In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler,OrdinalEncoder

from sklearn.metrics import r2_score

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import KFold

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor
```

In [2]:
```
# 1. Reading data from CSV
def read_csv(file_path):
    """
    Read data from a CSV file and return a pandas DataFrame.

    Parameters:
    - file_path: str, the path to the CSV file.

    Returns:
    - pd.DataFrame, the loaded DataFrame.
    """
    return pd.read_csv(file_path)
#2. Getting information and statistics about over dataset
def dataset_info_statistics(data):
    """
    Display information and basic statistics about the dataset.

    Parameters:
    - data: pandas DataFrame, input data.

    Returns:
    - None
    """
    # Display general information about the dataset
    print("Dataset Information:")
    print(data.info())
    print("\n")

    # Display basic statistics for numerical columns
    print("Basic Statistics for Numerical Columns:")
    print(data.describe())
    print("\n")

#3.check for the null values in the dataset
def check_null(data):
    """
    Check for null values in the dataset.

    Parameters:
    - data: pandas DataFrame, input data.

    Returns:
    - pd.Series, the count of null values for each column.
    """
    null_counts = data.isnull().sum()
    print("Null Values in the Dataset:")
    return null_counts

#4.check for duplicated rows in the dataset
def check_duplicates(data):
    """
    Check for duplicated rows in the dataset.
```

```
    Parameters:
    - data: pandas DataFrame, input data.

    Returns:
    - bool, True if any duplicated rows exist, False otherwise.
    """
    return data.duplicated().any()

#5. getting basic analysis for numerical and categorical columns
def plot_graph(data):
    """
    Plot graphs for numerical and categorical data in a dataframe.

    Parameters:
    - data: Pandas Dataframe, input data.

    Returns:
    - None

    """
    numerical_columns = data.select_dtypes(include=np.number).columns

    for column in numerical_columns:
        plt.figure(figsize=(5,3))
        sns.distplot(data[column],kde=True)
        plt.title(f"Histogram for {column}")
        plt.xlabel(column)
        plt.ylabel("Frequency")
        plt.show()

    categorical_columns = data.select_dtypes(include='object').columns
    for column in categorical_columns:
        plt.figure(figsize=(5, 3))
        sns.countplot(data[column])
        plt.title(f'Countplot for {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()

#6. Seperate feature and target
def seperate_features_target(data,target_column):
    """
    Separate features and target variable

    Parameters:
    - data: pandas DataFrame, input data.
    - target_column: str, the column representing the target varible.

    Returns:
    - X: pandas DataFrame, features.
    - y: pandas Series, target variable.

    """

    X = data.drop(columns=[target_column],axis=1)
    y = data[target_column]

    return X,y
#7. Train test split
def perform_train_test_split(X, y, test_size=0.20, random_state=42):
    """
    Perform train-test split on the dataset.

    Parameters:
    - X: pandas DataFrame, features.
    - y: pandas Series, target variable.
    - test_size: float, optional, the proportion of the dataset to include in the test split (default is 0.2).
    - random_state: int or None, optional, seed for random number generation (default is None).

    Returns:
    - X_train: pandas DataFrame, features for training.
    - X_test: pandas DataFrame, features for testing.
    - y_train: pandas Series, target variable for training.
    - y_test: pandas Series, target variable for testing.
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

    return X_train, X_test, y_train, y_test
```

In [3]:

```
calories = pd.read_csv(r'C:\Users\HP\Downloads\calories (1).csv')
exercise = pd.read_csv(r'C:\Users\HP\Downloads\exercise.csv')
```

In [4]:
```
data = pd.merge(calories, exercise, on='User_ID')
```

In [5]:
```
data.head()
```

Out[5]:

	User_ID	Calories	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp
0	14733363	231.0	male	68	190.0	94.0	29.0	105.0	40.8
1	14861698	66.0	female	20	166.0	60.0	14.0	94.0	40.3
2	11179863	26.0	male	69	179.0	79.0	5.0	88.0	38.7
3	16180408	71.0	female	34	179.0	71.0	13.0	100.0	40.5
4	17771927	35.0	female	27	154.0	58.0	10.0	81.0	39.8

In [6]:
```
dataset_info_statistics(data)
```

```
Dataset Information:
<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   Calories    15000 non-null  float64
 2   Gender      15000 non-null  object 
 3   Age         15000 non-null  int64  
 4   Height      15000 non-null  float64
 5   Weight      15000 non-null  float64
 6   Duration    15000 non-null  float64
 7   Heart_Rate  15000 non-null  float64
 8   Body_Temp   15000 non-null  float64
dtypes: float64(6), int64(2), object(1)
memory usage: 1.0+ MB
None


Basic Statistics for Numerical Columns:
            User_ID      Calories           Age        Height        Weight  \
count  1.500000e+04  15000.000000  15000.000000  15000.000000  15000.000000   
mean   1.497736e+07     89.539533     42.789800    174.465133     74.966867   
std    2.872851e+06     62.456978     16.980264     14.258114     15.035657   
min    1.000116e+07      1.000000     20.000000    123.000000     36.000000   
25%    1.247419e+07     35.000000     28.000000    164.000000     63.000000   
50%    1.499728e+07     79.000000     39.000000    175.000000     74.000000   
75%    1.744928e+07    138.000000     56.000000    185.000000     87.000000   
max    1.999965e+07    314.000000     79.000000    222.000000    132.000000   

           Duration    Heart_Rate     Body_Temp  
count  15000.000000  15000.000000  15000.000000  
mean      15.530600     95.518533     40.025453  
std        8.319203      9.583328      0.779230  
min        1.000000     67.000000     37.100000  
25%        8.000000     88.000000     39.600000  
50%       16.000000     96.000000     40.200000  
75%       23.000000    103.000000     40.600000  
max       30.000000    128.000000     41.500000  
```

In [8]:
```
check_null(data)
```

