Predicting Next Exercise, Time and Duration in Fitness Tracking Data

Author : Sunjida Haque

Student iD : 00368852

In the era of fitness tracking and wearables, our project, harnesses the power of machine learning to forecast users' next exercise categories, time and the expected duration of their activities. Leveraging historical fitness tracking data, we tackle both classification and regression tasks to provide real-time exercise recommendations and customizable training plans. This predictive technology holds the potential to optimize users' fitness routines, empower healthier choices, and open new avenues in fitness coaching, health monitoring, and product design.

We will start from **Exploratory Data Analysis**, then we do **Data Cleaning**, we will do **Feature Enginnering and Feature Selection** and finally build the **Machine Learning** model.

In [84]:
```
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
```

Loading the datasets

Here we will load the datasets from our folder where the json files are saved. These json files are the records of every activity performed by the user.

In [4]:
```
folder_path = "F:\Projects\Workout ML Project\WorkoutData_2017to2020-20231029T130842Z-001\WorkoutData_2017to2020"
# Initialize lists to store the number of columns in each DataFrame
num_columns_list = []

# Loop through all files in the folder
for filename in os.listdir(folder_path):
    if filename.endswith('.json'):
        file_path = os.path.join(folder_path, filename)

        # Read the JSON file into a Pandas DataFrame
        df = pd.read_json(file_path)

        # Get the number of columns in the DataFrame and add it to the list
        num_columns = len(df.columns)
        num_columns_list.append(num_columns)

# Calculate the maximum, minimum, and average number of columns
max_num_columns = max(num_columns_list)
min_num_columns = min(num_columns_list)
avg_num_columns = sum(num_columns_list) / len(num_columns_list)

print(f"Maximum number of columns: {max_num_columns}")
print(f"Minimum number of columns: {min_num_columns}")
print(f"Average number of columns: {avg_num_columns:.2f}")
```

```
Maximum number of columns: 16
Minimum number of columns: 10
Average number of columns: 15.14
```

Trying out the script provided by teacher

In [6]:

```
file_list = os.listdir(folder_path)

#%% Read example file
mov_ex0 = pd.read_json(folder_path+'/'+file_list[0], typ='series')

# Inspect example file
print(mov_ex0[14]['points'])

#%% Function definitions (HINT: you can move functions in a separate file to
# keep the length of the analysis script reasonable...)

def read_file_to_df(filename):
    data = pd.read_json(filename, typ='series')
    value = []
    key = []
    for j in list(range(0, data.size)):
        if list(data[j].keys())[0] != 'points':
            key.append(list(data[j].keys())[0])
            value.append(list(data[j].items())[0][1])
            dictionary = dict(zip(key, value))


    if list(data[j].keys())[0] == 'points':
        try:
            start = list(list(list(data[data.size-1].items()))[0][1][0][0].items())[0][1][0]
            dictionary['start_lat'] = list(start[0].items())[0][1]
            dictionary['start_long'] = list(start[1].items())[0][1]
            dictionary['end_lat'] = list(start[0].items())[0][1]
            dictionary['end_long'] = list(start[1].items())[0][1]
        except:
            print('No detailed data recorded')


    df = pd.DataFrame(dictionary, index = [0])

    return df
```


```
ude': 27.730344}]]}, {'altitude': 167.5}, {'distance_km': 0.35000000000000003}, {'speed_kmh': 3.6}, {'timestamp': 'Sun Ja
n 01 08:59:32 UTC 2017'}], [{'location': [[{'latitude': 64.234227}, {'longitude': 27.730152}]]}, {'altitude': 167.5}, {'d
istance_km': 0.36}, {'speed_kmh': 3.6}, {'timestamp': 'Sun Jan 01 08:59:40 UTC 2017'}], [{'location': [[{'latitude': 64.2
34309}, {'longitude': 27.73002}]]}, {'altitude': 172}, {'distance_km': 0.37}, {'speed_kmh': 3.6}, {'timestamp': 'Sun Jan
01 08:59:50 UTC 2017'}], [{'location': [[{'latitude': 64.234373}, {'longitude': 27.729824999999998}]]}, {'altitude': 17
6}, {'distance_km': 0.39}, {'speed_kmh': 2.7}, {'timestamp': 'Sun Jan 01 09:00:22 UTC 2017'}], [{'location': [[{'latitud
e': 64.234462}, {'longitude': 27.729689}]]}, {'altitude': 177}, {'distance_km': 0.4}, {'speed_kmh': 2.7}, {'timestamp':
'Sun Jan 01 09:00:30 UTC 2017'}], [{'location': [[{'latitude': 64.23451}, {'longitude': 27.729494}]]}, {'altitude': 175.
5}, {'distance_km': 0.41000000000000003}, {'speed_kmh': 1.8}, {'timestamp': 'Sun Jan 01 09:00:57 UTC 2017'}], [{'locatio
n': [[{'latitude': 64.23459}, {'longitude': 27.729353}]]}, {'altitude': 177.5}, {'distance_km': 0.42}, {'speed_kmh': 1.
8}, {'timestamp': 'Sun Jan 01 09:01:10 UTC 2017'}], [{'location': [[{'latitude': 64.234661}, {'longitude': 27.729224}]]},
{'altitude': 177.5}, {'distance_km': 0.43}, {'speed_kmh': 1.8}, {'timestamp': 'Sun Jan 01 09:01:24 UTC 2017'}], [{'locati
on': [[{'latitude': 64.234739}, {'longitude': 27.729101}]]}, {'altitude': 181}, {'distance_km': 0.44}, {'speed_kmh': 2.
7}, {'timestamp': 'Sun Jan 01 09:01:40 UTC 2017'}], [{'location': [[{'latitude': 64.234838}, {'longitude': 27.729083}]]},
{'altitude': 182.5}, {'distance_km': 0.45}, {'speed_kmh': 1.8}, {'timestamp': 'Sun Jan 01 09:01:52 UTC 2017'}], [{'locati
on': [[{'latitude': 64.234918}, {'longitude': 27.728933}]]}, {'altitude': 183.5}, {'distance_km': 0.46}, {'speed_kmh': 2.
7}, {'timestamp': 'Sun Jan 01 09:02:04 UTC 2017'}], [{'location': [[{'latitude': 64.23501}, {'longitude': 27.7288}]]},
{'altitude': 185}, {'distance_km': 0.47000000000000003}, {'speed_kmh': 2.7}, {'timestamp': 'Sun Jan 01 09:02:16 UTC 201
7'}], [{'location': [[{'latitude': 64.235107}, {'longitude': 27.728772}]]}, {'altitude': 186}, {'distance_km': 0.49}, {'s
```

In [8]:
```
#%% Read all files in a loop

# Create Empty DataFrame
df_res = pd.DataFrame()

# Read files to a common dataframe
for filename in file_list:
    print('\n'+filename)
    df_process = read_file_to_df(folder_path +'/'+ filename)
    df_res = pd.concat([df_res, df_process], 0)

df_res.reset_index(drop=True, inplace = True)
```

```
2017-01-01 08_54_23.0.json

2017-01-01 15_02_04.0.json
No detailed data recorded

2017-01-01 17_47_03.0.json

2017-01-02 08_57_23.0.json

2017-01-02 16_20_51.0.json

2017-01-03 09_25_22.0.json

2017-01-03 17_41_24.0.json
No detailed data recorded

2017-01-03 17_54_10.0.json
```

Now that our datasets have been loaded into a single dataframe, we will start exploring it.

In [9]:
```
df_res.head(30)
```

Out[9]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitu
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23.0	2017-01-01 08:53:04.0	2017-01-01 09:27:49.0	2084	2.15	171.65100	145.5	
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04.0	2017-01-01 14:01:00.0	2017-01-01 14:41:00.0	2400	0.00	393.33300	NaN	
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03.0	2017-01-01 17:46:00.0	2017-01-01 18:12:07.0	1566	1.69	132.16800	126.5	
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23.0	2017-01-02 08:55:52.0	2017-01-02 09:26:06.0	1812	2.07	157.82800	81.0	
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51.0	2017-01-02 16:13:34.0	2017-01-02 16:54:52.0	2444	5.87	591.40400	97.5	
5	WALKING	TRACK_MOBILE	2017-01-03 09:25:22.0	2017-01-03 09:19:16.0	2017-01-03 09:53:05.0	1963	2.15	167.02400	138.0	
6	SKIING_CROSS_COUNTRY	INPUT_MANUAL_MOBILE	2017-01-03 17:41:24.0	2017-01-03 11:40:00.0	2017-01-03 12:30:00.0	3000	7.40	787.00000	NaN	
7	WALKING	TRACK_MOBILE	2017-01-03 17:54:10.0	2017-01-03 17:52:29.0	2017-01-03 18:24:36.0	1927	2.46	178.90700	127.0	
8	WALKING	TRACK_MOBILE	2017-01-04 09:16:33.0	2017-01-04 08:52:12.0	2017-01-04 09:18:36.0	1583	1.75	135.38400	143.5	
9	WALKING	TRACK_MOBILE	2017-01-04 09:52:22.0	2017-01-04 09:50:10.0	2017-01-04 10:18:29.0	1695	1.10	109.32800	0.0	
10	WALKING	TRACK_MOBILE	2017-01-04 18:04:33.0	2017-01-04 15:47:22.0	2017-01-04 16:01:53.0	869	1.12	81.13470	117.5	
11	WALKING	TRACK_MOBILE	2017-01-04 18:05:32.0	2017-01-04 18:04:30.0	2017-01-04 18:20:42.0	854	0.99	75.00090	136.5	
12	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-04 18:05:48.0	2017-01-04 17:05:00.0	2017-01-04 18:00:00.0	3300	0.00	540.83300	NaN	
13	WALKING	TRACK_MOBILE	2017-01-05 08:42:51.0	2017-01-05 08:41:33.0	2017-01-05 09:08:28.0	1614	1.95	145.12400	149.0	
14	WALKING	TRACK_MOBILE	2017-01-05 18:06:57.0	2017-01-05 17:46:41.0	2017-01-05 18:09:10.0	1349	1.66	122.58700	124.5	
15	WALKING	TRACK_MOBILE	2017-01-06 09:19:39.0	2017-01-06 08:54:30.0	2017-01-06 09:19:28.0	1496	1.79	133.76800	143.5	
16	WALKING	TRACK_MOBILE	2017-01-06 15:04:55.0	2017-01-06 15:03:19.0	2017-01-06 15:26:39.0	1398	1.83	131.73200	126.0	
17	WALKING	TRACK_MOBILE	2017-01-07 09:34:13.0	2017-01-07 09:32:44.0	2017-01-07 10:04:02.0	1878	2.15	163.77400	150.0	
18	SWIMMING	INPUT_MANUAL_MOBILE	2017-01-07 15:23:19.0	2017-01-07 14:22:00.0	2017-01-07 14:52:00.0	1800	1.00	355.08800	NaN	
19	WALKING	TRACK_MOBILE	2017-01-07 17:43:47.0	2017-01-07 17:41:20.0	2017-01-07 18:14:41.0	2000	2.57	186.40300	149.5	
20	WALKING	TRACK_MOBILE	2017-01-08 07:03:22.0	2017-01-08 07:02:14.0	2017-01-08 07:34:25.0	1929	1.94	156.74200	146.0	
21	WALKING	TRACK_MOBILE	2017-01-08 13:17:43.0	2017-01-08 13:15:32.0	2017-01-08 13:17:27.0	114	0.11	9.06426	19.5	
22	WALKING	TRACK_MOBILE	2017-01-08 13:19:59.0	2017-01-08 13:18:48.0	2017-01-08 13:35:02.0	972	1.01	80.36870	4.0	
23	WALKING	TRACK_MOBILE	2017-01-09 09:18:14.0	2017-01-09 08:40:55.0	2017-01-09 09:17:56.0	2220	2.32	184.12300	145.0	
24	SKIING_CROSS_COUNTRY	TRACK_MOBILE	2017-01-09 12:33:28.0	2017-01-09 12:31:17.0	2017-01-09 13:16:22.0	2704	7.14	742.86300	120.5	

