Importing the Dependencies

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

Data Collection & Processing

```
#loading the data from csv file to Pandas dataframe
calories = pd.read_csv('/calories.csv')
```

```
# print the first 5 rows of the dataframe
calories.head()
```

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

Next steps: Generate code with `calories` View recommended plots New interactive sheet

```
exercise_data = pd.read_csv('/exercise.csv')
```

```
exercise_data.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

Next steps: Generate code with `exercise_data` View recommended plots New interactive sheet

Combining Two Data frames

```
calories_data = pd.concat([exercise_data, calories['Calories']], axis=1)
```

```
calories_data.head()
```

	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

Next steps: Generate code with `calories_data` View recommended plots New interactive sheet

```
# checking the number of rows and columns
calories_data.shape
```

```
(15000, 9)
```

```
# getting some informations about the data
calories_data.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
```

```
# checking for missing values
calories_data.isnull().sum()
```

	0
User_ID	0
Gender	0
Age	0
Height	0
Weight	0
Duration	0
Heart_Rate	0
Body_Temp	0
Calories	0

dtype: int64

Data Analysis

```
# get some statistical measures about the data
calories_data.describe()
```

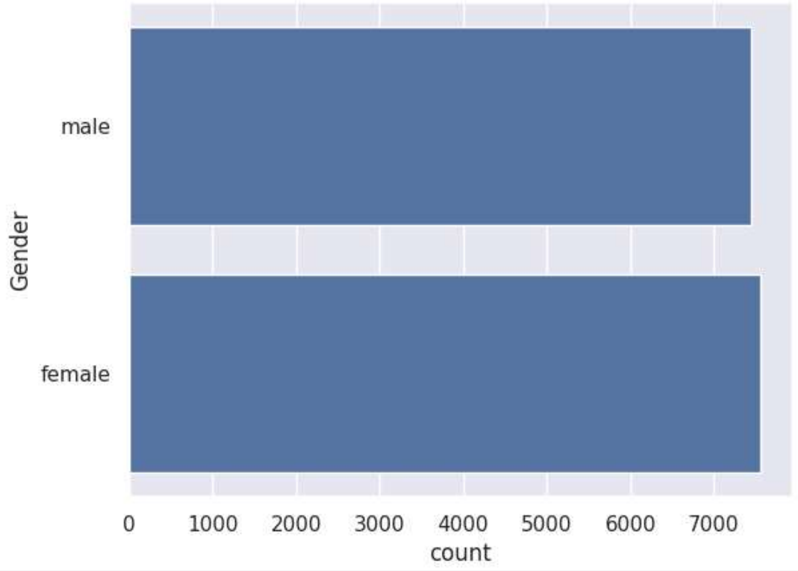
	User_ID	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
count	1.500000e+04	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000
mean	1.497736e+07	42.789800	174.465133	74.966867	15.530600	95.518533	40.025453	89.539533
std	2.872851e+06	16.980264	14.258114	15.035657	8.319203	9.583328	0.779230	62.456978
min	1.000116e+07	20.000000	123.000000	36.000000	1.000000	67.000000	37.100000	1.000000
25%	1.247419e+07	28.000000	164.000000	63.000000	8.000000	88.000000	39.600000	35.000000
50%	1.499728e+07	39.000000	175.000000	74.000000	16.000000	96.000000	40.200000	79.000000
75%	1.744928e+07	56.000000	185.000000	87.000000	23.000000	103.000000	40.600000	138.000000
max	1.999965e+07	79.000000	222.000000	132.000000	30.000000	128.000000	41.500000	314.000000

Data Visualization

