Table of Contents

1. Introduction

Bellabeat is a high-tech company that manufactures health-focused smart products. In 2014, Urška Sršen and Sando Mur founded the company and developed one of the first wearables specifically designed for women. It uses beautifully designed technology to inform and inspire women around the world. Bellabeat Collects data on activity, sleep, stress, and reproductive health, allowing it to empower women with knowledge about their health and habits.

Bellabeat had, in 2016, opened offices around the world and launched multiple products that became available through a growing number of online retailers in addition to their e-commerce channel on their website. Some of Bellabeat's products include:

- **Bellabeat app**: The Bellabeat app provides users with health data related to their activity, sleep, stress, menstrual cycle, and mindfulness habits.
- **Leaf**: Bellabeat's classic wellness tracker can be worn as a bracelet, necklace, or clip. The Leaf Tracker connects to the Bellabeat app to track activity, sleep, and stress.
- **Time**: This wellness watch combines the timeless look of a classic timepiece with smart technology to track user activity, sleep, and stress.
- **Spring**: This is a water bottle that tracks daily water intake using smart technology to ensure that you are appropriately hydrated throughout the day. The Spring bottle connects to the Bellabeat app to track your hydration levels.

As a data analyst working on the marketing team at Bellabeat, I've been asked to gain insight into how consumers use their smartwatches. I'll be presenting my analysis to the executive team along with high-level recommendations for Beallabeat's marketing strategy.

2. Ask

2.1 Business task

Analyze smart device usage data to gain insight into how consumers use non-Bellabeat smart devices. Then, select one Bellabeat product to apply these insights.

Using FitBit Fitness Tracker data available in the public domain through Mobius and hosted on Kaggle, I will analyze data from over 30 FitBit users over a two-month span. This analysis will provide insights into how these users interact with their smartwatches, which can inform Bellabeat's research and marketing strategies

Questions:

1. What are some trends in smart device usage?
2. How could these trends apply to Bellabeat customers?
3. How could these trends help influence Bellabeat marketing strategy?

2.2 Key Stakeholders

It's important to understand the audience to effectively communicate my analysis and share recommendations.

- **Urška Sršen**: Bellabeat's co-founder and Chief Creative Officer.
- **Sando Mur**: Bellabeat's cofounder; key member of the Bellabeat executive team.

- **Bellabeat marketing analytics team**: A team of data analysts responsible for collecting, analyzing, and reporting data that helps guide Bellabeat's marketing strategy.

3. Prepare

3.1 Dataset

The FitBit Fitness Tracker Data is a 3rd party, public dataset that measures smart device users' daily habits. This dataset contains personal fitness data from **35** fitbit users, including physical activity, heart rate, and sleep monitoring.

3.2 Data Source

The data is publically available and maintained through Mobius. It was collected through a survey via Amazon Mechanical Turk between 03.12.2016 and 05.12.2016. Dataset Link

3.3 Data Integrity and Credibility

The dataset comprises responses from only 35 users, which may not accurately reflect the broader population in consideration. Additionally, the absence of information regarding users' locations is noteworthy, as preferences, activities, and habits can vary significantly based on geographical and cultural factors. The age of the users wasn't provided, yet age plays a crucial role in determining users' agility, strength, and overall activity levels.

3.4 Data Selection

I did not consider all 18 datasets provided, as my focus is on the business tasks. The business task requires analyzing smart device usage data to gain insights into consumer behavior. After exploring the data using Excel and Kaggle's data viewer, I have selected the following files:

- mturkfitbit_export_3.12.16-4.11.16/Fitabase Data 3.12.16-4.11.16/dailyActivity_merged.csv
- mturkfitbit_export_4.12.16-5.12.16/Fitabase Data 4.12.16-5.12.16/dailyActivity_merged.csv

The dailyActivity_merged file contains user IDs, dates, step counts, distances, activity level measures, and calories burned. This data was recorded over two months, providing ample information for analysis.

Other data files, such as sleepDay_merged, could be useful. However, this dataset was recorded for only one month and includes less relevant measurements. I believe focusing on

activity data and calories burned will yield more valuable insights.

4. Process

I have chosen **Python** for processing, analysis, and visualization in this project due to the substantial size of some datasets, which contain thousands of rows. Using Kaggle, which supports Jupyter Notebooks, allows me to share this project with a broad audience of like-minded individuals and receive feedback from enthusiasts.

Additionally, Python enables me to seamlessly complete the entire data analysis process—from data cleaning to visualization—within the same environment.

4.1 Loading Packages

In [649…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import datetime as dt
```

4.2 Importing Dataset

In [611…

```
import os # Importing the os module to interact with the operating system
for dirname, _, filenames in os.walk('Kaggle'):  # Walking through the 'Kaggle' dat
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\dailyActivit
y_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\heartrate_se
conds_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\hourlyCalori
es_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\hourlyIntens
ities_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\hourlySteps_
merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\minuteCalori
esNarrow_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\minuteIntens
itiesNarrow_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\minuteMETsNa
rrow_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\minuteSleep_
merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\minuteStepsN
arrow_merged.csv
Kaggle\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\weightLogInf
o_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\dailyActivit
y_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\dailyCalorie
s_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\dailyIntensi
ties_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\dailySteps_m
erged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\heartrate_se
conds_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\hourlyCalori
es_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\hourlyIntens
ities_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\hourlySteps_
merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteCalori
esNarrow_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteCalori
esWide_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteIntens
itiesNarrow_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteIntens
itiesWide_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteMETsNa
rrow_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteSleep_
merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteStepsN
arrow_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\minuteStepsW
ide_merged.csv
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\sleepDay_mer
ged.csv
```

```
Kaggle\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\weightLogInf
o_merged.csv
```

Merging Dataframes:

We will now load the imported data into a dataframe and merge our two CSV files into a single dataset.

While we will further verify data integrity as we progress, checking the shape of our dataframes confirms that the two datasets have been successfully combined.

In [612…
```
df1= pd.read_csv('Kaggle/mturkfitbit_export_3.12.16-4.11.16/Fitabase Data 3.12.16-4

df2 =pd.read_csv('Kaggle/mturkfitbit_export_4.12.16-5.12.16/Fitabase Data 4.12.16-5