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitu
25	WALKING	TRACK_MOBILE	2017-01-10 05:53:36.0	2017-01-10 05:40:40.0	2017-01-10 05:53:27.0	766	1.20	99.61180	135.0	
26	WALKING	TRACK_MOBILE	2017-01-10 16:32:08.0	2017-01-10 16:31:05.0	2017-01-10 17:11:31.0	2425	3.12	226.17900	132.0	
27	WALKING	TRACK_MOBILE	2017-01-11 05:10:37.0	2017-01-11 05:09:27.0	2017-01-11 05:50:13.0	2445	1.48	152.48600	0.0	
28	WALKING	TRACK_MOBILE	2017-01-11 17:10:09.0	2017-01-11 17:09:06.0	2017-01-11 17:47:56.0	2329	5.51	555.89700	134.0	
29	BADMINTON	INPUT_MANUAL_MOBILE	2017-01-11 20:01:18.0	2017-01-11 18:00:00.0	2017-01-11 19:00:00.0	3600	0.00	540.83300	NaN	

Exploratory Data Analysis

Here we will do see some statistical figures about the dataset and visualize them.

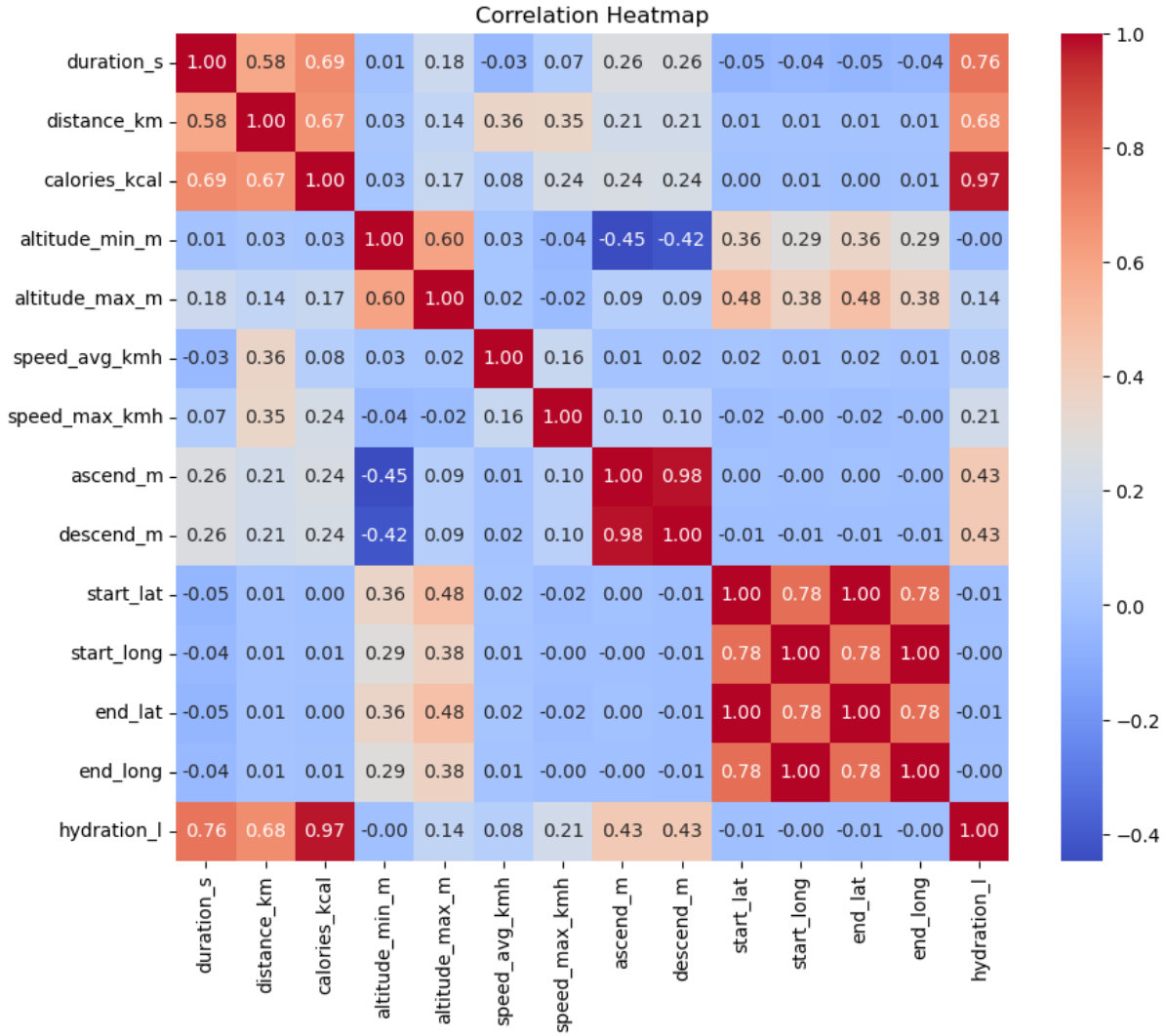
In [10]:
```
# Summary statistics
print("Summary Statistics:")
print(df_res.describe())

# Data types and missing values
print("\nData Types and Missing Values:")
print(df_res.info())
```

```
Summary Statistics:
         duration_s  distance_km  calories_kcal  altitude_min_m  \
count   3503.000000  3503.000000    3502.000000     3087.000000
mean    2254.239224     2.996376     273.285830      137.269455
std     1203.353560     2.437568     180.514293       40.889063
min        7.000000     0.000000       1.000000     -500.500000
25%     1500.000000     1.675341     158.000000      132.919000
50%     2032.000000     2.492906     220.000000      145.475000
75%     2720.500000     3.689069     336.000000      161.573000
max    16742.000000    35.498642    1540.000000      283.500000

       altitude_max_m  speed_avg_kmh  speed_max_kmh     ascend_m    descend_m  \
count     3087.000000    3503.000000    3082.000000  3084.000000  3084.000000
mean       197.357688       4.880691      13.892394    95.870472    98.773677
std         34.112645       5.114440      27.783113   132.687317   132.640471
min          0.000000       0.000000       3.600000     0.000000     0.000000
25%        181.379000       4.110670       8.355880    47.862500    52.486000
50%        205.500000       4.800000       9.846460    74.487900    75.705000
75%        220.113500       5.274849      12.309425   112.593750   114.414250
max        444.000000     267.428562     682.219000  4415.000000  4294.500000

         start_lat   start_long      end_lat     end_long  hydration_l
count  3081.000000  3081.000000  3081.000000  3081.000000  2575.000000
mean     63.906364    27.466786    63.906364    27.466786     0.154651
std       1.975902     2.163956     1.975902     2.163956     0.090070
min      40.761649    -0.375375    40.761649    -0.375375     0.000203
25%      64.231720    27.729424    64.231720    27.729424     0.093121
50%      64.231802    27.729543    64.231802    27.729543     0.132844
75%      64.231874    27.729683    64.231874    27.729683     0.188997
max      68.909689    29.648962    68.909689    29.648962     0.769048

Data Types and Missing Values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3503 entries, 0 to 3502
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sport           3503 non-null   object
 1   source          3503 non-null   object
 2   created_date    3503 non-null   object
 3   start_time      3503 non-null   object
 4   end_time        3503 non-null   object
 5   duration_s      3503 non-null   int64
 6   distance_km     3503 non-null   float64
 7   calories_kcal   3502 non-null   float64
 8   altitude_min_m  3087 non-null   float64
 9   altitude_max_m  3087 non-null   float64
 10  speed_avg_kmh   3503 non-null   float64
 11  speed_max_kmh   3082 non-null   float64
 12  ascend_m        3084 non-null   float64
 13  descend_m       3084 non-null   float64
 14  start_lat       3081 non-null   float64
 15  start_long      3081 non-null   float64
 16  end_lat         3081 non-null   float64
 17  end_long        3081 non-null   float64
 18  hydration_l     2575 non-null   float64
dtypes: float64(13), int64(1), object(5)
memory usage: 520.1+ KB
None
```

We can see that some columns have null values and some data types are inappropriate. We will resolve these issues. Now we will see the correlations among the variables.

