agg_watch_data_classifier

May 8, 2022

1 Aggregate Watch Data Classification Project

The goal of this project is to investigate and utilize the data collected from a personal smartwatch to provide daily workout recommendations. Using the data collected from the Withings brand watch, we want to predict whether or not a person will have a successful workout on a given day. Providing this insight to users in the morning could provide valuable information about how the user could structure their day or provide the necessary motivation to make a workout routine become a workout habit. The idea of a "successful workout" will be investigated as well as which data provides insights in workout performance during the next day.

As an initial analysis, the data that has been aggregated by day will be used to determine whether or not it is a good predictor of a workout the following day, additionally the sleep data will be organized and cleaned to provide additional insights

Convert to PDF: jupyter nbconvert –execute –to pdf agg_watch_data_classifier.ipynb

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

1.1 Importing the Data

In this case, we are only looking at the data that has already been aggregated by day and not every file included in the watch_data folder

[2]:
```
distance = pd.read_csv("../watch_data/aggregates_distance.csv", header=0)

passive_calories = pd.read_csv("../watch_data/aggregates_calories_passive.csv",␣
 ↪header=0)
active_calories = pd.read_csv("../watch_data/aggregates_calories_earned.csv",␣
 ↪header=0)

steps = pd.read_csv("../watch_data/aggregates_steps.csv", header=0)

sleep_data = pd.read_csv("../watch_data/sleep.csv", header=0)

workouts = pd.read_csv("../watch_data/activities.csv", header=0)
```

```
# Aggregate if I want to do same operation on all DataFrames
datasets = [distance, passive_calories, active_calories, steps, sleep_data,␣
↪workouts]
```

1.2 Data Preparation/Cleaning

Now that we have imported all of the relevant data, we need to clean the data and prepare it for model fitting

[3]:
```
def convert_to_datetime(df:pd.DataFrame):
    if 'date' in df.columns:
        df['date'] = pd.to_datetime(df['date'], infer_datetime_format=True)
    elif 'from' in df.columns:
        df['from'] = pd.to_datetime(df['from'], infer_datetime_format=True,␣
↪utc=True)
        df['to'] = pd.to_datetime(df['to'], infer_datetime_format=True,␣
↪utc=True)
    else:
        print('No columns defined as date/from/to in {}'.format(df))


# Start with the simple files, check for nans and turn the date columns into␣
↪datetime types
for df in datasets:
    convert_to_datetime(df)
```

[4]:
```
distance.rename(columns={'value':'distance'}, inplace=True)
passive_calories.rename(columns={'value':'passive calories'}, inplace=True)
active_calories.rename(columns={'value':'active calories'}, inplace=True)
steps.rename(columns={'value':'steps'}, inplace=True)
```

1.2.1 Now, we can focus on the "workouts" dataset. This dataset will require more work to get into a useable format. We will first drop the columns that contain only NaN values that provide no additional information that can be inferred or

[5]:
```
workouts.drop(['from (manual)', 'to (manual)','GPS', 'Modified'], axis=1,␣
↪inplace=True)
```

[6]:
```
workouts.head()
```

[6]:
```
                        from                        to          Timezone  \
0 2020-05-10 12:54:00+00:00 2020-05-10 15:12:00+00:00  America/New_York
1 2020-05-22 01:31:24+00:00 2020-05-22 02:16:10+00:00  America/New_York
2 2020-05-22 01:31:24+00:00 2020-05-22 02:16:10+00:00  America/New_York
3 2020-05-22 20:35:31+00:00 2020-05-22 21:41:57+00:00  America/New_York
4 2020-05-22 20:35:31+00:00 2020-05-22 21:41:57+00:00  America/New_York
```

```
  Activity type                                               Data
0       Walking  {"calories":390.5407409668,"effduration":8280,…
1          Yoga  {"calories":321.34295654297,"effduration":1080…
2         Other                                                 {}
3         Other  {"calories":85.029991149902,"effduration":3300…
4         Other                                                 {}
```

1.2.2 It's clear that a lot of information is located in the "Data" column of the workouts DataFrame and needs to be unpacked. Many of the elements in the Data column are the empty set, we also need to investigate this.

[7]:
```python
# Get the total number of times the Data column is the empty array
(workouts['Data'] == '{}').sum()
```

[7]:
```
241
```

[8]:
```python
# Maybe the "Other" workout types result in this output, however there are only␣
↪114 "Other" workouts registered
workouts['Activity type'].value_counts()
```

[8]:
```
Walking            442
Gym class          304
Other              114
Yoga                57
Cycling             10
Indoor Cycling       6
Weights              6
Tennis               4
Windsurfing          1
Elliptical           1
Name: Activity type, dtype: int64
```

[9]:
```python
# Showing which type of workouts results in the null bracket for the workout
workouts[workouts['Data'] == '{}']['Activity type'].value_counts()
```

[9]:
```
Gym class    146
Other         75
Yoga          14
Cycling        6
Name: Activity type, dtype: int64
```

1.2.3 The empty bracket categories are somewhat random, let's unpack the Data column to see what it contains for the different workouts