```
Null Values in the Dataset:
```

Out[8]:
```
User_ID       0
Calories      0
Gender        0
Age           0
Height        0
Weight        0
Duration      0
Heart_Rate    0
Body_Temp     0
dtype: int64
```

In [9]:
```
#plot_graph(data)
```

In [11]:
```
data.columns
```

Out[11]:
```
Index(['User_ID', 'Calories', 'Gender', 'Age', 'Height', 'Weight', 'Duration',
       'Heart_Rate', 'Body_Temp'],
      dtype='object')
```

In [12]:
```
X,y = seperate_features_target(data,'Calories')
```

In [13]:
```
X = X.drop(columns=['User_ID'])
```

In [14]:
```
X_train,X_test,y_train,y_test = perform_train_test_split(X, y, test_size=0.20, random_state=42)
```

Column Transformer and Pipeline

In [16]:
```
preprocessor = ColumnTransformer(transformers=[
    ('ordinal',OrdinalEncoder(),['Gender']),
    ('num',StandardScaler(),['Age',
                             'Height',
                             'Weight',
                             'Duration',
                             'Heart_Rate',
                             'Body_Temp']),
],remainder='passthrough')
```

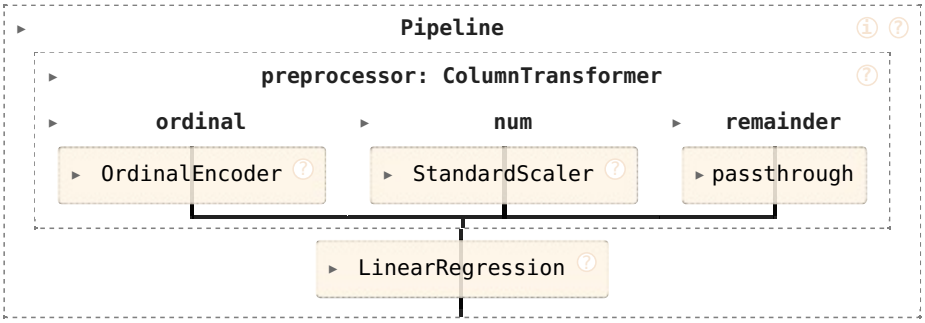
In [17]:
```
pipeline = Pipeline([("preprocessor",preprocessor),
                     ("model",LinearRegression())
                    ])
```

In [18]:
```
from sklearn import set_config
```

In [19]:
```
set_config(display='diagram')
```

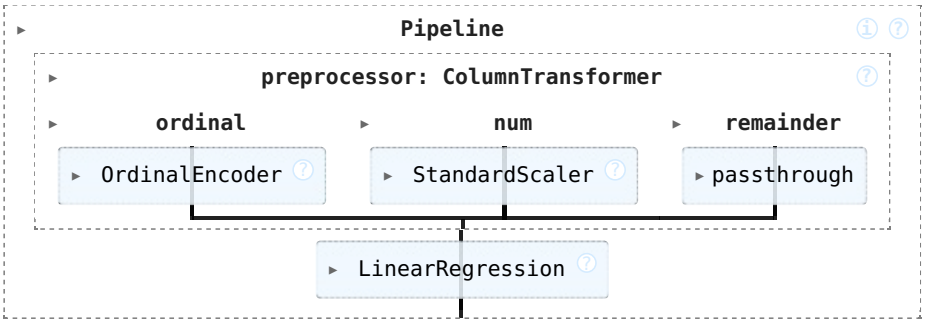
In [20]:
```
pipeline
```

Out[20]:



In [21]:
```
pipeline.fit(X_train,y_train)
```

Out[21]:



In [22]:
```
y_pred = pipeline.predict(X_test)
```

In [23]:
```
from sklearn.metrics import r2_score
```

In [24]:
```
r2_score(y_test,y_pred)
```

Out[24]:
```
0.9672937151257295
```

In [25]:
```
from sklearn.model_selection import KFold
```

In [27]:
```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

In [31]:
```
from sklearn.model_selection import cross_val_score
```

In [42]:
```
cv_results = cross_val_score(pipeline, X, y, cv=kfold, scoring='r2')
```

In [46]:
```
cv_results.mean()
```

Out[46]:
```
0.9671402283675841
```

In [54]:
```
from sklearn.metrics import mean_absolute_error
```

In [56]:
```
mean_absolute_error(y_test,y_pred)
```

Out[56]:
```
8.441513553849703
```

In [58]:
```
def model_scorer(model_name,model):

    output=[]


    output.append(model_name)

    pipeline = Pipeline([
    ('preprocessor',preprocessor),
    ('model',model)])

    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=42)

    pipeline.fit(X_train,y_train)

    y_pred = pipeline.predict(X_test)

    output.append(r2_score(y_test,y_pred))
    output.append(mean_absolute_error(y_test,y_pred))

    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    cv_results = cross_val_score(pipeline, X, y, cv=kfold, scoring='r2')
    output.append(cv_results.mean())

    return output
```

In [62]:
```
model_dict={
    'log':LinearRegression(),
    'RF':RandomForestRegressor(),
    'XGBR':XGBRegressor(),
}
```

In [64]:
```
model_output=[]
for model_name,model in model_dict.items():
    model_output.append(model_scorer(model_name,model))
```