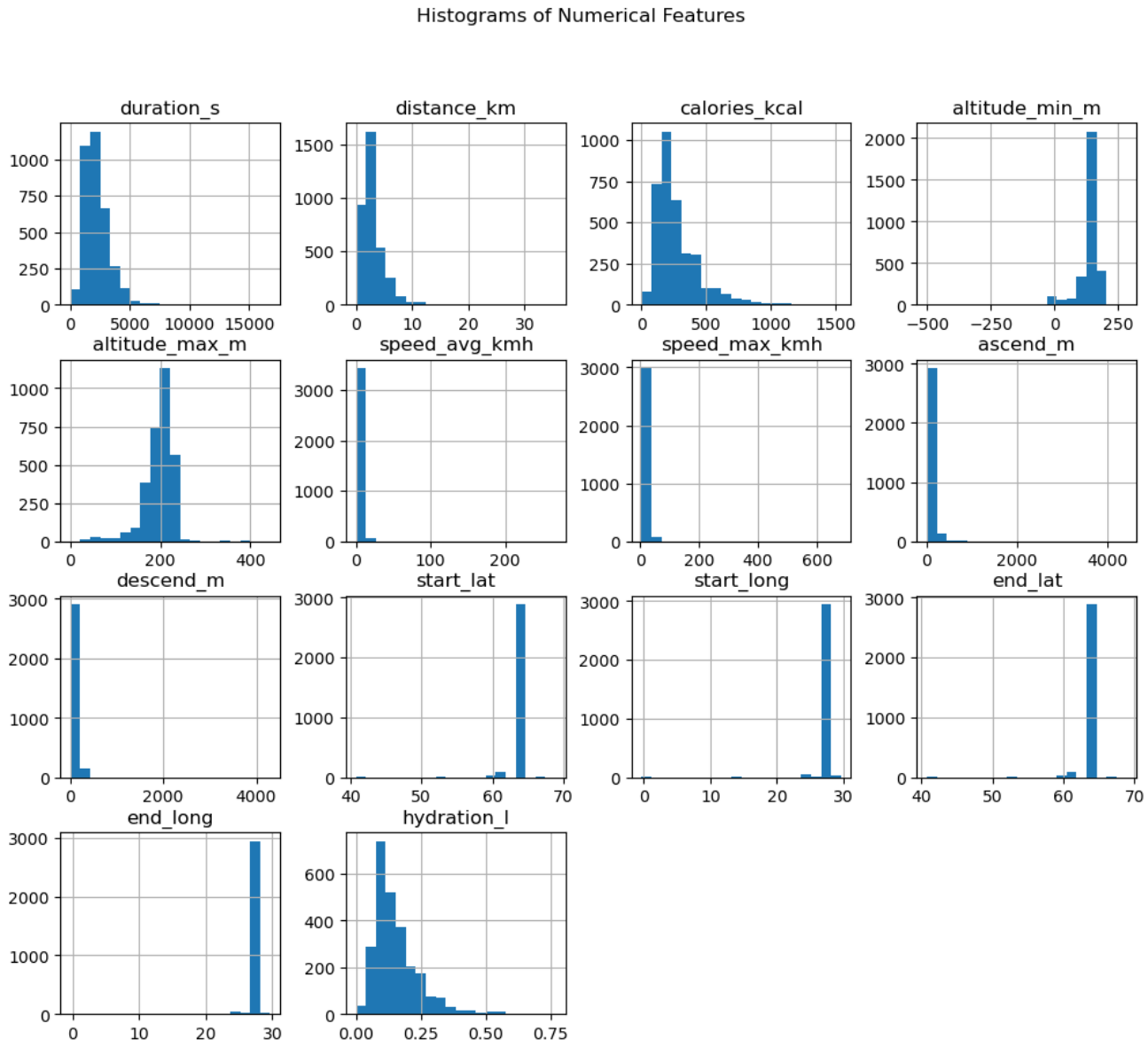
In [11]:
```
# Pairwise correlation matrix and heatmap
correlation_matrix = df_res.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

```
C:\Users\FARSIM\AppData\Local\Temp\ipykernel_12216\3855139582.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  correlation_matrix = df_res.corr()
```

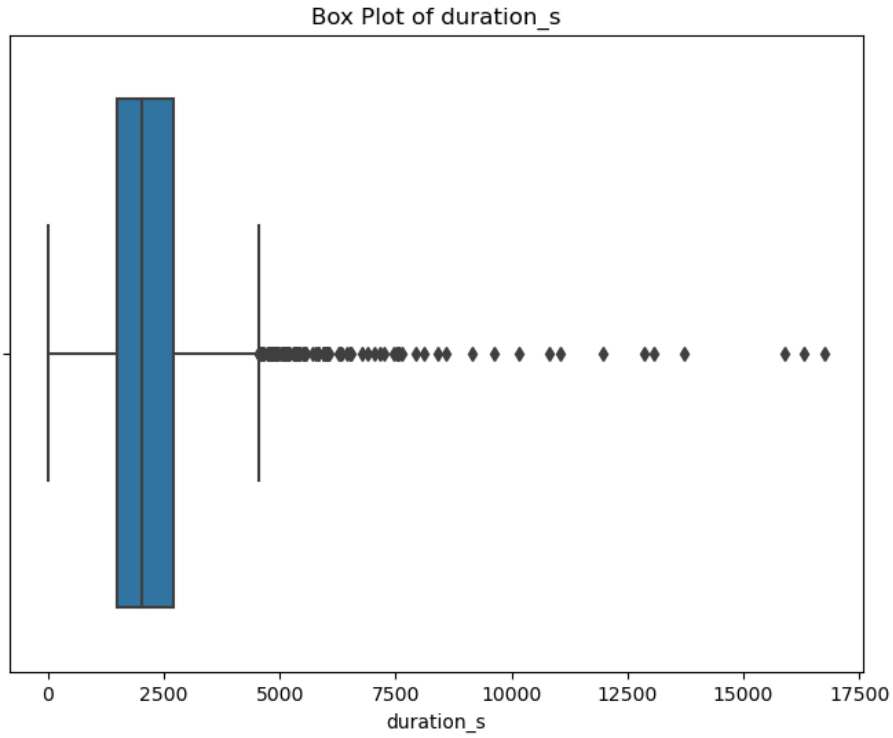


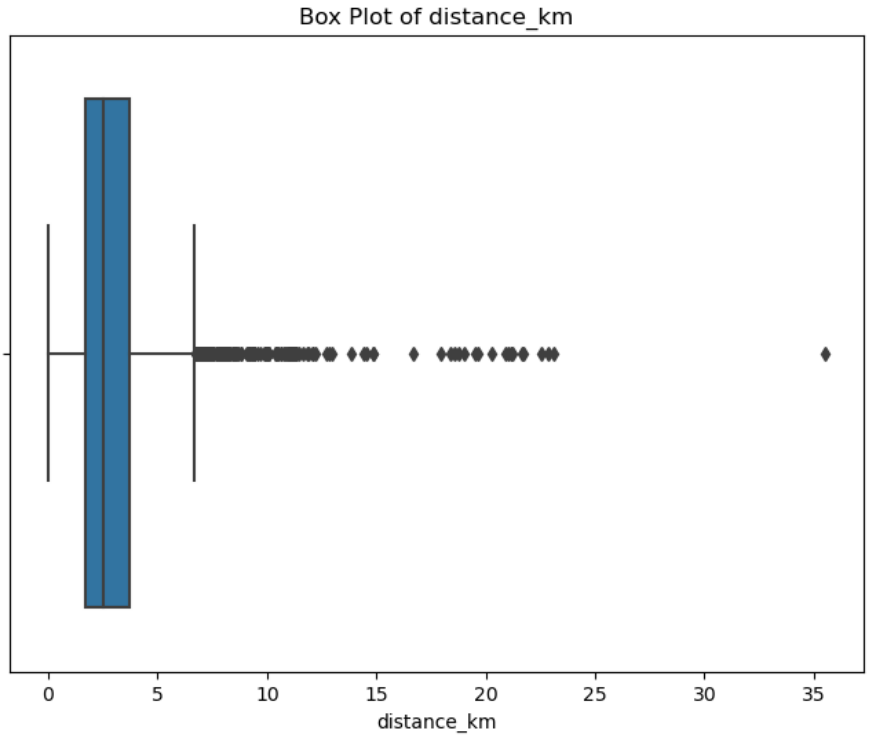
The darker colors denote high correlations between each other. Here some paired correlation can be the cause of **data leakage**. So we will remove them before modelling. Now we will see the distribution of the numerical data.

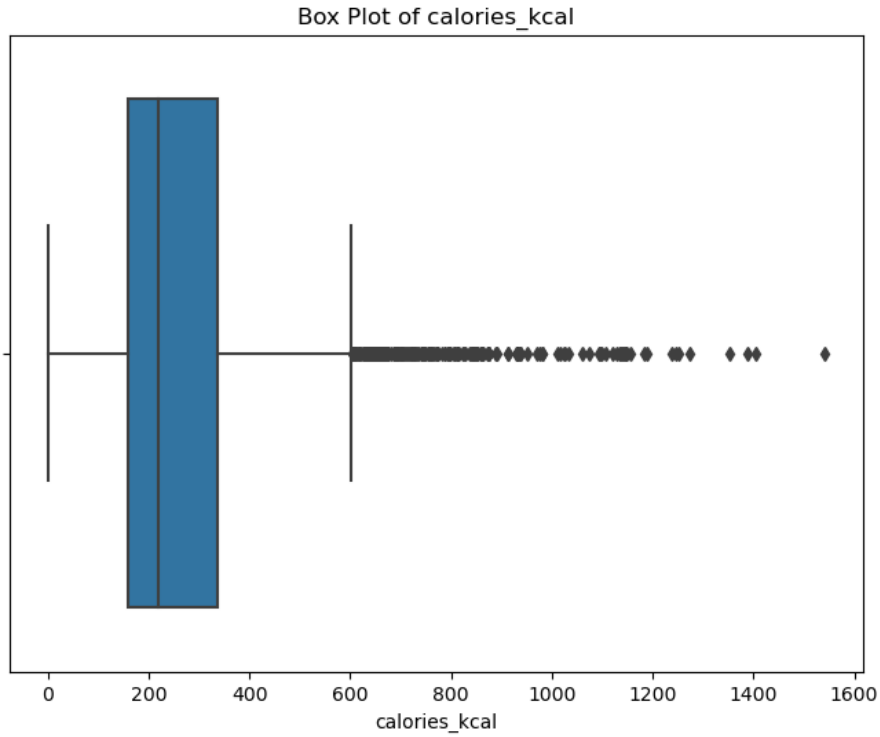
In [12]:
```
# Histograms of numerical features
numerical_features = df_res.select_dtypes(include=['int64', 'float64'])
numerical_features.hist(bins=20, figsize=(12, 10))
plt.suptitle("Histograms of Numerical Features")
plt.show()
```



In [13]:
```
# Box plots for selected features
selected_features = ['duration_s', 'distance_km', 'calories_kcal']
for feature in selected_features:
    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df_res, x=feature)
    plt.title(f"Box Plot of {feature}")
    plt.show()
```





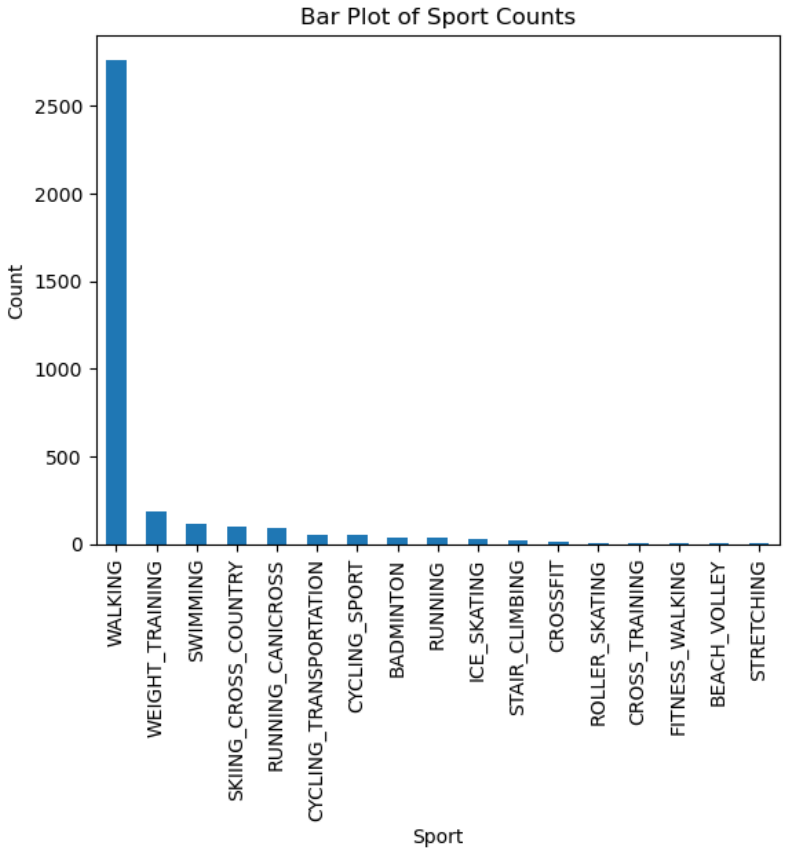


The box plots showing the distributions and outliers which we will work with. Now we will explore the types of activities performed by the user and the frequencies of those activities

In [14]:
```
df_res['sport'].value_counts()
```

Out[14]:
```
WALKING                  2763
WEIGHT_TRAINING           189
SWIMMING                  118
SKIING_CROSS_COUNTRY       95
RUNNING_CANICROSS          92
CYCLING_TRANSPORTATION     54
CYCLING_SPORT              51
BADMINTON                  38
RUNNING                    35
ICE_SKATING                27
STAIR_CLIMBING             17
CROSSFIT                    9
ROLLER_SKATING              6
CROSS_TRAINING              6
FITNESS_WALKING             1
BEACH_VOLLEY                1
STRETCHING                  1
Name: sport, dtype: int64
```

In [15]:
```
df_res['sport'].value_counts().plot(kind='bar')
plt.title('Bar Plot of Sport Counts')
plt.xlabel('Sport')
plt.ylabel('Count')
plt.show()
```



Walking is the most performed activity followed by weight training and swiimming.

Data Cleaning

We need to clean the data before fitting it into a machine learning model. We will convert the datetime columns to appropriate data types.

In [16]:
```
# Convert 'created_date' to datetime
df_res['created_date'] = pd.to_datetime(df_res['created_date'])

