Group B Assignment

Description

The goal of this Jupyter Notebook is to analyze data previously loaded into HDFS using NiFi. The data to be analyzed comes from a Kaggle project related to Fitbit users. The complete details and context of the information used can be found in the following link: https://www.kaggle.com/datasets/arashnic/fitbit

From all the available data, we will only focus on the daily information. Some of the information available in the dataset includes:

- Daily activity tracking
- Daily Calories
- Daily steps
- Sleep patterns
- Weight information

Using this different variables we will do some analysis in order to come up with some interesting insights, which we will later display with help of some visualization libraries (seaborn, matplotlib, etc.).

1. Data and libraries import

We will start by importing all the libraries that we will use for the analysis. Additionaly, we will also import from HDFS each of the `.csv` files in their raw form.

In [1]:
```
# Spark tools:
import findspark
findspark.init()

import pandas as pd
pd.set_option('display.max_colwidth', None)

import os
os.environ['PYSPARK_SUBMIT_ARGS'] = ' pyspark-shell'

import pyspark.sql.functions as F

# Visualization tools:
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Stats tools:
from scipy import stats
import numpy as np
```

1.1 Create SparkSession

In [2]:
```
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import *

spark = (SparkSession.builder
.appName("Fitbit - Data")
.config("spark.sql.warehouse.dir","hdfs://localhost:9000/warehouse")
.getOrCreate())
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/
spark3/jars/spark-unsafe_2.12-3.2.1.jar) to constructor java.nio.DirectByteBuffer
(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.uns
afe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflectiv
e access operations
WARNING: All illegal access operations will be denied in a future release
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel
(newLevel).
```

1.2 Import files in raw format

Since the `dailyActivity_merged.csv` file also includes information related to **calories**, **intensities** and **steps** we are only importing this file. The other files are no longer needed.

Given that the files we are using do not have more than 1000 rows, we can use the `inferSchema` option. If we were handling a larger dataset this would not be advised and we would manually input each columns data type.

In [3]:
```
fitbit_activity_raw = (spark.read
                       .option("header", "true")
                       .option("inferSchema", "true")
                       .csv("hdfs://localhost:9000/datalake/raw/fitbit/raw_data/dailyA

fitbit_sleep_raw = (spark.read
                         .option("header", "true")
                         .option("inferSchema", "true")
                         .csv("hdfs://localhost:9000/datalake/raw/fitbit/raw_data/slee
fitbit_weightLog_raw = (spark.read
                        .option("header", "true")
                        .option("inferSchema", "true")
                        .csv("hdfs://localhost:9000/datalake/raw/fitbit/raw_data/weig
```

2. Data cleaning:

In this step we are going to keep only the columns that are of interest for us. Since the dates are on `M/d/yyyy` format, we need to also transform this to a valid pySpark date format.

Later we will merge the information from across the tables using as a common key the **Id** and the **ActivityDate**.

2.1 Activity table:

In this step we are first creating an **ActiveMinutes** column, which will be the sum of all the **xxActiveMinutes** columns.We are also transforming the **ActivityDate** column from each initial value to a **date** format.

We are also adding a weekday column that we will later use in order to analyze differences between weekends and weekdays.

Since we are not going to use all the information from this table we are also dropping some columns.

In [4]:
```
fitbit_activity_raw.toPandas().head(5)
fitbit_activity = (fitbit_activity_raw
                   .withColumn("ActiveMinutes", F.expr("VeryActiveMinutes + Fairly/
                   .withColumn("ActivityDate", to_date(col("ActivityDate"), "M/d/y
                   .withColumn("Weekdays", date_format(col("ActivityDate"), "E"))
                   .withColumn("IsWeekend", F.expr("CASE WHEN Weekdays IN ('Sat',
                   .drop('Logged_activitiesDistance', 'VeryActiveDistance', 'Track
                        'ModeratelyActive', 'LightActiveDistance', 'SedentaryActiv
                         'FairlyActiveMinutes', 'LightlyActiveMinutes', 'Moderatel
                  )

# fitbit_activity.toPandas().head()
```

2.2 Sleep table:

For this table we are also correctly formatting the date. In this case we are only interested on the date part, so an extra step needs to be taken.