[10]:
```python
import ast
ast.literal_eval(workouts['Data'].iloc[0])
```

[10]:
```
{'calories': 390.5407409668,
 'effduration': 8280,
 'intensity': 0,
 'manual_distance': -1,
 'manual_calories': -1,
 'pause_duration': 0,
 'steps': 12591,
 'distance': 9793.205078125,
 'metcumul': 431.79629516602}
```

[11]:
```python
# Here we see that the 'effduration' category is simply the number of seconds␣
↪the workout is
(workouts['to'].iloc[0] - workouts['from'].iloc[0]).total_seconds()
```

[11]:
```
8280.0
```

1.2.4 As a first investigation before diving into the 'Data' column too heavily as it changes for different workout types, we can simply compute the workout duration and save it as a new column

[12]:
```python
# Create the column that generates a TimeDelta datetime object
workouts['Duration'] = (workouts['to'] - workouts['from']) / np.timedelta64(1,␣
↪'s')
workouts['Duration']
```

[12]:
```
0      8280.0
1      2686.0
2      2686.0
3      3986.0
4      3986.0
         …
940    2815.0
941    1046.0
942    1046.0
943    2393.0
944    2393.0
Name: Duration, Length: 945, dtype: float64
```

1.2.5 Here it becomes more clear that the workouts with the empty brackets are simply duplicates of the previous workouts without the data, so we can remove each of these rows from the dataset

[13]:
```
workouts = workouts[workouts['Data'] != '{}']
```

1.2.6 Now let's make a few changes to simplify the workouts data.

1. Replace the "from" and "to" columns with one date that indicates the date of the workout in addition to the "Duration" column
2. Remove the "Data" column (further investigation at a later point)

Care is needed because there may be multiple workouts in one day. In this case, if multiple workouts occur on the same day, they will be summed into one day and one duration value. In this case, we will also drop the activity type and timezone and simply find the total duration for each day that a workout was completed

[14]:
```
basicworkouts = workouts.copy()
basicworkouts['date'] = pd.to_datetime(workouts['from'].dt.date)
basicworkouts.drop(['from', 'to', 'Data', 'Timezone', 'Activity type'], axis=1,␣
↪inplace=True)
basicworkouts.set_index('date', inplace=True)
```

[15]:
```
# Sum workout duration on the days when there are more than one workout and␣
↪then consider a workout effective if it lasts longer than the 30 minutes␣
↪recommended by CDC
basicworkouts = basicworkouts.groupby(['date']).sum()
basicworkouts['Effective Workout'] = basicworkouts['Duration'] > 1800
basicworkouts['Effective Workout'].value_counts()
```

[15]:
```
True     323
False    105
Name: Effective Workout, dtype: int64
```

1.3 Next, we need to clean that sleep_data file

[16]:
```
sleep_data.head()
```

[16]:
```
                       from                         to  light (s)  deep (s)  \
0 2020-05-22 04:16:00+00:00 2020-05-22 12:19:00+00:00      17640     10260
1 2020-05-23 06:47:00+00:00 2020-05-23 12:30:00+00:00      11880      8280
2 2020-05-24 05:06:00+00:00 2020-05-24 12:13:00+00:00      11880     12900
3 2020-05-25 04:57:00+00:00 2020-05-25 14:02:00+00:00      16620     15000
4 2020-05-26 04:37:00+00:00 2020-05-26 12:23:00+00:00      14280     13560

   rem (s)  awake (s)  wake up  Duration to sleep (s)  \
0        0       1080        3                    120
1        0        420        0                    120
```

```
2        0      840      2                    120
3        0     1080      2                    120
4        0      120      0                    120

   Duration to wake up (s)  Snoring (s)  Snoring episodes  Average heart rate  \
0                      180            0                 0                  37
1                      300            0                 0                  46
2                       60            0                 0                  40
3                       60            0                 0                  40
4                        0            0                 0                  40

   Heart rate (min)  Heart rate (max)  Night events
0                33                55           NaN
1                36                95           NaN
2                32                73           NaN
3                32                77           NaN
4                31                77           NaN
```

[17]:
```python
sleep_data.describe()
```

[17]:
```
          light (s)      deep (s)  rem (s)    awake (s)     wake up  \
count    564.000000    564.000000    564.0   564.000000  564.000000
mean   13637.854610  12890.441489      0.0  1341.386525    1.765957
std     3670.772948   3078.419841      0.0   872.533436    1.421171
min        0.000000      0.000000      0.0     0.000000    0.000000
25%    11565.000000  10980.000000      0.0   720.000000    1.000000
50%    14010.000000  13020.000000      0.0  1200.000000    1.000000
75%    16020.000000  15060.000000      0.0  1800.000000    2.000000
max    23580.000000  21180.000000      0.0  4200.000000    8.000000

       Duration to sleep (s)  Duration to wake up (s)  Snoring (s)  \
count             564.000000               564.000000        564.0
mean              143.406028               187.553191          0.0
std               134.952251               253.224405          0.0
min                 0.000000                 0.000000          0.0
25%               120.000000                 0.000000          0.0
50%               120.000000               120.000000          0.0
75%               120.000000               300.000000          0.0
max              1740.000000              1860.000000          0.0

       Snoring episodes  Average heart rate  Heart rate (min)  \
count             564.0          564.000000        564.000000
mean                0.0           40.670213         32.686170
std                 0.0            7.675555          6.381235
min                 0.0            0.000000          0.000000
25%                 0.0           38.000000         32.000000
50%                 0.0           40.000000         32.000000
```

```
75%                 0.0          43.000000         33.000000
max                 0.0         112.000000        109.000000

       Heart rate (max)  Night events
count        564.000000           0.0
mean          78.333333           NaN
std           17.838795           NaN
min            0.000000           NaN
25%           70.000000           NaN
50%           80.000000           NaN
75%           88.000000           NaN
max          137.000000           NaN
```

