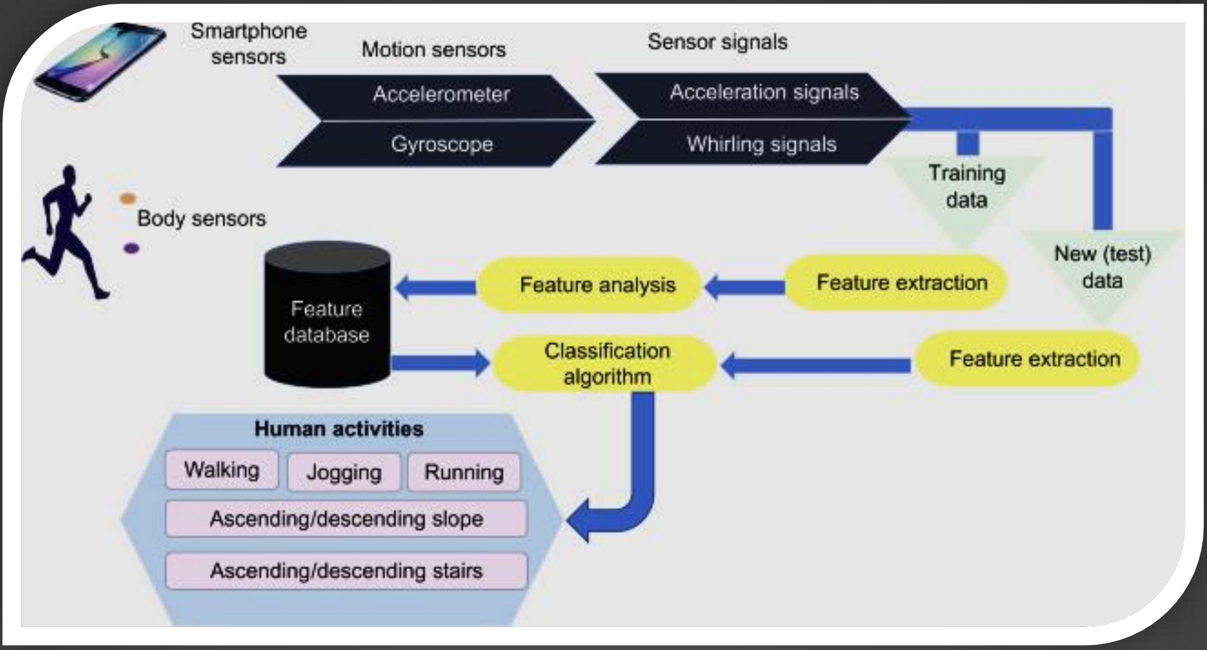
Sheffield Hallam University | Faculty of Science, Technology & Arts

2023



Activity Context Recognition Model (ACRM) For A Fitness System Application

PYTHON PROGRAMMING IMPLEMENTATION REPORT

FATOLA, JOBA (STUDENT NUMBER - 32074392)

TABLE OF CONTENTS

CHAPTER 1- INTRODUCTION

Human activity recognition (HAR) utilises data (acceleration, rotation speed, geographic coordinates, etc.) from personal digital devices, such as smartphones, to classify human activities. The widespread adoption of smartphones with built-in sensors has provided opportunities for studying human behaviour and health (Wilhelm & Kasbauer, 2021). Context-aware computing, enabled by technological advancements, allows for the development of personalised applications that adapt to user circumstances, enhancing user experiences across various sectors, including academics, health & hospitality, sports, and business (Tibensky & Kompan, 2021).

This project aims to develop an intelligent model for a mobile fitness application that accurately predicts human activities, such as walking upstairs, downstairs, sitting, standing, and laying down, leveraging smartphone sensor data. Dealing with low-level raw data poses a challenge, which is addressed through statistical feature extraction methods. The model is thoroughly analysed, designed, implemented, and evaluated, enabling effective recognition of different activity contexts. This development opens possibilities for personalised fitness guidance and promoting healthier lifestyles.

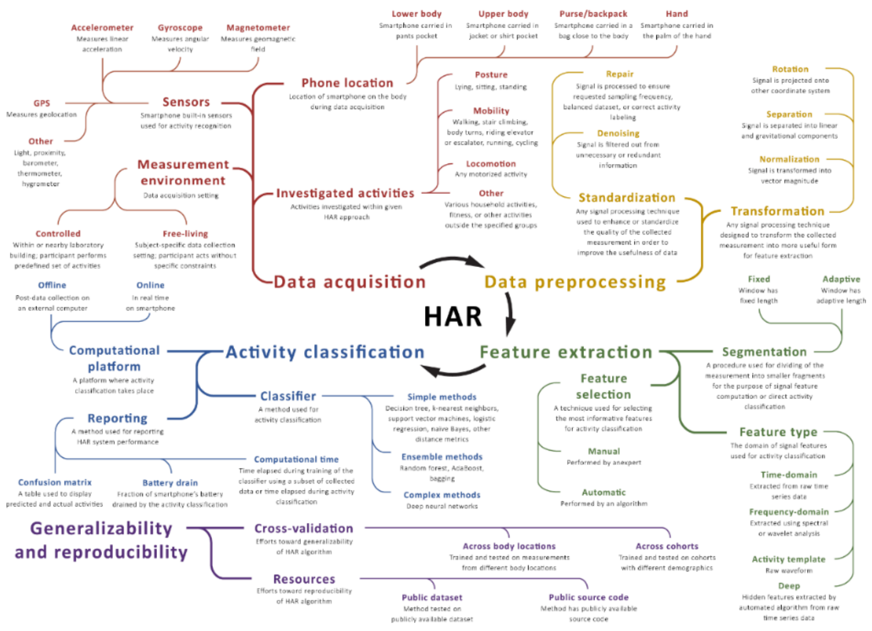

Accelerometer
Measures linear acceleration
Gyroscope
Measures angular velocity
Magnetometer
Measures geomagnetic field
Lower body
Smartphone carried in pants pocket
Upper body
Smartphone carried in jacket or shirt pocket
Purse/backpack
Smartphone carried in a bag close to the body
Hand
Smartphone carried in the palm of the hand
GPS
Measures geolocation
Sensors
Smartphone built-in sensors used for activity recognition
Phone location
Location of smartphone on the body during data acquisition
Other
Light, proximity, barometer, thermometer, hygrometer
Posture
Lying, sitting, standing
Mobility
Walking, stair climbing, body turns, riding elevator or escalator, running, cycling
Locomotion
Any motorized activity
Other
Various household activities, fitness, or other activities outside the specified groups
Repair
Signal is processed to ensure requested sampling frequency, balanced dataset, or correct activity labeling
Denoising
Signal is filtered out from unnecessary or redundant information
Rotation
Signal is projected onto other coordinate system
Separation
Signal is separated into linear and gravitational components
Normalization
Signal is transformed into vector magnitude
Measurement environment
Data acquisition setting
Investigated activities
Activities investigated within given HAR approach
Standardization
Any signal processing technique used to enhance or standardize the quality of the collected measurement in order to improve the usefulness of data
Transformation
Any signal processing technique designed to transform the collected measurement into more useful form for feature extraction
Controlled
Within or nearby laboratory building; participant performs predefined set of activities
Free-living
Subject-specific data collection setting; participant acts without specific constraints
Data acquisition
Data preprocessing
Fixed
Window has fixed length
Adaptive
Window has adaptive length
Offline
Post-data collection on an external computer
Online
In real time on smartphone
HAR
Computational platform
A platform where activity classification takes place
Activity classification
Feature extraction
Segmentation
A procedure used for dividing of the measurement into smaller fragments for the purpose of signal feature computation or direct activity classification
Feature selection
A technique used for selecting the most informative features for activity classification
Classifier
A method used for activity classification
Simple methods
Decision tree, k-nearest neighbors, support vector machines, logistic regression, naive Bayes, other distance metrics
Reporting
A method used for reporting HAR system performance
Manual
Performed by anexpert
Feature type
The domain of signal features used for activity classification
Ensemble methods
Random forest, AdaBoost, bagging
Computational time
Time elapsed during training of the classifier using a subset of collected data or time elapsed during activity classification
Time-domain
Extracted from raw time series data
Confusion matrix
A table used to display predicted and actual activities
Battery drain
Fraction of smartphone's battery drained by the activity classification
Complex methods
Deep neural networks
Automatic
Performed by an algorithm
Frequency-domain
Extracted using spectral or wavelet analysis
Generalizability and reproducibility
Cross-validation
Efforts toward generalizability of HAR algorithm
Across body locations
Trained and tested on measurements from different body locations
Across cohorts
Trained and tested on cohorts with different demographics
Activity template
Raw waveform
Resources
Efforts toward reproducibility of HAR algorithm
Public dataset
Method tested on publicly available dataset
Public source code
Method has publicly available source code
Deep
Hidden features extracted by automated algorithm from raw time series data

Figure 1: *Human activity recognition (HAR) concepts at a glance.*

CHAPTER 2- PROBLEM ANALYSIS

Smartphones, with their widespread adoption and built-in sensors, provide an excellent opportunity for health research and monitoring daily activities. With billions of mobile devices in use, equipped with various sensors (such as an accelerometer, gyroscope, orientation, rotation, magnetic and light), smartphones can record precise measurements of physical activity and other aspects of our lives, (Allahbakhshi et al., 2020). They are versatile instruments for accurately assessing both traditional and emerging risk factors across different populations.

The use of smartphones allows for monitoring behavioral aspects like sedentary behavior, sleep, and physical activity in real-life settings, leveraging individuals' unique experiences. Furthermore, addressing challenges related to low-level data, statistical feature extraction, and data balancing techniques can improve the performance of classifier models for more accurate activity context recognition.

This report focuses on developing an intelligent activity context recognition model for a fitness company, emphasising the importance of pre-processing low-level data to reduce bias and improve prediction accuracy, ultimately leading to improved customer service and business growth. The objective is to develop a robust and accurate model that can automatically classify and recognize these activities using the sensor data collected from smartphones.

CHAPTER 3- SOLUTION REQUIREMENT

Several programming techniques are used in this project, including pre-existing Python modules and libraries, custom functions/modules, Object-Oriented Programming (OOP) principles, exception handling, file processing, exploratory data analysis, data visualisation, feature selection, activity class, model performance and evaluation. It has also made use of libraries for scientific computing and machine learning. Examples are Pandas for data processing(framing), matplotlib and Seaborn for data visualisation, Imblearn for data balance, and Scikit-Learn for creating machine learning models are important libraries to employ. An integrated development environment (IDE), Jupyter Notebook, is also used to make the programming duties easier. The activity recognition context model for the fitness system application will be made using these methods and technologies.

CHAPTER 4- ACRM PROCESS FLOW DIAGRAM

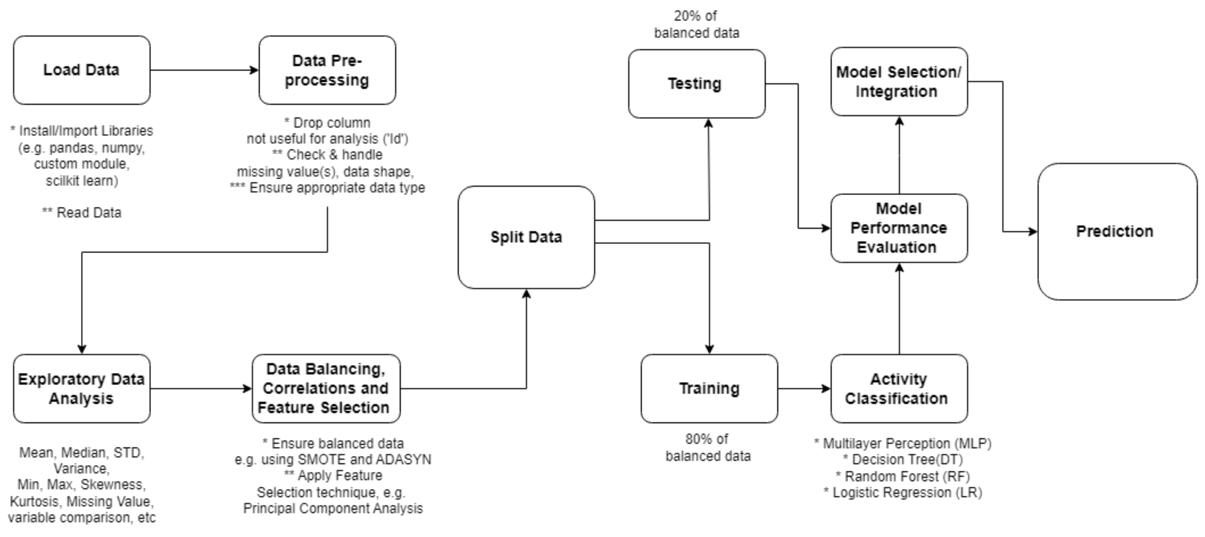

Load Data
* Install/Import Libraries (e.g. pandas, numpy, custom module, scilkit learn)
** Read Data
Data Pre-processing
* Drop column not useful for analysis ('Id')
** Check & handle missing value(s), data shape,
*** Ensure appropriate data type
Exploratory Data Analysis
Mean, Median, STD, Variance, Min, Max, Skewness, Kurtosis, Missing Value, variable comparison, etc
Data Balancing, Correlations and Feature Selection
* Ensure balanced data e.g. using SMOTE and ADASYN
** Apply Feature Selection technique, e.g. Principal Component Analysis
Split Data
20% of balanced data
Testing
80% of balanced data
Training
Activity Classification
* Multilayer Perception (MLP)
* Decision Tree(DT)
* Random Forest (RF)
* Logistic Regression (LR)
Model Performance Evaluation
Model Selection/ Integration
Prediction

