Fit Barbell Classifier

A major project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Dhruv Srivastava(201346)

Aahana Dutta(201178)

Under the guidance & supervision of

Dr. Ekta Gandotra

Associate Professor

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DECLARATION

We hereby declare that the work presented in this report entitled **'Fit Barbell Classifier'** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr. Ekta Gandotra** (Associate Professor, Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Student Name: Dhruv Srivastava
Roll No.: 201346

Student Name: Aahana Dutta
Roll No.: 201178

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervisor: Dr. Ekta Gandotra
Designation: Associate Professor
Department: Computer Science & Engineering Information Technology
Dated:

CERTIFICATE

This is to certify that the work which is being presented in the project report titled **'Fit Barbell Classifier'** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **Dhruv Srivastava(201346) and Aahana Dutta(201178)** during the period from August 2023 to May 2024 under the supervision of **Dr. Ekta Gandotra**, Associate Professor, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat.

Dhruv Srivastava (201346)
Aahana Dutta (201178)

The above statement is correct to the best of my knowledge.

Dr. Ekta Gandotra
Associate Professor
Computer Science & Engineering and Information
Technology Jaypee University of Information Technology,
Waknaghat

ACKNOWLEDGEMENT

Firstly, we express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

We are really grateful and wish my profound indebtedness to Supervisor **Dr. Ekta Gandotra, Associate Professor**, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Wakhnaghat. Deep knowledge & keen interest of my supervisor in the field of "**Fit Barbell Classifier**" to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, we might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated our undertaking.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

Dhruv Srivastava (201346)
Aahana Dutta (201178)

ABSTRACT

Exercise is one of the four key pillars of human well-being. In today's world where people are becoming more sedentary day by day, leading to conditions like cardiovascular diseases, obesity, and type 2 diabetes, with regular exercises, one can reduce the above risks and enjoy a playful and joyous healthy lifestyle. Calisthenics, cardio and yoga are one of the most popular forms of exercises to name a few, however strength training tops all of them. According to a research [1] conducted on 40,000 people, there was a significantly lower risk of death to those people who practiced strength training along with aerobic training than aerobic training alone. Gym is a perfect place for strength training using weights. Assistance in workout is easily accessible in gym through a trainer however fitness guidance is that luxury which not everyone can afford. There can be huge injury risks due to bad exercise form when one starts to prioritize repetition count over exercise form. According to the annual *ACSM's Health & Fitness Journal®* [2] the #1 trend for the year 2023 is wearable technology and this is where 'Fit Barbell Classifier' comes into the picture. 'Fit Barbell Classifier' provides an automated solution for tracking the type of exercise and counting repetition of it for barbell training through the device's accelerometer and gyroscope. The user need not to worry about counting the repetitions, it just needs to mindfully focus on the exercise at hand making the whole process safer and more enjoyable. The goal is to explore, build, and evaluate models that can, just like human personal trainers, track exercises and count repetitions. The methods evaluated in the project use a supervised learning approach for classification. Various machine learning algorithms were trained using the collected dataset and accuracies were compared to find and evaluate the right models. Random forest and the simple neural network performed best among other models with an accuracy of 99.37% for both of them. The repetition count algorithm had the mean absolute error as 0.88 for any given set. For video tracking, SVM achieved an accuracy 94.11%.

Chapter 01
INTRODUCTION

1.1 Introduction

Exercise is an essential component of a healthy lifestyle. From improving daily mood to improving the body's resistance towards various health issues like cardiovascular, obesity, diabetes, weakness, exercise has got you covered. From achieving general well-being to specific fitness goals, exercise can play a vital role. Some types of exercises are more beneficial than others [3], one of the best ways to achieve a healthy lifestyle is through *Strength Training*. Strength training involves your muscles contracting against an outside resistance. This resistance could be an external weight or your body weight. An excellent choice for this resistance to be is barbells. Barbells offer a huge variety of exercises that can benefit the full body without any other fancy equipment.

One of the great challenges of training through weights is supervision. Supervision makes the whole training process extremely easy, through the guidance on the types of exercises, the magnitude of weights and the number of repetitions in one particular set. However, not everyone has the luxury to afford a trainer. People training on their own might get tempted into tracking their progress by counting the repetitions and making them lose focus on the form of the exercise, potentially making them prone to injuries.

Last decade has seen a rise in monitoring gadgets like smartwatches enabling people to track the basic activities like walking and running. Accelerometer which measures the G-force and gyroscope which measures the angular velocity are two of the most important sensors which enables this tracking. Although basic human activity tracking has been widely used, there has been little to no work done to track complex movements like that of a barbell exercise. And this is where the *Fit Barbell Classifier* shines. *Fit Barbell Classifier* provides an automated solution for tracking barbell exercises without the manual need of counting the repetitions of a particular exercise, maintaining a journal and more. All the user has to do is mindfully focus on the exercise, making sure the form is proper and correct and the workout is enjoyable. With *Fit Barbell Classifier,* the risks of injury are reduced without the need of a trainer.

1.2 Problem Statement

In day to day life, fitness is one of the most important aspects of human well-being as it famously said “Health is Wealth " and exercise is the best way to maintain physical and mental well-being. Although, the benefits of exercise are well-known, obstacles like very high cost of a personal trainer and the risk of sustaining an injury often hinder people from regularly exercising, To tackle above issues, our problem statement for the project: *Fit Barbell Classifier* is to fulfill the absence of a personal trainer and accurately tracking the type of exercise as well the repetition count of it so that the person can single mindedly focus on the exercise form rather than worrying about the counting repetitions while performing the exercise which will reduce the risks of sustaining injuries.

Fit Barbell Classifier aims to use supervised machine learning models to accurately classify the barbell exercise from the data obtained from accelerometer and gyroscope present in the device attached to the individual’s arm and a custom algorithm to determine the count of repetitions of the exercise. The problem statement also extends to develop a user friendly interface so that any individual can track his/her barbell workout without any hassle.

1.3 Objective

- Create robust machine learning models to classify barbell exercises accurately using accelerometer and gyroscope data.
- Develop an algorithm for precise counting of repetitions within exercises, leveraging patterns in sensor data.
- Seamlessly integrate data processing, modeling, and counting to provide a user-friendly fitness tracking solution and bridge the gap between wearable sensor data and meaningful exercise interpretations

1.4 Significance and Motivation

A very popular article [4] published in 2020 stated that bad exercise form can lead to various injuries like *tendonitis, lower back pain, slip disc, fracture* and this can especially happen when an individual is training alone without any guidance of a trainer. There has been very little to no research in the scene of automated tracking of barbell workouts making the highly powerful sensors like accelerometer and gyroscope useless. There is no mainstream application that provides an automated solution for workout tracking. Figuring out what type of exercise to do, how many repetitions to do and manually counting the repetitions during workout, often hinders the individual from actually doing the training session. The motivation behind this project is to make one such end to end product that solves the problem of not having a personal trainer with you all the time, tracking your workouts, giving a feedback at the end of workout session, so that the user can enjoy the workout by being fully focused on the session and ensuring that the risk of injuries become minimum.

1.5 Organization Of The Report

The report has been organized in a logical and comprehensible manner starting from the first chapter: introduction to the last chapter: conclusion and future scope. A brief overview of each chapter is as follows:

- **Chapter 1, Introduction:** The very first chapter introduces the *what and why* of the project. It discusses the problem statement, objectives and the significance of the chosen project.

- **Chapter 2, Literature Survey:** The second chapter discusses the existing work that has already been done for the project. The two components of this chapter are the overview of the literature review and the gaps in it.

- **Chapter 3, System Development:** This chapter deals with the *how* of the project. It includes the planning and the implementation part of the project. The sections included in this chapter are, requirement and analysis, project design and architecture, data preparation, implementation and key challenges.

- **Chapter 4, Testing:** This chapter deals with the testing phase of the project. Testing is an important part in any of the project life cycle. Testing strategies and test cases are discussed in this chapter.

- **Chapter 5, Results and Evaluation:** In this chapter, all of the results related to the project are discussed. This includes, the evaluation of various machine learning models on many metrics as well evaluation of the custom repetition count algorithm.

- **Chapter 6, Conclusion and Future Scope:** In this chapter, conclusions are drawn. These conclusions include key findings, limitations and contribution to the field. The part of this chapter is future scope which discusses what could be done in the future.

Chapter 02

Literature Survey

2.1 Overview of relevant literature

Researchers have developed a huge form of system learning strategies to classify facts from wearable sensors into various exercising-associated categories.The accuracy of the techniques is decided through the specific assignment and facts used. In precise, the examination suggests that wearable sensors can reliably classify a vast range of sporting events, with accuracy prices ranging from 92% to 97% This information may be used to develop new fitness apps and offer what present accuracy has increased.

Different outcomes have been recorded from different papers. Some of the findings of the paper had been that a way for locating physical games the use of the data that turned into accumulated from wrist-worn sensors turned into located to be 96.2% correct and changed into taken into consideration a technique for classifying exclusive workout speeds by collecting records from a wearable sensor with 95% accuracy. A technique for figuring out activities the use of statistics accrued from hobby-manipulated video display units was discovered to be 92% correct. The new method of classifying daily sports using facts accumulated from cell phone sensors was found to be 93% correct.

Mostly sensors like accelerometers and gyroscopes were used in the following papers. A wide range of exercises were performed such as walking, sitting, running, cycling, and swimming. Some other researchers took gym exercises into consideration such as bicep curls, tricep extensions, lateral raises, chest presses, rows, overhead presses, squats, deadlifts, leg presses, and leg extensions.

These findings imply that wearable sensors could be used to create a variety of new fitness apps, like coaching apps, personalized fitness trackers, and any app that can help users prevent injuries by keeping an eye on their form and technique or track their progress towards fitness objectives.

S.no	Paper Title (Cite)	Journal/Conference (Year)	Tools/Techniques Dataset	Results	Limitations
1	Workout Detection by Wearable Device Data Using Machine Learning[5]	Applied Sciences (Switzerland) (2023)	Random Forest, K-nearest neighbors, and SVM. Accelerometer data collected from a wrist-band.	F-score of 96.3% on Random Forest	The authors collected data only from a single subject.
2.	Video-Based Human Activity Recognition using Deep Learning Methods [6]	Sensors 2023, MDPI	(ResNet) and a vision transformer architecture (ViT) Human motion database (HMDB51	96.7 ± 0.35% and 41.0 ± 0.27% in terms of accuracy in the train and test phase	Poor Test Accuracy
3.	Video Based Exercise Recognition Using GCN [7]	International Conference on Recent Advances in Electrical, Electronics & Digital Healthcare Technologies (2023)	Graph Convolutional Network (GCN) 2 different physical exercises (squats and lunges)	Accuracy of 94.44% and 98.65% of lunges and squats respectively.	Only two exercises were included in the research.
4	Classification of Various Workout Motions Using Wearable Sensors[8]	IEEE World AI IoT Congress (AIIoT)	Support Vector Machine (SVM) using Bayesian optimization	Accuracy of 99% on SVM	The authors didn't use cross-validation and used only SVM

S.no	Paper Title (Cite)	Journal/Conference (Year)	Tools/Techniques Dataset	Results	Limitations
		(2022)	Accelerometer of wearable sensor.		
5	Activity Identification using Supervised Machine Learning on Wearable Activity Tracker Data[9]	Asian Conference on Innovation in Technology (ASIANCON) (2021)	xGBoost, Random Forest, K-nearest neighbors, and Decision Trees. Accelerometer data from Apple Watch and Fitbit	Accuracy of 98% on xG Boost	The authors only used 6 rudimentary activities and didn't include any workout.
6	Human Activity Recognition using ML Techniques [10]	International Journal of Innovative Science and Research Technology (2021)	Decision Trees and Random Forest. UCI Human Activity Recognition	Accuracy of 93% on Random Forest	Only DT and RF were used for classification
7	Fusion of smartphone sensor data for classification of daily user activities [11]	Multimedia Tools and Applications International Journal (2021)	SVM, kNN Accelerometer and gyroscope of smartphone at 50 Hz from 20 participants	Accuracy of 98.32 % for SVM and 97.42 % for kNN.	Only 2 classifiers and 3 activities were used

S.no	Paper Title (Cite)	Journal/Conference (Year)	Tools/Techniques Dataset	Results	Limitations
8	Supervised Machine Learning Applied to Wearable Sensor Data Can Accurately Classify Functional Fitness Exercises Within a Continuous Workout [12]	Frontiers in Bioengineering and Biotechnology (2020)	Random Forest, K-nearest neighbors, and SVM. Accelerometer and gyroscope data of 14 participants performing 4 functional fitness drills.	Accuracy of 97.8% on kNN	The authors only used 4 exercises in the study.
9	ExerSense: Physical Exercise Recognition and Counting Algorithm from Wearable Robust to Positioning [13]	ICT Unit, Faculty of Engineering and Business (2020)	SVM Acceleration data of five types of regular exercises by four different wearable devices.	Accuracy of 93% on SVM	Data was collected under laboratory conditions.
10	Exercise motion classification from large-scale wearable sensor data using convolutional neural networks [14]	IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2017)	3-layered-CNN Accelerometers and gyroscopes	Accuracy of 92.1% on 3-layered CNN	Performed in a noisy environment.

S.no	Paper Title (Cite)	Journal/Conference (Year)	Tools/Techniques Dataset	Results	Limitations
11	Strength Training: A Fitness Application for Indoor Based Exercise Recognition and Comfort Analysis [15]	16th IEEE International Conference on Machine Learning and Applications (2017)	kNN and Neural Networks. Accelerometer data collected from Biceps curl, Chest fly, Row, Push up, Sit up, Squat and Triceps curl	Accuracy of 95.3% on DNN and 77% on KNN	Less exercises were put under consideration
12	Recognizing gym exercises using acceleration data from wearable sensors [16]	IEEE Symposium on Computational Intelligence and Data Mining (2014)	SVM, Random Forest, NN. GeneActiv 3D accelerometer at a frequency of 100 Hz from 36 exercises	99% accuracy on NN	Exercise variability. Each variation can produce different acceleration patterns.

Table 2.1 Literature Survey

2.2 Key Gaps in the Literature

- The data gathered from healthy adults in carefully monitored laboratory environments is the main emphasis of the study publications. This restricts the findings' applicability to real-world situations where people may display a greater variety of physical abilities, health issues, and environmental circumstances. Furthermore, a lot of the research uses small sample sizes, which might not be sufficient to reflect the diversity in human activity patterns.

- The accuracy of interest identification algorithms may be significantly impacted by using the positioning and alignment of wearable sensors. Research articles often make the idea that sensors are placed especially body components, just like the wrist or ankle. However, in realistic implementations, the vicinity of sensors may additionally alternate based totally on personal possibilities and the nature of the interest. Additionally, the orientation of sensors won't continually be consistent throughout various activities and may impact the first-class of records amassed.

- The class of activities within particular domains, like fitness or each day dwelling, is the main emphasis of the examination research. But humans additionally regularly take part in move-domain activities, which includes walking to work or doing home tasks whilst running out. Reliable activity category across domain names is critical for wearable sensor systems to be evolved that could provide deep insights into an individual's whole pastime profile.

- Since wearable sensor structures frequently run on batteries, strength economy is an important thing to recollect. Less interest is located on optimizing electricity use and greater on the accuracy of hobby popularity algorithms within the studies articles. Furthermore, wearable gadgets' overall performance in real time might be suffering from the computational value of algorithms, mainly on gadgets with low processing strength.

- The aforementioned shortcomings underscore the need for added investigation to address the restrictions of extant wearable sensor-based pastime detection structures and broaden their relevance in actual-world scenarios. By filling in those gaps, scientists can create wearable sensor structures that are extra dependable, accurate, and adaptable. These structures can assist human beings achieve their fitness and properly-being objectives with the aid of presenting individualized insights on their interest styles.

Chapter 03

System Development

3.1 Requirement and Analysis

3.1.1 Technical Requirements

- **Language Used:**
    - Python 3.10.12
- **Software Used:**
    - Python 3.10.12
    - Google Colab
    - VS Code
- **Libraries:**
    - Numpy
    - Matplotlib
    - Pandas
    - Scikit-Learn
    - Plotly
    - Glob
    - IPython
    - Seaborn
    - Scipy
    - Unittest
    - MediaPipe
    - OpenCV

3.1.2 Functional Requirements

- The ability to accept and process sensor data of exercises as input.
- The ability to classify the input data into one of several barbell exercises.
- The ability to output the predicted exercise with a corresponding accuracy.
- The ability to handle input sensor data with varying levels of noise or distortion.
- The ability to handle a large number of input data and perform classification efficiently.
- The ability to be easily trained and fine-tuned on new datasets.

3.1.3 Non Functional Requirements

- Performance: The system should be able to train the machine learning models within a reasonable amount of time. The classification time for new unseen data should be less as well.

- Scalability: As the project grows in size, the system should not falter and must be able to handle large amounts of data.

- Reliability: The system should be reliable enough to handle error gracefully. The system must be robust to failures.

- Usability: The system should be hassle free and friendly to the user. Any internal complications must be hidden from the end user.

- Portability: The system should be portable to different types of operating systems.

