Bellabeat Case Study

Sawandi Kirby

2024-01-17

Contents

1 BellaBeat Background **4**

1.1 About the Company 4

2 Ask **4**

2.1 Details 4

2.2 Questions: 4

3 Prepare **5**

3.1 Observe Dataset 5

4 Get environment ready. **5**

4.1 Load Packages 5

4.2 Import dataset. 6

5 Process **7**

5.1 Clean and Format Data 7
5.2 Sleep Data 8
5.3 Steps Data 13
5.4 Joins 14
5.5 Remove Unused Data 15
5.6 Date Columns 16
5.7 Days of the Week 17

6 Analyze **18**

6.1 Look for patterns 18
6.2 Averages 18
6.3 Categorize Users by average daily steps. 21
6.4 Percentages 22
6.5 Total Amounts 23
6.6 Categorize 25
6.7 Percent of minutes asleep 27
6.8 Weight Log 28
6.9 Frequency 30

7 Share **31**

7.1 Percent of User Frequency over Time 31
7.2 Frequency of Step Records 33
7.3 Amount of Steps by Day and Average Steps by Day 34
7.4 Percent of Average Steps by Day 35
7.5 Percent of Users by Step Activity 36
7.6 Activity & Step Distribution per Weekday 37
7.7 Percent of Activity 38
7.8 Daily Active Minutes 39
7.9 Active Minutes 41
7.10 Device Usage 42
7.11 Total Steps vs. Calories 43
7.12 Distance vs. Calories 44

7.13 Steps vs. Calories per Hour 45
7.14 Steps, Intensity and METS vs. Calories per Min 47
7.15 Average Calories Burned Across IDs 48
7.16 Weekly and Monthly Average Weight Trend 49
7.17 User Sleep Categories 50
7.18 Minutes Asleep per Day 51
7.19 Sleep vs. 53
8 Act **53**
8.1 Opportunities: Bellabeat App, Leaf, Time & Spring 54

1 BellaBeat Background

1.1 About the Company

1.1.1 Bellabeat is a health focused high-tech company that produces smart products founded by Urška Sršen and Sando Mur.

1.1.2 These smart products collect data on activity, sleep, stress, and reproductive health. These products include:

1.1.2.1 Bellabeat app: The Bellabeat app provides users with health data related to their activity, sleep, stress, menstrual cycle, and mindfulness habits.**

1.1.2.2 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.

1.1.2.3 Time: This wellness watch combines the timeless look of a classic timepiece with smart technology to track user activity, sleep, and stress.

1.1.2.4 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.

2 Ask

2.1 Details

2.1.1 Urška Sršen asks us to use outside Smart Device data to analyze their trends and see if any insights can be applied to Bellabeats products?

2.1.2 We are going to use public FitBit data provided to us by Möbius through Kaggle to analyze Fitbit trends. These trends can give us insights that can be applied to campaigns focused on Bellabeats' Leaf, Time and Bellabeat App products.

2.2 Questions:

2.2.0.1 How do FitBit customers use their smart devices and do they gain anything from extended use?

2.2.0.2 Can these insights be used on Bellabeat products?

2.2.0.3 Can we generate any sales, deals & offers to customers that reach specific milestones?

2.2.0.4 Are there any insights that can be used for promotional campaigns?

3 Prepare

3.1 Observe Dataset

3.1.1 FitBit data is a public dataset provided by Möbius. We can find their profile on Kaggle.com.

3.1.2 There are 18 .csv files in this data set. This dataset was generated by a survey via Amazon Mechanical Turk from 03/12/2016 to 05/12/2016. Thirty eligible Fitbit users consented to the submission of personal tracker data, including minute-level output for physical activity, heart rate, and sleep monitoring.

3.1.3 There are 33 unique Ids in this dataset and it only tracks data for 2 months. With this small of a sample size and this little of a time length there might be a sampling bias. Also we might not be able to find trends and the trends we do see might be misleading.

3.1.4 We will go through the data and check for duplicated, missing or incorrect data. Although we can assume most of the data's integrity is valid due to being tracked by the smart device.

4 Get environment ready.

4.1 Load Packages

4.1.1 Install Packages if this is the first time using them. this is only required if this is your first time using the package.

```
install.packages("tidyverse")
install.packages("dplyr")
install.packages("janitor")
install.packages("lubridate")
install.packages("here")
install.packages("skimr")
install.packages("ggplot2")
install.packages("ggpubr")
install.packages("ggrepel")
install.packages("reshape2")
install.packages("corrplot")
```

4.1.2 Note: Load packages from library. This step is required every time RStudio is restarted.

```
library(tidyverse)
library(dplyr)
library(janitor)
library(lubridate)
library(here)
library(skimr)
library(ggplot2)
```

```
library(ggpubr)
library(ggrepel)
library(reshape2)
library(corrplot)
library(writexl)
```

4.2 Import dataset.

```
minuteStepsNarrow_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteStepsNarrow_merged.csv")

minuteStepsWide_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteStepsWide_merged.csv")

sleepDay_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/sleepDay_merged.csv")

weightLogInfo_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/weightLogInfo_merged.csv")

minuteIntensitiesNarrow_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteIntensitiesNarrow_merged.csv")

minuteIntensitiesWide_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteIntensitiesWide_merged.csv")

minuteMETsNarrow_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteMETsNarrow_merged.csv")

minuteSleep_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteSleep_merged.csv")

minuteCaloriesNarrow_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteCaloriesNarrow_merged.csv")

minuteCaloriesWide_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/minuteCaloriesWide_merged.csv")

heartrate_seconds_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/heartrate_seconds_merged.csv")

hourlyCalories_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/hourlyCalories_merged.csv")

hourlyIntensities_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/hourlyIntensities_merged.csv")

hourlySteps_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/hourlySteps_merged.csv")
```

```
dailyActivity_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/dailyActivity_merged.csv")

dailyCalories_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/dailyCalories_merged.csv")

dailyIntensities_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/dailyIntensities_merged.csv")

dailySteps_merged <- read_csv(
  "archive/Bellabeat/Fitabase Data 4.12.16-5.12.16/dailySteps_merged.csv")
```

4.2.0.1 Note: Use `read_csv` to import files.

4.2.0.2 Note: As we can see there is duplicate data. There are data frames in long format and wide format. Although they have the same data values.

4.2.1 Lets remove the wide format data frames because we have non duplicates that are in long format and we want to keep them uniform.

```
rm(minuteStepsWide_merged, minuteIntensitiesWide_merged, minuteCaloriesWide_merged)
```

4.2.1.1 Note: Use `rm()` function to achieve this.

4.2.2 We will be able to determine how FitBit customers use their devices based on Time used / worn, steps taken, FitBit traveled and intensity of activities based on User, day of the week, daily activity, hourly and activity by the minute. We will also be able to calculate averages and percentage of usage based on these same categories which will then allow us to group our users into high, moderate and low usage of those categories.

4.2.3 We can see that there might be problems with sleep, heart rate and weight log data. These observations are based on the high amount of NA values and duplicates that need to be reformatted or removed from the datasets. The fact that they do not contain all 33 unique users that the survey is based on. Or that the weight log dataset is mostly not recorded by the smart device, but through manual reporting. Also 33 unique users is an extremely small sample size, and the data has only been recorded for 2 months, which is not long enough to see an accurate rate of change.

5 Process

5.1 Clean and Format Data

5.1.1 Let's check for duplicates and NA values.

```
sum(duplicated(dailyActivity_merged))
```

5.1.1.1 Note: First check the daily data frames for duplicate values using sum and `duplicated`.

```
## [1] 0
```

```
sum(duplicated(dailyCalories_merged))
```

```
## [1] 0
```

```
sum(duplicated(dailyIntensities_merged))
```

```
## [1] 0
```

```
sum(duplicated(dailySteps_merged))
```

```
## [1] 0
```

5.1.1.2 Note: 0's are returned for duplicates on all 4 dataframes.

5.1.2 Now check for `NA` values.

```
sum(is.na(dailyActivity_merged))
```

5.1.2.1 Note: use `sum` and `is.na` functions to achieve this.

```
## [1] 0
```

```
sum(is.na(dailyCalories_merged))
```

```
## [1] 0
```

```
sum(is.na(dailyIntensities_merged))
```

```
## [1] 0
```

```
sum(is.na(dailySteps_merged))
```

```
## [1] 0
```

5.1.2.2 Note: 0's returned for all 4 data frames.

5.1.3 Now check the hourly data frames.

```
sum(duplicated(hourlyCalories_merged))
```

5.1.3.1 Note: continue using `sum`, `duplicated` and `is.na`.

```
## [1] 0
```

```
sum(duplicated(hourlyIntensities_merged))
```

```
## [1] 0
```

```
sum(duplicated(hourlySteps_merged))
```

```
## [1] 0
```

```
sum(is.na(hourlyCalories_merged))
```

```
## [1] 0
```

```
sum(is.na(hourlyIntensities_merged))
```

```
## [1] 0
```

```
sum(is.na(hourlySteps_merged))
```

```
## [1] 0
```

5.1.3.2 Note: 0's returned for all 3 data frames.

5.1.4 Next check our data recorded by the minute.

```
sum(duplicated(minuteCaloriesNarrow_merged))
```

```
## [1] 0
```

```
sum(duplicated(minuteIntensitiesNarrow_merged))
```

```
## [1] 0
```

```
sum(duplicated(minuteMETsNarrow_merged))
```

```
## [1] 0
```

```
sum(duplicated(minuteStepsNarrow_merged))
```

```
## [1] 0
```

```
sum(is.na(minuteCaloriesNarrow_merged))
```

```
## [1] 0
```

```
sum(is.na(minuteIntensitiesNarrow_merged))
```

```
## [1] 0
```

```
sum(is.na(minuteMETsNarrow_merged))
```

```
## [1] 0
```

```
sum(is.na(minuteStepsNarrow_merged))
```

```
## [1] 0
```

5.1.4.1 Note: 0's returned for all 4 data frames.

```
sum(duplicated(sleepDay_merged))
```

5.1.4.2 Finally let's check our sleep, weight and heart rate frames.

```
## [1] 3
```

```
sum(is.na(sleepDay_merged))
```

```
## [1] 0
```

```
sum(duplicated(minuteSleep_merged))
```

```
## [1] 543
```

```
sum(is.na(minuteSleep_merged))
```

```
## [1] 0
```

```
sum(duplicated(weightLogInfo_merged))
```

```
## [1] 0
```

```
sum(is.na(weightLogInfo_merged))
```

```
## [1] 65
```

```
sum(duplicated(heartrate_seconds_merged))
```

```
## [1] 0
```

```
sum(is.na(heartrate_seconds_merged))
```

```
## [1] 0
```

5.2 Sleep Data

5.2.0.1 Note: 3 & 543 duplicates returned for `sleepDay_merged` and `minuteSleep_merged` respectively. Also 65 NA values returned for `weightLogInfo_merged`.

5.2.1 We need to remove duplicates for `sleepDay_merged` and `minuteSleep_merged` then drop the NA values for `weightLogInfo_merged`.

5.2.1.1 Note: Use `unique` function to remove duplicates and `drop_na` to remove the NA values. Although we see the 5th column `Fat` contains mostly NA values.

5.2.2 In order to preserve data we will first remove the 5 column before using `drop_na`.

```
sleepDay_merged <- unique(sleepDay_merged)
minuteSleep_merged <- unique(minuteSleep_merged)

weightLogInfo_merged <- weightLogInfo_merged[, -5]
weightLogInfo_merged <- drop_na(weightLogInfo_merged)
```

```
sum(duplicated(sleepDay_merged))
```

5.2.2.1 Note: Now confirm we no longer have duplicates or NAs.

```
## [1] 0
```

```r
sum(duplicated(minuteSleep_merged))
```

```
## [1] 0
```

```r
sum(is.na(weightLogInfo_merged))
```

```
## [1] 0
```

5.2.3 Let's preview our data.

```r
head(dailyCalories_merged)
```

```
## # A tibble: 6 x 3
##           Id ActivityDay Calories
##        <dbl> <chr>          <dbl>
## 1 1503960366 4/12/2016       1985
## 2 1503960366 4/13/2016       1797
## 3 1503960366 4/14/2016       1776
## 4 1503960366 4/15/2016       1745
## 5 1503960366 4/16/2016       1863
## 6 1503960366 4/17/2016       1728
```

```r
head(dailyIntensities_merged)
```

```
## # A tibble: 6 x 10
##           Id ActivityDay SedentaryMinutes LightlyActiveMinutes FairlyActiveMinutes
##        <dbl> <chr>                  <dbl>                <dbl>                <dbl>
## 1 1503960366 4/12/2016                728                  328                   13
## 2 1503960366 4/13/2016                776                  217                   19
## 3 1503960366 4/14/2016               1218                  181                   11
## 4 1503960366 4/15/2016                726                  209                   34
## 5 1503960366 4/16/2016                773                  221                   10
## 6 1503960366 4/17/2016                539                  164                   20
## # i 5 more variables: VeryActiveMinutes <dbl>, SedentaryActiveDistance <dbl>,
## #   LightActiveDistance <dbl>, ModeratelyActiveDistance <dbl>,
## #   VeryActiveDistance <dbl>
```

```r
head(dailySteps_merged)
```

```
## # A tibble: 6 x 3
##           Id ActivityDay StepTotal
##        <dbl> <chr>           <dbl>
## 1 1503960366 4/12/2016       13162
## 2 1503960366 4/13/2016       10735
## 3 1503960366 4/14/2016       10460
## 4 1503960366 4/15/2016        9762
## 5 1503960366 4/16/2016       12669
## 6 1503960366 4/17/2016        9705
```

```r
head(dailyActivity_merged)
```

```
## # A tibble: 6 x 15
##         Id ActivityDate TotalSteps TotalDistance TrackerDistance LoggedActivitiesDist~1
##      <dbl> <chr>             <dbl>         <dbl>           <dbl>                   <dbl>
## 1   1.50e9 4/12/2016         13162          8.5             8.5                        0
## 2   1.50e9 4/13/2016         10735          6.97            6.97                       0
## 3   1.50e9 4/14/2016         10460          6.74            6.74                       0
## 4   1.50e9 4/15/2016          9762          6.28            6.28                       0
## 5   1.50e9 4/16/2016         12669          8.16            8.16                       0
## 6   1.50e9 4/17/2016          9705          6.48            6.48                       0
## # i abbreviated name: 1: LoggedActivitiesDistance
## # i 9 more variables: VeryActiveDistance <dbl>, ModeratelyActiveDistance <dbl>,
## #   LightActiveDistance <dbl>, SedentaryActiveDistance <dbl>, VeryActiveMinutes <dbl>,
## #   FairlyActiveMinutes <dbl>, LightlyActiveMinutes <dbl>, SedentaryMinutes <dbl>,
## #   Calories <dbl>
```

5.2.3.1 Note We see that the `dailyActivity_merged` data frame consists of the info of `dailyCalories_merged`, `dailyIntensities_merged` and `dailySteps_merged`.

