IMPORTING ALL DEPENDENCIES

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')

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

```
df2=pd.read_csv('exercise.csv')
df2
```

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
0	14733363	male	68	NaN	94.0	29	105	40.8
1	14861698	female	20	166.0	60.0	14	94	40.3
2	11179863	male	69	179.0	79.0	5	88	38.7
3	16180408	female	34	179.0	71.0	13	100	40.5
4	17771927	female	27	154.0	58.0	10	81	39.8
...	...	...	...	...	...	...	...	...
14995	15644082	female	20	193.0	86.0	11	92	40.4
14996	17212577	female	27	165.0	65.0	6	85	39.2
14997	17271188	female	43	159.0	58.0	16	90	

```
40.1
14998  18643037    male   78   193.0    97.0          2           84
38.3
14999  11751526    male   63   173.0    79.0         18           92
40.5

[15000 rows x 8 columns]
```

Column names and descriptions

User_ID: A unique numeric identifier assigned to each user for tracking purposes.

Gender: The gender of the user, either "male" or "female," indicating their biological sex.

Age: The user’s age in years, which could provide insights into fitness levels, health, and performance.

Height: The height of the user in centimeters, a measure of their physical stature (NA means missing or unavailable data).

Weight: The weight of the user in kilograms, an important factor for calculating BMI or understanding physical condition.

Duration: The amount of time, in minutes, that the user engaged in physical activity or exercise.

Heart_Rate: The user’s heart rate (beats per minute) during the exercise, indicative of exercise intensity and cardiovascular health.

Body_Temp: The user’s body temperature in degrees Celsius, reflecting their physiological response to exercise or physical activity.

Calories: The number of calories burned by the user during a given activity, reflecting the intensity and duration of the exercise

EDA

```
df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   User_ID   15000 non-null  int64  
 1   Calories  15000 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 234.5 KB

df1.isnull().sum()

User_ID     0
Calories    0
dtype: int64

df2.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 8 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  int64
 6   Heart_Rate  15000 non-null  int64
 7   Body_Temp   15000 non-null  float64
dtypes: float64(3), int64(4), object(1)
memory usage: 937.6+ KB
```

```
df2.isnull().sum()
```

```
User_ID         0
Gender          0
Age             0
Height        716
Weight        623
Duration        0
Heart_Rate      0
Body_Temp       0
dtype: int64
```

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')

columns_to_impute = ['Height', 'Weight']
df2[columns_to_impute] = imputer.fit_transform(df2[columns_to_impute])

df2.head()
```

```
    User_ID  Gender  Age      Height  Weight  Duration  Heart_Rate
Body_Temp
0  14733363    male   68  174.426281    94.0        29         105
40.8
1  14861698  female   20  166.000000    60.0        14          94
40.3
2  11179863    male   69  179.000000    79.0         5          88
38.7
3  16180408  female   34  179.000000    71.0        13         100
40.5
4  17771927  female   27  154.000000    58.0        10          81
39.8
```

```
df2.isnull().sum()
```

```
User_ID        0
Gender         0
Age            0
```

```
Height        0
Weight        0
Duration      0
Heart_Rate    0
Body_Temp     0
dtype: int64
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 8 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  int64
 6   Heart_Rate  15000 non-null  int64
 7   Body_Temp   15000 non-null  float64
dtypes: float64(3), int64(4), object(1)
memory usage: 937.6+ KB
```

```
Calories_tar=pd.concat([df2.drop(columns='User_ID'),df1['Calories']],a
xis=1)
Calories_tar
```

```
       Gender  Age      Height  Weight  Duration  Heart_Rate
Body_Temp  \
0        male   68  174.426281    94.0        29         105
40.8
1      female   20  166.000000    60.0        14          94
40.3
2        male   69  179.000000    79.0         5          88
38.7
3      female   34  179.000000    71.0        13         100
40.5
4      female   27  154.000000    58.0        10          81
39.8
...       ...  ...         ...     ...       ...         ...         ..
.
14995  female   20  193.000000    86.0        11          92
40.4
14996  female   27  165.000000    65.0         6          85
39.2
14997  female   43  159.000000    58.0        16          90
40.1
14998    male   78  193.000000    97.0         2          84
```