df = pd.concat([df1,df2])
```

In [613…
```
df1.shape
```

Out[613… (457, 15)

In [614…
```
df2.shape
```

Out[614… (940, 15)

In [615…
```
df.shape
```

Out[615… (1397, 15)

In [616…
```
df.columns
```

Out[616…
```
Index(['Id', 'ActivityDate', 'TotalSteps', 'TotalDistance', 'TrackerDistance',
       'LoggedActivitiesDistance', 'VeryActiveDistance',
       'ModeratelyActiveDistance', 'LightActiveDistance',
       'SedentaryActiveDistance', 'VeryActiveMinutes', 'FairlyActiveMinutes',
       'LightlyActiveMinutes', 'SedentaryMinutes', 'Calories'],
      dtype='object')
```

In [617…
```
df.head()
```

Out[617…

	Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDis
0	1503960366	3/25/2016	11004	7.11	7.11	
1	1503960366	3/26/2016	17609	11.55	11.55	
2	1503960366	3/27/2016	12736	8.53	8.53	
3	1503960366	3/28/2016	13231	8.93	8.93	
4	1503960366	3/29/2016	12041	7.85	7.85	

In [618…
```
df.describe() # Descriptive statistics of the dataset.
```

Out[618…

	Id	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance
count	1.397000e+03	1397.000000	1397.000000	1397.000000	1397.000000
mean	4.781210e+09	7280.898354	5.219434	5.192219	0.131481
std	2.384293e+09	5214.336113	3.994206	3.980077	0.703683
min	1.503960e+09	0.000000	0.000000	0.000000	0.000000
25%	2.320127e+09	3146.000000	2.170000	2.160000	0.000000
50%	4.445115e+09	6999.000000	4.950000	4.950000	0.000000
75%	6.962181e+09	10544.000000	7.500000	7.480000	0.000000
max	8.877689e+09	36019.000000	28.030001	28.030001	6.727057

4.3 Data Cleaning

df.dtypes shows our data types for each column.

We'll set Id to a string and ActivityDate to the datetime format.

In [619…
```
df.dtypes
```

Out[619…
```
Id                          int64
ActivityDate               object
TotalSteps                  int64
TotalDistance             float64
TrackerDistance           float64
LoggedActivitiesDistance  float64
VeryActiveDistance        float64
ModeratelyActiveDistance  float64
LightActiveDistance       float64
SedentaryActiveDistance   float64
VeryActiveMinutes           int64
FairlyActiveMinutes         int64
LightlyActiveMinutes        int64
SedentaryMinutes            int64
Calories                    int64
dtype: object
```

In [620…
```
df['Id'] = df['Id'].astype(str) # Set Id as a string. Even as a number, this Id won
df['ActivityDate'] = pd.to_datetime(df['ActivityDate'], format='%m/%d/%Y') # Activi
```

In [621…
```
df.dtypes
```

Out[621…
```
Id                                 object
ActivityDate                 datetime64[ns]
TotalSteps                            int64
TotalDistance                       float64
TrackerDistance                     float64
LoggedActivitiesDistance            float64
VeryActiveDistance                  float64
ModeratelyActiveDistance            float64
LightActiveDistance                 float64
SedentaryActiveDistance             float64
VeryActiveMinutes                     int64
FairlyActiveMinutes                   int64
LightlyActiveMinutes                  int64
SedentaryMinutes                      int64
Calories                              int64
dtype: object
```

Checking missing/null values:

We will check for null values and duplicates that could impact the accuracy of our analysis.

In [622…
```
df.isna().sum() # Checking for missing (null/NaN) values in the dataset.
```

Out[622…
```
Id                          0
ActivityDate                0
TotalSteps                  0
TotalDistance               0
TrackerDistance             0
LoggedActivitiesDistance    0
VeryActiveDistance          0
ModeratelyActiveDistance    0
LightActiveDistance         0
SedentaryActiveDistance     0
VeryActiveMinutes           0
FairlyActiveMinutes         0
LightlyActiveMinutes        0
SedentaryMinutes            0
Calories                    0
dtype: int64
```

Checking duplicates:

In [623…
```
df.duplicated().sum() # Checking for duplicate rows in the dataset.
            # Note: This goes by row, not column, so only an exact match would give
```

Out[623…
```
0
```

Renaming column names for better redability:

In [624…
```
df.columns = df.columns.str.lower() # Converting all column names to lower case.

df.rename(columns =
          {'activitydate':'activity_date', 'totalsteps':'total_steps', 'totaldistan
          'loggedactivitiesdistance':'logged_activity_distance', 'veryactivedistanc
```

```
            'moderatelyactivedistance':'moderately_active_distance', 'lightactivedist
            'sedentaryactivedistance':'sedentary_active_distance', 'veryactiveminutes
            'lightlyactiveminutes':'lightly_active_minutes', 'sedentaryminutes':'sede
            inplace=True)
```

Let's add a few columns to facilitate proper analysis:

total_active_minutes shows the combined active time, regardless of level of activity.

day_of_week shows what day of the week activity took place

no_day shows the numeric equivalent of **day_of_week**

In [625…
```
df['total_active_minutes'] = df['very_active_minutes'] + df['fairly_active_minutes'

df['day_of_week'] = df['activity_date'].dt.day_name() # Adding a column for the day
```

Total no. of Participants:

In [626…
```
print(df['id'].nunique())
```

```
35
```

In [627…
```
print(df['activity_date'].dt.date.nunique())
```

```
62
```

The dataset consists of 35 participants over a span of 62 days

4.4 Sorting and Filtering

Let's subset our data to create a smaller, more focused dataframe containing only the columns relevant to our analysis.

In [628…
```
df_subset = df[['id', 'activity_date', 'total_steps','total_distance', 'calories',#
'very_active_minutes', 'fairly_active_minutes', 'lightly_active_minutes', 'sedentar

df_subset.head()
```

Out[628…

	id	activity_date	total_steps	total_distance	calories	very_active_minutes	fairl
0	1503960366	2016-03-25	11004	7.11	1819	33	
1	1503960366	2016-03-26	17609	11.55	2154	89	
2	1503960366	2016-03-27	12736	8.53	1944	56	
3	1503960366	2016-03-28	13231	8.93	1932	39	
4	1503960366	2016-03-29	12041	7.85	1886	28	

Categorizing Activity Levels Based on Total Steps:

To better measure a user's average daily activity, we will group the data by user ID. This allows us to categorize users into low, medium, and high activity levels based on their daily averages for step count, distance traveled, and calories burned.

These categories help structure the data and add more depth to our visualizations.

- **Low Stepper**: If average steps are less than 6000
- **Medium Stepper**: If average steps are between 6000 and 12000
- **High Stepper**: If average steps are more than 12000

In [629…

```
# Grouping the activity by the id and finding the mean of the total steps
id_avg_step = df_subset.groupby('id')['total_steps'].mean().round(2).sort_values(as
# Note:  In pandas, when you use groupby() along with an aggregation function (like
# .reset_index() converts the Series into a DataFrame with columns 'id' and 'total_

#Renamming the columns
id_avg_step.columns=['id','avg_steps']

id_avg_step.head()
```

Out[629…

	id	avg_steps
0	8877689391	16424.33
1	8053475328	14784.52
2	1503960366	11935.78
3	7007744171	11619.29
4	2022484408	11595.09

In [630…

```
# Define conditions for step level classification
conditions = [
    (id_avg_step['avg_steps'] < 6000),
    (id_avg_step['avg_steps'] >= 6000) & (id_avg_step['avg_steps'] <= 12000),
    (id_avg_step['avg_steps'] > 12000)
]
# Activity levels
values = ['low_stepper', 'mid_stepper', 'high_stepper']
id_avg_step['step_level'] = np.select(conditions, values)
id_avg_step
```

Out[630…

	id	avg_steps	step_level
0	8877689391	16424.33	high_stepper
1	8053475328	14784.52	high_stepper
2	1503960366	11935.78	mid_stepper
3	7007744171	11619.29	mid_stepper
4	2022484408	11595.09	mid_stepper
5	6962181067	10679.89	mid_stepper
6	3977333714	10321.52	mid_stepper
7	2347167796	9647.12	mid_stepper
8	4388161847	8595.69	mid_stepper
9	8378563200	8555.16	mid_stepper
10	5553957443	8540.63	mid_stepper
11	7086361926	8459.81	mid_stepper
12	5577150313	8385.93	mid_stepper
13	4702921684	8367.07	mid_stepper
14	1644430081	7780.92	mid_stepper
15	4319703577	7422.81	mid_stepper
16	6117666160	7363.00	mid_stepper
17	2873212765	7299.26	mid_stepper
18	4558609924	7154.93	mid_stepper
19	3372868164	6616.93	mid_stepper
20	8583815059	6346.62	mid_stepper
21	1624580081	5167.20	low_stepper
22	2026352035	4960.14	low_stepper
23	8253242879	4898.06	low_stepper
24	4445114986	4632.37	low_stepper
25	6290855005	4615.85	low_stepper
26	2320127002	4276.37	low_stepper
27	4020332650	4049.76	low_stepper
28	6775888955	3301.23	low_stepper
29	1844505072	2876.02	low_stepper

	id	avg_steps	step_level
30	8792009665	2217.00	low_stepper
31	4057192912	2103.97	low_stepper
32	6391747486	1336.89	low_stepper
33	1927972279	1269.07	low_stepper
34	2891001357	773.62	low_stepper

In [631…