5.3 Steps Data

5.3.1 If you look closely we can see that that data set has totals for `steps`, `calories` and `distance`. Although there is no column for total minutes worn or total active minutes. So let's create them.

```r
DailyActivity <- dailyActivity_merged %>%
  mutate(
    ActiveMinuteCount = LightlyActiveMinutes + FairlyActiveMinutes + VeryActiveMinutes,
    TotalWornCount = LightlyActiveMinutes + FairlyActiveMinutes + VeryActiveMinutes + SedentaryMinutes)

Daily_summary <- DailyActivity %>%
  group_by(Id) %>%
  summarise(
    TotalStepCount = sum(TotalSteps),
    TotalDistanceCount = sum(TotalDistance),
    TotalTrackerDistanceCount = sum(TrackerDistance),
    TotalCalories = sum(Calories),
    TotalActiveMinuteCount = sum(ActiveMinuteCount),
    TotalWornCount = sum(TotalWornCount))

head(DailyActivity)
```

5.3.1.1 Note: Also lets create a data frame that counts to total number of each category based on each unique user.

```
## # A tibble: 6 x 17
##         Id ActivityDate TotalSteps TotalDistance TrackerDistance LoggedActivitiesDist~1
```

```
##      <dbl> <chr>             <dbl>         <dbl>           <dbl>                  <dbl>
## 1   1.50e9 4/12/2016         13162          8.5             8.5                       0
## 2   1.50e9 4/13/2016         10735          6.97            6.97                      0
## 3   1.50e9 4/14/2016         10460          6.74            6.74                      0
## 4   1.50e9 4/15/2016          9762          6.28            6.28                      0
## 5   1.50e9 4/16/2016         12669          8.16            8.16                      0
## 6   1.50e9 4/17/2016          9705          6.48            6.48                      0
## # i abbreviated name: 1: LoggedActivitiesDistance
## # i 11 more variables: VeryActiveDistance <dbl>, ModeratelyActiveDistance <dbl>,
## #   LightActiveDistance <dbl>, SedentaryActiveDistance <dbl>, VeryActiveMinutes <dbl>,
## #   FairlyActiveMinutes <dbl>, LightlyActiveMinutes <dbl>, SedentaryMinutes <dbl>,
## #   Calories <dbl>, ActiveMinuteCount <dbl>, TotalWornCount <dbl>
```

5.3.2 Also let's preview our `Daily_summary` data frame

```
head(Daily_summary)
```

```
## # A tibble: 6 x 7
##           Id TotalStepCount TotalDistanceCount TotalTrackerDistanceCount TotalCalories
##        <dbl>          <dbl>              <dbl>                     <dbl>         <dbl>
## 1 1503960366         375619              242.                      242.          56309
## 2 1624580081         178061              121.                      121.          45984
## 3 1644430081         218489              159.                      159.          84339
## 4 1844505072          79982               52.9                      52.9         48778
## 5 1927972279          28400               19.7                      19.7         67357
## 6 2022484408         352490              251.                      251.          77809
## # i 2 more variables: TotalActiveMinuteCount <dbl>, TotalWornCount <dbl>
```

5.4 Joins

5.4.1 Let's combine the data for our hourly and minute data frames to match the format of `dailyActivity_merged`.

```
HourlyActivity <- full_join(hourlyCalories_merged,
                            hourlyIntensities_merged)
```

5.4.1.1 Note: Use `full_join` to achieve this. First Hourly data.

```
## Joining with `by = join_by(Id, ActivityHour)`
```

```
HourlyActivity <- full_join(HourlyActivity, hourlySteps_merged)
```

```
## Joining with `by = join_by(Id, ActivityHour)`
```

```
head(HourlyActivity)
```

```
## # A tibble: 6 x 6
##           Id ActivityHour          Calories TotalIntensity AverageIntensity StepTotal
##        <dbl> <chr>                    <dbl>          <dbl>            <dbl>     <dbl>
## 1 1503960366 4/12/2016 12:00:00 AM       81             20            0.333       373
## 2 1503960366 4/12/2016 1:00:00 AM        61              8            0.133       160
## 3 1503960366 4/12/2016 2:00:00 AM        59              7            0.117       151
## 4 1503960366 4/12/2016 3:00:00 AM        47              0            0             0
## 5 1503960366 4/12/2016 4:00:00 AM        48              0            0             0
## 6 1503960366 4/12/2016 5:00:00 AM        48              0            0             0
```

5.4.1.2 Note: Data frames are being joined based on `Id` and `ActivityHour` columns.

5.4.2 Now combine the data collected by the minute.

```
ActivitybyMinute <- full_join(minuteCaloriesNarrow_merged, minuteIntensitiesNarrow_merged)
```

5.4.2.1 Note: Use `full_join` function again.

```
## Joining with `by = join_by(Id, ActivityMinute)`
```

```
ActivitybyMinute2 <- full_join(minuteMETsNarrow_merged, minuteStepsNarrow_merged)
```

```
## Joining with `by = join_by(Id, ActivityMinute)`
```

```
ActivitybyMinute <- full_join(ActivitybyMinute, ActivitybyMinute2)
```

```
## Joining with `by = join_by(Id, ActivityMinute)`
```

```
head(ActivitybyMinute)
```

```
## # A tibble: 6 x 6
##           Id ActivityMinute        Calories Intensity  METs Steps
##        <dbl> <chr>                    <dbl>     <dbl> <dbl> <dbl>
## 1 1503960366 4/12/2016 12:00:00 AM    0.786         0    10     0
## 2 1503960366 4/12/2016 12:01:00 AM    0.786         0    10     0
## 3 1503960366 4/12/2016 12:02:00 AM    0.786         0    10     0
## 4 1503960366 4/12/2016 12:03:00 AM    0.786         0    10     0
## 5 1503960366 4/12/2016 12:04:00 AM    0.786         0    10     0
## 6 1503960366 4/12/2016 12:05:00 AM    0.944         0    12     0
```

5.4.2.2 Note: Data frames are joined based on `Id` and `ActivityMinute` columns.

5.4.3 Now that we have our combined data frames we can remove the ones no longer needed or that we cannot combine to view trends ir gain insights.

5.5 Remove Unused Data

```
rm(ActivitybyMinute2, dailyActivity_merged, dailyCalories_merged, dailyIntensities_merged, dailySteps_m
```

5.5.0.1 Note: Once again use `rm()` function.

5.6 Date Columns

5.6.0.1 Note: In all of our data frames the date columns are `Character` values. We need them to be `date` and `time` values.

5.6.1 We are going to use `as.POSIXct` to achieve this.

```
sleepDay_merged$SleepDay <- as.POSIXct(sleepDay_merged$SleepDay, format = "%m/%d/%Y %I:%M:%S %p", tz = 
weightLogInfo_merged$Date <- as.POSIXct(weightLogInfo_merged$Date, format = "%m/%d/%Y %I:%M:%S %p", tz =
ActivitybyMinute$ActivityMinute <-as.POSIXct(ActivitybyMinute$ActivityMinute, format = "%m/%d/%Y %I:%M:
HourlyActivity$ActivityHour <- as.POSIXct(HourlyActivity$ActivityHour, format = "%m/%d/%Y %I:%M:%S %p",
DailyActivity$ActivityDate <- as.POSIXct(DailyActivity$ActivityDate, format = "%m/%d/%Y", tz = "GMT")
```

5.6.1.1 Note: First change the format of the columns.

```
ActivitybyMinute$Date <- as.Date(ActivitybyMinute$ActivityMinute)
ActivitybyMinute$Time <- format(ActivitybyMinute$ActivityMinute, format = "%I:%M:%S %p", tz = "GMT")
sleepDay_merged$Date <- as.Date(sleepDay_merged$SleepDay)
sleepDay_merged$Time <- format(sleepDay_merged$SleepDay, format = "%I:%M:%S %p", tz = "GMT")
weightLogInfo_merged$Time <- format(weightLogInfo_merged$Date, format = "%I:%M:%S %p", tz = "GMT")
weightLogInfo_merged$Date <- as.Date(weightLogInfo_merged$Date)
HourlyActivity$Date <- as.Date(HourlyActivity$ActivityHour)
HourlyActivity$Time <- format(HourlyActivity$ActivityHour, format = "%I:%M:%S %p", tz = "GMT")
DailyActivity$Date <- as.Date(DailyActivity$ActivityDate)
```

5.6.1.2 Note: Then we separate the `date` from the `time` and give them their own column and classify the format once again.

5.7 Days of the Week

```
ActivitybyMinute$Weekday <- wday(ActivitybyMinute$Date, label = TRUE)

DailyActivity$Weekday <- wday(DailyActivity$Date, label = TRUE)

HourlyActivity$Weekday <- wday(HourlyActivity$Date, label = TRUE)

sleepDay_merged$Weekday <- wday(sleepDay_merged$Date, label = TRUE)

weightLogInfo_merged$Weekday <- wday(weightLogInfo_merged$Date, label = TRUE)
```

5.7.0.1 Note: Finally lets get the associating day of the week.

5.7.0.2 Note: For our final step we verify the amount of unique values is over 30. The amount of participants in FitBit survey.

5.7.1 In our data cleaning process we've checked for duplicates and NA values, then removed them. We've formatted our date and time columns while finding the associated day of the week. And we've combined relevant data frames based on Daily, hourly and minute activities.

5.7.2 We can verify that the data is clean based on the steps we have taken and by running a final check of duplicates, NA values and data types using the `str` function.

```
n_unique(ActivitybyMinute$Id)
```

5.7.2.1 Note: We can verify the number of Unique Id numbers using the `n_unique` function.

```
## [1] 33
```

```
n_unique(DailyActivity$Id)
```

```
## [1] 33
```

```
n_unique(HourlyActivity$Id)
```

```
## [1] 33
```

```
n_unique(sleepDay_merged$Id)
```

```
## [1] 24
```

```
n_unique(weightLogInfo_merged$Id)
```

```
## [1] 8
```

6 Analyze

6.1 Look for patterns

6.1.1 First we are going to calculate averages and percentages of all our categories. Using the mean function we can find the averages of steps, distance, calories, time used / worn and Intensity of activities. Then we are going to group the users into high, moderate and low based on these numbers. We can also calculate these numbers based on the day of the week to see which days get the most usage. Then we are going to do the same for total counts so we can get an idea of the volume of use. Finally we will run calculations on the sleep, heart rate and weight log datasets to see if we can gain any insights or verify that they cannot be used.

6.2 Averages

```
Average_DailyActivity <- DailyActivity %>%
  group_by(Id) %>%
  summarise(mean_totalsteps = mean(TotalSteps), mean_totaldistance = mean(TotalDistance), mean_sedentary

Average_HourlyActivity <-HourlyActivity %>%
  group_by(Id) %>%
  summarise(mean_calories = mean(Calories), mean_intensity = mean(TotalIntensity), mean_steps = mean(St

Average_ActivitybyMinute <- ActivitybyMinute %>%
  group_by(Id) %>%
  summarise(mean_calories = mean(Calories), mean_intensity = mean(Intensity), mean_METs = mean(METs), m

head(Average_DailyActivity)
```

6.2.0.1 Note: Let's start by calculating averages.

```
## # A tibble: 6 x 6
##          Id mean_totalsteps mean_totaldistance mean_sedentaryminutes mean_activeminutes
##       <dbl>           <dbl>              <dbl>                 <dbl>              <dbl>
## 1    1.50e9          12117.              7.81                   848.              278.
## 2    1.62e9           5744.              3.91                  1258.              168.
## 3    1.64e9           7283.              5.30                  1162.              209.
## 4    1.84e9           2580.              1.71                  1207.              117.
## 5    1.93e9            916.              0.635                 1317.               40.7
## 6    2.02e9          11371.              8.08                  1113.              313.
## # i 1 more variable: mean_calories <dbl>
```

```r
head(Average_HourlyActivity)
```

```
## # A tibble: 6 x 4
##           Id mean_calories mean_intensity mean_steps
##        <dbl>         <dbl>          <dbl>      <dbl>
## 1 1503960366          78.5          16.2       522.
## 2 1624580081          62.5           8.04      242.
## 3 1644430081         119.           10.5       308.
## 4 1844505072          66.6           5.02      109.
## 5 1927972279          91.5           1.86       38.6
## 6 2022484408         105.           17.0       478.
```

```r
head(Average_ActivitybyMinute)
```

```
## # A tibble: 6 x 5
##           Id mean_calories mean_intensity mean_METs mean_steps
##        <dbl>         <dbl>          <dbl>     <dbl>      <dbl>
## 1 1503960366          1.31          0.270      16.7      8.71
## 2 1624580081          1.04          0.134      12.5      4.03
## 3 1644430081          1.98          0.175      14.1      5.13
## 4 1844505072          1.11          0.0837     11.9      1.82
## 5 1927972279          1.52          0.0310     10.6      0.643
## 6 2022484408          1.76          0.284      17.0      7.96
```

6.2.1 Looking at the `sleepDay_merged` data frame, we can group the data based on unique users and count the totals like we did for the `Daily_summary` data frame.

```r
sleepDay_summary <- sleepDay_merged %>%
  group_by(Id) %>%
  summarise(
    SleepRecordsCount = sum(TotalSleepRecords),
    TotalAsleepMinutes = sum(TotalMinutesAsleep),
    TotalBedTime = sum(TotalTimeInBed))

Average_sleepDay_summary <- sleepDay_merged %>%
  group_by(Id) %>%
  summarise(
    SleepRecordsCount = sum(TotalSleepRecords),
    AverageAsleepMinutes = mean(TotalMinutesAsleep),
    AverageBedTime = mean(TotalTimeInBed))

head(sleepDay_summary)
```

6.2.1.1 Note: Use `group_by` and `summarise` functions to achieve this.

```
## # A tibble: 6 x 4
##           Id SleepRecordsCount TotalAsleepMinutes TotalBedTime
##        <dbl>             <dbl>              <dbl>        <dbl>
## 1 1503960366                27               9007         9580
```

```
## 2 1644430081                 4               1176         1384
## 3 1844505072                 3               1956         2883
## 4 1927972279                 8               2085         2189
## 5 2026352035                28              14173        15054
## 6 2320127002                 1                 61           69
```

```
head(Average_sleepDay_summary)
```

```
## # A tibble: 6 x 4
##           Id SleepRecordsCount AverageAsleepMinutes AverageBedTime
##        <dbl>             <dbl>                <dbl>          <dbl>
## 1 1503960366                27                360.           383.
## 2 1644430081                 4                294            346
## 3 1844505072                 3                652            961
## 4 1927972279                 8                417            438.
## 5 2026352035                28                506.           538.
## 6 2320127002                 1                 61             69
```

6.2.2 Now we can combine the `sleepDay_summary` data with our `Daily_summary` data and `Average_sleepday_ summary` with `Average_DailyActiviy`.

```
DailyActivitywithSleep <- full_join(Daily_summary, sleepDay_summary) %>%
  distinct(Id, .keep_all = TRUE) %>%
  drop_na()
```

6.2.2.1 Note: Use `drop_na` to ensure data frame stays clean although we will lose data for 9 users in this data frame because `sleepDay_summary` only has 24 unique Ids.

```
## Joining with `by = join_by(Id)`
```

```
AverageActivitywithSleep <- full_join(Average_DailyActivity, Average_sleepDay_summary) %>%
  distinct(Id, .keep_all = TRUE) %>%
  drop_na()
```

```
## Joining with `by = join_by(Id)`
```

```
head(DailyActivitywithSleep)
```

```
## # A tibble: 6 x 10
##           Id TotalStepCount TotalDistanceCount TotalTrackerDistanceCount TotalCalories
##        <dbl>          <dbl>              <dbl>                     <dbl>         <dbl>
## 1 1503960366         375619              242.                      242.          56309
## 2 1644430081         218489              159.                      159.          84339
## 3 1844505072          79982               52.9                      52.9         48778
## 4 1927972279          28400               19.7                      19.7         67357
## 5 2026352035         172573              107.                      107.          47760
## 6 2320127002         146223               98.8                      98.8         53449
## # i 5 more variables: TotalActiveMinuteCount <dbl>, TotalWornCount <dbl>,
## #   SleepRecordsCount <dbl>, TotalAsleepMinutes <dbl>, TotalBedTime <dbl>
```

```
head(AverageActivitywithSleep)
```

```
## # A tibble: 6 x 9
##          Id mean_totalsteps mean_totaldistance mean_sedentaryminutes mean_activeminutes
##       <dbl>           <dbl>              <dbl>                 <dbl>              <dbl>
## 1    1.50e9          12117.              7.81                  848.              278.
## 2    1.64e9           7283.              5.30                 1162.              209.
## 3    1.84e9           2580.              1.71                 1207.              117.
## 4    1.93e9            916.              0.635                1317.               40.7
## 5    2.03e9           5567.              3.45                  689.              257
## 6    2.32e9           4717.              3.19                 1220.              202.
## # i 4 more variables: mean_calories <dbl>, SleepRecordsCount <dbl>,
## #   AverageAsleepMinutes <dbl>, AverageBedTime <dbl>
```

```
sum(is.na(DailyActivitywithSleep))
```

6.2.2.2 Note: check to see if new data frame needs to be cleaned.

```
## [1] 0
```

```
sum(duplicated(DailyActivitywithSleep))
```

```
## [1] 0
```

6.2.2.3 Note: The categories we are going to create for average Step amounts are Low Average for anything under 4000, Moderate Average for anything in between 4001-7999 and High Average for anything over 8000 steps.