```
38.3
14999    male   63  173.000000    79.0        18           92
40.5

       Calories
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

[15000 rows x 8 columns]
```

```
Calories_tar.isnull().sum()
```

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

```
Calories_tar.head()
```

```
   Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
Calories
0     1.0   68   174.0    94.0        29         105       40.8
231.0
1     0.0   20   166.0    60.0        14          94       40.3
66.0
2     1.0   69   179.0    79.0         5          88       38.7
26.0
3     0.0   34   179.0    71.0        13         100       40.5
71.0
4     0.0   27   154.0    58.0        10          81       39.8
35.0
```

```
Calories_tar['Height']=Calories_tar['Height'].round(0)
Calories_tar['Weight']=Calories_tar['Weight'].round(0)
```

```
Calories_tar
```

```
       Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
Calories
0         1.0   68   174.0    94.0        29         105       40.8
231.0
1         0.0   20   166.0    60.0        14          94       40.3
66.0
2         1.0   69   179.0    79.0         5          88       38.7
26.0
3         0.0   34   179.0    71.0        13         100       40.5
71.0
4         0.0   27   154.0    58.0        10          81       39.8
35.0
...       ...  ...     ...     ...       ...         ...        ...
...
14995     0.0   20   193.0    86.0        11          92       40.4
45.0
14996     0.0   27   165.0    65.0         6          85       39.2
23.0
14997     0.0   43   159.0    58.0        16          90       40.1
75.0
14998     1.0   78   193.0    97.0         2          84       38.3
11.0
14999     1.0   63   173.0    79.0        18          92       40.5
98.0

[15000 rows x 8 columns]
```

```python
cat_col = ['Gender']
```

```python
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
Calories_tar[cat_col] = oe.fit_transform(Calories_tar[cat_col])
print(Calories_tar.head())
```

```
   Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
Calories
0     1.0   68   174.0    94.0        29         105       40.8
231.0
1     0.0   20   166.0    60.0        14          94       40.3
66.0
2     1.0   69   179.0    79.0         5          88       38.7
26.0
3     0.0   34   179.0    71.0        13         100       40.5
71.0
4     0.0   27   154.0    58.0        10          81       39.8
35.0
```