# Convert 'start_time' to datetime
df_res['start_time'] = pd.to_datetime(df_res['start_time'])

# Convert 'end_time' to datetime
df_res['end_time'] = pd.to_datetime(df_res['end_time'])
```

In [17]:
```
df = df_res
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3503 entries, 0 to 3502
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sport           3503 non-null   object
 1   source          3503 non-null   object
 2   created_date    3503 non-null   datetime64[ns]
 3   start_time      3503 non-null   datetime64[ns]
 4   end_time        3503 non-null   datetime64[ns]
 5   duration_s      3503 non-null   int64
 6   distance_km     3503 non-null   float64
 7   calories_kcal   3502 non-null   float64
 8   altitude_min_m  3087 non-null   float64
 9   altitude_max_m  3087 non-null   float64
 10  speed_avg_kmh   3503 non-null   float64
 11  speed_max_kmh   3082 non-null   float64
 12  ascend_m        3084 non-null   float64
 13  descend_m       3084 non-null   float64
 14  start_lat       3081 non-null   float64
 15  start_long      3081 non-null   float64
 16  end_lat         3081 non-null   float64
 17  end_long        3081 non-null   float64
 18  hydration_l     2575 non-null   float64
dtypes: datetime64[ns](3), float64(13), int64(1), object(2)
memory usage: 520.1+ KB
```

In [18]:
```
df.head()
```

Out[18]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitude_max_n
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23	2017-01-01 08:53:04	2017-01-01 09:27:49	2084	2.15	171.651	145.5	198.(
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04	2017-01-01 14:01:00	2017-01-01 14:41:00	2400	0.00	393.333	NaN	NaN
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03	2017-01-01 17:46:00	2017-01-01 18:12:07	1566	1.69	132.168	126.5	174.5
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23	2017-01-02 08:55:52	2017-01-02 09:26:06	1812	2.07	157.828	81.0	201.(
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51	2017-01-02 16:13:34	2017-01-02 16:54:52	2444	5.87	591.404	97.5	159.5

In [19]:
```
# Outlier removal

# Select numeric columns only (excluding non-numeric columns)
numeric_columns = df.select_dtypes(include=['int64', 'float64'])

# Calculate the IQR for each numeric column
Q1 = numeric_columns.quantile(0.25)
Q3 = numeric_columns.quantile(0.75)
IQR = Q3 - Q1

# Define a lower bound and upper bound for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Detect and remove outliers
outliers_removed_df = df[~((numeric_columns < lower_bound) | (numeric_columns > upper_bound)).any(axis=1)]
```

In [20]:
```
outliers_removed_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2054 entries, 0 to 3501
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sport           2054 non-null   object
 1   source          2054 non-null   object
 2   created_date    2054 non-null   datetime64[ns]
 3   start_time      2054 non-null   datetime64[ns]
 4   end_time        2054 non-null   datetime64[ns]
 5   duration_s      2054 non-null   int64
 6   distance_km     2054 non-null   float64
 7   calories_kcal   2054 non-null   float64
 8   altitude_min_m  1947 non-null   float64
 9   altitude_max_m  1947 non-null   float64
 10  speed_avg_kmh   2054 non-null   float64
 11  speed_max_kmh   1945 non-null   float64
 12  ascend_m        1944 non-null   float64
 13  descend_m       1944 non-null   float64
 14  start_lat       1943 non-null   float64
 15  start_long      1943 non-null   float64
 16  end_lat         1943 non-null   float64
 17  end_long        1943 non-null   float64
 18  hydration_l     1546 non-null   float64
dtypes: datetime64[ns](3), float64(13), int64(1), object(2)
memory usage: 320.9+ KB
```

It looks like a lot of rows have been dropped when we removed outliers. Taking that into consideration, I changed my mind to work with the original dataframe

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3503 entries, 0 to 3502
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sport           3503 non-null   object
 1   source          3503 non-null   object
 2   created_date    3503 non-null   datetime64[ns]
 3   start_time      3503 non-null   datetime64[ns]
 4   end_time        3503 non-null   datetime64[ns]
 5   duration_s      3503 non-null   int64
 6   distance_km     3503 non-null   float64
 7   calories_kcal   3502 non-null   float64
 8   altitude_min_m  3087 non-null   float64
 9   altitude_max_m  3087 non-null   float64
 10  speed_avg_kmh   3503 non-null   float64
 11  speed_max_kmh   3082 non-null   float64
 12  ascend_m        3084 non-null   float64
 13  descend_m       3084 non-null   float64
 14  start_lat       3081 non-null   float64
 15  start_long      3081 non-null   float64
 16  end_lat         3081 non-null   float64
 17  end_long        3081 non-null   float64
 18  hydration_l     2575 non-null   float64
dtypes: datetime64[ns](3), float64(13), int64(1), object(2)
memory usage: 520.1+ KB
```

Now we will finn the missing values.

