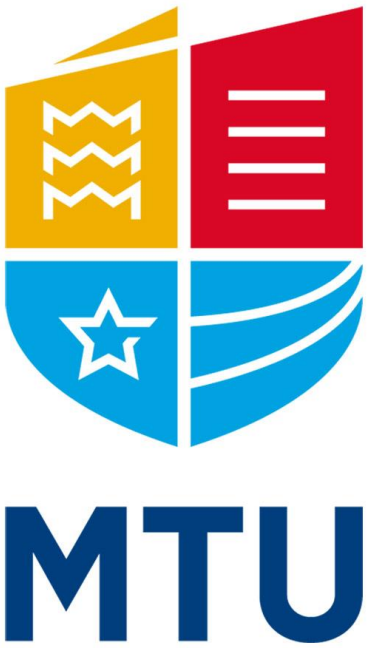


The classification of mental illness using motor activity data.

John Fitzgerald

Student No: R00156081

Supervisor: Aengus Daly

Industry Mentor: n/a

For the module DATA8006 – Data Science Analytics Project

as part of the Higher Diploma of Science in Data Science and Analytics, Department of Mathematics, May 2023

Declaration of Authorship

I, John Fitzgerald, declare that this thesis titled '**The classification of mental illness using motor activity data**' and the work presented in it are my own. I confirm that,

- This work was done wholly or mainly while in candidature for the Higher Diploma at Munster Technological University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Munster Technological University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work;
- I have acknowledged all main sources of help;
- I understand that my project documentation may be stored in the library at MTU, and may be referenced by others in the future.

Signed: *John Fitzgerald*

Date: *15/04/2023*

Table of Contents

List of Figures

List of Tables

Abstract

Depression and schizophrenia are prevalent mental health disorders that affect millions of people worldwide. The World Health Organization (WHO) reports that depression is a leading cause of disability globally, with an estimated 264 million people affected in 2020. Schizophrenia is a chronic and severe mental disorder that affects approximately 20 million people worldwide. According to a report by the Centers for Disease Control and Prevention (CDC), the COVID-19 pandemic led to increased levels of anxiety, depression, and stress in many individuals. A study published in The Lancet Psychiatry also found that COVID-19 lockdowns were associated with increased rates of depression and anxiety in some countries. Accurate and timely diagnosis of these disorders is crucial for effective treatment and management. However, traditional diagnostic methods can be challenging and may not always be sufficient.

Recent studies suggest that individuals with behavioural disorders such as depression and schizophrenia exhibit lower levels of mobility compared to healthy individuals. To address this issue, this thesis focuses on the classification of mental illness using motor activity data collected from wrist sensors worn by depressive and schizophrenic patients, as well as healthy patients. The study utilized public datasets containing activity readings recorded every minute over a period of approximately two weeks.

Machine learning algorithms were employed to analyse the data and assess classification accuracy. The best model was Radom Forest Classifier (RFC), which recorded 74% accuracy for multiclass classification. Results indicate that motor activity data can be a useful tool in classifying mental illness and have implications for the development of non-invasive diagnostic tools for mental health disorders. The findings of this thesis suggest that motor activity data may provide useful insights into the mobility patterns of individuals with behavioural disorders and could be used as a diagnostic tool to aid in the identification of depression and schizophrenia.

1. Introduction

1.1 Background

Mental illnesses are a leading cause of disability worldwide, affecting millions of people each year. According to the World Health Organization (WHO), approximately one in four people globally will experience some form of mental illness in their lifetime, with depression and anxiety being the most prevalent disorders[1]. Mental illnesses can cause significant distress and impairment in social, occupational, and other areas of functioning. Early detection and diagnosis of mental illness are crucial for effective treatment and management of symptoms.

Major Depressive Disorder (MDD) is the most common type of depression[2]. Patients diagnosed with MDD can show symptoms such as melancholy, hopelessness, low energy, sleep disturbance, and interference of ordinary activities such as work and study. Schizophrenia is a more complex mental disorder that can produce delusions or hallucinations in sufferers. Depression and schizophrenic can share symptoms as many cases of schizophrenic patients suffer from depression simultaneously and unipolar depressives can tend towards psychotic episodes.

Currently, the diagnosis of mental illness relies on clinical interviews, symptom reports, and observational measures. However, these methods have limitations, including subjectivity, reliance on self-report, and lack of objectivity. Additionally, the process of diagnosis can be time-consuming and costly, and misdiagnosis is not uncommon.