- Maintainability: The system must be easy to update as well as maintain. The system should be written in such a way, any updates on one module must not directly affect any other module

3.2 Project Design and Architecture

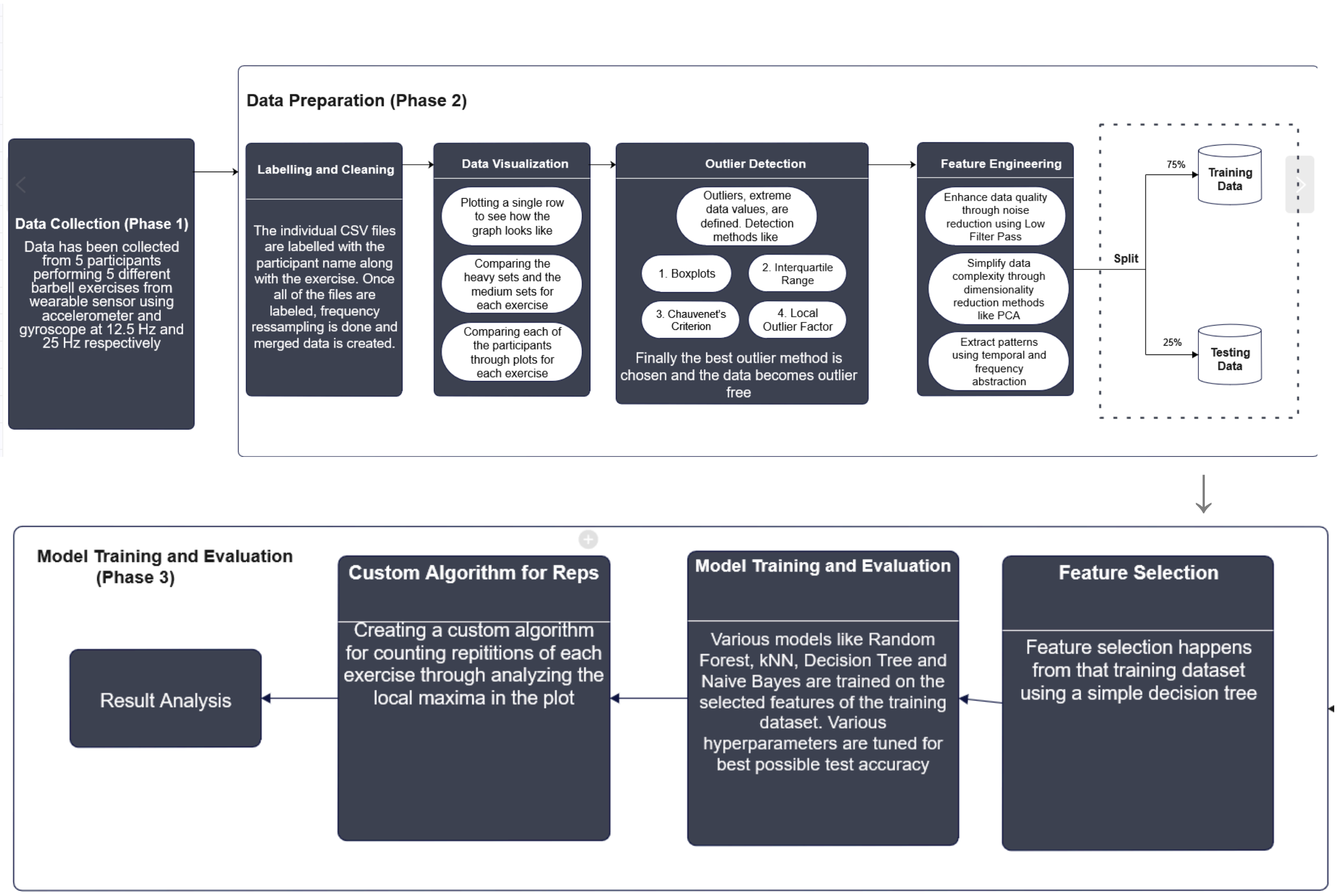


Figure 3.1 Project Design Architecture

The figure 3.1 shows an extensive project design plan for sensor tracking

Phase 1: Data Collection

Accelerometer & Gyroscope:

The data used for the project is a sensor based time series data based on the movement pattern during the barbell exercises. The author used five subjects including four males and one female and were asked to perform barbell exercises namely, deadlift, overhead press, squat, bench press and barbell row while wearing a wearable device having sensors. The wearable device consists of two sensors i.e. accelerometer and gyroscope. The accelerometer measures the gForce and the gyroscope measures the rotation. The accelerometer was configured to measure at 12.50 Hz whereas the gyroscope was configured to measure at 25.00 Hz. The subjects performed the exercises in two sets, a heavy and medium set. In between the sets, the resting period was also monitored to better generalize the model to be trained on the data. Raw data resulted in a total *187 csv* files.

Video Evaluation:

The first phase of video evaluation was to collect video data of exercises which would be used to train the model. A script has been written to collect the data which would make use of *OpenCV* [17] and *MediaPipe* [18] library to capture and annotate the frames of the videos, which then would be stored in a folder where each subfolder would have data of one particular exercise.

Phase 2: Data Preparation

Accelerometer & Gyroscope:

During the data collection phase, a total of *187 csv* files were generated which had the participant name, type of exercise and set type mentioned in it. Firstly, *labeling and cleaning* is done whose purpose is to create a dataframe from which features can be engineered and selected for model training. The next step in this phase is *data visualization* whose main goal is to identify patterns in the data through visualization. *Outlier detection* is the next step in this phase. Detecting outliers is an important step, as a model trained with outliers can have less accuracy. Various methods like *IQR, Chauvenet's criterion* [19] *and Local Outlier Factor* were used. The next step is f*eature engineering* so that engineered features can be passed in model training. Various feature engineering techniques like *Butterworth Low Pass Filter* [20]*, PCA, Sum of Squared features, Temporal abstraction, Frequency abstraction and cluster feature* were carried out. A final dataframe was exported with all the basic features as well the engineered features for model training. A lot of above methods are studied from a standard book for machine learning on sensor data named Machine Learning for the Quantified Self: On the Art of Learning from Sensory Data [21].

Video Evaluation:

During the data collection phase, a total of *150 video files (50 of each category)* are acquired. The main preprocessing task is to label the annotated frames of the videos which then can be used for supervised model building..

Phase 3: Model Training and Evaluation and Repetition Count

Accelerometer & Gyroscope:

Model training is that phase where learning actually takes place. The training data is passed to the model and the model learns from the training data by finding different patterns among the training features. At the end of feature engineering, we had 116 features in total. This includes 6 basic features, 3 PCA features, 2 sum of squared features, 16 time related features, 88 frequency related features and 1 cluster feature. Multiple sets of features are created to train the models on each one of them to find perfect accuracy to processing time ratio.

Feature set A includes basic features, feature set B includes feature set A plus PCA and sum of squared features, feature set C has all the features of feature set B and temporal features and the last feature set D contains feature set C plus all of frequency features. The length of feature sets A, B, C, and D are 6, 11, 27 and 116 respectively.

A number of models like logistic regression, support vector machine, naive bayes, kNN, decision tree and random forest are trained on each of the feature sets. Since feature set 4 contains 116 features and hence more processing time, we have come up with stacking of models and then apply the stacked models on feature set 3 to find the perfect balance between performance(accuracy) and processing model.

3 stacked models are created with the following definitions:

- Stacked model 1:
    - Weak Learners: Decision Tree, Logistic Regression, Support Vector Machine, k-Nearest Neighbors and Naive Bayes.
    - Final Estimator: Logistic Regression
    - Cross Folds: 5

- Stacked model 2:
    - Weak Learners: Logistic Regression, Random Forest and Decision Tree.
    - Final Estimator: Random Forest
    - Cross Folds: 5

- Stacked model 3:
    - Weak Learners: Random Forest, Decision Tree, Logistic Regression, Support Vector Machine, k-Nearest Neighbors and Naive Bayes.
    - Final Estimator: Logistic Regression
    - Cross Folds: 5

Model evaluation is that phase where the model is tested on the unseen data. A good model performs well on both training and testing data. To ensure this is the case, model evaluation is done. Various evaluation metrics like *accuracy, precision, recall* and *f-score* are calculated on the testing data. Based on the values of these evaluation metrics, further steps are taken place if required.

For the final part of this phase, a repetition count algorithm is built which counts the number of repetitions given any set. The basic idea was to smooth out the curve of the most dominant feature and count the number of local maxima in it. The repetition count algorithm was tested against a benchmark dataframe through the *mean absolute error* parameter.

Video Evaluation:

This final phase begins with the splitting the data into training and testing datasets. A ratio of 9:1 is chosen as training and testing data. After that *SVM* is trained on the frames of the videos stored. Once the model is fully trained it then is evaluated against the testing dataset on various parameters like *accuracy, precision, recall* and *f-score.*

As the final step of this phase, a repetition count algorithm is written which is based on *MediaPipe* annotations of the images. The basic idea is to calculate angles between joints and then depending upon the angles of the images, classify the pose into one of the predefined labels. When everything is done, the model and the repetition count algorithm is tested on real data.

3.3 Data Preparation

1. Data Collection

a. The data used for the project is a sensor based time series data based on the movement pattern during the barbell exercises. Five subjects including four males and one female were asked to perform barbell exercises namely, deadlift, overhead press, squat, bench press and barbell row while wearing a wearable device having sensors.
b. The wearable device consists of two sensors i.e. accelerometer and a gyroscope. The accelerometer measures the acceleration and the gyroscope measures the angular velocity. The accelerometer was configured to measure at 12.50 Hz whereas the gyroscope was configured to measure at 25.00 Hz.
c. The subjects performed the exercises in two sets, a heavy and medium set. In between

the sets, the resting period was also monitored to better generalize the model to be trained on the data.

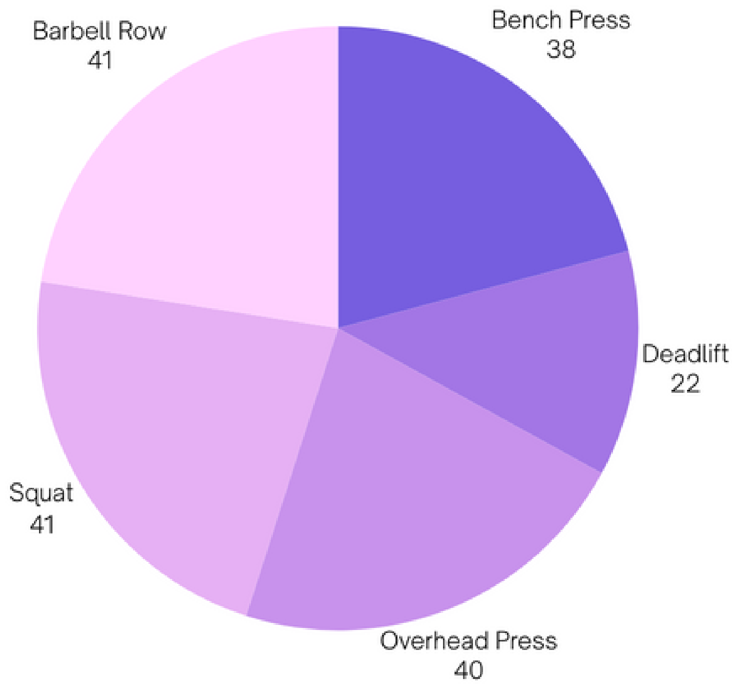


Figure 3.2 Dataset Pie Chart

The figure 3.2 shows a pie chart of how many csv files are there from each of the exercises.

2. Labeling and Cleaning

a. During the data collection phase, a total of *187 csv* files were generated which had the participant name, type of exercise and set type mentioned in it. In this part, each accelerometer and gyroscope files are separated and made into different *Panda's dataframes.*

b. Now, participant name (participant), type of exercise (label) and category of set (set) are extracted from the file name and then added into the *dataframe.* After each of the acceleration and gyroscope dataframe are labeled, and all of the dataframes are merged into a single dataframe which would be used throughout the project.

c. For cleaning purposes, the *epoch attribute* is set as the index and some redundant attributes are deleted. Since, the gyroscope was configured at 25.00 Hz and accelerometer was configured at 12.50 Hz, merging the two dataframes caused a lot of null values in it. To handle this problem, frequency resampling was done at an interval of 200ms.

epoch (ms)	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	participant	label	category	set
2019-01-11 15:08:05.200	0.013500	0.977000	-0.071000	-1.8904	2.4392	0.9388	B	bench	heavy	76.0
2019-01-11 15:08:05.400	-0.001500	0.970500	-0.079500	-1.6826	-0.8904	2.1708	B	bench	heavy	76.0
2019-01-11 15:08:05.600	0.001333	0.971667	-0.064333	2.5608	-0.2560	-1.4146	B	bench	heavy	76.0
2019-01-11 15:08:05.800	-0.024000	0.957000	-0.073500	8.0610	-4.5244	-2.0730	B	bench	heavy	76.0
2019-01-11 15:08:06.000	-0.028000	0.957667	-0.115000	2.4390	-1.5486	-3.6098	B	bench	heavy	76.0
...	...	...	...	...	...	...	...	...	...	...
2019-01-20 17:33:27.000	-0.048000	-1.041500	-0.076500	1.4146	-5.6218	0.2926	E	row	medium	15.0
2019-01-20 17:33:27.200	-0.037000	-1.030333	-0.053333	-2.7684	-0.5854	2.2440	E	row	medium	15.0
2019-01-20 17:33:27.400	-0.060000	-1.031000	-0.082000	2.8416	-5.1342	-0.1220	E	row	medium	15.0
2019-01-20 17:33:27.600	-0.038667	-1.025667	-0.044667	-0.2318	0.2562	1.1220	E	row	medium	15.0
2019-01-20 17:33:27.800	-0.044000	-1.034000	-0.059000	1.0980	-4.0240	0.9760	E	row	medium	15.0

9009 rows × 10 columns

Figure 3.3 Resampled Dataset

The figure 3.3 shows the resampled data frame obtained at the end of labeling and cleaning.

3. Data Visualization

a. The data visualization part of the project deals with visualization of data to gain insights of the underlying patterns hidden in the data and check for whether some of our assumptions match with the visualization or not.

b. To compare different types of exercises and how the sensor data differs in each, plots are drawn and conclusions are made. We need to make sure we know the difference between a heavy set and a medium set, for that we can draw a plot for each of them and then compare between them.

c. To make a generalized model, we need to look for the variance in data collected for the same exercise but different subject, for this we can visualize our data and check for our assumptions.

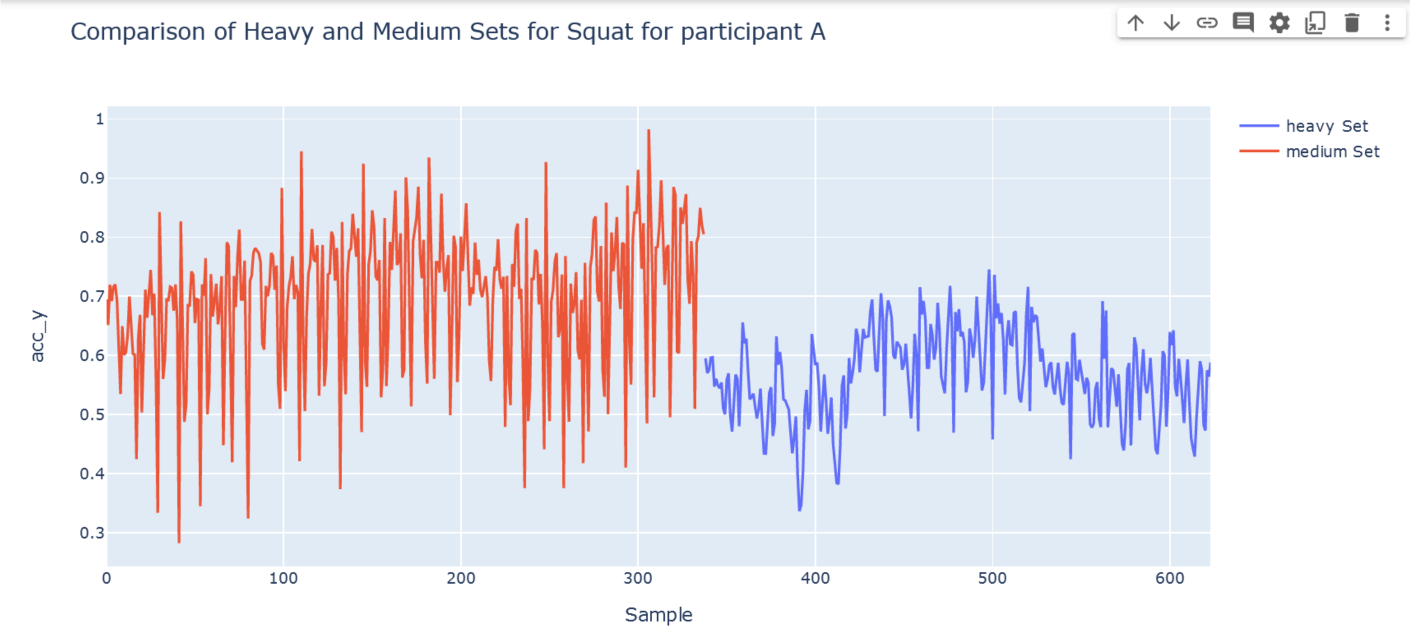


Figure 3.4 Comparison of heavy and medium set

The figure 3.4 visualizes the comparison between a heavy set and a medium set for participant A doing a squat. The heavy set produces a low range of acceleration in y direction when compared to that of a medium set.

4. Outlier Detection

a. The quality of the model directly depends on the quality of the data on which it is trained and any real world data is filled with noise. This noise is often termed as outliers which are basically, extreme values of the current context. Outliers are bad as they can give false information to the model and hence decrease the accuracy of it.

b. To detect outliers, various outlier detection methods like box-plots using the IQR, Chauvenet's criterion and Local Outlier Factor [22] are used and then finally, Chauvenet's criterion was chosen to detect the outliers and then mark them as null values. The null values are dealt with in the later part.

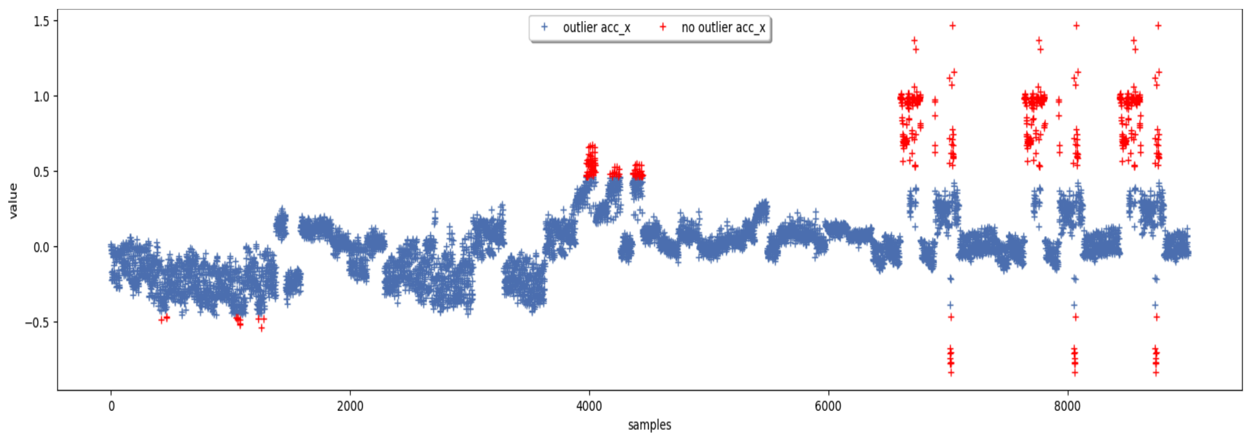


Figure 3.5 Outlier Detection using IQR

The figure 3.5 visualizes outliers with the help of IQR method

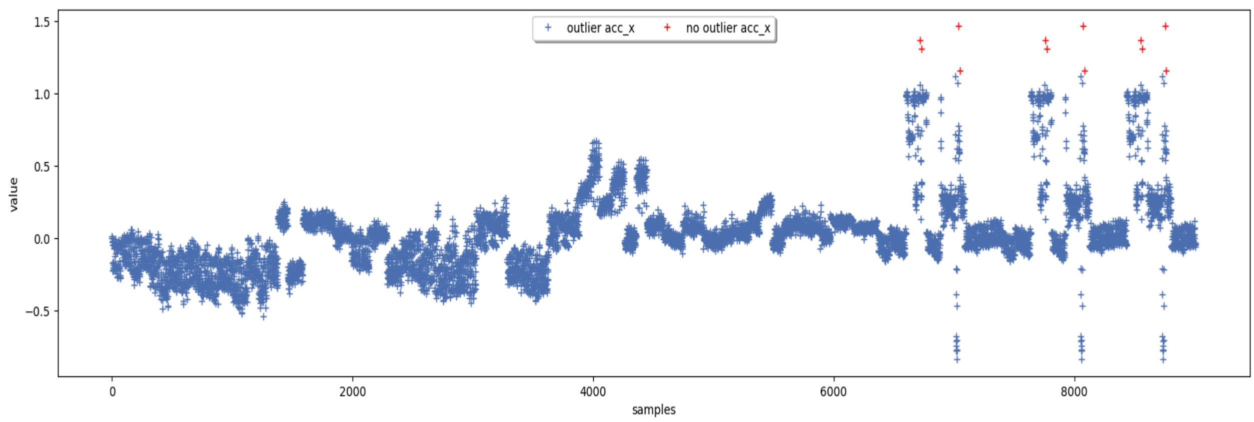


Figure 3.6 Outlier Detection using Chauvenet's Criterion

The figure 3.6 visualizes outliers with the help of Chauevenet's Criterion

5. Feature Engineering

a. Feature engineering is an important part of data preparation where we essentially try to manipulate/clean existing features or engineer new features from the existing ones so that maximum information in minimum number of features can be captured and then passed to the model for learning.

b. In this part, null values are interpolated, *Butterworth Low Pass Filter* is applied to each feature for the smoothening of the curves.