In [22]:
```
df.fillna(df.mean(), inplace=True)  # Replace missing values with the mean
```

```
C:\Users\FARSIM\AppData\Local\Temp\ipykernel_12216\4158336622.py:1: FutureWarning: DataFrame.mean and DataFrame.median with
numeric_only=None will include datetime64 and datetime64tz columns in a future version.
  df.fillna(df.mean(), inplace=True)  # Replace missing values with the mean
C:\Users\FARSIM\AppData\Local\Temp\ipykernel_12216\4158336622.py:1: FutureWarning: The default value of numeric_only in Dat
aFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is de
precated. Select only valid columns or specify the value of numeric_only to silence this warning.
  df.fillna(df.mean(), inplace=True)  # Replace missing values with the mean
```

In [23]:
```
# null count
df.isnull().sum()
```

Out[23]:
```
sport             0
source            0
created_date      0
start_time        0
end_time          0
duration_s        0
distance_km       0
calories_kcal     0
altitude_min_m    0
altitude_max_m    0
speed_avg_kmh     0
speed_max_kmh     0
ascend_m          0
descend_m         0
start_lat         0
start_long        0
end_lat           0
end_long          0
hydration_l       0
dtype: int64
```

Feature Engineering

Feature Engineering is very important for a machine learning project. It helps the model to understand the data better so that it can be well trained.

In [24]:
```
df.head()
```

Out[24]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitude_max_n
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23	2017-01-01 08:53:04	2017-01-01 09:27:49	2084	2.15	171.651	145.500000	198.00000
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04	2017-01-01 14:01:00	2017-01-01 14:41:00	2400	0.00	393.333	137.269455	197.35768
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03	2017-01-01 17:46:00	2017-01-01 18:12:07	1566	1.69	132.168	126.500000	174.50000
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23	2017-01-02 08:55:52	2017-01-02 09:26:06	1812	2.07	157.828	81.000000	201.00000
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51	2017-01-02 16:13:34	2017-01-02 16:54:52	2444	5.87	591.404	97.500000	159.50000

We are starting to create new features. We are creating the target variables along with new features

In [25]:
```
# Sort the DataFrame by some relevant columns (e.g., 'created_date' or another column with a logical order)
# This step is important to ensure that the rows are in the desired order
df1 = df.sort_values(by=['created_date'])

# Create 'next_exercise' and 'previous_exercise' columns
df1['next_exercise'] = df1['sport'].shift(-1)
df1['previous_exercise'] = df1['sport'].shift(1)

# Set NaN for the 'next_exercise' of the last row and 'previous_exercise' of the first row
df1.at[df1.index[-1], 'next_exercise'] = None
df1.at[df1.index[0], 'previous_exercise'] = None
```

In [26]:
```
df1.tail(50)
```

)	216.570000	...	7.715190	66.739000	67.875000	64.231750	27.729426	64.231750	27.729426	0.125206	SWIMMING	WAL
;	197.357688	...	13.892394	95.870472	98.773677	63.906364	27.466786	63.906364	27.466786	0.170183	WALKING	WAL
)	220.192000	...	7.802410	54.256000	65.363000	64.232438	27.731192	64.232438	27.731192	0.098057	SWIMMING	SWIM
;	197.357688	...	13.892394	95.870472	98.773677	63.906364	27.466786	63.906364	27.466786	0.170183	RUNNING_CANICROSS	WAL
)	173.130000	...	13.936900	51.102000	53.822000	64.231690	27.729530	64.231690	27.729530	0.232155	WALKING	SWIM
)	185.683000	...	8.812570	55.258000	65.117000	64.231778	27.730185	64.231778	27.730185	0.089959	WALKING	RUNNING_CANICF

In [27]:
```
# Create 'next_time' and 'previous_time' columns
df1['next_time'] = df1['start_time'].shift(-1)
df1['previous_time'] = df1['start_time'].shift(1)

# Create 'next_duration' and 'previous_duration' columns
df1['next_duration'] = df1['duration_s'].shift(-1)
df1['previous_duration'] = df1['duration_s'].shift(1)

# Set NaN for the 'next_time' and 'next_duration' of the last row and 'previous_time' and 'previous_duration' of the first ro
df1.at[df.index[-1], 'next_time'] = None
df1.at[df.index[-1], 'next_duration'] = None
df1.at[df.index[0], 'previous_time'] = None
df1.at[df.index[0], 'previous_duration'] = None
```

In [28]:
```
df1.head()
```

Out[28]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitude_max_n
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23	2017-01-01 08:53:04	2017-01-01 09:27:49	2084	2.15	171.651	145.500000	198.00000(
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04	2017-01-01 14:01:00	2017-01-01 14:41:00	2400	0.00	393.333	137.269455	197.35768(
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03	2017-01-01 17:46:00	2017-01-01 18:12:07	1566	1.69	132.168	126.500000	174.50000(
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23	2017-01-02 08:55:52	2017-01-02 09:26:06	1812	2.07	157.828	81.000000	201.00000(
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51	2017-01-02 16:13:34	2017-01-02 16:54:52	2444	5.87	591.404	97.500000	159.50000(

5 rows × 25 columns

In [29]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3503 entries, 0 to 3502
Data columns (total 25 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sport              3503 non-null   object
 1   source             3503 non-null   object
 2   created_date       3503 non-null   datetime64[ns]
 3   start_time         3503 non-null   datetime64[ns]
 4   end_time           3503 non-null   datetime64[ns]
 5   duration_s         3503 non-null   int64
 6   distance_km        3503 non-null   float64
 7   calories_kcal      3503 non-null   float64
 8   altitude_min_m     3503 non-null   float64
 9   altitude_max_m     3503 non-null   float64
 10  speed_avg_kmh      3503 non-null   float64
 11  speed_max_kmh      3503 non-null   float64
 12  ascend_m           3503 non-null   float64
 13  descend_m          3503 non-null   float64
 14  start_lat          3503 non-null   float64
 15  start_long         3503 non-null   float64
 16  end_lat            3503 non-null   float64
 17  end_long           3503 non-null   float64
 18  hydration_l        3503 non-null   float64
 19  next_exercise      3502 non-null   object
 20  previous_exercise  3502 non-null   object
 21  next_time          3502 non-null   datetime64[ns]
 22  previous_time      3502 non-null   datetime64[ns]
 23  next_duration      3502 non-null   float64
 24  previous_duration  3502 non-null   float64
dtypes: datetime64[ns](5), float64(15), int64(1), object(4)
memory usage: 840.6+ KB
```

In [30]:
```
# Fill NaN values in 'next_duration' and 'previous_duration' with the mean of their respective columns
mean_next_duration = df1['next_duration'].mean()
mean_previous_duration = df1['previous_duration'].mean()

df1['next_duration'].fillna(mean_next_duration, inplace=True)
df1['previous_duration'].fillna(mean_previous_duration, inplace=True)

# Fill NaN values in 'next_exercise' and 'previous_exercise' with the mode of their respective columns
mode_next_exercise = df1['next_exercise'].mode().iloc[0]
mode_previous_exercise = df1['previous_exercise'].mode().iloc[0]

df1['next_exercise'].fillna(mode_next_exercise, inplace=True)
df1['previous_exercise'].fillna(mode_previous_exercise, inplace=True)
```

In [31]:
```
pd.set_option('display.max_rows', 100)
df1.head()
```

Out[31]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitude_ma
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23	2017-01-01 08:53:04	2017-01-01 09:27:49	2084	2.15	171.651	145.500000	198.000
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04	2017-01-01 14:01:00	2017-01-01 14:41:00	2400	0.00	393.333	137.269455	197.357
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03	2017-01-01 17:46:00	2017-01-01 18:12:07	1566	1.69	132.168	126.500000	174.500
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23	2017-01-02 08:55:52	2017-01-02 09:26:06	1812	2.07	157.828	81.000000	201.000
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51	2017-01-02 16:13:34	2017-01-02 16:54:52	2444	5.87	591.404	97.500000	159.500

Seperating all date and time to month, day and hour with one hot encoding

In [34]:
```
# Define a list of date and time columns to perform feature extraction on
date_time_columns = ['created_date', 'start_time', 'end_time', 'next_time', 'previous_time']

# Perform feature extraction for each date and time column
for col in date_time_columns:
    df1[col + '_year'] = df1[col].dt.year
    df1[col + '_month'] = df1[col].dt.month
    df1[col + '_day'] = df1[col].dt.day
    df1[col + '_day_of_week'] = df1[col].dt.dayofweek
    df1[col + '_hour'] = df1[col].dt.hour