1.3.1 It's clear from the sleep data that REM sleep is not recorded, as well as snoring, snoring episodes, and night events. These columns can be dropped. Additionally, "from" and "to" columns are not necessary as we simply want the date that the sleep occurred that corresponds with a workout later that day

[18]:
```
sleep_data.drop(['rem (s)', 'Snoring (s)', 'Snoring episodes', 'Night events'],␣
↪axis=1, inplace=True)
```

[19]:
```
sleep_data['date'] = pd.to_datetime(sleep_data["from"].dt.date)
```

[20]:
```
# If we look at the values for the dates, we see that we have one day in which␣
↪there are 2 recorded sleeps! Let's investigate this date
sleep_data['date'].value_counts()
```

[20]:
```
2022-01-27    2
2021-04-03    1
2021-11-11    1
2021-08-26    1
2021-05-20    1
             ..
2021-03-04    1
2021-05-05    1
2021-10-27    1
2021-09-06    1
2022-02-28    1
Name: date, Length: 563, dtype: int64
```

[21]:
```
sleep_data[sleep_data['date'] == '20220127']
```

[21]:
```
                         from                        to  light (s)  deep (s)  \
498 2022-01-27 01:52:00+00:00 2022-01-27 10:49:00+00:00      14520     13860
499 2022-01-27 15:47:00+00:00 2022-01-27 21:28:00+00:00       2940     16620

     awake (s)  wake up  Duration to sleep (s)  Duration to wake up (s)  \
```

```
498        3840         4                   120                        300
499         900         1                   120                        600

     Average heart rate  Heart rate (min)  Heart rate (max)        date
498                  40                32                80  2022-01-27
499                 112               109               115  2022-01-27
```

1.3.2 After investigating, it's clear that the duplicate was due to a long nap I took on vacation :) I will remove this sleep record from the data set in this case. In other applications, it may become necessary to create some outlier detection to determine when a sleep event occurs outside of the usual time.

[22]:
```python
sleep_data.drop(499, inplace=True)
sleep_data.drop(['from', 'to'], axis=1, inplace=True)
sleep_data.set_index('date', inplace=True)
```

[23]:
```python
# Cleaned sleep data
sleep_data
```

[23]:
```
            light (s)  deep (s)  awake (s)  wake up  Duration to sleep (s)  \
date
2020-05-22      17640     10260       1080        3                    120
2020-05-23      11880      8280        420        0                    120
2020-05-24      11880     12900        840        2                    120
2020-05-25      16620     15000       1080        2                    120
2020-05-26      14280     13560        120        0                    120
…                 …         …         …        …                    …
2022-04-13      13500     14040       1560        2                    120
2022-04-14      18720     14640        120        0                    120
2022-04-15      16320     15120       1980        2                    120
2022-04-16      16800     15060       1740        2                    120
2022-04-17      15600     10500        180        0                    120

            Duration to wake up (s)  Average heart rate  Heart rate (min)  \
date
2020-05-22                      180                  37                33
2020-05-23                      300                  46                36
2020-05-24                       60                  40                32
2020-05-25                       60                  40                32
2020-05-26                        0                  40                31
…                             …                  …                …
2022-04-13                        0                  41                32
2022-04-14                        0                  44                33
2022-04-15                        0                  38                33
2022-04-16                        0                  36                32
2022-04-17                       60                  39                33
```

```
             Heart rate (max)
date
2020-05-22                 55
2020-05-23                 95
2020-05-24                 73
2020-05-25                 77
2020-05-26                 77
…                       …
2022-04-13                114
2022-04-14                 84
2022-04-15                 80
2022-04-16                 53
2022-04-17                 73

[563 rows x 9 columns]
```

1.4 Now that the more complicated datasets have been cleaned, let's quickly clean the distance, passive_calories, active_calories, and steps sets

[24]:
```
distance.set_index('date', inplace=True)
passive_calories.set_index('date', inplace=True)
active_calories.set_index('date', inplace=True)
steps.set_index('date', inplace=True)
```

[25]:
```
distance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 711 entries, 2022-04-17 to 2020-05-07
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   distance  711 non-null    float64
dtypes: float64(1)
memory usage: 11.1 KB
```

[26]:
```
basicworkouts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 428 entries, 2020-05-10 to 2022-04-15
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Duration           428 non-null    float64
 1   Effective Workout  428 non-null    bool
dtypes: bool(1), float64(1)
memory usage: 7.1 KB
```

1.5 Now, let's join the datasets together on the date

[27]:
```
X_data = pd.merge(distance, passive_calories, how='inner', on='date')
X_data = pd.merge(X_data, active_calories, how='inner', on='date')
X_data = pd.merge(X_data, steps, how='inner', on='date')
X_data = pd.merge(X_data, sleep_data, how='inner', on='date')
X_data = pd.merge(X_data, basicworkouts, how='inner', on='date')
X_data
```