For this we are only keeping the date part using Pyspark's `split()` function and then applying the same `to_date()` function as above.

We are also createing a new column called **SleepScore** which consists on the ratio between **TotalMinutesAsleep** and **TotalTimeInBed**.

Finally, we do not care about the **TotalSleepRecords** column, so we are dropping it.

In [6]:
```
fitbit_sleep = (fitbit_sleep_raw
                .withColumn("SleepScore", F.expr("coalesce(TotalMinutesAsleep/Total
                .withColumn("SleepDay", to_date(split(col("SleepDay"), ' ').getIte
                .withColumn("Weekdays", date_format(col("SleepDay"), "E"))
                .withColumn("IsWeekend", F.expr("CASE WHEN Weekdays IN ('Sat', 'Su
                .drop("TotalSleepRecords"))

# fitbit_sleep.toPandas().head()
```

2.3 Weight table:

For this table we do a similar exercise as before. We are transforming the **Date** column and also dropping unnecessary information.

However, since we do not expect the weight data to change that much from day to day, in a future transformation we are only going to keep the **Id** of each subject. Since some of the users inputted more than one entry, we are grouping this information and keeping the average.

In [7]:
```
fitbit_weight = (fitbit_weightLog_raw
                 .withColumn("Date", to_date(split(col("Date"), ' ').getItem(0), "
                 .drop("Fat", "IsManualReport", "LogId", "WeightPounds")
                )
# fitbit_weight.toPandas().head()
```

3. Grouping data:

Now we are going to do some data transformation in order to have the data grouped by the users and it it is either a weekend or a weekday.

For the weekend vs. weekday grouping we will need to pivot the **IsWeekend** columnn. This will be in order to be able to compare both samples and see if there is a noticeable difference in behaviour. For this we will do a hypothesis test comparing both means.

3.1 Activity table:

3.1.1 Grouping by users:

In [8]:
```
fitbit_activity_user_grouped = (fitbit_activity
                                .groupBy("Id")
                                .agg(
                                    F.round(F.avg("TotalSteps")).alias("AvgSteps")
                                    F.round(F.avg("TotalDistance")).alias("AvgDist
                                    F.round(F.avg("SedentaryMinutes")).alias("AvgS
                                    F.round(F.avg("ActiveMinutes")).alias("AvgActi
                                    F.round(F.avg("Calories")).alias("AvgCalories"
                                )
)

# fitbit_activity_user_grouped.toPandas().head(1)
```

3.1.2 Grouping by weekends vs. weekdays:

In [9]:
```
fitbit_activity_wknd_grouped = (fitbit_activity
                                .groupby("Id", "IsWeekend")
                                .agg(
                                    F.round(F.avg("TotalSteps")).alias("AvgSteps")
                                    F.round(F.avg("TotalDistance")).alias("AvgDist
                                    F.round(F.avg("SedentaryMinutes")).alias("AvgS
                                    F.round(F.avg("ActiveMinutes")).alias("AvgActi
                                    F.round(F.avg("Calories")).alias("AvgCalories"
                                )
                               )

# fitbit_activity_wknd_grouped.toPandas().head()
```

3.2 Sleep table:

3.2.1 Grouping by users:

In [10]:
```
fitbit_sleep_user_grouped = (fitbit_sleep
                                .groupBy("Id")
                                .agg(
                                    F.round(F.avg("TotalMinutesAsleep")).alias("Av
                                    F.round(F.avg("TotalTimeInBed")).alias("AvgBed
                                )

)

# fitbit_sleep_user_grouped.toPandas().head(1)
```