```python
Calories_tar.head()
```

```
   Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
Calories
0     1.0   68   174.0    94.0        29         105        40.8
231.0
1     0.0   20   166.0    60.0        14          94        40.3
66.0
2     1.0   69   179.0    79.0         5          88        38.7
26.0
3     0.0   34   179.0    71.0        13         100        40.5
71.0
4     0.0   27   154.0    58.0        10          81        39.8
35.0
```

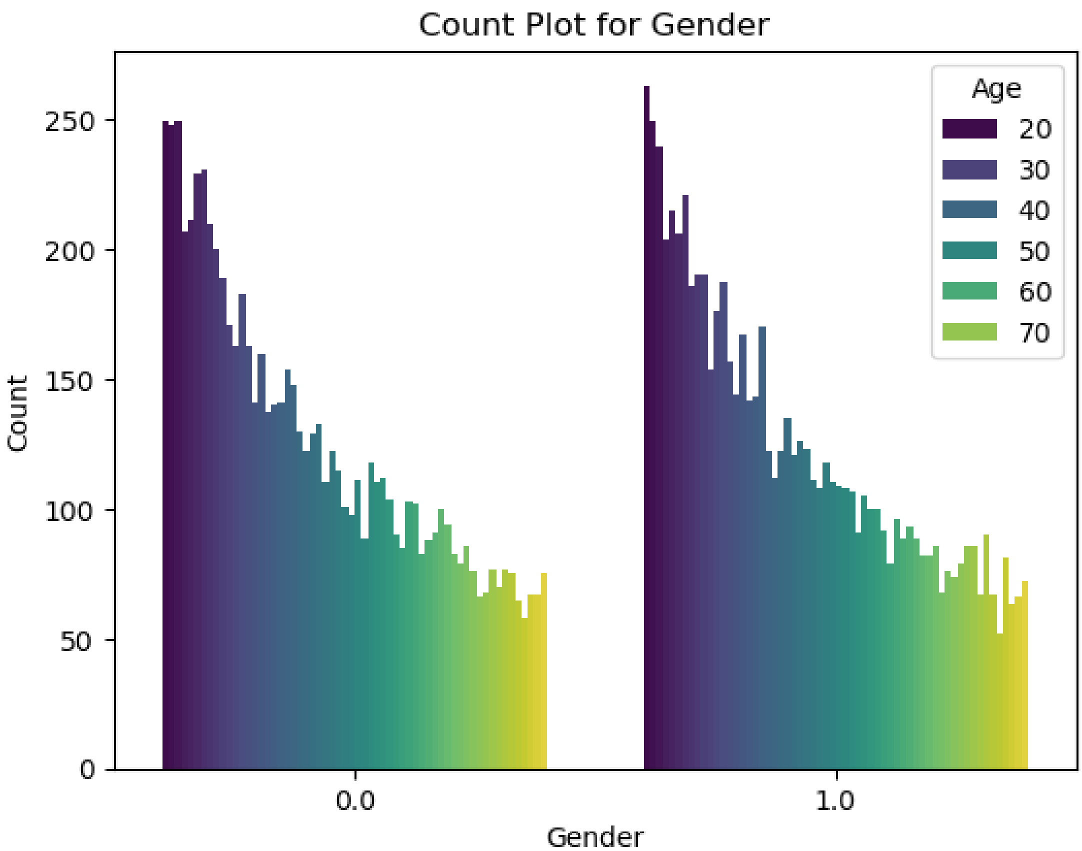
```
Calories_tar.describe()
```