Figure 2: *A process flow chart showing the Activity Context Recognition Model for a Fitness Application*

CHAPTER 5- SYSTEM IMPLEMENTATION

Highlighted and described below are the implementation process stages for this project:

A. **Installing and importing libraries/modules:** Libraries like Imblearn were installed, and all required libraries like pandas, Imblearn, NumPy, Scikit-learn, and seaborn included the custom statistic module, which has been created specifically for this data set.
B. **Loading of the dataset**: pandas csv read function, 'read_csv' was used to load the given data set ("activity_context_tracking_data.csv"). This dataset contains data captured from the built-in sensors of a smartphone. These sensors capture data related to individuals' movements, positions, and environmental factors. The dataset consists of various sensors, including the orientation, rotation, accelerometer, gyroscope, magnetic, sound, and light sensors. Accurately recognising different activities relies heavily on the value of this data. The target variable in the dataset is the activity category, which includes 13 distinct activity categories ranging from ascending and descending stairs to driving, jogging, lying, mountain ascending and descending, running, sitting, standing, and walking. Data shape is (205520, 19) as presented in Appendix A: Table 2.
C. **Data Preprocessing and Cleaning**: Various steps were taken to ensure its quality and usability. This involved shaping the data and removing unnecessary rows, such as the "Id" column, which served as a unique identifier. These steps resulted in a clean dataset free from duplicates, missing values, and outliers that could impact the model's performance. With the prepared dataset, further analysis can now be conducted, including exploratory data analysis, correlation analysis, addressing class imbalances, and feature selection.
D. **Statistical analysis (Exploratory Data Analysis):** The dataset was thoroughly analysed to understand its properties and distribution of activity groups. Visualisations, including box plots, frequency plots, bar charts, and histograms, were used to extract meaningful insights. The analysis revealed that sitting and walking were the most common activities, followed by lying and standing. Statistical measures such as mean, median, standard deviation, skewness, variance, kurtosis, minimum, maximum, and range were calculated for each sensor characteristic to further explore the dataset's patterns and distributions. Appendix A: Table 3 provides a summary statistics data frame table and plots for reference.
E. **Data balancing, correlations, and feature selection**: Upon investigation of the distribution of the activity categories column in the dataset, noteworthy/obvious class imbalances, which might lead to biased training, were identified in which certain activities were overrepresented while others were underrepresented. To tackle this problem, methods like **SMOTE** (Synthetic Minority Over-Sampling Technique) and **Random Oversampling** were employed to balance the classes by creating synthetic samples for the underrepresented ones (Kurniawati, 2019). This method contributed to a more balanced dataset for training our activity context recognition model. See Appendix A: plot 10

   Outliers were identified in specific features ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', and 'mZ'] of the dataset, accounting for around 7% of the data. Due to the absence of subject matter experts to validate these values, two approaches were employed. The first approach involved retaining the outliers as they are, while the second approach utilised winsorization/trimming. With winsorization, the outlier values were replaced by the nearest non-outlier values, using the 5th and 95th percentiles as limits. The impact of these approaches on model performance is discussed and presented in Table 1 of the model performance evaluation. Appendix A: plot 1a and 1b show the box plot before and after applying winsorization.

   Additionally, correlation matrices were used to analyse the interactions between different features in order to find dependencies and collinearity. Likewise, Principal Component Analysis (PCA) was used to determine the most informative and relevant features. The original sensor characteristics were converted into uncorrelated principal components that captured the most variance in the data using PCA. To ensure that the primary components accounting for a minimum of 90% of the variance in the dataset were retained, a threshold of 0.9 for the cumulative explained variance ratio was established.

Please refer to APPENDIX A: plots 2, 10, 11, 8 and 9 for a more comprehensive look at the data analysis, including visualisations, balanced data plots, and the correlation matrix.

F. **Data splitting**: The balanced dataset was split into training and testing dataset in preparation for further modelling processes in the ratio 75:25. That is, 75% of the balanced dataset is used for training a model, and the remaining 25% is used for testing to validate the performance of the model. Appendix A holds the training and testing data frames.

G. **Feature selection**: The *SelectKBest* value with a k value set to 12 is used to select the top 11 most informative features for the classification task (*the activity column is set as the target variable and already hot encoded at this point to make it numeric*). The scoring function used is *f_classif,* which calculates the ANOVA F-value between each feature and the target variable. The F-value measures the degree of linear dependency between each feature and the target variable. The selected features of the SMOTE dataset are orX, rY, rZ, accY, gY, gZ, mX, mY, mZ, lux, soundLevel. Additionally, the random state is set at 42 to ensure that it has consistent results across different runs of the algorithm. This is useful for debugging and reproducibility purposes. See Appendix A: Table 4

H. **Activity classification**: Four classifiers (MLP, DT, RF, and LR) were trained and assessed using the training and testing datasets. Logistic Regression (LR) was chosen for binary classification and modelling variable relationships. A Decision Tree (DT) provided non-linear decision-making based on a tree structure. Random Forest (RF) used an ensemble approach with multiple decision trees. Multilayer Perceptron (MLP) captured complex patterns with interconnected layers. These algorithms were selected to maximize predictive performance and accommodate diverse data patterns evaluate and choose the best strategy for the activity recognition task.

I. **Model performance and evaluation:** The metrics used for evaluation are **Accuracy** (which will answer how correct is the prediction), **Precision** (will answer how many were correct out of the predicted positives), **Recall** (how many were correct out of the actual positives), and **F1 score** (the harmonic mean of Precision and Recall). The result of the confusion matrix for each model is also presented in Appendix A: plots 12-15. Table 1 below presents each model's performance with & without winsorization:

Table 1: MODEL PERFORMANCE EVALUATION METRICS BEFORE AND AFTER WINSORIZATION

SN	MODEL	ACCURACY (True Positive +True Negatives) / (Total Predictions)			PRECISION (True Positives) / (True Positives + False Positives)			RECALL (True Positives) / (True Positives + False Negatives)			F1 SCORE (2 X Precision X Recall) / (precision + Recall)		
		WoW	WtW	Variance	WoW	WtW	Variance	WoW	WtW	Variance	WoW	WtW	Variance
1	Logistic Regression (LR)	75.173%	75.331%	↑ 0.158%	73.767%	73.963%	↑ 0.196%	75.173%	75.331%	↑ 0.158%	73.718%	73.892%	↑ 0.174%
2	Decision Tree (DT)	99.893%	99.889%	↓ -0.004%	99.893%	99.889%	↓ -0.004%	99.893%	99.889%	↓ -0.004%	99.893%	99.889%	↓ -0.004%
3	Multilayer Perceptron (MLP)	99.774%	99.741%	↓ -0.033%	99.774%	99.741%	↓ -0.033%	99.774%	99.741%	↓ -0.033%	99.774%	99.740%	↓ -0.034%
4	Random Forest (RF)	99.988%	99.993%	↑ 0.005%	99.988%	99.993%	↑ 0.005%	99.988%	99.993%	↑ 0.005%	99.988%	99.993%	↑ 0.005%

***WoW**: Performance Result Without Winsorization* | ***WtW**: Performance Results With Winsorization*

J. **Exception handling:** Exception handling, such as FileNotFoundError, NameError, AttributeError, and SyntaxError, was also extensively implemented across the program for a smooth user experience.

K. **MENU-DRIVEN USER INTERFACE:** This is included in the program to improve user experience during implementation.

CHAPTER 6- CONCLUSION, RECOMMENDATION AND REFLECTION

In conclusion, the Decision Tree, Multilayer Perceptron, and Random Forest models exhibit strong performance across various metrics. The Decision Tree model demonstrates exceptional performance, the Random Forest model achieves near-perfect scores, and the Multilayer Perceptron model performs well, comparable to the Decision Tree model.

Based on the observations presented in Table 1 above, the following recommendations are made:

1. **Impact of Potential Outliers**: The potential outliers identified during exploratory data analysis did not significantly affect the model's performance. This conclusion is supported by the minimal differences observed in metric scores between models, such as Decision Tree and Random Forest. Overall, the presence of these outliers had little impact on the performance of the models.

2. **Model Selection**: Considering the performance metrics, the Decision Tree and Random Forest models stand out as the top performers. The Decision Tree model demonstrates exceptional performance, while the Random Forest model achieves near-perfect scores as highlighted in Table 1. Either of these models would be a suitable choices depending on other considerations such as interpretability and computational efficiency. See Appendix A: plots 16 and 17
3. **Further Analysis**: While the provided results indicate strong performance, it is crucial to conduct further analysis and evaluate the model's robustness. Employ techniques like cross-validation to assess the models' performance on multiple train-test splits.

Additionally, consider fine-tuning the models by optimizing hyperparameters or exploring ensemble techniques to potentially improve their performance further.

Reflections: This project served as a valuable opportunity to apply machine learning algorithms and programming concepts using Python. Object-oriented programming played a significant role in enhancing the development process, while the utilization of various libraries facilitated efficient and enjoyable coding. The exploration of winsorization techniques for handling outliers was particularly rewarding, as it demonstrated their impact on model performance. It is worth noting that industry knowledge would have further informed the decision-making process regarding outlier handling. Additionally, experimenting with different data testing sizes consistently yielded reliable model performances. Overall, successfully executing the steps of ML programming and applying statistical knowledge acquired from previous courses was a source of satisfaction and pride.

However, a lot of challenges were observed during this project task like how to apply some libraries properly, which necessitated some further searches for me. An example is when I was not able to implement my custom statistics module until I realised that I must define a variable to store numeric columns from my data frame. Another example was my experience with data leakage, where the data frame for training and testing datasets contains some NaN values. I struggled to resolve this for a very long time until I changed my step to perform data splitting on the balanced dataset before performing feature selection. This ensured proper handling of data leakage without duplicates or NaN values. Additionally, some models, like support vector machines, exhibited slow execution times, prompting the decision to explore alternative models for improved efficiency.

Looking forward, I have acknowledged the challenges faced during this project and will use them as learning opportunities. To improve future model performance, I plan to explore techniques like cross-validation, pipeline functions, hyperparameter optimization, and ensemble methods. Implementing these strategies will enhance the project outcomes and contribute to my overall learning experience.

REFERENCES

Al-Shehari, T., & Alsowail, R. A. (2021, September 27). An Insider Data Leakage Detection Using One-Hot Encoding, Synthetic Minority Oversampling and Machine Learning Techniques. *Entropy*, *23*(10), 1258. https://doi.org/10.3390/e23101258

Kretinin, O., Popov, E., Tsapaev, A., Fedosova, L., & Tyurikov, M. (2021). Synthesis and Visualization of Image Datasets of Parametric 3D Model for Neural Network Training and Testing in Data-Poor Conditions. *Scientific Visualization*, *13*(5). https://doi.org/10.26583/sv.13.5.06

Allahbakhshi, H., Conrow, L., Naimi, B., & Weibel, R. (2020, January 21). Using Accelerometer and GPS Data for Real-Life Physical Activity Type Detection. *Sensors*, *20*(3), 588. https://doi.org/10.3390/s20030588

Plotting a diagonal correlation matrix — seaborn 0.12.2 documentation. (n.d.). Plotting a Diagonal Correlation Matrix — Seaborn 0.12.2 Documentation. https://seaborn.pydata.org/examples/many_pairwise_correlations.html

Recognition of Idleness among Patients based on Activity using Random Forest over Decision Tree Algorithm. (2022, January 1). *Journal of Pharmaceutical Negative Results*, *13*(SO4). https://doi.org/10.47750/pnr.2022.13.s04.217

An Improved Principal Component Analysis (PCA) Face Recognition Technique using Modular Approach. (2023, March 24). *PriMera Scientific Engineering*. https://doi.org/10.56831/psen-02-044

Kurniawati, Y. E. (2019, October 30). Class Imbalanced Learning Menggunakan Algoritma Synthetic Minority Over-sampling Technique – Nominal (SMOTE-N) pada Dataset Tuberculosis Anak. *Jurnal Buana Informatika*, *10*(2), 134. https://doi.org/10.24002/jbi.v10i2.2441