In [65]:
```
model_output
```

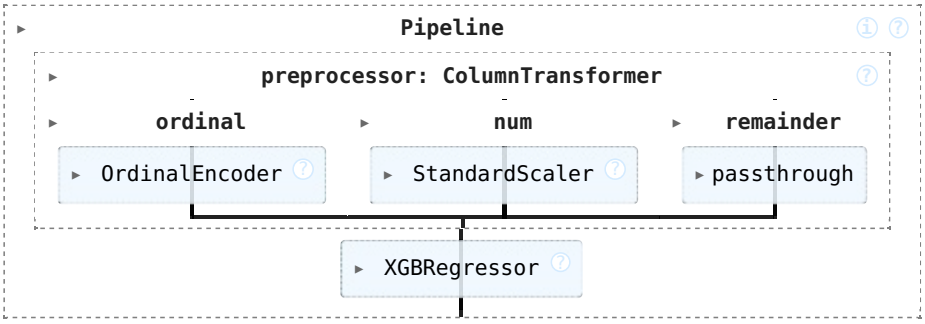
Out[65]:
```
[['log', 0.9672937151257295, 8.441513553849703, 0.9671402283675841],
 ['RF', 0.9981958807565852, 1.720766666666667, 0.9979185279853082],
 ['XGBR', 0.9988678909361673, 1.4981198125282924, 0.9988510864545181]]
```

In [66]:
```
preprocessor = ColumnTransformer(transformers=[
    ('ordinal',OrdinalEncoder(),['Gender']),
    ('num',StandardScaler(),['Age',
                             'Height',
                             'Weight',
                             'Duration',
                             'Heart_Rate',
                             'Body_Temp']),

],remainder='passthrough')
```

In [67]:
```
pipeline = Pipeline([
    ('preprocessor',preprocessor),
    ('model',XGBRegressor())

])
```

In [68]:
```
pipeline.fit(X,y)
```

Out[68]:



In [69]:
```
sample = pd.DataFrame({
   'Gender':'male',
    'Age':68,
```

```
    'Height':190.0,
    'Weight':94.0,
    'Duration':29.0,
    'Heart_Rate':105.0,
    'Body_Temp':40.8,
},index=[0])
```

In [70]:
```
pipeline.predict(sample)
```

Out[70]:
```
array([231.0721], dtype=float32)
```

Save The Model

In [72]:
```
import pickle
```

In [73]:
```
with open('pipeline.pkl','wb') as f:
    pickle.dump(pipeline,f)
```

In [74]:
```
with open('pipeline.pkl','rb') as f:
    pipeline_saved = pickle.load(f)
```

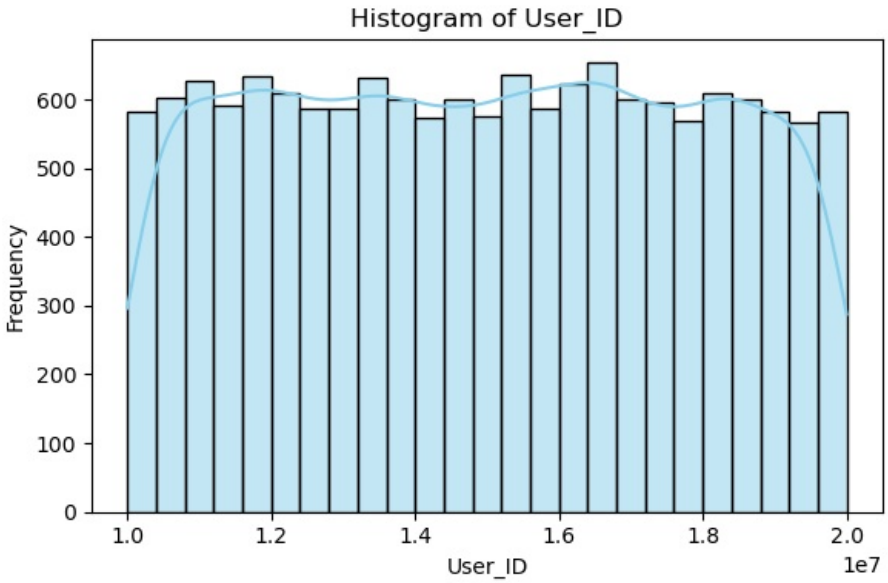
In [75]:
```
result = pipeline_saved.predict(sample)
```