6.3 Categorize Users by average daily steps.

```
Step_Type <- Average_DailyActivity %>%
  mutate(step_type = case_when(mean_totalsteps < 4000 ~ "Low Average", mean_totalsteps >= 4000 & mean_t

Step_percent <- Step_Type %>%
  group_by(step_type) %>%
  summarise(count = n()) %>%
  mutate(total = sum(count)) %>%
  group_by(step_type) %>%
  summarise(step_percent = count/total) %>%
  mutate(step_percent = scales::percent(step_percent, accuracy = 1))

Step_percent$step_percent <- as.numeric(gsub("%", "", Step_percent$step_percent))

head(Step_Type)
```

6.3.0.1 Note: After we group our categories we are going to calculate the percent of users in each category.

```
## # A tibble: 6 x 7
##          Id mean_totalsteps mean_totaldistance mean_sedentaryminutes mean_activeminutes
##       <dbl>           <dbl>              <dbl>                 <dbl>              <dbl>
## 1    1.50e9          12117.               7.81                  848.              278.
## 2    1.62e9           5744.               3.91                 1258.              168.
## 3    1.64e9           7283.               5.30                 1162.              209.
## 4    1.84e9           2580.               1.71                 1207.              117.
## 5    1.93e9            916.              0.635                 1317.               40.7
## 6    2.02e9          11371.               8.08                 1113.              313.
## # i 2 more variables: mean_calories <dbl>, step_type <chr>
```

```
head(Step_percent)
```

```
## # A tibble: 3 x 2
##   step_type        step_percent
##   <chr>                   <dbl>
## 1 High Average               42
## 2 Low Average                18
## 3 Moderate Average           39
```

6.3.1 Categorize Users based on active minutes and total time device was worn.

6.4 Percentages

```
DailyActivity <- DailyActivity %>%
  mutate(active_percent = round((ActiveMinuteCount / 1440) * 100),
    activity_type = case_when(
      active_percent <= 100 & active_percent >= 31 ~ "Very Active",
      active_percent <= 30 & active_percent >= 16 ~ "Moderately Active",
      active_percent <= 15 & active_percent > 0 ~ "Fairly Active",
      TRUE ~ "Not Active"),
    worn_percent = round((TotalWornCount / 1440) * 100),
    worn_type = case_when(
      worn_percent <= 100 & worn_percent >= 76 ~ "High Usage",
      worn_percent <= 75 & worn_percent >= 51 ~ "Moderate Usage",
      worn_percent <= 50 & worn_percent >= 26 ~ "Low Usage",
      worn_percent <= 25 & worn_percent > 0 ~ "No Usage",
      TRUE ~ "No Usage"))

active_percent <- DailyActivity %>%
  group_by(activity_type) %>%
  summarise(count = mean(ActiveMinuteCount)) %>%
  mutate(total = sum(count)) %>%
  group_by(activity_type) %>%
  summarise(activity_percent = count/total) %>%
  mutate(activity_percent = scales::percent(activity_percent,, accuracy = 1))
```

```
worn_percent <- DailyActivity %>%
  group_by(worn_type) %>%
  summarise(count = mean(TotalWornCount)) %>%
  mutate(total = sum(count)) %>%
  group_by(worn_type) %>%
  summarise(worn_percent = count/total) %>%
  mutate(worn_percent = scales::percent(worn_percent, accuracy = 1))

active_percent$activity_percent <- as.numeric(gsub("%", "", active_percent$activity_percent))

worn_percent$worn_percent <- as.numeric(gsub("%", "", worn_percent$worn_percent))

head(active_percent)
```

6.4.0.1 Note: Then we can find the percent of users in each category.

```
## # A tibble: 4 x 2
##   activity_type     activity_percent
##   <chr>                        <dbl>
## 1 Fairly Active                   15
## 2 Moderately Active               33
## 3 Not Active                       0
## 4 Very Active                     52
```

```
head(worn_percent)
```

```
## # A tibble: 4 x 2
##   worn_type      worn_percent
##   <chr>                 <dbl>
## 1 High Usage               45
## 2 Low Usage                20
## 3 Moderate Usage           31
## 4 No Usage                  5
```

6.5 Total Amounts

6.5.1 Now Let's find the total amount of steps and average amount of steps based on the day of the week.

```
Daily_steps <- DailyActivity %>%
  group_by(Weekday) %>%
  summarise(total_daily_steps = sum(TotalSteps), average_daily_steps = mean(TotalSteps))

head(Daily_steps)
```

6.5.1.1 Note: Use `group_by` and `summarise` functions.

```
## # A tibble: 6 x 3
```

```
##   Weekday total_daily_steps average_daily_steps
##   <ord>               <dbl>               <dbl>
## 1 Sun                838921               6933.
## 2 Mon                933704               7781.
## 3 Tue               1235001               8125.
## 4 Wed               1133906               7559.
## 5 Thu               1088658               7406.
## 6 Fri                938477               7448.
```

```
Total_step_percent <- DailyActivity %>%
  group_by(Weekday) %>%
  summarise(total_daily_steps = sum(TotalSteps)) %>%
  mutate(total = sum(total_daily_steps)) %>%
  group_by(Weekday) %>%
  summarise(daily_percent = total_daily_steps / total) %>%
  mutate(daily_percent = scales::percent(daily_percent, accuracy = 1))

Average_step_percent <- DailyActivity %>%
  group_by(Weekday) %>%
  summarise(average_daily_steps = mean(TotalSteps)) %>%
  mutate(total = sum(average_daily_steps)) %>%
  group_by(Weekday) %>%
  summarise(average_percent = average_daily_steps/total) %>%
  mutate(average_percent = scales::percent(average_percent, accuracy = 1))

Total_step_percent$daily_percent <- as.numeric(gsub("%", "", Total_step_percent$daily_percent))

Average_step_percent$average_percent <- as.numeric(gsub("%", "", Average_step_percent$average_percent))

head(Total_step_percent)
```

6.5.1.2 Note: Now we can calculate the percent of steps taken on the associated day. Total and average steps.

```
## # A tibble: 6 x 2
##   Weekday daily_percent
##   <ord>           <dbl>
## 1 Sun                12
## 2 Mon                13
## 3 Tue                17
## 4 Wed                16
## 5 Thu                15
## 6 Fri                13
```

```
head(Average_step_percent)
```

```
## # A tibble: 6 x 2
##   Weekday average_percent
##   <ord>             <dbl>
## 1 Sun                  13
```

```
## 2 Mon                  15
## 3 Tue                  15
## 4 Wed                  14
## 5 Thu                  14
## 6 Fri                  14
```

6.5.2 Categorize Users based on time spent asleep.

```
Daily_Sleep <- sleepDay_merged %>%
  group_by(Id) %>%
  summarise(
    total_minutes_asleep = sum(TotalMinutesAsleep),
    total_minutes_in_bed = sum(TotalTimeInBed),
    average_minutes_asleep = mean(TotalMinutesAsleep),
    average_minutes_in_bed = mean(TotalTimeInBed),
    total_hrs_asleep = total_minutes_asleep / 60,
    average_hrs_asleep = average_minutes_asleep / 60
  ) %>%
  mutate(average_hrs_asleep = round(average_hrs_asleep), total_hrs_asleep = round(total_hrs_asleep))

head(Daily_Sleep)
```

6.5.2.1 Note: First create data frame that calculates total mins asleep / in bed and average mins asleep / in bed.

```
## # A tibble: 6 x 7
##           Id total_minutes_asleep total_minutes_in_bed average_minutes_asleep
##        <dbl>                <dbl>                <dbl>                   <dbl>
## 1 1503960366                 9007                 9580                    360.
## 2 1644430081                 1176                 1384                    294
## 3 1844505072                 1956                 2883                    652
## 4 1927972279                 2085                 2189                    417
## 5 2026352035                14173                15054                    506.
## 6 2320127002                   61                   69                     61
## # i 3 more variables: average_minutes_in_bed <dbl>, total_hrs_asleep <dbl>,
## #   average_hrs_asleep <dbl>
```

6.6 Categorize

6.6.1 Then place users into category based on average minutes asleep.

```
Sleep_Type <- Daily_Sleep %>%
  mutate(sleep_type = case_when(average_minutes_asleep < 300 ~ "Light Sleep", average_minutes_asleep >=

Sleep_percent <- Sleep_Type %>%
  group_by(sleep_type) %>%
  summarise(count = n()) %>%
```

```
  mutate(total = sum(count)) %>%
  group_by(sleep_type) %>%
  summarise(sleep_percent = count/total) %>%
  mutate(sleep_percent = scales::percent(sleep_percent, accuracy = 1))

Sleep_percent$sleep_percent <- as.numeric(gsub("%", "", Sleep_percent$sleep_percent))

head(Sleep_Type)
```

6.6.1.1 Note: Under 300 mins = Light Sleep, between 301-420 mins = Moderate Sleep and anything above 421 min = Heavy Sleep.

```
## # A tibble: 6 x 8
##           Id total_minutes_asleep total_minutes_in_bed average_minutes_asleep
##        <dbl>                <dbl>                <dbl>                   <dbl>
## 1 1503960366                 9007                 9580                    360.
## 2 1644430081                 1176                 1384                    294
## 3 1844505072                 1956                 2883                    652
## 4 1927972279                 2085                 2189                    417
## 5 2026352035                14173                15054                    506.
## 6 2320127002                   61                   69                     61
## # i 4 more variables: average_minutes_in_bed <dbl>, total_hrs_asleep <dbl>,
## #   average_hrs_asleep <dbl>, sleep_type <chr>
```

```
head(Sleep_percent)
```

```
## # A tibble: 3 x 2
##   sleep_type     sleep_percent
##   <chr>                  <dbl>
## 1 Heavy Sleep               46
## 2 Light Sleep               25
## 3 Moderate Sleep            29
```

```
Daily_Sleep_Count <- sleepDay_merged %>%
  group_by(Weekday) %>%
  summarise(
    total_minutes_asleep = sum(TotalMinutesAsleep),
    average_minutes_asleep = mean(TotalMinutesAsleep),
    average_hrs_asleep = average_minutes_asleep / 60
  ) %>%
  mutate(average_hrs_asleep = round(average_hrs_asleep))

head(Daily_Sleep_Count)
```

6.6.1.2 Note: Let's get the average and total amounts of sleep based on the day of the week to see if we gain any insights.

```
## # A tibble: 6 x 4
##   Weekday total_minutes_asleep average_minutes_asleep average_hrs_asleep
```

```
##   <ord>                <dbl>                <dbl>              <dbl>
## 1 Sun                  24901                 453.                  8
## 2 Mon                  19297                 420.                  7
## 3 Tue                  26295                 405.                  7
## 4 Wed                  28689                 435.                  7
## 5 Thu                  25683                 401.                  7
## 6 Fri                  23109                 405.                  7
```

6.7 Percent of minutes asleep

```
Total_sleep_percent <- sleepDay_merged %>%
  group_by(Weekday) %>%
  summarise(total_minutes_asleep = sum(TotalMinutesAsleep)) %>%
  mutate(total = sum(total_minutes_asleep)) %>%
  group_by(Weekday) %>%
  summarise(daily_percent = total_minutes_asleep/total) %>%
  mutate(daily_percent = scales::percent(daily_percent, accuracy = 1))

Average_sleep_percent <- sleepDay_merged %>%
  group_by(Weekday) %>%
  summarise(average_min_asleep = mean(TotalMinutesAsleep)) %>%
  mutate(total = sum(average_min_asleep)) %>%
  group_by(Weekday) %>%
  summarise(average_percent = average_min_asleep/total) %>%
  mutate(average_percent = scales::percent(average_percent, accuracy = 1))

Total_sleep_percent$daily_percent <- as.numeric(gsub("%", "", Total_sleep_percent$daily_percent))

Average_sleep_percent$average_percent <- as.numeric(gsub("%", "", Average_sleep_percent$average_percent)

head(Total_sleep_percent)
```

6.7.0.1 Note: Next we calculate to percent of min asleep based on weekday.

```
## # A tibble: 6 x 2
##   Weekday daily_percent
##   <ord>           <dbl>
## 1 Sun                14
## 2 Mon                11
## 3 Tue                15
## 4 Wed                17
## 5 Thu                15
## 6 Fri                13
```

```
head(Average_sleep_percent)
```

```
## # A tibble: 6 x 2
##   Weekday average_percent
##   <ord>             <dbl>
## 1 Sun                  15
```

```
## 2 Mon                  14
## 3 Tue                  14
## 4 Wed                  15
## 5 Thu                  14
## 6 Fri                  14
```

6.8 Weight Log

6.8.1 Let's run some calculations on `weightLofInfo_merged` to see if can gain any insights.

```
weekly_avg <- weightLogInfo_merged %>%
  group_by(Week = lubridate::week(Date), Year = lubridate::year(Date)) %>%
  summarize(AvgWeight = mean(WeightPounds, na.rm = TRUE))
```

6.8.1.1 Note: We can find the weekly average and monthly average to see if there is any change over time.

```
## `summarise()` has grouped output by 'Week'. You can override using the `.groups`
## argument.
```

```
monthly_avg <- weightLogInfo_merged %>%
  group_by(Month = lubridate::month(Date), Year = lubridate::year(Date)) %>%
  summarize(AvgWeight = mean(WeightPounds, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'Month'. You can override using the `.groups`
## argument.
```

```
max_weight <- weightLogInfo_merged %>%
  summarize(MaxWeight = max(WeightPounds, na.rm = TRUE))

min_weight <- weightLogInfo_merged %>%
  summarize(MinWeight = min(WeightPounds, na.rm = TRUE))

head(weekly_avg)
```

```
## # A tibble: 5 x 3
## # Groups:   Week [5]
##    Week  Year AvgWeight
##   <dbl> <dbl>     <dbl>
## 1    15  2016      181.
## 2    16  2016      158.
## 3    17  2016      161.
## 4    18  2016      152.
## 5    19  2016      155.
```

```
head(monthly_avg)
```

```
## # A tibble: 2 x 3
## # Groups:   Month [2]
##   Month  Year AvgWeight
##   <dbl> <dbl>     <dbl>
## 1     4  2016      163.
## 2     5  2016      152.
```

6.8.1.2 Note: The Min weight entered into our data set is ~116 pounds and the Max is ~294 pounds.

```
Daily_weight_data <- merge(DailyActivity, weightLogInfo_merged, by = c("Id", "Date"))

head(Daily_weight_data)
```

6.8.1.3 Note: Because the `weightLogInfo_merged` and `DailyActivity` data frames contain info for daily counts we can combine their info.

```
##           Id       Date ActivityDate TotalSteps TotalDistance TrackerDistance
## 1 1503960366 2016-05-02   2016-05-02      14727          9.71            9.71
## 2 1503960366 2016-05-03   2016-05-03      15103          9.66            9.66
## 3 1927972279 2016-04-13   2016-04-13        356          0.25            0.25
## 4 2873212765 2016-04-21   2016-04-21       8859          5.98            5.98
## 5 2873212765 2016-05-12   2016-05-12       7566          5.11            5.11
## 6 4319703577 2016-04-17   2016-04-17         29          0.02            0.02
##   LoggedActivitiesDistance VeryActiveDistance ModeratelyActiveDistance
## 1                        0              3.21                     0.57
## 2                        0              3.73                     1.05
## 3                        0              0.00                     0.00
## 4                        0              0.13                     0.37
## 5                        0              0.00                     0.00
## 6                        0              0.00                     0.00
##   LightActiveDistance SedentaryActiveDistance VeryActiveMinutes FairlyActiveMinutes
## 1                5.92                    0.00                41                  15
## 2                4.88                    0.00                50                  24
## 3                0.25                    0.00                 0                   0
## 4                5.47                    0.01                 2                  10
## 5                5.11                    0.00                 0                   0
## 6                0.02                    0.00                 0                   0
##   LightlyActiveMinutes SedentaryMinutes Calories ActiveMinuteCount TotalWornCount
## 1                  277              798     2004               333           1131
## 2                  254              816     1990               328           1144
## 3                   32              986     2151                32           1018
## 4                  371             1057     1970               383           1440
## 5                  268              720     1431               268            988
## 6                    3             1363     1464                 3           1366
##   Weekday.x active_percent     activity_type worn_percent      worn_type WeightKg
## 1       Mon             23 Moderately Active           79     High Usage     52.6
## 2       Tue             23 Moderately Active           79     High Usage     52.6
## 3       Wed              2     Fairly Active           71 Moderate Usage    133.5
## 4       Thu             27 Moderately Active          100     High Usage     56.7
```

```
## 5       Thu             19 Moderately Active           69 Moderate Usage     57.3
## 6       Sun              0        Not Active           95     High Usage     72.4
##   WeightPounds   BMI IsManualReport        LogId        Time Weekday.y
## 1     115.9631 22.65           TRUE 1.462234e+12 11:59:59 PM       Mon
## 2     115.9631 22.65           TRUE 1.462320e+12 11:59:59 PM       Tue
## 3     294.3171 47.54          FALSE 1.460510e+12 01:08:52 AM       Wed
## 4     125.0021 21.45           TRUE 1.461283e+12 11:59:59 PM       Thu
## 5     126.3249 21.69           TRUE 1.463098e+12 11:59:59 PM       Thu
## 6     159.6147 27.45           TRUE 1.460938e+12 11:59:59 PM       Sun
```

6.9 Frequency

6.9.1 Finally, we want to see how many days each user has a recorded event in our data set.

```
Id_Date_Counts <- DailyActivity %>%
  group_by(Date, Id) %>%
  summarise(count = n()) %>%
  group_by(Id) %>%
  summarise(total = sum(count))
```

6.9.1.1 Note: This will give an idea of user frequency over time.

```
## `summarise()` has grouped output by 'Date'. You can override using the `.groups`
## argument.
```

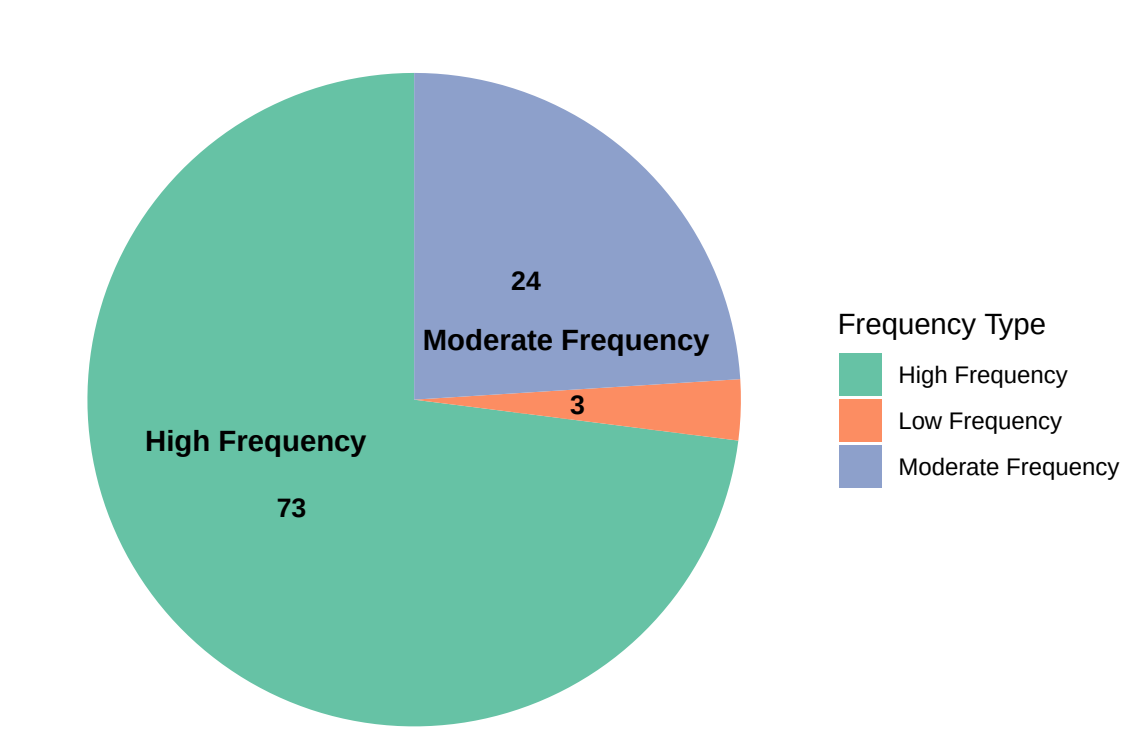
```
Frequency_type <- Id_Date_Counts %>%
  mutate(Frequency_Type = case_when(
      total >= 30 ~ "High Frequency",
      total <= 29 & total >= 15 ~ "Moderate Frequency",
      total <= 14 & total >= 1 ~ "Low Frequency",
      TRUE ~ "No Usage"))

Frequency_percent <- Frequency_type %>%
  group_by(Frequency_Type) %>%
  summarise(count = n()) %>%
  mutate(total = sum(count)) %>%
  group_by(Frequency_Type) %>%
  summarise(Frequency_percent = count / total) %>%
  mutate(Frequency_percent = scales::percent(Frequency_percent, accuracy = 1))

Frequency_percent$Frequency_percent <- as.numeric(gsub("%", "", Frequency_percent$Frequency_percent))

head(Frequency_percent)
```

```
## # A tibble: 3 x 2
##   Frequency_Type     Frequency_percent
##   <chr>                          <dbl>
## 1 High Frequency                    73
## 2 Low Frequency                      3
## 3 Moderate Frequency                24
```