```
             Gender           Age        Height        Weight
Duration  \
count  15000.000000  15000.000000  15000.000000  15000.000000
15000.000000
mean       0.496467     42.789800    174.405933     74.721333
15.530600
std        0.500004     16.980264     13.970122     14.954199
8.319203
min        0.000000     20.000000    123.000000     38.000000
1.000000
25%        0.000000     28.000000    164.000000     63.000000
8.000000
50%        0.000000     39.000000    174.000000     74.000000
16.000000
75%        1.000000     56.000000    184.000000     87.000000
23.000000
max        1.000000     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
```

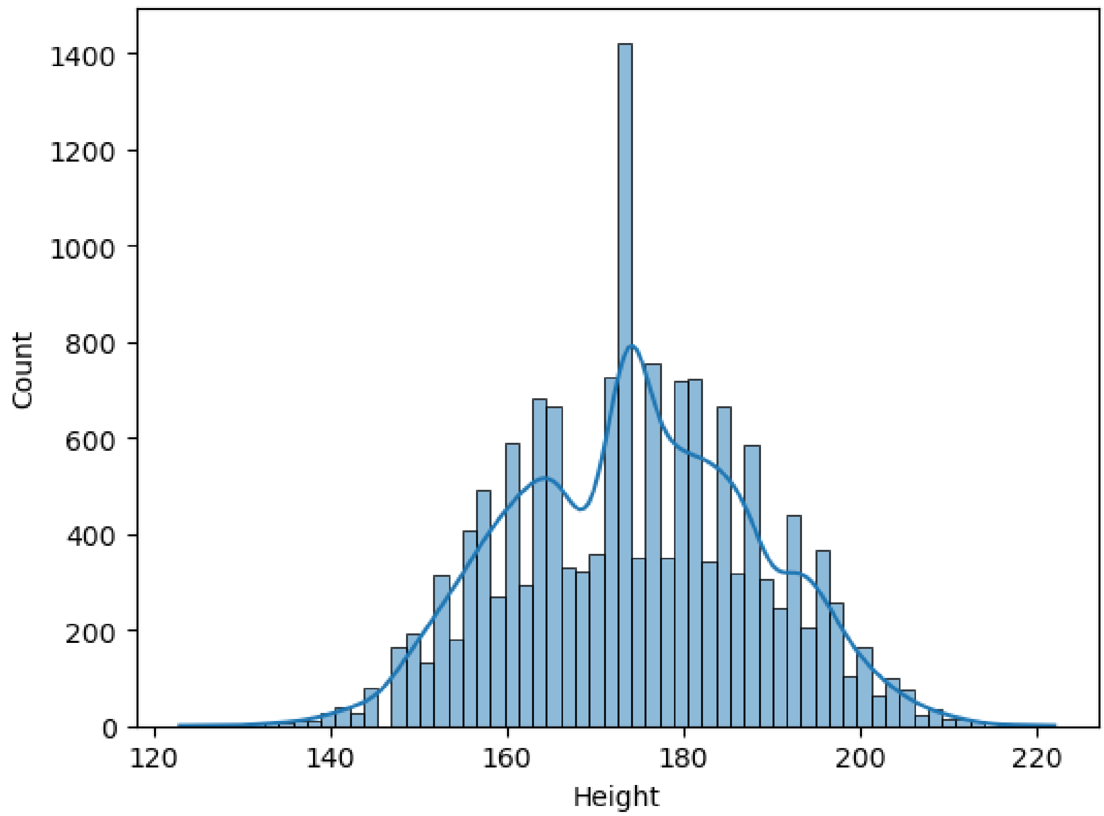
```
sns.countplot(data=Calories_tar, x='Gender',
palette='viridis',hue='Age')
plt.title('Count Plot for Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



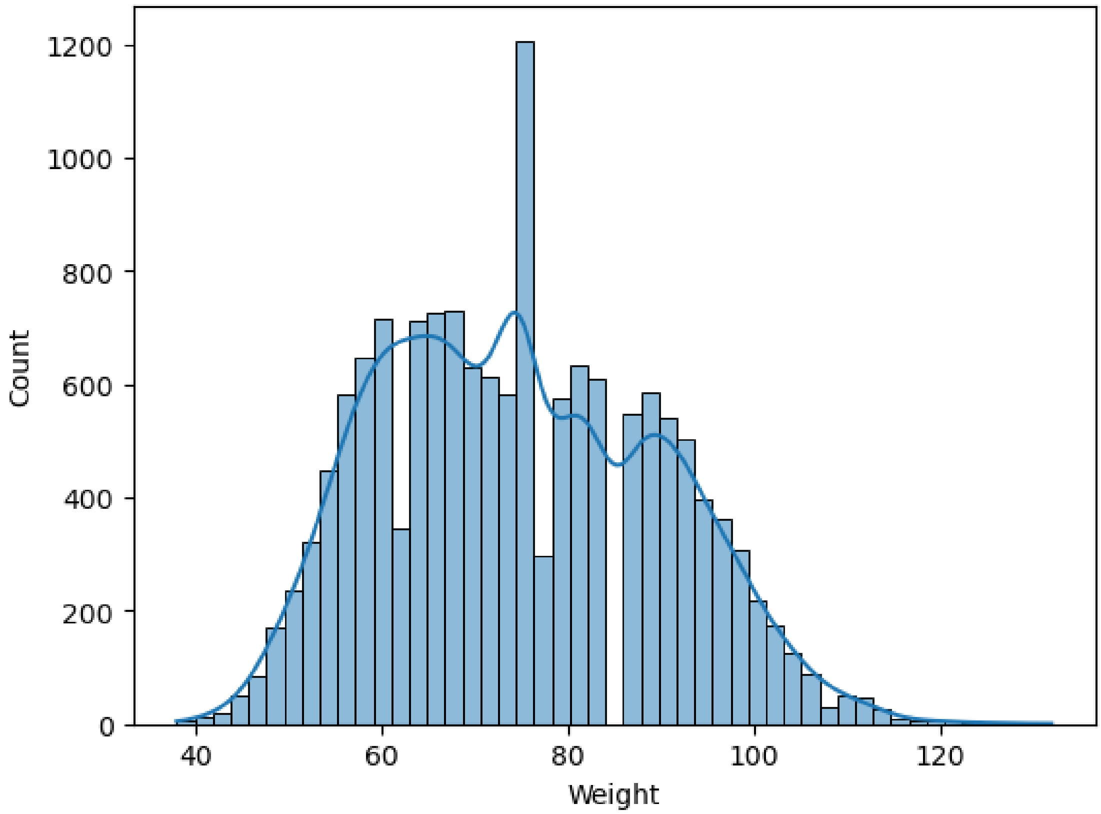

```
sns.histplot(Calories_tar['Height'], kde=True)
```

```
<Axes: xlabel='Height', ylabel='Count'>
```