```
sns.set()
```

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

```
<Axes: xlabel='count', ylabel='Gender'>
```



```
# finding the distribution of "Age" column
sns.distplot(calories_data['Age'])
```

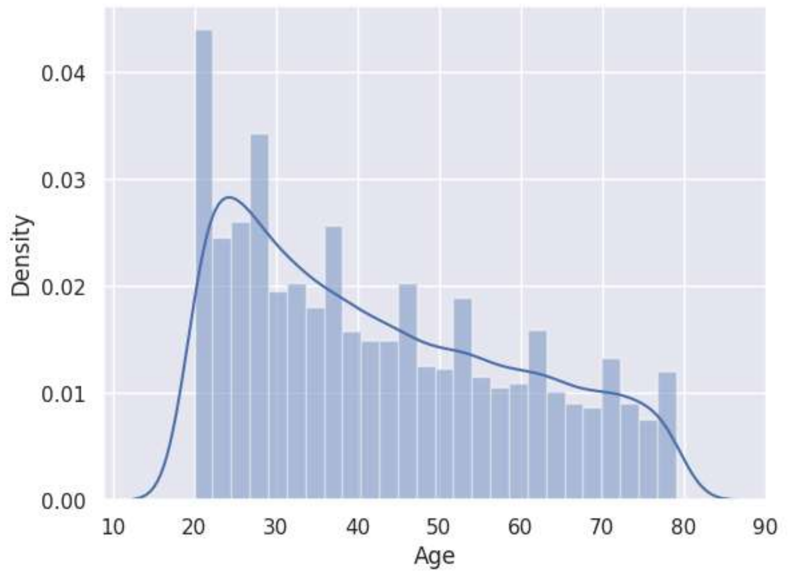
```
<ipython-input-32-6cbf196d4d06>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(calories_data['Age'])
<Axes: xlabel='Age', ylabel='Density'>
```



```
# finding the distribution of "Height" column
sns.distplot(calories_data['Height'])
```

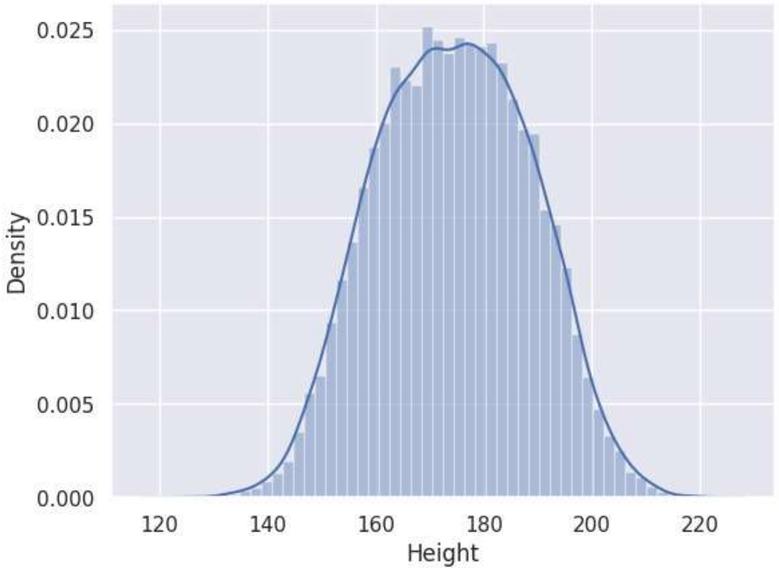
```
<ipython-input-33-fdc2a1fecb6d>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(calories_data['Height'])
<Axes: xlabel='Height', ylabel='Density'>
```