[27]:
```
             distance  passive calories  active calories  steps  light (s)  \
date
2022-04-15   2579.809          1869.678          106.592   3082      16320
2022-04-14   5363.195          1869.678          208.608   6596      18720
2022-04-13   2611.168          1869.678          102.255   3212      13500
2022-04-12   2773.922          1869.678          112.184   3429      14400
2022-04-10   3565.152          1869.678          139.220   4370      12540
…                 …               …                …      …        …
2020-05-28   5218.729          1914.158          206.033   6483      16020
2020-05-27   2222.013          1914.158           87.538   2596      15120
2020-05-25   1772.393          1914.158           69.988   2052      16620
2020-05-24  12708.024          1914.158          502.375  16357      11880
2020-05-22   5737.927          1914.158          226.621   6758      17640

            deep (s)  awake (s)  wake up  Duration to sleep (s)  \
date
2022-04-15     15120       1980        2                    120
2022-04-14     14640        120        0                    120
2022-04-13     14040       1560        2                    120
2022-04-12     12420       1500        1                    180
2022-04-10     13440       2520        3                    120
…              …         …      …                    …
2020-05-28     13740       1980        1                    120
2020-05-27      8820       1260        3                    120
2020-05-25     15000       1080        2                    120
2020-05-24     12900        840        2                    120
2020-05-22     10260       1080        3                    120

            Duration to wake up (s)  Average heart rate  Heart rate (min)  \
date
2022-04-15                        0                  38                33
2022-04-14                        0                  44                33
2022-04-13                        0                  41                32
2022-04-12                      480                  36                33
2022-04-10                       60                  38                33
…                            …                  …                …
2020-05-28                     1680                  41                33
2020-05-27                      240                  38                33
```

```
2020-05-25                       60                  40                 32
2020-05-24                       60                  40                 32
2020-05-22                      180                  37                 33

            Heart rate (max)  Duration  Effective Workout
date
2022-04-15                80    3439.0               True
2022-04-14                84    2815.0               True
2022-04-13               114    3574.0               True
2022-04-12                47    1080.0              False
2022-04-10                69    6548.0               True
…                        …       …                 …
2020-05-28                99    1980.0               True
2020-05-27                47    2647.0               True
2020-05-25                77    1560.0              False
2020-05-24                73    8676.0               True
2020-05-22                55    7932.0               True

[350 rows x 15 columns]
```

2 2. Exploratory Data Analysis

2.0.1 Now that the data has been cleaned to a usable format, we can briefly explore the data before applying different ML techniques. It should be noted that this data will only include the days in which data exists for each of the previous datasets, i.e. days in which I had a workout, recorded my sleep, and other data exists (recorded automatically every day)

[28]:
```
fig, axes = plt.subplots(figsize=(12, 6))
axes.scatter(X_data.index, X_data['steps'])
axes.set_xlabel('Date')
axes.set_ylabel('Steps')

axes.set_title("My Steps on Workout Days Over Time")
```

[28]:
```
Text(0.5, 1.0, 'My Steps on Workout Days Over Time')
```

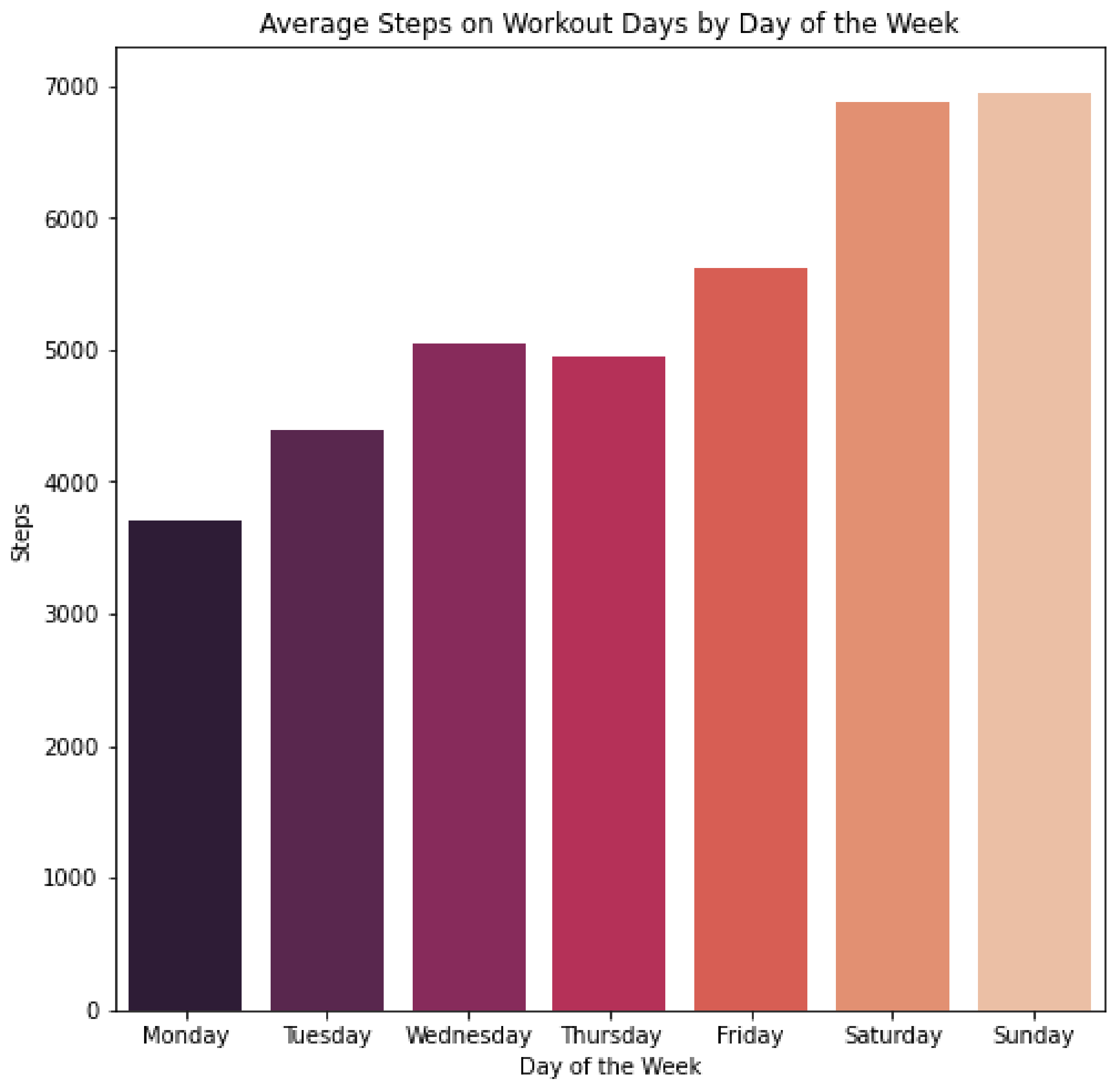
[29]:
```
cats = [ 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',␣
↪'Sunday']
steps_by_day = X_data.groupby(X_data.index.day_name()).mean().reindex(cats)
```