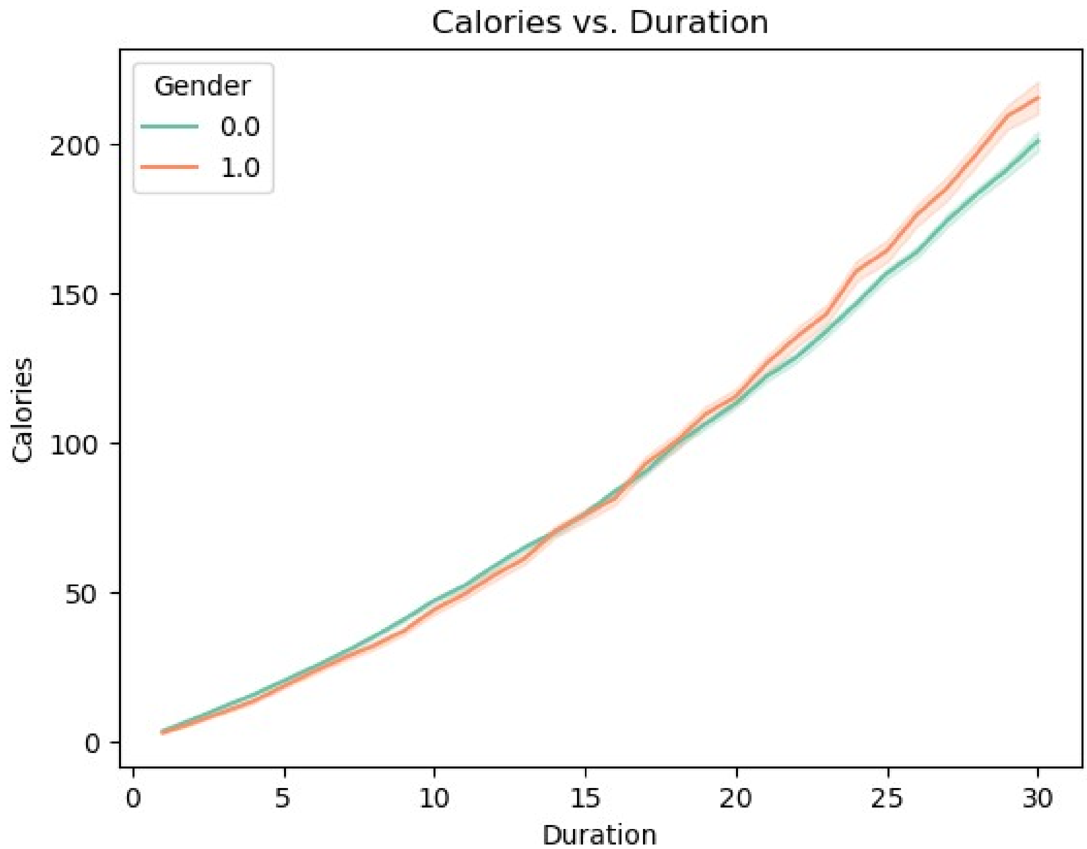



```
sns.histplot(Calories_tar['Weight'], kde=True)
```