```
# finding the distribution of "Weight" column
sns.distplot(calories_data['Weight'])
```

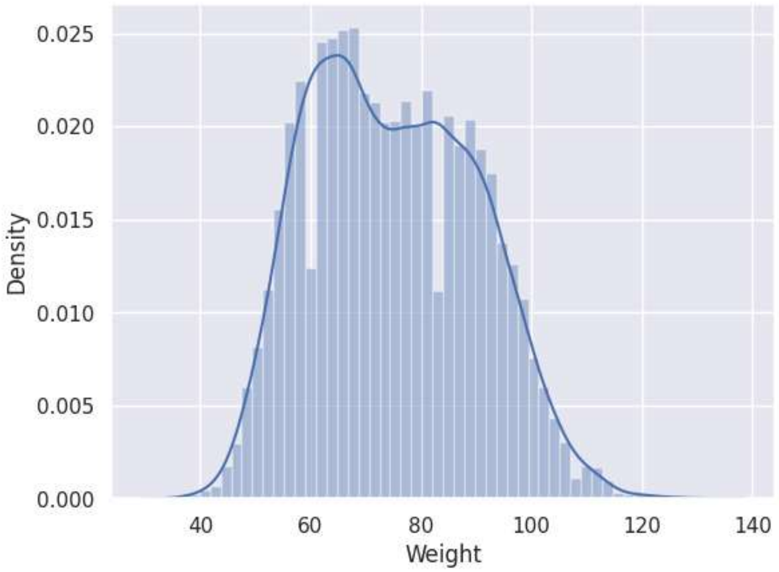
```
<ipython-input-34-ac6457c483b4>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(calories_data['Weight'])
<Axes: xlabel='Weight', ylabel='Density'>
```



Finding the Correlation in the dataset

1. Positive Correlation
2. Negative Correlation

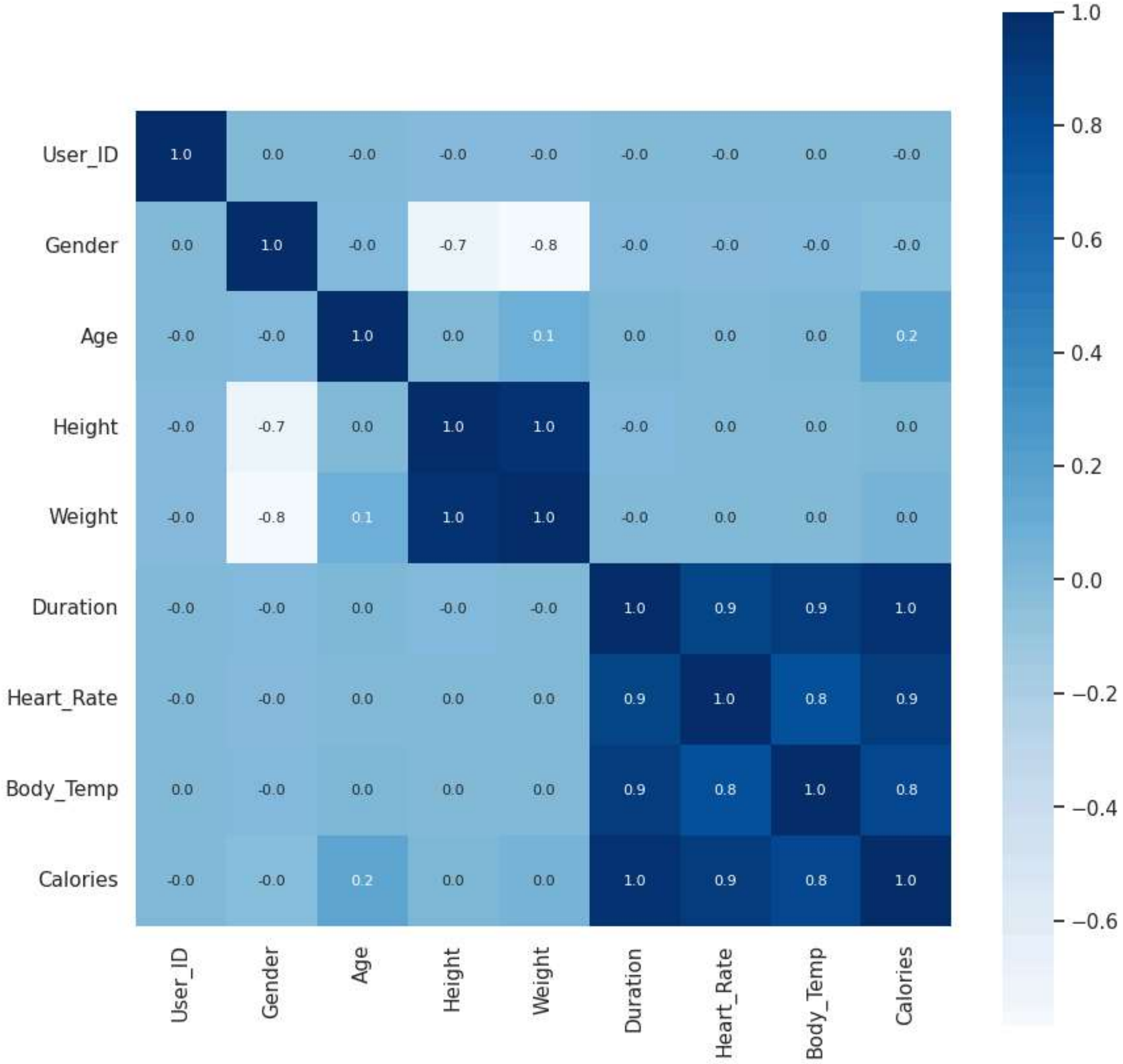
```
calories_data.replace({"Gender":{'male':0,'female':1}}, inplace=True)
```