[30]:
```
import seaborn as sb

fig, axes = plt.subplots(figsize=(8, 8))
sb.barplot(x = steps_by_day.index, y = steps_by_day['steps'], palette="rocket")
axes.set_xlabel('Day of the Week')
axes.set_ylabel('Steps')

axes.set_title("Average Steps on Workout Days by Day of the Week")
```

[30]:
```
Text(0.5, 1.0, 'Average Steps on Workout Days by Day of the Week')
```



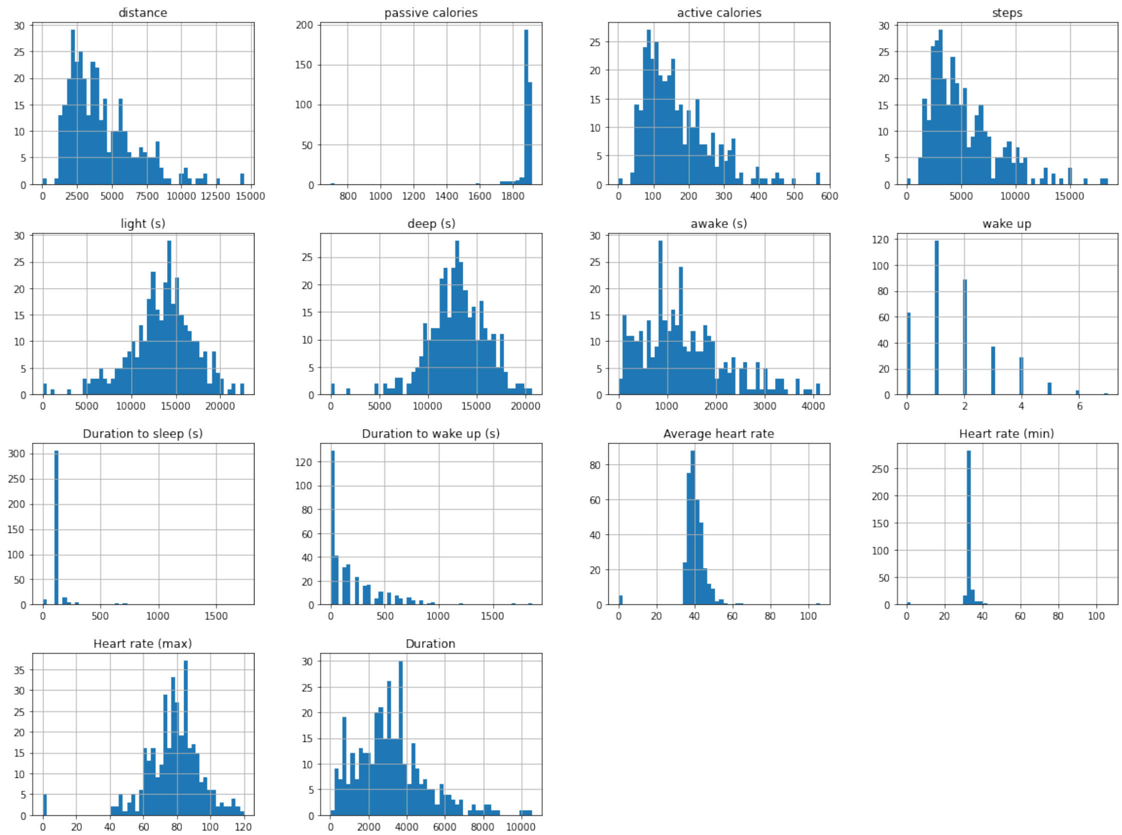
[31]:
```
X_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 350 entries, 2022-04-15 to 2020-05-22
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   distance              350 non-null    float64
 1   passive calories      350 non-null    float64
 2   active calories       350 non-null    float64
 3   steps                 350 non-null    int64
 4   light (s)             350 non-null    int64
 5   deep (s)              350 non-null    int64
 6   awake (s)             350 non-null    int64
```

```
 7   wake up                  350 non-null    int64
 8   Duration to sleep (s)    350 non-null    int64
 9   Duration to wake up (s)  350 non-null    int64
 10  Average heart rate       350 non-null    int64
 11  Heart rate (min)         350 non-null    int64
 12  Heart rate (max)         350 non-null    int64
 13  Duration                 350 non-null    float64
 14  Effective Workout        350 non-null    bool
dtypes: bool(1), float64(4), int64(10)
memory usage: 41.4 KB
```