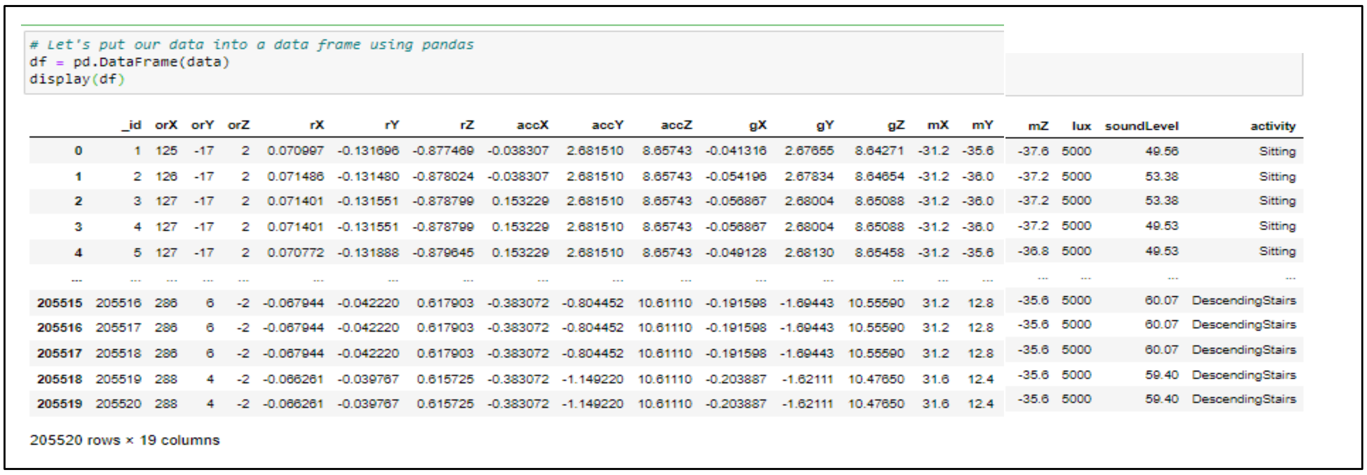
Tibensky, P., & Kompan, M. (2021). Context-aware adaptive personalised recommendation: a meta-hybrid. *International Journal of Web Engineering and Technology*, *16*(3), 235. https://doi.org/10.1504/ijwet.2021.119874

Wilhelm, S., & Kasbauer, J. (2021, December 1). Exploiting Smart Meter Power Consumption Measurements for Human Activity Recognition (HAR) with a Motif-Detection-Based Non-Intrusive Load Monitoring (NILM) Approach. *Sensors*, *21*(23), 8036. https://doi.org/10.3390/s21238036

McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

Errors and Exceptions — Python 3.10.4 documentation. (n.d.). Retrieved March 28, 2022, from https://docs.python.org/3/tutorial/errors.html

APPENDIX A- PLOTS AND GRAPHS

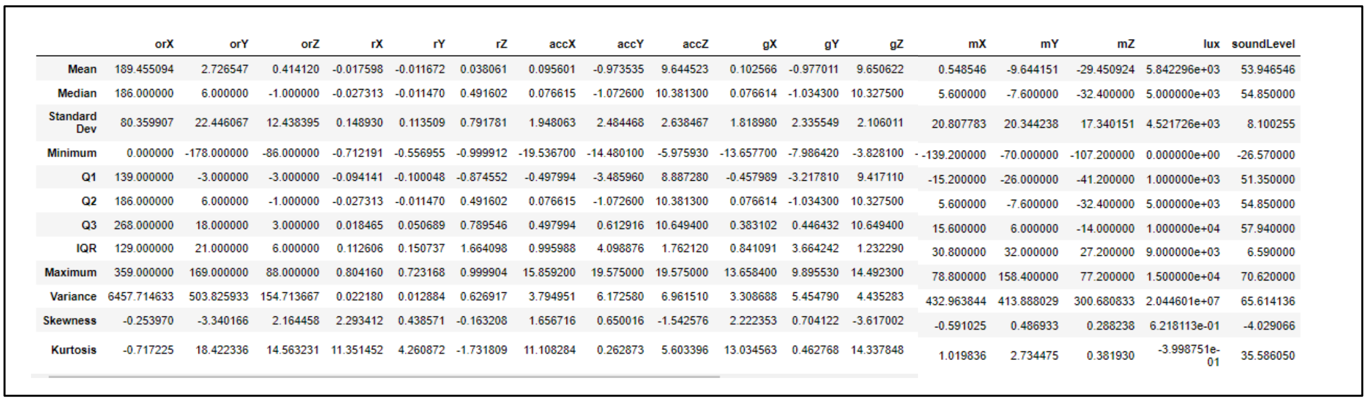


```python
# Let's put our data into a data frame using pandas
df = pd.DataFrame(data)
display(df)
```

	_id	orX	orY	orZ	rX	rY	rZ	accX	accY	accZ	gX	gY	gZ	mX	mY	mZ	lux	soundLevel	activity
0	1	125	-17	2	0.070997	-0.131696	-0.877469	-0.038307	2.681510	8.65743	-0.041316	2.67655	8.64271	-31.2	-35.6	-37.6	5000	49.56	Sitting
1	2	126	-17	2	0.071486	-0.131480	-0.878024	-0.038307	2.681510	8.65743	-0.054196	2.67834	8.64654	-31.2	-36.0	-37.2	5000	53.38	Sitting
2	3	127	-17	2	0.071401	-0.131551	-0.878799	0.153229	2.681510	8.65743	-0.056867	2.68004	8.65088	-31.2	-36.0	-37.2	5000	53.38	Sitting
3	4	127	-17	2	0.071401	-0.131551	-0.878799	0.153229	2.681510	8.65743	-0.056867	2.68004	8.65088	-31.2	-36.0	-37.2	5000	49.53	Sitting
4	5	127	-17	2	0.070772	-0.131888	-0.879645	0.153229	2.681510	8.65743	-0.049128	2.68130	8.65458	-31.2	-35.6	-36.8	5000	49.53	Sitting
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
205515	205516	286	6	-2	-0.067944	-0.042220	0.617903	-0.383072	-0.804452	10.61110	-0.191598	-1.69443	10.55590	31.2	12.8	-35.6	5000	60.07	DescendingStairs
205516	205517	286	6	-2	-0.067944	-0.042220	0.617903	-0.383072	-0.804452	10.61110	-0.191598	-1.69443	10.55590	31.2	12.8	-35.6	5000	60.07	DescendingStairs
205517	205518	286	6	-2	-0.067944	-0.042220	0.617903	-0.383072	-0.804452	10.61110	-0.191598	-1.69443	10.55590	31.2	12.8	-35.6	5000	60.07	DescendingStairs
205518	205519	288	4	-2	-0.066261	-0.039767	0.615725	-0.383072	-1.149220	10.61110	-0.203887	-1.62111	10.47650	31.6	12.4	-35.6	5000	59.40	DescendingStairs
205519	205520	288	4	-2	-0.066261	-0.039767	0.615725	-0.383072	-1.149220	10.61110	-0.203887	-1.62111	10.47650	31.6	12.4	-35.6	5000	59.40	DescendingStairs

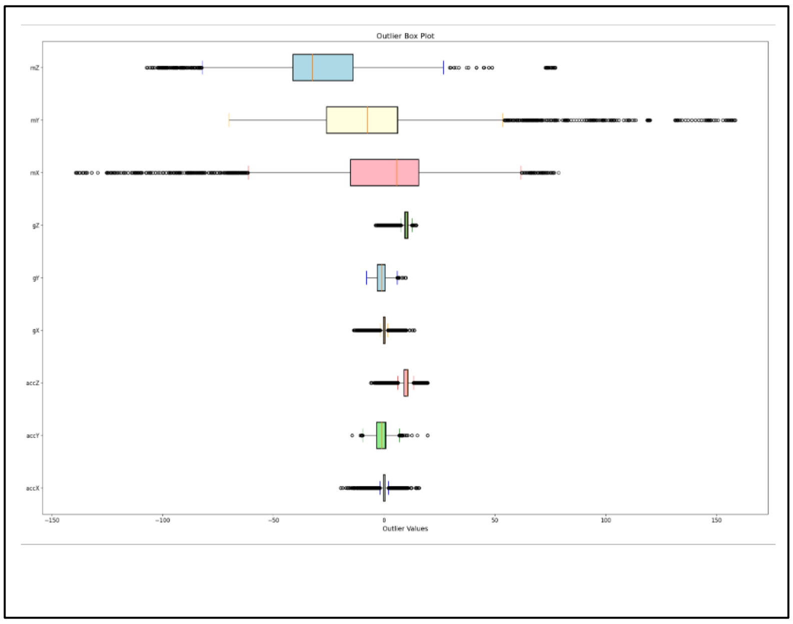
```
205520 rows × 19 columns
```

Table 2: *Pandas Data Frame of Activity Context Data*

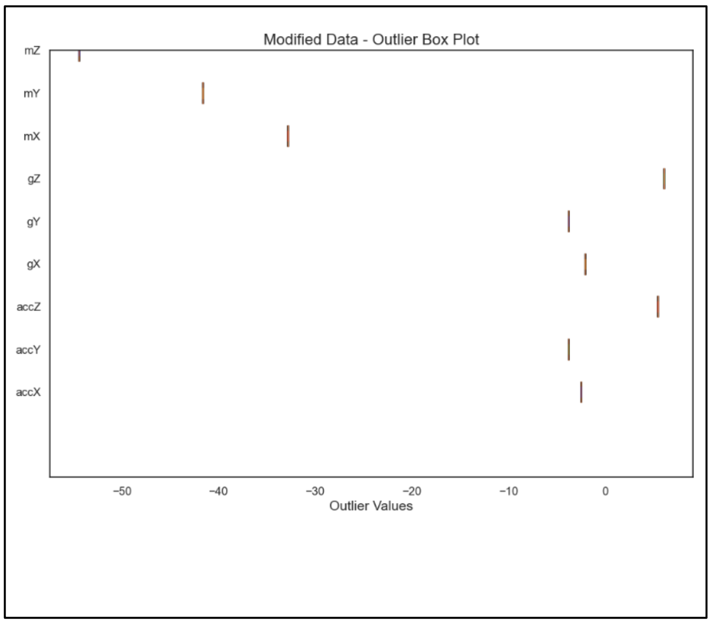


	orX	orY	orZ	rX	rY	rZ	accX	accY	accZ	gX	gY	gZ	mX	mY	mZ	lux	soundLevel
Mean	189.455094	2.726547	0.414120	-0.017598	-0.011672	0.038061	0.095601	-0.973535	9.644523	0.102566	-0.977011	9.650622	0.548546	-9.644151	-29.450924	5.842296e+03	53.946546
Median	186.000000	6.000000	-1.000000	-0.027313	-0.011470	0.491602	0.076615	-1.072600	10.381300	0.076614	-1.034300	10.327500	5.600000	-7.600000	-32.400000	5.000000e+03	54.850000
Standard Dev	80.359907	22.446067	12.438395	0.148930	0.113509	0.791781	1.948063	2.484468	2.638467	1.818980	2.335549	2.106011	20.807783	20.344238	17.340151	4.521726e+03	8.100255
Minimum	0.000000	-178.000000	-86.000000	-0.712191	-0.556955	-0.999912	-19.536700	-14.480100	-5.975930	-13.657700	-7.986420	-3.828100	- -139.200000	-70.000000	-107.200000	0.000000e+00	-26.570000
Q1	139.000000	-3.000000	-3.000000	-0.094141	-0.100048	-0.874552	-0.497994	-3.485960	8.887280	-0.457989	-3.217810	9.417110	-15.200000	-26.000000	-41.200000	1.000000e+03	51.350000
Q2	186.000000	6.000000	-1.000000	-0.027313	-0.011470	0.491602	0.076615	-1.072600	10.381300	0.076614	-1.034300	10.327500	5.600000	-7.600000	-32.400000	5.000000e+03	54.850000
Q3	268.000000	18.000000	3.000000	0.018465	0.050689	0.789546	0.497994	0.612916	10.649400	0.383102	0.446432	10.649400	15.600000	6.000000	-14.000000	1.000000e+04	57.940000
IQR	129.000000	21.000000	6.000000	0.112606	0.150737	1.664098	0.995988	4.098876	1.762120	0.841091	3.664242	1.232290	30.800000	32.000000	27.200000	9.000000e+03	6.590000
Maximum	359.000000	169.000000	88.000000	0.804160	0.723168	0.999904	15.859200	19.575000	19.575000	13.658400	9.895530	14.492300	78.800000	158.400000	77.200000	1.500000e+04	70.620000
Variance	6457.714633	503.825933	154.713667	0.022180	0.012884	0.626917	3.794951	6.172580	6.961510	3.308688	5.454790	4.435283	432.963844	413.888029	300.680833	2.044601e+07	65.614136
Skewness	-0.253970	-3.340166	2.164458	2.293412	0.438571	-0.163208	1.656716	0.650016	-1.542576	2.222353	0.704122	-3.617002	-0.591025	0.486933	0.288238	6.218113e-01	-4.029066
Kurtosis	-0.717225	18.422336	14.563231	11.351452	4.260872	-1.731809	11.108284	0.262873	5.603396	13.034563	0.462768	14.337848	1.019836	2.734475	0.381930	-3.998751e-01	35.586050

