Random Regressor for Calories

March 15, 2025

[8]:
```
## Import required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
```

[9]:
```
## Load the dataset
df = pd.read_csv("C:/Users/PC/OneDrive/Desktop/Data Science/Datasets/Datasets/
 ↪calories.csv")
```

[10]:
```
## Inspect the few observations of the dataset
df.head()
```

[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

[11]:
```
## Assess the structure of the dataset
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
```

[13]:
```
## Check for duplicates
df.duplicated().sum()
```

[13]:
```
0
```

[14]:
```
## Check for missing values
df.isnull().sum()
```

[14]:
```
User_ID       0
Gender        0
Age           0
Height        0
Weight        0
Duration      0
Heart_Rate    0
Body_Temp     0
Calories      0
dtype: int64
```

[16]:
```
## Summary statistics
df = df.drop(columns = ["User_ID"])
df.describe()
```

[16]:

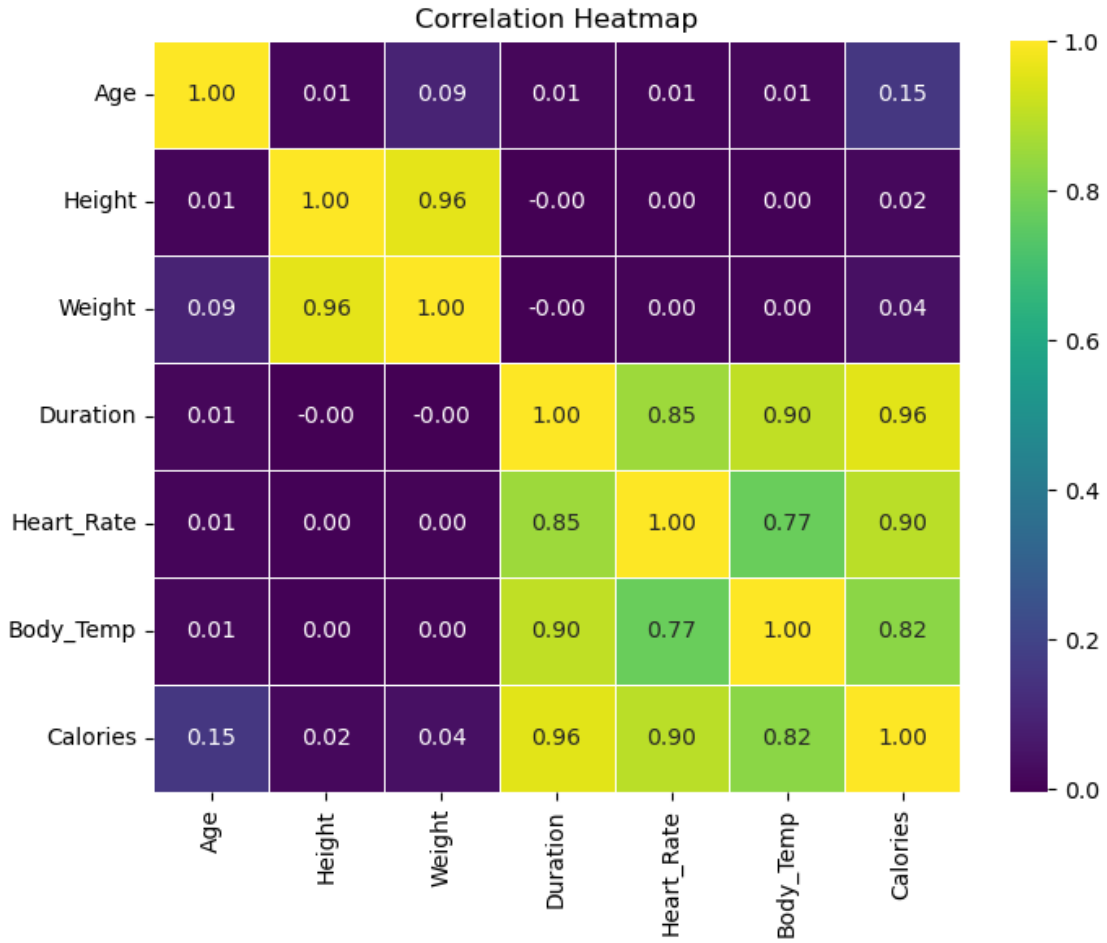
	Age	Height	Weight	Duration	Heart_Rate
count	15000.000000	15000.000000	15000.000000	15000.000000	15000.000000
mean	42.789800	174.465133	74.966867	15.530600	95.518533
std	16.980264	14.258114	15.035657	8.319203	9.583328
min	20.000000	123.000000	36.000000	1.000000	67.000000
25%	28.000000	164.000000	63.000000	8.000000	88.000000
50%	39.000000	175.000000	74.000000	16.000000	96.000000
75%	56.000000	185.000000	87.000000	23.000000	103.000000
max	79.000000	222.000000	132.000000	30.000000	128.000000

```
          Body_Temp      Calories
count  15000.000000  15000.000000
mean      40.025453     89.539533
std        0.779230     62.456978
min       37.100000      1.000000
25%       39.600000     35.000000
50%       40.200000     79.000000
75%       40.600000    138.000000
max       41.500000    314.000000
```

