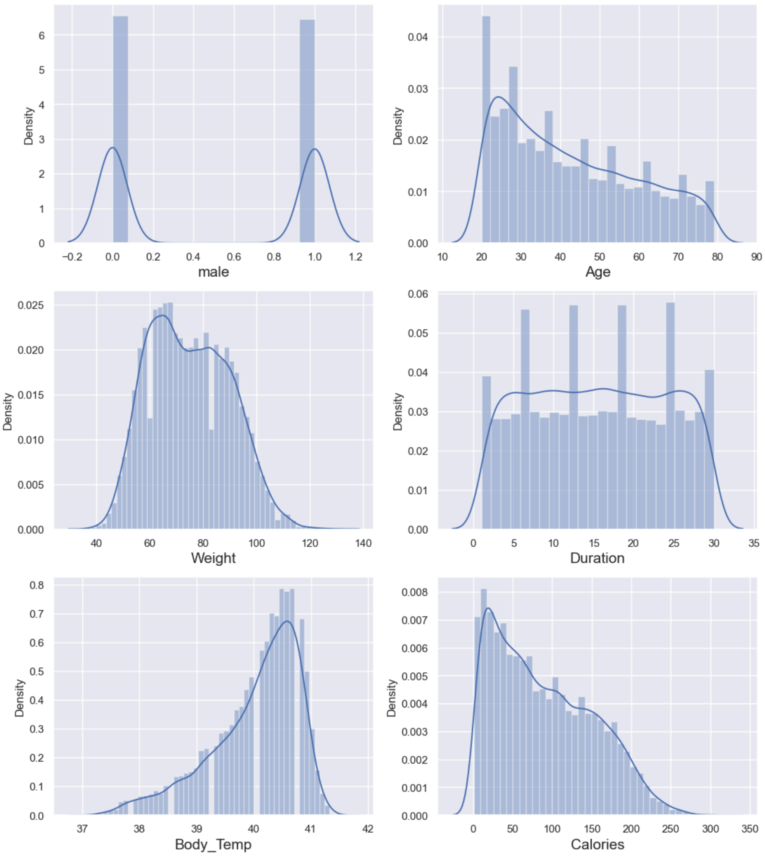
Out[47]: <Axes: >



In [48]:

```
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()
```



In [50]:
```
data.columns
```

Out[50]:
```
Index(['male', 'Age', 'Height', 'Weight', 'Duration', 'Heart_Rate',
       'Body_Temp', 'Calories'],
      dtype='object')
```

In [51]:
```
X = data.drop(columns = ["Calories"],axis = 1)
y = data["Calories"]
```

In [52]:
```
X.head()
```

Out[52]:

	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

In [53]:
```
y.head()
```

Out[53]:
```
0    231.0
1     66.0
2     26.0
3     71.0
4     35.0
Name: Calories, dtype: float64
```

In [54]:
```
# Split the Data
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_stat
```

In [55]:

```
print("Shape of X Train: ",X_train.shape)
print("Shape of X Test: ",X_test.shape)
print("Shape of y Train: ",y_train.shape)
print("Shape of y Test: ",y_test.shape)
```

```
Shape of X Train:  (12000, 7)
Shape of X Test:  (3000, 7)
Shape of y Train:  (12000,)
Shape of y Test:  (3000,)
```