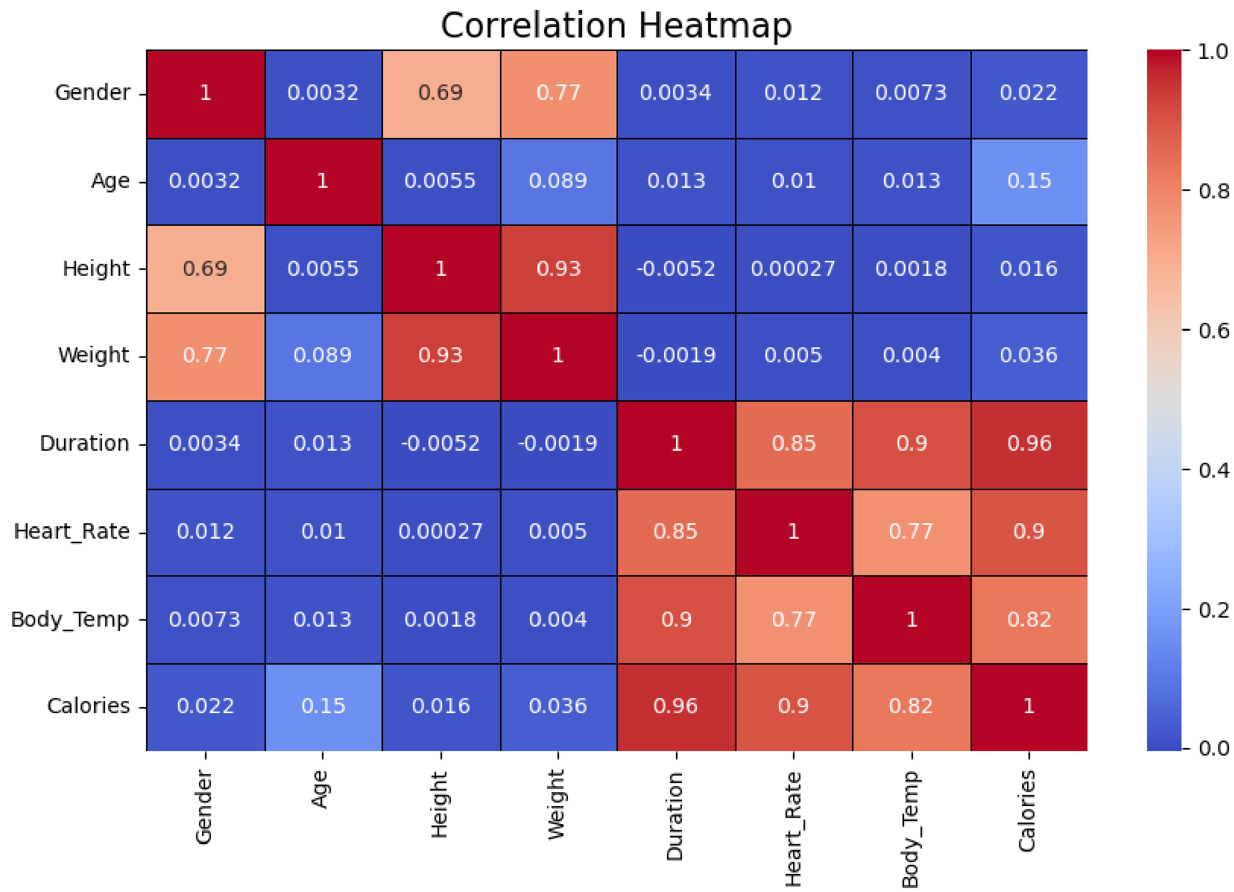
```
<Axes: xlabel='Weight', ylabel='Count'>
```




```
sns.lineplot(data=Calories_tar, x='Duration', y='Calories',
hue='Gender', palette='Set2')
plt.title('Calories vs. Duration')
plt.show()
```




```python
# Compute the correlation matrix
correlation = Calories_tar.corr()
plt.figure(figsize=(10, 6))
sns.heatmap(correlation,annot=True,
cmap='coolwarm',linewidths=0.5,linecolor='black')
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```


Correlation Heatmap
Gender
Age
Height
Weight
Duration
Heart_Rate
Body_Temp
Calories
1 0.0032 0.69 0.77 0.0034 0.012 0.0073 0.022
0.0032 1 0.0055 0.089 0.013 0.01 0.013 0.15
0.69 0.0055 1 0.93 -0.0052 0.00027 0.0018 0.016
0.77 0.089 0.93 1 -0.0019 0.005 0.004 0.036
0.0034 0.013 -0.0052 -0.0019 1 0.85 0.9 0.96
0.012 0.01 0.00027 0.005 0.85 1 0.77 0.9
0.0073 0.013 0.0018 0.004 0.9 0.77 1 0.82
0.022 0.15 0.016 0.036 0.96 0.9 0.82 1
1.0
0.8
0.6
0.4
0.2
0.0

Strong Correlations: Features like Duration(0.96), Heart Rate(0.93), and Calories are highly interrelated, which makes sense as longer durations and higher heart rates are indicators of more physical activity and energy expenditure.

Weak/No Correlation: Some features like Gender(0.2),Age(0.15) and Weight(0.036) with most other variables show very weak correlation, suggesting less influence on those outcomes.

Splitting Data