The development of new techniques to improve precision in diagnosing mental illness has become a priority[3]. In psychiatry, the lack of measurement and patient monitoring had led to the development of precision psychiatry. This involves techniques such as speech analysis, activity monitoring, facial expression analysis, heart rate and other techniques. Machine Learning is a promising tool that will analyse and detect patterns in data obtained from psychiatric patients. This is an emerging field of study.

Motor activity data, collected through wearable devices such as accelerometers or actigraphy, has emerged as a potential biomarker for mental illness. These devices provide continuous monitoring of physical activity, movement, and sleep patterns, which can reflect changes in mental health status. Studies show that individuals with mental illness have altered motor activity patterns compared to healthy controls, with reduced physical activity, disturbed sleep, and abnormal movements being reported in several conditions [4] [5].

The potential use of motor activity data for the diagnosis and classification of mental illness is a relatively new area of research, but one that has garnered significant interest. Several studies have shown promising results in the use of motor activity data for the prediction of mood disorders, schizophrenia, and other mental illnesses [6] [7] . However, further research is needed to establish the validity and reliability of this approach and to develop accurate algorithms for classification.

1.2 Aims

The aim of this thesis is to investigate the potential of using motor activity data for the classification of mental illness. The thesis will explore the use of machine learning algorithms to analyse motor activity data collected from wearable devices and to develop models for the classification of mental illness. Specifically, the thesis will address the following research questions:

- Can motor activity data collected from wearable devices be used to accurately classify individuals with mental illness?
- Which machine learning algorithms are most effective for the classification of mental illness using motor activity data?

To answer these research questions, the thesis will conduct a systematic review of the literature on the use of motor activity data for the classification of mental illness. The review will examine the current state of research in this area, identify gaps in knowledge, and provide a foundation for the subsequent research. Following the review, the thesis will use motor activity data from individuals with mental illness and healthy controls, using wearable devices such as accelerometers or actigraphy. The data will be analysed using machine learning algorithms, to develop classification models for different mental illnesses. The performance of these models will be evaluated using metrics such as sensitivity, specificity, accuracy, and Area Under receiver operator Curve (AUC).

The findings of this thesis will contribute to the growing body of research on the use of motor activity data for the classification of mental illness. The thesis will provide insights into the potential of this approach and its limitations, as well as identifying areas for future research. The thesis will also have practical implications for the development of diagnostic tools for mental illness, which could improve the accuracy and efficiency of diagnosis and treatment. Overall, this thesis aims to advance our understanding of the relationship between motor activity and mental illness and to provide a foundation for the development of novel and effective diagnostic tools.

2. Literature Review

In this chapter, we review relevant literature related to the classification of mental illness using motor activity data. We discuss four studies that have used motor activity data to classify depressive and schizophrenic episodes.

The studies described in 2.2 and 2.3 below are baseline studies for this topic. The aim of both papers is to provide the datasets and perform baseline analysis to enable further research. These papers do not directly deal with motor activity data or severity estimation.

The studies described in 2.4 and 2.5 focus on developing machine learning models for the classification and severity estimation of behavioural disorders using motor activity data. This is also the primary focus of this paper.

2.1 Dataset description

Two independent datasets published by the same research group [8] [9] are used in the all the projects reviewed below. These are:

> The '*Psychose*' dataset which comprises of 22 subjects diagnosed with Schizophrenia and 32 healthy patients. This dataset was intended for use as a resource for researchers studying schizophrenia and other psychotic disorders. The authors suggest that the database could be used to develop new algorithms for detecting and monitoring psychosis, as well as for identifying patterns in motor activity data that are associated with different types of psychotic episodes.
>
> The '*Depresjon*' dataset which consists of 23 subjects suffering from a depressive disorder and 32 healthy patients. This dataset was also intended to help researchers to develop systems capable of automatically detecting depression states based on sensor data.

The healthy group of 32 that are part of both datasets consist of the same group of individuals, with the same readings.

The depression groups are labelled as 'Condition' in the dataset, the schizophrenic group are labelled 'Patient' and the healthy group are described as 'Control'. Table 1 shows the description of each group and the number of participants.

Table 1 Dataset description

Grouping	Description	No. participants
Condition	Patients diagnosed with depression.	23
Patient	Patients diagnosed with schizophrenia.	22
Control	Healthy group, not diagnosed with any psychological disorder.	32
	Total participants:	**77**

The data collected from the actigraphy watches was continuously recorded in one-minute intervals and were stored as activity counts. The three datasets comprise of a .csv file for each patient containing the actigraphy data with a date and timestamp. The data collection process is described elsewhere [10]. Most of the participants have been recorded over 13 days. Some of the participants have less than 13 days and some have shared more. For this study, the length (in days) of the participation was not a particular issue. It was important for the dataset to have complete 24-hour readings to use (1440 readings per day) for as many of the days as possible to ensure feature integrity.