```
id_avg_calorie = df_subset.groupby('id')['calories'].mean().round(2).sort_values(as
id_avg_calorie.columns=['id','avg_calories']
id_avg_calorie.describe()
```

Out[631…

	avg_calories
count	35.000000
mean	2249.179429
std	554.298030
min	1433.780000
25%	1855.245000
50%	2160.630000
75%	2616.185000
max	3428.880000

In [632…

```
conditions = [
  (id_avg_calorie['avg_calories'] <=2000),
  (id_avg_calorie['avg_calories'] > 2000) & (id_avg_calorie['avg_calories'] <= 3000
  (id_avg_calorie['avg_calories'] > 3000)
]
values = ['low_calorie', 'mid_calorie', 'high_calorie']
id_avg_calorie['calorie_level'] =np.select(conditions, values)
id_avg_calorie
```

Out[632...

	id	avg_calories	calorie_level
0	8877689391	3428.88	high_calorie
1	8378563200	3414.14	high_calorie
2	5577150313	3343.71	high_calorie
3	8053475328	2932.02	mid_calorie
4	4702921684	2918.57	mid_calorie
5	1644430081	2837.58	mid_calorie
6	4388161847	2829.54	mid_calorie
7	4020332650	2736.06	mid_calorie
8	8583815059	2662.13	mid_calorie
9	7007744171	2570.24	mid_calorie
10	2022484408	2500.30	mid_calorie
11	6290855005	2488.33	mid_calorie
12	7086361926	2457.70	mid_calorie
13	6775888955	2284.26	mid_calorie
14	2891001357	2273.38	mid_calorie
15	6117666160	2218.55	mid_calorie
16	1927972279	2195.47	mid_calorie
17	4445114986	2160.63	mid_calorie
18	2347167796	2033.39	mid_calorie
19	4319703577	2025.56	mid_calorie
20	6962181067	2015.38	mid_calorie
21	8792009665	1994.90	low_calorie
22	4558609924	1976.58	low_calorie
23	4057192912	1911.33	low_calorie
24	3372868164	1908.83	low_calorie
25	5553957443	1855.26	low_calorie
26	2873212765	1855.23	low_calorie
27	1503960366	1808.74	low_calorie
28	6391747486	1763.11	low_calorie
29	2320127002	1670.56	low_calorie

	id	avg_calories	calorie_level
30	8253242879	1662.19	low_calorie
31	1844505072	1585.33	low_calorie
32	2026352035	1488.98	low_calorie
33	3977333714	1480.64	low_calorie
34	1624580081	1433.78	low_calorie

In [633…

```
id_avg_distance = df_subset.groupby('id')['total_distance'].mean().round(2).sort_va
id_avg_distance.columns=['id','avg_distance']
id_avg_distance.head()
```

Out[633…

	id	avg_distance
0	8877689391	13.46
1	8053475328	11.50
2	2022484408	8.28
3	7007744171	8.28
4	1503960366	7.73

In [634…

```
conditions = [
  (id_avg_distance['avg_distance'] <= 3),
  (id_avg_distance['avg_distance'] > 3) & (id_avg_distance['avg_distance'] <= 7),
  (id_avg_distance['avg_distance'] > 7)
]
values = ['low_distance', 'mid_distance', 'high_distance']
id_avg_distance['distance_level'] = np.select(conditions, values)
id_avg_distance
```

Out[634...

	id	avg_distance	distance_level
0	8877689391	13.46	high_distance
1	8053475328	11.50	high_distance
2	2022484408	8.28	high_distance
3	7007744171	8.28	high_distance
4	1503960366	7.73	high_distance
5	6962181067	7.23	high_distance
6	3977333714	7.03	high_distance
7	4702921684	6.79	mid_distance
8	8378563200	6.78	mid_distance
9	4388161847	6.67	mid_distance
10	2347167796	6.43	mid_distance
11	5577150313	6.28	mid_distance
12	7086361926	5.75	mid_distance
13	1644430081	5.66	mid_distance
14	5553957443	5.59	mid_distance
15	6117666160	5.58	mid_distance
16	4319703577	4.99	mid_distance
17	8583815059	4.95	mid_distance
18	2873212765	4.93	mid_distance
19	4558609924	4.73	mid_distance
20	3372868164	4.54	mid_distance
21	8253242879	3.51	mid_distance
22	6290855005	3.49	mid_distance
23	1624580081	3.47	mid_distance
24	4445114986	3.14	mid_distance
25	2026352035	3.08	mid_distance
26	4020332650	2.90	low_distance
27	2320127002	2.89	low_distance
28	6775888955	2.37	low_distance
29	1844505072	1.90	low_distance

	id	avg_distance	distance_level
30	4057192912	1.55	low_distance
31	8792009665	1.42	low_distance
32	6391747486	1.07	low_distance
33	1927972279	0.88	low_distance
34	2891001357	0.60	low_distance

In [635…

```
df_subset = df_subset.copy() # Creating a copy of the dataframe to avoid SettingWit
df_subset.loc[:, 'step_level'] = df_subset['id'].map(id_avg_step.set_index('id')['s
#df_subset.loc[:, 'avg_steps'] = df_subset['id'].map(id_avg_step.set_index('id')['a
df_subset.loc[:, 'calorie_level'] = df_subset['id'].map(id_avg_calorie.set_index('i
df_subset.loc[:, 'distance_level'] = df_subset['id'].map(id_avg_distance.set_index(

df_subset.head()
```

Out[635…

	id	activity_date	total_steps	total_distance	calories	very_active_minutes	fairl
0	1503960366	2016-03-25	11004	7.11	1819	33	
1	1503960366	2016-03-26	17609	11.55	2154	89	
2	1503960366	2016-03-27	12736	8.53	1944	56	
3	1503960366	2016-03-28	13231	8.93	1932	39	
4	1503960366	2016-03-29	12041	7.85	1886	28	

In [636…