    # You can add more date and time features as needed

# One-hot encoding for day of the week in all columns
for col in date_time_columns:
    df1 = pd.get_dummies(df1, columns=[col + '_day_of_week'], prefix=col + '_day')
```

In [35]:
```
df1.head()
```

Out[35]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitude_max_n
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23	2017-01-01 08:53:04	2017-01-01 09:27:49	2084	2.15	171.651	145.500000	198.00000
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04	2017-01-01 14:01:00	2017-01-01 14:41:00	2400	0.00	393.333	137.269455	197.35768
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03	2017-01-01 17:46:00	2017-01-01 18:12:07	1566	1.69	132.168	126.500000	174.50000
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23	2017-01-02 08:55:52	2017-01-02 09:26:06	1812	2.07	157.828	81.000000	201.00000
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51	2017-01-02 16:13:34	2017-01-02 16:54:52	2444	5.87	591.404	97.500000	159.50000

5 rows × 80 columns

In [38]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3503 entries, 0 to 3502
Data columns (total 80 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   sport                  3503 non-null   object
 1   source                 3503 non-null   object
 2   created_date           3503 non-null   datetime64[ns]
 3   start_time             3503 non-null   datetime64[ns]
 4   end_time               3503 non-null   datetime64[ns]
 5   duration_s             3503 non-null   int64
 6   distance_km            3503 non-null   float64
 7   calories_kcal          3503 non-null   float64
 8   altitude_min_m         3503 non-null   float64
 9   altitude_max_m         3503 non-null   float64
 10  speed_avg_kmh          3503 non-null   float64
 11  speed_max_kmh          3503 non-null   float64
 12  ascend_m               3503 non-null   float64
 13  descend_m              3503 non-null   float64
 14  start_lat              3503 non-null   float64
 15  start_long             3503 non-null   float64
 16  end_lat                3503 non-null   float64
 17  end_long               3503 non-null   float64
 18  hydration_l            3503 non-null   float64
 19  next_exercise          3503 non-null   object
 20  previous_exercise      3503 non-null   object
 21  next_time              3502 non-null   datetime64[ns]
 22  previous_time          3502 non-null   datetime64[ns]
 23  next_duration          3503 non-null   float64
 24  previous_duration      3503 non-null   float64
 25  created_date_year      3503 non-null   int64
 26  created_date_month     3503 non-null   int64
 27  created_date_day       3503 non-null   int64
 28  created_date_hour      3503 non-null   int64
 29  start_time_year        3503 non-null   int64
 30  start_time_month       3503 non-null   int64
 31  start_time_day         3503 non-null   int64
 32  start_time_hour        3503 non-null   int64
 33  end_time_year          3503 non-null   int64
 34  end_time_month         3503 non-null   int64
 35  end_time_day           3503 non-null   int64
 36  end_time_hour          3503 non-null   int64
 37  next_time_year         3502 non-null   float64
 38  next_time_month        3502 non-null   float64
 39  next_time_day          3502 non-null   float64
 40  next_time_hour         3502 non-null   float64
 41  previous_time_year     3502 non-null   float64
 42  previous_time_month    3502 non-null   float64
 43  previous_time_day      3502 non-null   float64
 44  previous_time_hour     3502 non-null   float64
 45  created_date_day_0     3503 non-null   uint8
 46  created_date_day_1     3503 non-null   uint8
 47  created_date_day_2     3503 non-null   uint8
 48  created_date_day_3     3503 non-null   uint8
 49  created_date_day_4     3503 non-null   uint8
 50  created_date_day_5     3503 non-null   uint8
 51  created_date_day_6     3503 non-null   uint8
 52  start_time_day_0       3503 non-null   uint8
 53  start_time_day_1       3503 non-null   uint8
 54  start_time_day_2       3503 non-null   uint8
 55  start_time_day_3       3503 non-null   uint8
 56  start_time_day_4       3503 non-null   uint8
 57  start_time_day_5       3503 non-null   uint8
 58  start_time_day_6       3503 non-null   uint8
 59  end_time_day_0         3503 non-null   uint8
 60  end_time_day_1         3503 non-null   uint8
 61  end_time_day_2         3503 non-null   uint8
 62  end_time_day_3         3503 non-null   uint8
 63  end_time_day_4         3503 non-null   uint8
 64  end_time_day_5         3503 non-null   uint8
 65  end_time_day_6         3503 non-null   uint8
 66  next_time_day_0.0      3503 non-null   uint8
 67  next_time_day_1.0      3503 non-null   uint8
 68  next_time_day_2.0      3503 non-null   uint8
 69  next_time_day_3.0      3503 non-null   uint8
 70  next_time_day_4.0      3503 non-null   uint8
 71  next_time_day_5.0      3503 non-null   uint8
 72  next_time_day_6.0      3503 non-null   uint8
 73  previous_time_day_0.0  3503 non-null   uint8
 74  previous_time_day_1.0  3503 non-null   uint8
 75  previous_time_day_2.0  3503 non-null   uint8
 76  previous_time_day_3.0  3503 non-null   uint8
 77  previous_time_day_4.0  3503 non-null   uint8
```

```
 78  previous_time_day_5.0  3503 non-null   uint8
 79  previous_time_day_6.0  3503 non-null   uint8
dtypes: datetime64[ns](5), float64(23), int64(13), object(4), uint8(35)
memory usage: 1.5+ MB
```

In [39]:
```
df1.isnull().sum()
```

Out[39]:
```
sport                   0
source                  0
created_date            0
start_time              0
end_time                0
duration_s              0
distance_km             0
calories_kcal           0
altitude_min_m          0
altitude_max_m          0
speed_avg_kmh           0
speed_max_kmh           0
ascend_m                0
descend_m               0
start_lat               0
start_long              0
end_lat                 0
end_long                0
hydration_l             0
next_exercise           0
previous_exercise       0
next_time               1
previous_time           1
next_duration           0
previous_duration       0
created_date_year       0
created_date_month      0
created_date_day        0
created_date_hour       0
start_time_year         0
start_time_month        0
start_time_day          0
start_time_hour         0
end_time_year           0
end_time_month          0
end_time_day            0
end_time_hour           0
next_time_year          1
next_time_month         1
next_time_day           1
next_time_hour          1
previous_time_year      1
previous_time_month     1
previous_time_day       1
previous_time_hour      1
created_date_day_0      0
created_date_day_1      0
created_date_day_2      0
created_date_day_3      0
created_date_day_4      0
created_date_day_5      0
created_date_day_6      0
start_time_day_0        0
start_time_day_1        0
start_time_day_2        0
start_time_day_3        0
start_time_day_4        0
start_time_day_5        0
start_time_day_6        0
end_time_day_0          0
end_time_day_1          0
end_time_day_2          0
end_time_day_3          0
end_time_day_4          0
end_time_day_5          0
end_time_day_6          0
next_time_day_0.0       0
next_time_day_1.0       0
next_time_day_2.0       0
next_time_day_3.0       0
next_time_day_4.0       0
next_time_day_5.0       0
next_time_day_6.0       0
previous_time_day_0.0   0
previous_time_day_1.0   0
previous_time_day_2.0   0
previous_time_day_3.0   0
previous_time_day_4.0   0
previous_time_day_5.0   0
previous_time_day_6.0   0
dtype: int64
```

We have new features now to help the model understand the data better. We will make sure that we dont have any null values.

In [40]:
```
# removing the null rows
df1 = df1.dropna()
```

In [41]:
```
df1.head()
```

Out[41]:

	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min_m	altitude_max_n
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04	2017-01-01 14:01:00	2017-01-01 14:41:00	2400	0.00	393.333	137.269455	197.35768
2	WALKING	TRACK_MOBILE	2017-01-01 17:47:03	2017-01-01 17:46:00	2017-01-01 18:12:07	1566	1.69	132.168	126.500000	174.50000
3	WALKING	TRACK_MOBILE	2017-01-02 08:57:23	2017-01-02 08:55:52	2017-01-02 09:26:06	1812	2.07	157.828	81.000000	201.00000
4	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51	2017-01-02 16:13:34	2017-01-02 16:54:52	2444	5.87	591.404	97.500000	159.50000
5	WALKING	TRACK_MOBILE	2017-01-03 09:25:22	2017-01-03 09:19:16	2017-01-03 09:53:05	1963	2.15	167.024	138.000000	198.00000

5 rows × 80 columns

Now that we dont need the real datetime columns (because we have extracted the date time information into seperate columns), we will remove them.

In [42]:
```
# Define a list of columns to drop, including the original datetime columns
columns_to_drop = ['created_date', 'start_time', 'end_time', 'next_time', 'previous_time']

# Drop the datetime columns from the DataFrame
df1 = df1.drop(columns=columns_to_drop)
```

In [44]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3501 entries, 1 to 3501
Data columns (total 75 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   sport                  3501 non-null   object
 1   source                 3501 non-null   object
 2   duration_s             3501 non-null   int64
 3   distance_km            3501 non-null   float64
 4   calories_kcal          3501 non-null   float64
 5   altitude_min_m         3501 non-null   float64
 6   altitude_max_m         3501 non-null   float64
 7   speed_avg_kmh          3501 non-null   float64
 8   speed_max_kmh          3501 non-null   float64
 9   ascend_m               3501 non-null   float64
 10  descend_m              3501 non-null   float64
 11  start_lat              3501 non-null   float64
 12  start_long             3501 non-null   float64
 13  end_lat                3501 non-null   float64
 14  end_long               3501 non-null   float64
 15  hydration_l            3501 non-null   float64
 16  next_exercise          3501 non-null   object
 17  previous_exercise      3501 non-null   object
 18  next_duration          3501 non-null   float64
 19  previous_duration      3501 non-null   float64
 20  created_date_year      3501 non-null   int64
 21  created_date_month     3501 non-null   int64
 22  created_date_day       3501 non-null   int64
 23  created_date_hour      3501 non-null   int64
 24  start_time_year        3501 non-null   int64
 25  start_time_month       3501 non-null   int64
 26  start_time_day         3501 non-null   int64
 27  start_time_hour        3501 non-null   int64
 28  end_time_year          3501 non-null   int64
 29  end_time_month         3501 non-null   int64
 30  end_time_day           3501 non-null   int64
 31  end_time_hour          3501 non-null   int64
 32  next_time_year         3501 non-null   float64
 33  next_time_month        3501 non-null   float64
 34  next_time_day          3501 non-null   float64
 35  next_time_hour         3501 non-null   float64
 36  previous_time_year     3501 non-null   float64
 37  previous_time_month    3501 non-null   float64
 38  previous_time_day      3501 non-null   float64
 39  previous_time_hour     3501 non-null   float64
 40  created_date_day_0     3501 non-null   uint8
 41  created_date_day_1     3501 non-null   uint8
 42  created_date_day_2     3501 non-null   uint8
 43  created_date_day_3     3501 non-null   uint8
 44  created_date_day_4     3501 non-null   uint8
 45  created_date_day_5     3501 non-null   uint8
 46  created_date_day_6     3501 non-null   uint8
 47  start_time_day_0       3501 non-null   uint8
 48  start_time_day_1       3501 non-null   uint8
 49  start_time_day_2       3501 non-null   uint8
 50  start_time_day_3       3501 non-null   uint8
 51  start_time_day_4       3501 non-null   uint8
 52  start_time_day_5       3501 non-null   uint8
 53  start_time_day_6       3501 non-null   uint8
 54  end_time_day_0         3501 non-null   uint8
 55  end_time_day_1         3501 non-null   uint8
 56  end_time_day_2         3501 non-null   uint8
 57  end_time_day_3         3501 non-null   uint8
 58  end_time_day_4         3501 non-null   uint8
 59  end_time_day_5         3501 non-null   uint8
 60  end_time_day_6         3501 non-null   uint8
 61  next_time_day_0.0      3501 non-null   uint8
 62  next_time_day_1.0      3501 non-null   uint8
 63  next_time_day_2.0      3501 non-null   uint8
 64  next_time_day_3.0      3501 non-null   uint8
 65  next_time_day_4.0      3501 non-null   uint8
 66  next_time_day_5.0      3501 non-null   uint8
 67  next_time_day_6.0      3501 non-null   uint8
 68  previous_time_day_0.0  3501 non-null   uint8
 69  previous_time_day_1.0  3501 non-null   uint8
 70  previous_time_day_2.0  3501 non-null   uint8
 71  previous_time_day_3.0  3501 non-null   uint8
 72  previous_time_day_4.0  3501 non-null   uint8
 73  previous_time_day_5.0  3501 non-null   uint8
 74  previous_time_day_6.0  3501 non-null   uint8
dtypes: float64(23), int64(13), object(4), uint8(35)
memory usage: 1.2+ MB
```

We still need to convert some columns to integer.

In [46]:
```
# Define the list of columns to convert to integer
int_columns = [
    'next_time_year', 'next_time_month', 'next_time_day', 'next_time_hour',
    'previous_time_year', 'previous_time_month', 'previous_time_day', 'previous_time_hour'
]