The datasets also contain general demographic data information including age and gender, and clinical information such as the Montgomery-Asberg Depression Rating Scale (MADRS) score [11] and the Brief Psychiatric Rating Scale (BPRS) score [12]. These scores represent the severity levels of depression and schizophrenia respectively. A higher score generally represents a higher intensity of illness. This information was useful to us comparing activity levels at an individual level – to compare individuals with similar demographics and illness levels to see if mobility patterns are distinguishable between a particular patient with schizophrenia, a person with a depressive condition and a person of similar age and gender from the control group.

2.2 'Depresjon: A Motor Activity Database of Depression Episodes in Unipolar and Bipolar Patients'

The paper "Depresjon: A Motor Activity Database of Depression Episodes in Unipolar and Bipolar Patients" [8] uses the dataset containing sensor data collected from both patients suffering from depression and the healthy (control) group.

The patients' depressive state was labelled using ratings done by medical experts on the MADRS. This scoring consists of ten items that assess various symptoms of depression, including mood, anxiety, sleep, appetite, and concentration. The scores for each item are added together to obtain a total score, which ranges from 0 to 60. A higher score indicates greater severity of depression symptoms.

The paper evaluated algorithms include Nearest Neighbors, Linear SVM, RBF SVM, Gaussian Process, Decision Tree, Random Forest, Neural Net, AdaBoost, Naive Bayes, and QDA. The evaluation was performed using 10-fold cross-validation and the performance metrics presented in the table include Precision, Recall, Accuracy, Specificity, Matthews Correlation Coefficient (MCC), and F1-score.

The results indicate that Linear SVM obtained the best overall weighted recall, accuracy, MCC, and F1-score. In general, all methods tested performed better than the ZeroR baseline (majority class) in terms of classification performance. However, the performance of the classifiers is still moderate, with the highest weighted average F1-score being 78% for the Linear SVM classifier.

2.3 'PSYKOSE: A Motor Activity Database of Patients with Schizophrenia'

The paper PSYKOSE: A Motor Activity Database of Patients with Schizophrenia [9] is the source study of the Schizophrenia data that will be used in this project. The authors describe the design and implementation of the PSYKOSE database, including the data collection process and the pre-processing steps used to clean and normalize the data. They also provide some initial analysis of the data, including a comparison of the motor activity patterns of patients during psychotic and non-psychotic episodes.
The paper demonstrates that patients with schizophrenia exhibited significantly lower levels of physical activity and spent more time sedentary compared to healthy controls. Furthermore, they found that patients with schizophrenia had significantly higher variability in their physical activity levels than healthy controls.

Four different machine learning algorithms to classify patients - Logistic Regression, Random Forest, Extreme Gradient Boosting, and Light Gradient Boosting, with an ensemble method used to combine them. The metrics used to evaluate their performance were average precision and area under the curve. The testing was stratified to balance the number of schizophrenic and non-schizophrenic data points. The source code '*schizophrenia_baseline_experiment.py*' that produced the results was made available.

The results show that all four algorithms performed well with average precision and AUC of 80%. Logistic regression performed best in terms of average precision and area under the curve. The precision-recall curve for Logistic Regression and 90% of the data as a test set showed very good performance even with a small number of training data. The random baseline threshold was 41% for true positive divided by all samples. The receiver operating characteristic (ROC) for the Logistic Regression using 90% of the dataset as a test set also showed good performance with an AUC of 92%.

Overall, the results suggest that the motor activity data of patients with schizophrenia can be effectively classified using machine learning algorithms, particularly logistic regression.

2.4 'Classification of Depressive and Schizophrenic Episodes using Night-Time Motor Activity Signals'

In the study Classification of Depressive and Schizophrenic Episodes using Night-Time Motor Activity Signals [7], the objective was to increase the accuracy of the classification by employing only data from night-time activity (00:00-05:59). Random Forest Classifier (RFC) models were trained to classify activity data from the datasets already described: 32 healthy controls, 22 schizophrenic patients, and 23 depressive patients. The data was divided into time segments (night-time, morning, afternoon, and evening), and fivefold cross-validation was performed to obtain accuracy metrics for each model. Additionally, precision, recall, F1 score, and MCC were calculated for each model and class, and ROC curves were generated to provide a general model representation.