```
# Group by 'id' and get unique values for step_level, calorie_level, and distance_l
df_result = df_subset.groupby('id').agg({  # Using the lambda function to get uniqu
    'step_level': lambda x: list(x.unique()),
    'calorie_level': lambda x: list(x.unique()),
    'distance_level': lambda x: list(x.unique())
}).reset_index()

df_result
```

Out[636...

	id	step_level	calorie_level	distance_level
0	1503960366	[mid_stepper]	[low_calorie]	[high_distance]
1	1624580081	[low_stepper]	[low_calorie]	[mid_distance]
2	1644430081	[mid_stepper]	[mid_calorie]	[mid_distance]
3	1844505072	[low_stepper]	[low_calorie]	[low_distance]
4	1927972279	[low_stepper]	[mid_calorie]	[low_distance]
5	2022484408	[mid_stepper]	[mid_calorie]	[high_distance]
6	2026352035	[low_stepper]	[low_calorie]	[mid_distance]
7	2320127002	[low_stepper]	[low_calorie]	[low_distance]
8	2347167796	[mid_stepper]	[mid_calorie]	[mid_distance]
9	2873212765	[mid_stepper]	[low_calorie]	[mid_distance]
10	2891001357	[low_stepper]	[mid_calorie]	[low_distance]
11	3372868164	[mid_stepper]	[low_calorie]	[mid_distance]
12	3977333714	[mid_stepper]	[low_calorie]	[high_distance]
13	4020332650	[low_stepper]	[mid_calorie]	[low_distance]
14	4057192912	[low_stepper]	[low_calorie]	[low_distance]
15	4319703577	[mid_stepper]	[mid_calorie]	[mid_distance]
16	4388161847	[mid_stepper]	[mid_calorie]	[mid_distance]
17	4445114986	[low_stepper]	[mid_calorie]	[mid_distance]
18	4558609924	[mid_stepper]	[low_calorie]	[mid_distance]
19	4702921684	[mid_stepper]	[mid_calorie]	[mid_distance]
20	5553957443	[mid_stepper]	[low_calorie]	[mid_distance]
21	5577150313	[mid_stepper]	[high_calorie]	[mid_distance]
22	6117666160	[mid_stepper]	[mid_calorie]	[mid_distance]
23	6290855005	[low_stepper]	[mid_calorie]	[mid_distance]
24	6391747486	[low_stepper]	[low_calorie]	[low_distance]
25	6775888955	[low_stepper]	[mid_calorie]	[low_distance]
26	6962181067	[mid_stepper]	[mid_calorie]	[high_distance]
27	7007744171	[mid_stepper]	[mid_calorie]	[high_distance]
28	7086361926	[mid_stepper]	[mid_calorie]	[mid_distance]
29	8053475328	[high_stepper]	[mid_calorie]	[high_distance]

	id	step_level	calorie_level	distance_level
30	8253242879	[low_stepper]	[low_calorie]	[mid_distance]
31	8378563200	[mid_stepper]	[high_calorie]	[mid_distance]
32	8583815059	[mid_stepper]	[mid_calorie]	[mid_distance]
33	8792009665	[low_stepper]	[low_calorie]	[low_distance]
34	8877689391	[high_stepper]	[high_calorie]	[high_distance]

5. Analyze

Our data has been prepared and processed for analysis.

We can now view our organized data, perform calculations, and identify trends and relationships.

In [637…
```
df_subset.describe().style.background_gradient() # Descriptive statistics of the da
```

Out[637…

	activity_date	total_steps	total_distance	calories	very_active_minutes	f
count	1397	1397.000000	1397.000000	1397.000000	1397.000000	
mean	2016-04-19 01:26:35.132426496	7280.898354	5.219434	2266.265569	19.679313	
min	2016-03-12 00:00:00	0.000000	0.000000	0.000000	0.000000	
25%	2016-04-09 00:00:00	3146.000000	2.170000	1799.000000	0.000000	
50%	2016-04-19 00:00:00	6999.000000	4.950000	2114.000000	2.000000	
75%	2016-04-30 00:00:00	10544.000000	7.500000	2770.000000	30.000000	
max	2016-05-12 00:00:00	36019.000000	28.030001	4900.000000	210.000000	
std	nan	5214.336113	3.994206	753.005527	31.675878	

In [638…
```
id_avg_step['step_level'].value_counts()
```

Out[638…
```
step_level
mid_stepper     19
low_stepper     14
high_stepper     2
Name: count, dtype: int64
```

In [639…
```
id_avg_calorie['calorie_level'].value_counts()
```

Out[639…
```
calorie_level
mid_calorie     18
low_calorie     14
high_calorie     3
Name: count, dtype: int64
```

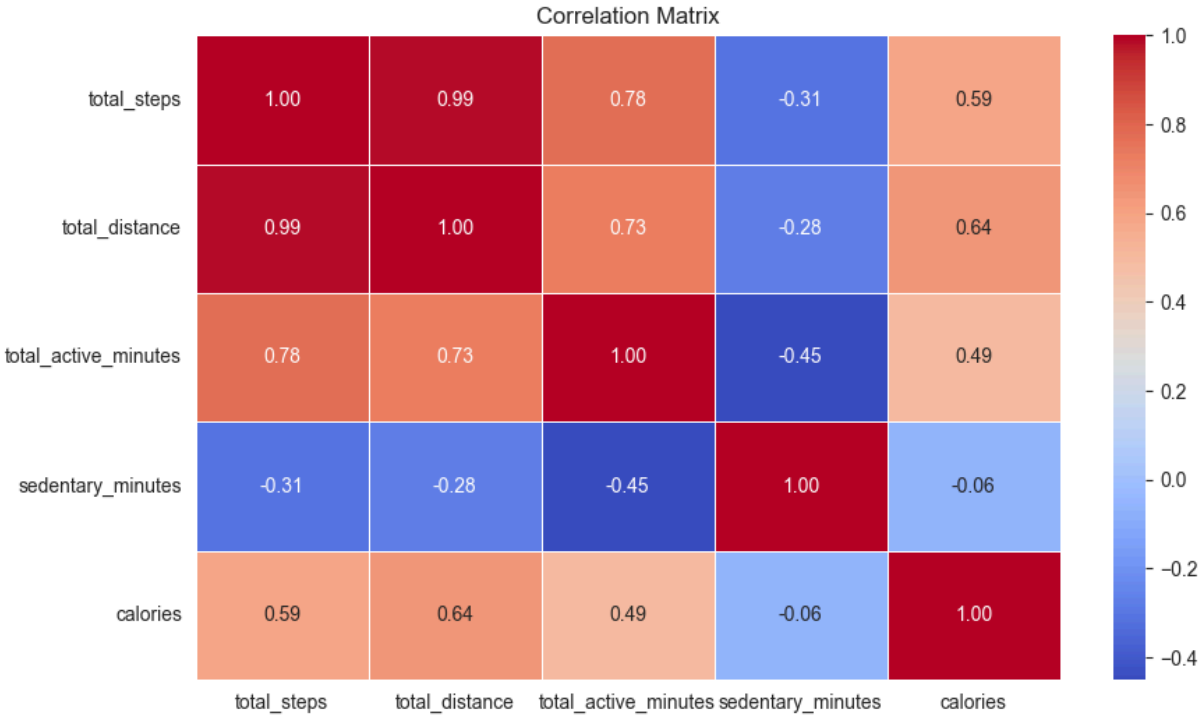
In [640…
```
id_avg_distance['distance_level'].value_counts()
```

Out[640…
```
distance_level
mid_distance     19
low_distance      9
high_distance     7
Name: count, dtype: int64
```

In [641…
```
day_grp = df_subset.groupby('day_of_week') # Grouping the data by 'day_of_week' to
avg_daily_steps = day_grp['total_steps'].mean().round(2).sort_values(ascending=Fals
avg_daily_steps.head(7)
```

Out[641…
```
day_of_week
Saturday     7752.27
Wednesday    7547.58
Monday       7541.32
Thursday     7268.30
Friday       7187.53
Tuesday      7083.51
Sunday       6606.73
Name: total_steps, dtype: float64
```

In [642…
```
correlation = ["total_steps", "total_distance", "total_active_minutes", "sedentary_