```
x = Calories_tar.drop(columns=[ "Calories"])
y = Calories_tar["Calories"]

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size =
0.2,random_state = 1)

x.head()
```

	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
0	1.0	68	174.0	94.0	29	105	40.8
1	0.0	20	166.0	60.0	14	94	40.3
2	1.0	69	179.0	79.0	5	88	38.7
3	0.0	34	179.0	71.0	13	100	40.5
4	0.0	27	154.0	58.0	10	81	39.8

```
y.head()
```

0	231.0
1	66.0
2	26.0
3	71.0

```
4     35.0
Name: Calories, dtype: float64
```

In our case, the target variable is Calories (the number of calories burned). Since Calories is a continuous numeric value, this is a regression problem.

LINEAR REGRESSION

```
from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest = scaler.transform(xtest)

lr=LinearRegression()
lr.fit(xtrain,ytrain)
ypred=lr.predict(xtest)

train=lr.score(xtrain,ytrain)
test=lr.score(xtest,ytest)
print(train)
print(test)

0.9675849149538076
0.965485218543629

from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(ytest, ypred)
r2 = r2_score(ytest, ypred)
rmse = np.sqrt(mse)

print(f"Linear Regression R2_Score: {r2:.4f}")
print(f"Linear Regression MSE: {mse:.2f}")
print(f"Linear Regression RMSE: {rmse:.2f}")

Linear Regression R2_Score: 0.9655
Linear Regression MSE: 138.58
Linear Regression RMSE: 11.77
```

DECISION TREE

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(xtrain,ytrain)
ypred_dt=dt.predict(xtest)

mse_dt = mean_squared_error(ytest, ypred_dt)
r2_dt = r2_score(ytest, ypred_dt)
rmse_dt = np.sqrt(mse_dt)

print(f"Decision Tree R2_Score: {r2_dt:.4f}")
print(f"Decision Tree MSE: {mse_dt:.2f}")
print(f"Decision Tree RMSE: {rmse_dt:.2f}")

Decision Tree R2_Score: 0.9919
Decision Tree MSE: 32.36
Decision Tree RMSE: 5.69
```