7 Share

7.1 Percent of User Frequency over Time

```
Frequency_percent %>%
    ggplot(aes(x = "", y = Frequency_percent, fill = Frequency_Type)) +
    geom_bar(stat = "identity", width = 2) +
    geom_text(aes(label = Frequency_percent),
              position = position_stack(vjust = 0.5),
              size = 3.5, fontface = "bold") +
    coord_polar("y", start = 0) +
    labs(title = "Percent of User Frequency over Time", fill = "Frequency Type") +
    scale_fill_brewer(palette = "Set2") +
    theme_void() +
  annotate("text", x = 1, y = 71, label = "High Frequency", fontface = "bold") +
  annotate("text", x = 1, y = 19, label = "Moderate Frequency", fontface = "bold")
```



7.1.0.1 Note: Most of FitBit's users are either High Frequency users 73% or Moderate Frequency users 24%. This suggests that it is very likely that customers will have extended use of the device after purchase.

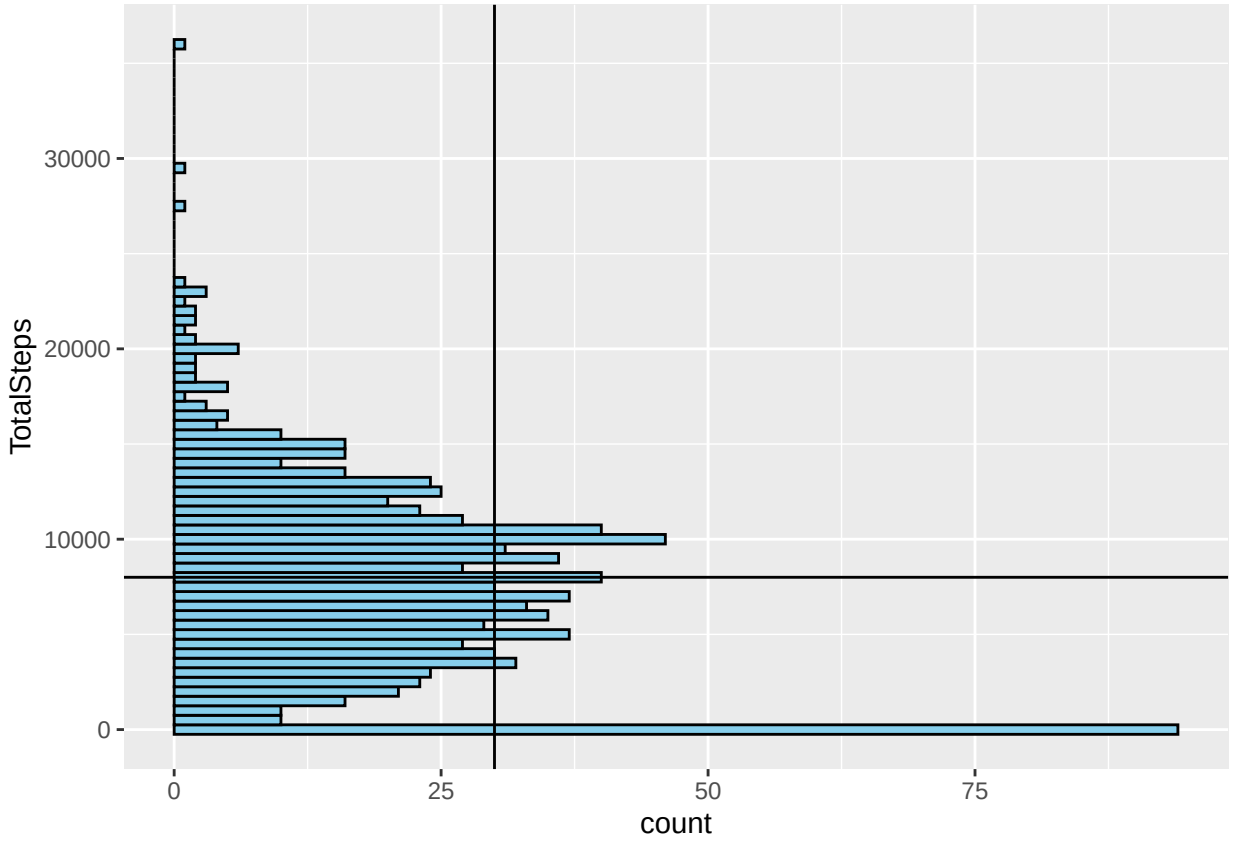
7.1.0.2 Note: We Use Steps as the best indicator of activity being recorded so we should visualize the frequency in Step count recording to get a quick glance at how many steps are most often taken each day.

7.1.0.3 Note: According to Medical News Today, a recent study in 2020 found that participants who took 8,000 steps per day had a 51% lower risk of dying by any cause compared to those who took 4,000 per day.

7.2 Frequency of Step Records

7.2.1 We can use a side ways bar chart to easily visualize this.

```
ggplot(DailyActivity, aes(y = TotalSteps)) +
  geom_histogram(binwidth = 500, fill = "skyblue", color = "black") +
  geom_vline(xintercept = 30) +
  geom_hline(yintercept = 8000)
```




```
  labs(title = "Frequency of Step Records", x = "Frequency", y = "Total Steps")
```

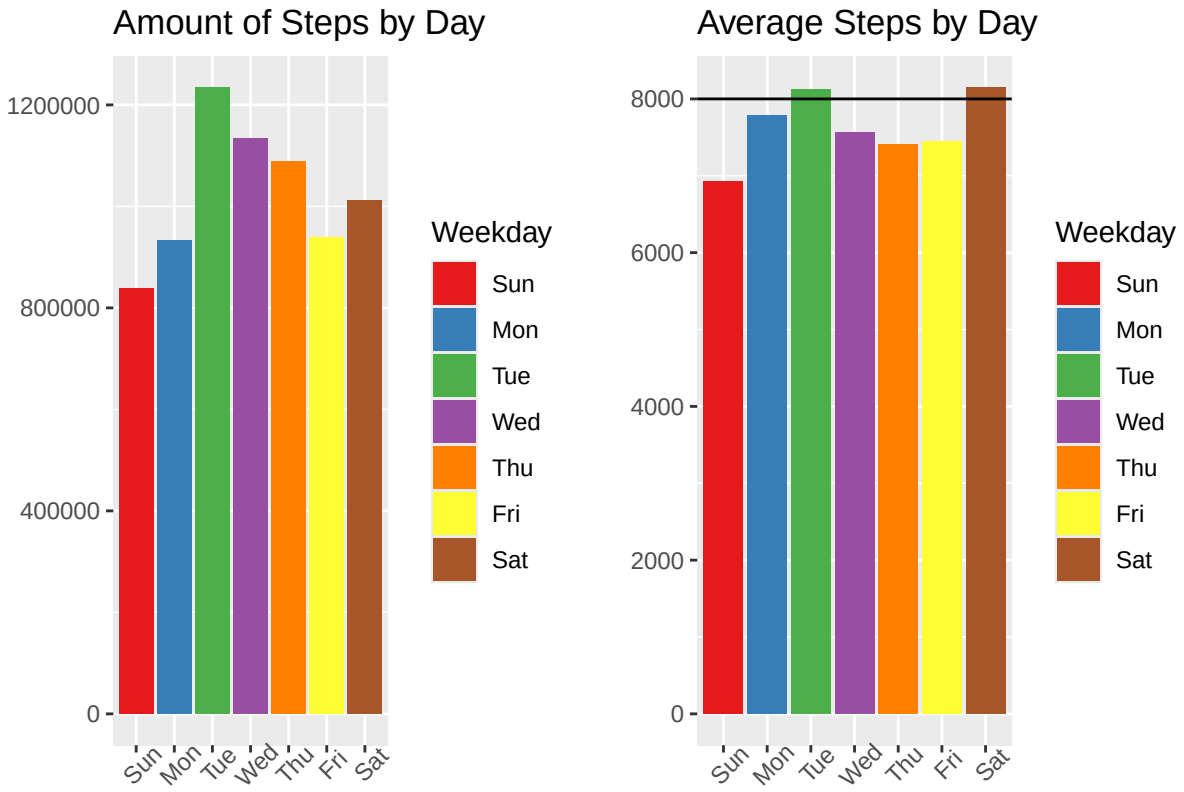
```
## $x
## [1] "Frequency"
##
## $y
## [1] "Total Steps"
##
## $title
## [1] "Frequency of Step Records"
##
```

```
## attr(,"class")
## [1] "labels"
```

7.2.1.1 Note: We can break down the steps taken per day of the week to see if we gain any insights.

7.3 Amount of Steps by Day and Average Steps by Day

```
Daily_steps$Weekday <- factor(Daily_steps$Weekday)

ggarrange(
  ggplot(Daily_steps) +
    geom_col(aes(x = Weekday, y = total_daily_steps, fill = Weekday)) +
    labs(title = "Amount of Steps by Day", x = "", y = "") +
    theme(axis.text.x = element_text(angle = 45)) +
    scale_fill_brewer(palette = "Set1"),
  ggplot(Daily_steps) +
    geom_col(aes(x = Weekday, y = average_daily_steps, fill = Weekday)) +
    geom_hline(yintercept = 8000) +
    labs(title = "Average Steps by Day", x = "", y = "") +
    theme(axis.text.x = element_text(angle = 45)) +
    scale_fill_brewer(palette = "Set1")
)
```



7.3.0.1 Note: We can see that Tuesday, Wednesday, and Thursday are the most active days in terms of total steps taken. While Monday, Tuesday, and Saturday are the most active days in terms of average steps taken per day.

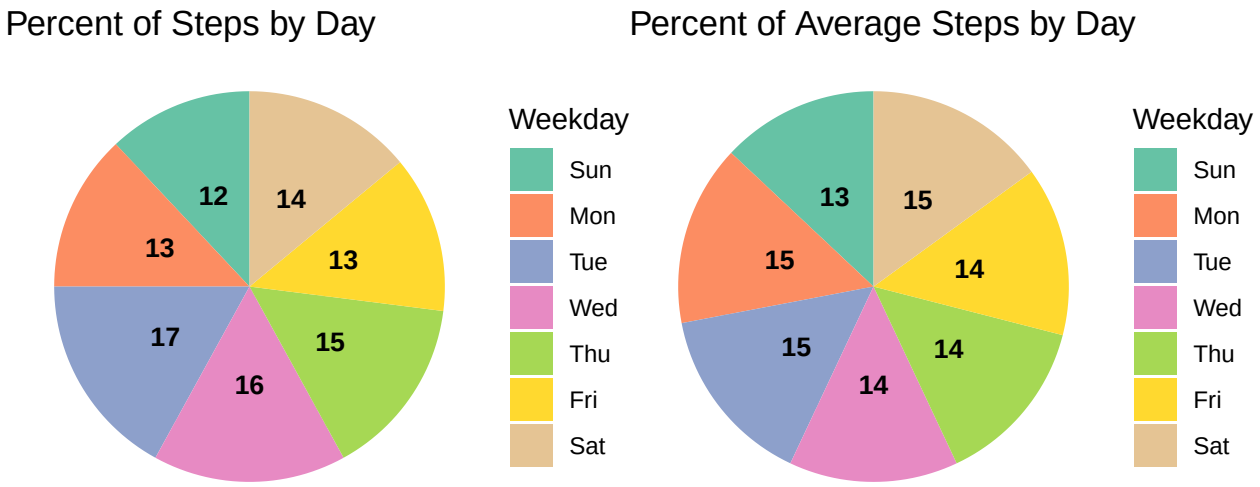
7.3.0.2 Note: Tuesday seems to hold true as the most active day in terms of total steps and average steps. Although it seems that only Tuesday and Saturday pass the recommended amount of 8,000 steps per day.

7.3.1 We can view this breakdown in pie chart form with the corresponding percentage of total and average steps taken on that particular day.

7.3.1.1 Note: This will give us more insights into the difference in days.

7.4 Percent of Average Steps by Day

```
ggarrange(
  Total_step_percent %>%
    ggplot(aes(x = "", y = daily_percent, fill = Weekday)) +
    geom_bar(stat = "identity", width = 2) +
    geom_text(aes(label = daily_percent),
              position = position_stack(vjust = 0.5),
              size = 3.5, fontface = "bold") +
    coord_polar("y", start = 0) +
    labs(title = "Percent of Steps by Day") +
    scale_fill_brewer(palette = "Set2") +
    theme_void(),

  Average_step_percent %>%
    ggplot(aes(x = "", y = average_percent, fill = Weekday)) +
    geom_bar(stat = "identity", width = 2) +
    geom_text(aes(label = average_percent),
              position = position_stack(vjust = 0.5),
              size = 3.5, fontface = "bold") +
    coord_polar("y", start = 0) +
    labs(title = "Percent of Average Steps by Day") +
    scale_fill_brewer(palette = "Set2") +
    theme_void())
```



7.4.0.1 Note: In this visualization we see that although there is a clear difference in the most active days. Tuesday, Wednesday and Thursday at 17%, 16% and 15% for total steps respectively. Also Monday, Tuesday and Saturday at 15% each for average steps.

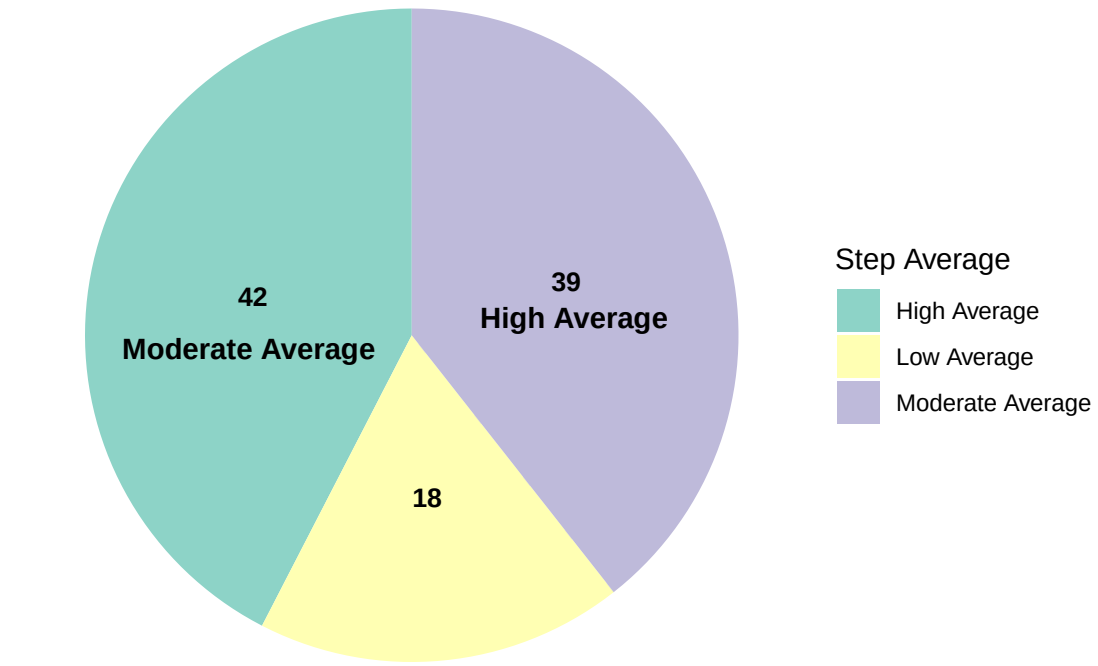
7.4.0.2 Note: Being that the numbers are so close together the Bellabeat stakeholders could use this opportunity to create incentives that will boost activity on Sunday, Wednesday, Thursday and Friday, seeing that increased daily use correlates to increased metrics.