In [57]:

```
#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)
```

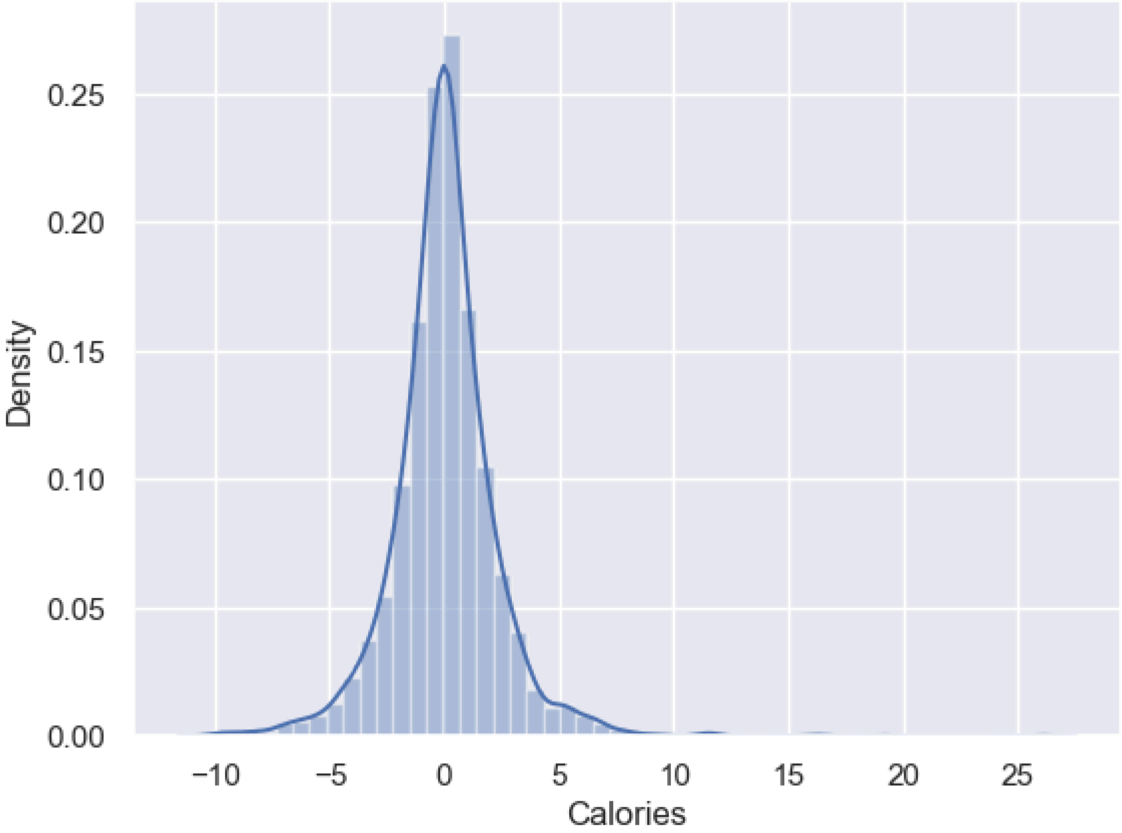
XGB Regressor

In [58]:

```
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
```



Save the Model

In [59]:

```
# saving the model to the local file system
filename = 'finalized_model.pickle'
pickle.dump(regression, open(filename, 'wb'))
```