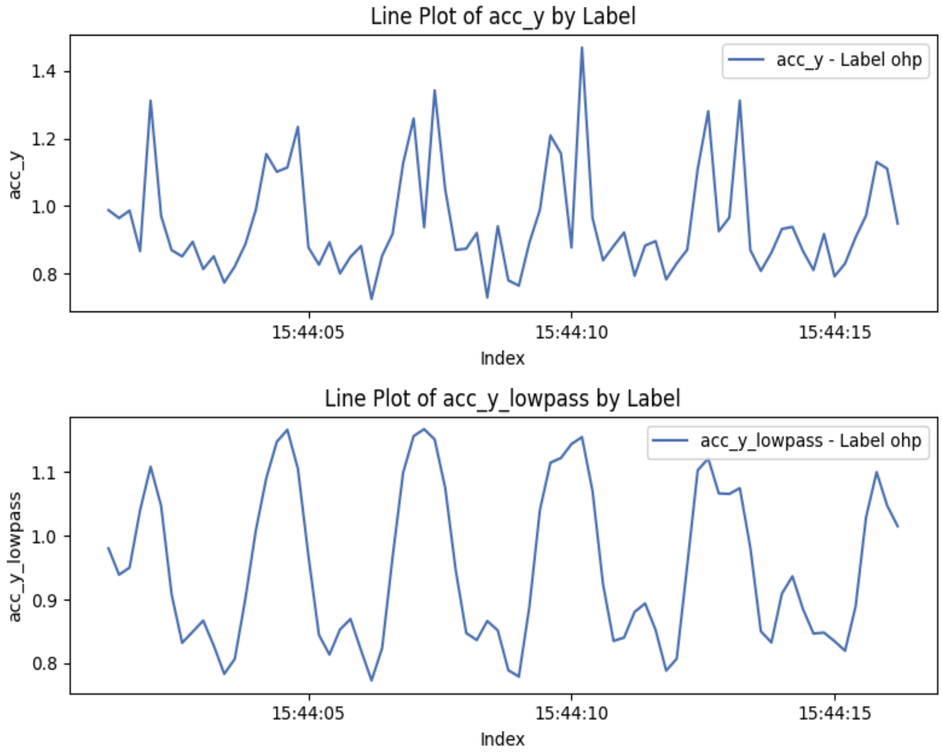


Figure 3.7 Butterworth Low Pass Filter

The figure 3.7 visualizes the smoothing of the curve through the Butterworth Low Pass Filter.

c. After Butterworth Low Pass Filter, Principal Component Analysis was performed. A total of three principal components were extracted and were labeled as `pca_1`, `pca_2` and `pca_3`.

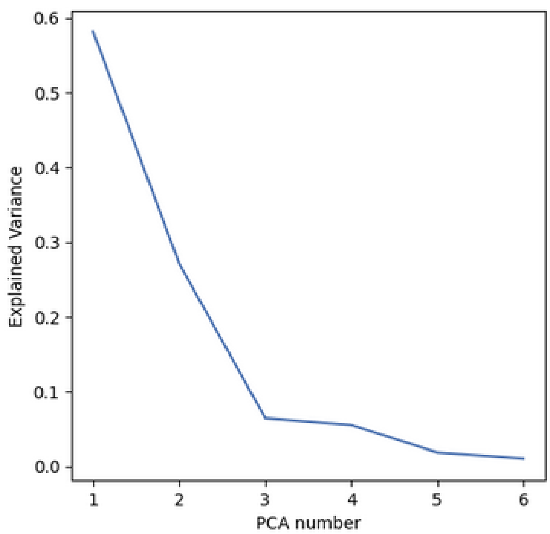


Figure 3.8 Elbow Graph of Explained Variance and PCA Number

Figure 3.8 depicts the elbow graph to correctly identify number of components

d. Sum of Squared features are also engineered to reduce bias towards device's orientation. \`acc_r\` was added for accelerometer and \`gyr_r\` was added for gyroscope readings. $acc\_r = \sqrt{(acc\_x^2 + acc\_y^2 + acc\_z^2)}$ and $gyr\_r = \sqrt{(gyr\_x^2 + gyr\_y^2 + gyr\_z^2)}$.

e. *Temporal abstraction* is also done through the help of rolling mean and rolling standard deviation. It was applied to all of the six basic features and squared features and resulted in a total of 16 new features.

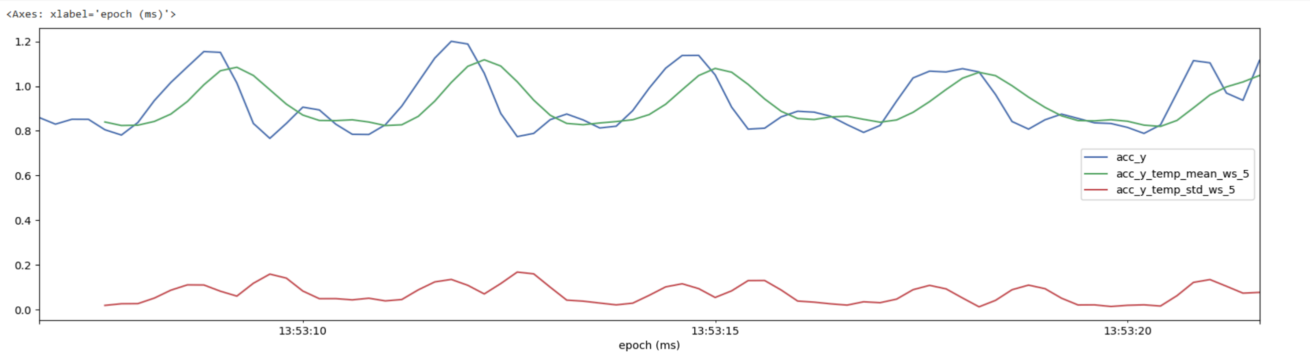


Figure 3.9 Temporal Features

Figure 3.8 visualizes the temporal features generated using rolling means and standard deviation

f. After temporal abstraction, frequency abstraction was done with the help of Fast Fourier Transform [23]. Each set was given as an input and it resulted in 88 new frequency features.

g. Feature engineering is concluded by clustering using the *K-Means algorithm* to generate a cluster feature. At the end of feature engineering, a total of *116 features* were present in the dataframe.

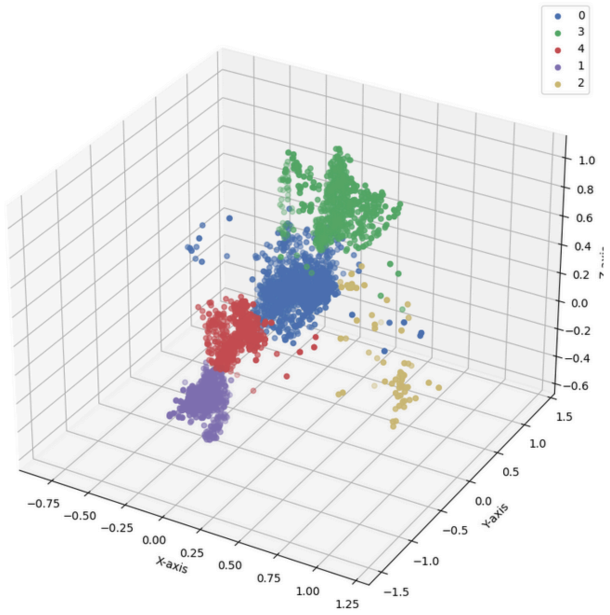


Figure 3.10 Cluster Feature

Figure 3.9 visualizes the clustering feature generated using K-Means clustering

3.4 Implementation

3.4.1 Screenshots of the various stages of the Project

Type: For tracking through acceleration and gyroscope.

Stage 1: Module imports and labeling and cleaning

One of the most popular and easiest libraries for data handling in python is pandas [24]. It makes the data manipulation process extremely easy for the developer by providing many functions for the manipulation tasks.

```
!pip install IPython  #here
```

```
import numpy as np
import pandas as pd
from glob import glob
from IPython.display import display
```

```
single_file_acc=pd.read_csv("/content/drive/MyDrive/MetaMotion/A-bench-heavy2-rpe8_MetaWear_2019-01-11T16.10.08.270_C42732BE255C_Accelerometer_12.500Hz_1.4.4.csv")
```

```
single_file_acc
```

	epoch (ms)	time (01:00)	elapsed (s)	x-axis (g)	y-axis (g)	z-axis (g)
0	1547219408431	2019-01-11T16:10:08.431	0.00	0.010	0.964	-0.087
1	1547219408511	2019-01-11T16:10:08.511	0.08	0.000	0.961	-0.069
2	1547219408591	2019-01-11T16:10:08.591	0.16	0.001	0.974	-0.087
3	1547219408671	2019-01-11T16:10:08.671	0.24	-0.012	0.971	-0.084
4	1547219408751	2019-01-11T16:10:08.751	0.32	-0.013	0.954	-0.094
...	...	...	...	...	...	...
201	1547219424511	2019-01-11T16:10:24.511	16.08	0.027	0.970	-0.091
202	1547219424591	2019-01-11T16:10:24.591	16.16	0.010	0.990	-0.068
203	1547219424671	2019-01-11T16:10:24.671	16.24	0.011	0.978	-0.091
204	1547219424751	2019-01-11T16:10:24.751	16.32	0.008	0.937	-0.085
205	1547219424831	2019-01-11T16:10:24.831	16.40	0.021	0.966	-0.108

206 rows × 6 columns

```
single_file_gyro=pd.read_csv("/content/drive/MyDrive/MetaMotion/A-bench-heavy2-rpe8_MetaWear_2019-01-11T16.10.08.270_C42732BE255C_Gyroscope_25.000Hz_1.4.4.csv")
```

```
single_file_gyro #to here
```

	epoch (ms)	time (01:00)	elapsed (s)	x-axis (deg/s)	y-axis (deg/s)	z-axis (deg/s)
0	1547219408351	2019-01-11T16:10:08.351	0.00	0.122	-5.488	-3.841
1	1547219408391	2019-01-11T16:10:08.391	0.04	2.195	-9.695	-0.610
2	1547219408431	2019-01-11T16:10:08.431	0.08	2.622	-8.110	-4.024
3	1547219408471	2019-01-11T16:10:08.471	0.12	1.951	-4.695	-4.634
4	1547219408511	2019-01-11T16:10:08.511	0.16	1.524	-2.561	-2.500
...	...	...	...	...	...	...
409	1547219424711	2019-01-11T16:10:24.711	16.36	2.195	-1.402	4.268
410	1547219424751	2019-01-11T16:10:24.751	16.40	4.451	-1.220	0.244
411	1547219424791	2019-01-11T16:10:24.791	16.44	7.195	-4.329	6.524
412	1547219424831	2019-01-11T16:10:24.831	16.48	8.963	-1.037	-0.366
413	1547219424871	2019-01-11T16:10:24.871	16.52	-3.476	2.134	-5.305

414 rows × 6 columns

Snippet 3.1

```
#here
acc_df
```

epoch (ms)	epoch (ms)	time (01:00)	elapsed (s)	x-axis (g)	y-axis (g)	z-axis (g)	participant	label	category	set
2019-01-18 17:33:08.416	1547832788416	2019-01-18T18:33:08.416	0.00	-0.012	-1.007	-0.097	D	row	medium	1
2019-01-18 17:33:08.496	1547832788496	2019-01-18T18:33:08.496	0.08	-0.010	-1.020	-0.106	D	row	medium	1
2019-01-18 17:33:08.576	1547832788576	2019-01-18T18:33:08.576	0.16	-0.012	-1.025	-0.095	D	row	medium	1
2019-01-18 17:33:08.656	1547832788656	2019-01-18T18:33:08.656	0.24	-0.012	-1.032	-0.110	D	row	medium	1
2019-01-18 17:33:08.736	1547832788736	2019-01-18T18:33:08.736	0.32	-0.002	-1.032	-0.099	D	row	medium	1
...	...	...	...	...	...	...	...	...	...	...
2019-01-11 15:49:16.116	1547221756116	2019-01-11T16:49:16.116	21.20	-0.318	1.236	-0.010	B	ohp	medium	94
2019-01-11 15:49:16.196	1547221756196	2019-01-11T16:49:16.196	21.28	-0.204	0.862	0.031	B	ohp	medium	94
2019-01-11 15:49:16.276	1547221756276	2019-01-11T16:49:16.276	21.36	-0.194	0.772	0.052	B	ohp	medium	94
2019-01-11 15:49:16.356	1547221756356	2019-01-11T16:49:16.356	21.44	-0.203	0.891	0.014	B	ohp	medium	94
2019-01-11 15:49:16.436	1547221756436	2019-01-11T16:49:16.436	21.52	-0.164	1.002	0.001	B	ohp	medium	94

23578 rows × 10 columns

Snippet 3.2

```
#here
gyr_df
```

epoch (ms)	epoch (ms)	time (01:00)	elapsed (s)	x-axis (deg/s)	y-axis (deg/s)	z-axis (deg/s)	participant	label	category	set
2019-01-15 13:27:00.993	1547558820993	2019-01-15 14:27:00.993	0.00	2.317	-2.500	-9.695	E	bench	heavy	1
2019-01-15 13:27:01.033	1547558821033	2019-01-15 14:27:01.033	0.04	3.110	-0.793	-9.207	E	bench	heavy	1
2019-01-15 13:27:01.073	1547558821073	2019-01-15 14:27:01.073	0.08	3.232	2.134	-2.500	E	bench	heavy	1
2019-01-15 13:27:01.113	1547558821113	2019-01-15 14:27:01.113	0.12	2.988	2.195	-11.159	E	bench	heavy	1
2019-01-15 13:27:01.153	1547558821153	2019-01-15 14:27:01.153	0.16	18.354	-7.866	-14.451	E	bench	heavy	1
...	...	...	...	...	...	...	...	...	...	...
2019-01-14 13:50:00.395	1547473800395	2019-01-14T14:50:00.395	13.72	-7.927	0.244	0.793	A	ohp	heavy	93
2019-01-14 13:50:00.435	1547473800435	2019-01-14T14:50:00.435	13.76	-9.695	-1.463	1.524	A	ohp	heavy	93
2019-01-14 13:50:00.475	1547473800475	2019-01-14T14:50:00.475	13.80	-11.951	-1.402	0.793	A	ohp	heavy	93
2019-01-14 13:50:00.515	1547473800515	2019-01-14T14:50:00.515	13.84	-6.646	1.524	0.000	A	ohp	heavy	93
2019-01-14 13:50:00.555	1547473800555	2019-01-14T14:50:00.555	13.88	0.549	3.049	7.683	A	ohp	heavy	93

47218 rows × 10 columns

Snippet 3.3

```
[ ] #here
    data_merged=pd.concat([acc_df.iloc[:,:3],gyr_df],axis=1)
```

```
data_merged
```

epoch (ms)	x-axis (g)	y-axis (g)	z-axis (g)	x-axis (deg/s)	y-axis (deg/s)	z-axis (deg/s)	participant	label	category	set
2019-01-11 15:08:04.950	NaN	NaN	NaN	-10.671	-1.524	5.976	B	bench	heavy	76.0
2019-01-11 15:08:04.990	NaN	NaN	NaN	-8.720	-2.073	3.171	B	bench	heavy	76.0
2019-01-11 15:08:05.030	NaN	NaN	NaN	0.488	-3.537	-4.146	B	bench	heavy	76.0
2019-01-11 15:08:05.070	NaN	NaN	NaN	0.244	-5.854	3.537	B	bench	heavy	76.0
2019-01-11 15:08:05.110	NaN	NaN	NaN	-0.915	0.061	-2.805	B	bench	heavy	76.0
...	...	...	...	...	...	...	...	...	...	...
2019-01-20 17:35:13.382	-0.060	-1.021	-0.058	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-01-20 17:35:13.462	-0.035	-1.037	-0.026	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-01-20 17:35:13.542	-0.045	-1.029	-0.033	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-01-20 17:35:13.622	-0.039	-1.027	-0.039	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2019-01-20 17:35:13.702	-0.049	-1.031	-0.049	NaN	NaN	NaN	NaN	NaN	NaN	NaN

69677 rows × 10 columns

Snippet 3.4

```
#here
data_resampled=pd.concat(
    [df.resample(rule="200ms").apply(sampling).dropna() for df in days]
)
```

```
[ ] #to here
    data_resampled
```

epoch (ms)	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	participant	label	category	set
2019-01-11 15:08:05.200	0.013500	0.977000	-0.071000	-1.8904	2.4392	0.9388	B	bench	heavy	76.0
2019-01-11 15:08:05.400	-0.001500	0.970500	-0.079500	-1.6826	-0.8904	2.1708	B	bench	heavy	76.0
2019-01-11 15:08:05.600	0.001333	0.971667	-0.064333	2.5608	-0.2560	-1.4146	B	bench	heavy	76.0
2019-01-11 15:08:05.800	-0.024000	0.957000	-0.073500	8.0610	-4.5244	-2.0730	B	bench	heavy	76.0
2019-01-11 15:08:06.000	-0.028000	0.957667	-0.115000	2.4390	-1.5486	-3.6098	B	bench	heavy	76.0
...	...	...	...	...	...	...	...	...	...	...
2019-01-20 17:33:27.000	-0.048000	-1.041500	-0.076500	1.4146	-5.6218	0.2926	E	row	medium	15.0
2019-01-20 17:33:27.200	-0.037000	-1.030333	-0.053333	-2.7684	-0.5854	2.2440	E	row	medium	15.0
2019-01-20 17:33:27.400	-0.060000	-1.031000	-0.082000	2.8416	-5.1342	-0.1220	E	row	medium	15.0
2019-01-20 17:33:27.600	-0.038667	-1.025667	-0.044667	-0.2318	0.2562	1.1220	E	row	medium	15.0
2019-01-20 17:33:27.800	-0.044000	-1.034000	-0.059000	1.0980	-4.0240	0.9760	E	row	medium	15.0

9009 rows × 10 columns

Snippet 3.5

The code snippet 3.1 to 3.5 shows the final labeled and cleaned data which would be in next steps for feature engineering and then later on modeling.