[17]:
```
## Perform correlation analysis
numeric_vars = df.select_dtypes(include = ["float64", "int64"])
correlation_matrix = numeric_vars.corr()
print(correlation_matrix)
```

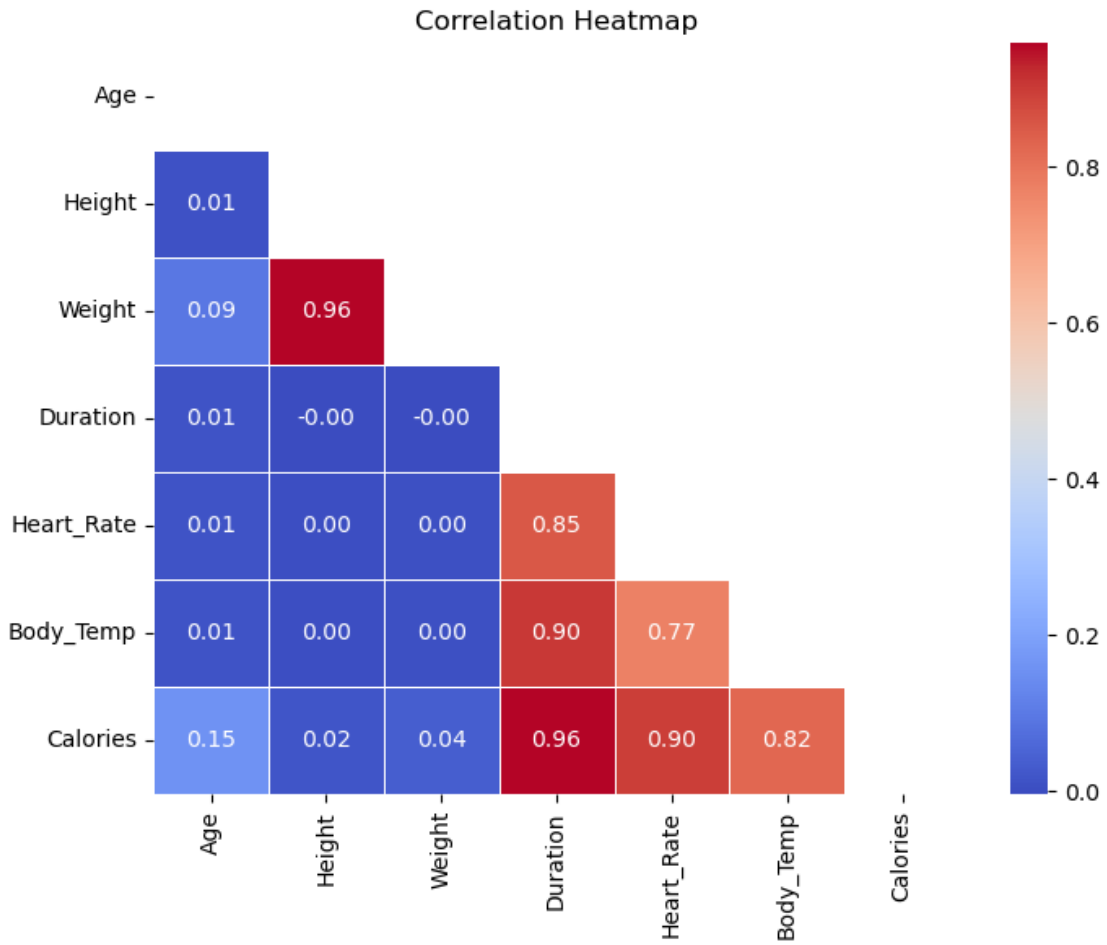
```
                Age    Height    Weight  Duration  Heart_Rate  Body_Temp  \
Age        1.000000  0.009554  0.090094  0.013247    0.010482   0.013175
Height     0.009554  1.000000  0.958451 -0.004625    0.000528   0.001200
Weight     0.090094  0.958451  1.000000 -0.001884    0.004311   0.004095
Duration   0.013247 -0.004625 -0.001884  1.000000    0.852869   0.903167
Heart_Rate 0.010482  0.000528  0.004311  0.852869    1.000000   0.771529
Body_Temp  0.013175  0.001200  0.004095  0.903167    0.771529   1.000000
Calories   0.154395  0.017537  0.035481  0.955421    0.897882   0.824558

            Calories
Age         0.154395
Height      0.017537
Weight      0.035481
Duration    0.955421
Heart_Rate  0.897882
Body_Temp   0.824558
Calories    1.000000
```