3.2.2 Grouping by weekends vs. weekdays:

In [11]:
```
fitbit_sleep_wknd_grouped = (fitbit_sleep
                                .groupBy("Id", "IsWeekend")
                                .agg(
                                    F.round(F.avg("TotalMinutesAsleep")).alias("Av
                                    F.round(F.avg("TotalTimeInBed")).alias("AvgBed
                                )

)

# fitbit_sleep_wknd_grouped.toPandas().head()
```

3.3 Fitbit Weight:

In [12]:
```
fitbit_weight_grouped = (fitbit_weight
                          .groupBy("Id")
                          .agg(
                             F.round(F.avg("WeightKg")).alias("avg_weight"),
                             F.round(F.avg("BMI")).alias("avg_BMI")
                                 )

)

fitbit_weight_grouped.toPandas().shape
```

Out[12]:
```
(8, 3)
```

Unfortunately, since we only have information from 8 users about the weight data we are not going to be able to use them for our analysis.

From now forward we will only include in our analysis the **Activity** and the **Sleep** tables.

4. Joining the tables:

In this step we will join the four tables we have (Activiity and Sleep, by User and by Weekday) and only have two tables in total (one for the grouping by User and another one for the grouping by Weekday).

4.1 Joining the user tables:

Since the sleep score is calculated as a ratio, it is necessary to calculate it once more with the aggregated tables.

In [13]:
```
fitbit_user_joined = (fitbit_activity_user_grouped.alias('a')
                      .join(fitbit_sleep_user_grouped.alias('s'), F.col('a.Id')==F

)

fitbit_user_joined = (fitbit_user_joined
                          .withColumn('AvgSleepScore', F.expr("round(AvgSleepMins/A
                        )

# fitbit_user_joined.toPandas()['Id'].nunique()
```

4.2 Joining the weekends tables:

In [14]:
```
fitbit_wknd_joined = (fitbit_activity_wknd_grouped.alias('a')
                      .join(fitbit_sleep_wknd_grouped.alias('s'),
                           (F.col('a.Id') == F.col('s.Id')) & (F.col('a.IsWeekend'
                            'inner')
                      .withColumn('AvgSleepScore', F.expr("round(AvgSleepMins/AvgB
)

# fitbit_wknd_joined.toPandas().head()
```

5. Univariate analysis:

Now we are going to use the pySpark function in order to summarize the information of the numeric variables. This is in order to understand overall how our population looks and if there are some columns that are more prone to variability.

For legibility purposes we are formatting the columns (in order to avoid having a lot of decimals).

5.1 Summary statistics:

In [15]:
```
fitbit_user_summary = (fitbit_user_joined.select(['AvgSteps', 'AvgDistance',
                                                  'AvgSedentaryMins', 'AvgActiveMi
                                                  'AvgActiveMins', 'AvgCalories',
                                                  'AvgSleepMins', 'AvgBedMins'
                                                 ])
                       .summary()
                       .toPandas()
                      )

conversion_dict = {'AvgSteps': float, 'AvgDistance': float, 'AvgSedentaryMins': fl
                   'AvgCalories': float, 'AvgSleepMins': float, 'AvgBedMins': float
fitbit_user_summary.astype(conversion_dict).round(1)
```

Out[15]:

	summary	AvgSteps	AvgDistance	AvgSedentaryMins	AvgActiveMins	AvgActiveMins	AvgCalo
0	count	24.0	24.0	24.0	24.0	24.0	2
1	mean	7391.6	5.2	931.1	221.1	221.1	229
2	stddev	3609.4	2.6	228.4	80.3	80.3	56
3	min	916.0	1.0	662.0	41.0	41.0	151
4	25%	4717.0	3.0	716.0	202.0	202.0	187
5	50%	7685.0	5.0	837.0	245.0	245.0	213
6	75%	9520.0	7.0	1148.0	273.0	273.0	256
7	max	14763.0	11.0	1317.0	328.0	328.0	341