Stage 2: Data Visualization

Phase 2: Data Visualization

Critical Questions:

1. Why we should bother about data visualization?

```
[40]
#import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
[41]
#Plot single column of a specific set
set_df=data_resampled[data_resampled["set"]==1]
```

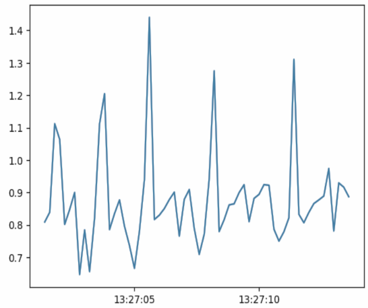
```
set_df
```

epoch (ms)	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	participant	label	category	set
2019-01-15 13:27:01.400	-0.192000	0.809000	-0.409333	14.0858	-10.1828	-2.9512	E	bench	heavy	1
2019-01-15 13:27:01.600	-0.211500	0.839000	-0.442500	12.3658	-15.3172	16.1342	E	bench	heavy	1
2019-01-15 13:27:01.800	-0.271667	1.112000	-0.526333	-4.7318	-2.9144	-5.2804	E	bench	heavy	1
2019-01-15 13:27:02.000	-0.294000	1.063500	-0.516000	-9.9878	2.7072	-19.3170	E	bench	heavy	1
2019-01-15 13:27:02.200	-0.281000	0.801333	-0.432667	-6.4268	1.7926	-1.3780	E	bench	heavy	1
...	...	...	...	...	...	...	...	...	...	...
2019-01-15 13:27:12.800	-0.190000	0.974000	-0.344500	-14.6218	5.2928	20.5976	E	bench	heavy	1
2019-01-15 13:27:13.000	-0.106000	0.781333	-0.290667	5.6708	0.6950	8.2684	E	bench	heavy	1
2019-01-15 13:27:13.200	-0.101000	0.929500	-0.330500	14.9512	-2.8778	0.8294	E	bench	heavy	1
2019-01-15 13:27:13.400	-0.090667	0.916333	-0.364000	3.9636	-0.4024	0.5608	E	bench	heavy	1
2019-01-15 13:27:13.600	-0.090500	0.887000	-0.373500	8.3690	-1.8595	0.7015	E	bench	heavy	1

Snippet 3.6

```
#here
plt.plot(set_df[["acc_y"]]) #-> this plot shows how long the set is
```

```
[<matplotlib.lines.Line2D at 0x7b0988fdad10>]
```

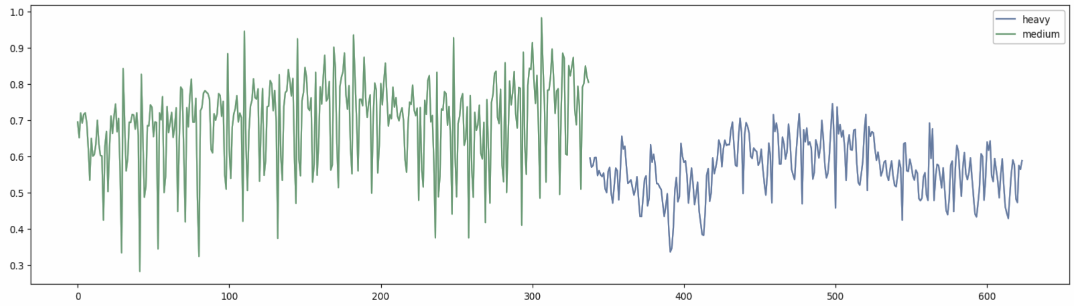


Snippet 3.7

```
[ ] #here

#Comparing medium and heavy set
category_df=data_resampled.query("label=='squat'").query("participant=='A'").reset_index() #Selects only A's squat data and resets the index
```

```
#to here
fig,ax=plt.subplots()
category_df.groupby(["category"])["acc_y"].plot()
ax.set_ylabel="acc_y"
ax.set_xlabel="Samples"
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b097fbacbb0>
```



Snippet 3.8

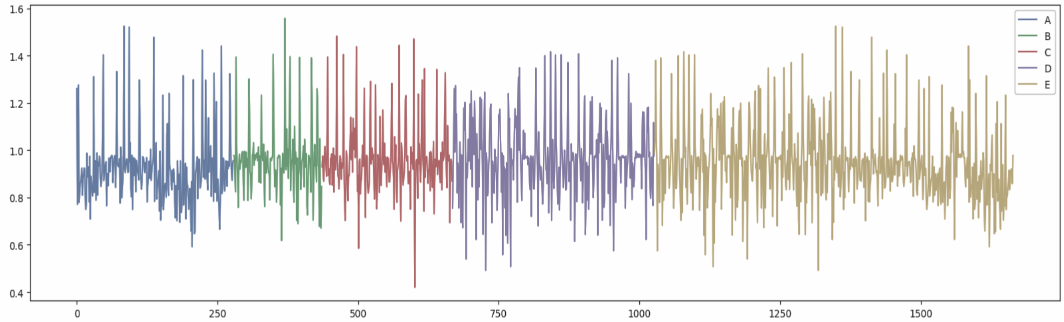
For visualization purposes, matplotlib [25] is used. Matplotlib makes is extremely easy for the developer to visualize the data with various plots like charts, histogram, etc

The snippet 3.8 visualizes the comparison between a heavy set and a medium set for participant A doing a squat. The heavy set produces a low range of acceleration in y direction when compared to that of a medium set.

```
#here
#Compare participants
participant_df=data_resampled.query("label=='bench'").sort_values("participant").reset_index()
```

```
#to here
fig,ax=plt.subplots()
participant_df.groupby(["participant"])["acc_y"].plot()
ax.set_ylabel="acc_y"
ax.set_xlabel="Samples"
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b097fade830>
```



Snippet 3.9

The snippet 3.9 visualizes the comparison among all the participants doing a bench press. It is important to check the movement patterns of all of the participants for better modeling.

Stage 3: Outlier Detection

Phase 3: **Outlier Detection**

```
#here
data_resampled
```

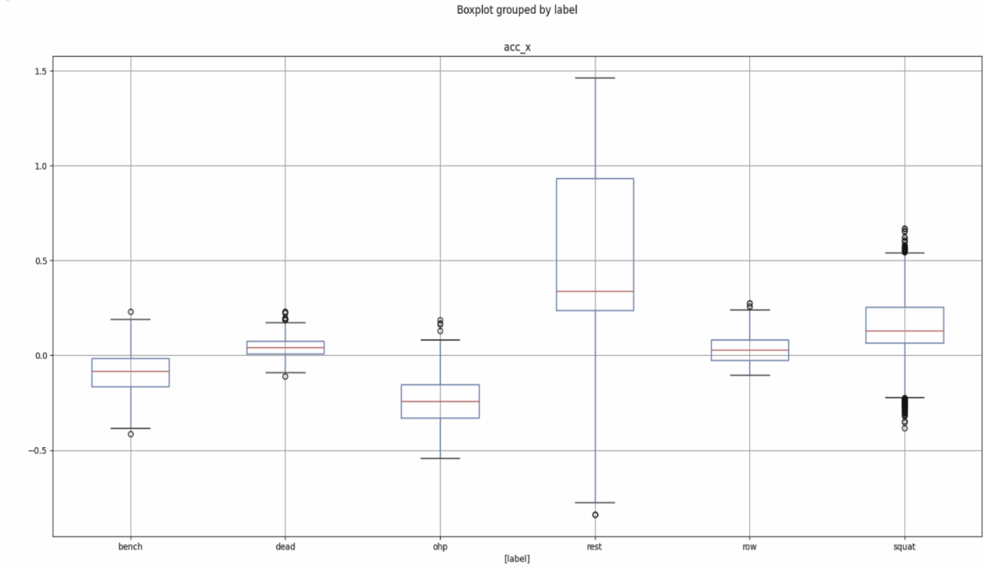
epoch (ms)	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	participant	label	category	set
2019-01-11 15:08:05.200	0.013500	0.977000	-0.071000	-1.8904	2.4392	0.9388	B	bench	heavy	76
2019-01-11 15:08:05.400	-0.001500	0.970500	-0.079500	-1.6826	-0.8904	2.1708	B	bench	heavy	76
2019-01-11 15:08:05.600	0.001333	0.971667	-0.064333	2.5608	-0.2560	-1.4146	B	bench	heavy	76
2019-01-11 15:08:05.800	-0.024000	0.957000	-0.073500	8.0610	-4.5244	-2.0730	B	bench	heavy	76
2019-01-11 15:08:06.000	-0.028000	0.957667	-0.115000	2.4390	-1.5486	-3.6098	B	bench	heavy	76
...	...	...	...	...	...	...	...	...	...	...
2019-01-20 17:33:27.000	-0.048000	-1.041500	-0.076500	1.4146	-5.6218	0.2926	E	row	medium	15
2019-01-20 17:33:27.200	-0.037000	-1.030333	-0.053333	-2.7684	-0.5854	2.2440	E	row	medium	15
2019-01-20 17:33:27.400	-0.060000	-1.031000	-0.082000	2.8416	-5.1342	-0.1220	E	row	medium	15
2019-01-20 17:33:27.600	-0.038667	-1.025667	-0.044667	-0.2318	0.2562	1.1220	E	row	medium	15
2019-01-20 17:33:27.800	-0.044000	-1.034000	-0.059000	1.0980	-4.0240	0.9760	E	row	medium	15

9009 rows × 10 columns

Snippet 3.10

```
#here
#Method 1: Boxplots

data_resampled[["acc_x","label"]].boxplot(by="label",figsize=(20,10))
```

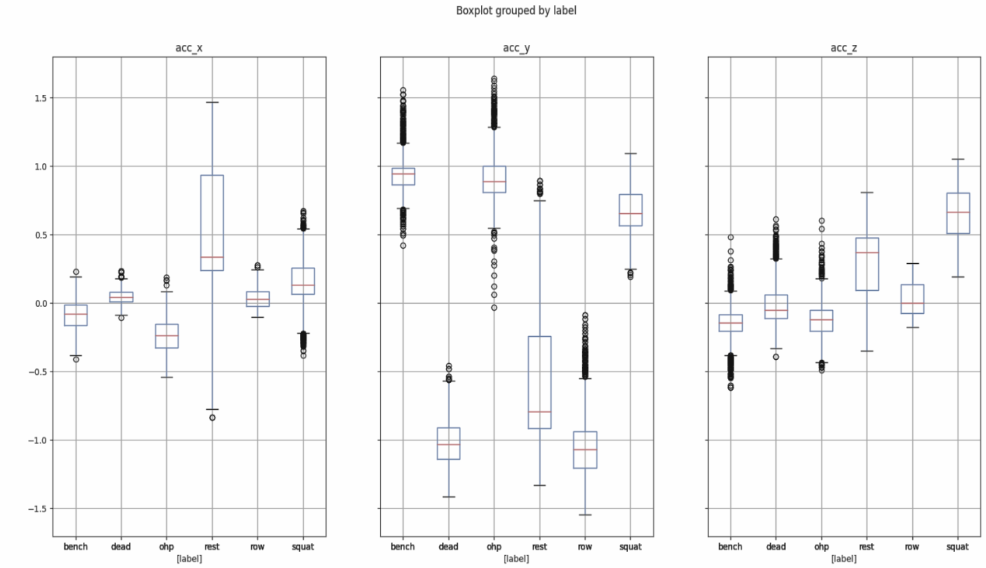
```
<Axes: title={'center': 'acc_x'}, xlabel='[label]'>
```



Snippet 3.11

```
#here
acc_df.boxplot(by="label",figsize=(20,10),layout=(1,3))
```

```
array([<Axes: title={'center': 'acc_x'}, xlabel='[label]'>,
       <Axes: title={'center': 'acc_y'}, xlabel='[label]'>,
       <Axes: title={'center': 'acc_z'}, xlabel='[label]'>], dtype=object)
```

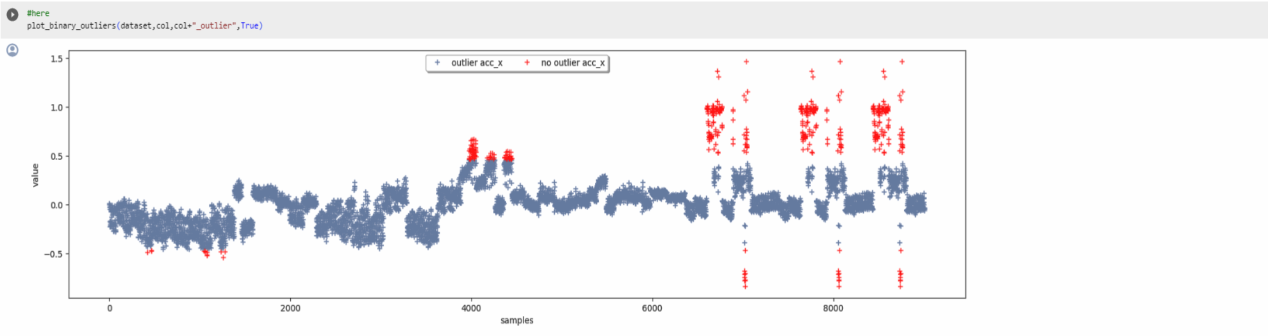


Snippet 3.12

The code snippet 3.12 shows a box plot for basic outlier detection for each of the x, y and z values of the acceleration for each exercise.

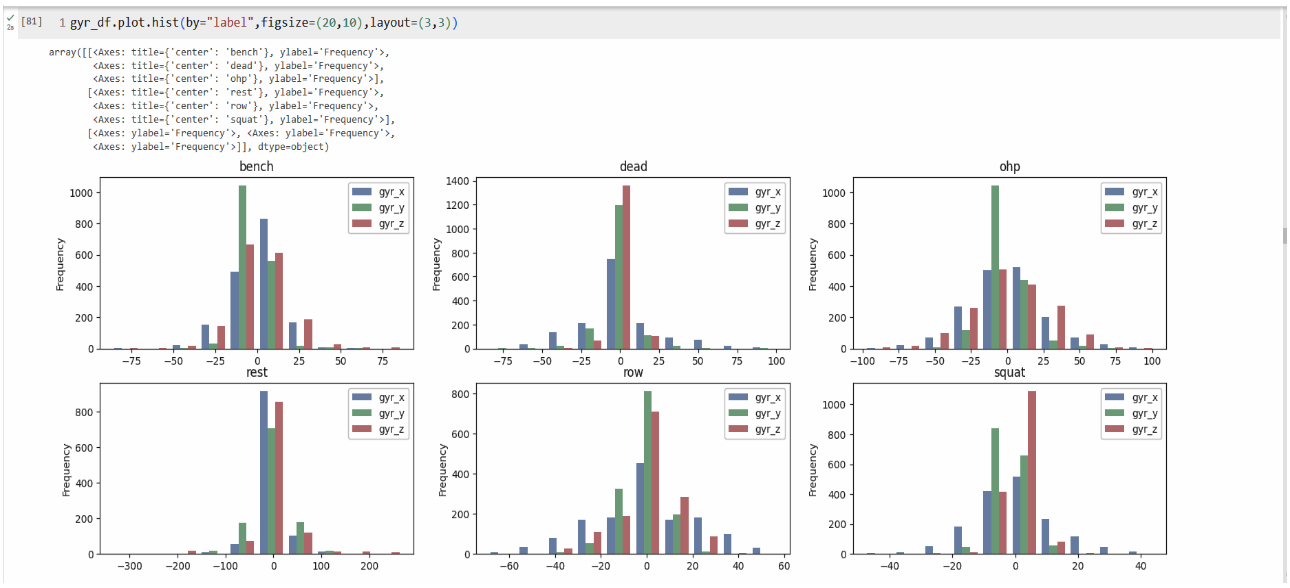
```
#here
def mark_outliers_iqr(dataset, col):
    dataset = dataset.copy()

    Q1 = dataset[col].quantile(0.25)
    Q3 = dataset[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    dataset[col + "_outlier"] = (dataset[col] < lower_bound) | (
        dataset[col] > upper_bound
    )

    return dataset
```

Snippet 3.13



Snippet 3.14

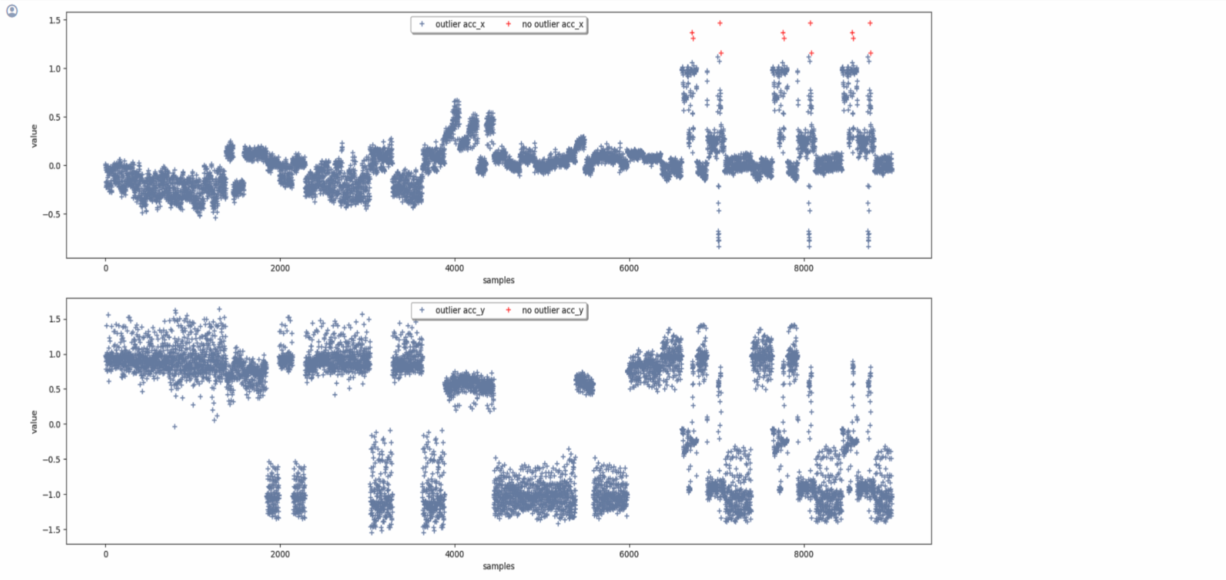
The code snippet 3.14 demonstrates a beautiful visualization of outliers detected for *acc_x* feature using the IQR method. The IQR method is a popular method to detect outliers. IQR is a distribution based method.



Snippet 3.15

To apply *Chauvenet's Criterion,* the data must be normally distributed. It is well clear from the visualization that data is normally distributed.