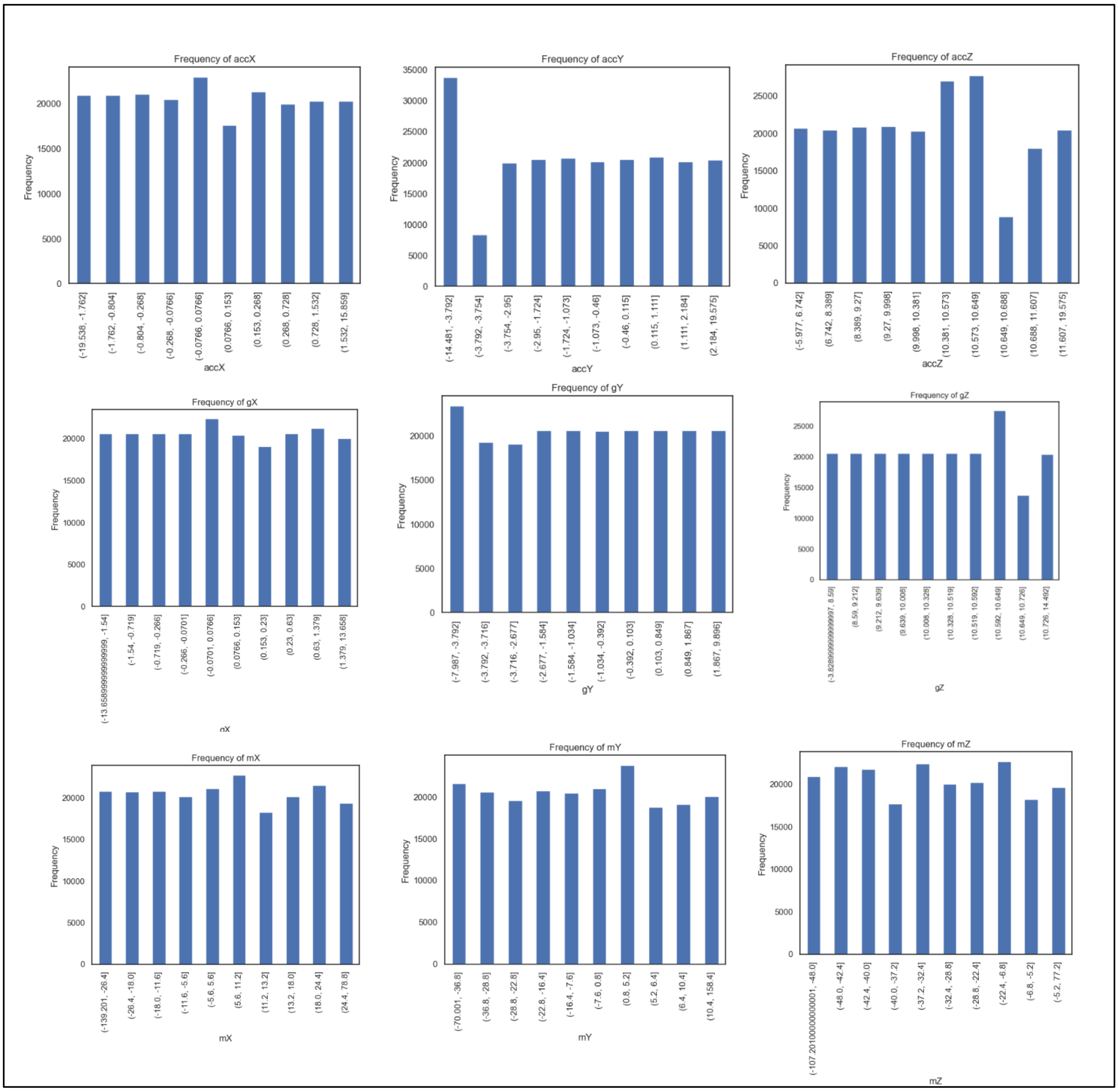
Table 3: *Exploratory Data Analysis of Activity Context Data*

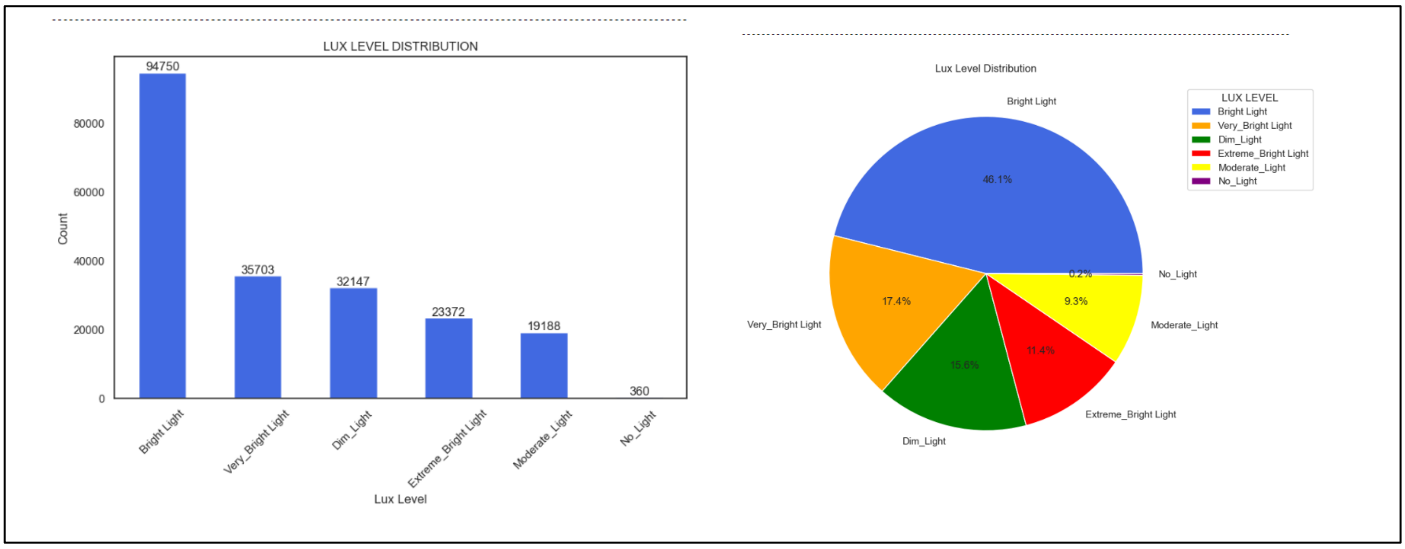


Plot 1a: *Group plots of potential outliers in ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', and 'mZ']*

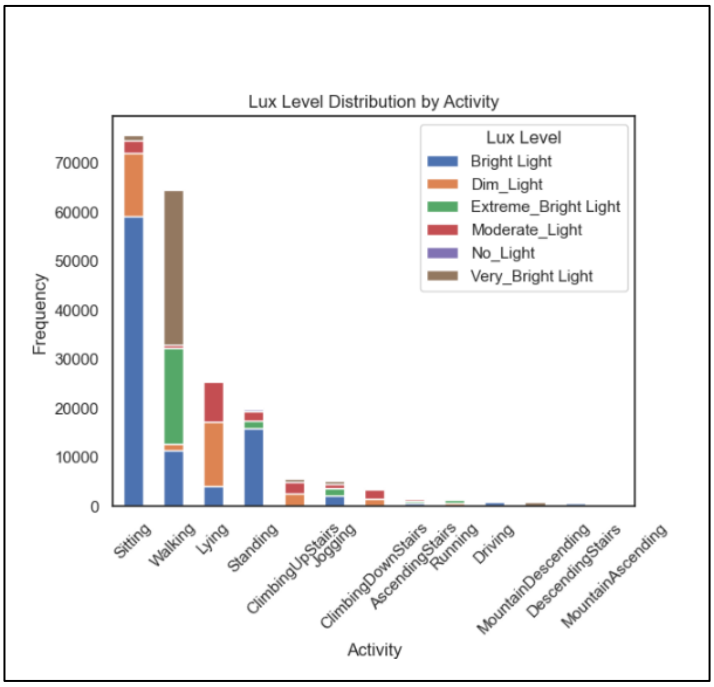


Plot 1b: *Group plots of potential outliers in ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', and 'mZ'] after winsorization*

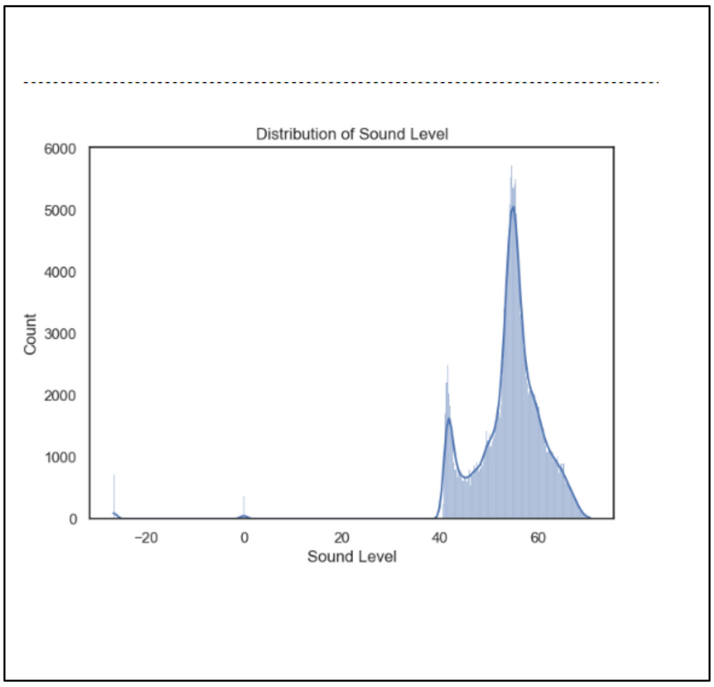

***Plot 2:** Frequency Distribution plots of ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', and 'mZ']*



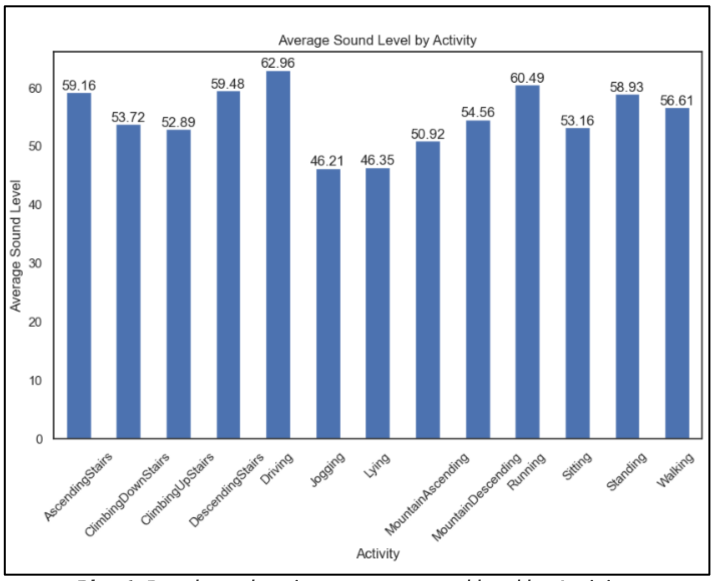
***Plot 3:** Bar and pie chart showing distribution of light (Lux Level) of light sensors*



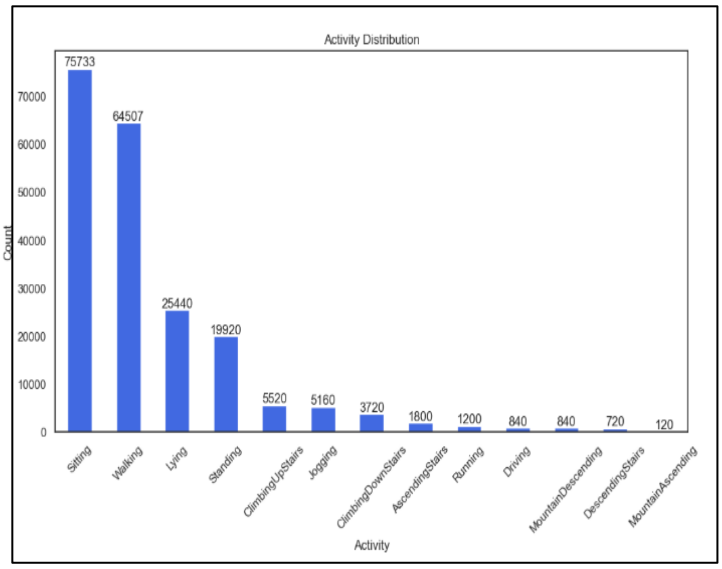
Plot 4: *Stacked bar chart showing lux level distribution by Activity.*