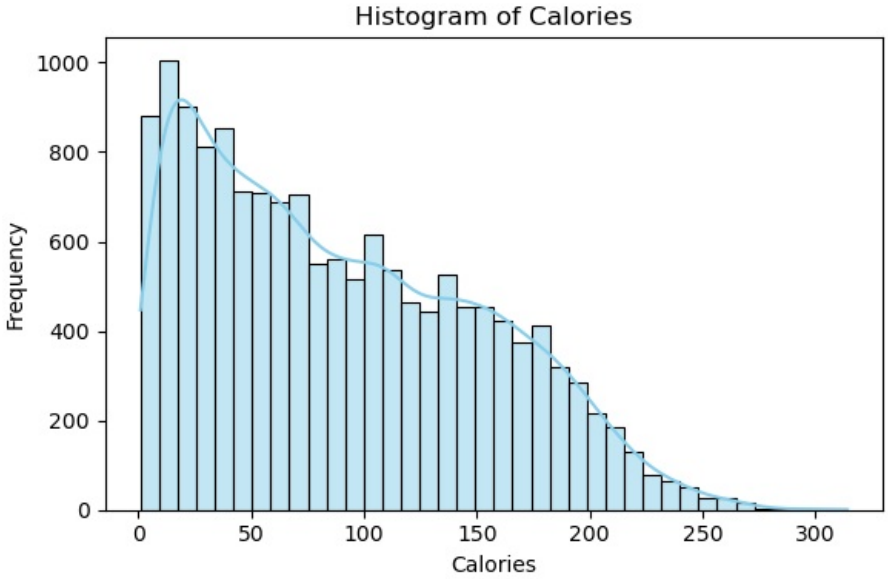
In [76]:
```
result
```

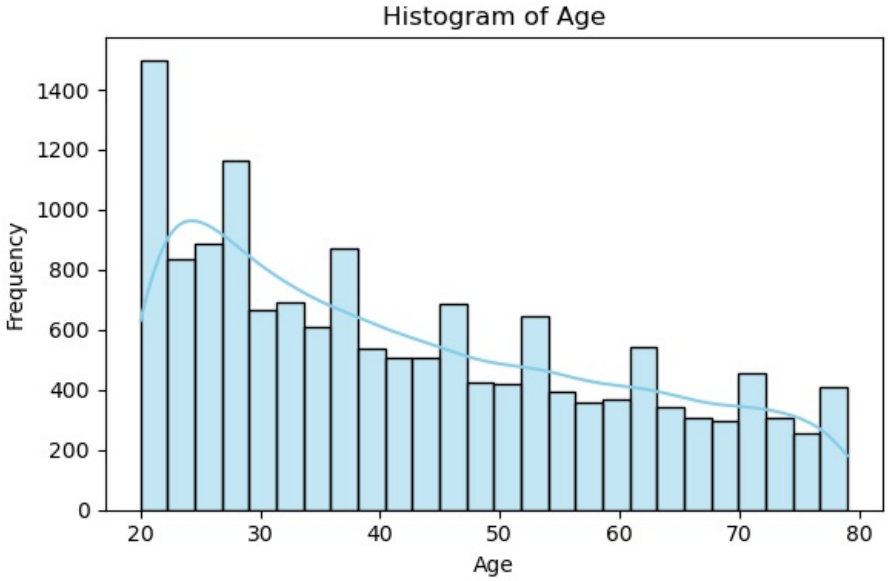
Out[76]:
```
array([231.0721], dtype=float32)
```

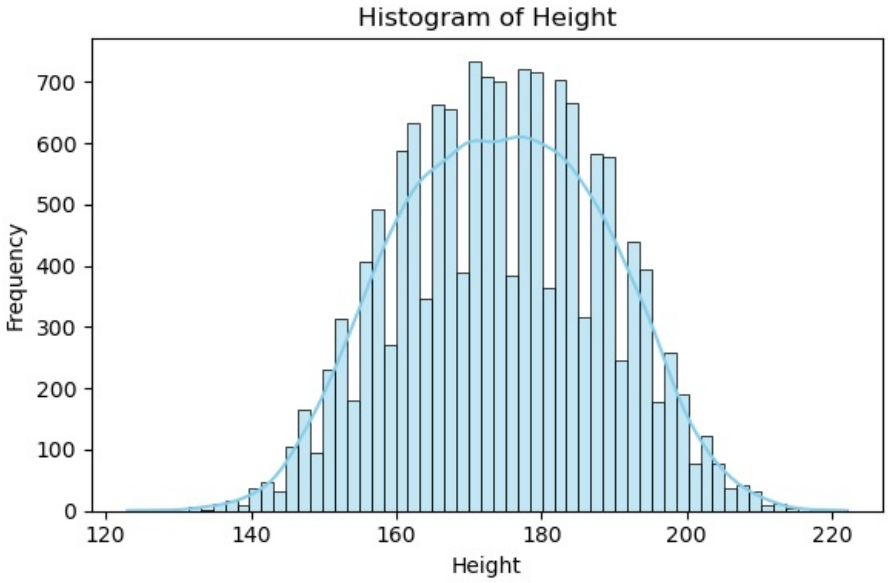
In [92]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

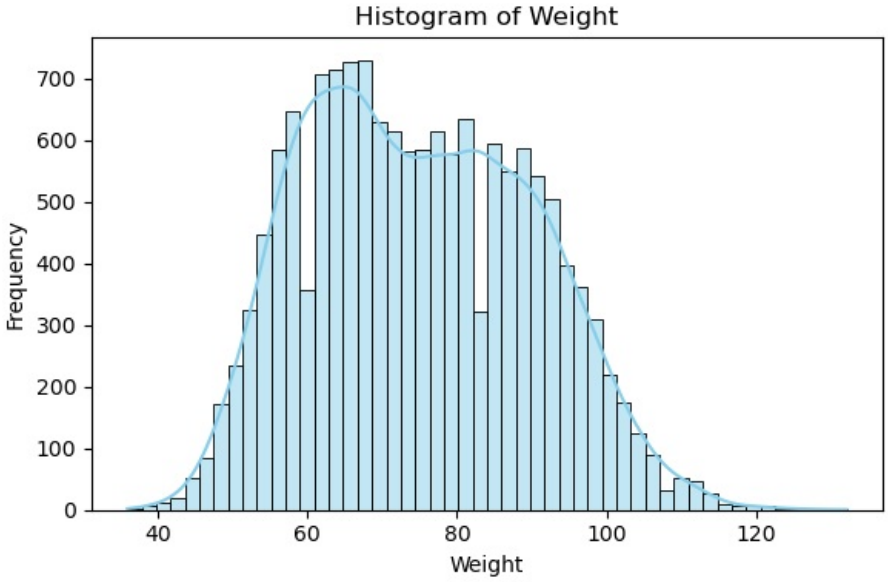
In [94]:
```
numerical_columns = data.select_dtypes(include=np.number).columns
for col in numerical_columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(data[col], kde=True, color='skyblue')
    plt.title(f"Histogram of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()
```