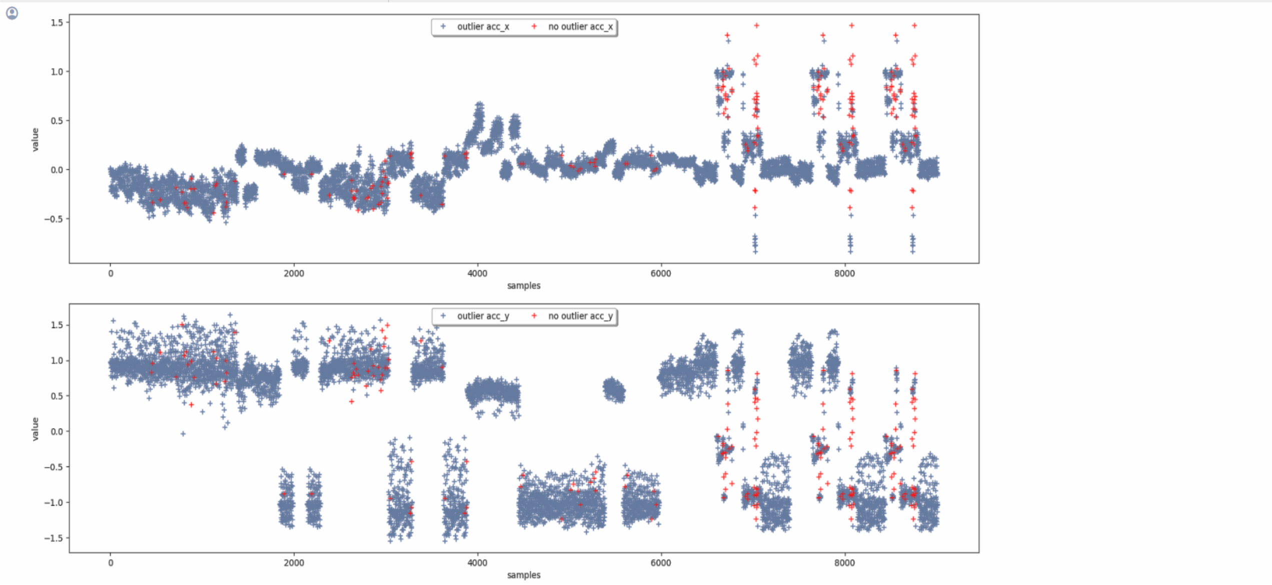
```
#here
for col in outlier_columns:
  dataset=mark_outliers_chauvenet(data_resampled,col)
  plot_binary_outliers(dataset,col,col+"_outlier",True)
```



Snippet 3.16

Outlier detection from C*hauvenet's Criterion* is shown in the snippet 3.16. C*hauvenet's Criterion* is also a distribution based method to detect outliers based on probability theory.

```
#here
dataset, outliers, X_scores=mark_outliers_lof(data_resampled,outlier_columns)
for col in outlier_columns:
  plot_binary_outliers(dataset,col,"outlier_lof",True)
```



Snippet 3.17

Finally, the outliers are detected using *Local Outlier Factor* which is distance based outlier detection method.

```python
#here
for col in outlier_columns:
  for label in data_resampled["label"].unique():
    dataset=mark_outliers_chauvenet(data_resampled[data_resampled["label"]==label],col)

    #Replace value with Nan marked as outlier
    dataset.loc[dataset[col+"_outlier"],col]=np.nan

    # Update the column in the original df
    outliers_removed_df.loc[(outliers_removed_df["label"]==label),col]=dataset[col]
    n_outliers=len(dataset)-len(dataset[col].dropna())
    print(f"Number of outliers for label:{label} and attribute:{col}={n_outliers}")
```

```
Number of outliers for label:bench and attribute:acc_x=0
Number of outliers for label:ohp and attribute:acc_x=2
Number of outliers for label:squat and attribute:acc_x=0
Number of outliers for label:dead and attribute:acc_x=2
Number of outliers for label:row and attribute:acc_x=0
Number of outliers for label:rest and attribute:acc_x=0
Number of outliers for label:bench and attribute:acc_y=5
Number of outliers for label:ohp and attribute:acc_y=6
Number of outliers for label:squat and attribute:acc_y=0
Number of outliers for label:dead and attribute:acc_y=0
Number of outliers for label:row and attribute:acc_y=0
Number of outliers for label:rest and attribute:acc_y=0
Number of outliers for label:bench and attribute:acc_z=3
Number of outliers for label:ohp and attribute:acc_z=6
Number of outliers for label:squat and attribute:acc_z=0
Number of outliers for label:dead and attribute:acc_z=1
Number of outliers for label:row and attribute:acc_z=0
Number of outliers for label:rest and attribute:acc_z=0
Number of outliers for label:bench and attribute:gyr_x=2
Number of outliers for label:ohp and attribute:gyr_x=4
Number of outliers for label:squat and attribute:gyr_x=1
Number of outliers for label:dead and attribute:gyr_x=6
Number of outliers for label:row and attribute:gyr_x=0
Number of outliers for label:rest and attribute:gyr_x=12
Number of outliers for label:bench and attribute:gyr_y=14
Number of outliers for label:ohp and attribute:gyr_y=15
Number of outliers for label:squat and attribute:gyr_y=9
Number of outliers for label:dead and attribute:gyr_y=14
Number of outliers for label:row and attribute:gyr_y=10
Number of outliers for label:rest and attribute:gyr_y=9
Number of outliers for label:bench and attribute:gyr_z=13
Number of outliers for label:ohp and attribute:gyr_z=1
Number of outliers for label:squat and attribute:gyr_z=12
Number of outliers for label:dead and attribute:gyr_z=14
Number of outliers for label:row and attribute:gyr_z=0
Number of outliers for label:rest and attribute:gyr_z=24
```

Snippet 3.18

At last, *Chauvenet's Criterion* is chosen and the entire dataset is run on it to mark outliers and given a value of null.

Stage 4: Feature Engineering

Part 4: Feature Engineering

Feature engineering is the process of transforming raw data into meaningful features that can be used for training machine learning models.

Snippet 3.19

```python
#here
#Building features
from DataTransformation import LowPassFilter,PrincipalComponentAnalysis
from TemporalAbstraction import NumericalAbstraction
```

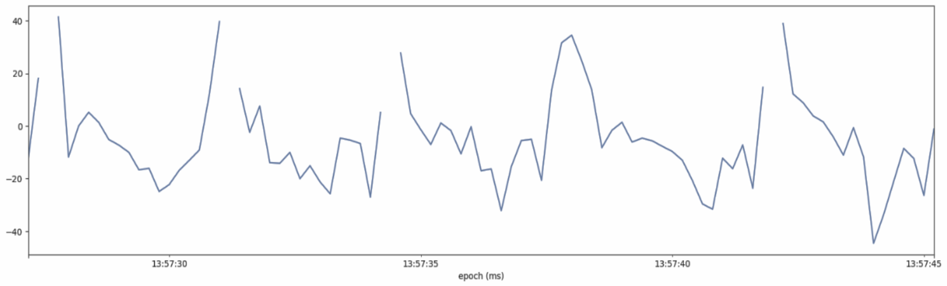
```python
df=outliers_removed_df.copy()
df
```

epoch (ms)	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	participant	label	category	set
2019-01-11 15:08:05.200	0.013500	0.977000	-0.071000	-1.8904	2.4392	0.9388	B	bench	heavy	76
2019-01-11 15:08:05.400	-0.001500	0.970500	-0.079500	-1.6826	-0.8904	2.1708	B	bench	heavy	76
2019-01-11 15:08:05.600	0.001333	0.971667	-0.064333	2.5608	-0.2560	-1.4146	B	bench	heavy	76
2019-01-11 15:08:05.800	-0.024000	0.957000	-0.073500	8.0610	-4.5244	-2.0730	B	bench	heavy	76
2019-01-11 15:08:06.000	-0.028000	0.957667	-0.115000	2.4390	-1.5486	-3.6098	B	bench	heavy	76
...	...	...	...	...	...	...	...	...	...	...
2019-01-20 17:33:27.000	-0.048000	-1.041500	-0.076500	1.4146	-5.6218	0.2926	E	row	medium	15
2019-01-20 17:33:27.200	-0.037000	-1.030333	-0.053333	-2.7684	-0.5854	2.2440	E	row	medium	15
2019-01-20 17:33:27.400	-0.060000	-1.031000	-0.082000	2.8416	-5.1342	-0.1220	E	row	medium	15
2019-01-20 17:33:27.600	-0.038667	-1.025667	-0.044667	-0.2318	0.2562	1.1220	E	row	medium	15
2019-01-20 17:33:27.800	-0.044000	-1.034000	-0.059000	1.0980	-4.0240	0.9760	E	row	medium	15

9009 rows × 10 columns

Snippet 3.20

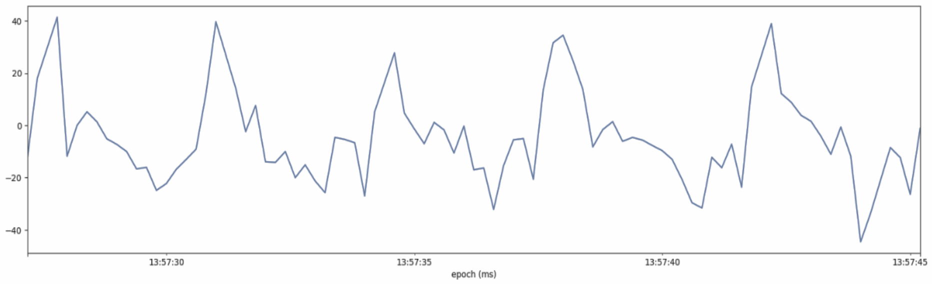
```
#here
#Step 1: Dealing with missing values (Imputation)
subset=df[df["set"]==12]["gyr_y"].plot()
#We can see a lot of missing values, how do we deal with them?
```



Snippet 3.21

```
#here
#We can interpolate the data
for col in predictor_columns:
  df[col]=df[col].interpolate()
```

```
subset=df[df["set"]==12]["gyr_y"].plot()
```



Snippet 3.22

The code snippets 3.22 show how the outliers having a value of null are *interpolated* as a part of feature engineering.

```
#here
df_lowpass=LowPass.low_pass_filter(
    df_lowpass,
    "acc_y",
    sampling_freq,
    cutoff
)
```

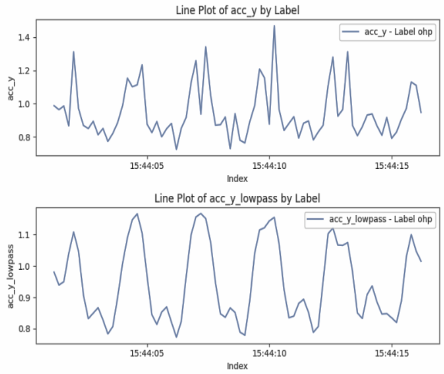
```
df_lowpass
```

epoch (ms)	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	participant	label	category	set	duration	acc_y_lowpass
2019-01-11 15:08:05.200	0.013500	0.977000	-0.071000	-1.8904	2.4392	0.9388	B	bench	heavy	76	16.0	0.977001
2019-01-11 15:08:05.400	-0.001500	0.970500	-0.079500	-1.6826	-0.8904	2.1708	B	bench	heavy	76	16.0	0.970257
2019-01-11 15:08:05.600	0.001333	0.971667	-0.064333	2.5608	-0.2560	-1.4146	B	bench	heavy	76	16.0	0.963589
2019-01-11 15:08:05.800	-0.024000	0.957000	-0.073500	8.0610	-4.5244	-2.0730	B	bench	heavy	76	16.0	0.965441
2019-01-11 15:08:06.000	-0.028000	0.957667	-0.115000	2.4390	-1.5486	-3.6098	B	bench	heavy	76	16.0	0.966784
...	...	...	...	...	...	...	...	...	...	...	...	...
2019-01-20 17:33:27.000	-0.048000	-1.041500	-0.076500	1.4146	-5.6218	0.2926	E	row	medium	15	19.0	-0.974791
2019-01-20 17:33:27.200	-0.037000	-1.030333	-0.053333	-2.7684	-0.5854	2.2440	E	row	medium	15	19.0	-1.020916
2019-01-20 17:33:27.400	-0.060000	-1.031000	-0.082000	2.8416	-5.1342	-0.1220	E	row	medium	15	19.0	-1.051656
2019-01-20 17:33:27.600	-0.038667	-1.025667	-0.044667	-0.2318	0.2562	1.1220	E	row	medium	15	19.0	-1.040440
2019-01-20 17:33:27.800	-0.044000	-1.034000	-0.059000	1.0980	-4.0240	0.9760	E	row	medium	15	19.0	-1.033252

9009 rows × 12 columns

Snippet 3.23

```python
#here
# Create a 2x1 subplot
fig, axes = plt.subplots(2, 1, figsize=(8, 6))

# Reset the index of the subset dataset
subset.reset_index(drop=True, inplace=False)

# Create line plots for "acc_y" and "acc_y_lowpass" with legends
for i, column in enumerate(["acc_y", "acc_y_lowpass"]):
    ax = axes[i]
    for label, group in subset.groupby("label"):
        ax.plot(group.index, group[column], label=f"{column} - Label {label}")

    ax.set_title(f"Line Plot of {column} by Label")
    ax.set_xlabel("Index")
    ax.set_ylabel(column)
    ax.legend(loc='upper right')

plt.tight_layout()
plt.show()
```



Snippet 3.24

The second part of feature engineering was to apply a *Butterworth lowpass filter*. The main goal for doing so is to smoothen the curve for better pattern detection by the models during training. It filters out regions of high frequency for smoothing.

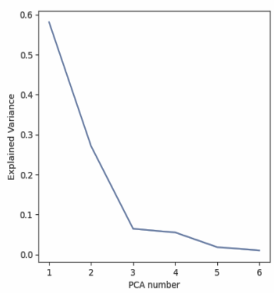
```python
#here
#Step 3: PCA
df_pca=df_lowpass.copy()
PCA=PrincipalComponentAnalysis()
```

```python
pc_values=PCA.determine_pc_explained_variance(df_pca,predictor_columns)
pc_values
```

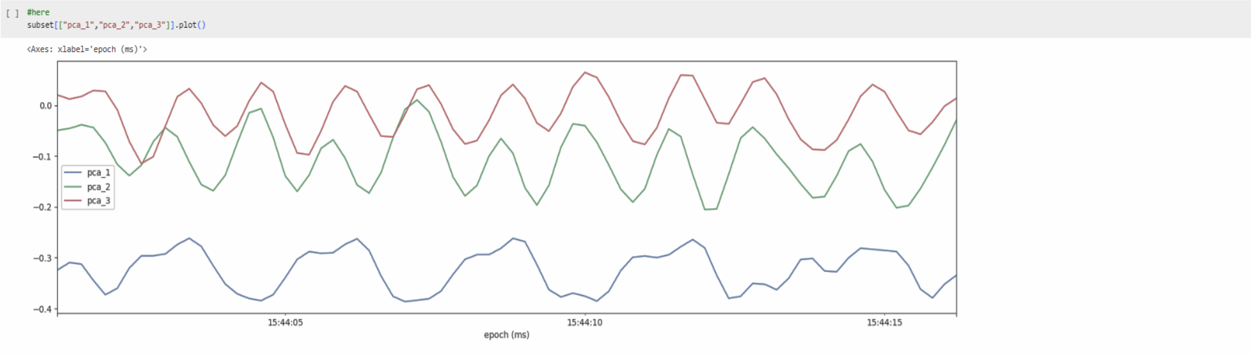
```
array([0.58094932, 0.27090665, 0.06427472, 0.05519685, 0.01836808,
       0.01030437])
```

Snippet 3.25

```python
#here
#Elbow Technique to determine the optimal number of PCs
plt.figure(figsize=(5,5))
plt.plot(range(1,len(predictor_columns)+1),pc_values)
plt.xlabel("PCA number")
plt.ylabel("Explained Variance")
plt.show()
```

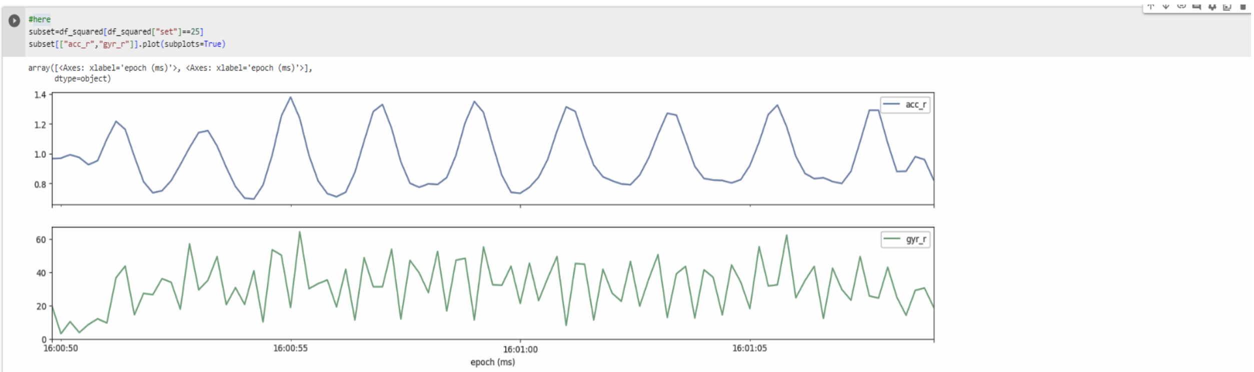


Snippet 3.26



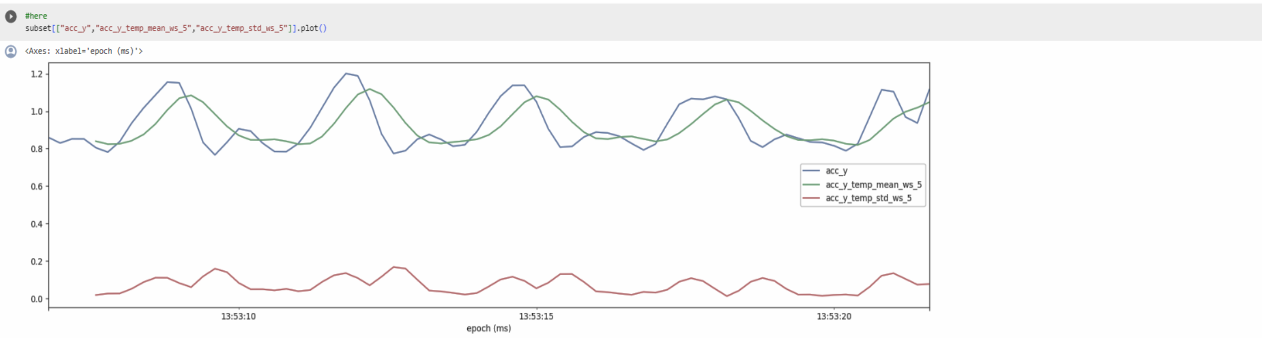
Snippet 3.27

The code snippets 3.24, 3.25 and 3.26 show the working for *Principal Component Analysis (PCA)*. New features are engineered from the basic six features. Through the *elbow method,* we came to the conclusion that *three* principal features should be extracted from the basic features.



Snippet 3.28

The next part of feature engineering is to engineer the *sum of squared features* to minimize the bias towards device's orientation.