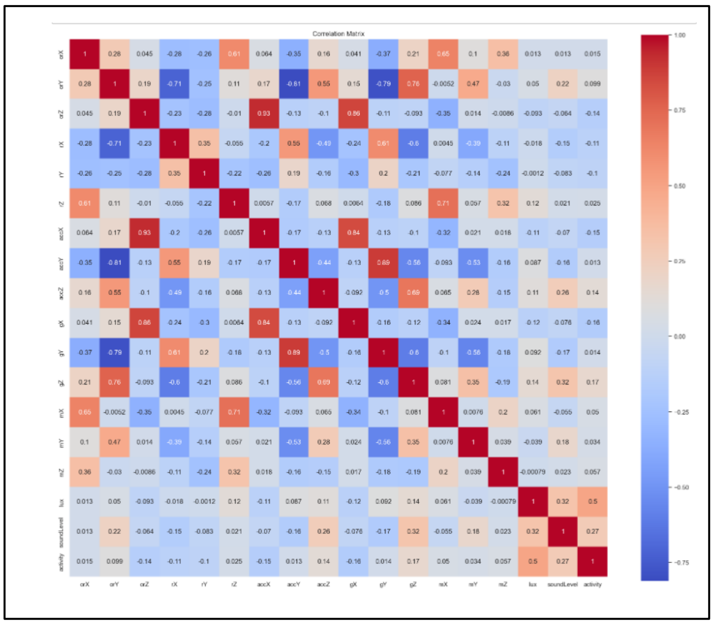
Plot 5: *Density plot showing sound level distribution.*



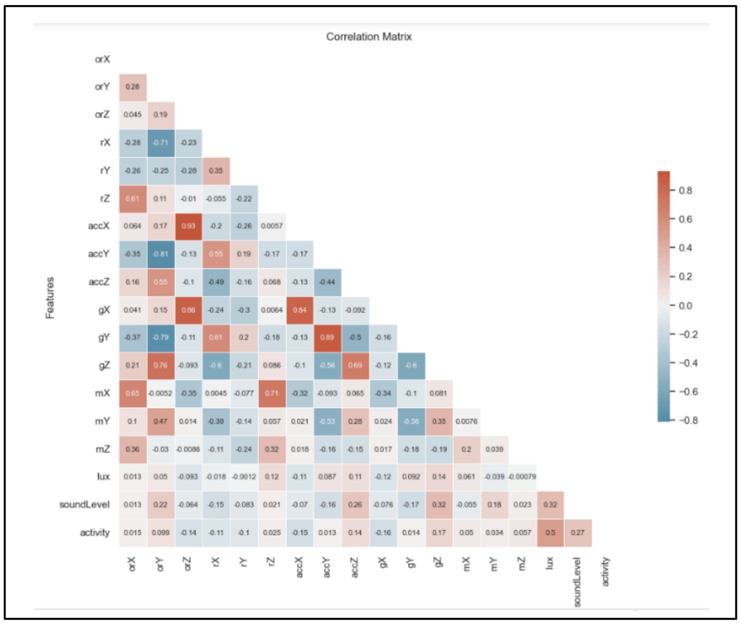
Plot 6: *Bar chart showing average sound level by Activity.*



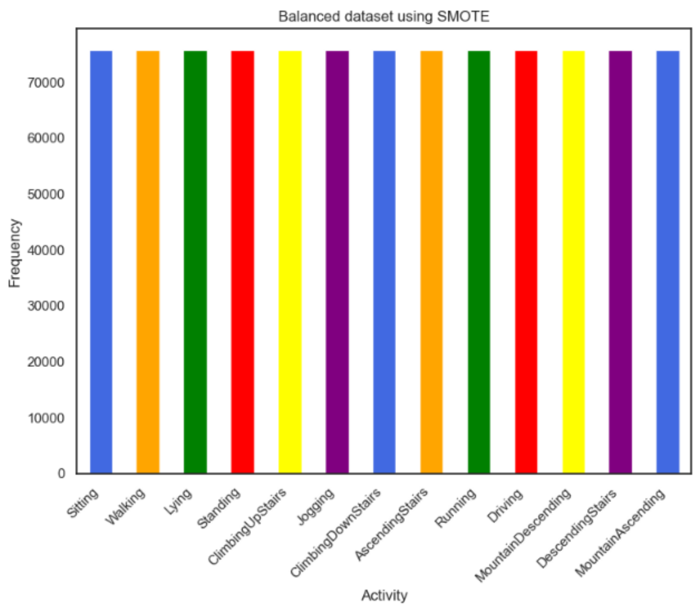
Plot 7: *Bar chart showing imbalance class of Activities.*



Plot 8: *Square Heat map showing correlations between features.*



Plot 9: *Triangle Heat map showing correlations between features.*

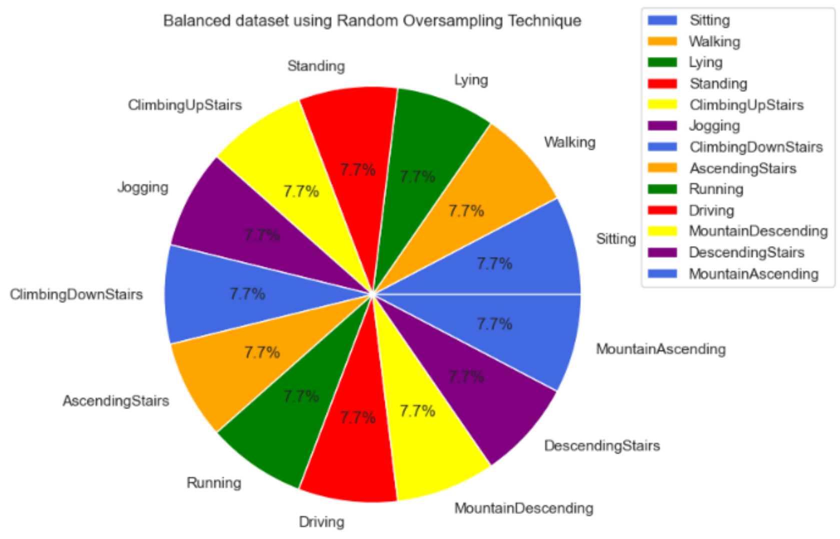


```
-------------------------------------

Sitting               75733
Walking               75733
Lying                 75733
Standing              75733
ClimbingUpStairs      75733
Jogging               75733
ClimbingDownStairs    75733
AscendingStairs       75733
Running               75733
Driving               75733
MountainDescending    75733
DescendingStairs      75733
MountainAscending     75733
Name: activity, dtype: int64
-------------------------------------
```

Plot 10: *Bar chart showing balanced Activity class distribution using SMOTE technique.*

```
-----------------------------------------------------------------
```

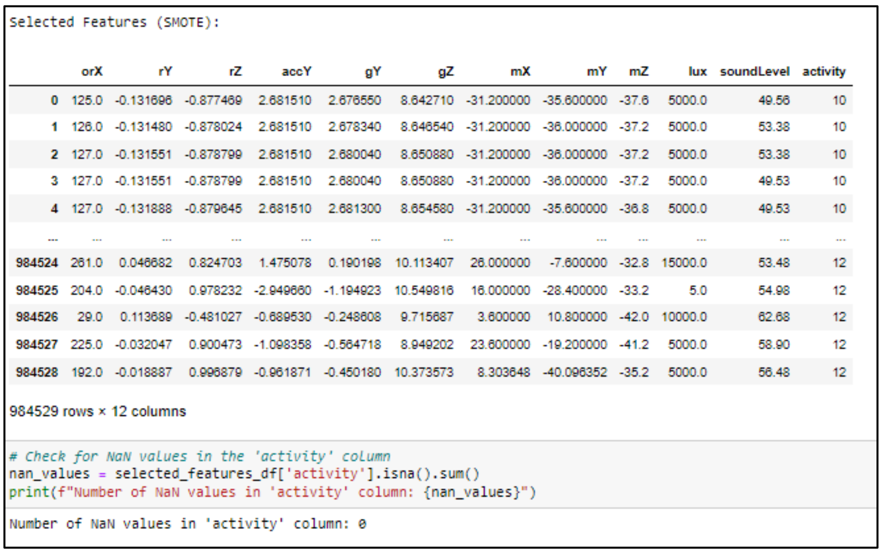


```
-------------------------------------

Sitting               75733
Walking               75733
Lying                 75733
Standing              75733
ClimbingUpStairs      75733
Jogging               75733
ClimbingDownStairs    75733
AscendingStairs       75733
Running               75733
Driving               75733
MountainDescending    75733
DescendingStairs      75733
MountainAscending     75733
Name: activity, dtype: int64
-------------------------------------
```

```
-----------------------------------------------------------------
```

Plot 11: *Pie chart showing balanced Activity class distribution using Random Oversampling Technique.*



```
Selected Features (SMOTE):
```

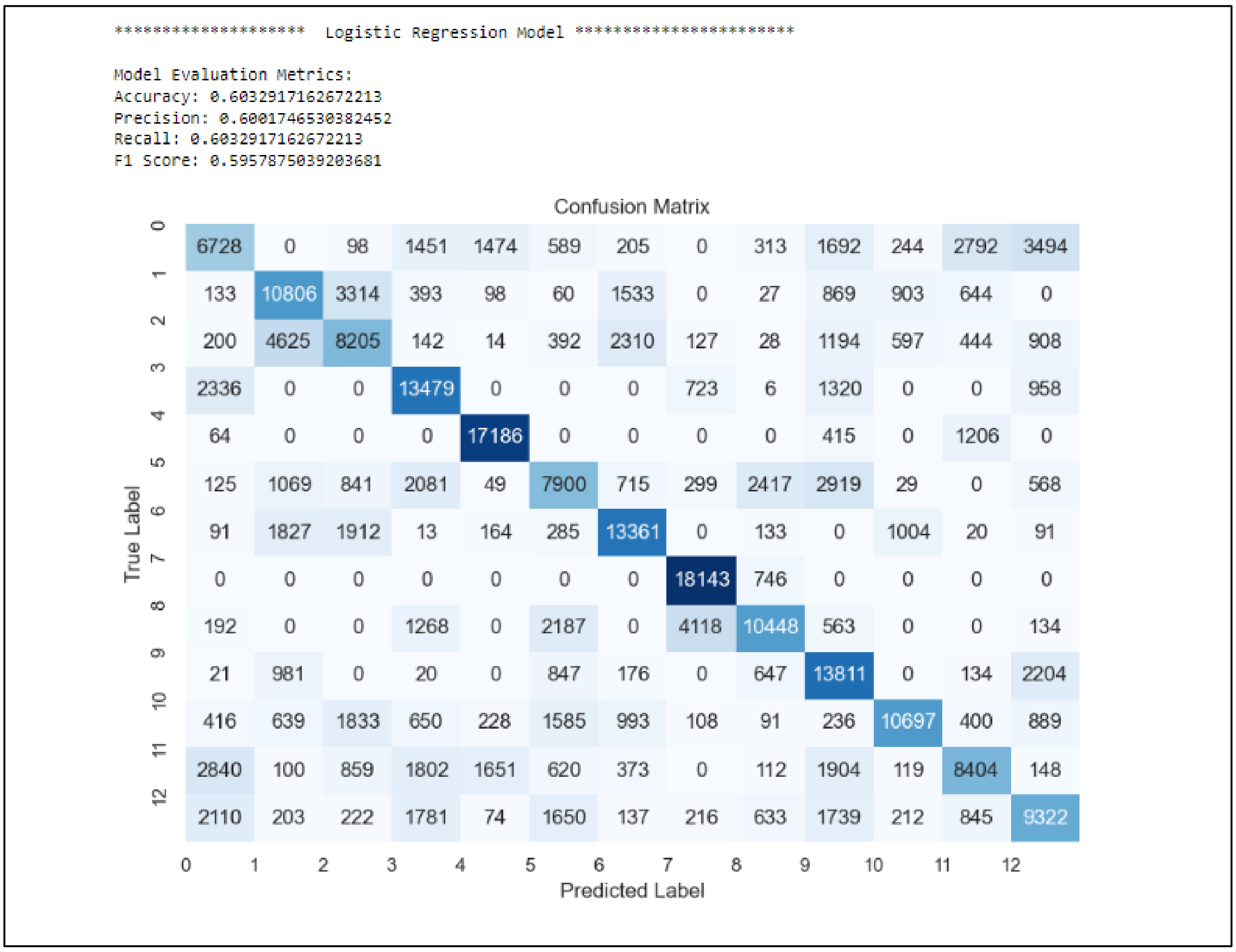
	orX	rY	rZ	accY	gY	gZ	mX	mY	mZ	lux	soundLevel	activity
0	125.0	-0.131696	-0.877469	2.681510	2.676550	8.642710	-31.200000	-35.600000	-37.6	5000.0	49.56	10
1	126.0	-0.131480	-0.878024	2.681510	2.678340	8.646540	-31.200000	-36.000000	-37.2	5000.0	53.38	10
2	127.0	-0.131551	-0.878799	2.681510	2.680040	8.650880	-31.200000	-36.000000	-37.2	5000.0	53.38	10
3	127.0	-0.131551	-0.878799	2.681510	2.680040	8.650880	-31.200000	-36.000000	-37.2	5000.0	49.53	10
4	127.0	-0.131888	-0.879645	2.681510	2.681300	8.654580	-31.200000	-35.600000	-36.8	5000.0	49.53	10
...	...	...	...	...	...	...	...	...	...	...	...	...
984524	261.0	0.046682	0.824703	1.475078	0.190198	10.113407	26.000000	-7.600000	-32.8	15000.0	53.48	12
984525	204.0	-0.046430	0.978232	-2.949660	-1.194923	10.549816	16.000000	-28.400000	-33.2	5.0	54.98	12
984526	29.0	0.113689	-0.481027	-0.689530	-0.248608	9.715687	3.600000	10.800000	-42.0	10000.0	62.68	12
984527	225.0	-0.032047	0.900473	-1.098358	-0.564718	8.949202	23.600000	-19.200000	-41.2	5000.0	58.90	12
984528	192.0	-0.018887	0.996879	-0.961871	-0.450180	10.373573	8.303648	-40.096352	-35.2	5000.0	56.48	12