7.4.1 Now let's visualize the breakdown of step Categories we've created for our users.

7.4.1.1 Note: As a reminder our created categories were High Average, Moderate Average and Low average.

7.5 Percent of Users by Step Activity

```
Step_percent %>%
    ggplot(aes(x = "", y = step_percent, fill = step_type)) +
    geom_bar(stat = "identity", width = 2) +
    geom_text(aes(label = step_percent),
              position = position_stack(vjust = 0.5),
              size = 3.5, fontface = "bold") +
    coord_polar("y", start = 0) +
    labs(title = "Percent of Users by Step Activity ", fill ="Step Average") +
    scale_fill_brewer(palette = "Set3") +
    theme_void() +
  annotate("text", x = 1, y = 73, label = "Moderate Average", fontface = "bold") +
  annotate("text", x = 1, y = 23, label = "High Average", fontface = "bold")
```



7.5.0.1 Note: Once again most users are either High Average step (8,000+) users at 39% or Moderate average step (4,000+) users at 42%, which suggest for the most part that our users are far more likely to have a Moderate Average amount of steps.

7.5.0.2 Note: As stated before according to Medical News Today participants that took over 8,000 steps (High Average) were 51% less likely to die from all causes than those who took over 4,000 steps (Moderate average).

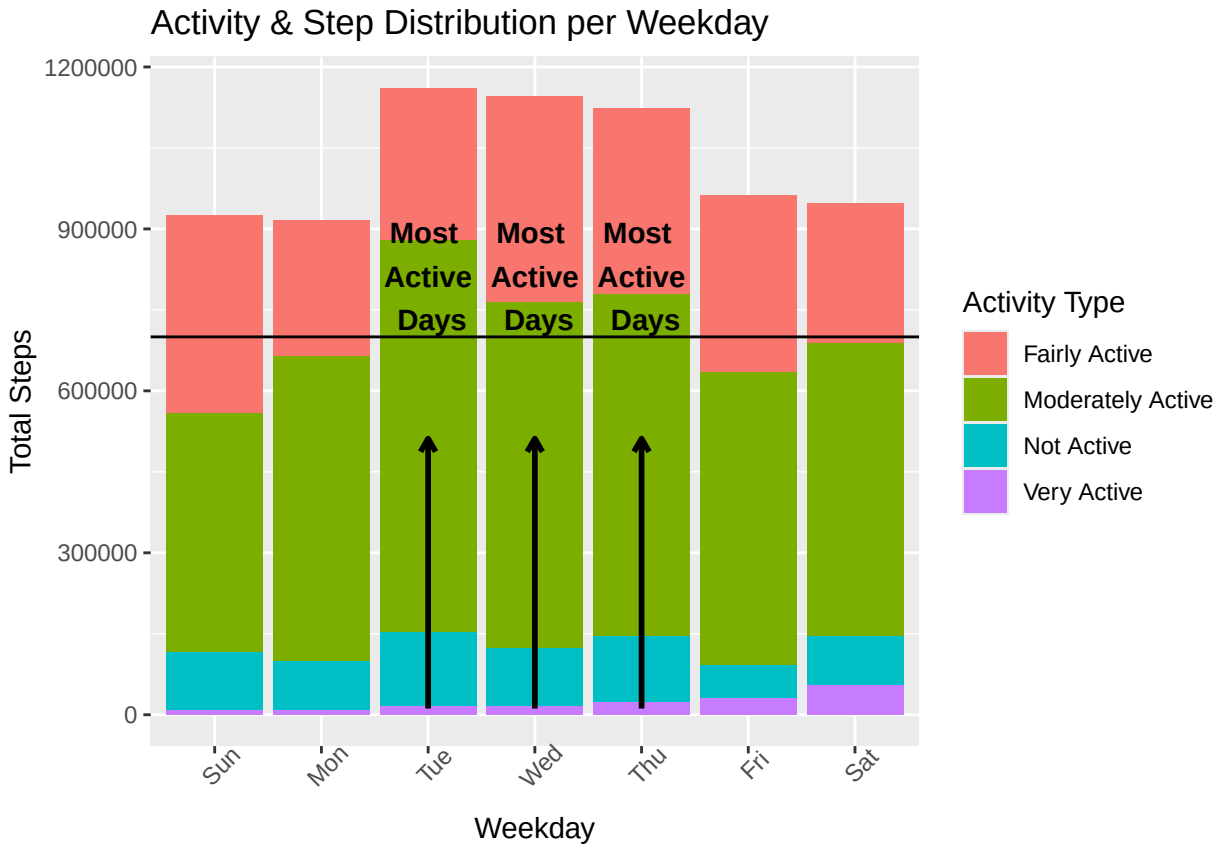
7.5.1 We can also visualize the Step amounts per day with the activity levels to see if activity levels affect the amount of steps taken by our users.

7.5.1.1 Note: Another area where Bellabeat's stakeholders could benefit from is an incentive program to increase users' average step count to over 8,000 steps on days that do not normally reach.

7.6 Activity & Step Distribution per Weekday

```
ggplot(DailyActivity, aes(x = Weekday, y = mean(TotalSteps), fill = activity_type)) +
  geom_bar(stat = "identity", position = "stack") +
  geom_hline(yintercept = 700000) +
  labs(title = "Activity & Step Distribution per Weekday", x = "Weekday", y = "Total Steps", fill = "Ac
  theme(axis.text.x = element_text(angle = 45)) +
  geom_segment(data = head(arrange(DailyActivity, desc(mean(TotalSteps))), 3),
               aes(x = Weekday, xend = Weekday, y = mean(TotalSteps), yend = mean(TotalSteps) + 500000)
```

```
            arrow = arrow(length = unit(0.2, "cm")), color = "black", linewidth = 1) +
  geom_text(data = head(arrange(DailyActivity, desc(mean(TotalSteps))), 3),
            aes(x = Weekday, y = mean(TotalSteps) + 600000, label = "Most \n Active \n Days"),
            vjust = -0.5, hjust = 0.5, size = 4, fontface = "bold", color = "black")
```



7.6.0.1 Note: Surprisingly even though we have a good number of Very Active users, their Very Active minutes do not make up the majority of steps taken throughout the day.

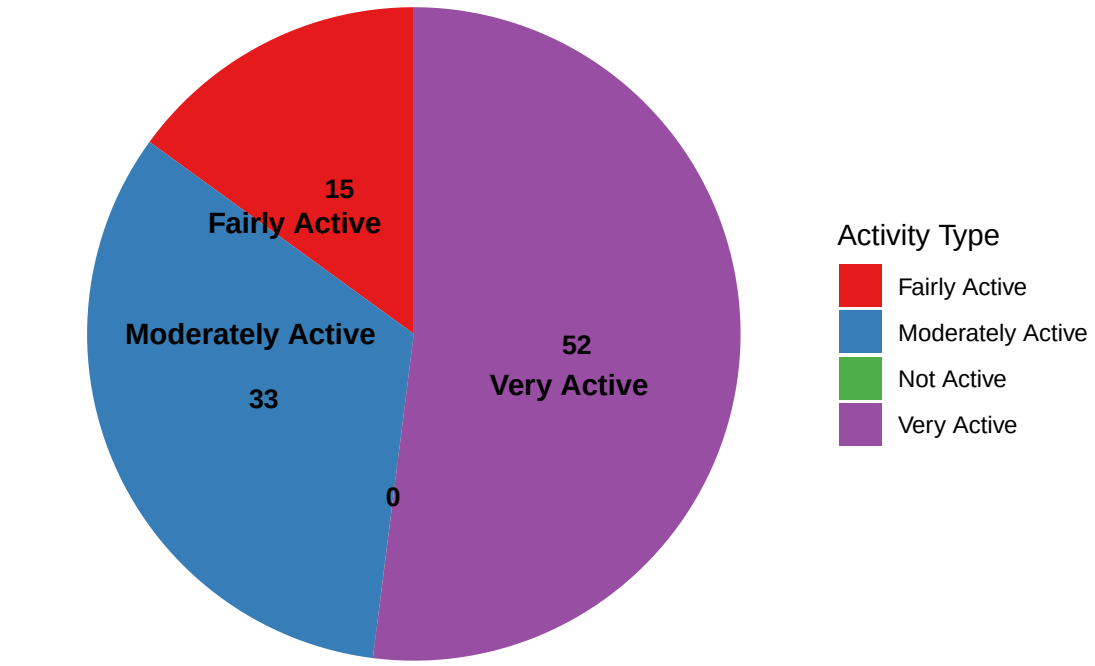
7.6.0.2 Note: The majority of steps taken are actually made up of Moderately Active and Fairly Active Minutes. Stakeholders could use this opportunity to promote extended use of their products during non active minutes throughout the day.

7.7 Percent of Activity

7.7.1 Now let's see what percent of our users fit the Very Active category and what percent fits the Moderately Active category.

```
active_percent %>%
  ggplot(aes(x = "", y = activity_percent, fill = activity_type)) +
  geom_bar(stat = "identity", width = 1) +
  geom_text(aes(label = activity_percent),
            position = position_stack(vjust = 0.5),
```

```
            size = 3.5, fontface = "bold") +
  coord_polar("y", start = 0) +
  labs(title = "Percent of Activity", fill = "Activity Type") +
  scale_fill_brewer(palette = "Set1") +
  theme_void() +
annotate("text", x = 1, y = 75, label = "Moderately Active", fontface = "bold") +
annotate("text", x = 1, y = 30, label = "Very Active", fontface = "bold") +
annotate("text", x = 1, y = 87, label = "Fairly Active", fontface = "bold")
```



7.7.1.1 Note: When looking at the activity time we notice that most users are normally either Moderately active 57% or Fairly active 31% throughout the day. While Very active is the smallest segment at 2%, and there is a 10% chance that at some point throughout the day customers will not be active at all.

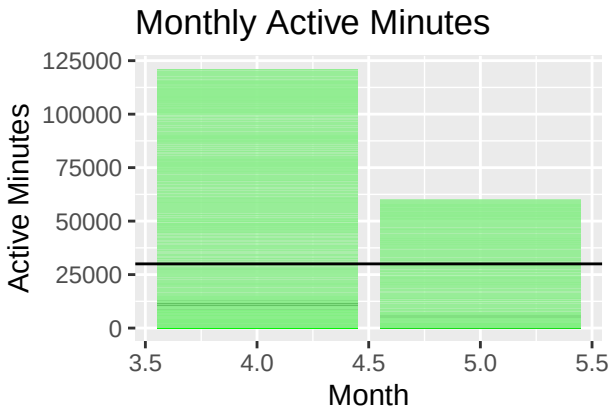
7.7.2 Now we can get a better understanding of how activity minutes were recorded throughout the lifetime of the survey with a bar chart.

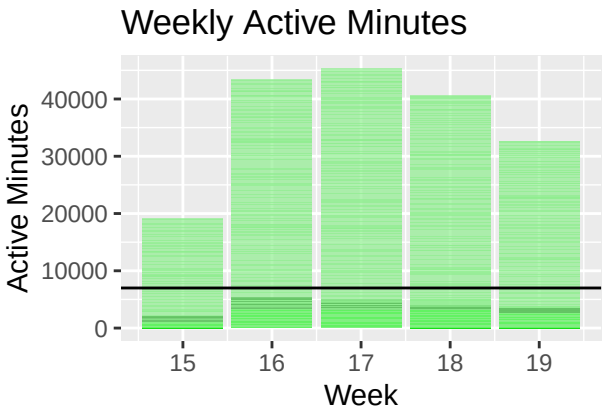
7.7.2.1 Note: According to Mayoclinic.org active persons should aim for at least 30 min of Moderate activity per day.

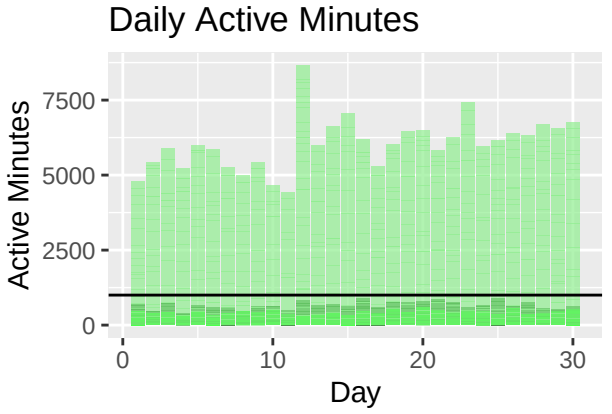
7.8 Daily Active Minutes

```
ggarrange(ggplot(DailyActivity, aes(x = month(Date))) +
  geom_bar(aes(y = VeryActiveMinutes), stat = "identity", fill = "darkgreen", alpha = 0.7) +
```

```
  geom_bar(aes(y = FairlyActiveMinutes), stat = "identity", fill = "green", alpha = 0.7) +
  geom_bar(aes(y = LightlyActiveMinutes), stat = "identity", fill = "lightgreen", alpha = 0.7) +
  geom_hline(yintercept = 30000) +
  labs(title = "Monthly Active Minutes", x = "Month", y = "Active Minutes") +
  scale_alpha_manual(values = c(0.7, 0.7, 0.7)),
  ggplot(DailyActivity, aes(x = week(Date))) +
  geom_bar(aes(y = VeryActiveMinutes), stat = "identity", fill = "darkgreen", alpha = 0.7) +
  geom_bar(aes(y = FairlyActiveMinutes), stat = "identity", fill = "green", alpha = 0.7) +
  geom_bar(aes(y = LightlyActiveMinutes), stat = "identity", fill = "lightgreen", alpha = 0.7) +
  geom_hline(yintercept = 7000) +
  labs(title = "Weekly Active Minutes", x = "Week", y = "Active Minutes") +
  scale_alpha_manual(values = c(0.7, 0.7, 0.7)),
  ggplot(DailyActivity, aes(x = day(Date))) +
  geom_bar(aes(y = VeryActiveMinutes), stat = "identity", fill = "darkgreen", alpha = 0.7) +
  geom_bar(aes(y = FairlyActiveMinutes), stat = "identity", fill = "green", alpha = 0.7) +
  geom_bar(aes(y = LightlyActiveMinutes), stat = "identity", fill = "lightgreen", alpha = 0.7) +
  geom_hline(yintercept = 1000) +
  labs(title = "Daily Active Minutes", x = "Day", y = "Active Minutes") +
  scale_alpha_manual(values = c(0.7, 0.7, 0.7)))
```







7.8.0.1 Note: Dark Green stands for Very active minutes, Regular Green stands for Fairly active minutes and Light Green stands for Lightly active minutes. All added up together gives us our ActiveMinuteCount which we can categorize our users with.

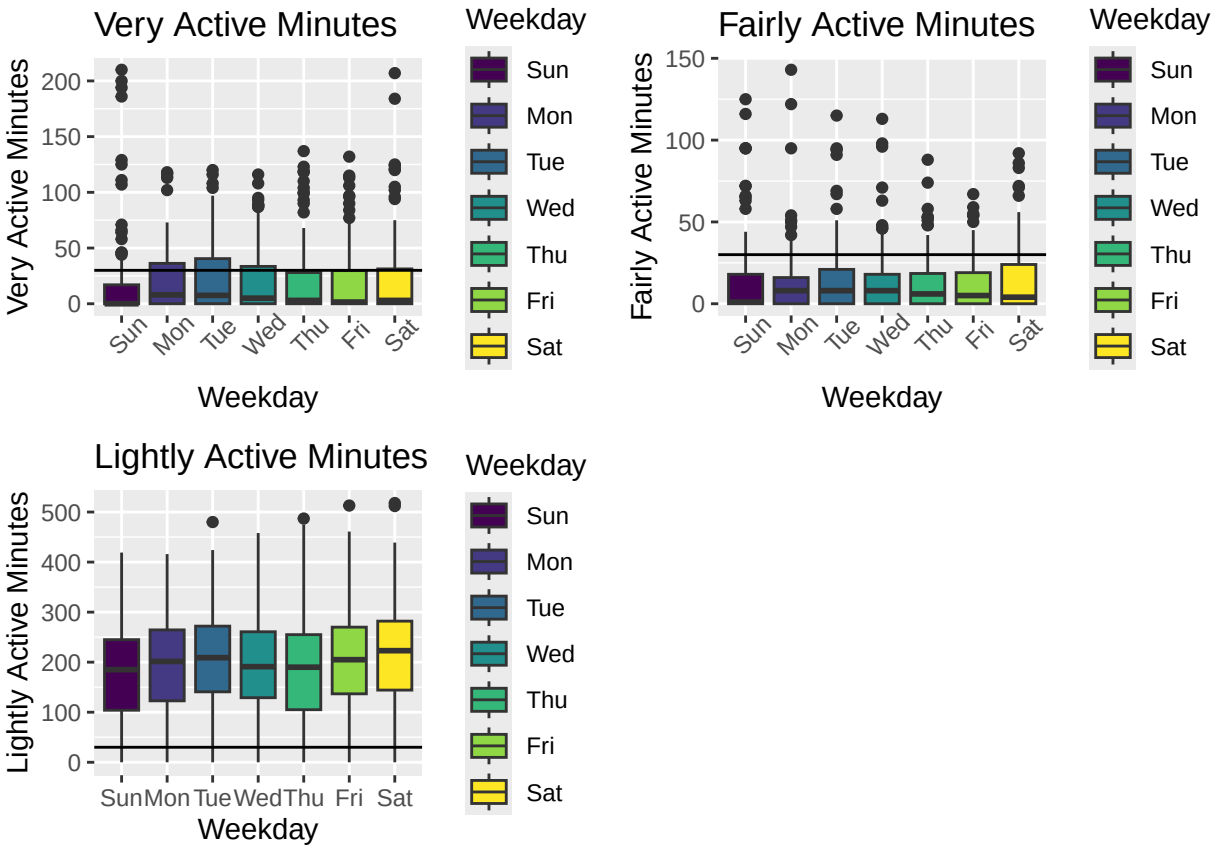
7.8.0.2 Note: It seems that Fairly Active minutes exceed all recommended time average amounts which is 150 minutes per day. Although users do not reach daily, weekly or monthly recommended average times for Moderately Active minutes and Very Active minutes.