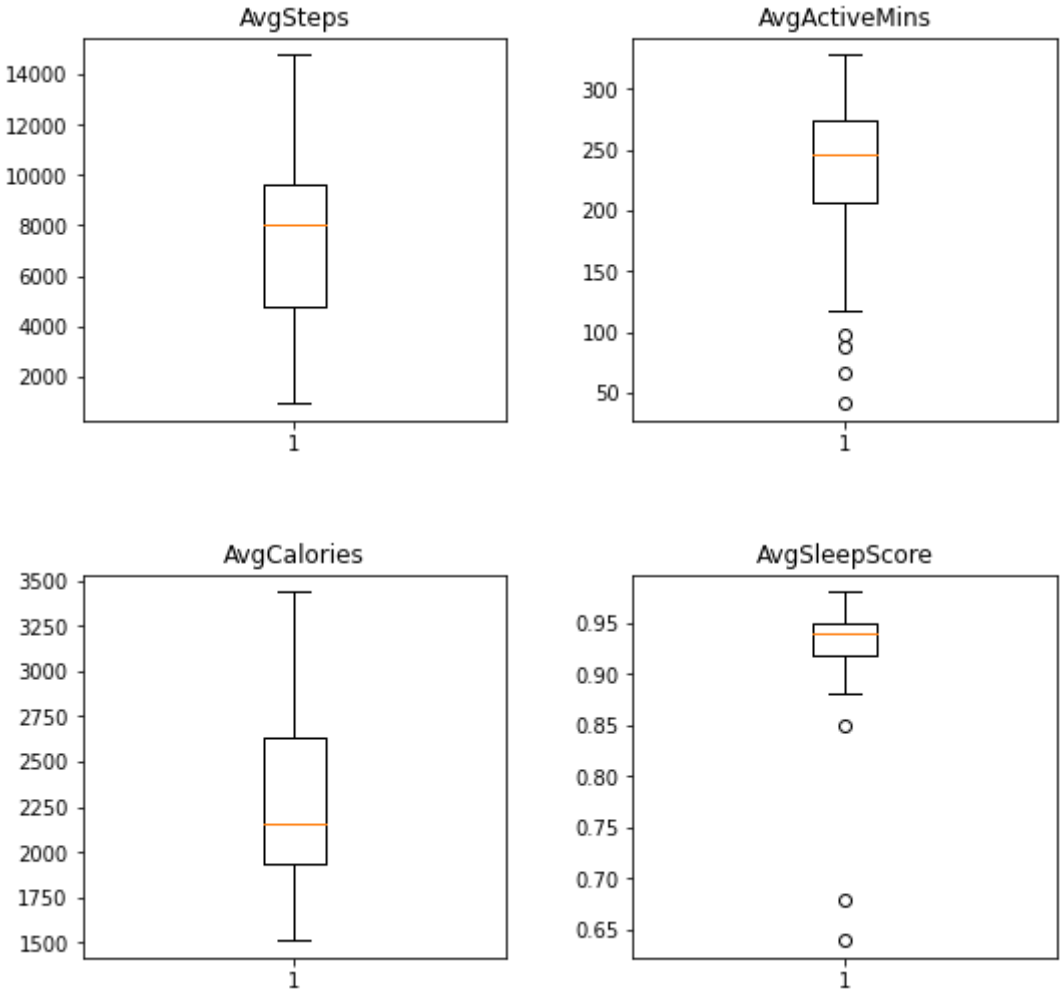
One thing we can notice is that overall there is some spread among the users. For example, the max values for the **AvgSteps** is 3 times bigger than the IQR. This type of observation could be considered and outlier. Something similar happens with the other metrics, which might lead us to think that the population that was sampled differ in behaviours.

Another thing we can do to better understand this behaviour is computing the boxplots of each of the variables of interest.