984529 rows × 12 columns

```
# Check for NaN values in the 'activity' column
nan_values = selected_features_df['activity'].isna().sum()
print(f"Number of NaN values in 'activity' column: {nan_values}")
```

```
Number of NaN values in 'activity' column: 0
```

Table 4: *Quick View of the 11 selected features of the balanced data after using SMOTE technique.*



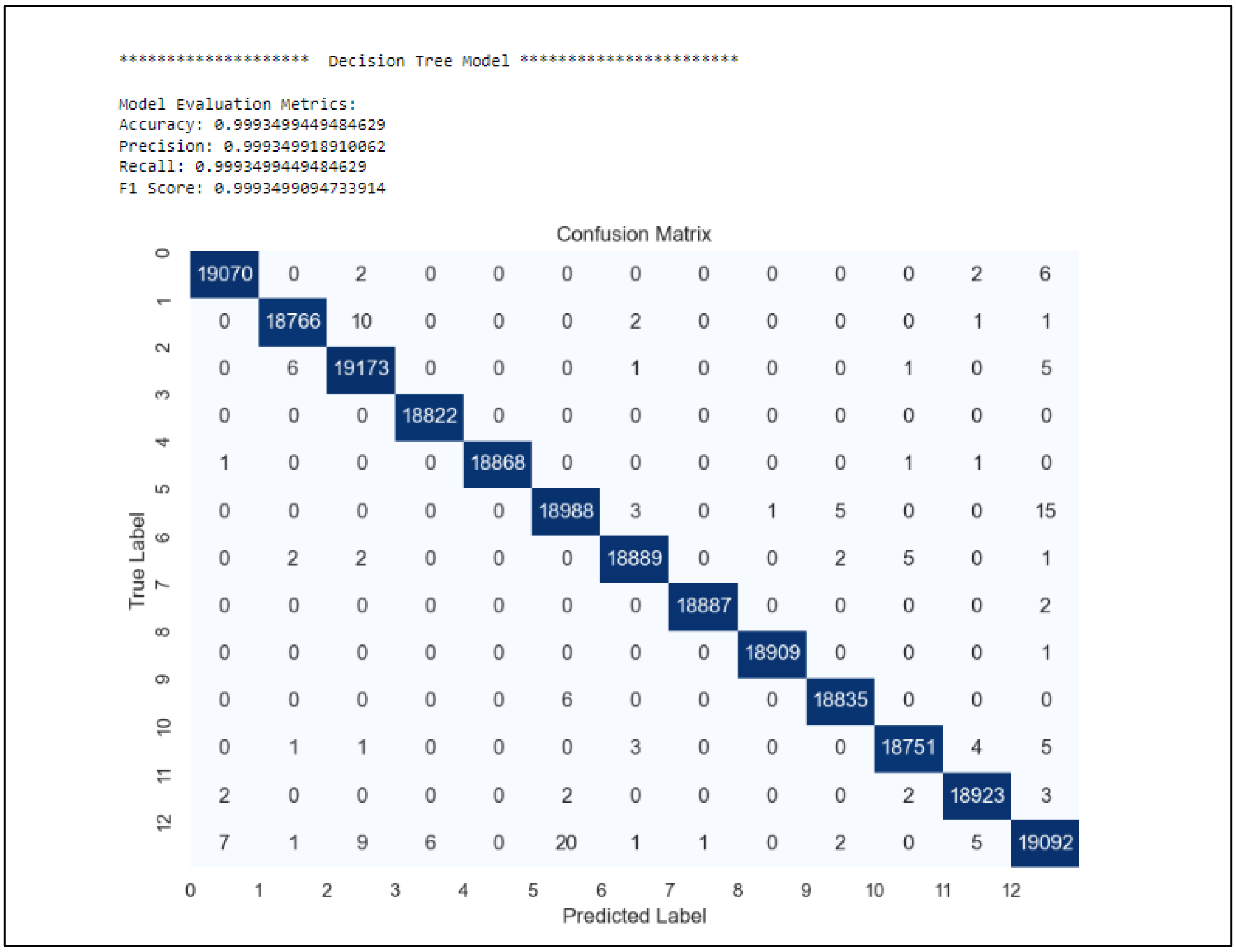
```
*******************  Logistic Regression Model **********************

Model Evaluation Metrics:
Accuracy: 0.6032917162672213
Precision: 0.6001746530382452
Recall: 0.6032917162672213
F1 Score: 0.5957875039203681
```

Confusion Matrix

True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9	10	11	12
0	6728	0	98	1451	1474	589	205	0	313	1692	244	2792	3494
1	133	10806	3314	393	98	60	1533	0	27	869	903	644	0
2	200	4625	8205	142	14	392	2310	127	28	1194	597	444	908
3	2336	0	0	13479	0	0	0	723	6	1320	0	0	958
4	64	0	0	0	17186	0	0	0	0	415	0	1206	0
5	125	1069	841	2081	49	7900	715	299	2417	2919	29	0	568
6	91	1827	1912	13	164	285	13361	0	133	0	1004	20	91
7	0	0	0	0	0	0	0	18143	746	0	0	0	0
8	192	0	0	1268	0	2187	0	4118	10448	563	0	0	134
9	21	981	0	20	0	847	176	0	647	13811	0	134	2204
10	416	639	1833	650	228	1585	993	108	91	236	10697	400	889
11	2840	100	859	1802	1651	620	373	0	112	1904	119	8404	148
12	2110	203	222	1781	74	1650	137	216	633	1739	212	845	9322

Plot 12: *Confusion Matrix of the trained Logistics Regression Classification Model*



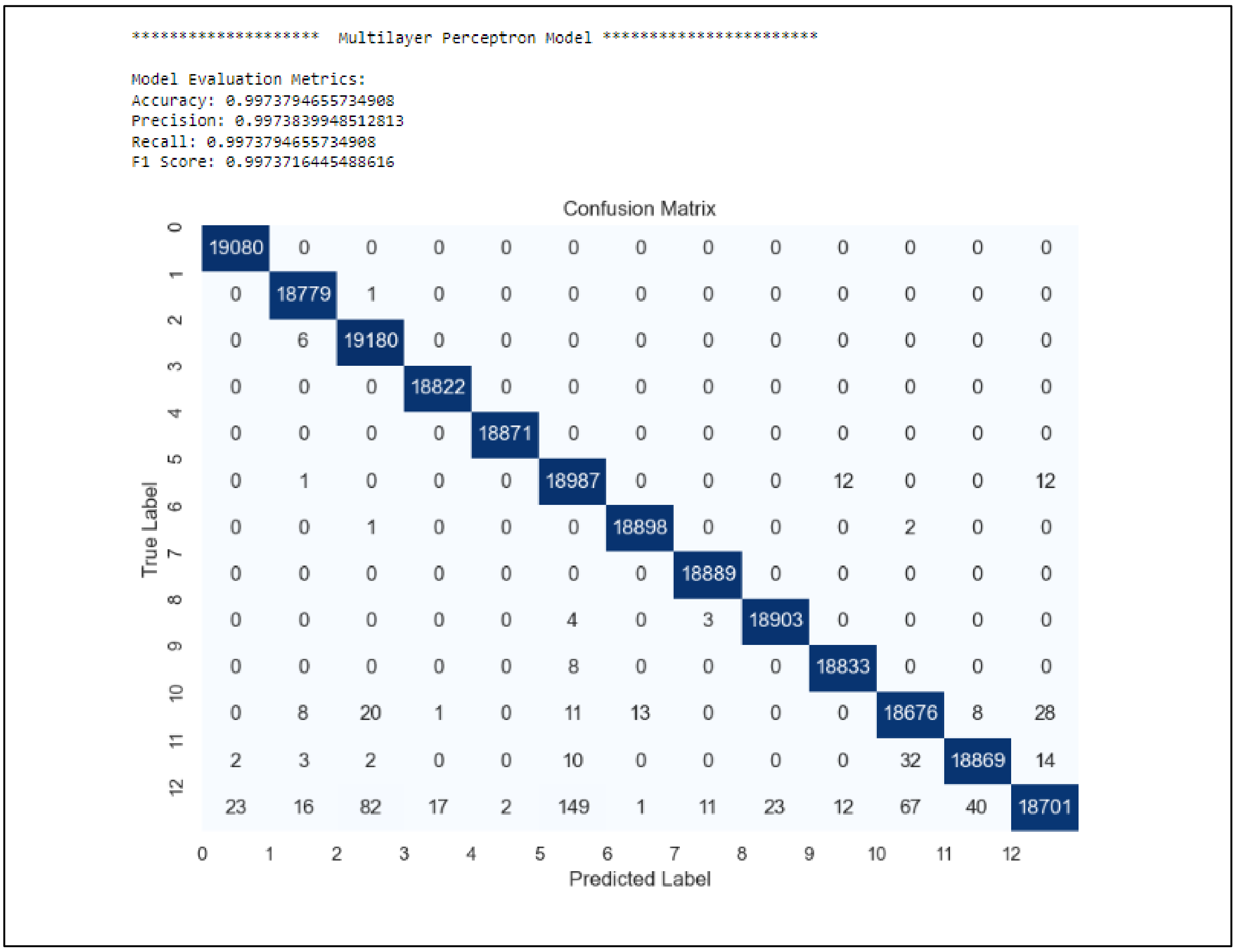
```
*******************  Decision Tree Model **********************

Model Evaluation Metrics:
Accuracy: 0.9993499449484629
Precision: 0.999349918910062
Recall: 0.9993499449484629
F1 Score: 0.9993499094733914
```

Confusion Matrix

True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19070	0	2	0	0	0	0	0	0	0	0	2	6
1	0	18766	10	0	0	0	2	0	0	0	0	1	1
2	0	6	19173	0	0	0	1	0	0	0	1	0	5
3	0	0	0	18822	0	0	0	0	0	0	0	0	0
4	1	0	0	0	18868	0	0	0	0	0	1	1	0
5	0	0	0	0	0	18988	3	0	1	5	0	0	15
6	0	2	2	0	0	0	18889	0	0	2	5	0	1
7	0	0	0	0	0	0	0	18887	0	0	0	0	2
8	0	0	0	0	0	0	0	0	18909	0	0	0	1
9	0	0	0	0	0	6	0	0	0	18835	0	0	0
10	0	1	1	0	0	0	3	0	0	0	18751	4	5
11	2	0	0	0	0	2	0	0	0	0	2	18923	3
12	7	1	9	6	0	20	1	1	0	2	0	5	19092