RANDOM FOREST

```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)
rf_regressor.fit(xtrain, ytrain)
ypred_rf = rf_regressor.predict(xtest)

mse_rf = mean_squared_error(ytest, ypred_rf)
r2_rf = r2_score(ytest, ypred_rf)
rmse_rf = np.sqrt(mse_rf)

print(f"Random Forest R2_Score: {r2_rf:.4f}")
print(f"Random Forest MSE: {mse_rf:.2f}")
print(f"Random Forest RMSE: {rmse_rf:.2f}")

Random Forest R2_Score: 0.9976
Random Forest MSE: 9.49
Random Forest RMSE: 3.08
```

XGBoost Regression

```
from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators=100, learning_rate=0.1,
random_state=42)
xgb.fit(xtrain, ytrain)
ypred_xgb = xgb.predict(xtest)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest = scaler.transform(xtest)

mse_xgb = mean_squared_error(ytest, ypred_xgb)
r2_xgb = r2_score(ytest, ypred_xgb)
rmse_xgb = np.sqrt(mse_xgb)

print(f"XGBoost R2_Score: {r2_xgb:.4f}")
print(f"XGBoost MSE: {mse_xgb:.2f}")
print(f"XGBoost RMSE: {rmse_xgb:.2f}")

XGBoost R2_Score: 0.9989
XGBoost MSE: 4.32
XGBoost RMSE: 2.08
```

Support Vector Regression (SVR)

```
from sklearn.svm import SVR
svr = SVR(kernel='rbf')
svr.fit(xtrain, ytrain)
ypred_svr = svr.predict(xtest)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
xtrain = scaler.fit_transform(xtrain)
xtest = scaler.transform(xtest)

mse_svr = mean_squared_error(ytest, ypred_svr)
r2_svr = r2_score(ytest, ypred_svr)
rmse_svr = np.sqrt(mse_svr)

print(f"SVR R@_Score: {r2_svr:.4f}")
print(f"SVR MSE: {mse_svr:.2f}")
print(f"SVR RMSE: {rmse_svr:.2f}")

SVR R@_Score: 0.9897
SVR MSE: 41.45
SVR RMSE: 6.44

import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score


models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(),
```

```python
    "Random Forest": RandomForestRegressor(),
    "SVR": SVR(),
    "XGBoost": XGBRegressor()
}


results = {"Model": [], "R² Score": [], "MSE": [], "RMSE": []}

for name, model in models.items():
    model.fit(xtrain, ytrain)
    ypred = model.predict(xtest)

    mse = mean_squared_error(ytest, ypred)
    r2 = r2_score(ytest, ypred)
    rmse = np.sqrt(mse)

    results["Model"].append(name)
    results["R² Score"].append(r2)
    results["MSE"].append(mse)
    results["RMSE"].append(rmse)


results_df = pd.DataFrame(results)

print(results_df)
```

```
               Model  R² Score         MSE       RMSE
0  Linear Regression  0.965485  138.575794  11.771822
1      Decision Tree  0.992203   31.303333   5.594938
2      Random Forest  0.997647    9.449119   3.073942
3                SVR  0.989675   41.452875   6.438391
4            XGBoost  0.998620    5.540678   2.353864
```

XGBoost is the top performer with the highest accuracy (R^2: 0.9986), lowest MSE (5.54), and lowest RMSE (2.35), making it the most reliable model for this dataset.

Random Forest follows closely with a strong performance (R^2: 0.9976), though slightly higher error metrics (MSE: 9.45, RMSE: 3.07) compared to XGBoost.

Linear Regression shows a lower R^2 score (0.9655) and higher error metrics (MSE: 138.58, RMSE: 11.77), indicating less predictive accuracy.

Decision Tree and SVR perform the weakest, with higher error metrics and lower R^2 scores.