The study found that despite data imbalance, the RFC models performed well in classifying the different groups, with accuracy ranging from 80.92% to 98.62% depending on the time segment. The results showed that the best model was night-featured data and the random forest classifier, with 98% accuracy for the classification of three classes.

2.5 'A Data-Driven approach for the Analysis of Behavioural Disorders with a Focus on Classification and Severity Estimation'

This study [6] proposed to differentiate between multiple behavioural disorders and assess their severity level by creating a Behavioural Health Score (BHS) index for each subject.

The mobility data was processed, 48 features were created to represent the motor activity of each subject in the dataset [10]. For each participant, every single day activity is segmented into 24 hour-wise mean activity features and 24 hour-wise SD activity features. A total of 48 features are obtained for each person for the total number of days. A unique id is also appended to the dataset to identify each participant. The final dataset contains 55 observations, each representing a person with 49 feature variables, including the 48 hour-wise mean and SD activity features and a unique id.

Then a correlation network graph was built by estimating pair-wise correlation. Markov Clustering (MCL) technique was employed for classification. MCL is an unsupervised clustering algorithm that is used for extracting clusters in biological networks. It works by the random walk property of a graph where all nodes are visited (77 nodes in this case, representing each participant) to find the most strongly connected nodes in the graph.

The obtained results demonstrate that the proposed correlation network model can distinguish between different types of disorders, such as depression, schizophrenia, and healthy individuals. The BHS index can be utilized to estimate the severity levels of the described disorder.

2.5 Proposed approach

The studies reviewed in this chapter have demonstrated the potential of using motor activity data for the classification of mental illness. Two of the studies reviewed are baseline and clearly demonstrate that a high accuracy of illness prediction is possible from the data provided.

The paper 'PSYKOSE: A Motor Activity Database of Patients with Schizophrenia' produced effective accuracy results using the following features: Daily 'mean', 'Standard deviation' and 'Proportion of Zero' on the activity readings. Although, it only attempts to classify schizophrenia against the control group. We propose to use the same features on the Depression and Schizophrenia data combined.

This project takes a different approach from the previous projects reviewed, including the combination of data from two types of behavioural disorder, the use of 24-hour data for analysis, the inclusion of sleep pattern analysis, the evaluation of different machine learning methods and approaches, and the comparison of different machine learning classification approaches. This approach will enable a more comprehensive analysis of the relationship between motor activity and mental illness and may lead to the development of more accurate and personalized approaches to the diagnosis and treatment of depression.

3. Methodology

3.1 Materials and Methods

3.1.1 Data selection Collecting the data is the first step in the process. The collection process is outlined here [9]. For this project the data (as described at 2.1 above) described here will be used [8]. As stated, the '*Depresjon*' dataset of 23 CSV files of depressive subjects data readings, the '*Psychose*' dataset consisting of 22 CSV files of schizophrenic subjects, and 32 CSV files of control group data readings of healthy participants, with each .csv representing a participant

The data contains the activity count along with a per-minute timestamp for all participants. The data was collected from actigraphy devices worn by the subjects for (in most cases) 16 days. Each participant's data was stored in a separate data file. Files were distinguishable by each participants unique id.

3.1.2 Pre-processing

Our proposed approach involves amalgamating all the data from depressive and schizophrenic patients and healthy controls into one dataset. The proposed approach of amalgamating all the data presents a different approach to other classification projects and aims to increase the sample size for improved generalisation.

We plan to use cross-validation algorithms such as LeavePGroupsOut (LOGO) and K-Fold stratification to help maximize the learning from limited data. We plan to extract daily average, daily standard deviation, and daily proportion of zero activity readings as features from the amalgamated dataset. We will then use machine learning algorithms to classify the data into the three groups of depressive patients, schizophrenic patients, and healthy controls.

The raw sensor data from every participant data file was combined into a single dataset. The numbers of days recorded for all participants varied significantly. The minimum number of days recorded was 14 (condition_8: 06/05/2004-19/05/2004), the maximum number of days recorded was 47 (control_3: 6/11/2002-22/12/2002).

The number of days for each participant conflicts with the information provided with the datasets in the *Days.csv* file of the dataset. An analysis of the daily recordings was performed – there were significantly more days of valid data recorded than was indicated in the information files released with the dataset. Table 2 shows the statistical results of the merged dataset. Most of the participants recorded for 16 days. There were no participants who recorded less than 14 days.