Snippet 3.29

The fifth part of feature engineering was to engineer temporal features. Temporal features give a clear idea of the data is changing throughout the time. *Rolling mean and rolling standard deviation* were used to extract temporal features of each corresponding attribute.

```python
#here
df_freq_list=[]
for s in df_freq["set"].unique():
  print(f"Applying Fourier Transformation to set {s}")
  subset=df_freq[df_freq["set"]==s].reset_index(drop=True).copy()
  subset=FreqAbs.abstract_frequency(subset,predictor_columns,ws,fs)
  df_freq_list.append(subset)
```

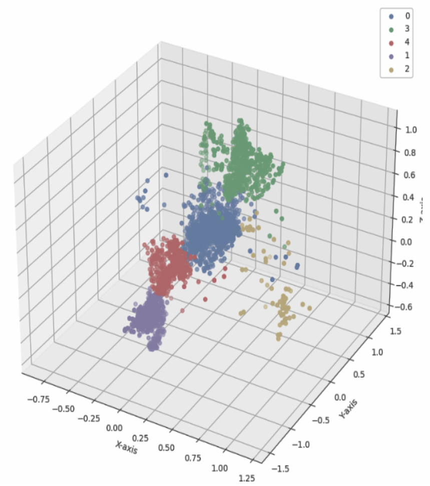
```
Applying Fourier Transformation to set 76
Applying Fourier Transformation to set 34
Applying Fourier Transformation to set 26
Applying Fourier Transformation to set 73
Applying Fourier Transformation to set 17
Applying Fourier Transformation to set 3
Applying Fourier Transformation to set 88
Applying Fourier Transformation to set 68
Applying Fourier Transformation to set 30
Applying Fourier Transformation to set 48
Applying Fourier Transformation to set 18
Applying Fourier Transformation to set 9
Applying Fourier Transformation to set 60
Applying Fourier Transformation to set 59
Applying Fourier Transformation to set 63
Applying Fourier Transformation to set 25
Applying Fourier Transformation to set 41
Applying Fourier Transformation to set 39
Applying Fourier Transformation to set 40
Applying Fourier Transformation to set 42
Applying Fourier Transformation to set 23
Applying Fourier Transformation to set 47
Applying Fourier Transformation to set 78
Applying Fourier Transformation to set 45
Applying Fourier Transformation to set 70
Applying Fourier Transformation to set 77
Applying Fourier Transformation to set 79
Applying Fourier Transformation to set 24
Applying Fourier Transformation to set 93
Applying Fourier Transformation to set 69
Applying Fourier Transformation to set 35
Applying Fourier Transformation to set 85
Applying Fourier Transformation to set 27
Applying Fourier Transformation to set 12
Applying Fourier Transformation to set 33
Applying Fourier Transformation to set 54
Applying Fourier Transformation to set 62
Applying Fourier Transformation to set 61
Applying Fourier Transformation to set 19
Applying Fourier Transformation to set 81
Applying Fourier Transformation to set 1
Applying Fourier Transformation to set 36
Applying Fourier Transformation to set 22
Applying Fourier Transformation to set 38
Applying Fourier Transformation to set 13
Applying Fourier Transformation to set 10
Applying Fourier Transformation to set 52
Applying Fourier Transformation to set 82
Applying Fourier Transformation to set 90
Applying Fourier Transformation to set 71
```

Snippet 3.30

The next features to be engineered were *frequency* features. *Fast Fourier Transform* was used to engineer the frequency features.

```python
#Lets plot the clusters
fig=plt.figure(figsize=(10,10))
ax=fig.add_subplot(projection="3d")
for c in df_cluster["cluster"].unique():
  subset=df_cluster[df_cluster["cluster"]==c]
  ax.scatter(subset["acc_x"],subset["acc_y"],subset["acc_z"],label=c)

ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b096b2bde70>
```



Snippet 3.31

Feature engineering is concluded by clustering using the *K-Means algorithm.* At the end of feature engineering, a total of *116 features* were present in the dataframe.

Stage 5: Predictive Modeling

For predictive modeling, python provides a comprehensive library for model training and evaluation. The name of the library is *scikit-learn* [26]. It has a comprehensive list of models for all type of problems be it classification, regression or clustering.

```
Part 5: Predictive Modeling
```

Snippet 3.32

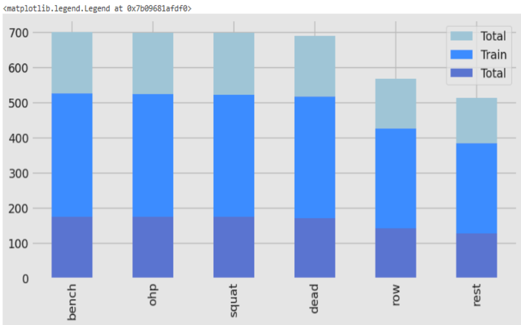
```
#here
df_train=df.drop(["participant","category","set","duration"],axis=1) #drop the columns mentioned in the list
```

```
X=df_train.drop("label",axis=1)
y=df_train["label"]
```

Snippet 3.33

```
#here

#Lets plot it
fig,ax=plt.subplots(figsize=(10,5))
df_train["label"].value_counts().plot(kind="bar",ax=ax,color="lightblue",label="Total")
y_train.value_counts().plot(kind="bar",ax=ax,color="dodgerblue",label="Train")
y_test.value_counts().plot(kind="bar",ax=ax,color="royalblue",label="Total")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b09681afdf0>
```



Snippet 3.34

The code snippet 3.33 shows the splitting of data frame created in the last phase into training and testing in a 75% to 25% ratio. The splitting for each label is also visualized.

```
#here
#Step 2: Split features
basic_features=["acc_x","acc_y","acc_z","gyr_x","gyr_y","gyr_z"]
square_features=["acc_r","gyr_r"]
pca_features=["pca_1","pca_2","pca_3"]
time_features=[f for f in df_train.columns if "_temp_" in f]
freq_features=[f for f in df_train.columns if ("_freq" in f) or ("_pse" in f)]
cluster_features=["cluster"]
```

Snippet 3.35

```
#here
print("Basic Features:",len(basic_features))
print("Square Features:",len(square_features))
print("PCA Features:",len(pca_features))
print("Time Features:",len(time_features))
print("Frquency Features:",len(freq_features))
print("Cluster Features:",len(cluster_features))
```

```
Basic Features: 6
Square Features: 2
PCA Features: 3
Time Features: 16
Frquency Features: 88
Cluster Features: 1
```

Snippet 3.36

The snippet 3.34 and 3.35 shows the splitting of features into various sets like basic, temporal, frequency and more.

```
for i, f in zip(range(len(possible_feature_sets)), feature_names):
    print("Feature set:", i)
    selected_train_X = X_train[possible_feature_sets[i]]
    selected_test_X = X_test[possible_feature_sets[i]]

    # First run non deterministic classifiers to average their score.
    performance_test_nn_acc = 0
    performance_test_nn_pre = 0
    performance_test_nn_rec = 0
    performance_test_nn_f1 = 0
    performance_test_rf_acc = 0
    performance_test_rf_pre = 0
    performance_test_rf_rec = 0
    performance_test_rf_f1 = 0

    for it in range(0, iterations):
        print("\tTraining neural network,", it)
        (
            class_train_y,
            class_test_y,
            class_train_prob_y,
            class_test_prob_y,
        ) = learner.feedforward_neural_network(
            selected_train_X,
            y_train,
            selected_test_X,
            gridsearch=False,
        )
        performance_test_nn_acc += accuracy_score(y_test, class_test_y)
        performance_test_nn_pre += precision_score(y_test, class_test_y,average='weighted')
        performance_test_nn_rec += recall_score(y_test, class_test_y, average='weighted')
        performance_test_nn_f1 += f1_score(y_test, class_test_y, average='weighted')

        print("\tTraining random forest,", it)
        (
            class_train_y,
            class_test_y,
            class_train_prob_y,
            class_test_prob_y,
            rf
        ) = learner.random_forest(
            selected_train_X, y_train, selected_test_X, gridsearch=True
        )
        performance_test_rf_acc += accuracy_score(y_test, class_test_y)
```

Snippet 3.37

```
Feature set: 0
        Training neural network, 0
        Training random forest, 0
        Training KNN
        Training decision tree
        Training naive bayes
Feature set: 1
        Training neural network, 0
        Training random forest, 0
        Training KNN
        Training decision tree
        Training naive bayes
Feature set: 2
        Training neural network, 0
        Training random forest, 0
        Training KNN
        Training decision tree
        Training naive bayes
Feature set: 3
        Training neural network, 0
        Training random forest, 0
        Training KNN
        Training decision tree
        Training naive bayes
Feature set: 4
        Training neural network, 0
        Training random forest, 0
        Training KNN
        Training decision tree
        Training naive bayes
```

Snippet 3.38

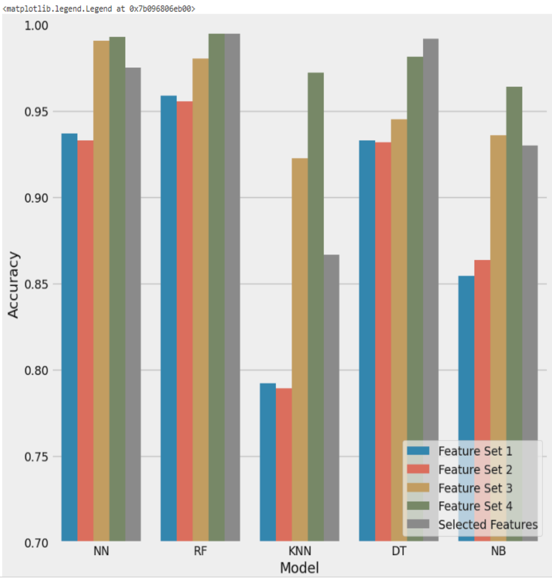
The code snippet 3.37 shows the training of various models. A s*imple neural network, decision tree, random free, kNN and naive bayes* were used. Each model was trained on different feature sets created previously.

```
[ ] #here
    score_df.sort_values(by="accuracy",ascending=False)
```

	model	feature_set	accuracy	precision	recall	f1-score
1	RF	Feature Set 4	0.994829	0.994856	0.994829	0.994827
1	RF	Selected Features	0.994829	0.994917	0.994829	0.994832
0	NN	Feature Set 4	0.992761	0.992793	0.992761	0.992762
3	DT	Selected Features	0.991727	0.991847	0.991727	0.991731
0	NN	Feature Set 3	0.990693	0.990780	0.990693	0.990702
3	DT	Feature Set 4	0.981386	0.981821	0.981386	0.981421
1	RF	Feature Set 3	0.980352	0.980357	0.980352	0.980351
0	NN	Selected Features	0.975181	0.975522	0.975181	0.975175
2	KNN	Feature Set 4	0.972079	0.972290	0.972079	0.972074
4	NB	Feature Set 4	0.963806	0.963896	0.963806	0.963757
1	RF	Feature Set 1	0.958635	0.959067	0.958635	0.958598
1	RF	Feature Set 2	0.955533	0.955621	0.955533	0.955545
3	DT	Feature Set 3	0.945191	0.945270	0.945191	0.945056
0	NN	Feature Set 1	0.936918	0.937306	0.936918	0.937045
4	NB	Feature Set 3	0.935884	0.935813	0.935884	0.935725
0	NN	Feature Set 2	0.932782	0.934278	0.932782	0.932750
3	DT	Feature Set 1	0.932782	0.935343	0.932782	0.932754
3	DT	Feature Set 2	0.931748	0.933351	0.931748	0.931326
4	NB	Selected Features	0.929679	0.930951	0.929679	0.929082
2	KNN	Feature Set 3	0.922441	0.922881	0.922441	0.922481
2	KNN	Selected Features	0.866598	0.869051	0.866598	0.867154
4	NB	Feature Set 2	0.863495	0.865488	0.863495	0.862685
4	NB	Feature Set 1	0.854188	0.858232	0.854188	0.851393
2	KNN	Feature Set 1	0.792141	0.792366	0.792141	0.791443
2	KNN	Feature Set 2	0.789038	0.788309	0.789038	0.787923

```
plt.figure(figsize=(10,10))
sns.barplot(x="model",y="accuracy",hue="feature_set",data=score_df)
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.ylim(0.7,1)
plt.legend(loc="lower right")
```

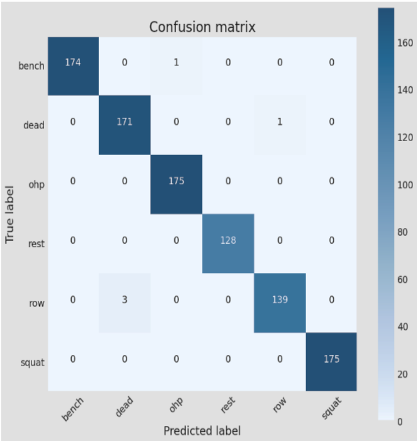
```
<matplotlib.legend.Legend at 0x7b096806eb00>
```



Snippet 3.39

The dataframe and the visualization shows all of the evaluation metrics in an easy to comprehend way. We can see that the neural network and random forest performed almost identically. Feature set 4 performs the best for each of the models except the decision tree, where overfitting is happening.

```python
plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Confusion matrix")
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

thresh = cm.max() / 2.0
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(
        j,
        i,
        format(cm[i, j]),
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black",
    )
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.grid(False)
plt.show()
```



Snippet 3.40

This stage ends with the visualization of a confusion matrix for the random forest classifier.

Stage 6: Repetition Count

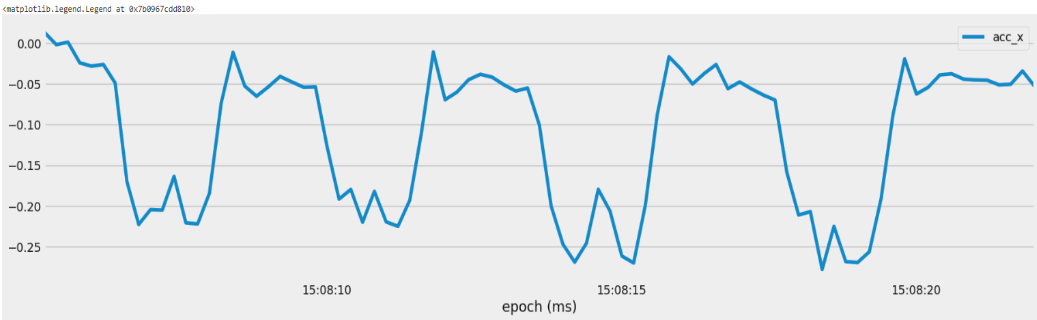
Phase 6: Repetition Counting

```python
#here
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from DataTransformation import LowPassFilter
from scipy.signal import argrelextrema
from sklearn.metrics import mean_absolute_error

pd.options.mode.chained_assignment = None


# Plot settings
plt.style.use("fivethirtyeight")
plt.rcParams["figure.figsize"] = (20, 5)
plt.rcParams["figure.dpi"] = 100
```

```python
#here
#Step 2: Split data for each exercise
bench_df=df[df["label"]=="bench"]
squat_df=df[df["label"]=="squat"]
row_df=df[df["label"]=="row"]
ohp_df=df[df["label"]=="ohp"]
dead_df=df[df["label"]=="dead"]
```

```
#here
#Step 3: Visualize data to identify patterns
plot_df=bench_df
plot_df[plot_df["set"]==plot_df["set"].unique()[0]]["acc_x"].plot()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b0967cdd810>
```



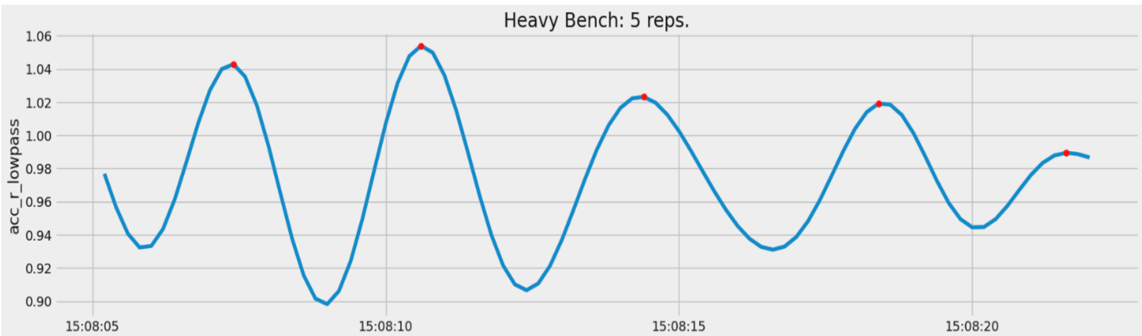
Snippet 3.41

The code snippets 3.40 show the pattern of *acc_x* over a set from the bench press. We can infer from the visualization that there are roughly 4-5 repeating patterns.

```
#here
def count_reps(dataset,cutoff=0.4,order=10,column="acc_r"):
  data=LowPass.low_pass_filter(dataset,col=column,sampling_frequency=fs,cutoff_frequency=cutoff,order=order)
  indexes=argrelextrema(data[column+"_lowpass"].values,np.greater)
  peaks=data.iloc[indexes]

  fig,ax=plt.subplots()
  plt.plot(dataset[f"{column}_lowpass"])
  plt.plot(peaks[f"{column}_lowpass"],"o",color="red")
  ax.set_ylabel(f"{column}_lowpass")
  exercise=dataset["label"].iloc[0].title()
  category=dataset["category"].iloc[0].title()
  plt.title(f"{category} {exercise}: {len(peaks)} reps.")
  plt.show()
  plt.legend()
  return len(peaks)
```

Snippet 3.42

```
count_reps(bench_set,cutoff=0.4)
count_reps(squat_set,cutoff=0.35)
count_reps(row_set,cutoff=0.65,column="gyr_x")
count_reps(ohp_set,cutoff=0.35)
count_reps(dead_set,cutoff=0.4)
```



Snippet 3.43

The code snippet 3.42 is of the custom repetition count algorithm. The basic idea for the algorithm was to smooth the curve of the most dominant feature and then count the number of maximas in it which would indicate the number of repetitions.

```
[ ] #here
rep_df
```

	label	category	set	reps	reps_pred
0	bench	heavy	1	5	4
1	bench	heavy	24	5	5
2	bench	heavy	26	5	5
3	bench	heavy	34	5	5
4	bench	heavy	47	5	5
...	...	...	...	...	...
80	squat	medium	39	10	7
81	squat	medium	40	10	8
82	squat	medium	41	10	5
83	squat	medium	42	10	8
84	squat	medium	50	10	9

85 rows × 5 columns

Snippet 3.44

```
#here
#Evaluate the results

error=mean_absolute_error(rep_df["reps"],rep_df["reps_pred"]).round(2)
error

#On an average for each set we have an error of 1 rep only
```

```
0.88
```