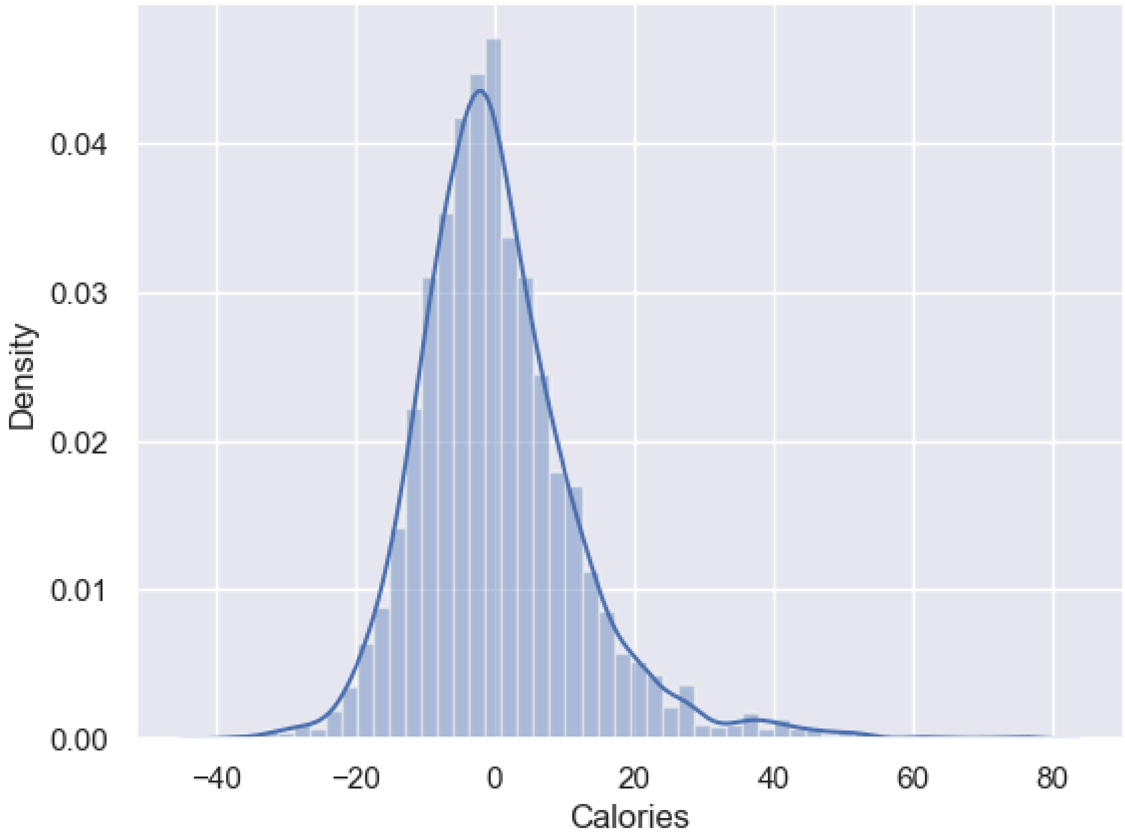
Linear Regression

In [60]:

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

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


r2 score: 0.9655977245826503
MAE: 8.479071745987945
MSE: 138.12408611460907
RMSE: 11.75262039353816
```

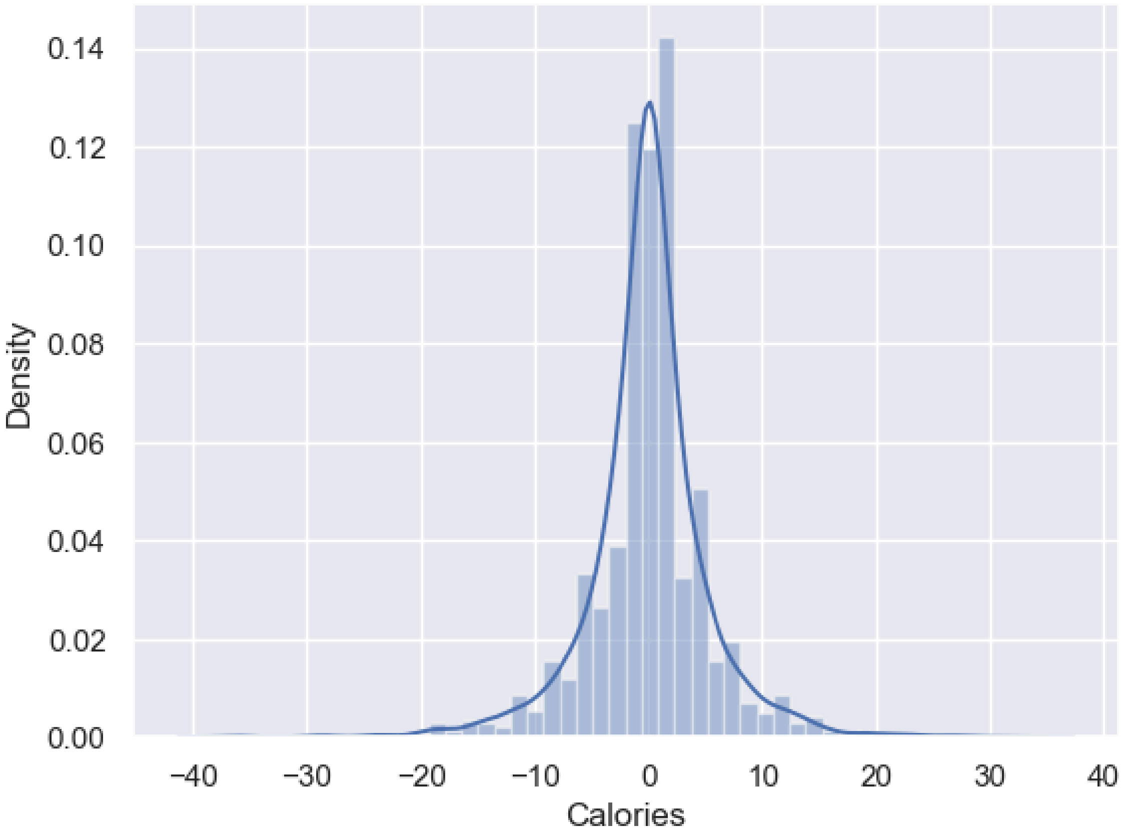


Decision Tree Regression

