Nathan Glen, Shannon Dowdy, Aubrey Fortmayer

Dr. Harrington

MAD2502 Fall 2023

11 Dec 2023

# Workout Data Analysis App Group

## Introduction:

Many people work out, and it can be tiring to constantly try to plan out workouts every single day. In our project, we created a locally hosted digital web app that takes workout data and uses it to make data-informed recommendations to our users based on their planned workout. The primary goal of our project is to revolutionize workout experiences by offering personalized fitness plans and music selections tailored to individual users' preferences and exercise routines. The existing challenge in many fitness applications is the provision of generic workout plans that may not align with users' specific needs, fitness levels, or preferred exercises. By leveraging user data, our aim is to bridge this gap and generate recommendations that are finely tuned to match each user's workout duration, intensity, and exercise preferences. This ensures that users receive tailored workout plans and music selections that resonate with their unique fitness goals, ultimately enhancing their workout experiences. The plan for this project was to tackle the problem of making workouts more interesting – through general data analysis and connecting the user with their favorite songs through this same data analysis.

**Previous Work:**

In developing our project, we sought to incorporate a Spotify API to curate music recommendations based on users' heart rates. To achieve this, we referenced and adapted code demonstrated in the YouTube video titled 'How to Use Spotify's API with Python | Write a Program to Display Artist, Tracks, and More.' This video, created by Tim with TechwithTim on the Akamai Developer channel, provided invaluable insights and guidance on interfacing with Spotify's API using Python Our adaptation of this code allowed us to integrate heart rate data (Kaggle) derived from our project's analysis with Spotify's vast music library, facilitating personalized song recommendations for users based on their workout intensity. This utilization of the code significantly enhanced the functionality and user experience of our project. Another facet of our code that required outside assistance was found in the algorithm used in the function *find_closest_bpm* in the Running.py file. We were having trouble initializing the variables, but found the answer in a stackoverflow forum (Barmar). Shown below is the function we used the forum for:

```
def find_closest_bpm(songs_dict, target_bpm):
    closest_song = None
    # https://stackoverflow.com/questions/68198688/whats-
    min_difference = float('inf')

    for song, bpm in songs_dict.items():
        difference = abs(bpm - target_bpm)
        if difference < min_difference:
            min_difference = difference
            closest_song = song

    return closest_song
```

**Methods:**

**Where the Data Originated**

Our running data originated from the workout app Strava. Strava is a mobile app that track's users' workouts using GPS and heart rate monitoring. The dataset we selected was ideal for our project as it was a large dataset with multiple, varied samples.

**Data Analysis**

For the analysis, we used the packages **pandas** and **matplot.pyplot** in order to read in a csv and make sense of our data by analyzing a DataFrame. We split the data into two DataFrames, grouped by gender in order to account for the differences in heart rate that might occur between the two. We also found the **.describe()** function very useful - as we wanted to use the summary statistics to give ourselves a range for the bpm we look at in our app (Real Python).

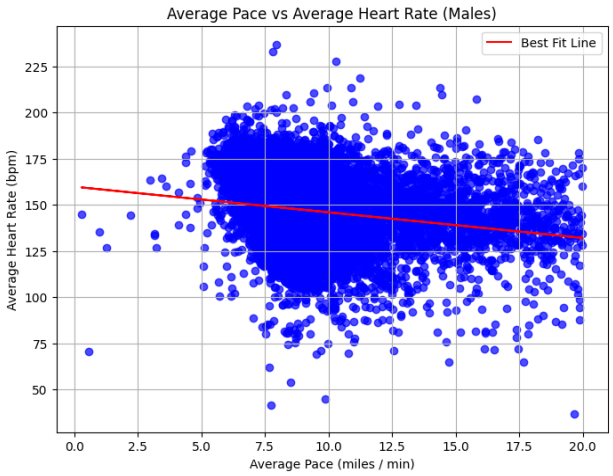
| | distance (m) | elapsed time (s) | elevation gain (m) | average heart rate (bpm) | average pace (miles / min) |
|---|---|---|---|---|---|
| **count** | 4365.000000 | 4365.000000 | 4365.000000 | 4365.000000 | 4365.000000 |
| **mean** | 10482.614066 | 4182.217411 | 197.874158 | 153.972371 | 10.570557 |
| **std** | 6289.093979 | 3319.972791 | 389.490402 | 14.174523 | 3.406561 |
| **min** | 105.000000 | 114.000000 | 2.000000 | 72.000000 | 1.285649 |
| **25%** | 7100.600000 | 2685.000000 | 35.200000 | 145.300000 | 9.201122 |
| **50%** | 9628.500000 | 3540.000000 | 83.300000 | 154.400000 | 9.827859 |
| **75%** | 12092.300000 | 4592.000000 | 197.000000 | 163.700000 | 10.791689 |
| **max** | 90281.000000 | 55226.000000 | 11128.000000 | 196.300000 | 114.873315 |

Table 1

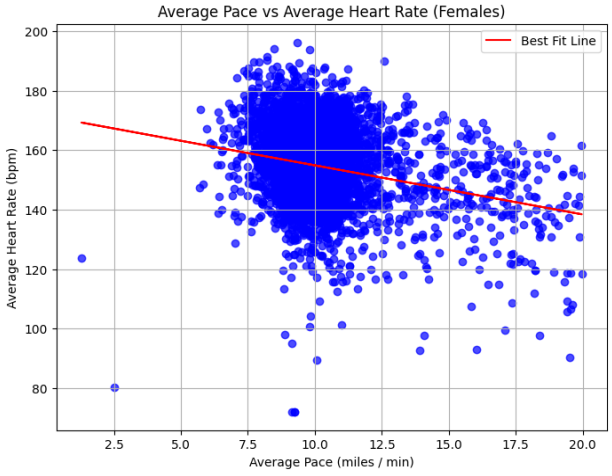
| | distance (m) | elapsed time (s) | elevation gain (m) | average heart rate (bpm) | average pace (miles / min) |
|---|---|---|---|---|---|
| **count** | 4365.000000 | 4365.000000 | 4365.000000 | 4365.000000 | 4365.000000 |
| **mean** | 10482.614066 | 4182.217411 | 197.874158 | 153.972371 | 10.570557 |
| **std** | 6289.093979 | 3319.972791 | 389.490402 | 14.174523 | 3.406561 |
| **min** | 105.000000 | 114.000000 | 2.000000 | 72.000000 | 1.285649 |
| **25%** | 7100.600000 | 2685.000000 | 35.200000 | 145.300000 | 9.201122 |
| **50%** | 9628.500000 | 3540.000000 | 83.300000 | 154.400000 | 9.827859 |
| **75%** | 12092.300000 | 4592.000000 | 197.000000 | 163.700000 | 10.791689 |
| **max** | 90281.000000 | 55226.000000 | 11128.000000 | 196.300000 | 114.873315 |

Table 2

We used **pyplot** in order to create two scatterplots of our data. We measured the pace of the run in mi/minute (as the x-value) against the average heart rate (y-value). As we predicted, the scatter plots seem to have different bpm ranges, and they are both negatively correlated (as the pace gets faster, the average bpm goes up). We ended up using **scipy.optimize** in order to perform a linear regression and draw lines of best fit on the data. To plot the data, we began by setting maximum allowable pace to 20 miles per min, which allowed us to filter for most outliers. We then used **scipy** package's **curve_fit** function to find the best-fit line. Using matplotlib.py, we plotted the data set on a scatter plot with average pace on the x-axis and average heart rate on the y-axis. Shown below are the two plots we got as a result of our analysis.




-1.3879786907495584x + 159.88667781131127

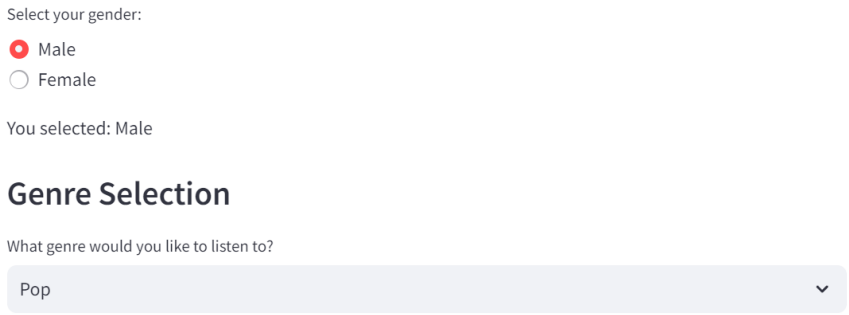



-1.6532179784683314x + 171.44424342083707

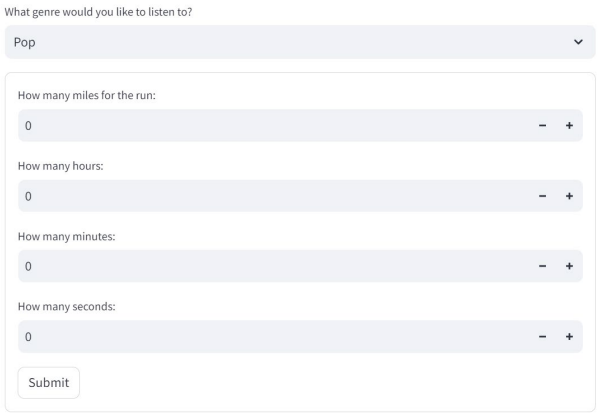
We saved the equations these two lines gave us for future use in our Running.py file, as we need to plug in the average pace we get from collecting user data as the x in the linear equation(s).

**Running Program**

To get the average pace of the user - we used a UI entirely provided by the package **streamlit**. We first ask the user what gender they listen to, and what genre they want to listen to. Gender changes the equation used to calculate heart rate and the genre changes what list of music the user can listen to. Shown below is this interface:




The specific object we used to collect user data for Running.py is something known as an st.form. This form collects information about the predicted miles run, and the predicted hours, minutes, and seconds the user will run the miles in. Below is the UI with some of its code.




```
with st.form(key='run_details_form'):
    miles = st.number_input("How many miles for the run:", value=0, max_value=59)
    hours = st.number_input("How many hours:", value=0, max_value=59)
    minutes = st.number_input("How many minutes:", value=0, max_value=59)
    seconds = st.number_input("How many seconds:", value=0, max_value=59)

    submitted = st.form_submit_button("Submit")
```

Then we store that information in a dictionary, and then we add that dictionary to a **pandas** DataFrame, with an extra column in that DataFrame being calculated - the pace of the run. After the DataFrame is created, we take the mean() of all the values in the pace column and get the average pace.

Current Runs:

| | Miles | Hours | Minutes | Seconds | Pace (mi/min) |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 20 | 0 | 20 |
| 1 | 1 | 0 | 15 | 0 | 15 |

**Average Pace:** 17.50 minutes per mile

```
if submitted:
    new_run_data = {
        'Miles': [miles],
        'Hours': [hours],
        'Minutes': [minutes],
        'Seconds': [seconds]
    }
    new_run = pd.DataFrame(new_run_data)
    if (new_run['Miles'] != 0).any(): # can't divide by 0
        new_run['Pace (mi/min)'] = ((new_run['Hours'] * 60 + new_run['Minutes'] + new_run['Seconds'] / 60.0) / new_run['Miles'])
    st.session_state.runs = pd.concat([st.session_state.runs, new_run], ignore_index=True)
    form_submitted = True
    st.success("Run details added to DataFrame!")
```

After the **Average Pace** is calculated, we plug it into our equation (for male or female) and then we call the Spotify API. Through many function calls, we get the bpm of a song from a selection curated by us that most closely matches the users predicted heartrate. More information on how the Spotify API works is in our presentation video. Located below is an example of what it looks like when the heartrate is matched to a song:

based, on your bpm, the best song for you is: Must Be Love (Laufey)

## Song Details

### Album Artwork

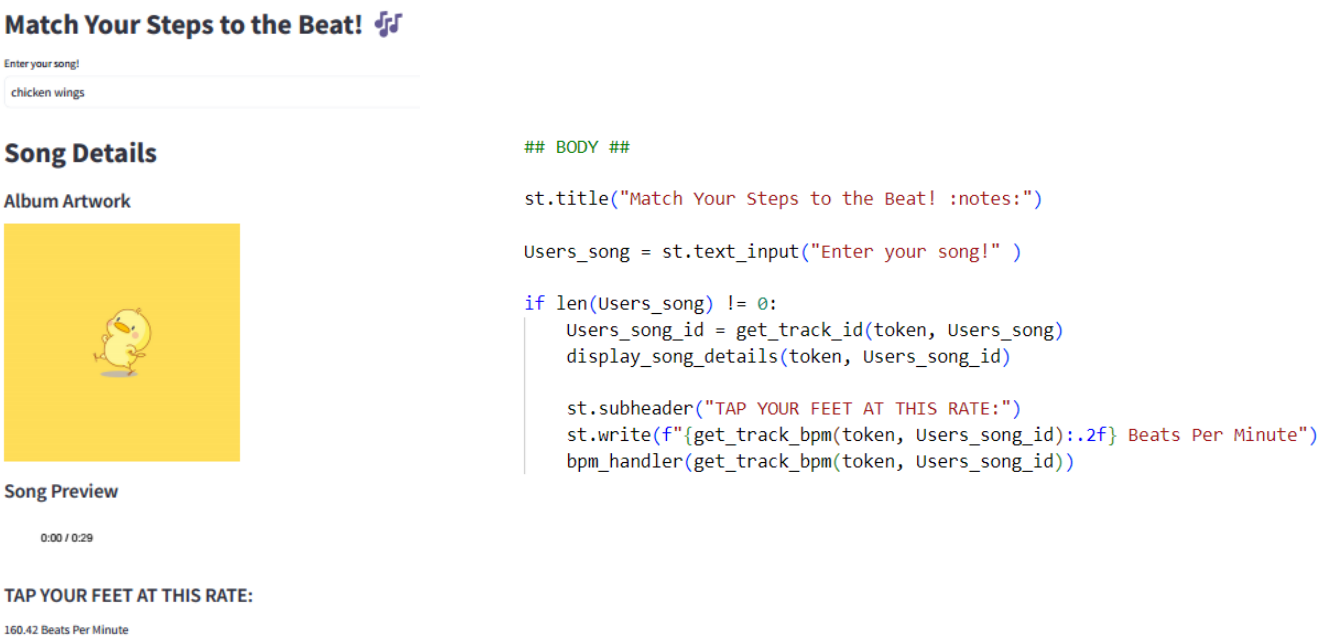
```
# get the song closest to your predicted heartrate
best_song = find_closest_bpm(songs_dict, predicted_bpm)
print(f"songs dict is {songs_dict}")
print(find_closest_bpm(songs_dict, predicted_bpm))
st.write(f"based, on your bpm, the best song for you is: {best_song} ")

# get the id, important for visual components
best_song_id = get_track_id(token, best_song)
display_song_details(token, best_song_id)
```

**Matching Footsteps:**

This program only has one unique function that is not found inside the Running.py program. That function is shown here:

```
def bpm_handler(bpm):
    if bpm < 120:
        st.write("You should be **Speed Walking**!")
    if bpm >= 120 and bpm <= 140:
        st.write("You should be **Jogging**!")
    if bpm > 140:
        st.write("You should be **Running**!")
```

Essentially, this function takes the bpm of a song the user enters in and tells the user what form of cardio that should be partaking in based on the bpm of that song. The goal is to have the user match their feet to beat. Shown below is the UI element of this:

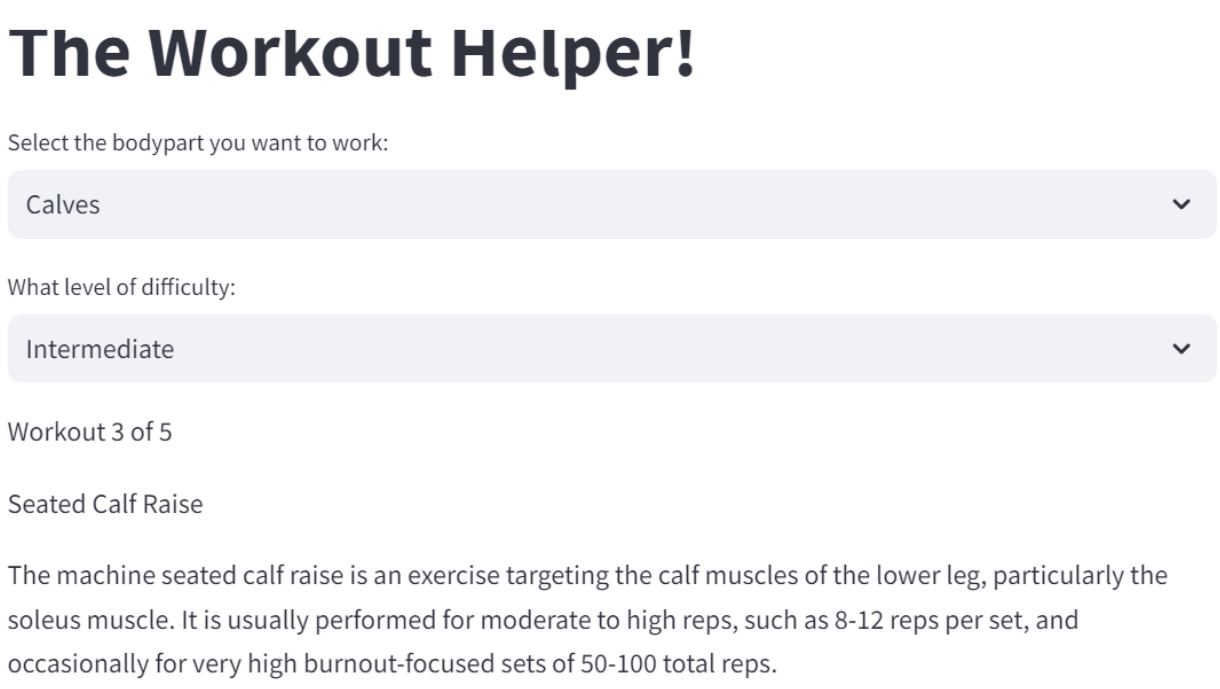


**Weightlifting:**

For the weightlifting program, we made a large DataFrame out of a csv file, in which two of the prominent column titles were “Title”, “BodyPart” and “Difficulty” (Kaggle). Immediately we decided to connect these titles, and came up with a simple recommendation system that takes whatever body part you want to work, and the difficulty of your desired exercise and outputs a description of the exercise you want to do.

```
def filter_workouts(body_part, difficulty):
    filtered_workouts = workouts_df[(workouts_df['BodyPart'] == body_part) & (workouts_df['Level'] == difficulty)]
    return filtered_workouts

def display_workout(index, filtered_results):
    workout_index.write(f"Workout {index + 1} of {len(filtered_results)}")
    workout_title.write(filtered_results.iloc[index]['Title'])
    workout_description.write(filtered_results.iloc[index]['Desc'])
```

As you can see from these functions, the basic logic behind the program is indexing through the DataFrame, filtering out workouts by body part, level of difficulty and then iterating through workouts that share body part and difficulty and displaying a workout of n index in the format of **“Title: Description”.** Shown below is the UI of the program:

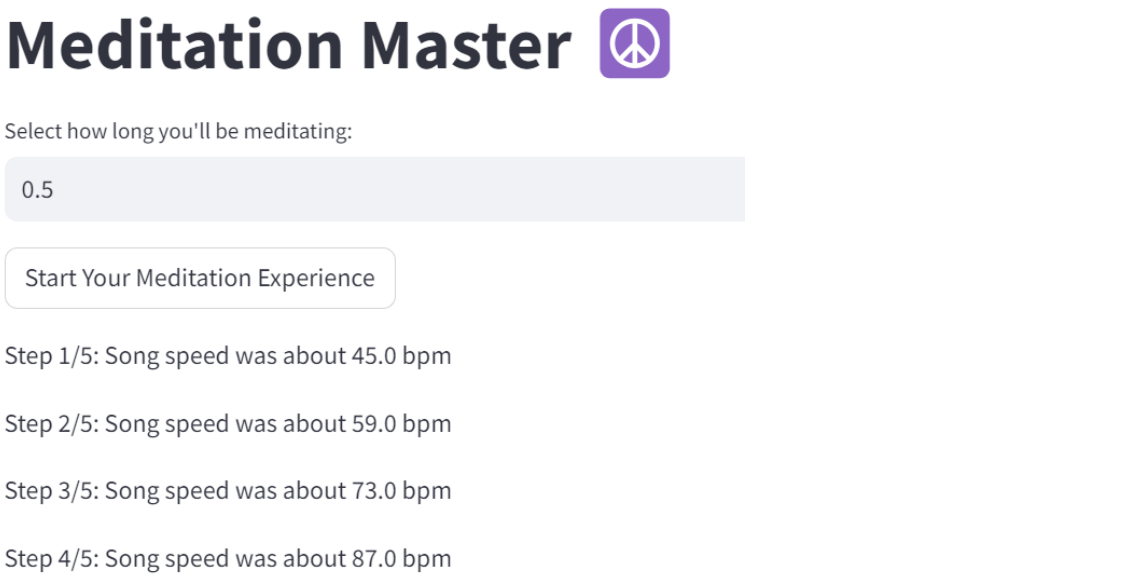


**Yoga:**

In our final program, we simply take 5 songs that range from very slow to moderately fast and play them in that order. In order to accomplish this we needed extra packages, one called **simpleaudio** and one called **time. Simpleaudio** just allows us to play and stop sounds from mp3 files. **Time** is a built-in python module that allows us to utilize system time to space the distance between songs based on the time the user selects for their cumulative meditation time. We created our own class called **AudioSong** to handle the playing and stopping of songs. We make 5 **AudioSong** objects and then have two for loops, one going from index 0-4 and one going from index 4-0. Each of the for loops plays songs at specific intervals of time and stops any sound playing before the next song is played. Below is some of the code showcasing these behaviors:

```python
class AudioSong:  # just a class for handling audio because tha
    def __init__(self, file_path):
        self.file_path = file_path
        self.wave_obj = sa.WaveObject.from_wave_file(file_path)
        self.play_obj = None

    def play(self):
        self.play_obj = self.wave_obj.play()

    def stop(self):
        if self.play_obj:
            self.play_obj.stop()
```

```python
for i in range(steps):
    if i != 0: # if we're playing something and we're not on the first iteration
        if song_playing:
            song_list[i - 1].stop()
            print("previous song stopped")
            song_playing = False

    song_list[i].play()
    song_playing = True
    print(i)
    current_speed = start_speed + i * step_size
    time.sleep(duration_seconds / steps)
    st.write(f"Step {i+1}/{steps}: Song speed was about {current_speed} bpm")

    if i == steps - 1:
        song_list[i].stop()
        print("Last song stopped")
        song_playing = False
```



**Results:**

Reflecting on the outcome of project, our development endeavors have resulted in a multifaceted web application tailored to diverse workout categories. Our application, built on the **streamlit** platform, offers a diverse range of functionalities organized into different modules: Match Footsteps, Running, Weightlifting, and Yoga. The Match Footsteps feature allows users to select songs from Spotify, preview the song, display the song's beats per minute, and receive pace recommendations—whether it's speed walking, jogging, or running—tailored to the song's tempo range. Moving to the Running module, users input their gender, preferred music genre, run distance, and duration. Here, utilizing a linear regression model fueled by Strava Kaggle data, we derive the average heart rate based on the average pace, a pivotal metric used to curate song suggestions aligned with individual preferences and heart rates. Within the Weightlifting section, users specify their targeted body part and preferred difficulty level, prompting the system to craft custom workout routines sourced from the Kaggle dataset, complete with detailed exercise descriptions. Finally, within our Yoga module, users can set their workout duration, triggering dynamic music playback that transitions from a slower tempo at the start, progressively increases the beats per minute in the middle, and then gradually decreases the beats per minute towards the end of the workout, synchronizing seamlessly with the workout's duration. This comprehensive application seamlessly integrates tailored workout suggestions and dynamic music options, catering to diverse fitness preferences and providing a holistic exercise experience.

## Conclusion:

Our data analysis highlighted a significant correlation between average pace and heart rate during running workouts. Utilizing user input regarding their run duration, our application accurately calculates the average pace for that specific duration, subsequently estimating the average heart rate during the workout through a robust linear regression model. This insightful correlation between pace and heart rate allowed for precise BPM matching of songs, enhancing the music selection feature based on the user's workout intensity. Additionally, in weightlifting, we harnessed data to create personalized workouts based on user-specified fitness levels and targeted body parts. For yoga, user-provided workout duration input facilitated music curation, smoothly transitioning from slower to faster beats during the workout duration, amplifying the exercise experience. The code achieves its tasks by leveraging various data analysis and computation techniques tailored to each workout type within the digital web application.

Given more time, we would extend the scope of our project in terms of data sources and features in the app. For example, we originally wanted to analyze Apple Watch data, but we never found the correct way to parse and use the data. Apple Watch data would allow us to truly make a personalized experience for users, with access to real-time heart-rate data, workout dynamics, and other metrics related to fitness, the possibility for statistical analysis and discovery is vast. Additionally, we would flesh out the features that already exist, and make sure that they are fully functional and customizable. One possible future feature is adding something that keeps track of user’s data across multiple uses of our app, as we currently only allow for the user and their input data to live in the confines of one session in **streamlit**. We could also work on more sophisticated, or complicated recommendation systems in the future - incorporating more factors

than just heart rate, and maybe even utilizing machine learning. One thing that would stay constant, though, is the use of regression models.

Our team underwent a profound learning journey that extended beyond technical skills throughout the completion of this project. We've seen the potential of data-driven personalization in reshaping user experiences, particularly in fitness apps. It's been eye-opening to learn about the intricate relationship between user data, statistical analysis, and application development. Furthermore, investigating and integrating the Spotify API in Python expanded our capabilities, allowing us to seamlessly integrate dynamic music selection features based on BPM matching into our fitness application. In the future, we hope to apply these insights to new projects by emphasizing user-centric design and leveraging data analytics to create solutions that adapt dynamically to individual preferences and evolve with user needs. This project was critical in demonstrating the symbiotic relationship between data analysis and application development, allowing us to envision and create innovative, personalized solutions across various domains.

**References:**


Akamai Developer. “How to Use Spotify’s API with Python | Write a Program to Display Artist, Tracks, and More.” *YouTube*, YouTube, 7 Dec. 2022, www.youtube.com/watch?v=WAmEZBEeNmg.

Barmar. “What's a good starting value for a min (or max) variable”, *stackoverflow*, Jun 30, 2021. Forum post. https://stackoverflow.com/questions/68198688/whats-a-good-starting-value-for-a-min-or-max-variable.

Olegoaer. “Running Races from Strava 👟.” *Kaggle*, 16 Sept. 2022, www.kaggle.com/datasets/olegoaer/running-races-strava.

niharika41298. “Ultimate Gym Exploratory Data Analysis.” *Kaggle*, Kaggle, 2 May 2023, www.kaggle.com/code/niharika41298/ultimate-gym-exploratory-data-analysis.

“Pythonic Data Cleaning with Pandas and NumPy.” *Real Python*, Real Python, 7 Feb. 2023, realpython.com/python-data-cleaning-numpy-pandas/.