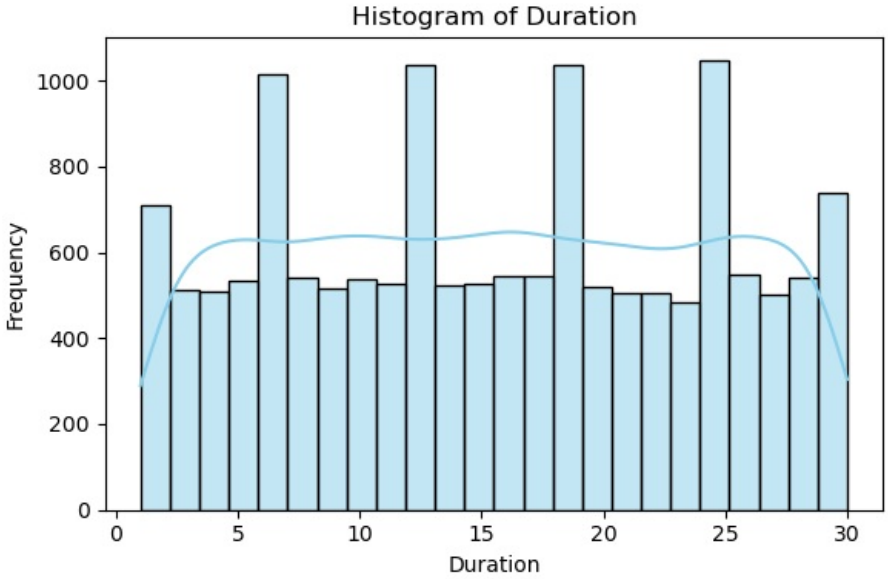


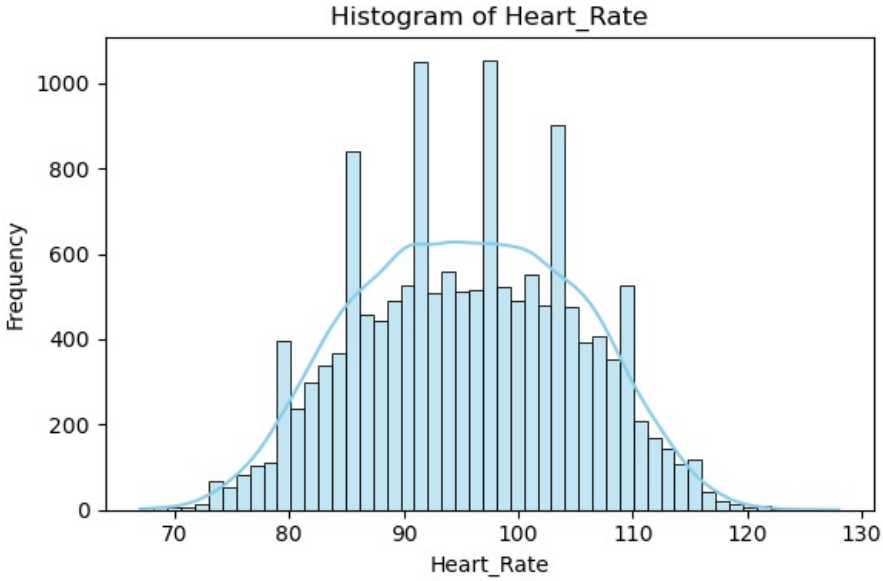

Histogram of Calories
Frequency
Calories
0
200
400
600
800
1000
0
50
100
150
200
250
300

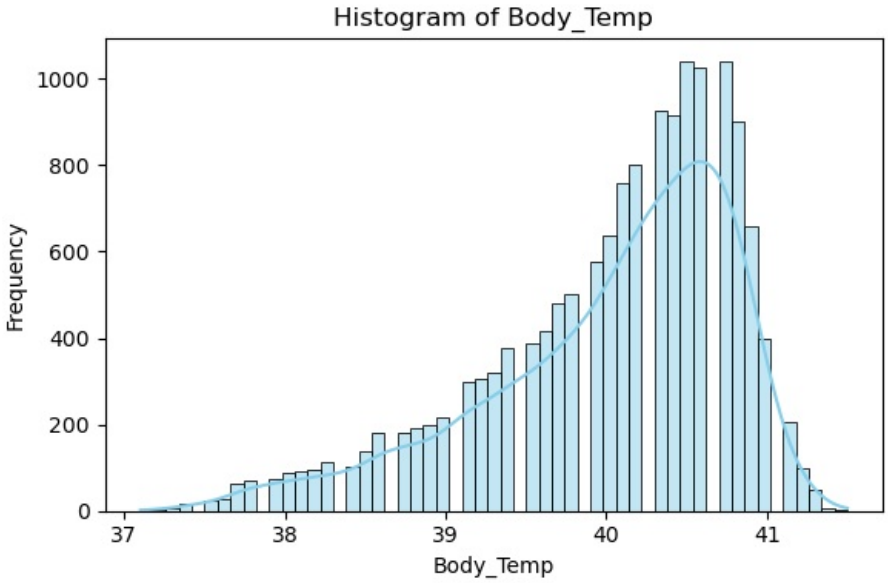

Histogram of Age
Frequency
Age
0
200
400
600
800
1000
1200
1400
20
30
40
50
60
70
80


Histogram of Height
Frequency
Height
0
100
200
300
400
500
600
700
120
140
160
180
200
220