[18]:
```
## Visualize your correlation matrix
plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap="viridis", fmt=".2f",␣
 ↪linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```



[19]:
```
## Label encoding
## Select categorical columns
categorical_cols = df.select_dtypes(include = ["object"]).columns
## Initialize the label encoder
label_encoder = LabelEncoder()
## Apply label encooding to selected columns
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

[21]:
```
## Self-Correlation
plt.figure(figsize=(8,6))
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))  # Mask upper␣
 ↪triangle
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",␣
 ↪linewidths=0.5, mask=mask)
plt.title("Correlation Heatmap")
plt.show()
```



[22]:
```
## Define Independent and Dependent Variables
X = df.drop(columns = ["Calories"])
y = df["Calories"]
```

[23]:
```
## Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[24]:
```
## Train the Random Forest Model
## Create the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
## Train the model
rf_model.fit(X_train, y_train)
```

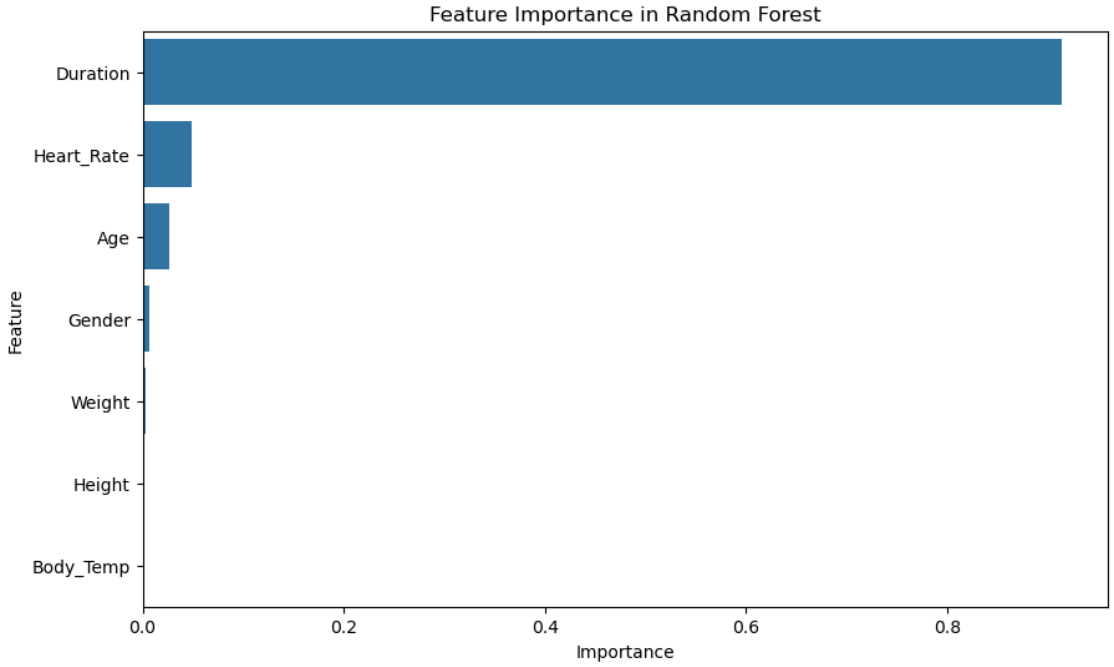
[24]:
```
RandomForestRegressor(random_state=42)
```

[25]:
```
## Make Predictions
y_pred = rf_model.predict(X_test)
```

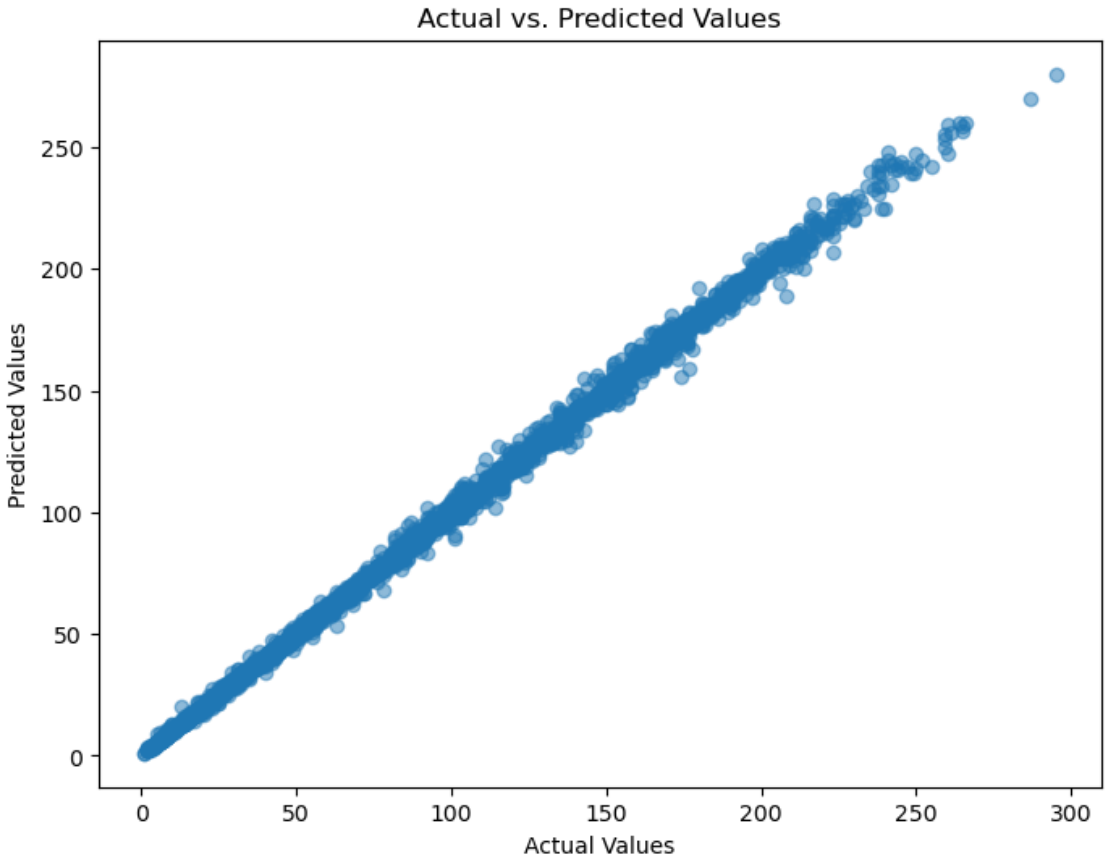
[26]:
```
## Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error: {rmse}")
r2 = r2_score(y_test, y_pred)
print(f"R-Squared Score: {r2}")
```