7.8.0.3 Stakeholders could use this as an opportunity to incentivize users to increase their Moderately and Very Active minutes. Also they could use this to promote the extended use of their products because the data suggest simply using the devices promotes better health standards.

7.8.1 Let's break this chart down, one for each segment using a histogram so we can get an understanding if there are outlying entries also.

7.9 Active Minutes

```
ggarrange(ggplot(DailyActivity, aes(x = Weekday, y = VeryActiveMinutes, fill = Weekday)) +
  geom_boxplot() +
    geom_hline(yintercept = 30) +
  labs(title = "Very Active Minutes", x = "Weekday", y = "Very Active Minutes") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(DailyActivity, aes(x = Weekday, y = FairlyActiveMinutes, fill = Weekday)) +
  geom_boxplot() +
    geom_hline(yintercept = 30) +
  labs(title = "Fairly Active Minutes", x = "Weekday", y = "Fairly Active Minutes") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(DailyActivity, aes(x = Weekday, y = LightlyActiveMinutes, fill = Weekday)) +
  geom_boxplot() +
    geom_hline(yintercept = 30) +
  labs(title = "Lightly Active Minutes", x = "Weekday", y = "Lightly Active Minutes")) +
  theme(axis.text.x = element_text(angle = 45))
```

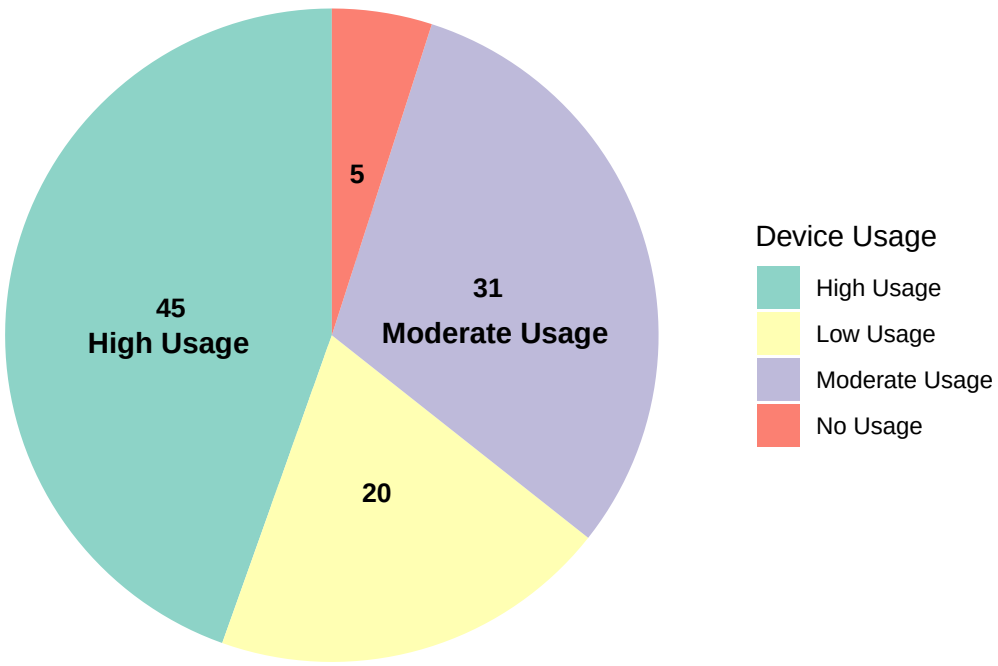


7.9.0.1 Note: As we look closer to the breakdown of daily activity we can clearly see Lightly Active minutes greatly exceed the recommended amounts. Fairly Active minutes never meet the recommended amounts and Very Active minutes almost reach recommended amounts except on Monday, Tuesday and Wednesday where it exceeds recommendations.

7.9.1 Now let's visualize the total time worn breakdown to see if we can gain insights on users Bellabeat's stakeholders could leverage.

7.10 Device Usage

```
worn_percent %>%
   ggplot(aes(x = "", y = worn_percent, fill = worn_type)) +
   geom_bar(stat = "identity", width = 1) +
   geom_text(aes(label = worn_percent),
             position = position_stack(vjust = 0.5),
             size = 3.5, fontface = "bold") +
   coord_polar("y", start = 0) +
   labs(title = "Device Usage", fill = "Device Usage") +
   scale_fill_brewer(palette = "Set3") +
   theme_void() +
 annotate("text", x = 1, y = 75, label = "High Usage", fontface = "bold") +
 annotate("text", x = 1, y = 25, label = "Moderate Usage", fontface = "bold")
```



7.10.0.1 Note: In this final pie chart we can see that we have a very small group of customers that do not use their device or use it for a very small amount of time at 5%, based on mins. While most customers have a Moderate worn time usage amount or a High worn time amount at 31% and 45% respectively.

7.10.0.2 Note: Bellabeat stakeholders can leverage Low usage users being that 20% of users have a low device usage rate.

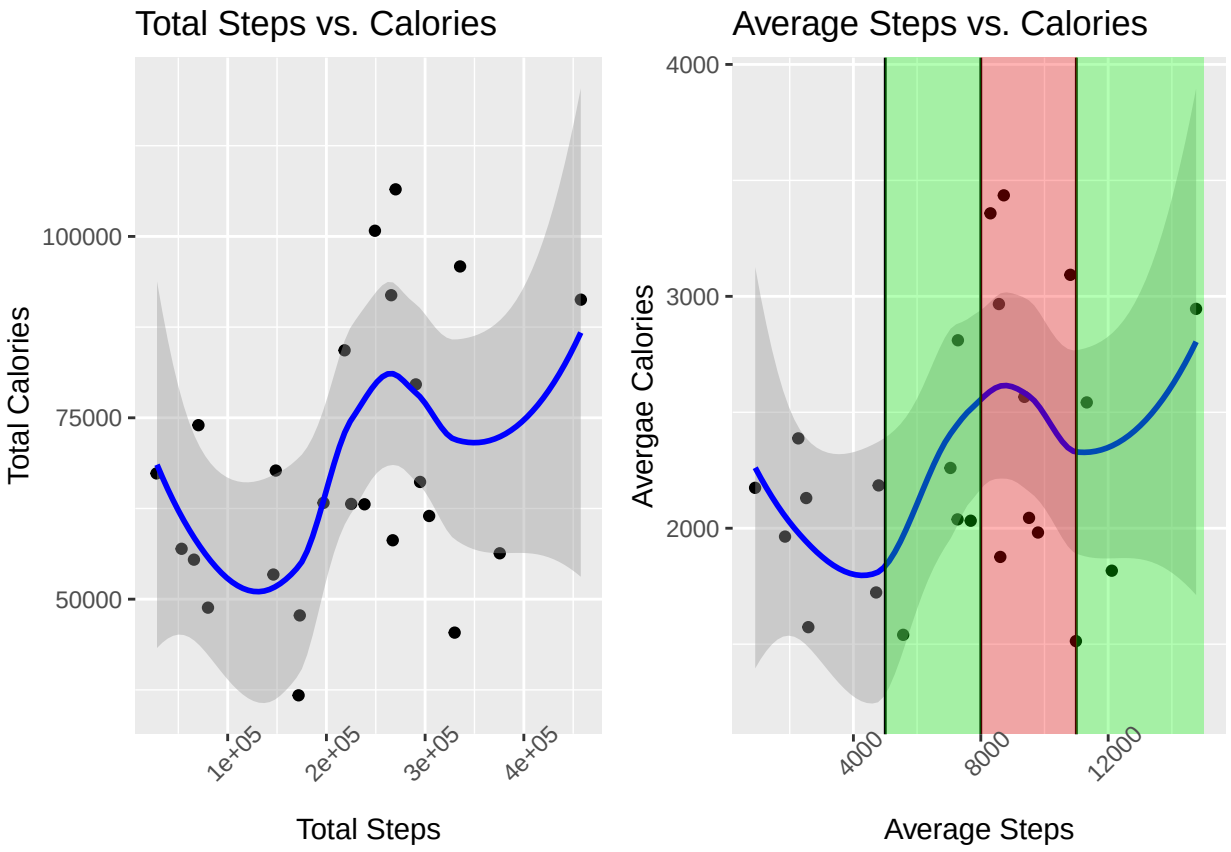
7.10.1 Now that we have a better idea of the consistency of steps recorded let's see if steps taken.

7.11 Total Steps vs. Calories

```
ggarrange(ggplot(DailyActivitywithSleep, aes(x = TotalStepCount, y = TotalCalories)) +
  geom_jitter() +
  geom_smooth(color = "blue") +
  labs(title = "Total Steps vs. Calories", x = "Total Steps", y = "Total Calories") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(AverageActivitywithSleep, aes(x = mean_totalsteps, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "blue") +
    geom_vline(xintercept = 5000) +
    geom_vline(xintercept = 8000) +
    geom_vline(xintercept = 11000) +
  annotate("rect", xmin = 5000, xmax = 8000, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
```

```
  annotate("rect", xmin = 8000, xmax = 11000, ymin = -Inf, ymax = Inf, fill = "red", alpha = 0.3) +
  annotate("rect", xmin = 11000, xmax = 15000, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "Average Steps vs. Calories", x = "Average Steps", y = "Avergae Calories") +
  theme(axis.text.x = element_text(angle = 45)))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



7.11.0.1 Note: Looking at this graph we can see that there is a strong positive relationship between the steps taken to calories burned. Especially from above 5,000-8,000 steps on average.

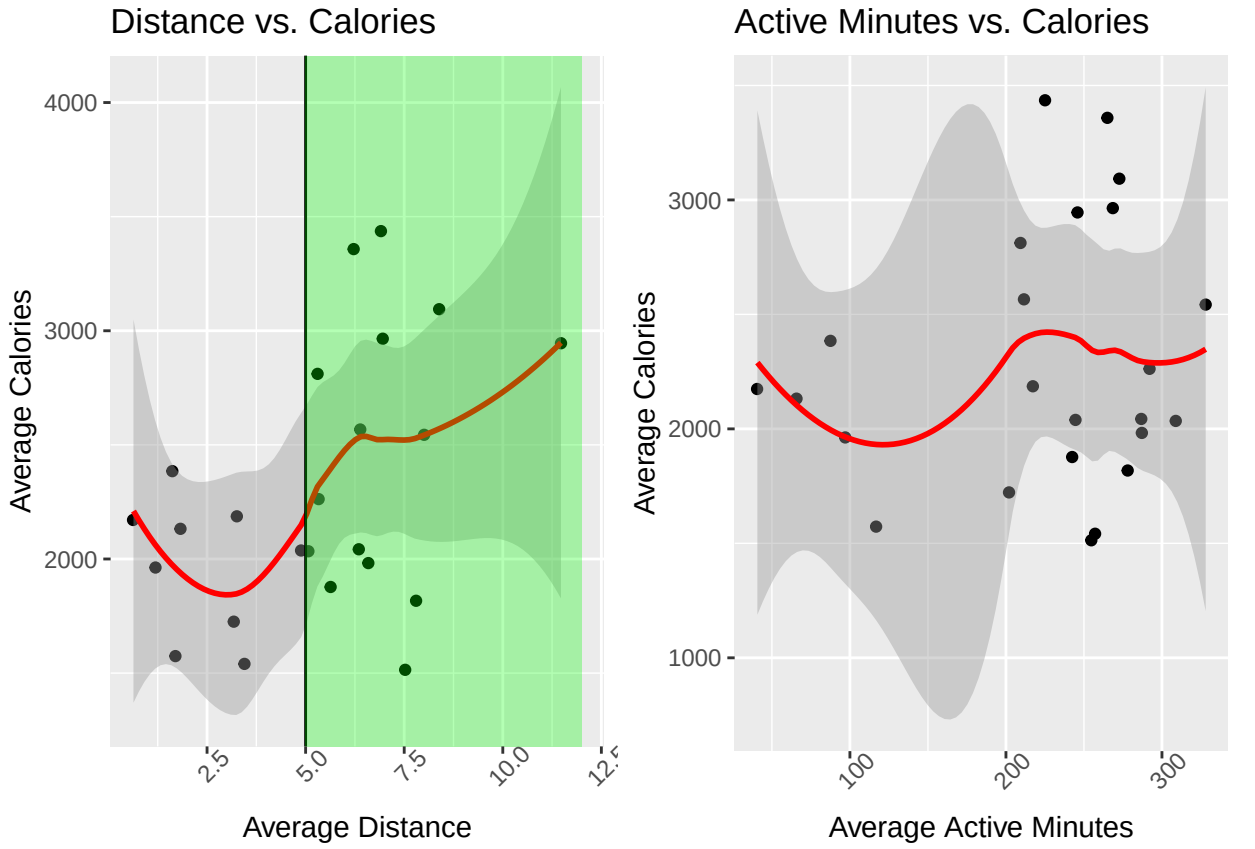
7.11.0.2 Note: there is a slight downtick from 8,000 steps to 11,000 steps but a retrace to an uptick after 11,000 steps. Bellabeat stakeholders could use this opportunity to incentivise users that are already very active to increase their very active minutes to reach this 11,000 step goal. This would increase usage in Bellabeat's products in two areas.

7.12 Distance vs. Calories

7.12.1 Let's see if any other factors have a positive relationship to calories burned.

```
ggarrange(ggplot(AverageActivitywithSleep, aes(x = mean_totaldistance, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
    geom_vline(xintercept = 5) +
  annotate("rect", xmin = 5, xmax = 12, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "Distance vs. Calories", x = "Average Distance", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(AverageActivitywithSleep, aes(x = mean_activeminutes, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
  labs(title = "Active Minutes vs. Calories", x = "Average Active Minutes", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



7.12.1.1 Note: We can notice that average distance traveled has a very strong positive relationship with calories burned, seeing a positive uptick right after the recommended amount of 5 miles per day.

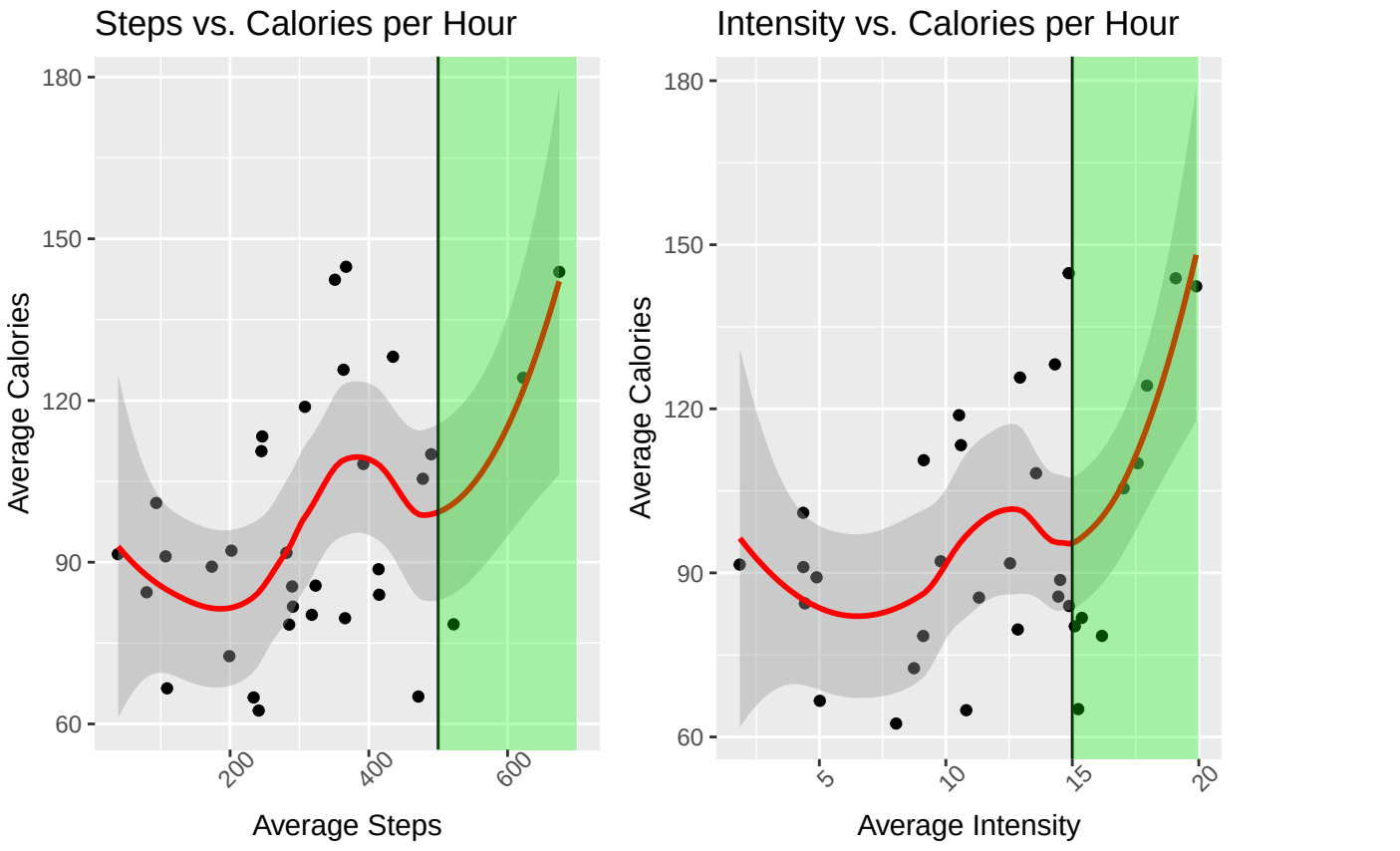
7.12.1.2 Note: We also notice that average active minutes does not have a real positive effect seeing that the amount of calories burned stays around the same throughout the amounts of active minutes.