[32]:
```python
X_data.hist(bins=50, figsize=(20,15))
```

[32]:
```
array([[<AxesSubplot:title={'center':'distance'}>,
        <AxesSubplot:title={'center':'passive calories'}>,
        <AxesSubplot:title={'center':'active calories'}>,
        <AxesSubplot:title={'center':'steps'}>],
       [<AxesSubplot:title={'center':'light (s)'}>,
        <AxesSubplot:title={'center':'deep (s)'}>,
        <AxesSubplot:title={'center':'awake (s)'}>,
        <AxesSubplot:title={'center':'wake up'}>],
       [<AxesSubplot:title={'center':'Duration to sleep (s)'}>,
        <AxesSubplot:title={'center':'Duration to wake up (s)'}>,
        <AxesSubplot:title={'center':'Average heart rate'}>,
        <AxesSubplot:title={'center':'Heart rate (min)'}>],
       [<AxesSubplot:title={'center':'Heart rate (max)'}>,
        <AxesSubplot:title={'center':'Duration'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



3 3. Feature Exploration/Engineering

3.1 Many of the techniques have been chosen from the text "Hands on Machine Learning"

[33]:
```
corr_mat = X_data.corr()
corr_mat['Effective Workout'].sort_values(ascending=False)
```

[33]:
```
Effective Workout          1.000000
Duration                   0.645952
light (s)                  0.124674
Duration to wake up (s)    0.108354
distance                   0.092069
active calories            0.090891
steps                      0.090876
deep (s)                   0.034190
awake (s)                  0.032055
wake up                    0.017298
Heart rate (max)           0.005117
passive calories          -0.000140
```

```
Duration to sleep (s)      -0.016959
Average heart rate         -0.033412
Heart rate (min)           -0.051027
Name: Effective Workout, dtype: float64
```

3.2 After looking at the correlations, it's clear that some of the data would not make sense to predict whether or not someone will have an effective workout. After consideration, it makes sense to predict whether an effective workout will occur or not based off of information from the sleep data OR data from the previous day. Because of this, the following steps will be made to modify the data:

1. The Duration, distance, active calories, steps, and passive calories categories will be removed from the current day as they occur concurrently with the current day's workout and may be confounding variables
2. The distance, active/passive calories, and steps from the previous day will be added in as possible influence over an effective workout or not
3. The total time asleep is added as a feature

[34]:
```
# Remove confounding variables
x_test = X_data.drop(['Duration', 'distance', 'active calories', 'steps',␣
↪'passive calories'], axis=1)
```

[35]:
```
# Add total time asleep as a column
x_test['total sleep'] = X_data['light (s)'] + X_data['deep (s)']
```

[36]:
```
from datetime import datetime, timedelta
prev_day_index = x_test.index - timedelta(days=1)
x_test['prevday steps'] = steps['steps'].loc[prev_day_index].values
x_test['prevday active cals'] = active_calories['active calories'].
↪loc[prev_day_index].values
x_test['prevday passive cals'] = passive_calories['passive calories'].
↪loc[prev_day_index].values
x_test['prevday distance'] = distance['distance'].loc[prev_day_index].values
```

[37]:
```
corr_mat = x_test.corr()
corr_mat['Effective Workout'].sort_values(ascending=False)
```

[37]:
```
Effective Workout           1.000000
light (s)                   0.124674
total sleep                 0.111419
Duration to wake up (s)     0.108354
deep (s)                    0.034190
awake (s)                   0.032055
wake up                     0.017298
Heart rate (max)            0.005117
Duration to sleep (s)      -0.016959
```