```
Mean Squared Error: 7.200538999999999
Root Mean Squared Error: 2.6833820078401063
R-Squared Score: 0.9982158297720679
```

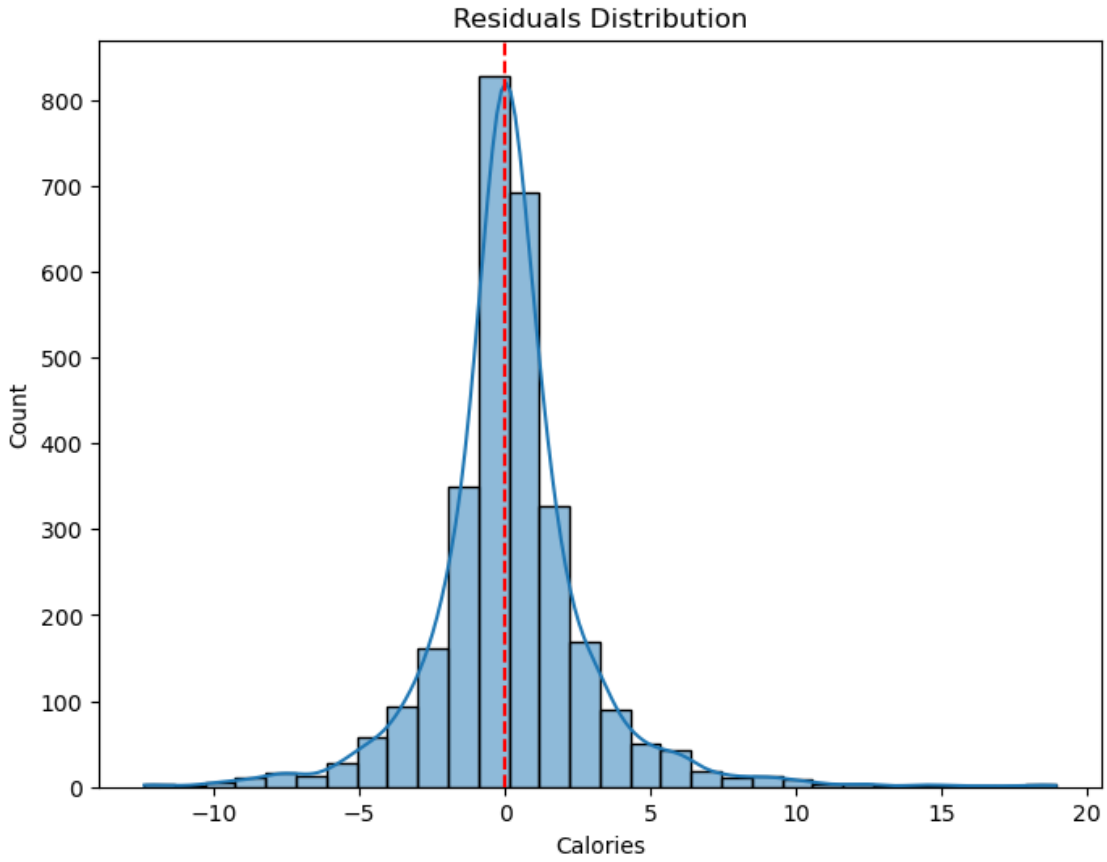
[27]:
```
## Feature importance analysis
feature_importance = rf_model.feature_importances_

# Convert to DataFrame
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance':␣
 ↪feature_importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot Feature Importance
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title("Feature Importance in Random Forest")
plt.show()
```



[28]:
```
## Visualizing predictions
## Plot Actual vs. Predicted Values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()
```



[29]:
```
## Residual plot
residuals = y_test - y_pred

plt.figure(figsize=(8, 6))
sns.histplot(residuals, bins=30, kde=True)
plt.axvline(0, color='red', linestyle='dashed')
plt.title("Residuals Distribution")
plt.show()
```



[30]:
```
## Safe the model
joblib.dump(rf_model, "random_forest_model.pkl")
```

[30]:
```
['random_forest_model.pkl']
```