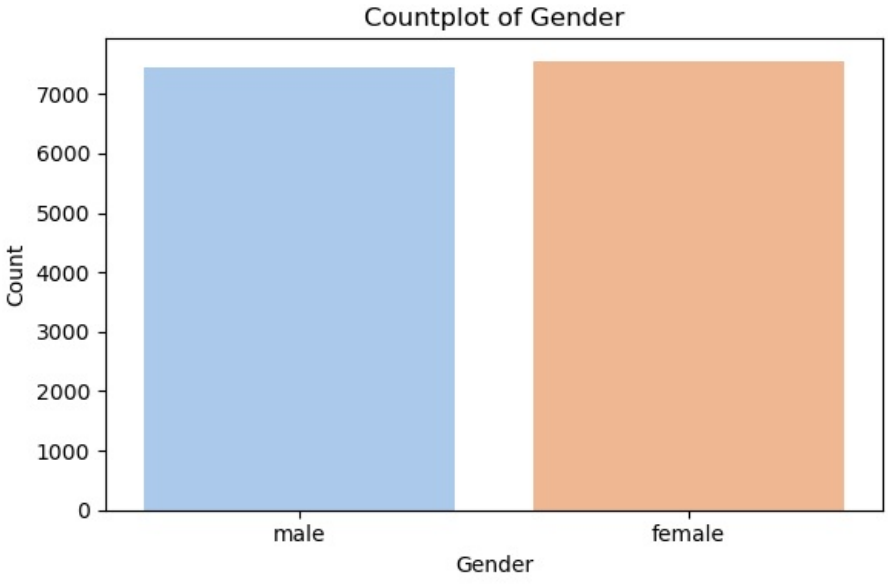

Histogram of Weight
Frequency
Weight
0
100
200
300
400
500
600
700
40
60
80
100
120


Histogram of Duration
Frequency
Duration
0
200
400
600
800
1000
0
5
10
15
20
25
30


Histogram of Heart_Rate
Frequency
Heart_Rate
0
200
400
600
800
1000
70
80
90
100
110
120
130



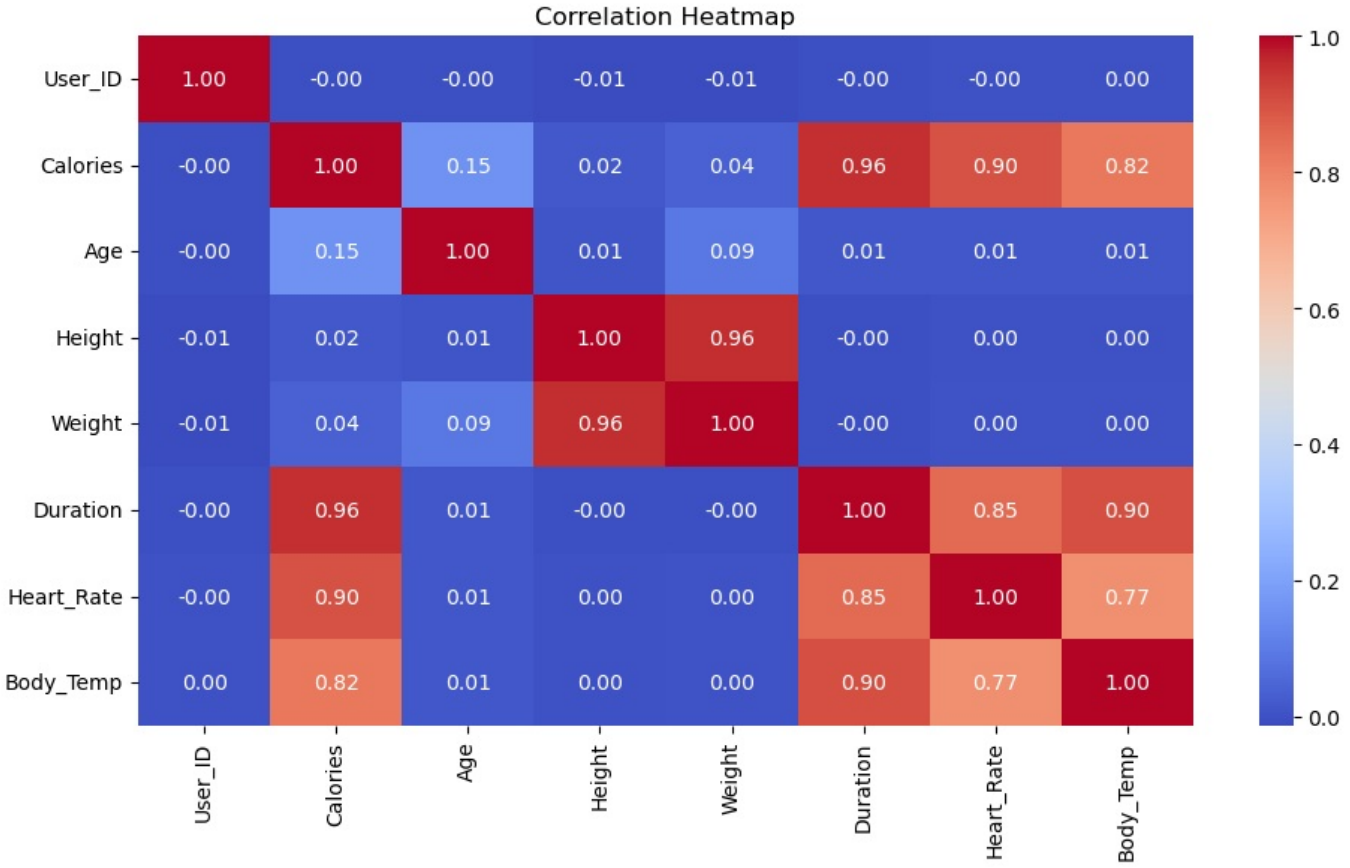
In [96]:
```
categorical_columns = data.select_dtypes(include='object').columns
for col in categorical_columns:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=data[col], palette="pastel")
    plt.title(f"Countplot of {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.tight_layout()
    plt.show()
```