```
correlation = calories_data.corr()
```

```
# constructing a heatmap to understand the correlation

plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap='Blues')
```

```
<Axes: >
```



	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
User_ID	1.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	-0.0
Gender	0.0	1.0	-0.0	-0.7	-0.8	-0.0	-0.0	-0.0	-0.0
Age	-0.0	-0.0	1.0	0.0	0.1	0.0	0.0	0.0	0.2
Height	-0.0	-0.7	0.0	1.0	1.0	-0.0	0.0	0.0	0.0
Weight	-0.0	-0.8	0.1	1.0	1.0	-0.0	0.0	0.0	0.0
Duration	-0.0	-0.0	0.0	-0.0	-0.0	1.0	0.9	0.9	1.0
Heart_Rate	-0.0	-0.0	0.0	0.0	0.0	0.9	1.0	0.8	0.9
Body_Temp	0.0	-0.0	0.0	0.0	0.0	0.9	0.8	1.0	0.8
Calories	-0.0	-0.0	0.2	0.0	0.0	1.0	0.9	0.8	1.0

Separating features and Target

```
X = calories_data.drop(columns=['User_ID','Calories'], axis=1)
Y = calories_data['Calories']
```

```
print(X)
```

```
       Gender  Age  Height  Weight  Duration  Heart_Rate  Body_Temp
0           0   68   190.0    94.0      29.0       105.0       40.8
1           1   20   166.0    60.0      14.0        94.0       40.3
2           0   69   179.0    79.0       5.0        88.0       38.7
3           1   34   179.0    71.0      13.0       100.0       40.5
```

```
4           1   27   154.0    58.0      10.0        81.0       39.8
...       ...  ...     ...     ...       ...         ...        ...
14995       1   20   193.0    86.0      11.0        92.0       40.4
14996       1   27   165.0    65.0       6.0        85.0       39.2
14997       1   43   159.0    58.0      16.0        90.0       40.1
14998       0   78   193.0    97.0       2.0        84.0       38.3
14999       0   63   173.0    79.0      18.0        92.0       40.5

[15000 rows x 7 columns]
```

```
print(Y)
```

```
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
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(15000, 7) (12000, 7) (3000, 7)
```

Model Training

XGBoost Regression

Generate create a dataframe with 2 columns and 10 rows Close

```
# loading the model
model = XGBRegressor()
```

```
# training the model with X_train
model.fit(X_train, Y_train)
```

```
                              XGBRegressor
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, ...)
```

Evaluation

Prediction on Test Data

```
test_data_prediction = model.predict(X_test)
```

```
print(test_data_prediction)
```

```
[125.58828  222.11377   38.725952 ... 144.3179    23.425894  90.100494]
```

Mean Absolute Error

```
mae = metrics.mean_absolute_error(Y_test, test_data_prediction)
```

```
print("Mean Absolute Error = ", mae)
```

```
Mean Absolute Error =  1.4833678883314132
```