5.2 Boxplots:

In [16]:

```
fitbit_user_boxplot = (fitbit_user_joined.select(['AvgSteps', 'AvgDistance', 'AvgS
                                                  'AvgActiveMins', 'AvgCalories',
                                                 ])
                       .toPandas()
                      )

figure, axis = plt.subplots(2, 2, figsize=(7.5, 7.5))
axis[0, 0].boxplot(fitbit_user_boxplot['AvgSteps'], )
axis[0, 0].set_title('AvgSteps')

axis[0, 1].boxplot(fitbit_user_boxplot['AvgActiveMins'])
axis[0, 1].set_title('AvgActiveMins')

axis[1, 0].boxplot(fitbit_user_boxplot['AvgCalories'])
axis[1, 0].set_title('AvgCalories')

axis[1, 1].boxplot(fitbit_user_boxplot['AvgSleepScore'])
axis[1, 1].set_title('AvgSleepScore')

figure.subplots_adjust(left=0.08, right=0.98, bottom=0.05, top=0.9,
                    hspace=0.4, wspace=0.3)

plt.show()
```



As we can see, variables such as **AvgSteps** and **AvgCalories** tend to have a smoother behaviour with no extreme values. In contrast, **AvgActiveMins** has plenty of outliers more on the higher end.

Contrary to the others, **AvgSleepScore** has extreme values that drift off the general observations by a longer stretch. This could mean that some users have a high sedentary tendency (spending a lot of time in bed without sleeping) or have a bad sleep quality (insomnia, trouble sleeping, etc.).

6. Analysis of populations (Weekend vs. Weekdays):

In this part of our analysis we now want to see if the overall behaviour of our users varies depending if it is a weekday or weekend. For this we would do a *paired two-sample t-test* for each of the variables.

As usual, the **null hypothesis** will be

$$H_0 : \mu_{weekend} = \mu_{weekdays}.$$

This means we assume that there is no change on the behaviour due to it being a weekend or weekday. We are going to use an $\alpha = 0.05$ for the test in order to either accept or reject the null hypothesis.

The analysis will be performed for the same variables as shown in the boxplots, that is: **AvgSteps**, **AvgActiveMins**, **AvgCalories** and **AvgSleepMins**.

6.1 Daily Average active minutes comparison:

It is interesting to understand if there are changes in the behaviour given the day of the week. Several factors might influence the activity of a user, such as having to move less in a weekend due to not having to commute, or in contrast being more active as a consequence of having more time.

In [17]:
```
df_avg_steps = fitbit_wknd_joined.select(['a.Id', 'a.IsWeekend', 'AvgSteps']).toPa
avg_steps_weekend = df_avg_steps[df_avg_steps['IsWeekend']=='Weekend']['AvgSteps']
avg_steps_weekday = df_avg_steps[df_avg_steps['IsWeekend']=='Weekday']['AvgSteps']

result = stats.ttest_rel(avg_steps_weekend, avg_steps_weekday)
pval = float(result.pvalue)
if pval > 0.05:
    print(f"The p-value is of{pval: .3} > 0.05, therefore we do not reject the nul
else:
    print(f"The p-value is of{pval: .3} <= 0.05, therefore we do reject the null hy
```

```
The p-value is of 0.742 > 0.05, therefore we do not reject the null hypothesis.
```

In this case we are unable to reject the null hypothesis, which translates to the population having a similar behaviour regardless of it being a weekday or weekend. This means that for **daily average steps** the day of the week does not play a defining role.

6.2 Daily Average active minutes comparison:

In this case we could expect a similar conclusion as with the activity of the users. Since the calories burnt might be related to how much active an user was, we could expect that if the weekend vs. weekday did not affected on the activity of the users, the same conclusion will be derived for the calories.