Snippet 3.45

```
#Stacking!
#Stack-1

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble  import StackingClassifier

stacked_model1 = StackingClassifier(
    estimators=[('dt',dt), ('lr', lr), ('svm',svm),('knn', knn), ('nb', nb)],
    final_estimator=lr,
    cv=5  # You can adjust the number of folds for cross-validation
)
stacked_model1.fit(X_train[feature_set_3], y_train.values.ravel())
stacked_predictions = stacked_model1.predict(X_test[feature_set_3])
accuracy_stacked = accuracy_score(y_test, stacked_predictions)
precision_stacked = precision_score(y_test, stacked_predictions,average='weighted')
recall_stacked = recall_score(y_test, stacked_predictions,average='weighted')
f1_stacked = f1_score(y_test, stacked_predictions,average='weighted')
print("Accuracy of Stacked Model:", accuracy_stacked)
print("Precision of Stacked Model:", precision_stacked)
print("Recall of Stacked Model:", recall_stacked)
print("f1 of Stacked Model:", f1_stacked)
```

Snippet 3.46

Code snippet 3.46 defines the first stacked model calling is stacked_model1, trains it and then evaluates it based on the test data.

```
#Stack-2

stacked_model2 = StackingClassifier(
    estimators=[('rf',rf), ('lr', lr), ('dt',dt)],
    final_estimator=rf,
    cv=5  # You can adjust the number of folds for cross-validation
)
stacked_model2.fit(X_train[feature_set_3], y_train.values.ravel())
stacked_predictions = stacked_model2.predict(X_test[feature_set_3])
accuracy_stacked = accuracy_score(y_test, stacked_predictions)
precision_stacked = precision_score(y_test, stacked_predictions,average='weighted')
recall_stacked = recall_score(y_test, stacked_predictions,average='weighted')
f1_stacked = f1_score(y_test, stacked_predictions,average='weighted')
print("Accuracy of Stacked Model:", accuracy_stacked)
print("Precision of Stacked Model:", precision_stacked)
print("Recall of Stacked Model:", recall_stacked)
print("f1 of Stacked Model:", f1_stacked)
```

Snippet 3.47

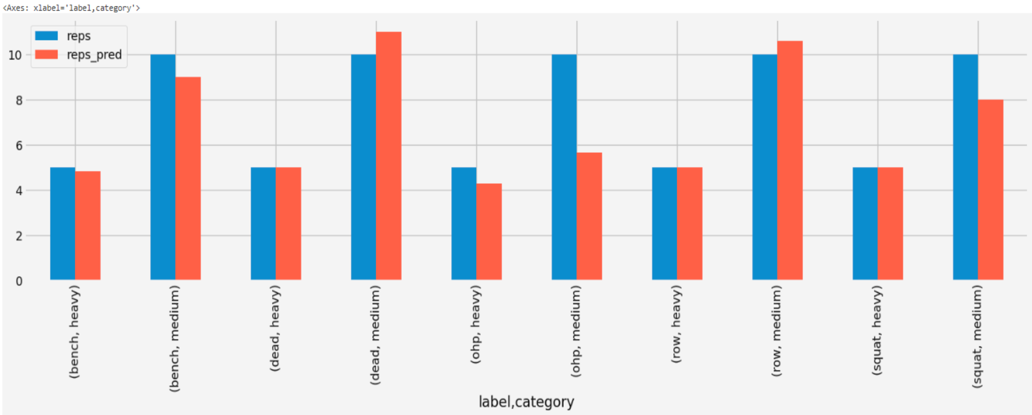
Code snippet 3.47 defines the second stacked model calling is stacked_mode2, trains it and then evaluates it based on the test data.

```
#Stack-3
stacked_model3 = StackingClassifier(
    estimators=[('rf',rf),('dt',dt), ('lr', lr), ('svm',svm),('knn', knn), ('nb', nb)],
    final_estimator=lr,
    cv=5  # You can adjust the number of folds for cross-validation
)
stacked_model3.fit(X_train[feature_set_3], y_train.values.ravel())
stacked_predictions = stacked_model3.predict(X_test[feature_set_3])
accuracy_stacked = accuracy_score(y_test, stacked_predictions)
precision_stacked = precision_score(y_test, stacked_predictions,average='weighted')
recall_stacked = recall_score(y_test, stacked_predictions,average='weighted')
f1_stacked = f1_score(y_test, stacked_predictions,average='weighted')
print("Accuracy of Stacked Model:", accuracy_stacked)
print("Precision of Stacked Model:", precision_stacked)
print("Recall of Stacked Model:", recall_stacked)
print("f1 of Stacked Model:", f1_stacked)
```

Snippet 3.48

Code snippet 3.48 defines the third stacked model calling is stacked_mode3, trains it and then evaluates it based on the test data.

```
#here
rep_df.groupby(["label","category"])["reps","reps_pred"].mean().plot.bar()

<ipython-input-235-c68c5cd0be8d>:1: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.

<Axes: xlabel='label,category'>
```



Snippet 3.49

This implementation part ends with the evaluation of performance of the custom repetition count algorithm. A benchmark dataset was created from knowing the fact that heavy sets had 5 repetitions and medium sets had 10 repetitions. *Mean absolute error* was calculated as the evaluation metric and it was just *0.88*.

Type: For tracking through camera.

Stage 1: Module imports and basic detection

The most important library that has been used for object detection using cameras is MediaPipe. MediaPipe allows us to mark body-landmarks which then in turn can be used to detect poses of various kinds including exercises.

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
import tensorflow as tf
import math

from sklearn.model_selection import train_test_split
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score, classification_report
from tensorflow.keras.utils import to_categorical

import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import (LSTM, Dense, Concatenate, Attention, Dropout, Softmax,
                                     Input, Flatten, Activation, Bidirectional, Permute, multiply,
```

Snippet 3.50

```
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results
```
✓ 0.0s

Snippet 3.51

Code Snippet 3.51 explains a function takes in an image and annotates it using the model provided.

```python
def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )
```

✓ 0.0s

Snippet 3.52

Code snippet 3.52 is a function that takes in an annotated image and its results and draws body-landmarks on it using MediaPipe.

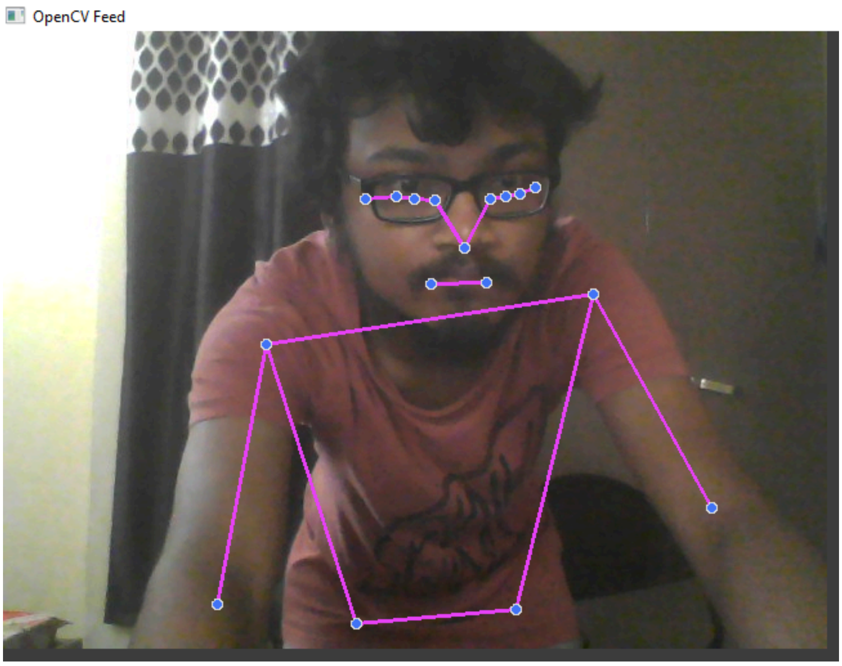


Figure 3.11

Figure 3.11 shows the body landmark identified by MediaPipe for the particular image

```python
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).
    flatten() if results.pose_landmarks else np.zeros(33*4)
    return pose
```

Snippet 3.53

```python
# Recollect and organize keypoints from the test
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
```

```python
# There are a total of 33 landmarks with 4 values (x,y,z,visibility)
num_landmarks = len(landmarks)
num_values = len(test)
num_input_values = num_landmarks*num_values
```

Snippet 3.54

Code snippet 3.53 and 3.54 extracts the landmark which would be fed into the model for classification.

Stage 2: Data Collection for classification

Data collection is a necessary step before model training. Good quality data always increases the chances of better model evaluation.

```python
exercises = np.array(['rest','curl', 'press', 'squat'])
num_classes = len(exercises)

num_videos = 50

sequence_length = FPS*1

start_folder = 1
```

Snippet 3.55

Code Snippet 3.55 defines the type of exercises and number of videos to be collected for each of the exercises.

```python
# Collect Training Data

cam = cv2.VideoCapture(0)
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    for idx, action in enumerate(exercises):
        for sequence in range(start_folder, start_folder+num_videos):
            for frame_num in range(sequence_length):
                ret, frame = cam.read()
                image, results = mediapipe_detection(frame, pose)
                try:
                    landmarks = results.pose_landmarks.landmark
                except:
                    pass

                draw_landmarks(image, results)
```

```python

                # Export keypoints (sequence + pose landmarks)
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
```

✓ 47.3s

Snippet 3.56

Code Snippet 3.56 defines the data collection process through *OpenCV* library. The *OpenCV* library enables the webcam and reads the video in frames. These frames are then annotated using the *MediaPipe* library and then saved to the disk.

Stage 3: Data Preprocessing and labeling

Data Preprocessing and labeling is an important step before model building, we are using supervised machine learning we must label our data as per the needs

```
label_map = {label:num for num, label in enumerate(exercises)}
```

```
sequences, labels = [], []
for action in exercises:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        window = []
        for frame_num in range(sequence_length):
            # LSTM input data
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)

        sequences.append(window)
        labels.append(label_map[action])
```

Snippet 3.57

Code snippet 3.57, for each of the exercises reads the data collected from its folder and then labels each frame with the corresponding label type. As a result of which, each frame of the data gets labeled.

```
# Split into training, validation, and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=1)
print(X_train.shape, y_train.shape)

```

Snippet 3.58

Code snippet 3.58, splits the data into training and testing dataset. A split ratio of 9:1 has been taken meaning 90% of the data is gone to training whereas 10% of the total data is considered for testing.

Stage 4: Model Building

After the data preprocessing and labeling, model building is the next step. Model building or training is the step where the model is actually learning from the labeled data so that it can predict on unseen data as well.

```
# SVM model
svm = SVC(kernel='linear', probability=True)

# Fit SVM model to the training data
svm.fit(X_train.reshape(X_train.shape[0], -1), y_train)

# Evaluate the SVM model
```

Snippet 3.59

Code snippet 3.59 depicts that SVC class is instantiated, the instance aka our model is train/fit on the labeled training data where it starts learning.

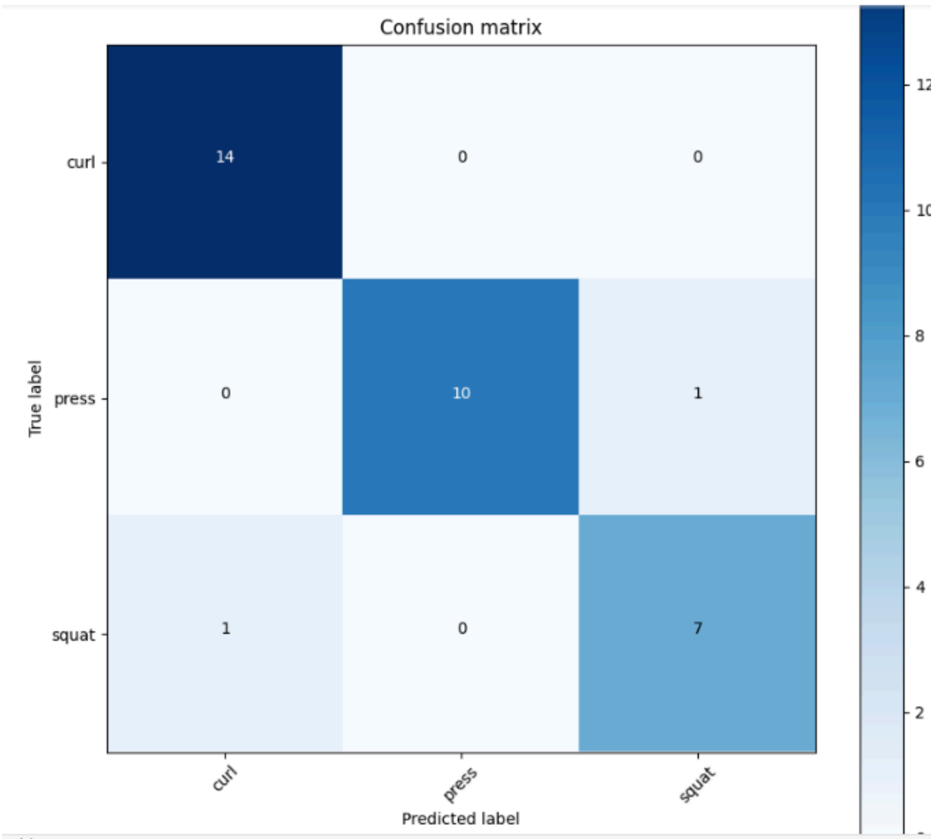
```
import joblib

# Assuming 'svc_model' is your SVC model
save_dir = os.path.join(os.getcwd(), "svm.joblib")
joblib.dump(svm, save_dir)
```

Snippet 3.60

Code snippet 3.60 saves the trained model's weight into the same directory and name "svm.joblib" for future purposes.

Stage 3: Model Evaluation

```python
import itertools
yhat = svm.predict(X_test_reshaped)
unique_labels = np.unique(y_test)
cm = confusion_matrix(y_test, yhat, labels=unique_labels)

plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Confusion matrix")

plt.colorbar()
tick_marks = np.arange(len(unique_labels))
plt.xticks(tick_marks, exercises, rotation=45)
plt.yticks(tick_marks, exercises)

thresh = cm.max() / 2.0
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(
        j,
        i,
        format(cm[i, j]),
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black",
    )
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.grid(False)
plt.show()
```



Snippet 3.61

Code snippet 3.61 shows how to create the confusion matrix using the confusion_matrix function by sk-learn.

```python
for model_name, model in models.items():
    yhat = model.predict(X_test_reshaped)

    # Model accuracy
    classification_accuracies[model_name] = accuracy_score(y_test, yhat)
    print(f"{model_name} classification accuracy = {round(classification_accuracies[model_name]*100,3)}%")

# Collect results
eval_results['accuracy'] = classification_accuracies
```

```
SVM classification accuracy = 94.118%
```

Snippet 3.62

Code snippet 3.62 checks the model accuracy by evaluating the unseen data. SVM achieved an accuracy of 94.118%

```python
for model_name, model in models.items():
    yhat = model.predict(X_test_reshaped)


    # Precision, recall, and f1 score
    report = classification_report(y_test, yhat, target_names=exercises, output_dict=True)

    precisions[model_name] = report['weighted avg']['precision']
    recalls[model_name] = report['weighted avg']['recall']
    f1_scores[model_name] = report['weighted avg']['f1-score']

    print(f"{model_name} weighted average precision = {round(precisions[model_name],3)}")
    print(f"{model_name} weighted average recall = {round(recalls[model_name],3)}")
    print(f"{model_name} weighted average f1-score = {round(f1_scores[model_name],3)}\n")

# Collect results
eval_results['precision'] = precisions
eval_results['recall'] = recalls
eval_results['f1 score'] = f1_scores
```

```
SVM weighted average precision = 0.95
SVM weighted average recall = 0.941
SVM weighted average f1-score = 0.941
```

Snippet 3.63

Code snippet 3.63 calculates the weighted average precision, recall and f1-score. SVM achieved a weighted average precision of 95.0%, recall of 94.1% and f1-score of 94.1% as well.

```python
def count_reps(image, current_action, landmarks, mp_pose):
    global curl_counter, press_counter, squat_counter, curl_stage, press_stage, squat_stage

    if current_action == 'curl':
        # Get coords
        shoulder = get_coordinates(landmarks, mp_pose, 'left', 'shoulder')
        elbow = get_coordinates(landmarks, mp_pose, 'left', 'elbow')
        wrist = get_coordinates(landmarks, mp_pose, 'left', 'wrist')

        # calculate elbow angle
        angle = calculate_angle(shoulder, elbow, wrist)

        # curl counter logic
        if angle < 30:
            curl_stage = "up"
        if angle > 140 and curl_stage =='up':
            curl_stage="down"
            curl_counter +=1
        press_stage = None
        squat_stage = None

        # Viz joint angle
        viz_joint_angle(image, angle, elbow)

```

Snippet 3.64

Code snippet 3.64 defines a function which would count repetitions based on the frame provided considering the type of exercise passed into it and updating the global state.

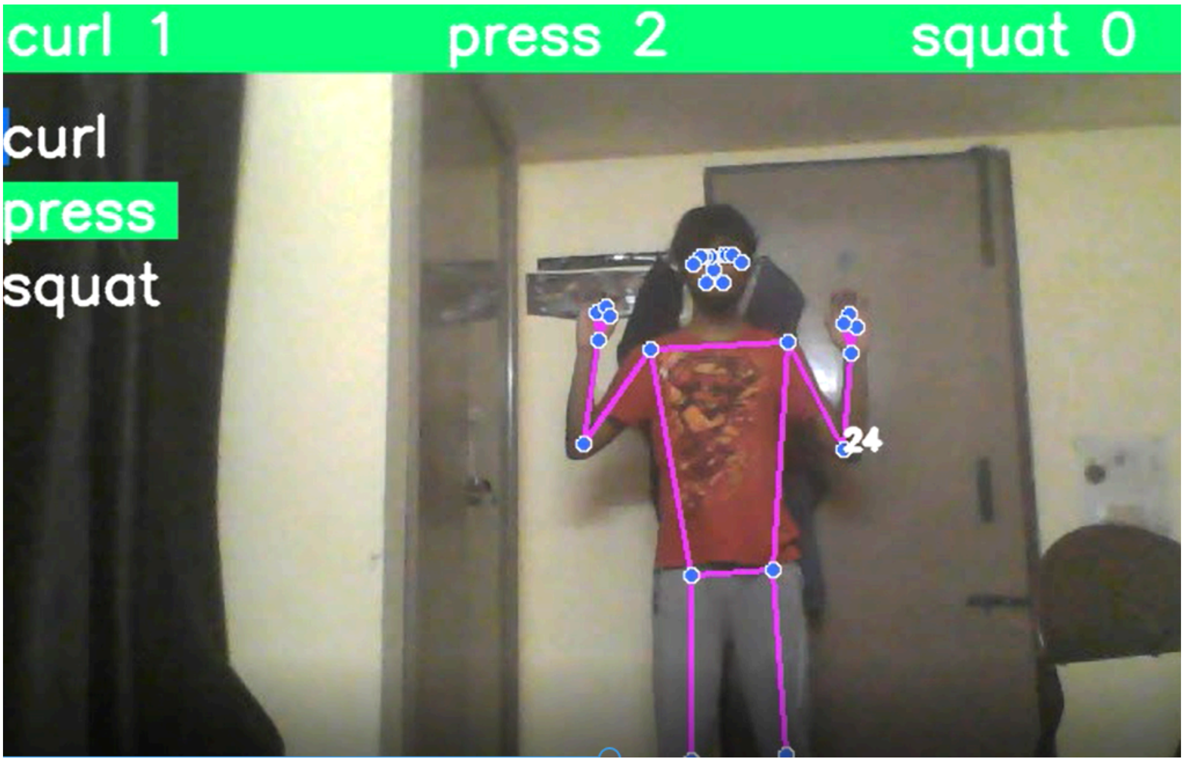


Figure 3.12

The figure 3.12 depicts the whole project in action where the exercise is being tracked and its repetition is being counted as well.

3.5 Key Challenges

- **Lack of good research**

  Working with sensor data is not a day to day task like working with images or plain text. Sensor data is highly scarce and hence finding good research was not an easy task. After thorough consultation with our supervisor, we got to know about a few good researches which enabled us to study proper methodology to work and excel with sensor data.

- **Development of custom repetition count algorithm**

  All of the research that has happened in the context of our project, mainly focuses on classification of exercises with some machine learning algorithm. However, if you want to actually build a fitness tracker product, just classifying exercise is of no use. We must classify as well as count the number of repetitions. It was a great task for us to develop one such algorithm. After much thought, we identified a pattern that the local maximas in the most dominant attribute could be counted as one repetition. We tried and tested this method and it turned out to be a great solution for our problem.

Chapter 04

Testing

4.1 Testing Strategies

As of the current status of *Fit Barbell Classifier,* there's no UI of it, so sophisticated testing is not possible. However, *Unit Testing* is done on both classification and the repetition count algorithm.

Strategy: The strategy for unit testing is simple. For classification, random rows from the unseen data (test data) have been selected then passed to the unit tests created. A total of 5 random rows are selected for classification unit testing. For the repetition count algorithm, one set is passed to the unit test function and the tests are run.

Tools: For unit testing, python's inbuilt *unittest framework* [27] is used. The *unittest* is a simple framework based on *object oriented principles.* To use it, we must create a class which inherits from *unittest.TestCase class.* Then we write our unit test functions, one important thing to remember is that all of the unit test functions must begin with 'test'. We can *assert* using the *assertEqual* to check whether the actual value is equal to expected value or not. If they are not equal then the test will end in a failure, otherwise the test will run normally.

4.2 Test Cases

Test Case 1: Unit Test for the repetition count algorithm

Input: A dataframe of one particular set

Expected Output: If the set is heavy, then 5 else if the set is medium then 10.