7.13 Steps vs. Calories per Hour

7.13.1 Let's see if metrics in our hourly and activity by the minute data frames also show positive relationships to calories burned.

```
ggarrange(ggplot(Average_HourlyActivity, aes(x = mean_steps, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
    geom_vline(xintercept = 500) +
  annotate("rect", xmin = 500, xmax = 700, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "Steps vs. Calories per Hour", x = "Average Steps", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(Average_HourlyActivity, aes(x = mean_intensity, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
    geom_vline(xintercept = 15) +
  annotate("rect", xmin = 15, xmax = 20, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "Intensity vs. Calories per Hour", x = "Average Intensity", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



7.13.1.1 Note: We see a very strong positive relationship between Average Steps per hour after 500, which equals around 12,000 steps for a 24 hour period.

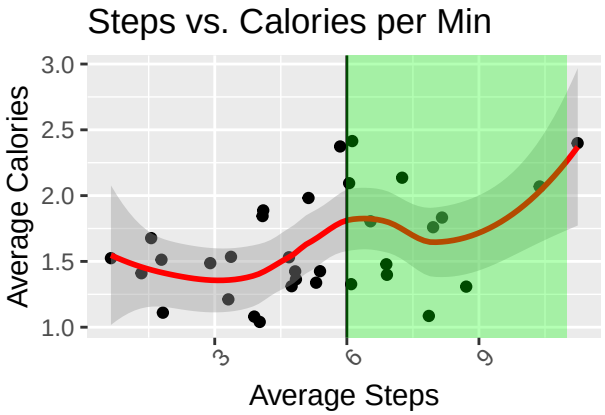
7.13.1.2 Note: Also we can see there is a strong positive relationship between Average intensity per hour after 15, which equals around 360 METs for a 24 hr period.

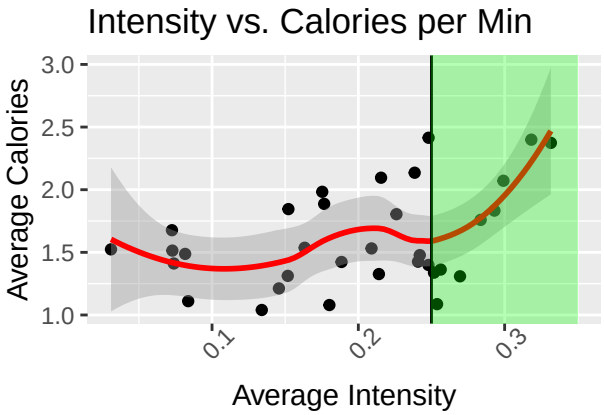
7.13.2 Now we'll check metrics of our Activity by the minute, these metrics include `steps`, `intensity` and `METs`.

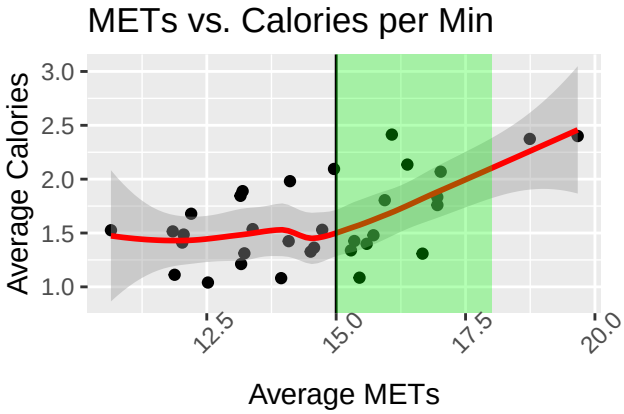
7.14 Steps, Intensity and METS vs. Calories per Min

```
ggarrange(ggplot(Average_ActivitybyMinute, aes(x = mean_steps, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
    geom_vline(xintercept = 6) +
  annotate("rect", xmin = 6, xmax = 11, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "Steps vs. Calories per Min", x = "Average Steps", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(Average_ActivitybyMinute, aes(x = mean_intensity, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
    geom_vline(xintercept = 0.25) +
  annotate("rect", xmin = 0.25, xmax = 0.35, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "Intensity vs. Calories per Min", x = "Average Intensity", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(Average_ActivitybyMinute, aes(x = mean_METs, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "red") +
    geom_vline(xintercept = 15) +
  annotate("rect", xmin = 15, xmax = 18, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  labs(title = "METs vs. Calories per Min", x = "Average METs", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45)))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```







7.14.0.1 Note: We see a positive relationship between average steps per minute and calories burned after around 6 steps per minute which equals 360 steps per hour, which is around 9,000 steps per day. Above our recommended amount of 8,000.

7.14.0.2 Note: We see that the average intensity count of 0.25 per minute also has a positive relationship with calories burned. 0.25 per minute equals 15 per hour, 360 per day.

7.14.0.3 Note: Finally we see that an average MET count of 15 per minute gives us a positive relationship to calories burned. 15 per min = 900 per hr = 22,000 per day.

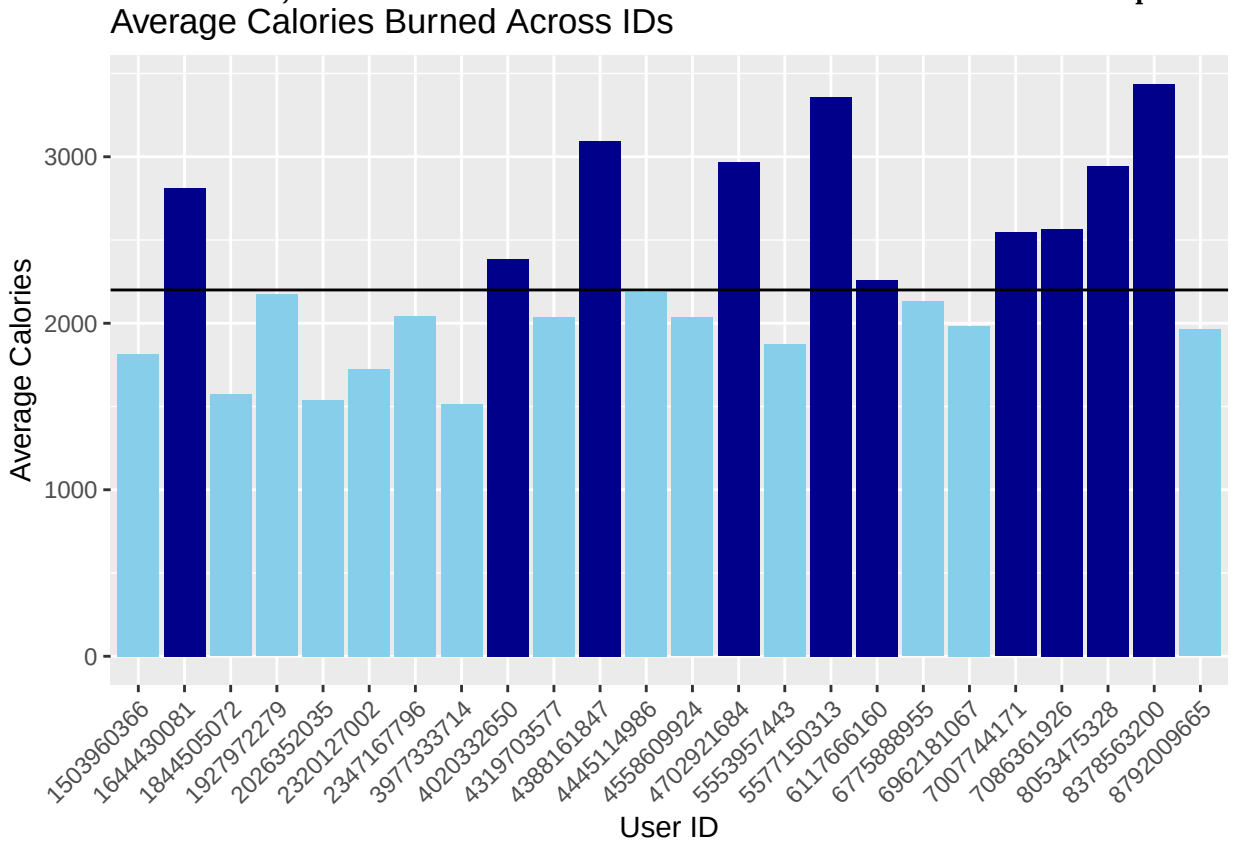
7.14.0.4 Note: Giving positive reinforcement in the terms of notifications, achievements and awards would be a good way to engage customers and encourage them to continue the pace of these checkmarks to reach recommended goals.

7.14.1 In this data Steps taken is the way activity is measured toward the ultimate goal of weight loss through daily calories burned. This is why we have been using calories burned as our variable for positive relationships.

7.15 Average Calories Burned Across IDs

```
ggplot(AverageActivitywithSleep, aes(x = factor(Id), y = mean_calories)) +
  geom_bar(stat = "identity", aes(fill = ifelse(mean_calories > 2200, "DarkBlue", "SkyBlue"))) +
  geom_hline(yintercept = 2200) +
  labs(title = "Average Calories Burned Across IDs", x = "User ID", y = "Average Calories") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_manual(values = c("SkyBlue" = "skyblue", "DarkBlue" = "darkblue")) +
  guides(fill = "none")
```

7.15.0.1 Note: According to everyday health adult women should burn 1,600-2,400 calories per day while adult men should aim to burn 2,000-3,000 calories per day so we will use 2,200 as our recommended amount of calories to burn per day.



7.15.0.2 Note: The dark blue bars are the users that average over 2200 calories per day, while the sky blue bar are users that average under 2200 calories per day.

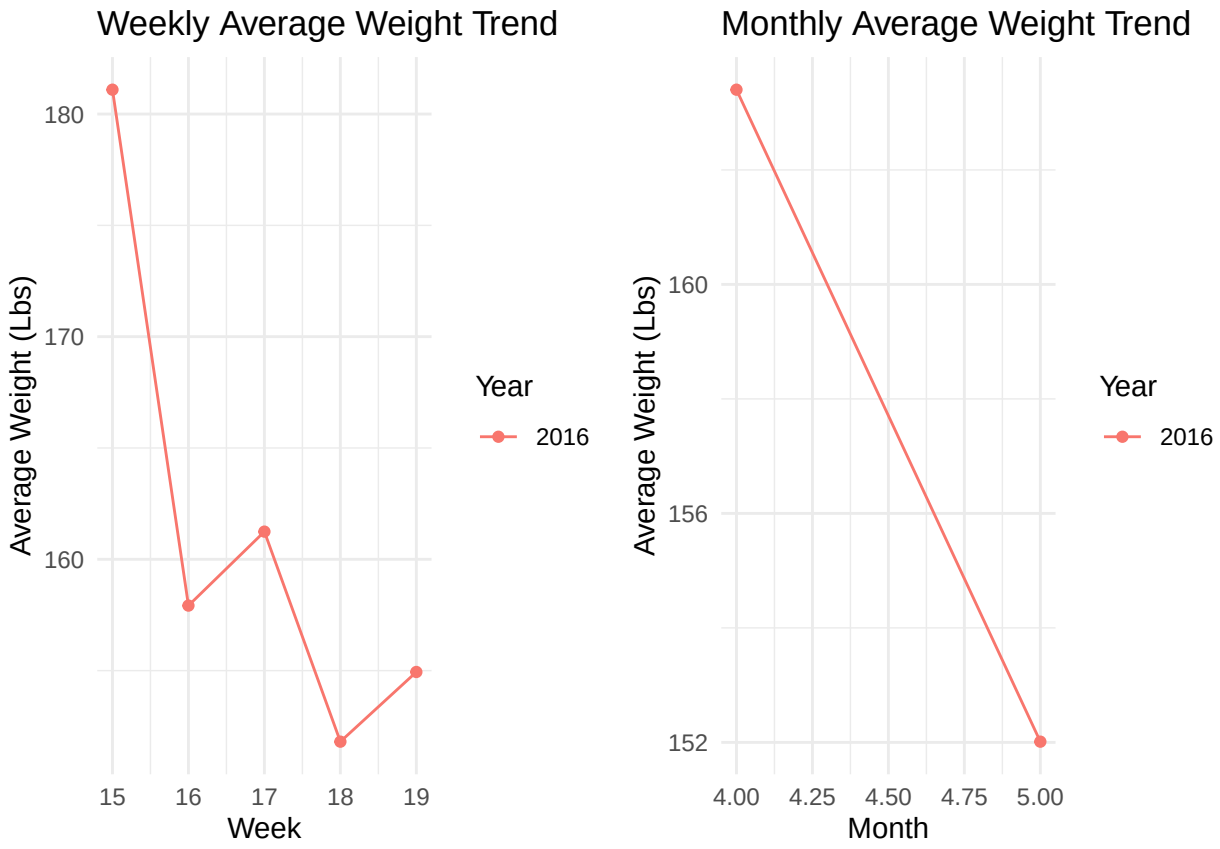
7.15.0.3 Note: Bellabeat stakeholders should target users that pass this recommended mark with achievements, incentives and offers if they pass for extended periods of time. While also targeting users that do not have incentives for reaching these goals, then positive reinforcement to reach them multiple times and finally incentives / offers for passing goals for extended periods of time.

7.15.1 An area where Bellabeat stakeholders could take advantage of where FitBit lacks, is in the recording of weight loss data. If there is a way to ensure regular daily, accurate reporting Bellabeat could gain market share from FitBit customers.

7.15.1.1 Note: Although we know there is a sampling bias due to there only being complete and usable data for only 2 of our users we can calculate it and see if we get anything.

7.16 Weekly and Monthly Average Weight Trend

```
ggarrange(ggplot(weekly_avg, aes(x = Week, y = AvgWeight, group = Year, color = as.factor(Year))) +
  geom_line() +
  geom_point() +
  labs(title = "Weekly Average Weight Trend", x = "Week", y = "Average Weight (Lbs)", color = "Year") +
  theme_minimal(),
  ggplot(monthly_avg, aes(x = Month, y = AvgWeight, group = Year, color = as.factor(Year))) +
  geom_line() +
  geom_point() +
  labs(title = "Monthly Average Weight Trend", x = "Month", y = "Average Weight (Lbs)", color = "Year")
  theme_minimal())
```



7.16.0.1 Note: Even though the usable data is only for 2 users we do see a decrease in average weight loss per week and per month.

7.16.0.2 Note: Bellabeat should also have a more extended period of time for recording. A 6 to 12 month recording period is better for more accurate readings on weight loss.

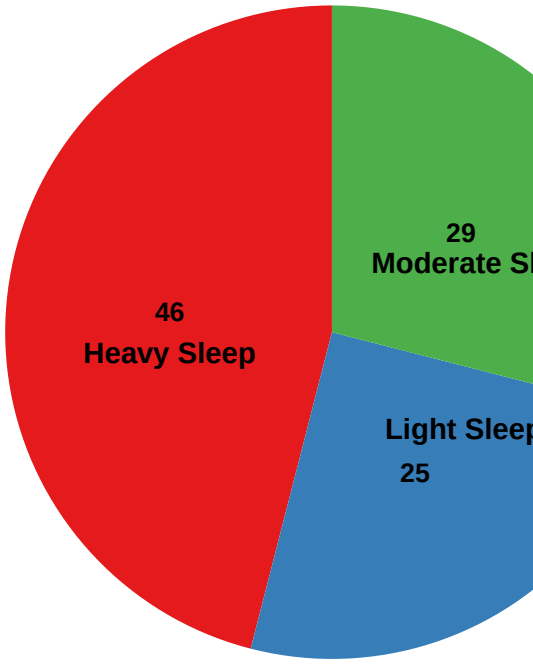
7.16.0.3 Note: Bellabeat should allow users to enter weight loss or gain goals in a small survey during sign-up. Then offer bi-annual notifications for weight loss or gains and annual achievements and offers for reaching their goals.