Plot 13: *Confusion Matrix of the trained Decision Tree Classification Model*



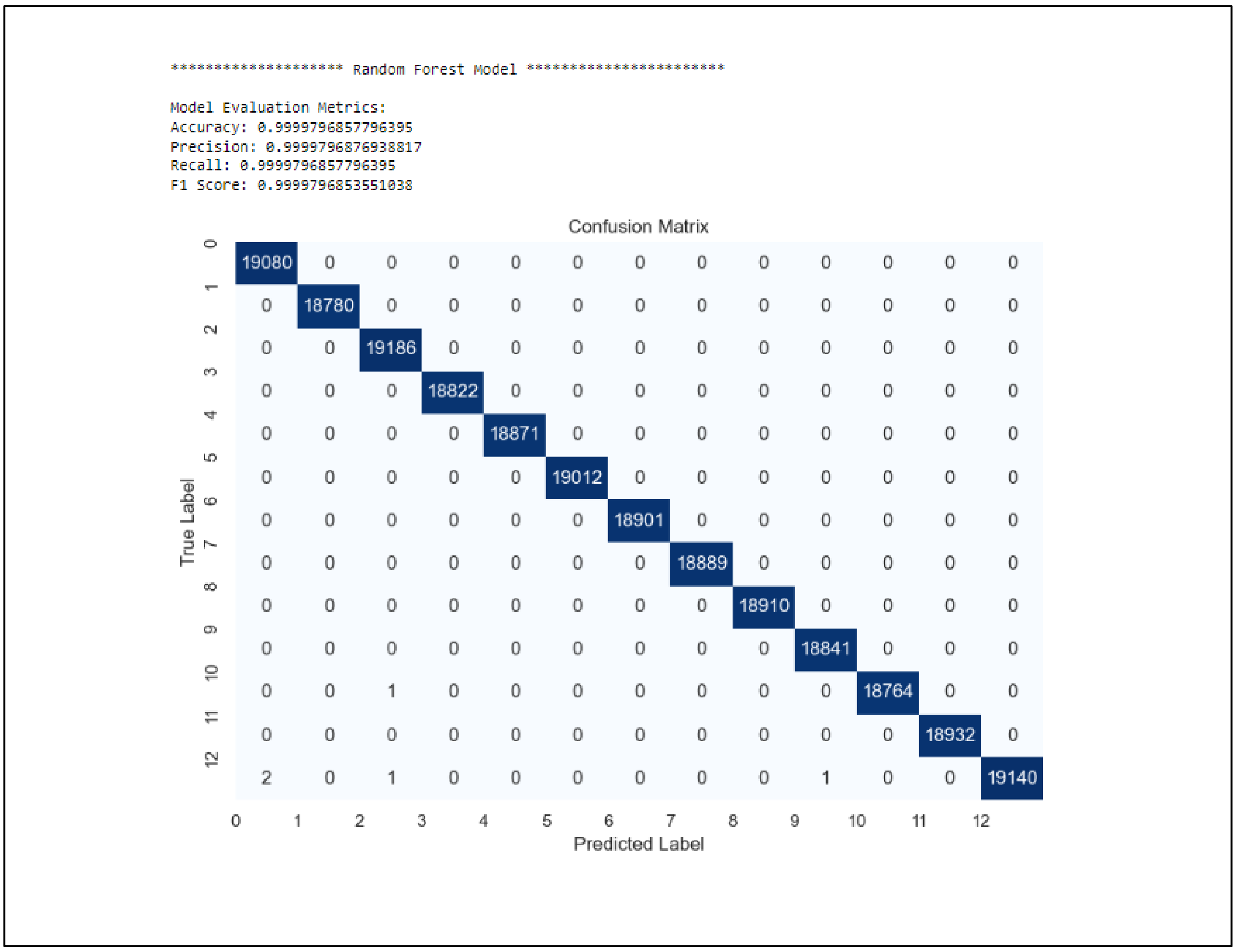
```
******************* Multilayer Perceptron Model **********************

Model Evaluation Metrics:
Accuracy: 0.9973794655734908
Precision: 0.9973839948512813
Recall: 0.9973794655734908
F1 Score: 0.9973716445488616
```

Confusion Matrix

True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19080	0	0	0	0	0	0	0	0	0	0	0	0
1	0	18779	1	0	0	0	0	0	0	0	0	0	0
2	0	6	19180	0	0	0	0	0	0	0	0	0	0
3	0	0	0	18822	0	0	0	0	0	0	0	0	0
4	0	0	0	0	18871	0	0	0	0	0	0	0	0
5	0	1	0	0	0	18987	0	0	0	12	0	0	12
6	0	0	1	0	0	0	18898	0	0	0	2	0	0
7	0	0	0	0	0	0	0	18889	0	0	0	0	0
8	0	0	0	0	0	4	0	3	18903	0	0	0	0
9	0	0	0	0	0	8	0	0	0	18833	0	0	0
10	0	8	20	1	0	11	13	0	0	0	18676	8	28
11	2	3	2	0	0	10	0	0	0	0	32	18869	14
12	23	16	82	17	2	149	1	11	23	12	67	40	18701

Plot 14: *Confusion Matrix of the trained Multilayer Perceptron Classification Model*



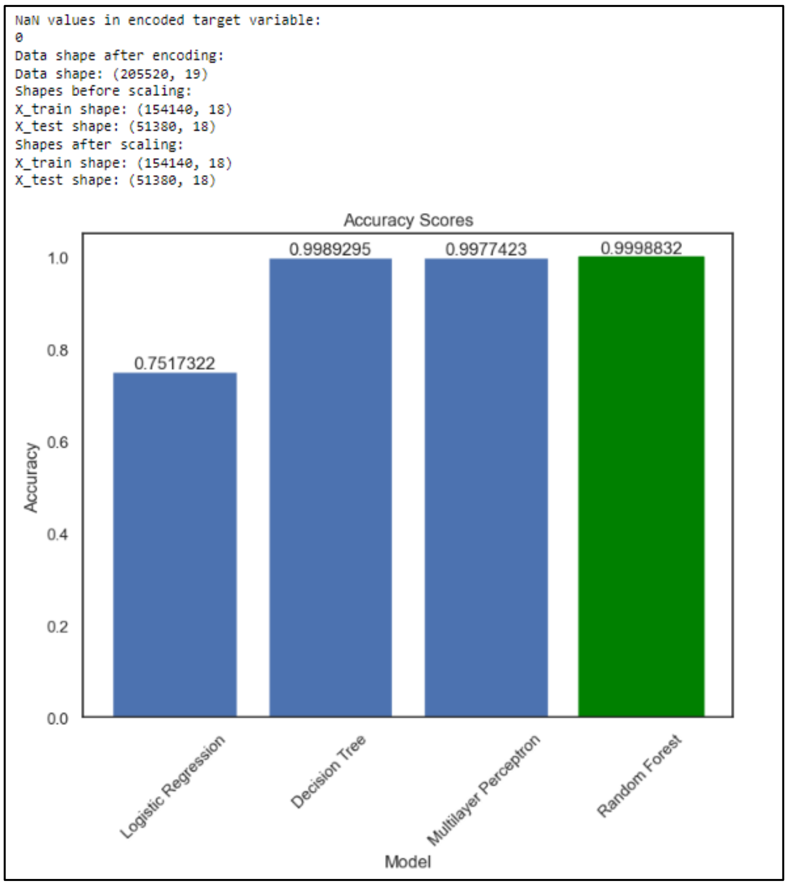
```
******************* Random Forest Model **********************

Model Evaluation Metrics:
Accuracy: 0.9999796857796395
Precision: 0.9999796876938817
Recall: 0.9999796857796395
F1 Score: 0.9999796853551038
```

Confusion Matrix

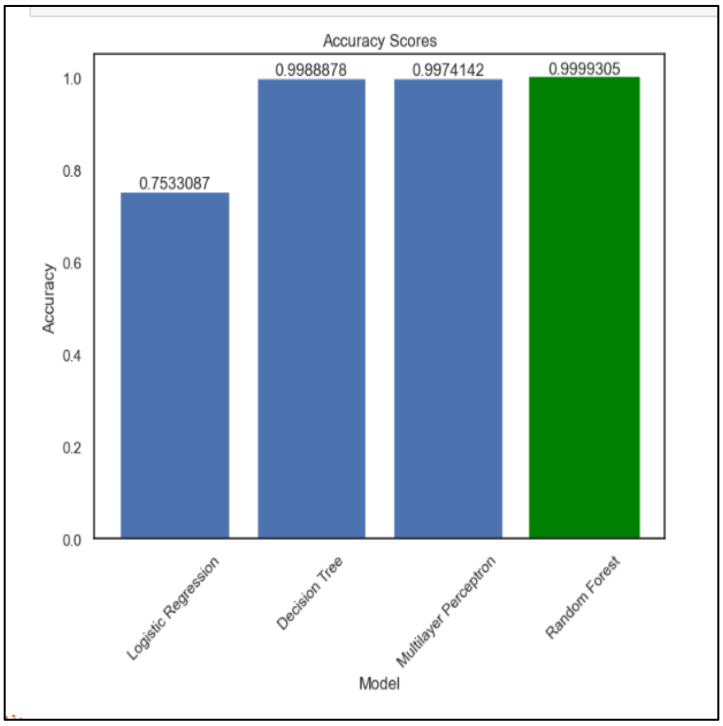
True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19080	0	0	0	0	0	0	0	0	0	0	0	0
1	0	18780	0	0	0	0	0	0	0	0	0	0	0
2	0	0	19186	0	0	0	0	0	0	0	0	0	0
3	0	0	0	18822	0	0	0	0	0	0	0	0	0
4	0	0	0	0	18871	0	0	0	0	0	0	0	0
5	0	0	0	0	0	19012	0	0	0	0	0	0	0
6	0	0	0	0	0	0	18901	0	0	0	0	0	0
7	0	0	0	0	0	0	0	18889	0	0	0	0	0
8	0	0	0	0	0	0	0	0	18910	0	0	0	0
9	0	0	0	0	0	0	0	0	0	18841	0	0	0
10	0	0	1	0	0	0	0	0	0	0	18764	0	0
11	0	0	0	0	0	0	0	0	0	0	0	18932	0
12	2	0	1	0	0	0	0	0	0	1	0	0	19140

Plot 15: *Confusion Matrix of the trained Random Forest Classification Model*

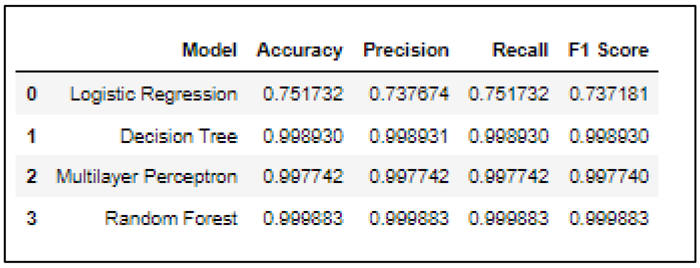


```
NaN values in encoded target variable:
0
Data shape after encoding:
Data shape: (205520, 19)
Shapes before scaling:
X_train shape: (154140, 18)
X_test shape: (51380, 18)
Shapes after scaling:
X_train shape: (154140, 18)
X_test shape: (51380, 18)
```

Plot 16: *Bar chart showing Accuracy score of the 4 classification Models of the data without winsorization.*

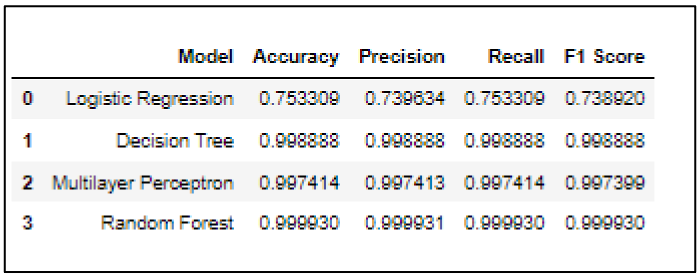


Plot 17: *Bar chart showing Accuracy score of the 4 classification Models of the data after winsorization.*



	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.751732	0.737674	0.751732	0.737181
1	Decision Tree	0.998930	0.998931	0.998930	0.998930
2	Multilayer Perceptron	0.997742	0.997742	0.997742	0.997740
3	Random Forest	0.999883	0.999883	0.999883	0.999883

Table 5: *Performance metrics for classification models without winsorization.*



	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.753309	0.739634	0.753309	0.738920
1	Decision Tree	0.998888	0.998888	0.998888	0.998888
2	Multilayer Perceptron	0.997414	0.997413	0.997414	0.997399
3	Random Forest	0.999930	0.999931	0.999930	0.999930

Table 6: *Performance metrics for classification models after winsorization.*

APPENDIX B- PROGRAM PSEUDOCODES

CUSTOM STATISTICS MODULE

```
1. Import necessary libraries: pandas, numpy, seaborn, StandardScaler from sklearn. preprocessing,
   train_test_split from sklearn.model_selection
2. Define a class named ExploratoryDataAnalysis:
   • Initialize the class:
       - Set the data attribute to None
   • Define a static method named load_data(file_path):
       - Try to read the CSV file at the given file_path using pandas read_csv() function
       - If the file is found, return the data
       - If FileNotFoundError occurs, print an error message and return None.
   • Define a method named load(self, path: str):
       - Load the data set from the CSV file at the given path using pandas read_csv() function
       - Assign the loaded data to the data attribute of the class
   • Define a method named clean(self):
       - Drop rows with missing values from the data using dropna() function
       - Remove duplicate values from the data using drop_duplicates() function
       - Extract specific numeric columns from the data
   • Define a private method named _remove_duplicate(self):
       - Remove duplicate rows from the data using drop_duplicates() function, excluding the
         '_Id' column
   • Define a private method named _extract_numeric(self):
       - Create a list of desired numeric column names
       - Select only the numeric columns from the data based on the column list
   • Define a method named statistics(self):
       - Select the numeric columns from the data
       - Calculate various statistics (mean, median, standard deviation, minimum, Q1, Q2, Q3, IQR,
         maximum,
       - variance, skewness, and kurtosis) using appropriate pandas functions
       - Return the DataFrame
3. End of the class definition
```