In [100…
```
plt.figure(figsize=(10, 6))

# Use only numeric columns for correlation
```

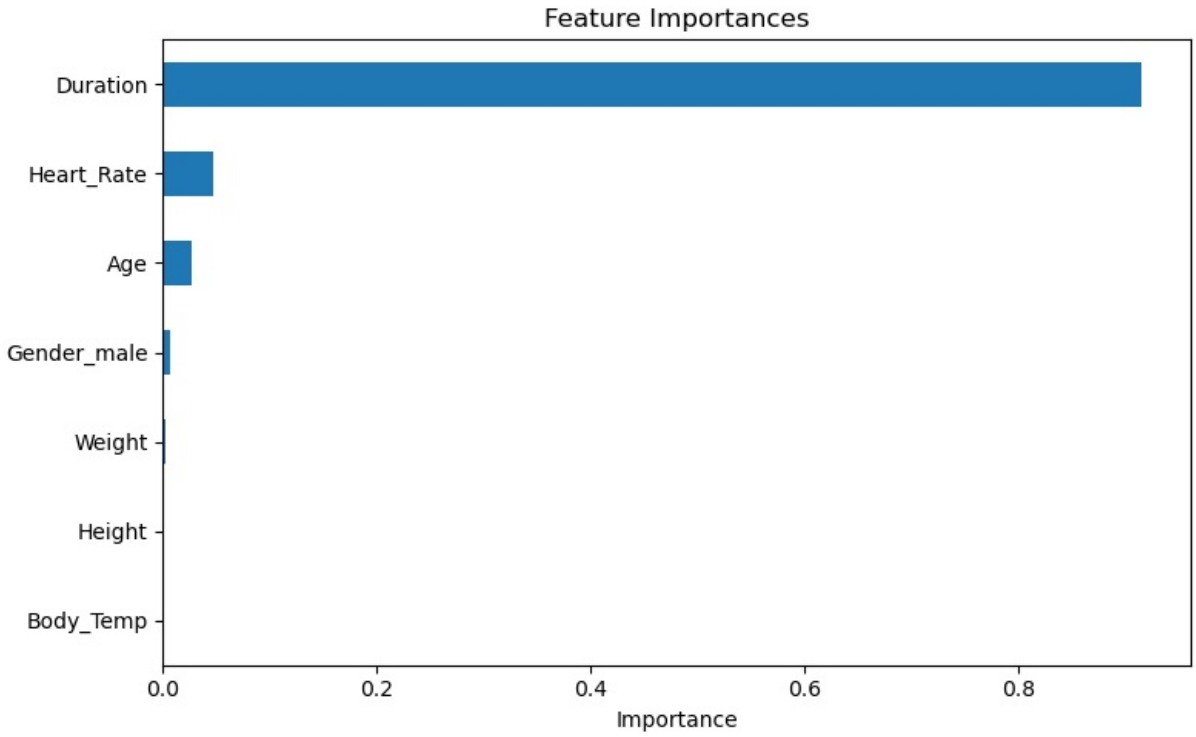
```
numeric_data = data.select_dtypes(include=[np.number])

sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



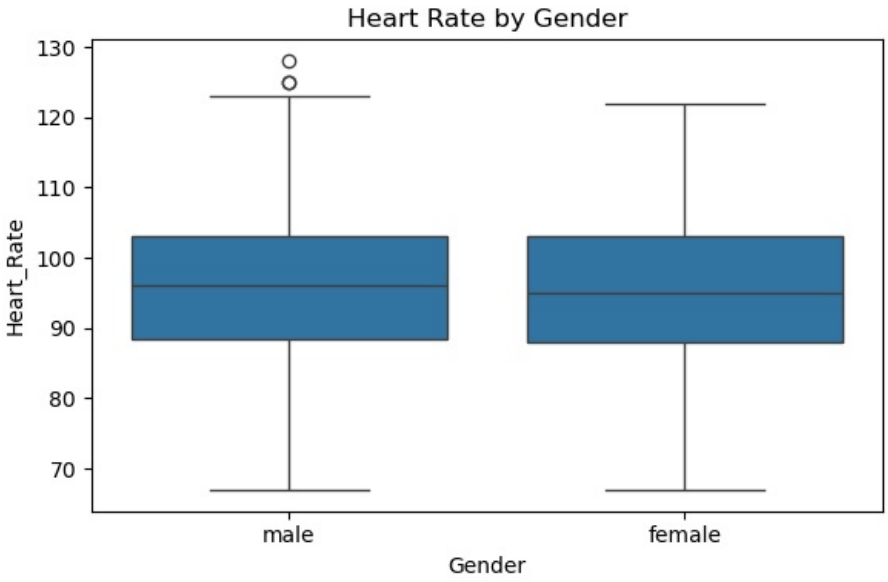
In [104…

```
from sklearn.ensemble import RandomForestRegressor

X = data.drop(columns=['User_ID', 'Calories'])
X = pd.get_dummies(X, drop_first=True)
y = data['Calories']

model = RandomForestRegressor()
model.fit(X, y)

importances = pd.Series(model.feature_importances_, index=X.columns)
importances.sort_values().plot(kind='barh', figsize=(8, 5), title="Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



In [105…
```
plt.figure(figsize=(6, 4))
sns.boxplot(x='Gender', y='Calories', data=data)
plt.title("Calories Burnt by Gender")
plt.tight_layout()
plt.show()
```

In [108…
```
plt.figure(figsize=(6, 4))
sns.boxplot(x='Gender', y='Heart_Rate', data=data)
plt.title("Heart Rate by Gender")
plt.tight_layout()
plt.show()
```