```
#Unit Testing for Repetition Count algorithm
import unittest
class TestRepetitonCount(unittest.TestCase):
    def setUp(self):
      self.count_reps_test=count_reps_test

    def test_predict_set1(self):
        s=random.choice(df["set"].unique())
        subset=df[df["set"]==s]
        column="acc_r"
        cutoff=0.4

        if(subset["label"].iloc[0]=="squat"):
          cutoff=0.35

        if(subset["label"].iloc[0]=="row"):
          cutoff=0.65
          column="gyr_x"

        if(subset["label"].iloc[0]=="ohp"):
          cutoff=0.35

        reps_predicted=self.count_reps_test(subset,cutoff,10,column)
        reps_actual=subset["reps"].iloc[0]
        self.assertAlmostEqual(reps_predicted, reps_actual, delta=1)

unittest.main(argv=[''],verbosity=2,exit=False)
```

```
test_predict_set (__main__.TestRepetitonCount) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.030s

OK
<unittest.main.TestProgram at 0x7b0967e67700>
```

Figure 4.1 Unit Test Function

The figure 4.1 shows the unit test function used for testing of the custom repetition count algorithm.

Chapter 05

Results and Evaluation

5.1 Results

Our project "**Fit Barbell Classifier**" has two phases of results, one of exercise classification and the other for repetition count.

- **Exercise classification:**

a) Accelerometer & Gyroscope:

The sensor data of accelerometer and gyroscope was trained on a variety of models including a *simple neural network, random forest, decision tree, kNN* and naive bayes. The training and testing split was 75% and 25% respectively. Since it's a classification problem therefore, various evaluation metrics like accuracy, precision, recall and f-score are shown below:

Serial No.	Model	Accuracy	Precision	Recall	F1-Score
1.	Neural Network	0.993795	0.993947	0.993795	0.993795
2.	Random Forest	0.993795	0.993852	0.993795	0.993794
3.	Decision Tree	0.984488	0.984669	0.984488	0.984456
4.	Naive Bayes	0.963806	0.963896	0.963806	0.963757
5.	K-Nearest Neighbors	0.972079	0.972290	0.972079	0.972074

Table 5.1 Performance Metrics on Feature Set 4

S.No.	Model	Accuracy	Precision	Recall	F-1
1.	SVM	0.961737	0.962901	0.961737	0.961776
2.	RF	0.983454	0.983567	0.983454	0.983473
3.	LR	0.972079	0.972289	0.983454	0.983473
4.	DT	0.949328	0.949506	0.949328	0.949225
5.	KNN	0.922441	0.922881	0.922441	0.922481
6.	NB	0.935884	0.935813	0.935884	0.935725
7.	Stacked Model 1 (DT, LR, SVM, kNN, NB)	0.97931	0.97932	0.97931	0.97930
8.	Stacked Model 2 (RF, LR, DT)	0.987590	0.987613	0.987590	0.987590
9.	Stacked Model 3 (RF, DT, LR, SVM, kNN, NB)	0.98345	0.98353	0.983453	0.98344

Table 5.2 Performance Metrics on feature set 3

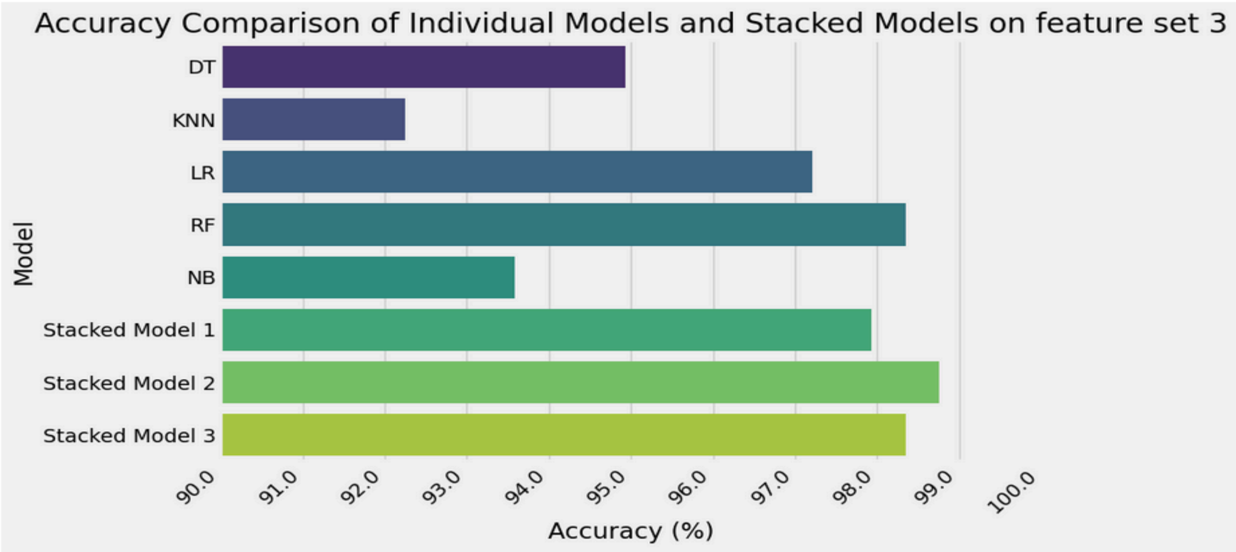


Figure.5.1 Accuracy graph on feature set 3

The figure 5.1 shows the comparison of accuracy among various models trained on feature set 3.

b) Camera/Video:

For **video evaluation** SVM achieved following evaluation metrics:

S.No.	Metric	Value
1.	Accuracy	0.941118
2.	Precision	0.950000
3.	Recall	0.941000
4.	F1-Score	0.941000

Table 5.3 Performance Metrics of Camera Evaluation

- **Repetition Count:** The other puzzle of the project is the repetition count algorithm. The sets were divided into two categories; heavy and medium. Heavy set had 5 repetitions whereas the medium set had 10 repetitions. The algorithm had to count the number of repetitions given any set. *Mean absolute error* was chosen as the evaluation metric and it turned out to be that the *mean absolute error* for the repetition count algorithm for all the sets was *0.88,* which means that on an average the repetition count algorithm is off by just *0.88 repetitions* for any set of any exercise.

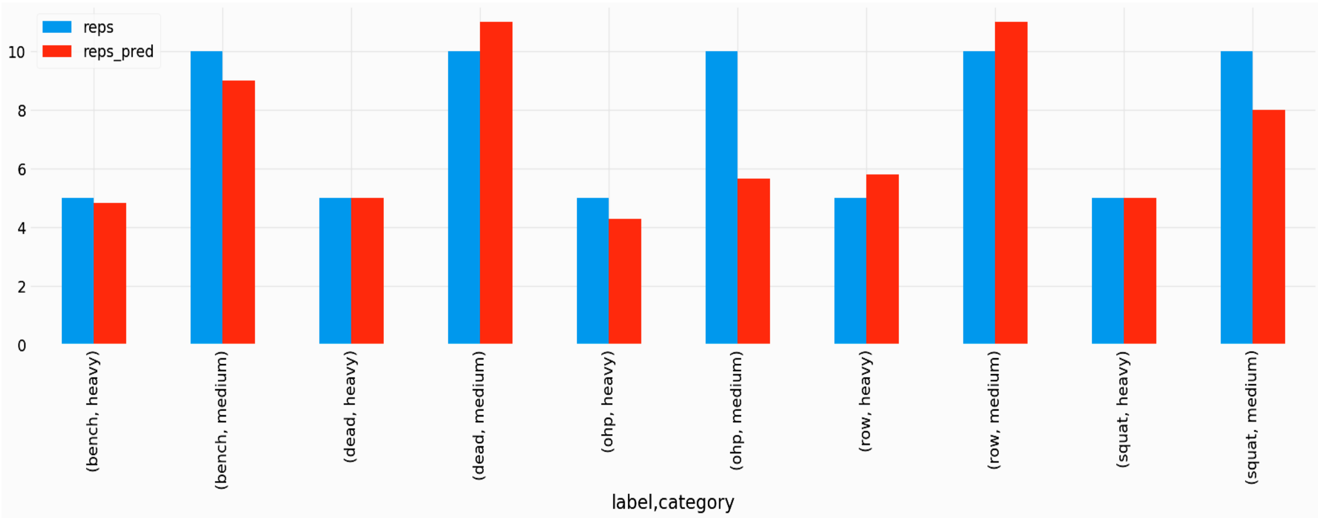


Figure 5.2 Actual Repetition vs Predicted Repetitions

The figure 5.2 shows the comparison between the actual repetition count and the predicted count.

Chapter 06

Conclusion and Future Scope

6.1 Conclusion

Our *Fit Barbell Classifier* is one of those projects which are unique in its own way. By combining the powers of machine learning and sensor data, an automated solution for fitness/workout tracking can be made possible. Any device that has an accelerometer and a gyroscope or a camera could be converted into a fitness tracker.

- **Key Findings:**
    - Tracking through gyroscope & accelerometer is slightly better than video in terms of accuracy.
    - Feature engineering techniques like frequency and temporal abstraction came out to be one of the most important steps to extract meaningful patterns from the raw data.
    - The *ten features* selected through forward selection performed almost similarly when compared to a total *116 features.*
    - *Random Forest and Simple Neural Network* performed best among all of the models for feature set 4.
    - For feature set 3, the stacked model 2 performed the best.
    - *kNN* when trained with only basic features performed the worst.
    - The models were also trained on every other participant but one for better generalization and they performed very well.
    - Repetition algorithm didn’t require any machine for either of the trackings.

- **Limitations:**
  - The data on which the model has been trained on has been collected from a wearable sensor on the wrist. For better coherence with the UI, models should be trained on the data collected through a mobile device.
  - The data has been collected from *five participants* only. Data from more participants can better generalize the model for exercises.
  - At this point, the model is trained on only *five barbell exercises.*
  - The *Fit Barbell Classifier* is currently inaccessible due to a lack of deployment through an app or a website.

- **Contribution to the field:**
  - The potential of *Fit Barbell Classifier* is massive and can completely revolutionize the fitness industry. As of now, there's no such product in the market that can automate the tracking of barbell workouts. The closest products for automated human activity recognition are smartwatches and fitness bands [28] which can track activities like walking, running, swimming and cycling. *The Fit Barb]ell Classifier* opens this same door. Possibilities are endless, from acting as a personal trainer to automatically counting the repetitions to even suggesting workout routines based on the individual's information, *Fit Barbell Classifier* has got you covered.

6.2 Future Scope

The Future scope of *Fit Barbell Classifier* are, but not limited to:

- Collect sensor data from mobile devices to train the model on it for better generalization.
- Include more barbell exercises for both sensor and video tracking in the final system.
- Include more participants in the data collection process.
- Develop an easy accessible user interface so that the end user can track workouts without any hassle.

References

[1] C. J. Coleman, D. J. McDonough, Z. C. Pope, and C. A. Pope, "Dose-response association of aerobic and muscle-strengthening physical activity with mortality: a national cohort study of 416,420 US adults," Br J Sports Med, vol. -, no. -, p. bjsports-2022-105519, Feb. 2024.

[2] W. R. Thompson, "Worldwide Survey of Fitness Trends for 2023," ACSM's Health & Fitness Journal, Jan./Feb. 2023.

[3] "Weightlifting is better for the heart than cardio," News-Medical.Net, Jul. 9, 2019. [Online]. Available:https://www.news-medical.net/news/20190709/Weightlifting-is-better-for-the-heart-than-cardio.aspx.

[4] "Weightlifting Injury: Common Injuries and How to Prevent Them," Integrative Rehabilitation, [Online]. Available: https://integrehab.com/blog/sports-injuries/weightlifting-injury/.

[5] Y. Yoshida and E. Yuda, "Workout Detection by Wearable Device Data Using Machine Learning," Applied Sciences, vol. 13, no. 7, 2023.

[6] G. A. S. Surek, L. O. Seman, S. F. Stefenon, V. C. Mariani, and L. d. S. Coelho, "Video-Based Human Activity Recognition Using Deep Learning Approaches, *Sensors*, 2023.

[7] U. Aiman and T. Ahmad, "Video Based Exercise Recognition Using GCN," in 2023 International Conference on Recent Advances in Electrical, Electronics & Digital Healthcare Technologies, 2023.

[8] C. O'Brien and C. H. Min, "Classification of Various Workout Motions Using Wearable Sensors," , IEEE World AI IoT Congress (AIIoT), 2022.

[9] S. Bhatia, "Activity Identification using Supervised Machine Learning on Wearable Activity Tracker Data," 2021 Asian Conference on Innovation in Technology (ASIANCON), 2021.

[10] M. Yerramsetti, "Human Activity Recognition using ML Techniques," International Journal of Innovative Science and Research Technology, vol. 6, no. 9, 2021.

[11] G Şengül, E Ozcelik, S Misra, R Damaševičius, R Maskeliūnas , "Fusion of smartphone sensor data for classification of daily user activities," Multimedia Tools and Applications_International Journal, 2021.

[12] E. Preatoni, S. Nodari, and N. F. Lopomo, "Supervised machine learning applied to wearable sensor data can accurately classify functional fitness exercises within a continuous workout,", Frontiers in Bioengineering and Biotechnology, 2020.

[13] S. Ishii, A. Yokokubo, M. Luimula, and G. Lopez, "ExerSense: Physical Exercise Recognition and Counting Algorithm from Wearable Robust to Positioning," Sensors, vol. 21, no. 1, 2020.

[14] T. T. Um, C. D. Yoo, and J. H. Lee, "Exercise motion classification from large-scale wearable sensor data using convolutional neural networks," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.

[15] D. Das, S. M. Busetty, V. Bharti, and P. K. Hegde, "Strength Training: A Fitness Application for Indoor Based Exercise Recognition and Comfort Analysis," in 2017 16th IEEE International Conference on Machine Learning (ICML), 2017.

[16] H. Koskimäki and P. Siirtola, "Recognizing gym exercises using acceleration data from wearable sensors," in Proceedings of the Symposium on Computational Intelligence and Data Mining (CIDM), 2014.

[17] OpenCV Documentation. [Online]. Available:https://www.docs.opencv.org/

[18] MediaPipe Documentation. [Online]. Available: https://www.developers.google.com/mediapipe

[19] Chauvenet’s Criterion. [Online]. Available:https://www.statisticshowto.com/chauvenets-criterion/

[20] Butterworth Filter. [Online]. Available: https://en.wikipedia.org/wiki/Butterworth_filter.

[21] M. Hoogendoorn and B. Funk, "Machine Learning for the Quantified Self: On the Art of Learning from Sensory Data," 1st ed. ©2018

[22] Local Outlier Factor. [Online]. Available: https://en.wikipedia.org/wiki/Local_outlier_factor

[23] Fast Fourier Transform. [Online]. Available: http://en.wikipedia.org/wiki/Fast_Fourier_transform

[24] Pandas Documentation. [Online]. Available: https://pandas.pydata.org/docs/

[25] Matplotlib Documentation. [Online]. Available: https://matplotlib.org/docs/

[26] Scikit-learn Documentation. [Online]. Available: https://scikit-learn.org/0.21/documentation.html

[27] Unittest framework. [Online]. Available: https://docs.python.org/3/library/unittest.html

[28] Fitness Bands: All About Their Tracking Features Reliance Digital Solutionbox. Available: https://www.reliancedigital.in/solutionbox/fitness-bands-all-about-their-tracking-features

ss

ORIGINALITY REPORT

6%	5%	5%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.mdpi.com Internet Source	1%
2	www.coursehero.com Internet Source	1%
3	rgu-repository.worktribe.com Internet Source	<1%
4	Submitted to Massey University Student Paper	<1%
5	researchoutput.csu.edu.au Internet Source	<1%
6	Martina Ravizza, Laura Giani, Francesco Jamal Sheiban, Alessandra Pedrocchi, John DeWitt, Giancarlo Ferrigno. "IMU-based classification of resistive exercises for real-time training monitoring on board the international space station with potential telemedicine spin-off", PLOS ONE, 2023 Publication	<1%
7	researchportal.bath.ac.uk Internet Source	<1%

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ……………………….

Type of Document (Tick): **PhD Thesis** **M.Tech/M.Sc. Dissertation** **B.Tech./B.Sc./BBA/Other**

Name:______________________________**Department:**__________________**Enrolment No** __________

Contact No.______________________________**E-mail.**____

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): __________________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at................ (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Abstract & Chapters Details**	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
			Character Counts	
Report Generated on		**Submission ID**	Page counts	
			File Size	

Checked by
Name & Signature **Librarian**

……

Please send your complete Thesis/Report in (PDF) & DOC (Word File) through your Supervisor/Guide at **plagcheck.juit@gmail.com**