# Convert the specified columns to integer
df1[int_columns] = df1[int_columns].astype(int)
```

In [47]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3501 entries, 1 to 3501
Data columns (total 75 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   sport                  3501 non-null   object
 1   source                 3501 non-null   object
 2   duration_s             3501 non-null   int64
 3   distance_km            3501 non-null   float64
 4   calories_kcal          3501 non-null   float64
 5   altitude_min_m         3501 non-null   float64
 6   altitude_max_m         3501 non-null   float64
 7   speed_avg_kmh          3501 non-null   float64
 8   speed_max_kmh          3501 non-null   float64
 9   ascend_m               3501 non-null   float64
 10  descend_m              3501 non-null   float64
 11  start_lat              3501 non-null   float64
 12  start_long             3501 non-null   float64
 13  end_lat                3501 non-null   float64
 14  end_long               3501 non-null   float64
 15  hydration_l            3501 non-null   float64
 16  next_exercise          3501 non-null   object
 17  previous_exercise      3501 non-null   object
 18  next_duration          3501 non-null   float64
 19  previous_duration      3501 non-null   float64
 20  created_date_year      3501 non-null   int64
 21  created_date_month     3501 non-null   int64
 22  created_date_day       3501 non-null   int64
 23  created_date_hour      3501 non-null   int64
 24  start_time_year        3501 non-null   int64
 25  start_time_month       3501 non-null   int64
 26  start_time_day         3501 non-null   int64
 27  start_time_hour        3501 non-null   int64
 28  end_time_year          3501 non-null   int64
 29  end_time_month         3501 non-null   int64
 30  end_time_day           3501 non-null   int64
 31  end_time_hour          3501 non-null   int64
 32  next_time_year         3501 non-null   int32
 33  next_time_month        3501 non-null   int32
 34  next_time_day          3501 non-null   int32
 35  next_time_hour         3501 non-null   int32
 36  previous_time_year     3501 non-null   int32
 37  previous_time_month    3501 non-null   int32
 38  previous_time_day      3501 non-null   int32
 39  previous_time_hour     3501 non-null   int32
 40  created_date_day_0     3501 non-null   uint8
 41  created_date_day_1     3501 non-null   uint8
 42  created_date_day_2     3501 non-null   uint8
 43  created_date_day_3     3501 non-null   uint8
 44  created_date_day_4     3501 non-null   uint8
 45  created_date_day_5     3501 non-null   uint8
 46  created_date_day_6     3501 non-null   uint8
 47  start_time_day_0       3501 non-null   uint8
 48  start_time_day_1       3501 non-null   uint8
 49  start_time_day_2       3501 non-null   uint8
 50  start_time_day_3       3501 non-null   uint8
 51  start_time_day_4       3501 non-null   uint8
 52  start_time_day_5       3501 non-null   uint8
 53  start_time_day_6       3501 non-null   uint8
 54  end_time_day_0         3501 non-null   uint8
 55  end_time_day_1         3501 non-null   uint8
 56  end_time_day_2         3501 non-null   uint8
 57  end_time_day_3         3501 non-null   uint8
 58  end_time_day_4         3501 non-null   uint8
 59  end_time_day_5         3501 non-null   uint8
 60  end_time_day_6         3501 non-null   uint8
 61  next_time_day_0.0      3501 non-null   uint8
 62  next_time_day_1.0      3501 non-null   uint8
 63  next_time_day_2.0      3501 non-null   uint8
 64  next_time_day_3.0      3501 non-null   uint8
 65  next_time_day_4.0      3501 non-null   uint8
 66  next_time_day_5.0      3501 non-null   uint8
 67  next_time_day_6.0      3501 non-null   uint8
 68  previous_time_day_0.0  3501 non-null   uint8
 69  previous_time_day_1.0  3501 non-null   uint8
 70  previous_time_day_2.0  3501 non-null   uint8
 71  previous_time_day_3.0  3501 non-null   uint8
 72  previous_time_day_4.0  3501 non-null   uint8
 73  previous_time_day_5.0  3501 non-null   uint8
 74  previous_time_day_6.0  3501 non-null   uint8
dtypes: float64(15), int32(8), int64(13), object(4), uint8(35)
memory usage: 1.1+ MB
```

Encoding the categorical columns : We will use One Hot Encoding for this. Encoding is important for models to understand the data.

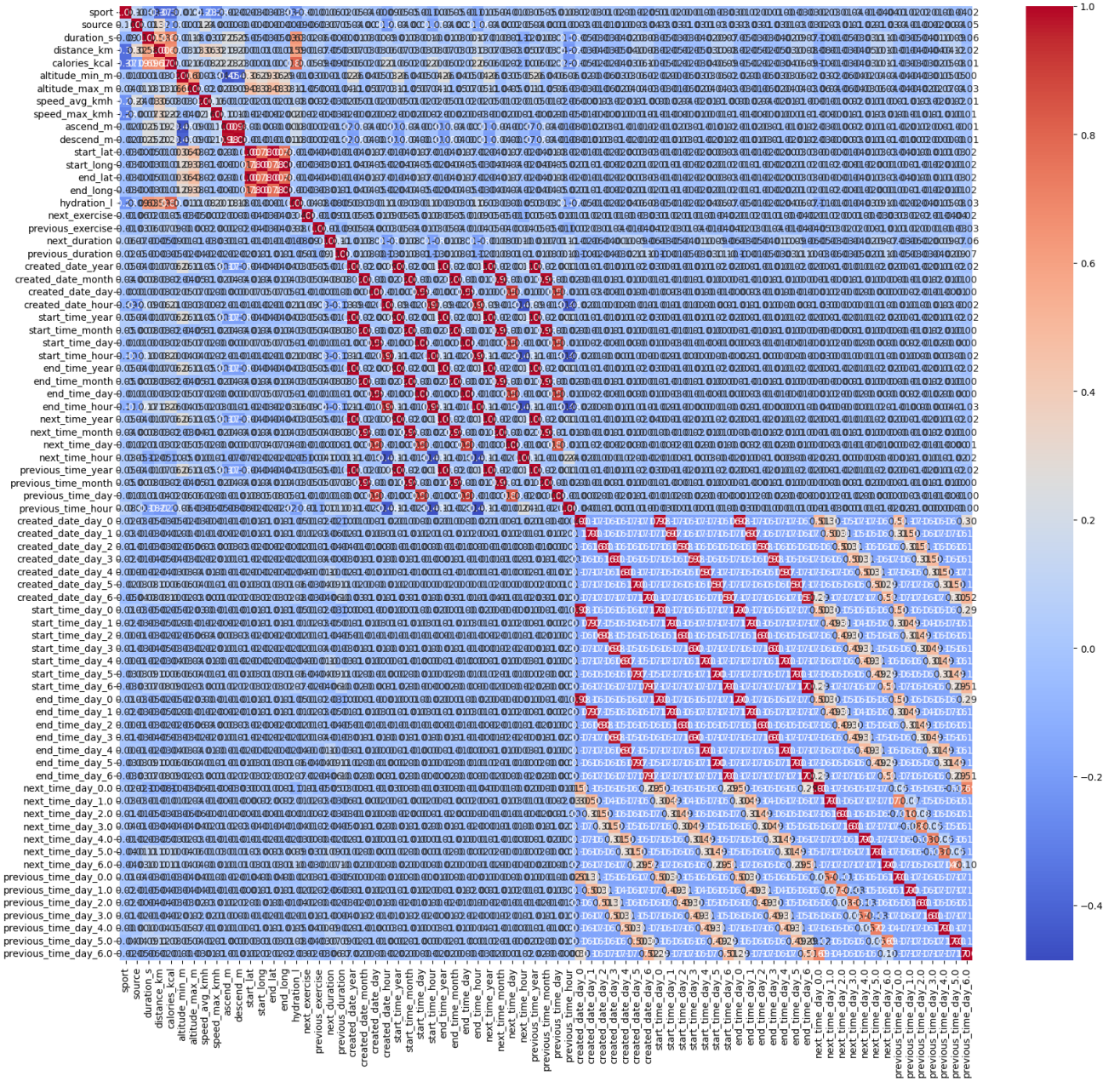
In [51]:
```
# Identify object (categorical) columns
object_columns = df1.select_dtypes(include=['object']).columns

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Encode each object column
for col in object_columns:
    df1[col] = label_encoder.fit_transform(df1[col])
```

Feature Selection

This is the time select only important features for our model. We will remove those features are not important for the model and can introduce data leakage.