```
prevday passive cals      -0.023298
Average heart rate        -0.033412
Heart rate (min)          -0.051027
prevday distance          -0.154032
prevday steps             -0.155316
prevday active cals       -0.156304
Name: Effective Workout, dtype: float64
```

3.2.1 Now, we can plot the correlation matrix with all of our data we will use to predict an effective workout.

From this data, the main correlation to "Effective Workout" come from the sleep data with the highest correlation related to light sleep.

[38]:
```
corr_mat.style.background_gradient(cmap='coolwarm')
```

[38]:
```
<pandas.io.formats.style.Styler at 0x7fc90031f670>
```

[39]:
```
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 350 entries, 2022-04-15 to 2020-05-22
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   light (s)               350 non-null    int64
 1   deep (s)                350 non-null    int64
 2   awake (s)               350 non-null    int64
 3   wake up                 350 non-null    int64
 4   Duration to sleep (s)   350 non-null    int64
 5   Duration to wake up (s) 350 non-null    int64
 6   Average heart rate      350 non-null    int64
 7   Heart rate (min)        350 non-null    int64
 8   Heart rate (max)        350 non-null    int64
 9   Effective Workout       350 non-null    bool
 10  total sleep             350 non-null    int64
 11  prevday steps           350 non-null    int64
 12  prevday active cals     350 non-null    float64
 13  prevday passive cals    350 non-null    float64
 14  prevday distance        350 non-null    float64
dtypes: bool(1), float64(3), int64(11)
memory usage: 41.4 KB
```

3.3 Prepping Data for Fitting Classifiers

[40]:
```
# Getting a test set
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(x_test, test_size=0.2, random_state=42)
```

[41]:
```
y_train = train_set['Effective Workout'].values.astype(int)
x_train = train_set.drop(['Effective Workout'], axis=1)

y_test = test_set['Effective Workout'].values.astype(int)
x_test = test_set.drop(['Effective Workout'], axis=1)
```

[42]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('std_scaler', StandardScaler()),
    ])
x_train_prepared = num_pipeline.fit_transform(x_train)
x_test_prepared  = num_pipeline.fit_transform(x_test)
```

3.4 Stochastic Gradient Descent classifier

[43]:
```
# Training the classifier in the standard way
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(x_train_prepared, y_train)
```

[43]:
```
SGDClassifier(random_state=42)
```

[44]:
```
# Getting the cross-validation score
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, x_train_prepared, y_train, cv=3, scoring="accuracy")
```

[44]:
```
array([0.61702128, 0.65591398, 0.65591398])
```

[45]:
```
# Getting the confusion matrix
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

y_train_pred = cross_val_predict(sgd_clf, x_train_prepared, y_train, cv=3)


y_scores = cross_val_predict(sgd_clf, x_train_prepared, y_train, cv=3,␣
 ↪method="decision_function")
```

[46]:
```
confusion_matrix(y_train, y_train_pred)
```

[46]:
```
array([[ 22,  49],
       [ 51, 158]])
```

3.4.1 Looking at the results we see:

1. Our classifier performs equally poorly at Type 1 and 2 errors and that our resulting precision and recall is very similar. This may mean that our SGD classifier can do no better than what is shown here without tradeoffs between precision and recall
2. The average cross-validation accuracy of around 0.63 is somewhat better than random chance, but doesn't tell the full story because the precision and recall are decent compared with the accuracy
3. Precision: 0.763
4. Recall : 0.756
5. F1 score : 0.7596
6. The classifier is not good at predicting true negative, or cases when a poor workout is expected. This could mean that we need to train a different model, or that the input features are simply not good predictors of a good workout or not

[47]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

precision_score(y_train, y_train_pred)
```

[47]:
```
0.7632850241545893
```

[48]:
```
recall_score(y_train, y_train_pred)
```

[48]:
```
0.7559808612440191
```

[49]:
```
f1_score(y_train, y_train_pred)
```

[49]:
```
0.7596153846153845
```

[50]:
```
# What would the accuracy be if we simply guessed a good workout every time?
sum(np.ones(len(y_train)) == y_train)/len(y_train)
```

[50]:
```
0.7464285714285714
```

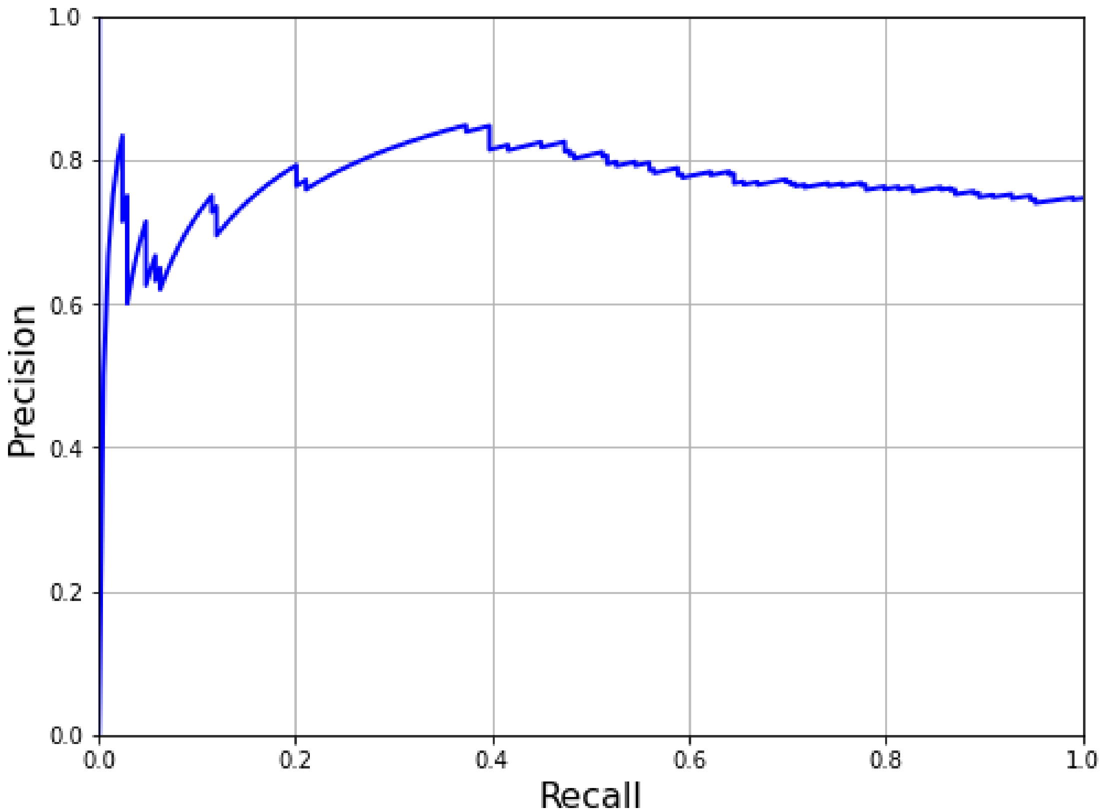
[51]:
```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train, y_scores)