7.17 User Sleep Categories

7.17.1 Our final set of data indicates sleep records, total time asleep and minutes in bed. Let's first see what percent of our Users are Heavy, Moderate and Light sleepers based on minutes asleep.

```
Sleep_percent %>%
   ggplot(aes(x = "", y = sleep_percent, fill = sleep_type)) +
   geom_bar(stat = "identity", width = 1) +
   geom_text(aes(label = sleep_percent),
             position = position_stack(vjust = 0.5),
             size = 3.5, fontface = "bold") +
   coord_polar("y", start = 0) +
   labs(title = "User Sleep Categories") +
   scale_fill_brewer(palette = "Set1") +
 annotate("text", x = 1, y = 73, label = "Heavy Sleep", fontface = "bold") +
 annotate("text", x = 1, y = 18, label = "Moderate Sleep", fontface = "bold") +
 annotate("text", x = 1, y = 35, label = "Light Sleep", fontface = "bold") +
   theme_void()
```

7.17.1.1 Note: According to Mayoclinic.org the recommended amount of sleep for adults is 7 or more hours of sleep, which equates to around 420 minutes.

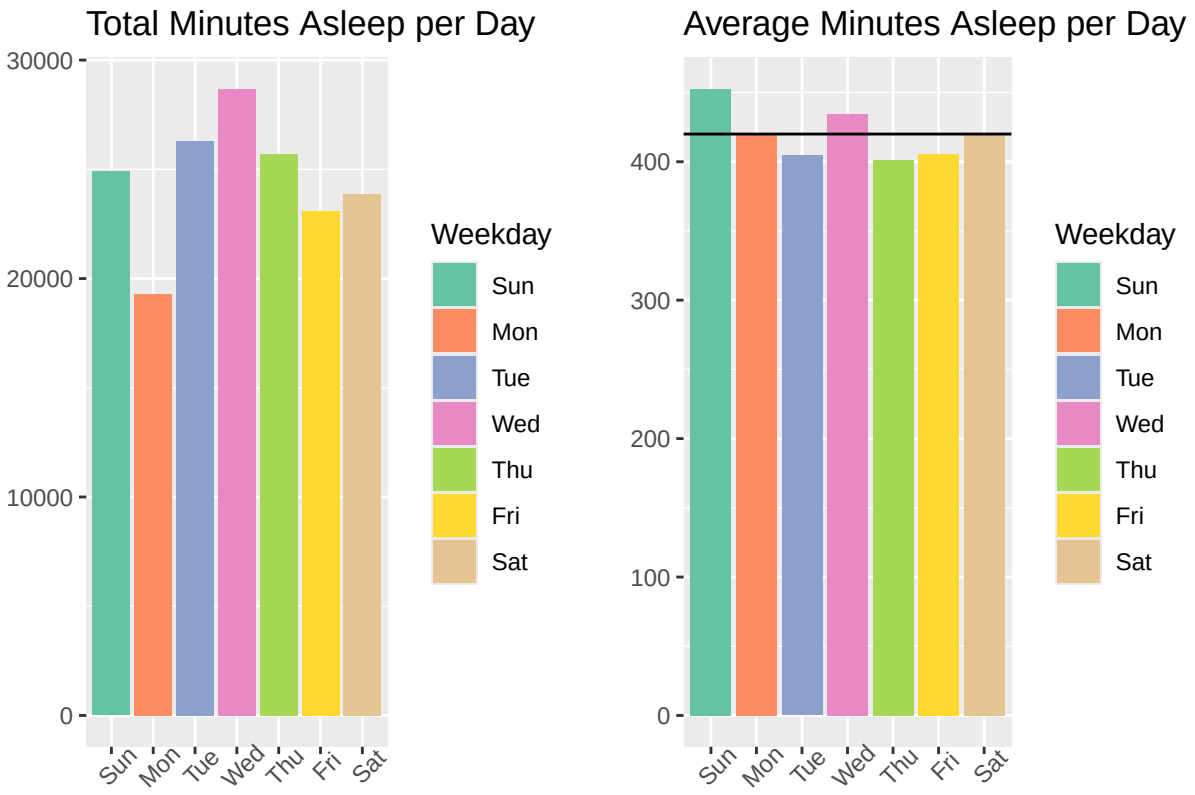


7.17.1.2 Note: We can see that Most of our users are Heavy sleepers (> 420 mins) at 46% followed by Moderately heavy sleepers (300-420 mins) at 29% and finally closely followed by Light sleepers (< 300) at 25%.

7.18 Minutes Asleep per Day

7.18.1 Now let's get a breakdown of time asleep per day.

```
Daily_Sleep_Count$Weekday <- factor(Daily_Sleep_Count$Weekday)

ggarrange(
  ggplot(Daily_Sleep_Count) +
    geom_col(aes(x = Weekday, y = total_minutes_asleep, fill = Weekday)) +
    labs(title = "Total Minutes Asleep per Day", x = "", y = "") +
    theme(axis.text.x = element_text(angle = 45)) +
    scale_fill_brewer(palette = "Set2"),
  ggplot(Daily_Sleep_Count) +
    geom_col(aes(x = Weekday, y = average_minutes_asleep, fill = Weekday)) +
    geom_hline(yintercept = 420) +
    labs(title = "Average Minutes Asleep per Day", x = "", y = "") +
    theme(axis.text.x = element_text(angle = 45)) +
    scale_fill_brewer(palette = "Set2")
)
```



7.18.1.1 Note: We can see that Wednesday, Tuesday and Thurs4day are the sleepiest days in terms of total minutes asleep. While Sunday, Wednesday, Saturday and Monday are the sleepiest days in terms of average minutes asleep per day.

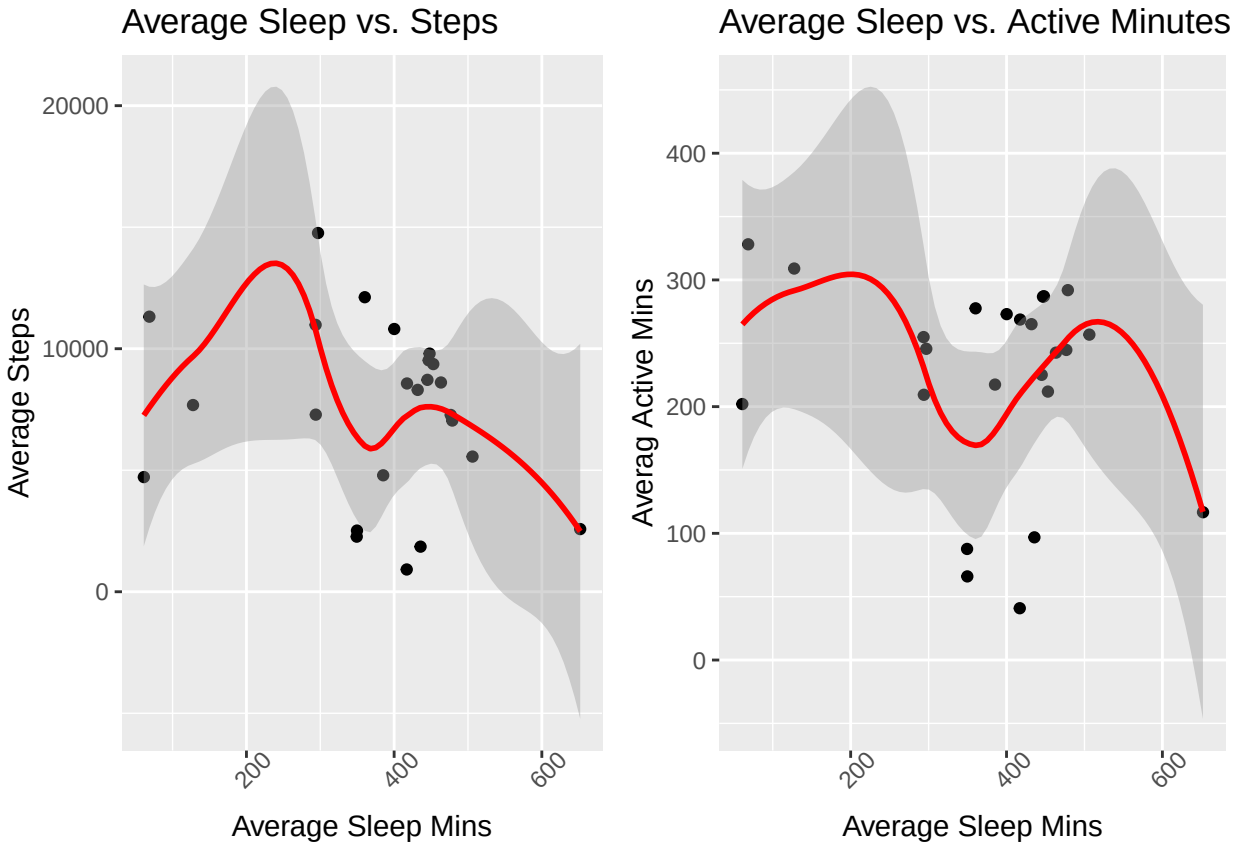
7.18.1.2 Note: Although it seems that only Sunday and Wednesday exceed the recommended amount of 420 minutes of sleep. While Monday and Saturday seem to just barely make the recommended mark.

7.19 Sleep vs.

7.19.1 Let's see if we can get a positive correlation between either sleep and steps or sleep and active mins.

```
ggarrange(ggplot(AverageActivitywithSleep, aes(x = AverageAsleepMinutes, y = mean_totalsteps)) +
  geom_jitter() +
  geom_smooth(color = "red") +
  labs(title = "Average Sleep vs. Steps", x = "Average Sleep Mins", y = "Average Steps") +
  theme(axis.text.x = element_text(angle = 45)),
  ggplot(AverageActivitywithSleep, aes(x = AverageAsleepMinutes, y = mean_activeminutes)) +
  geom_jitter() +
  geom_smooth(color = "red") +
  labs(title = "Average Sleep vs. Active Minutes", x = "Average Sleep Mins", y = "Averag Active Mins") +
  theme(axis.text.x = element_text(angle = 45)))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



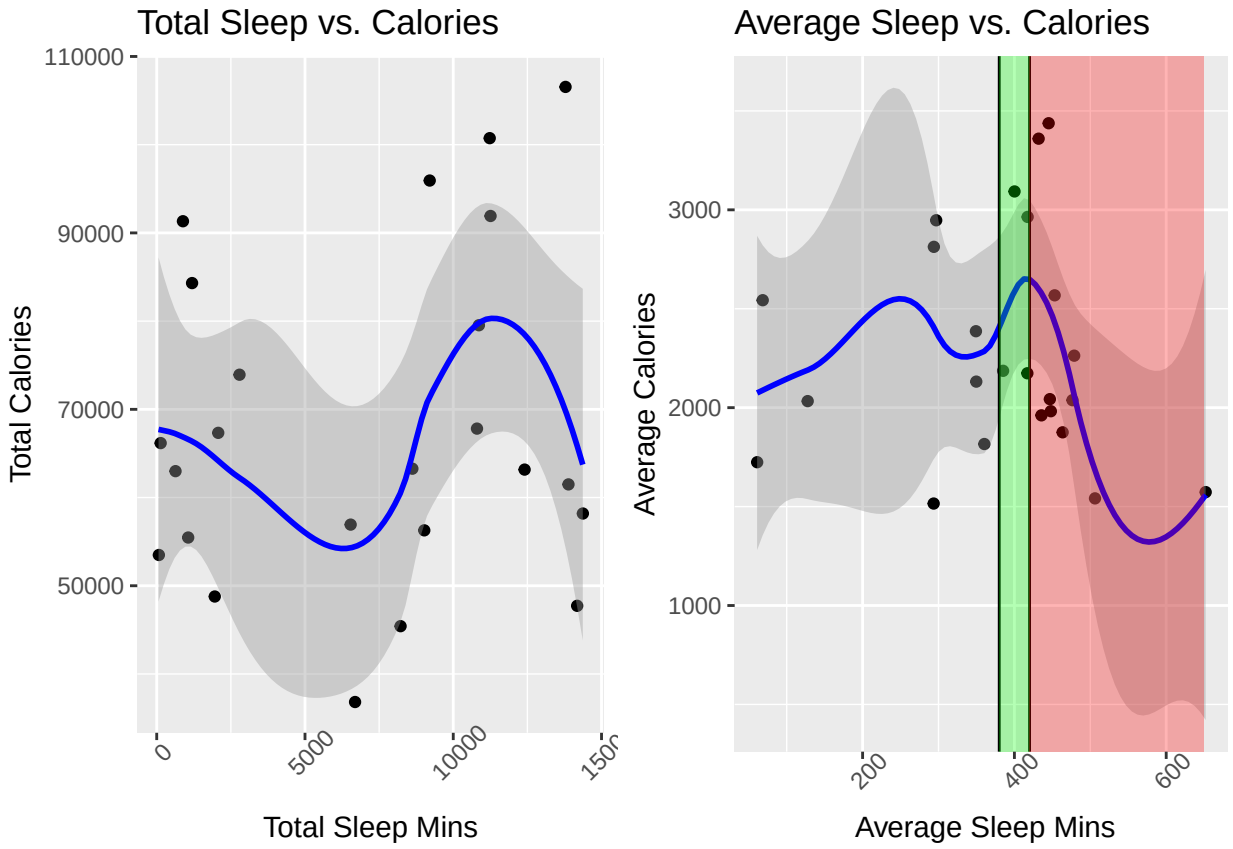
7.19.1.1 Note: We see a negative relationship between average sleep mins and average steps taken or average active minutes throughout the lifetime of these charts.

7.19.2 Finally let's see if the total or average amount of sleep mins has a positive or negative correlation with calories burned.

```
ggarrange(
  ggplot(DailyActivitywithSleep, aes(x = TotalAsleepMinutes, y = TotalCalories)) +
    geom_jitter() +
    geom_smooth(color = "blue") +
    labs(title = "Total Sleep vs. Calories", x = "Total Sleep Mins", y = "Total Calories") +
    theme(axis.text.x = element_text(angle = 45)),
  ggplot(AverageActivitywithSleep, aes(x = AverageAsleepMinutes, y = mean_calories)) +
  geom_jitter() +
  geom_smooth(color = "blue") +
  geom_vline(xintercept = 380) +
  geom_vline(xintercept = 420) +
  annotate("rect", xmin = 380, xmax = 420, ymin = -Inf, ymax = Inf, fill = "green", alpha = 0.3) +
  annotate("rect", xmin = 420, xmax = 650, ymin = -Inf, ymax = Inf, fill = "red", alpha = 0.3) +
  labs(title = "Average Sleep vs. Calories", x = " Average Sleep Mins", y = "Average Calories") +
```

```
  theme(axis.text.x = element_text(angle = 45))
)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



7.19.2.1 Note: There seems to be a slight positive relationship between sleep and calories burned right around the recommended amount of 420 mins of sleep per day.

7.19.2.2 Note: While there actually appears to be a steep drop off to a negative relationship after 420 mins of sleep.

7.19.2.3 Note: Bellabeat's stakeholders would benefit from setting up an alarm system either through the app or directly through the device because data suggest strict schedules for positive relationships.

8 Act

8.1 Opportunities: Bellabeat App, Leaf, Time & Spring

8.1.0.1 Existing Customers: For already existing customers, Bellabeat stakeholders should target Moderate and Low Frequency users with notifications and achievements for extended

use to get those customers excited and engaged with positive reinforcement on the use of Bellabeat's products.

8.1.0.2 Active Minutes: Customers should receive notifications and achievements for reaching recommended goals for activity minutes. According to Mayoclinic.org that would be 120 minutes of Light Activity per day, 30 minutes of Moderate Activity per day and 10 min of Intensive Activity per day. After reaching an extended time of use, such as 3-6 months. Bellabeat should offer these customers discounted prices for Bellabeat Spring products.

8.1.0.3 Steps: Customers should receive notifications and achievements for reaching recommended goals for steps taken per day, informational promotions about the benefits of step activities and prompts to reach the step goal throughout the day if they have not. Also Bellabeat's app should record and average customers' times of activity to generate prompts at these times of day to encourage customers to reach their goals. According to Medical News Today, the recommended amount of steps are 8,000 steps for a 51% lower risk. After customers reach the 8,000 steps per day for extended periods of time, such as 1-3 months Bellabeat should offer discounted prices for the Spring products.

8.1.0.4 Calories: Customers should receive notifications and achievements for reaching recommended goals for calories burned per day. Informational promotions about the benefits and relationships of burned calories and weight loss/gain. Also achievements and prompts to reach the calorie goals throughout the day if they have not. Bellabeat stakeholders should use 2200 calories burned per day as the benchmark goal as Everyday Health suggest adult women burn between 1600-2400 and men 2000-3000 calories per day for adequate weight loss. Bellabeat would also benefit from allowing users to change the calories burned goals with a slider on the app. After 1-3 months of extended use Bellabeat should offer customers discounted prices on Spring products.

8.1.0.5 Sleep: Bellabeat's stakeholders would benefit from setting up an alarm system through either the app or directly through their wearable devices because data suggest positive relationships between sleep and calories burned right around the recommended amount of 420 min by Mayoclinic.org although a very steep drop off after 430 mins. So sticking to a strict schedule is important. The Bellabeat app should give suggestions on adequate sleep time goals that align with step, activity and calorie recommendations. The app should also provide a slider where users can adjust these sleep times although the other recommendations should adjust higher for longer sleep and lower for less sleep.

8.1.0.6 Weight: Bellabeat's stakeholders would benefit from developing a weight scale product that would connect to the app for adequate weight data records. Data shows that manual reporting most of the time does not offer enough clean data to be used. Also they should then offer a section in the app that allows users to adjust weight loss / gain goals while adjusting recommended activity amounts.