fig, ax = plt.subplots(figsize=(10, 6))
ax = sb.heatmap(df_subset[correlation].corr(),
            annot = True,
            fmt = ".2f",
            linewidths=0.5,
           cmap="coolwarm")

ax.set_title('Correlation Matrix');
```



Key Insights

- On average, users walked 7,280 steps per day and burned 2,266 calories.
- The highest average step count was recorded on Saturdays (7,752 steps), while Sundays had the lowest average at 6,606 steps.
- A small subset of users demonstrated exceptional activity levels:
  - 2 users averaged over 12,000 steps per day.
  - 7 users walked more than 7 miles per day.
  - 3 users burned over 3,000 calories per day.
- The majority of users spent a significant portion of their day sedentary:
  - 992 minutes per day were spent inactive, compared to 218 minutes of active time on average.
- A strong correlation was observed between steps taken and distance traveled, which aligns with expectations.
- Factors influencing calorie expenditure:
  - Total active minutes and total steps had a positive correlation with calories burned.
  - Total distance traveled exhibited the strongest correlation with calorie expenditure.
- While sedentary time had a negative relationship with steps, distance, and active minutes, it showed almost no correlation with calories burned.

Maximizing total distance traveled and steps taken is the most effective way to burn calories, whereas the amount of sedentary time has little to no impact on calorie expenditure.

6. Share

Our analysis is now complete, and we will develop visualizations to effectively present our findings. Well-designed visuals enhance accessibility and emphasize key insights.

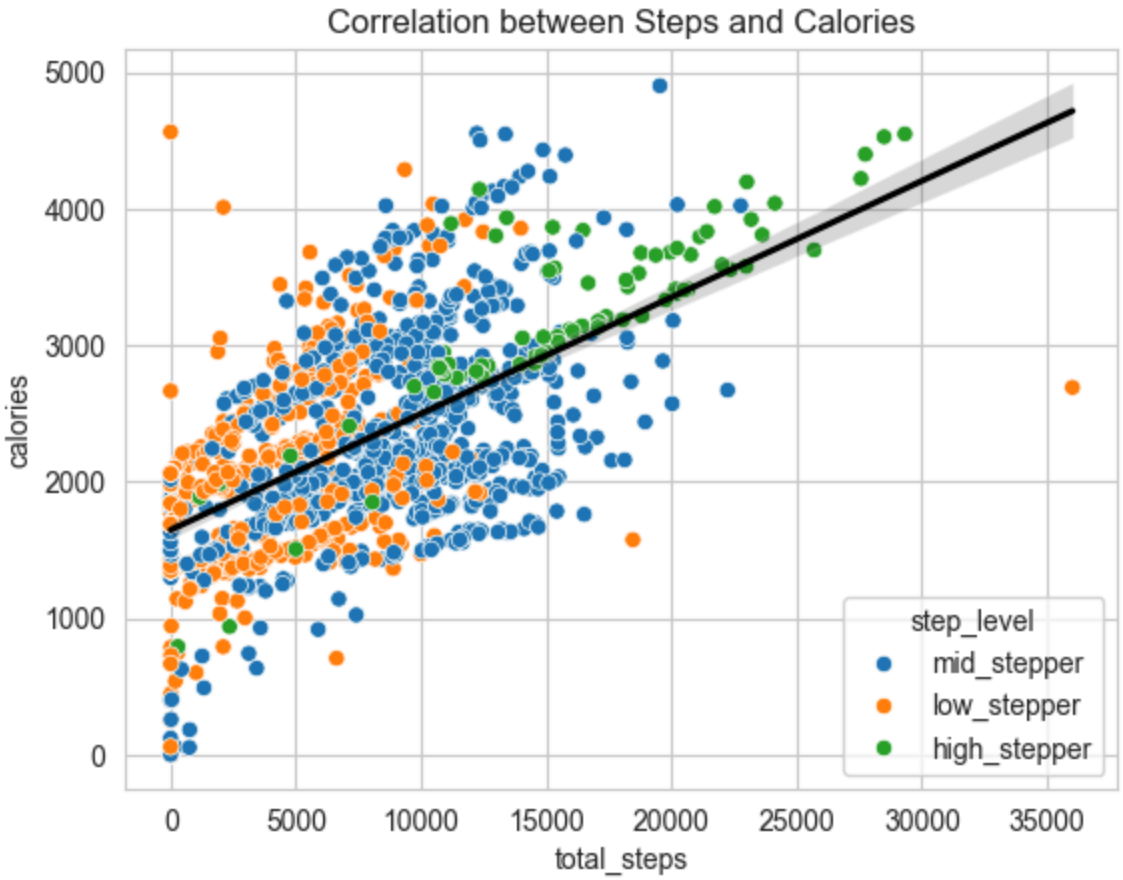
To illustrate the relationship between activity levels and calories burned, we have created three scatterplots:

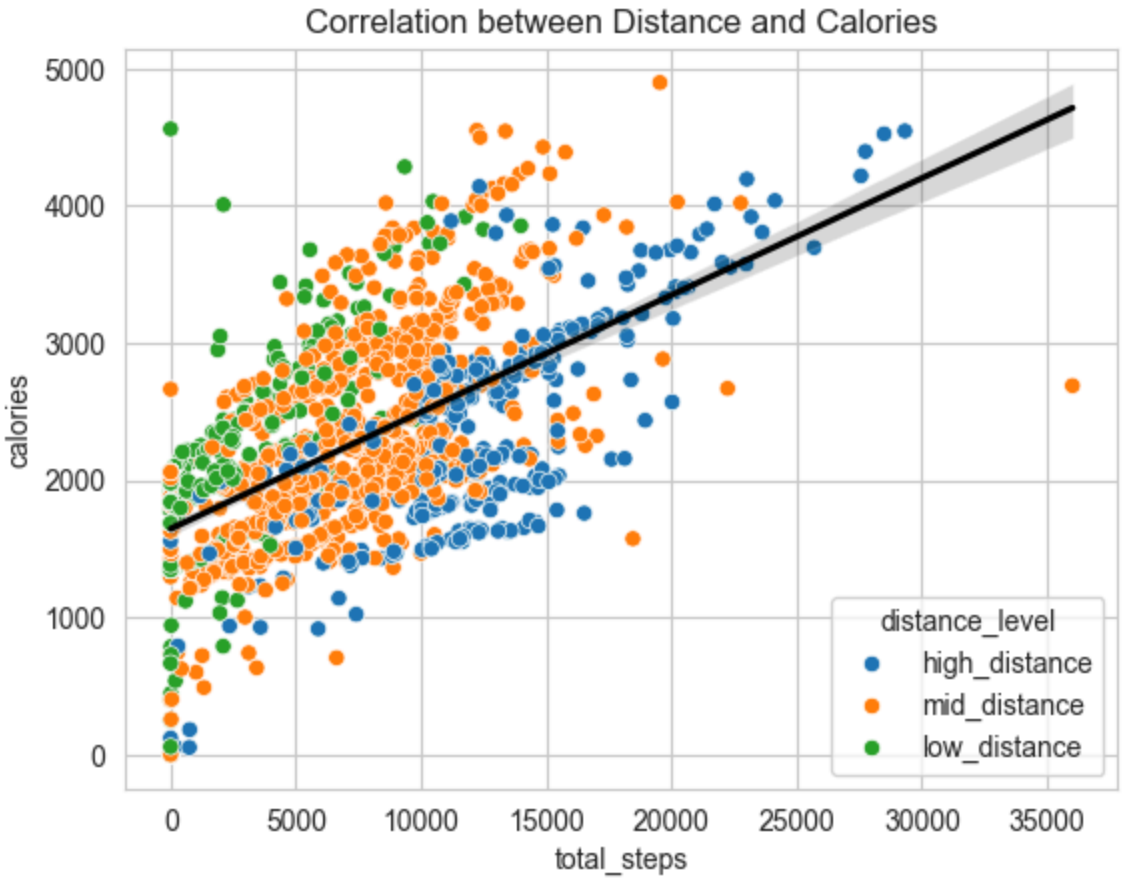
- Steps taken and calories burned show a strong positive correlation.
- Distance traveled and calories burned display an even stronger correlation.
- Users are color-coded into low, mid, and high activity levels for better visualization.

Notably, users with higher step counts and greater distances traveled appear in the upper right quadrant, reinforcing the key takeaway: increasing steps and distance is the most effective way to maximize calorie burn.

In [643…