def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
```

```
    plt.axis([0, 1, 0, 1])
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.show()
```



3.4.2 We can look at the ROC curve to see the performance of our SGD classifier

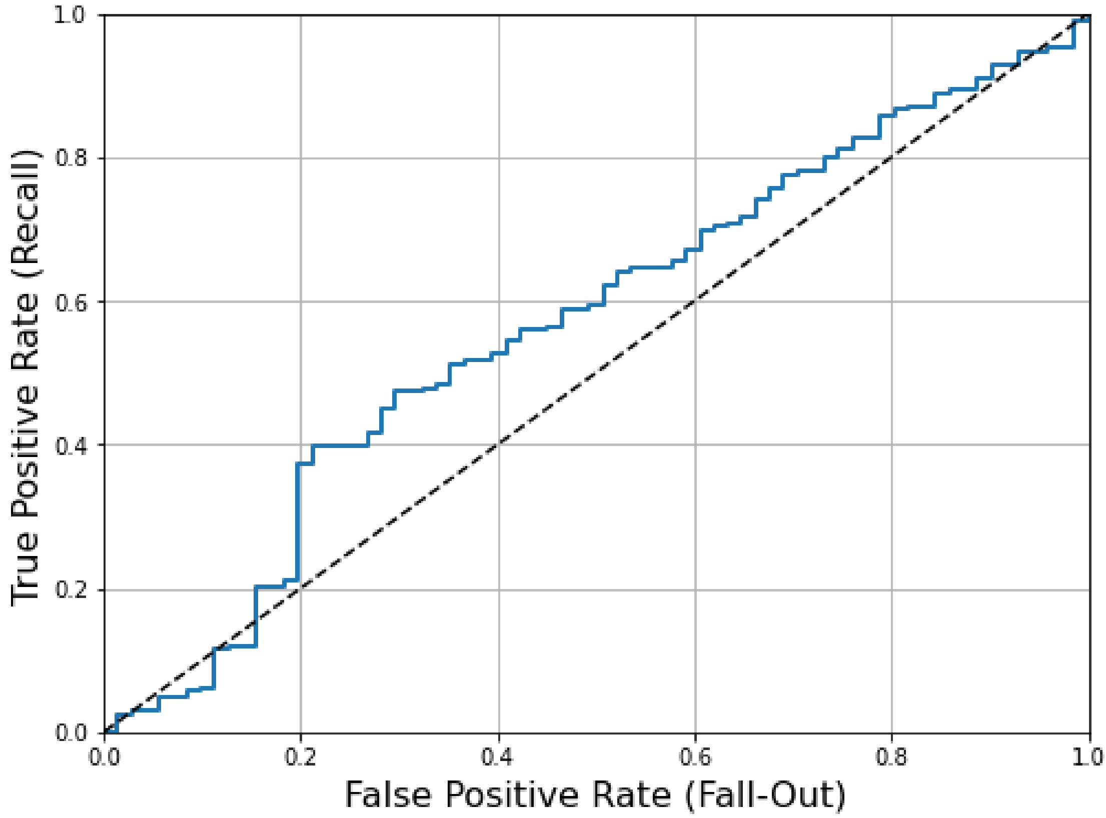
[52]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train, y_scores)



def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16)
    plt.ylabel('True Positive Rate (Recall)', fontsize=16)
```

```
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_roc_curve(fpr, tpr)
plt.show()
```



[53]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train, y_scores)
```

[53]: 0.5665476110250017

3.5 From this initial investigation, the current classifier performs poorly. This could be due to a number of factors:

1. Small datasets (more data may differentiate bad workouts from good workouts more)
2. Class imbalance (75% are effective workouts and 25% are not)
3. Incorrect metrics (perhaps a 30 minute workout is not the sure-fire metric that was expected)

3.6 First, other binary classifiers will be tested and then further analysis will be done