```
In [61]: predict(DecisionTreeRegressor())
```

```
Score : 1.0
predictions are:
 [194.  75. 204. ...  30. 112.  14.]


r2 score: 0.9923607555609444
MAE: 3.52
MSE: 30.671333333333333
RMSE: 5.538170576402765
```

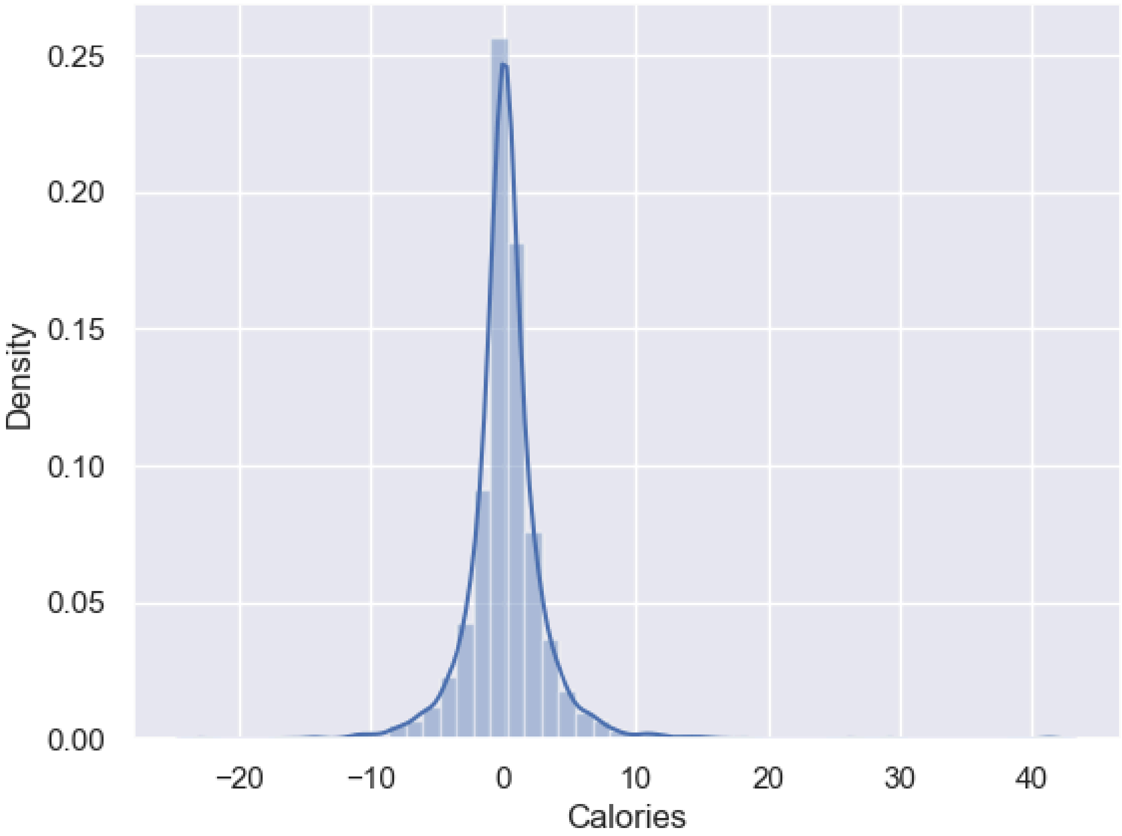


Random Forest Regression

```
In [62]: predict(RandomForestRegressor())
```

```
Score : 0.9996837423878473
predictions are:
 [197.15  65.97 196.35 ...  27.8  111.56  14.09]


r2 score: 0.9976679175699232
MAE: 1.8203333333333334
MSE: 9.3632398
RMSE: 3.059941143224817
```



```
In [ ]:
```