In [110…
```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor  # Make sure xgboost is installed

# Preprocessing
X = data.drop(columns=['User_ID', 'Calories'])
X = pd.get_dummies(X, drop_first=True)
y = data['Calories']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [112…
```python
def evaluate_model(name, model):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\n {name} Performance:")
    print("R² Score (Accuracy):", r2_score(y_test, y_pred))
    print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

In [114…
```python
# Random Forest
rf_model = RandomForestRegressor(random_state=42)
evaluate_model("Random Forest Regressor", rf_model)

# XGBoost
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
evaluate_model("XGBoost Regressor", xgb_model)
```

```
 Random Forest Regressor Performance:
R² Score (Accuracy): 0.998221953940476
Mean Absolute Error: 1.7139333333333333
Mean Squared Error: 7.175823133333334
Root Mean Squared Error: 2.678772691613332

 XGBoost Regressor Performance:
R² Score (Accuracy): 0.9988680981634738
Mean Absolute Error: 1.4984578529596329
Mean Squared Error: 4.568119785030486
Root Mean Squared Error: 2.1373160236685838
```

In [ ]:
```python
import pickle
import pandas as pd
from tkinter import *
import threading

# Function to load the machine learning model in the background
def load_model():
    global pipeline
    with open('pipeline.pkl', 'rb') as f:
        pipeline = pickle.load(f)

# Function to handle the prediction (in a separate thread)
def show_entry():
    # Disable the predict button to prevent multiple clicks
    predict_button.config(state=DISABLED)
```

```
    # Start the prediction in a new thread
    thread = threading.Thread(target=make_prediction)
    thread.daemon = True  # This allows the thread to be killed when the main program exits
    thread.start()

# Function to make the prediction
def make_prediction():
    try:
        # Collect the inputs from the user
        p1 = str(gender_var.get())
        p2 = float(e2.get())
        p3 = float(e3.get())
        p4 = float(e4.get())
        p5 = float(e5.get())
        p6 = float(e6.get())
        p7 = float(e7.get())

        # Prepare the input data as a DataFrame
        sample = pd.DataFrame({
            'Gender': [p1],
            'Age': [p2],
            'Height': [p3],
            'Weight': [p4],
            'Duration': [p5],
            'Heart_Rate': [p6],
            'Body_Temp': [p7],
        })

        # Make the prediction
        result = pipeline.predict(sample)

        # Schedule the update of the result label in the main thread
        master.after(0, update_result, result[0])

    except Exception as e:
        # Handle errors and update result label
        master.after(0, update_result, f"Error: {e}")

    # Re-enable the predict button after the prediction is done
    master.after(0, enable_button)

# Function to update the result label
def update_result(result):
    result_label.config(text=f"Amount of Calories Burnt: {result:.2f}")

# Function to enable the predict button again after prediction
def enable_button():
    predict_button.config(state=NORMAL)

# Setting up the main window
master = Tk()
master.title("Calories Burnt Prediction using Machine Learning")
master.configure(bg="#f0f0f0")  # Light grey background

# Load the model when the application starts in a separate thread
load_thread = threading.Thread(target=load_model)
load_thread.daemon = True
load_thread.start()

# Header Label with background color
header = Label(master, text="Calories Burnt Prediction", bg="#4CAF50", fg="white", font=("Helvetica", 18, "bold
header.grid(row=0, columnspan=2, pady=20)

# Gender selection label and options
Label(master, text="Select Gender", bg="#f0f0f0", font=("Arial", 12)).grid(row=1, sticky=W, padx=10)
gender_var = StringVar()
gender_var.set("male")  # Default gender
gender_menu = OptionMenu(master, gender_var, "male", "female")
gender_menu.config(width=15, font=("Arial", 12), bg="#f1f1f1", relief="solid")
gender_menu.grid(row=1, column=1)

# Create Entry widgets for inputs
e2 = Entry(master, font=("Arial", 12), bd=2, relief="solid", width=20)
e3 = Entry(master, font=("Arial", 12), bd=2, relief="solid", width=20)
e4 = Entry(master, font=("Arial", 12), bd=2, relief="solid", width=20)
e5 = Entry(master, font=("Arial", 12), bd=2, relief="solid", width=20)
e6 = Entry(master, font=("Arial", 12), bd=2, relief="solid", width=20)
e7 = Entry(master, font=("Arial", 12), bd=2, relief="solid", width=20)

# Other input fields with labels
input_labels = [
    ("Enter Your Age", e2),
    ("Enter Your Height (cm)", e3),
```

```
    ("Enter Your Weight (kg)", e4),
    ("Enter Duration (min)", e5),
    ("Enter Heart Rate", e6),
    ("Enter Body Temperature (°C)", e7),
]

# Place labels and input fields in the grid
for i, (text, entry_widget) in enumerate(input_labels, 2):
    Label(master, text=text, bg="#f0f0f0", font=("Arial", 12)).grid(row=i, sticky=W, padx=10)
    entry_widget.grid(row=i, column=1, padx=10, pady=5)

# Prediction button
predict_button = Button(master, text="Predict", command=show_entry, font=("Arial", 14), bg="#4CAF50", fg="white
predict_button.grid(row=8, columnspan=2, pady=20)

# Label to display the result of the prediction
result_label = Label(master, text="Amount of Calories Burnt", font=("Helvetica", 14), bg="#f0f0f0", fg="#4CAF50
result_label.grid(row=9, columnspan=2, pady=10)

# Start the Tkinter event loop
master.mainloop()
```

In [18]:
```
print("hello")
```

```
hello
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: