

Final Project: Walking/Jumping Classification

ELEC 292

Faculty of Engineering and Applied Science

Queen's University

Prepared by [Group 58]

Anapayaan Pakerathan – 20347629 – 21arp10@queensu.ca

Hendrix Gryspeerdt – 20337154 – 21hgg3@queensu.ca

Sunday April 7th, 2024

“We do hereby verify that this written report is our own individual work and contains our own original ideas. No portion of this report has been copied in whole or in part from another source, with the possible exception of properly referenced material.”

1. Data Collection

The data collection process involved working with an app called phyphox denoted as an application mainly for phones with various sensors. These sensors serve to collect input from users and provide an in-depth breakdown of their movements. With several graphs depicting time measurements, motion analysis, sound intensity and numerous other data visualizations.

Phyphox offers a platform to conduct these experiments and explore the physical aspects of various phenomena with accuracy and simplicity. Data was collected from the raw sensors specifically acceleration(without a gravitational force) provided by the linear accelerometer. The data included measurements of linear acceleration across all dimensions in the x-y-z plane respectively as well as absolute acceleration with time being the independent variable. The user input involved all members of the team to record their own data inorder to eliminate any bias and effectively compare results, ensuring the credibility and reliability of the findings. Each individual was required to walk and jump in 40 second intervals in various positions ranging from front to back, chest to hand pocket. The total duration of the data collection process was 5 minutes and 20 seconds. Upon collection of the data an export feature in the phyphox application was done to save the measurements as a single csv file alongside the uploaded metadata. The metadata essentially consisted of information about the device with specific references to the time the data had been performed. It's important to note that during the data collection process files were exported in 40 second intervals, resulting in a total of 8 csv files for each individual.

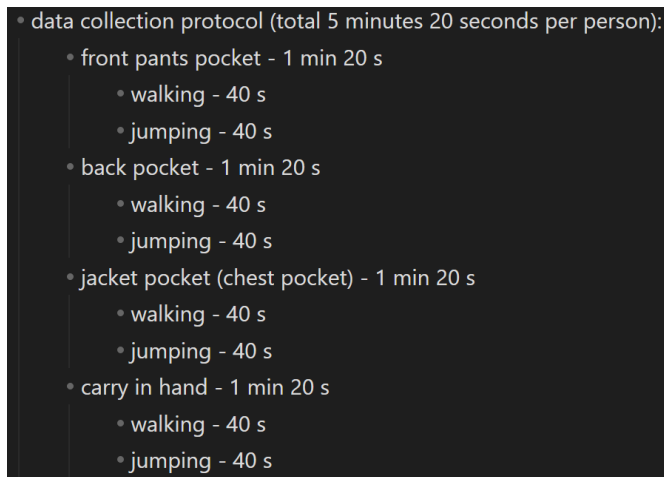
- 
- data collection protocol (total 5 minutes 20 seconds per person):
 - front pants pocket - 1 min 20 s
 - walking - 40 s
 - jumping - 40 s
 - back pocket - 1 min 20 s
 - walking - 40 s
 - jumping - 40 s
 - jacket pocket (chest pocket) - 1 min 20 s
 - walking - 40 s
 - jumping - 40 s
 - carry in hand - 1 min 20 s
 - walking - 40 s
 - jumping - 40 s

Figure 1: Data collection protocol.

2. Data Storing

Data storing occurred in 3 stages. The first stage of data was to organize the direct email exports from phyphox stored in zip files. The second stage of data resulted from unzipping the compressed files from phyphox. The third stage was to store all datasets in an hdf5 file in the format as collected and in a format more suited for training and testing a classification model. A diagram illustrating this process is shown in Figure 2 below.

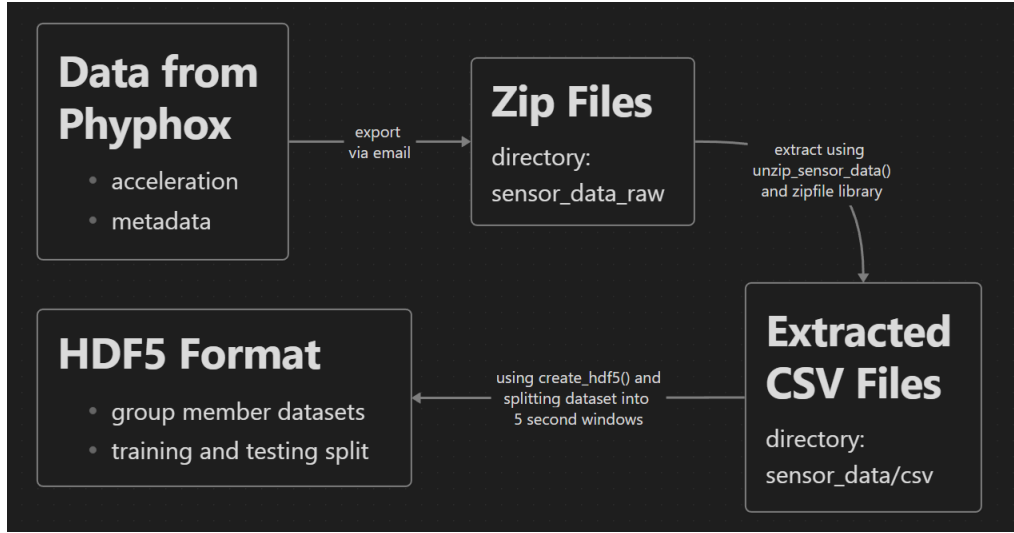


Figure 2: Stages in the data storage process.

The resulting file structure of the first two stages of data storage are shown in the two figures below.



Figure 3: (left) zip files as exported from phyphox [1]. (right) File structure of compressed data exported from phyphox [1].

The acceleration data in each time-stamped directory is found in the file “Raw Data.csv”. Additional metadata regarding the device used to collect data is found under the meta subdirectory in files “device.csv” and “time.csv”. The headers of those files are shown below in Figure 4 .

	A	B
1	property	value
2	version	1.1.15
3	build	11233
4	fileFormat	1.17
5	deviceModel	iPhone14,6
6	deviceBrand	Apple
7	deviceBoard	
8	deviceManufacturer	
9	deviceBaseOS	
10	deviceCodename	
11	deviceRelease	17.3.1
12	depthFrontSensor	0
13	depthFrontResolution	
14	depthFrontRate	
15	depthBackSensor	0
16	depthBackResolution	
17	depthBackRate	

	A	B	C	D
1	event	experiment time	system time	system time text
2	START	0.000000000000000E0	1.711238907671333E9	2024-03-23 20:08:27.671 UTC-04:00
3	PAUSE	4.829145120833346E1	1.711238955962178E9	2024-03-23 20:09:15.962 UTC-04:00

	A	B	C	D	E
1	Time (s)	Linear Acceleration x (m/s ²)	Linear Acceleration y (m/s ²)	Linear Acceleration z (m/s ²)	Absolute acceleration (m/s ²)
2	3.63887500E-3	2.219339937E-2		-6.60E-02	-1.67E-01 1.808364450E-1
3	1.362887500E-2		-1.08E-01	1.834154606E-2	-3.86E-01 4.015481038E-1
4	2.361887500E-2		-5.48E-02	1.276879871E-1	-4.34E-01 4.554400836E-1
5	3.360887500E-2	2.907577685E-1		1.263615462E-1	-3.11E-01 4.439929658E-1
6	4.359887500E-2	7.862759659E-1		6.610862017E-3	7.863068863E-1
7	5.358887500E-2	1.003307800E0		-8.09E-02	-7.78E-02 1.009570423E0
8	6.357887500E-2	7.572457096E-1		8.348215967E-2	-1.66E-02 7.620145381E-1
9	7.356887500E-2	2.562201659E-1		1.823708555E-1	

Figure 4: Files located in directory "sensor_data/csv/hendrix/jumping/back_pant_pocket Acceleration without g 2024-03-23 20-09-28". "meta/device.csv" on the left, "meta/time.csv" on the top, and "Raw Data.csv" is on the bottom. The larger figure is truncated as there are thousands of rows.

	A	B
1	property	value
2	version	1.1.13
3	build	1011300
4	fileFormat	1.17
5	deviceModel	SM-G781W
6	deviceBrand	samsung
7	deviceBoard	kona
8	deviceManufacturer	samsung
9	deviceBaseOS	samsung/r8qcsx/r8q:13/TP1A.220624.014/G781WVLUAHM6:user/release-keys
10	deviceCodename	REL
11	deviceRelease	
12	depthFrontSensor	

Figure 5: metadata for anapayaan's smartphone accelerometer. File is large so it is truncated in this figure. File name is "sensor_data/csv/anapayaan/jumping/back_pant_pocket Acceleration without g 2024-04-02 14-57-08\meta\device.csv".

Metadata indicates that sampling rates for the different devices may differ. This is accounted for by ensuring windows sizes are based on the "Time (s)" column and not just the number of rows.

The file data_storing.py contains the functions used to process the data for storage. The function `unzip_sensor_data()` was used to extract the zip files. The second function, `create_hdf5()` is used to then walk through the "sensor_data/csv" directory to create an HDF5 file with the structure shown in Figure 6.

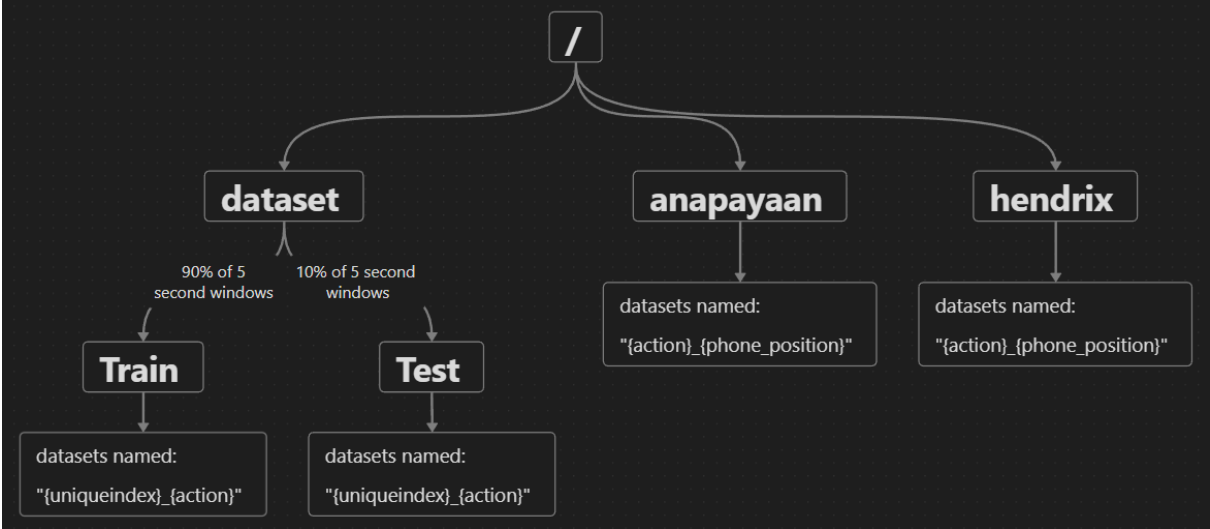


Figure 6: Structure of data stored in file "sensor_data.hdf5". All datasets have the same header as the bottom image in Figure 4.

Each dataset from files named “Raw Data.csv” are stored in the HDF5 file using the following code snippet from `create_hdf5()`.

```

2  def create_hdf5():
3      """
4      Create one HDF5 file for all the data.
5      """
6      intervals, labels = [], []
7      with h5py.File(_hdf5_file, 'w') as f:
8          # Create a group for the organized data.
9          dataset_group = f.create_group('dataset')
10         Train_group = dataset_group.create_group('Train')
11         Test_group = dataset_group.create_group('Test')
12         # Create a group for each person.
13         for person in os.listdir('./sensor_data/csv'):
14             person_group = f.create_group(person)
15             # Create a group for each action (walking, jumping) and phone position. Then, store the data from csv's in it.
16             for action in os.listdir(f'./sensor_data/csv/{person}'):
17                 for i, dir in enumerate(os.listdir(f'./sensor_data/csv/{person}/{action}')):
18                     phone_position = dir.split(' ')[0]
19                     df = pd.read_csv(
20                         f'./sensor_data/csv/{person}/{action}/{dir}/Raw Data.csv', index_col=0)
21                     # Store sensor data in person group.
22                     person_group.create_dataset(
23                         f'{action}_{phone_position}', data=df.reset_index(), compression='gzip', compression_opts=9)
24
25                     # Store 5 second windows of sensor data in dataset group.
26                     df.index = pd.to_datetime(df.index, unit='s')
27                     for interval in df.rolling(window='5.1s'):
28                         if interval.index[-1] - interval.index[0] < pd.Timedelta('5s'):
29                             continue
30                         intervals.append(interval)
31                         labels.append(action)
32
33         # Split (90%/10%) train vs. test and shuffle.
34         X_train, X_test, Y_train, Y_test = train_test_split(
35             intervals, labels, test_size=0.1, random_state=42, shuffle=True)
36         # Save dataset into HDF5 file.
37         for i, (interval, label) in enumerate(zip(X_train, Y_train)):
38             # Revert the time column to seconds.
39             interval.reset_index(inplace=True)
40             interval['Time (s)'] = interval['Time (s)'].apply(
41                 lambda x: x.value / 1e9)
42             Train_group.create_dataset(
43                 f'{i}_{label}', data=interval, compression='gzip', compression_opts=9)
44         for i, (interval, label) in enumerate(zip(X_test, Y_test)):
45             interval.reset_index(inplace=True)
46             interval['Time (s)'] = interval['Time (s)'].apply(
47                 lambda x: x.value / 1e9)
48             Test_group.create_dataset(
49                 f'{i}_{label}', data=interval, compression='gzip', compression_opts=9)

```

Figure 7: `create_hdf5()` function located in `data_storing.py`.

Slicing the datasets using 5 second windows significantly increase the amount of data being stored due to overlap of the windows. This results in “sensor_data.hdf5” having a large file size of near 3.7 GB. This is significantly more than the data 9.67MB of data stored in raw csv format in the entire directory “sensor_data/csv”.

The last two functions in `data_storage.py` are used to load the Train and Test datasets into Pandas Dataframe Objects [2] for the purposes of training and testing the model. Loading the datasets into Pandas Dataframes the sizes of datasets can be identified. The code snippet in Figure 8 shows the different shapes of the structures.

```

# For use in train_test_split and other randomized functions.
random_state_seed = 42

# Read from HDF5 file, ignoring the time column (first column) and converting the labels to (jumping)/(not walking).
with h5py.File('sensor_data.hdf5', 'r') as f:
    train_data = pd.DataFrame.from_records(
        ((1 if n.split('_')[1] == 'jumping' else 0, pd.DataFrame(d[:, 1:]))
         for n, d in f['dataset/Train'].items()), columns=['label', 'interval'])
    test_data = pd.DataFrame.from_records(
        ((1 if n.split('_')[1] == 'jumping' else 0, pd.DataFrame(d[:, 1:]))
         for n, d in f['dataset/Test'].items()), columns=['label', 'interval'])

[2] ✓ 1m 16.3s

print(train_data.shape)
print(test_data.shape)
# Different shapes since the two devices have different sampling rates.
print(train_data.iloc[0, 1].shape)
print(train_data.iloc[3640, 1].shape)

[4] ✓ 0.0s

... (96514, 2)
(10724, 2)
(1076, 4)
(511, 4)

```

Figure 8: Loading data to show the different interval sizes as a result of accelerometer sampling rates.

This concludes data storing, data is now organized and ready to be analysed and used to train a binary classification model. Since the HDF5 file is too large to be uploaded on OnQ, you need to run “data_storing.py” to load the hdf5 file on the resulting computer.

3. Visualization

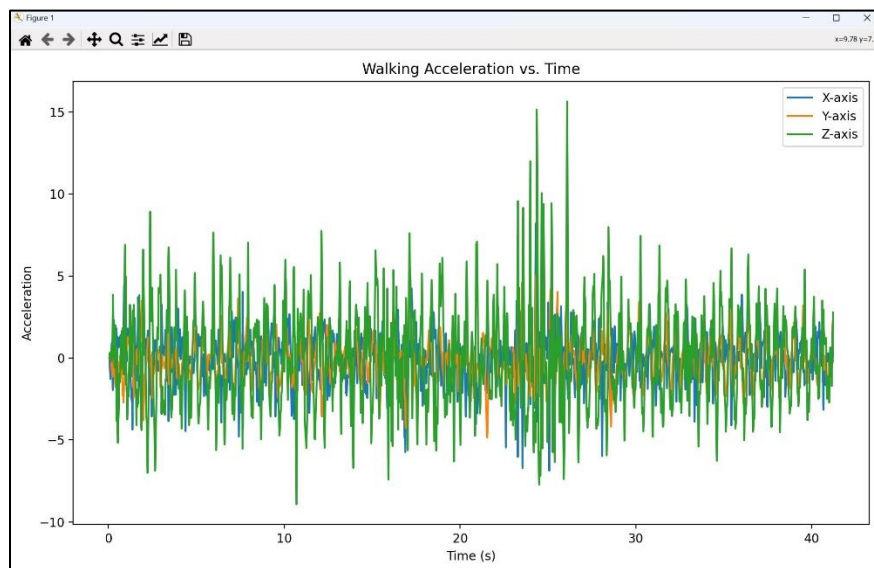


Figure 9 Walking with phone in hand (Anapayaan)

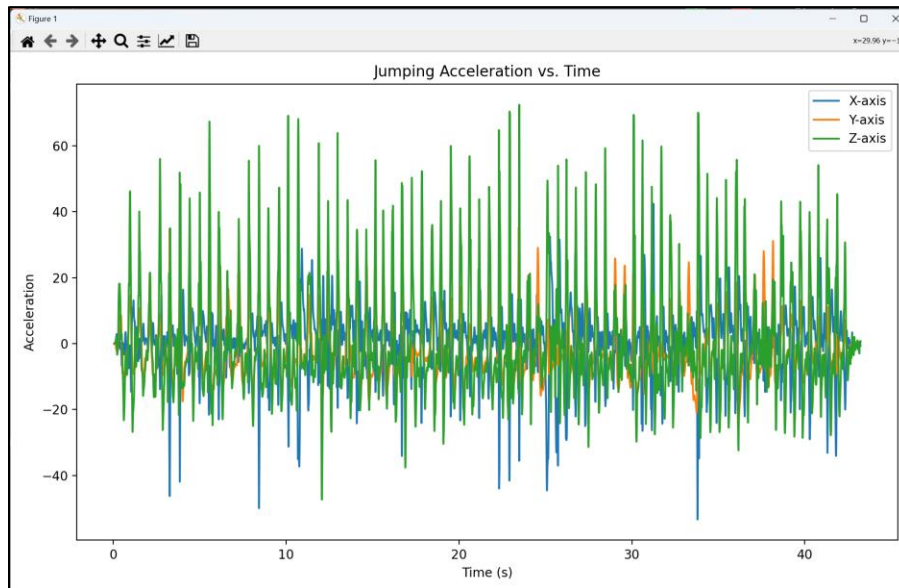


Figure 10 *Jumping with phone in hand (Anapayaan)*

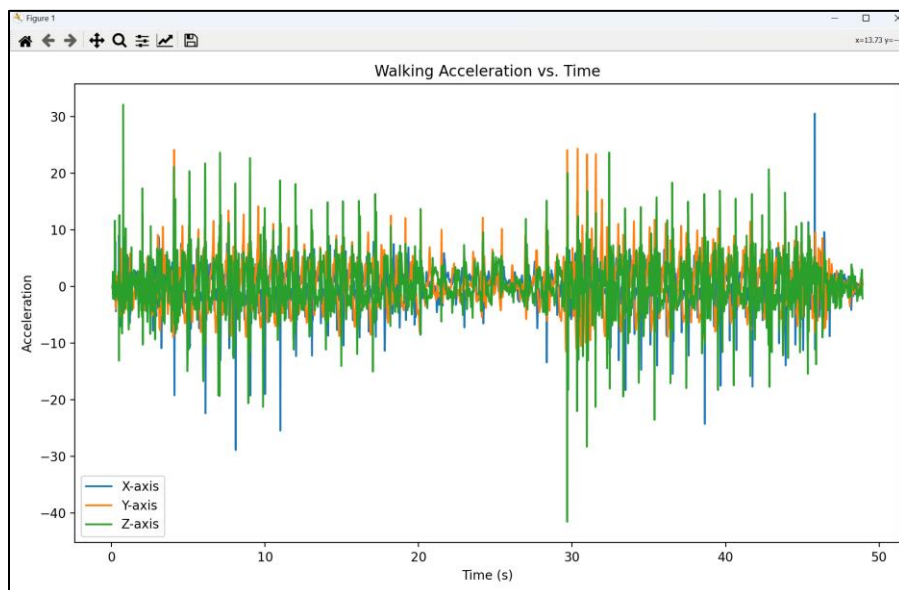


Figure 11 *Walking with phone in front pocket (Anapayaan)*

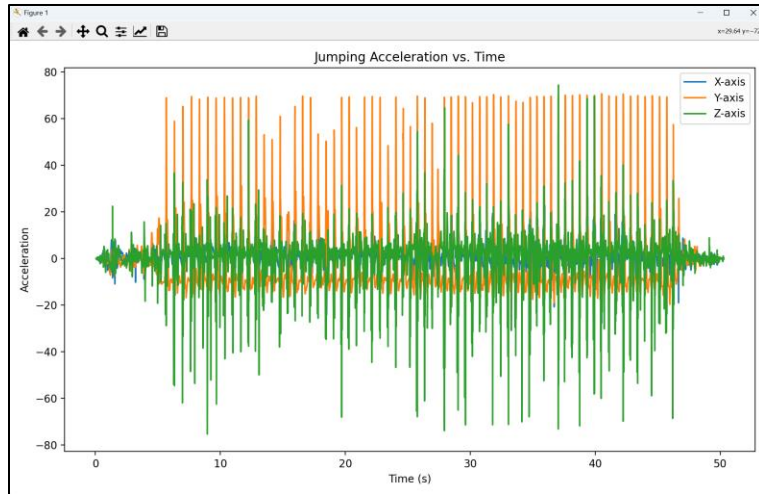


Figure 12 Jumping with phone in front pocket (Anapayaan)

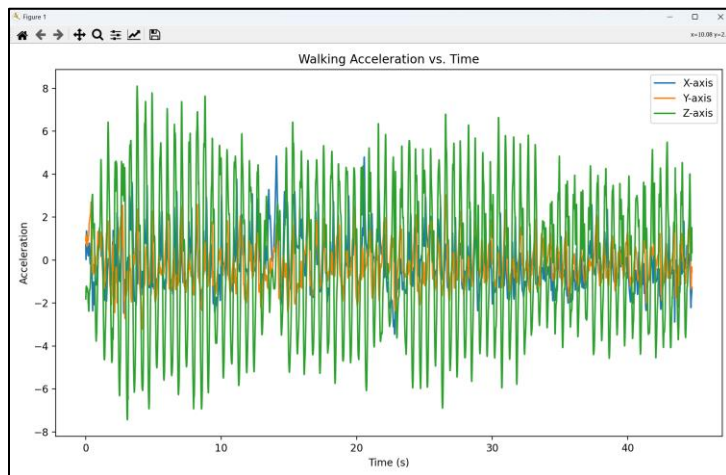


Figure 13 Walking with phone in hand (Hendrix)

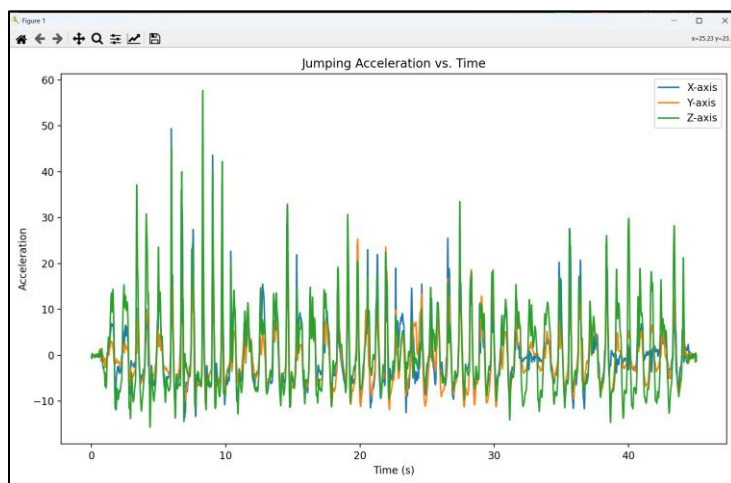


Figure 14 Jumping with phone in hand (Hendrix)

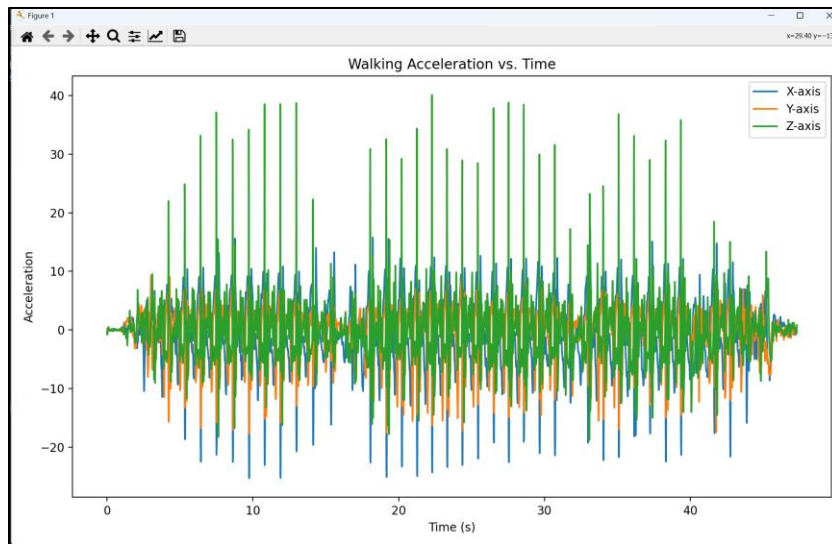


Figure 15 Walking with phone in front pocket (Hendrix)

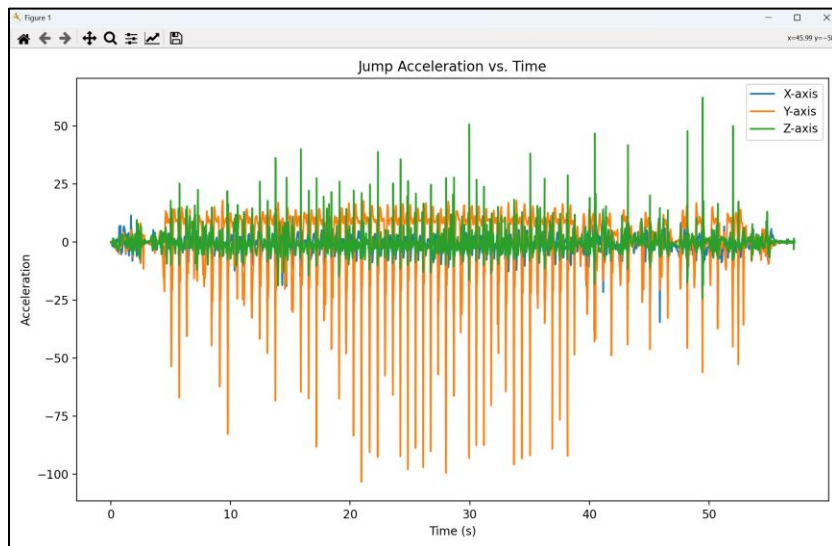


Figure 16 Jumping with phone in front pocket (Hendrix)

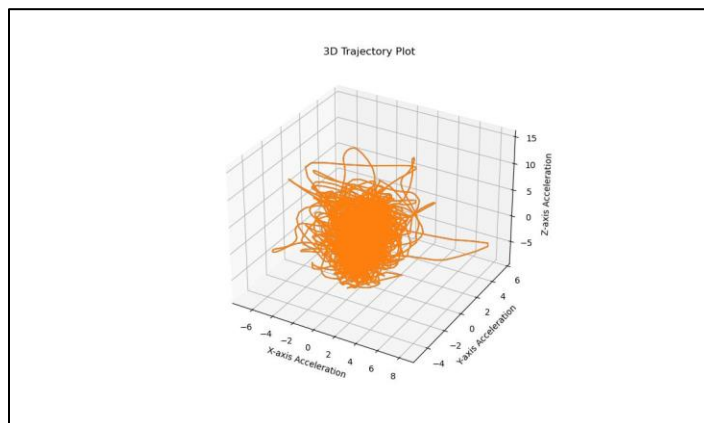


Figure 17 3D Trajectory Plot of Walking with phone in hand

Data Visualization Code:

```
import tkinter as tk
from tkinter import filedialog
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def upload_csv():
    file_path = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if file_path:
        # Load dataset
        data = pd.read_csv(file_path)

        plt.figure(figsize=(10, 6))

        # Walking Acceleration vs. Time
        plt.subplot(1, 1, 1)
        plt.plot(data['Time (s)'], data['Linear Acceleration x (m/s^2)'],
label='X-axis')
        plt.plot(data['Time (s)'], data['Linear Acceleration y (m/s^2)'],
label='Y-axis')
        plt.plot(data['Time (s)'], data['Linear Acceleration z (m/s^2)'],
label='Z-axis')
        plt.title('Walking Acceleration vs. Time')
        plt.xlabel('Time (s)')
        plt.ylabel('Acceleration')
        plt.legend()
        plt.tight_layout()
        plt.show()

        # Additional Creative Visualization Ideas
        # 3D Trajectory Plot
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot(data['Linear Acceleration x (m/s^2)'], data['Linear
Acceleration y (m/s^2)'], data['Linear Acceleration z (m/s^2)'],
label='Walking')
        ax.plot(data['Linear Acceleration x (m/s^2)'], data['Linear
Acceleration y (m/s^2)'], data['Linear Acceleration z (m/s^2)'],
label='Jumping')
        ax.set_xlabel('X-axis Acceleration')
        ax.set_ylabel('Y-axis Acceleration')
        ax.set_zlabel('Z-axis Acceleration')
        ax.set_title('3D Trajectory Plot')
        plt.show()

        # Load metadata (assuming metadata.csv is in the same directory as
the uploaded CSV file)
        metadata_path = file_path.replace('.csv', '_metadata.csv')
        metadata = pd.read_csv(metadata_path)

        # Visualization of Meta-Data
```

```

# Histogram of Sampling Rates
plt.hist(metadata['sampling_rate'], bins=10)
plt.title('Distribution of Sampling Rates')
plt.xlabel('Sampling Rate')
plt.ylabel('Frequency')
plt.show()

# Box Plots for Sensor Locations
plt.boxplot([metadata['sensor_x'], metadata['sensor_y'],
metadata['sensor_z']], labels=['X', 'Y', 'Z'])
plt.title('Sensor Locations')
plt.xlabel('Axis')
plt.ylabel('Position')
plt.show()

# Create the main window
root = tk.Tk()
root.title("CSV File Upload")

# Create a button to upload CSV file
upload_button = tk.Button(root, text="Upload CSV File", command=upload_csv)
upload_button.pack(pady=20)

# Run the Tkinter event loop
root.mainloop()

```

In terms of data visualization there are numerous things I learned about this process from the significance of data pre-processing to the specific selection of various graphs to be used can ultimately hinder the quality of visualizations. With multiple methods for expressing data, it's crucial to really understand what information is most important and how the reflection of various graphs outline those features. It's honestly a big learning curve and requires a lot of strategizing and planning to figure out the most optimal way to display the significance of information. If data collection were to be performed again there would be more attention towards the general audience and how the data outputted can be more easily interpretable. There would also exist more design features like aesthetics and layout contributing to the overall user experience.

4. Pre-processing

- just doing moving average to smooth out any potential noise from the accelerometers

The purpose of Pre-processing was to reduce noise produced by the accelerometers. Ideally, pre-processing would also include outlier removal, however, this was not done due to limited time and since the classifier was already quite accurate (as will be seen in section 6. Creating a Classifier).

As covered in class, the moving average filter is effective in reducing high-frequency noise in the dataset. However, since the accelerometer data is already high-frequency, too-large a window size would eliminate too much information. Therefore, a small window size of 10 was selected for applying the simple moving average. Examples of applying this moving average filter on 5 second interval of jumping and walking data are shown in the following table.

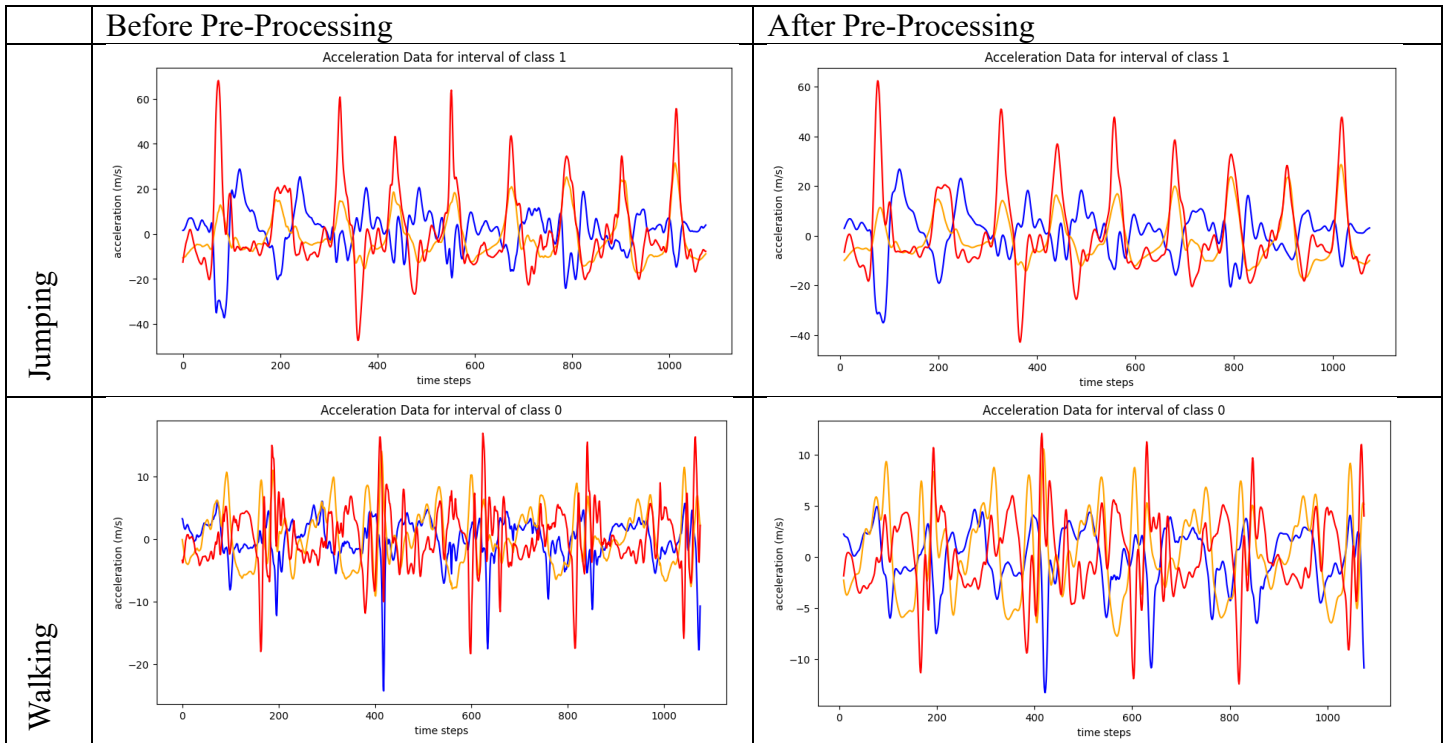


Figure 18: Illustrating the effects of the moving average filter on the accelerometer data. Graphs produce in sections 3 and 4 in the jupyter notebook classifier.ipynb.

Other forms of outlier removal could have been done, such as excluding the first and last 3 seconds of each csv file from being used. This would eliminate the time the spent to take the phone in/out of the pocket when starting/ending data collection. Overall, the sensor data was quite consistent, this led to little pre-processing being necessary. Final implementation of pre-processing can be found in the file classifier.py.

5. Feature Extraction & Normalization

After the data was pre-processed, each 5 second interval had to have features extracted to reduce the dimensionality of the 500-1000 timesteps and the 4 acceleration columns in x, y, z coordinates and absolute acceleration to something more manageable for a logistic regression model. The choice then is to calculate a short list of features for each 5 second window of acceleration data.

The features extracted for each column were min, max, range, mean, median, standard deviation, and variance, for a total of 7 different features per column. The feature extraction functions were supplied by the Pandas library. With 4 columns and 7 features per column, this resulted in a total of 28 features per 5 second interval.

After extracting the features, any intervals that resulted in missing values (NaN) were removed. Normalizing the features was then considered so to not have certain features of higher magnitude disproportionately influence the results of classification. Min-Max Scaling was applied to ensure all features were in the range from zero to one. Z-Score standardization did not seem reasonable

here since the features were not expected to form a normal distribution and this normalization technique does not ensure that all the features have values in a consistent range. Hence the MinMaxScaler [3] normalization method was used as was supplied by the scikit-learn [4] python library.

Normalization was not done on its own, as it made sense to combine normalization and logistic regression into one pipeline for the classifier. Final implementation of normalization can be found in the `classifier_create()` function of `classifier.py` and whereas experimentation can be found in section 6. Creating a Classifier in `classifier.ipynb`.

6. Creating a Classifier

The problem of classifying between two possible actions, walking and jumping, is a binary classification problem, and as taught in class, logistic regression models are effective at performing this task.

After the features were extracted they were fed into the classification model pipeline which included `MinMaxScaler()` and `LogisticRegression()` whose implementations were provided by the scikit-learn library.

Once the model was fit to the data, the classifier model was saved to a binary file `classifier.pkl` using the python pickle library `pickle.dump` [5] function. This allowed for future re-use without the need for re-training.

The creation and training of the model was completed and all implementation details can be found in the function `classifier_create()` in `classifier.py` file.

To test the classifier to see how it performed on the test split of the data. The `classifier_test()` function is used. The testing data is loaded from the hdf5 file “`sensor_data.hdf5`”; the model is deserialized from the save file “`classifier.pkl`” using `pickle.load` [5]. Then predictions are performed to calculate Accuracy, Recall, F1 Score, and Confusion Matrix, as well as ROC AUC and ROC Curves were calculated. Outputs from running `classifier_test()` can be seen in the figures below.

```
Accuracy: 0.991514360313316
Recall: 0.9860729338464358
F1 Score: 0.9887861609066784
ROC AUC: 0.9984601290535644
```

Figure 19: Console output from `classifier_test()`, listing model evaluation metrics.

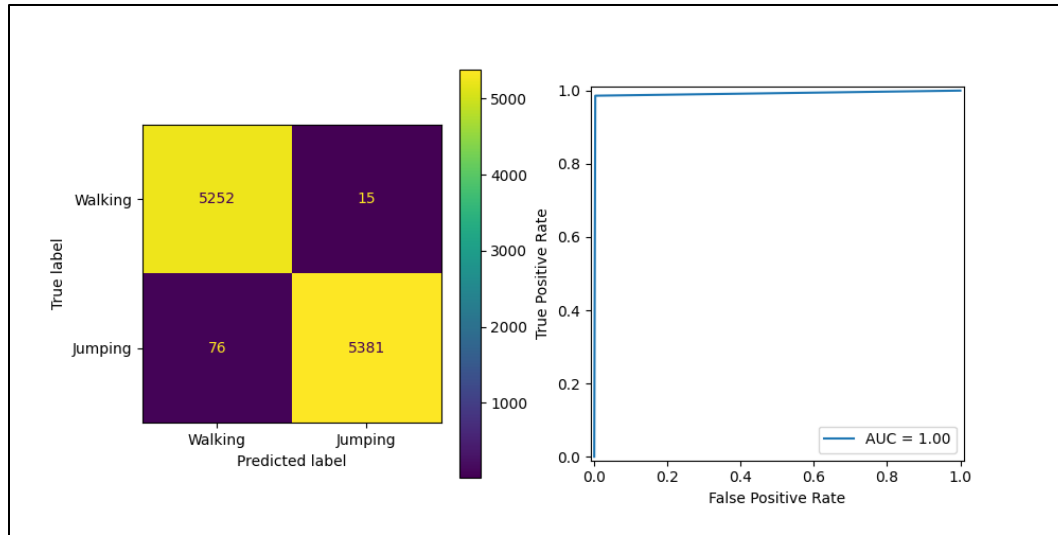


Figure 20: (left) Confusion matrix, (right) ROC curve. These figures show how the model performed.

Below is the code which was used to achieve the results in training and testing. Separation of training and testing data was ensured by only loading the training data when training occurred and only loading testing data when testing.

```

1  def preprocess(intervals: pd.Series, window_size: int) -> pd.Series:
2      """
3      Preprocess the data by applying SMA.
4      """
5      return intervals.apply(lambda x: x.rolling(window=window_size).mean())
6
7
8  def feature_extract(intervals):
9      # range is calculated by doing c.max() - c.min()
10     return pd.DataFrame.from_records(intervals.apply(lambda c: pd.DataFrame([c.max(), c.min(), c.max() - c.min(), c.mean(), c.median(), c.std(), c.var()]).to_numpy().flatten()))
11
12
13  def classifier_create(save_location: str):
14     train_data = load_hdf5_train()
15
16     print("Train data loaded.")
17
18     # Extract features from each interval. Drop NaN valued rows.
19     train_data = pd.concat(
20         [feature_extract(preprocess(train_data['interval'], 10)), train_data['label']], axis=1).dropna()
21
22     print("Features extracted.")
23
24     # Train the classifier.
25     classifier_minmax = make_pipeline(
26         MinMaxScaler(), LogisticRegression(random_state=42, max_iter=1000))
27     classifier_minmax.fit(train_data.iloc[:, :-1], train_data['label'])
28
29     with open(save_location, 'wb') as f:
30         pickle.dump(classifier_minmax, f)
31
32     print("Classifier saved.")

```

Figure 21: Python code for `classifier_create()` and associated functions in the `classifier.py` file.

```

28 def classifier_test(save_file: str):
29     # Prepare the test data.
30     test_data = load_hdf5_test()
31     test_data = pd.concat(
32         [feature_extract(preprocess(test_data['interval'], 10)), test_data
33         ['label']], axis=1).dropna()
34
35     # Load the model.
36     with open(save_file, 'rb') as f:
37         classifier = pickle.load(f)
38
39     # Evaluate the classifier.
40     X_test = test_data.iloc[:, :-1]
41     Y_test = test_data['label']
42     Y_pred = classifier.predict(X_test)
43     Y_pred_proba = classifier.predict_proba(X_test)
44     accuracy = accuracy_score(Y_test, Y_pred)
45     recall = recall_score(Y_test, Y_pred)
46     f1_score = 2 * (accuracy * recall) / (accuracy + recall)
47     roc_auc = roc_auc_score(Y_test, Y_pred_proba[:, 1])
48     print(f"Accuracy: {accuracy}")
49     print(f"Recall: {recall}")
50     print(f"F1 Score: {f1_score}")
51     print(f"ROC AUC: {roc_auc}")
52
53     figure, axis = plt.subplots(1, 2, figsize=(10, 5))
54     ConfusionMatrixDisplay(confusion_matrix(Y_test, Y_pred), display_labels=[
55         'Walking', 'Jumping']).plot(ax=axis[0])
56     fpr, tpr, _ = roc_curve(Y_test, Y_pred)
57     RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc).plot(ax=axis[1])
58     plt.show()

```

Figure 22: `classifier_test()` function as found in the `classifier.py` file.

This concludes training and testing the model. From the results were successful due to the high scores for the various evaluation metrics mentioned earlier in this section.

7. Model Deployment

Before the model can be deployed and the app can be run, you must have run the python files in the following order: `data_storing.py`, `classifier.py` (to produce the `classifier.pkl` file), then run `app.py`. If you already have access to `classifier.pkl` saved, then `app.py` can be run immediately.

Tkinter is a python library that is specifically used for creating graphical user-interfaces. It consists of several built-in GUI components such as buttons, labels and various menus that can be used to build desktop applications with a graphical interface. In the user-interface created below Tkinter was employed on multiple occasions. The main application window denoted as `'Tk()'` was utilized to essentially create the main application window while `'Button()'` allowed the creation of buttons with specified features like text and command functions. The creation of these widgets were then packed into the window from the `'pack()'` feature to be arranged within the GUI. In the grand scheme of model deployment Tkinter played a crucial role in implementing user-interface and user-interaction within the usability aspect of the application. The model was exported to a binary file to be reusable without having to be re-trained each time the application was opened. All parts of the code was consolidated into reusable modules and functions for proper organization of the application. Individual testing was also performed which

involved smaller functions and components to be tested to ensure the necessary output was attained. A performance test was also conducted to test the system in various conditions like window sizes and overall scalability of the system.

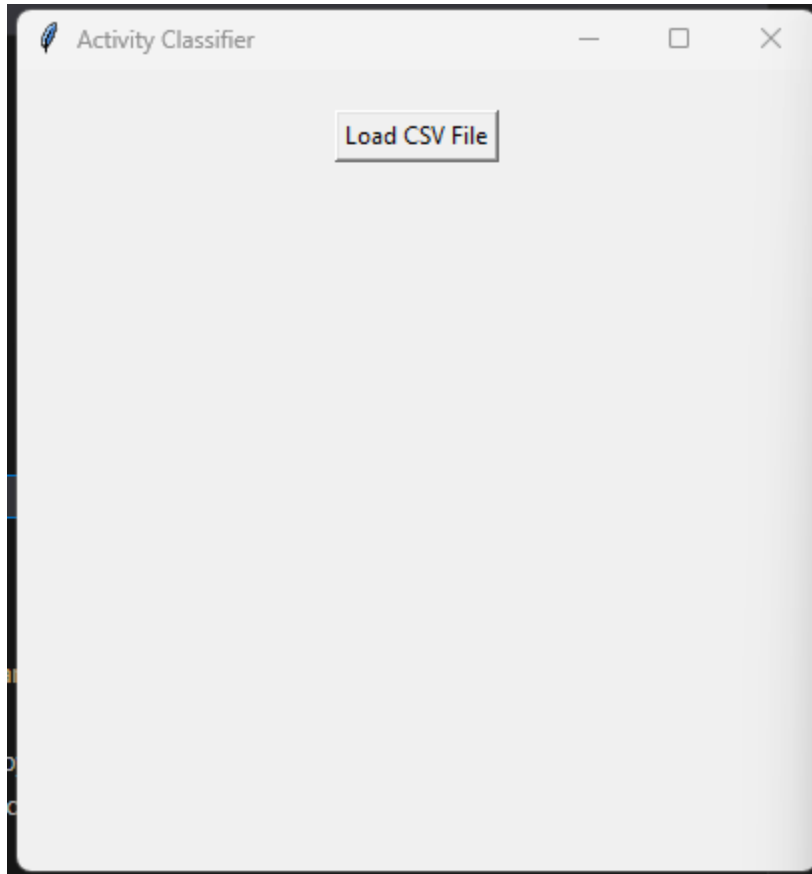


Figure 23 Input csv file

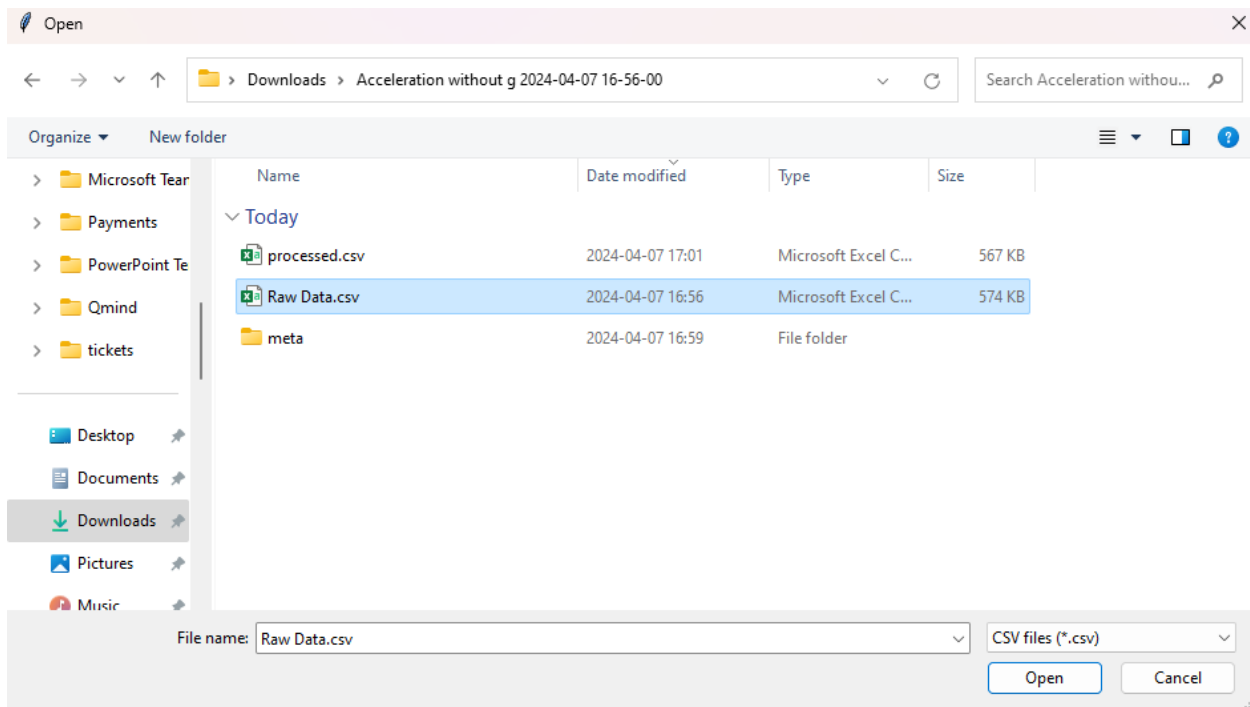
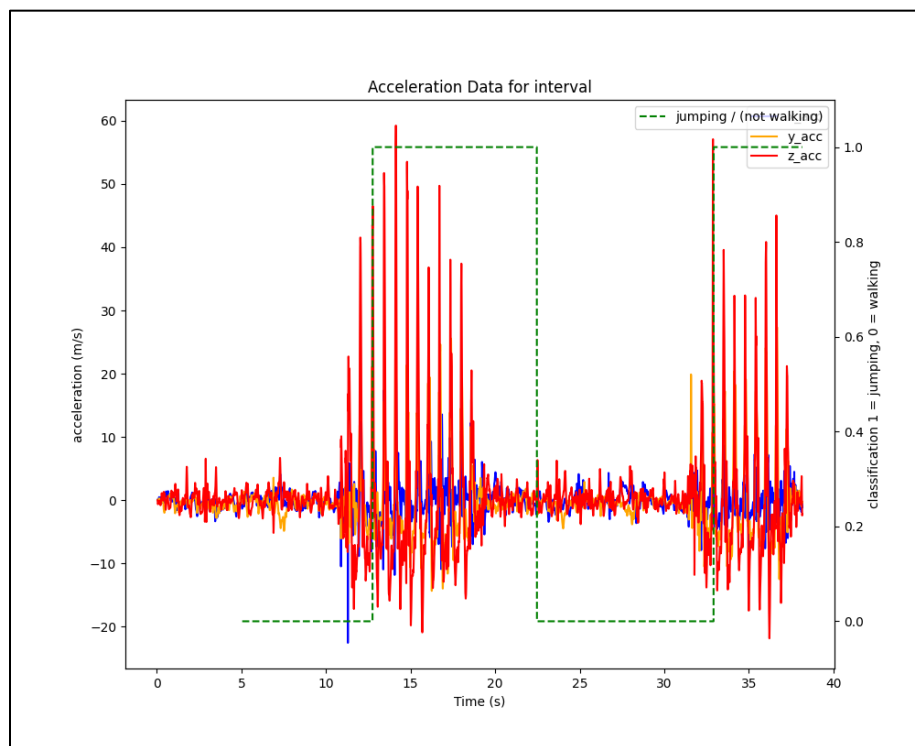


Figure 24 Raw Data Set csv file



	A	B	C	D	E	F
1	Time (s)	Linear Acceleration x (m/s ²)	Linear Acceleration y (m/s ²)	Linear Acceleration z (m/s ²)	Absolute acceleration (m/s ²)	
2	4.428704300E-2	0.000000000E0	0.000000000E0	0.000000000E0	0.000000000E0	
3	4.894928300E-2	7.952499390E-2	9.655237190E-3	-2.34E-01	2.469828120E-1	
4	5.369152200E-2	1.595895290E-1	3.112578392E-2	-4.53E-01	4.812205924E-1	
5	5.843376200E-2	1.767761707E-1	5.587458611E-2	-5.92E-01	6.201019868E-1	
6	6.317600100E-2	1.467871666E-1	6.924796104E-2	-6.24E-01	6.446478642E-1	
7	6.791824100E-2	5.827736855E-2	1.106655598E-1	-5.97E-01	6.101774234E-1	

Figure 25 Raw Data set

	A	B	C	D	E	F
1	Time (s)	Linear Acceleration x (m/s ²)	Linear Acceleration y (m/s ²)	Linear Acceleration z (m/s ²)	Absolute acceleration (m/s ²)	Labels
1054	5.033055168	-1.252093554	-0.153713942	1.389921188	1.877031473	
1055	5.03779746	-1.328058243	-0.215293869	1.189204216	1.795632706	
1056	5.042539699	-1.327662945	-0.204974494	0.943444252	1.647054478	
1057	5.047281991	-1.285717368	-0.278563291	0.745859146	1.512273958	0
1058	5.052024283	-1.239343643	-0.277793318	0.601082802	1.405148507	0
1059	5.056766574	-1.186998844	-0.29542926	0.530513763	1.33330025	0
1060	5.061508814	-1.151167154	-0.262276381	0.48139286	1.275034024	0
1061	5.066251105	-1.127665281	-0.216880232	0.393878937	1.214004382	0
1	Time (s)	Linear Acceleration x (m/s ²)	Linear Acceleration y (m/s ²)	Linear Acceleration z (m/s ²)	Absolute acceleration (m/s ²)	Labels
2670	12.6966135	-5.760048866	12.1762495	0.320890427	13.47375915	0
2671	12.70135579	-3.050240993	14.46757126	0.075674057	14.78581465	0
2672	12.70609808	0.454320312	16.47060204	2.445576668	16.65737025	0
2673	12.71084038	4.791610718	18.08452988	8.645719528	20.60966327	0
2674	12.71558267	9.427105904	19.48921204	17.39330673	27.77097101	0
2675	12.72032496	13.20569801	20.66271782	26.65465546	36.21890425	0
2676	12.72506725	15.53135014	21.12286758	34.66046143	43.459705	0
2677	12.7298096	16.78605652	20.25524712	40.24849701	48.08313884	0
2678	12.73455189	16.86181831	18.15032768	42.97561646	49.6050292	0
2679	12.73929418	15.94319725	15.32091236	43.51702118	48.81236551	0
2680	12.74403647	14.54114723	12.60291672	43.15633392	47.25195901	0
2681	12.74877876	13.21809578	10.51609707	42.96526337	46.16622369	0
2682	12.75352105	12.40495968	9.193201065	43.58262253	46.23681387	1
2683	12.7582634	11.87109947	8.169697762	45.13409424	47.37882889	1
2684	12.76300569	11.20765972	6.83585453	46.50343323	48.32090485	1
2685	12.76774798	10.41708469	5.271095276	46.50543976	47.94847261	1

Figure 26 Cleaned up data set

Participation Report

Individuals Name	Respective Task
Anapayaan Pakerathan	Data Collection Written, Personal Data collection, Data visualization code, App GUI code, Visualization written, Model Deployment written, Demo Video
Hendrix Gryspeerdt	Data storing (code+written), Personal Data, Pre-processing(code+written), Feature extraction and normalization(code+written), Training the classifier(code+written), APP GUI code+plotting, Demo Video
James Dockrill	Proof-reading entire document

References

- [1] “Your smartphone is a mobile lab.” phyphox. Accessed: Apr. 07, 2024. [Online]. Available: <https://phyphox.org>
- [2] “pandas.DataFrame — pandas 2.2.1 documentation.” Accessed: Apr. 07, 2024. [Online]. Available: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>
- [3] “sklearn.preprocessing.MinMaxScaler,” scikit-learn. Accessed: Apr. 07, 2024. [Online]. Available: <https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>
- [4] “scikit-learn: machine learning in Python — scikit-learn 1.4.1 documentation.” Accessed: Apr. 07, 2024. [Online]. Available: <https://scikit-learn.org/stable/index.html>

- [5] “pickle — Python object serialization,” Python documentation. Accessed: Apr. 07, 2024. [Online]. Available: <https://docs.python.org/3/library/pickle.html>