In [18]:
```
df_avg_active = fitbit_wknd_joined.select(['a.Id', 'a.IsWeekend', 'AvgActiveMins']
avg_active_weekend = df_avg_active[df_avg_active['IsWeekend']=='Weekend']['AvgActi
avg_active_weekday = df_avg_active[df_avg_active['IsWeekend']=='Weekday']['AvgActi

result = stats.ttest_rel(avg_active_weekend, avg_active_weekday)
pval = float(result.pvalue)
if pval > 0.05:
    print(f"The p-value is of{pval: .3} > 0.05, therefore we do not reject the nul
else:
    print(f"The p-value is of{pval: .3} <= 0.05, therefore we do reject the null hy
```

```
The p-value is of 0.412 > 0.05, therefore we do not reject the null hypothesis.
```

In a similar result, we are unable to reject the null hypothesis. This means that the activity of the users remains constant all throughout the week.

6.3 Daily Average active calories comparison:

Similar to the **daily average steps** we are interested in knowing if how active the users are could be influenced by the weekend factor. One idea behind this reasoning is that since on weekends users usually are not working, we can expect a change in their activities either due to resting more or doing more physical activities.

In [19]:
```
df_avg_calories = fitbit_wknd_joined.select(['a.Id', 'a.IsWeekend', 'AvgCalories']
avg_calories_weekend = df_avg_calories[df_avg_calories['IsWeekend']=='Weekend']['A
avg_calories_weekday = df_avg_calories[df_avg_calories['IsWeekend']=='Weekday']['A

result = stats.ttest_rel(avg_calories_weekend, avg_calories_weekday)
pval = float(result.pvalue)
if pval > 0.05:
    print(f"The p-value is of{pval: .3} > 0.05, therefore we do not reject the nul
else:
    print(f"The p-value is of{pval: .3} <= 0.05, therefore we do reject the null hy
```

```
The p-value is of 0.995 > 0.05, therefore we do not reject the null hypothesis.
```

In this case it is even clearer that the populations do not vary at all regarding to the weekend or weekday factor. An explanation for this is that most of the caloric consumption of a person is due to the body functions, this means, what the body needs just to keep going. Only a lower percentage of the caloric consumption is attributed to activity, so as so we would not expect this number to change drastically in one week.

6.4 Sleep score comparison:

A reason to think that the average sleep minutes vary between weekends and weekdays is due to having more time to sleep. Since people usually do not have to wake up as early in the weekends to go to work, we could suspect to see some changes within this metric.

In [20]:
```
df_avg_sleep = fitbit_wknd_joined.select(['a.Id', 'a.IsWeekend', 'AvgSleepScore'])
avg_sleep_weekend = df_avg_sleep[df_avg_sleep['IsWeekend']=='Weekend']['AvgSleepSc
avg_sleep_weekday = df_avg_sleep[df_avg_sleep['IsWeekend']=='Weekday']['AvgSleepSc

result = stats.ttest_rel(avg_sleep_weekend, avg_sleep_weekday)
pval = float(result.pvalue)
if pval > 0.05:
    print(f"The p-value is of{pval: .3} > 0.05, therefore we do not reject the nul
else:
    print(f"The p-value is of{pval: .3} <= 0.05, therefore we do reject the null hy
```

```
The p-value is of 0.948 > 0.05, therefore we do not reject the null hypothesis.
```

Finally, we are also rejecting the hypothesis that weekend drastically changes the sleep behaviour of the users.

7. Multivariate Analysis

The last analysis we are doing is comparing how each variable relates to each other. As a respondent variable we are going to choose **AvgCalories** with the objective of seeing if some of the users behaviour impacts in a more significant way the total number of calories burnt.

7.1 Correlation Matrix

We are starting with a correlation matrix in order to see the joint effect among the variables.