```
ax = sb.scatterplot(data=df_subset, x='total_steps', y='calories', hue =df_subset['
sb.regplot(x='total_steps', y='calories', data=df_subset, scatter=False, ax=ax, col
plt.title('Correlation between Steps and Calories')
plt.show()

ax = sb.scatterplot(data=df_subset, x='total_steps', y='calories', hue =df_subset['
sb.regplot(x='total_steps', y='calories', data=df_subset, scatter=False, ax=ax, col
plt.title('Correlation between Distance and Calories')
plt.show()
```



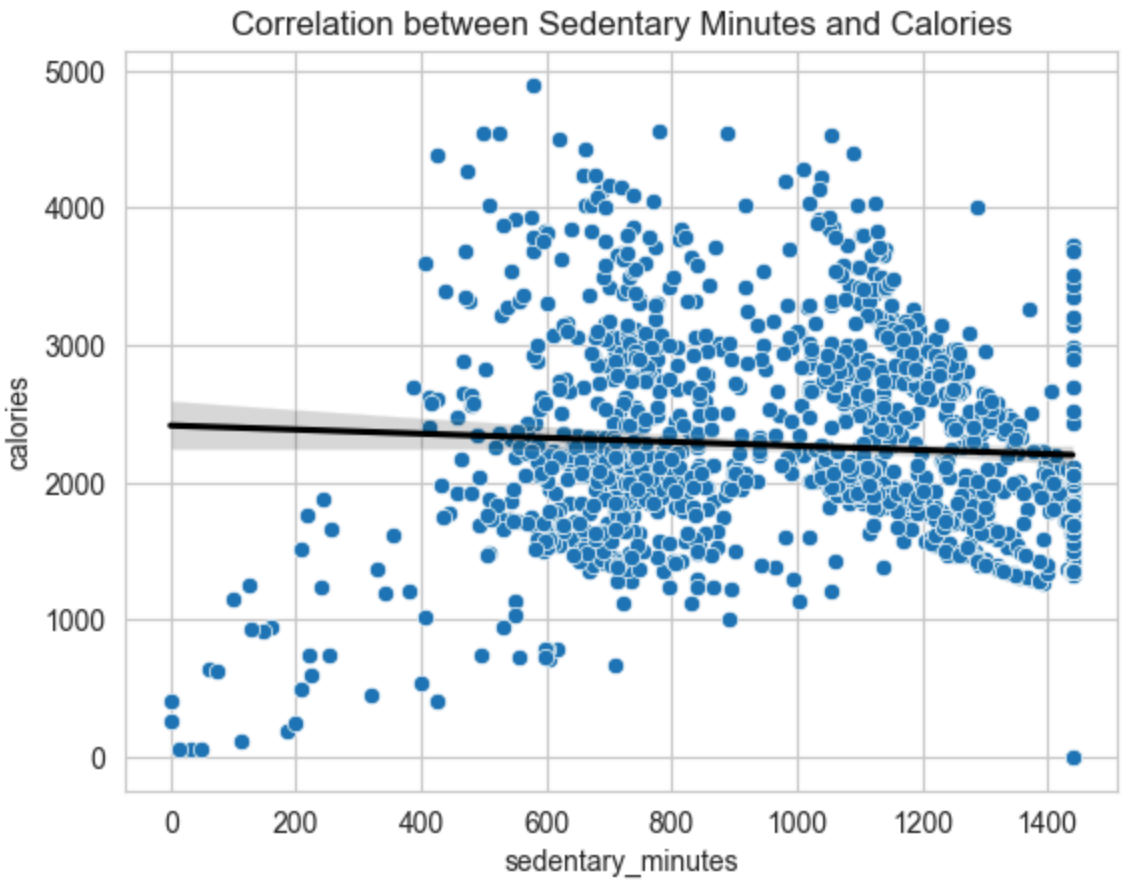


The correlation between sedentary time and calories burned is slightly negative, but close to zero, as shown in our final scatterplot.

While remaining inactive all day won't contribute to calorie burn, the data suggests that sedentary time has no significant impact on overall caloric output. Many users in the lower left quadrant, who spent less time sedentary, still burned fewer calories than those in the upper right quadrant, who were inactive for over 1,200 minutes. This reinforces the idea that steps taken and distance traveled are the key drivers of calorie expenditure, while time spent sedentary has minimal influence.

In [644…