In [62]:
```
# Define your target variables
target_variables = ['next_exercise', 'next_time_hour', 'next_time_day', 'next_duration']

# Calculate correlations with the target variables
correlations = df1.corr()
```

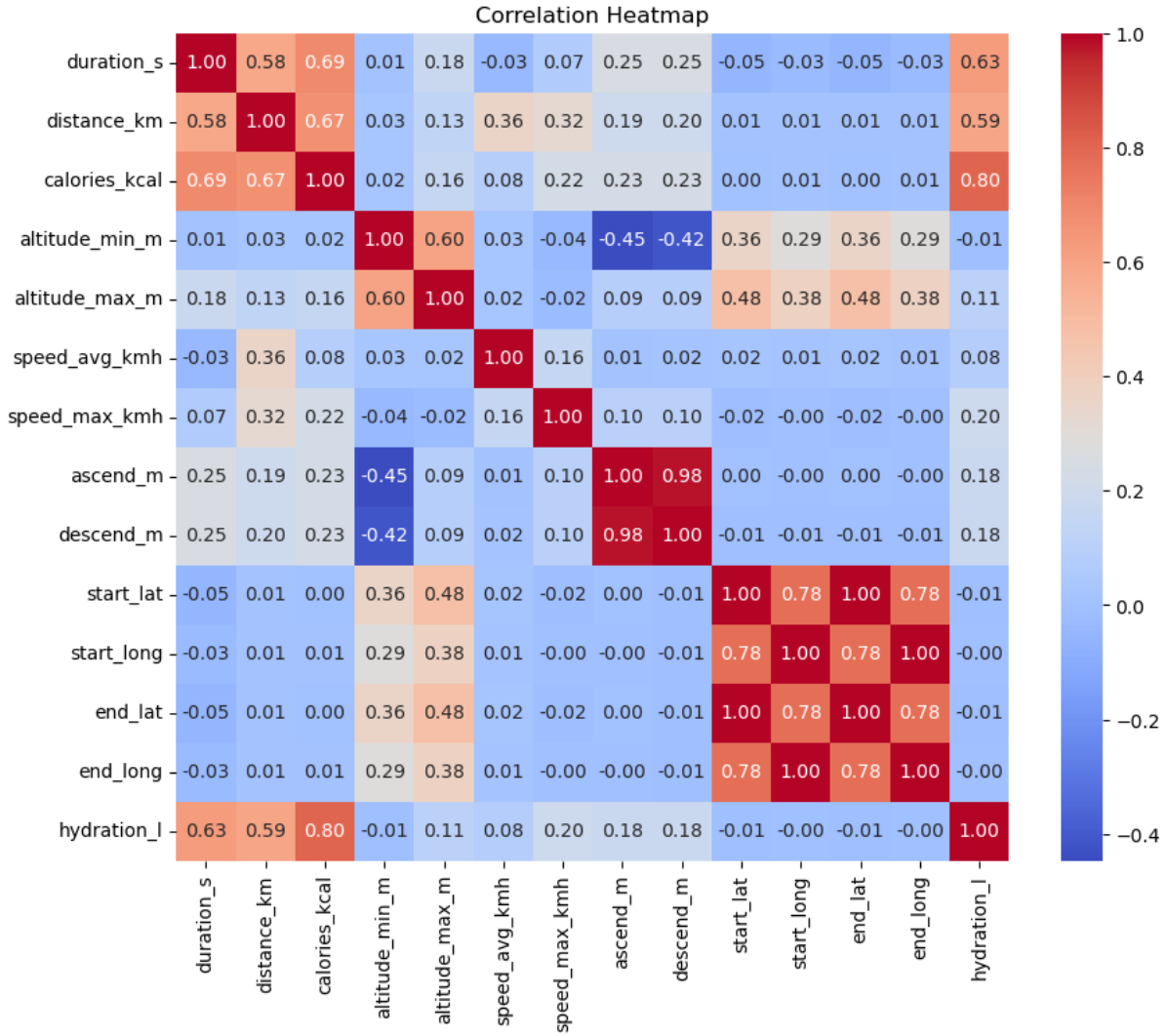
In [64]:
```
# Set the figure size
plt.figure(figsize=(20, 18))  # Adjust the width and height as needed

# Create a heatmap
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt=".2f")

# Show the heatmap
plt.show()
```



From the above correlation, its difficult to understand if there is a possibility to introduce any data leakage in the model or not but we can see it from the previous heatmap where we had only the raw features.

In [65]:
```
# Pairwise correlation matrix and heatmap
correlation_matrix = df_res.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

```
C:\Users\FARSIM\AppData\Local\Temp\ipykernel_12216\3855139582.py:2: FutureWarning: The default value of numeric_only in Dat
aFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  correlation_matrix = df_res.corr()
```



We can get rid of start latitude, end longditude columns

In [66]:
```
columns_to_drop = ['start_lat', 'end_long']

# Drop the specified columns from the DataFrame
df1 = df1.drop(columns=columns_to_drop, axis=1)
```

In [67]:
```
df1.columns
```

Out[67]:
```
Index(['sport', 'source', 'duration_s', 'distance_km', 'calories_kcal',
       'altitude_min_m', 'altitude_max_m', 'speed_avg_kmh', 'speed_max_kmh',
       'ascend_m', 'descend_m', 'start_long', 'end_lat', 'hydration_l',
       'next_exercise', 'previous_exercise', 'next_duration',
       'previous_duration', 'created_date_year', 'created_date_month',
       'created_date_day', 'created_date_hour', 'start_time_year',
       'start_time_month', 'start_time_day', 'start_time_hour',
       'end_time_year', 'end_time_month', 'end_time_day', 'end_time_hour',
       'next_time_year', 'next_time_month', 'next_time_day', 'next_time_hour',
       'previous_time_year', 'previous_time_month', 'previous_time_day',
       'previous_time_hour', 'created_date_day_0', 'created_date_day_1',
       'created_date_day_2', 'created_date_day_3', 'created_date_day_4',
       'created_date_day_5', 'created_date_day_6', 'start_time_day_0',
       'start_time_day_1', 'start_time_day_2', 'start_time_day_3',
       'start_time_day_4', 'start_time_day_5', 'start_time_day_6',
       'end_time_day_0', 'end_time_day_1', 'end_time_day_2', 'end_time_day_3',
       'end_time_day_4', 'end_time_day_5', 'end_time_day_6',
       'next_time_day_0.0', 'next_time_day_1.0', 'next_time_day_2.0',
       'next_time_day_3.0', 'next_time_day_4.0', 'next_time_day_5.0',
       'next_time_day_6.0', 'previous_time_day_0.0', 'previous_time_day_1.0',
       'previous_time_day_2.0', 'previous_time_day_3.0',
       'previous_time_day_4.0', 'previous_time_day_5.0',
       'previous_time_day_6.0'],
      dtype='object')
```

Modelling

We have both categorical and continuous columns to predict. So we will use both classifier and regression models

Predicting Categorical Variables

In [69]:
```
# Define your categorical target variables
categorical_targets = ['next_exercise', 'next_time_hour', 'next_time_day']

# Define your continuous target variable
continuous_target = 'next_duration'
```

In [70]:
```
# Split the data into features (X) and categorical target variables (y_categorical)
X = df1.drop(categorical_targets + [continuous_target], axis=1)
y_categorical = df1[categorical_targets]

# Split the data into features (X) and continuous target variable (y_continuous)
y_continuous = df1[continuous_target]

# Perform train-test split for categorical target variables
X_train, X_test, y_categorical_train, y_categorical_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42
```

In [71]:
```
# Train separate Random Forest Classifier models for categorical target variables
models_categorical = {}
for target in categorical_targets:
    model = RandomForestClassifier()
    model.fit(X_train, y_categorical_train[target])
    models_categorical[target] = model
```

In [72]:
```
# Evaluate the performance of categorical models (e.g., accuracy)
for target in categorical_targets:
    y_categorical_pred = models_categorical[target].predict(X_test)
    accuracy = accuracy_score(y_categorical_test[target], y_categorical_pred)
    print(f"Target: {target}")
    print(f"Accuracy: {accuracy}\n")
```

```
Target: next_exercise
Accuracy: 0.776034236804565

Target: next_time_hour
Accuracy: 0.2710413694721826

Target: next_time_day
Accuracy: 0.7874465049928673
```

Predicting Continuous Target Variable ('next_duration')

In [74]:
```
# Perform train-test split for the continuous target variable
X_train, X_test, y_continuous_train, y_continuous_test = train_test_split(X, y_continuous, test_size=0.2, random_state=42)
```

In [75]:
```
# Create a Linear Regression model for the continuous target variable
model_continuous = LinearRegression()
model_continuous.fit(X_train, y_continuous_train)
```

Out[75]:
```
LinearRegression

LinearRegression()
```

In [76]:
```
# Evaluate the performance of the Linear Regression model (e.g., Mean Squared Error and R-squared)
y_continuous_pred = model_continuous.predict(X_test)
mse = mean_squared_error(y_continuous_test, y_continuous_pred)
r2 = r2_score(y_continuous_test, y_continuous_pred)

print("Target: next_duration")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Target: next_duration
Mean Squared Error: 1665440.4288097578
R-squared: 0.04266002229979138
```

Conclusion

In this project, we endeavored to predict users' next exercise categories and the duration of their activities using machine learning. The results are both promising and provide valuable insights for further refinement. For the prediction of "next_exercise," we achieved an accuracy of approximately **77.6%**, suggesting that our model effectively captures exercise patterns and preferences. However, predicting "next_time_hour" presents a greater challenge, yielding an accuracy of around **27.1%**, indicative of the inherent complexity in forecasting precise timing. In contrast, "next_time_day" prediction achieved an accuracy of about **78.7%**, showcasing our model's aptitude in categorizing exercises by broad time intervals. Lastly, for "next_duration," our model exhibited a Mean Squared Error of approximately 1,665,440, and an R-squared value of **0.043**, suggesting room for improvement in accurately estimating exercise duration.

To further enhance the project, future steps may include feature engineering to extract more relevant information from the data, hyperparameter tuning to optimize model performance, and exploring more sophisticated algorithms tailored to time-based predictions. Deployment of these predictive models into fitness tracking apps and wearables could empower users with personalized recommendations and workout plans. Additionally, post-deployment, gathering user feedback and behavior data could facilitate ongoing model enhancement and fine-tuning. Overall, this project represents an exciting exploration of predictive technology in the realm of fitness, with the potential to transform the way individuals approach and engage in their exercise routines.