In [21]:
```
columns_keep = ['AvgCalories', 'AvgSteps', 'AvgSedentaryMins', 'AvgActiveMins', 'A
fitbit_user_corr = fitbit_user_joined.select(columns_keep).toPandas().round(2)
```

```
corr = fitbit_user_corr.corr()
corr.style.set_precision(4).background_gradient(cmap='coolwarm', axis=None)
```

Out[21]:

	AvgCalories	AvgSteps	AvgSedentaryMins	AvgActiveMins	AvgSleepScore
AvgCalories	1.0000	0.3215	-0.0548	0.1571	0.4140
AvgSteps	0.3215	1.0000	-0.5061	0.8058	0.0981
AvgSedentaryMins	-0.0548	-0.5061	1.0000	-0.6699	-0.0599
AvgActiveMins	0.1571	0.8058	-0.6699	1.0000	0.1068
AvgSleepScore	0.4140	0.0981	-0.0599	0.1068	1.0000

From the correlation matrix we can notice that the most relevant variables for **AvgCalories** are **AvgSleepScore** in first place and **AvgSteps** in second place. As a reminder, the **AvgSleepScore** is calculated as the time spent sleeping divided by the total time in bed. Therefore is an indicator of the sleep quality.

It is a surprise that both the **AvgSedentaryMins** and the **AvgActiveMins** are not as impactful for the number of calories spent.

Another insight we can get from the matrix is that the **AvgActiveMins** and the **AvgSteps** are highly correlated. This makes sense given that the more a person walks throughout the day, the more active they will be.

Lastly, there does not appears to be a relation between the **SleepScore** and the other variables (excluding calories). This is interesting given that one would expect people that are more active throughout the day to have a better sleep quality.

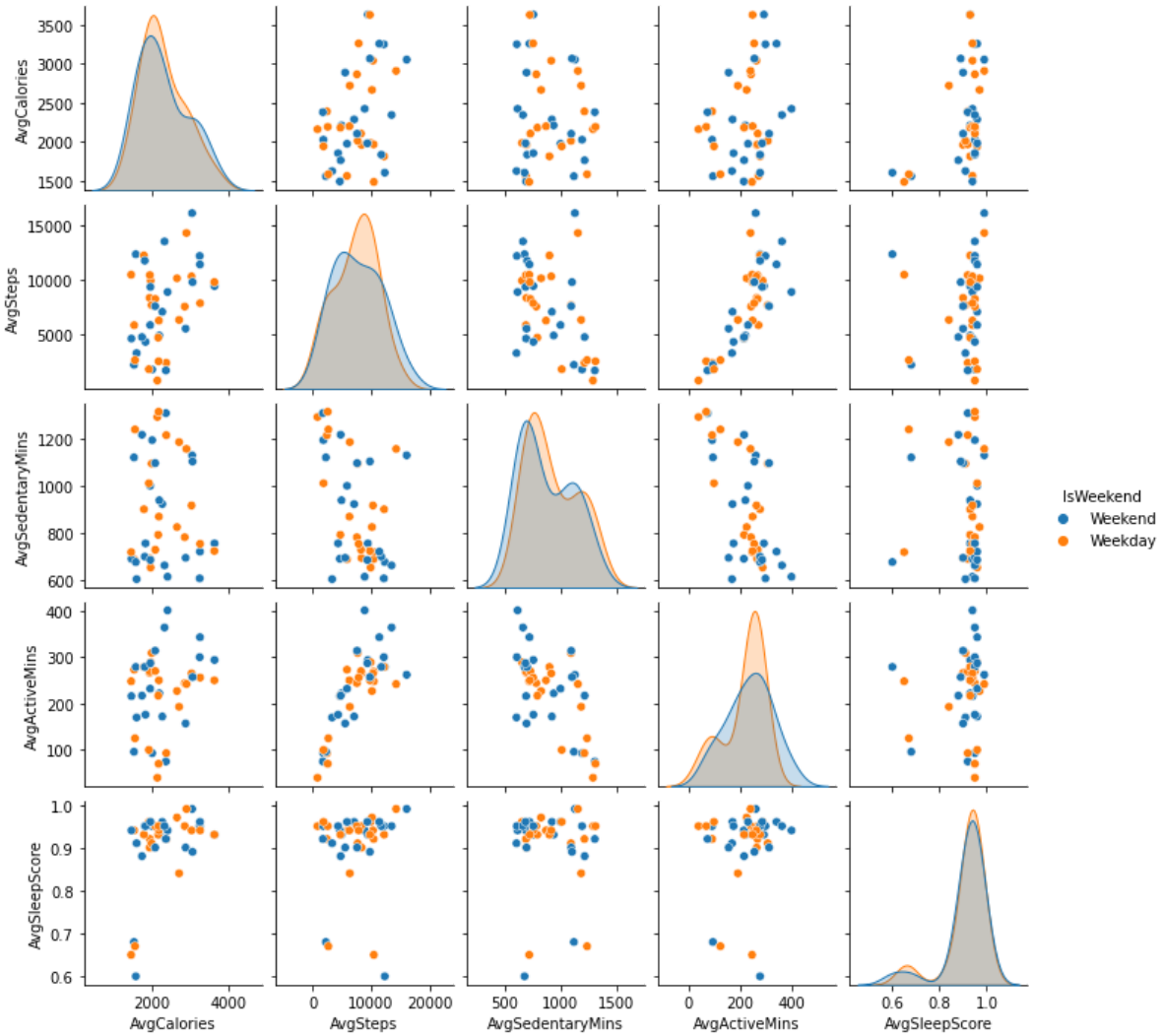
7.2 Scatterplot Matrix:

One last thing we can do with the data is to plot a scatterplot matrix, even though the conclusions from the correlation matrix will not change, this will allow us to better understand the behaviour we are noticing.

As a bonus, the scatterplot matrix plots an histogram in the diagonal, which will also yield extra information about the univariate behaviour of each of the metrics. Since we also count with the **IsWeekend** label, we are able to visually compare both populations at the same time.

In [22]:
```
columns_keep = ['AvgCalories', 'AvgSteps', 'AvgSedentaryMins', 'AvgActiveMins', 'Av
fitbit_splot_matrix = fitbit_wknd_joined.select(columns_keep).toPandas().round(2)

sns.pairplot(fitbit_splot_matrix, hue = 'IsWeekend', height= 2)
```

Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x7f161334b970>
```



As we suspected we can see that there is a marked relationship between the **AvgCalories** and the **AvgSleepScore**. However, since we have presence of an atypical group, this might also yield to a higher linear dependence.

Another thing that can be noticed more clearly is the lack of dependence between **AvgCalories** and both **AvgSedentaryMins** and **AvgActiveMins**. We see what can be called random behaviour.

By adding the density plots for both **weekends** and **wekdays** we can see something interesting. Even though the hypothesis test did not let us conclude a difference in the activity patterns, we can see in the density plots (diagonal) for **AvgActiveMins** that there appears to be a change in behaviour. Perhaps with a larger sample size this could be further tested.

Finally, we see that for the rest of the metrics there appears to be a similar behaviour regarding to the density plots.

8. Conclusions:

From the analysis we performed we can draw the following conclusions:

8.1 Population comparisons:

From the population comparisons we can conclude that there is not enough evidence to conclude that the weekend vs. weekdays is a relevant factor to mark a difference between the behaviour of the metrics.

Even though our assumptions pointed into a different direction, we noticed that the behaviour of the users remains similar across the week. It is important to note that the sample taken is of 24 users, which might not be enough to draw high level conclusions. Another remark that can be done is that, since we our using data from **Fitbit users**, we can expect them to have a specific behaviour. For example, people that buy a Fitbit might be more prone to have an active lifestyle, regardless of the day of the week.

8.2 Correlations with calories:

One key takeaway from this analysis is that we can notice a relation between the quality of sleep and the calories. Another result that was unexpected was the low correlation between the activity metrics and the calories. We expected this number to be higher, given that we usually tend to relate a higher activity to a higher caloric use.