```
ax = sb.scatterplot(data=df_subset, x='sedentary_minutes', y='calories')
sb.regplot(x='sedentary_minutes', y='calories', data=df_subset, scatter=False, ax=a
plt.title('Correlation between Sedentary Minutes and Calories')
plt.show()
```



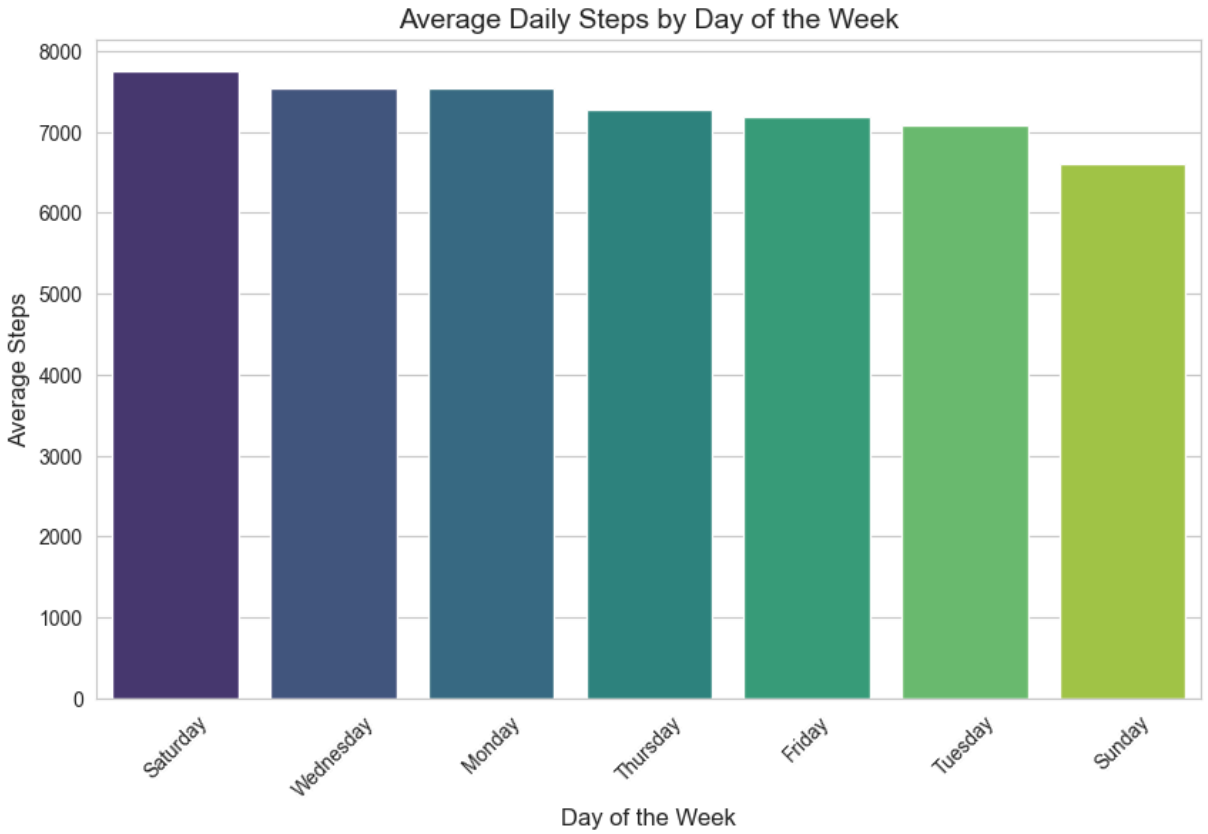
Users' activity levels vary throughout the week, and visualizing these trends with a bar graph helps illustrate average daily step counts and time spent inactive.

While weekday activity remains relatively consistent, Mondays and Wednesdays see slightly higher step counts compared to Tuesdays. However, the weekends show the most noticeable contrast—Saturdays have the highest step counts, while Sundays see a significant drop in activity.

In [645…

```
# Set figure size and style
plt.figure(figsize=(10, 6))
sb.set_style("whitegrid")

# Convert avg_daily_steps to a DataFrame for compatibility
df_avg_steps = avg_daily_steps.reset_index()
df_avg_steps.columns = ['day_of_week', 'avg_steps']

# Plot the bar chart with hue set to 'day_of_week'
sb.barplot(data=df_avg_steps, x='day_of_week', y='avg_steps', hue='day_of_week', pa

# Customize labels and title
plt.xlabel("Day of the Week", fontsize=12)
plt.ylabel("Average Steps", fontsize=12)
plt.title("Average Daily Steps by Day of the Week", fontsize=14)

plt.xticks(rotation=45) # Rotate x-axis labels for better readability
```

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



In [646…

```
avg_sedentary_minutes = df_subset.groupby('day_of_week')['sedentary_minutes'].mean(

# Convert avg_sedentary_minutes to a DataFrame for compatibility
df_avg_sedentary_minutes = avg_sedentary_minutes.reset_index()
df_avg_sedentary_minutes.columns = ['day_of_week', 'avg_sedentary_minutes']

df_avg_sedentary_minutes.head(7)
```

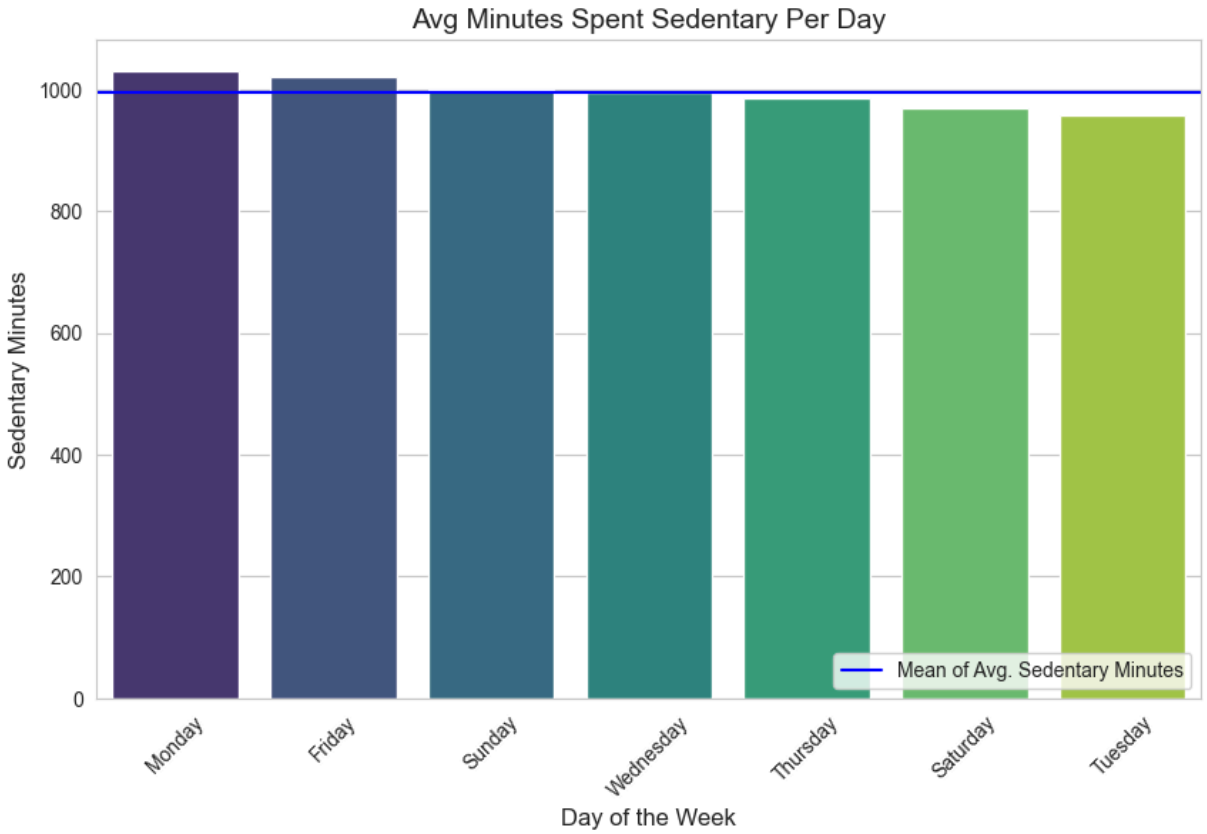
Out[646…

	day_of_week	avg_sedentary_minutes
0	Monday	1029.72
1	Friday	1020.27
2	Sunday	999.73
3	Wednesday	994.71
4	Thursday	984.90
5	Saturday	968.93
6	Tuesday	956.39

In [647…

```
# Set figure size and style
plt.figure(figsize=(10, 6))
sb.set_style("whitegrid")
```

```
# Plot the bar chart with hue set to 'day_of_week'
sb.barplot(data=df_avg_sedentary_minutes, x='day_of_week', y='avg_sedentary_minutes
plt.axhline(y=avg_sedentary_minutes.mean(), color='blue',label='Mean of Avg. Sedent