ABOUT THE DATA

```
1. Import necessary libraries: pandas, qgrid
2. Define a function named load_data(file_path):
   - Try to read the CSV file at the given file_path using pandas read_csv() function
   - If the file is found, return the data
   - If FileNotFoundError occurs, print an error message and return None
3. Define a variable named file_path and assign the path to the activity dataset
4. Call the load_data() function with the file_path to load the activity dataset
5. Display the first 10 rows of the dataset using the head() function
6. Get the number of entries in the dataset using the shape attribute
```

7. Display information about the dataset using the info() function
8. Convert the data into a DataFrame using the pd.DataFrame() function
9. Display the DataFrame
10. Make the DataFrame more interactive using the qgrid.show_grid() function and assign it to qgrid_widget
11. Count the occurrences of each activity in the 'activity' column using the value_counts() function
12. Check for missing values in the dataset using the isnull().sum() function
13. Print unique values of the 'activity' column using the unique().tolist() functions
14. Count the number of unique activities in the 'activity' column using the nunique() function

DESCRIPTIVE STATISTICS AND DATA CLEANING

1. Import necessary libraries and modules:
   - pandas
   - dataframe_image
   - eda_activity_monitor
2. Load the activity dataset into a variable 'data'.
3. Create an instance of the ExploratoryDataAnalysis class.
4. Assign the 'data' to the 'data' attribute of the ExploratoryDataAnalysis instance.
5. Clean the data by calling the 'clean()' method of the ExploratoryDataAnalysis instance.
6. Perform statistical analysis by calling the 'statistics()' method of the ExploratoryDataAnalysis instance.
7. Export the statistics DataFrame as an image using the dataframe_image.export() function.
8. Display the statistics DataFrame.
9. Define a function named 'check_duplicates' that takes 'data' as input:
   - Check for duplicates in the 'data' using the duplicated() function.
   - If duplicates are found, print the duplicated rows.
   - If no duplicates are found, print a message indicating the absence of duplicates.
10. Define a function named 'check_outliers' that takes 'data', 'columns', and 'threshold' as inputs:
    - Calculate the z-scores of the specified 'columns' in the 'data' using the mean() and std() functions.
    - Identify outliers by comparing the absolute z-scores to the 'threshold'.
    - Create a DataFrame 'outliers_df' containing the rows with outliers.
    - Return the 'outliers_df'.
11. Call the 'check_duplicates' function with 'data' as input and print the results.
12. Call the 'check_outliers' function with 'data', ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', 'mZ'] as inputs and assign the result to 'outliers'.
13. If 'outliers' is not empty, print the outliers.

DATA VISUALIZATION

Step 1: Create a box plot for each column containing outliers

outlier_columns = ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', 'mZ']

Create a figure with a suitable size

For each column in outlier_columns:

  Create a box plot of the column values

  Set the x-axis label to the column name

```
    Show the plot
```

#Step 2: Define a function named 'plot_frequencies' to plot the frequency of variables

```
Define a function named 'plot_frequencies' that takes 'interest_vars' as a parameter:
    Create subplots for the number of variables in 'interest_vars'
    For each column in 'interest_vars':
        Split the data into equal sections
        Calculate the frequencies for each section
        Plot a bar chart of the frequencies in the corresponding subplot
```

Step 3: Call the 'plot_frequencies' function with the specified variables

```
Call the 'plot_frequencies' function with the list of variables ['accX', 'accY', 'accZ', 'gX', 'gY', 'gZ', 'mX', 'mY', 'mZ']
```

Step 4: Define a function named 'graphs' to plot various graphs

```
Define a function named 'graphs' that takes 'data' as a parameter:
    Plot bar charts, pie charts, stacked bar charts, histograms, etc., using the 'data' dataframe
```

Step 5: Call the 'graphs' function with the 'data' dataframe

```
Call the 'graphs' function with the 'data' dataframe
```

DATA CORRELATION

Step 1: Define a function named 'correlation' to calculate and plot the correlation matrix

```
Define a function named 'correlation' that takes 'data' as a parameter:
    Try the following block of code:
        - Convert the 'activity' column to numbers using LabelEncoder
        - Exclude the '_id' column from correlation calculation
        - Set up the seaborn theme
        - Compute the correlation matrix of the remaining columns
        - Generate a mask for the upper triangle of the correlation matrix
        - Set up the matplotlib figure
        - Generate a custom colormap for the heatmap
        - Draw the heatmap using Seaborn's heatmap function, with the mask applied
        - Set the title and axis labels of the plot
    Except if any of the specified exceptions occur:
        - Print an error message indicating the occurrence of the error
```

Step 2: Call the 'correlation' function with the 'data' dataframe

Call the 'correlation' function with the 'data' dataframe

CLASS BALANCING

Step 1: Import the necessary libraries

Import the 'OrderedDict' class from the 'collections' module

Step 2: Define a function named 'balance' to balance the dataset using SMOTE and Random Oversampling

```
Define a function named 'balance' that takes 'file' as a parameter:
  Try the following block of code:
    - Separate the features (X) and target variable (y) from the 'file' dataframe
    - Define a dictionary mapping activity codes to activity names
    - Balance the dataset using SMOTE (Synthetic Minority Over-sampling Technique)
      - Instantiate an SMOTE object with a random state
      - Resample the dataset using SMOTE and store the results in new variables
      - Concatenate the resampled features and target variable into a new dataframe
      - Plot the frequency of activities in the resampled dataset using a bar chart
    - Balance the dataset using Random Oversampling
      - Instantiate a RandomOverSampler object with a random state
      - Resample the dataset using Random Oversampling and store the results in new variables
      - Concatenate the resampled features and target variable into a new dataframe
      - Plot the frequency of activities in the resampled dataset using a pie chart
    - Return the resampled datasets (resampled_sm, resampled_ros)
  Except if any of the specified exceptions occur:
    - Print an error message indicating the occurrence of the error
    - Return None
```

Step 3: Call the 'balance' function with the 'data' dataframe

Call the 'balance' function with the 'data' dataframe and assign the returned values to 'balanced_smote' and 'balanced_ros'

DATA SPLITTING

Step 1: Import the necessary libraries

```
    Import the 'LabelEncoder' class from the 'sklearn.preprocessing' module
    Import the 'train_test_split' function from the 'sklearn.model_selection' module
    Import the 'StandardScaler' class from the 'sklearn.preprocessing' module
```

Step 2: Define a function named 'split_data_without_leakage' to split the data without leakage

Define a function named 'split_data_without_leakage' that takes 'data' as a parameter:

Try the following block of code:

- Encode the target variable using the LabelEncoder
- Print the number of NaN values in the encoded target variable
- Print the shape of the data after encoding
- Split the data into features (X) and target variable (y)
- Split the data into training and testing sets using train_test_split
- Print the shapes of the datasets before scaling
- Scale the training and testing datasets using the StandardScaler
- Print the shapes of the datasets after scaling
- Return the split datasets (X_train, X_test, y_train, y_test)

Except if any of the specified exceptions occur:

- Print an error message indicating the occurrence of the error
- Return None

Step 3: Call the 'split_data_without_leakage' function with the 'data' dataframe

Call the 'split_data_without_leakage' function with the 'data' dataframe and assign the returned values to 'X_train', 'X_test', 'y_train', 'y_test'

FEATURE SELECTION

Step 1: Import the necessary libraries

Import the 'SMOTE' class from the 'imblearn.over_sampling' module

Import the 'LabelEncoder' class from the 'sklearn.preprocessing' module

Import the 'SelectKBest' class from the 'sklearn.feature_selection' module

Import the 'f_classif' function from the 'sklearn.feature_selection' module

Step 2: Define a function named 'feature_selection_balanced' for feature selection on balanced data

Define a function named 'feature_selection_balanced' that takes 'data' as a parameter:

Try the following block of code:

- Balance the data using SMOTE
- Split the features (X) and target variable (y)
- Encode the target variable using the LabelEncoder
- Perform feature selection on the balanced data using SelectKBest
- Get the names of the selected features
- Create a DataFrame of the selected features

- Add the encoded 'activity' column back to the selected features DataFrame

- Return the selected features DataFrame

Except if any of the specified exceptions occur:

- Print an error message indicating the occurrence of the error

- Return None

Step 3: Call the 'feature_selection_balanced' function with the 'data' dataframe

Call the 'feature_selection_balanced' function with the 'data' dataframe and assign the returned DataFrame to a variable

ACTIVITY CLASS

1. Import the necessary libraries and modules.
2. Define a function to balance the data using SMOTE and Random Oversampling techniques.
3. Define a function to split the data into training and testing sets without leakage.
4. Define a function for feature selection using SelectKBest.
5. Define a function to plot the confusion matrix.
6. Define a function for logistic regression:
    - Split the data using the **split_data_without_leakage** function.
    - Train a logistic regression model on the training data.
    - Make predictions on the test data.
    - Evaluate the model using accuracy, precision, recall, and F1 score.
    - Plot the confusion matrix using the **plot_confusion_matrix** function.
7. Define a function for the decision tree:
    - Split the data using the **split_data_without_leakage** function.
    - Train a decision tree model on the training data.
    - Make predictions on the test data.
    - Evaluate the model using accuracy, precision, recall, and F1 score.
    - Plot the confusion matrix using the **plot_confusion_matrix** function.
8. Define a function for the multilayer perceptron (MLP):
    - Split the data using the **split_data_without_leakage** function.
    - Train an MLP model on the training data.
    - Make predictions on the test data.
    - Evaluate the model using accuracy, precision, recall, and F1 score.
    - Plot the confusion matrix using the **plot_confusion_matrix** function.
9. Define a function for random forest:

- Split the data using the **split_data_without_leakage** function.
- Train a random forest model on the training data.
- Make predictions on the test data.
- Evaluate the model using accuracy, precision, recall, and F1 score.
- Plot the confusion matrix using the **plot_confusion_matrix** function.

10. Call the necessary functions in the appropriate order to perform the desired tasks:
    - Balance the data using the **balance** function.
    - Perform feature selection using the **feature_selection_balanced** function.
    - Call each classification model function (**logistic_regression**, **decision_tree**, **mlp**, **random_forest**) passing the selected features as input.
    - Display the evaluation metrics and confusion matrix for each model.

MODEL COMPARISON

1. Define a dictionary **models** with the model names as keys and corresponding model instances as values.
2. Initialize empty lists to store the performance metrics: **accuracy_scores**, **precision_scores**, **recall_scores**, and **f1_scores**.
3. Iterate over the models using a **for** loop with variables **model_name** and **model**:
    - Fit the model using the training data (**X_train** and **y_train**).
    - Make predictions on the test data (**X_test**).
    - Calculate the accuracy, precision, recall, and F1 score using the predicted labels and the true labels.
    - Append the scores to the respective lists.
4. Create a DataFrame **performance_df** to store the results:
    - Use the model names, accuracy scores, precision scores, recall scores, and F1 scores to create the DataFrame.
5. Plot a bar chart to visualize the accuracy scores:
    - Set the figure size.
    - Find the maximum accuracy value.
    - Plot the bar chart using **plt.bar**.
    - Set a different color for the bar with the highest accuracy.
    - Set the title, x-label, y-label, and rotate the x-axis labels.
    - Add labels to the bars using **plt.text**.
    - Display the plot using **plt.show()**.
6. Display the **performance_df** DataFrame.

OPTIONAL STATISTICS

1. Import the necessary libraries: **pandas** and **numpy**.
2. Define the function **sliding_window_statistics** that takes the **data** and **window_size** as input parameters.
3. Calculate the number of sliding windows: **num_windows = len(data) - window_size + 1**.
4. Initialize an empty dictionary **sliding_statistics** to store the sliding window statistics.
5. Iterate over each column in the **data** DataFrame:
    - Initialize empty lists for each statistic: **variance**, **median**, **mean**, **std**, **rms**, **zero_crossing**, **sum_squares**, and **covariance**.
    - Iterate over the sliding windows using a **for** loop with index **i** from 0 to **num_windows**:
        - Extract the data within the window: **window_data = data[column].iloc[i:i+window_size]**.
        - Calculate the statistics for the window using NumPy functions: **np.var**, **np.median**, **np.mean**, **np.std**, **np.sqrt**, **np.sum**, and **np.cov**.
        - Append the computed statistics to their respective lists.
    - Store the computed statistics lists in the **sliding_statistics** dictionary with keys that include the column name and the statistic name.
6. Convert the **sliding_statistics** dictionary to a DataFrame **sliding_statistics_df** using **pd.DataFrame**.
7. Return the **sliding_statistics_df** DataFrame from the function.
8. Load the dataset from the CSV file into the **data** DataFrame using **pd.read_csv**.
9. Drop the 'activity' column from the **data** DataFrame using **data.drop('activity', axis=1)**.
10. Specify the sliding window size (**window_size**).
11. Compute the sliding window statistics by calling the **sliding_window_statistics** function with the **data** and **window_size** as arguments, and store the result in the **sliding_statistics_df** DataFrame.
12. Print the computed statistics using **print(sliding_statistics_df)**.