# Customize labels and title
plt.xlabel("Day of the Week", fontsize=12)
plt.ylabel("Sedentary Minutes", fontsize=12)
plt.title("Avg Minutes Spent Sedentary Per Day", fontsize=14)
plt.legend(loc = "lower right")
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
# Show the plot
plt.show()
```



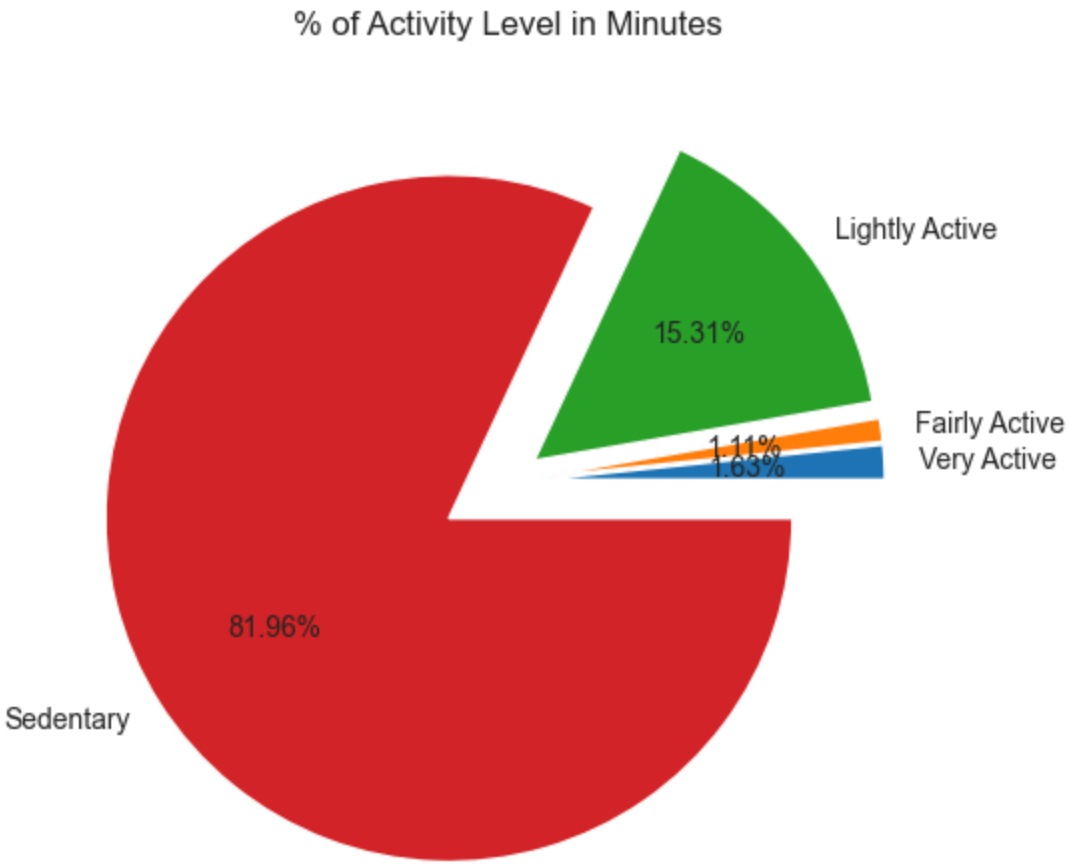
It is unsurprising that Saturdays, the day with the highest average step count, also had lower-than-average sedentary time.

However, a surprising insight is that Tuesdays had the lowest sedentary time, despite users taking fewer-than-average steps on that day. While there is a negative relationship between steps, distance, and sedentary time, the correlation is not strong. This further reinforces the finding that periods of inactivity do not significantly impact overall activity levels or calorie expenditure.

In [648…

```
very_active_mins = df_subset['very_active_minutes'].sum()
fairly_active_mins = df_subset['fairly_active_minutes'].sum()
lightly_active_mins = df_subset['lightly_active_minutes'].sum()
sedentary_mins = df_subset['sedentary_minutes'].sum()

slices = [very_active_mins, fairly_active_mins, lightly_active_mins, sedentary_mins
```

```
labels = ['Very Active', 'Fairly Active', 'Lightly Active', 'Sedentary']
explode = [0.1,0.1,0.1,0.2]

plt.pie(slices, labels=labels, explode = explode, autopct = '%1.2f%%')
plt.title('% of Activity Level in Minutes')
plt.tight_layout()
```



The weak correlation between sedentary time and overall activity levels may be due to the fact that all users spent a significant portion of their day inactive.

This pie chart illustrates that 82% of recorded minutes were spent sedentary, while only 1.1% of minutes were classified as fairly active and 1.6% as very active. Despite the high percentage of inactivity, it did not hinder users from achieving their step goals or burning calories, suggesting that even brief periods of movement can contribute to overall activity levels.

7. Act

Questions

1. What are some trends in smart device usage?

- Users spend the majority of their time inactive, with only 18% of their logged minutes being active.
- Activity levels fluctuate throughout the week, with Saturdays being the most active and Sundays the least active on average.
- The most effective way to maximize calorie burn is by increasing steps taken and distance traveled.

---

2. How could these trends apply to Bellabeat customers?

- While the dataset comes from Fitbit users, the insights can be applied to Bellabeat customers to optimize their fitness experience.
- Encouraging users to set step goals or maintain a minimum daily distance can help improve calorie expenditure.
- Designing fitness plans that leverage high-activity Saturdays and accommodate lower-activity Sundays can align with natural user behavior, improving engagement and retention.

---

3. How could these trends help influence Bellabeat marketing strategy?

- Targeting active users who value step and distance tracking can enhance advertising effectiveness.
  - This can be achieved by placing ads on fitness-oriented websites such as SELF.com and WomensHealthMag.com or featuring Bellabeat products in popular gyms like Planet Fitness and LA Fitness.
  - A more impactful approach could include sponsoring fitness events, such as running groups or races, to actively engage with the fitness community.
- Given that users spend a significant amount of time inactive, marketing campaigns should also highlight Bellabeat's features beyond activity tracking, such as sleep monitoring, heart rate tracking, and stylish designs, to align with a holistic health and wellness lifestyle.

Recommendation

The key to boosting retention and attracting new users lies in customizing the Bellabeat experience to meet diverse user needs. By personalizing fitness plans and emphasizing them as a core feature, Bellabeat can appeal to users from all backgrounds.

- For users focused on weight loss, fitness plans should emphasize step-counting and distance-tracking, while also promoting the importance of rest and recovery. Our analysis indicates that sedentary time does not directly impact calorie burn, so balancing activity and rest can prevent burnout and improve user retention.

- For high-performing athletes, introducing competitive features like regional leaderboards could enhance engagement. These leaderboards could rank users based on fastest mile times or highest step counts during the week, encouraging them to stay active throughout weekdays when activity levels typically drop. Heart rate and sleep tracking can also be integrated to help users optimize performance and recovery.

- While online and physical advertisements remain valuable, interactive community engagement can leave a stronger and more lasting impact.

    - Hosting Bellabeat-sponsored events, such as walkathons or running races, can promote both healthy activity and brand visibility.
    - Organizing weekly running clubs through the Bellabeat app can encourage users to wear their devices more often, while also attracting new users to join.
    - Since Saturdays are the most active days, capitalizing on this trend by hosting weekend fitness events could further enhance user participation and brand recognition.

By aligning Bellabeat's marketing and product strategy with these insights, the company can strengthen user engagement, improve retention, and establish itself as a leader in the smart wellness device market.