Table 2 Activity summary statistics

Days Recorded				
Total days overall	**Minimum**	**Maximum**	**Median & Mode**	**Total days with 1440 minutes**
1534	**14**	**47**	**16**	**1369**

Once a single dataset was created, a category of '*class*' was generated (0: Control; 1: Depression; 2: Schizophrenia) which will be the target variable.

Based on the results of the previous experiments and referring to source materials [13], we developed the features that we will use in modelling. We created statistical features from the activity data. They are calculated by patient, by day. The daily average (*f.mean*), daily standard deviation (*f.sd*), and daily proportion of zero (*f.propZeros*) activity readings fields for each day was created.

Only days which had recordings of 1440 minutes were used to create features. Features which produced a zero value for *f.mean* were dropped. There were 86 features produced that had a zero value for mean and standard deviation. Some of these cases were checked, there had been zero activity recorded by various participants for 86 days. It can be assumed that the device was not worn for these days, but still recorded. For example, patient_9 has 24 hours of zero values for the dates 10/10-11/10 and 12/10.

Once the 86 cases were dropped the resulting daily data had 1283 data points (feature vectors). There were 354 depression cases, 605 control and 324 schizophrenic data points.

Table 3 24-hour features dataset

id	date	f.mean	f.sd	f.propZeros	class	category	counter	patientID
condition_16	20/10/2005	34.51042	183.9244	0.042361111	1	Depressive	129	8
control_30	05/02/2006	212.0819	355.9682	0.043055556	0	Control	399	47
patient_3	09/09/2003	257.9528	291.6843	0.047916667	2	Schizophrenic	227	71
control_5	28/02/2003	3.510417	1.233605	0.050694444	0	Control	492	51
control_30	08/02/2006	207.6069	330.3308	0.0625	0	Control	402	47
patient_2	15/09/2003	132.5194	168.4309	0.068055556	2	Schizophrenic	168	67
control_30	01/02/2006	297.6243	363.3132	0.070138889	0	Control	395	47
control_30	11/02/2006	298.8188	484.2024	0.075	0	Control	405	47
control_7	17/04/2003	2.770833	0.797135	0.076388889	0	Control	564	53
control_30	04/02/2006	222.1667	359.5693	0.079166667	0	Control	398	47
patient_20	27/01/2004	12.05764	57.24741	0.079166667	2	Schizophrenic	185	68
condition_8	18/05/2004	23.88125	107.296	0.085416667	1	Depressive	341	22

Table 3 shows the first set of records from the 24-hour features dataset. Daily feature values can be seen (*f.mean*, *f.sd* & *f.propZeroes*). A '*patientID*' variable was generated assigning a number to each participant (1-77), to be used by the Leave-One-Out cross-validation model, '*counter*' and '*category*' variables were created for plotting and visualisations

3.1.3 Data Analysis

As part of our pre-processing and preliminary investigations and using the demographic and severity score information about the cases provided, we compared individuals with from similar gender and age group from each category group. This experiment was performed to investigate if mobility patterns are distinguishable between patients with a disorder and a healthy patient.

For example, a healthy female aged 40-44 who participated for 17 days is compared to a reported depressive and schizophrenic with the same characteristics. The same comparisons were done with males aged 30-34 and females aged 50-59. Also, all individuals had per minute readings for 24 hours for each day for varying amounts of days from 13-32 days (after data cleaning). *Note: The first 13 days of data for each of the selected individuals was used for comparison. 13 days was chosen for consistency between all individuals as it was the minimum of days recorded post data cleaning.*

Of the three depressive patients selected, all recorded a mean MADRS score of 16-26. This indicates moderate depression. Symptoms of moderate depression may interfere with an individual's ability to function normally in their daily life, affecting their work, social relationships, and personal life. People with moderate depression may experience a range of symptoms such as persistent sadness, loss of interest or pleasure in activities, feelings of worthlessness or guilt, changes in appetite and sleep patterns, fatigue, and difficulty concentrating.

The 3 schizophrenic patients selected all recorded mean BPRS scores of 38-59. A BPRS score of 38-59 generally indicates moderate schizophrenia denoting symptoms like those described for depression.

In both groups, moderate symptoms can interfere with a person's daily functioning and quality of life, but they may not be as severe as those seen in severe or acute mental illness.

The results are consistent for the 3 groups. As the boxplots for the 3 comparison age groups show, the control person in each case recorded a higher median of activity over the 13-day cycle [*figures 2,4 and 6*]. The 24-hour activity readings for the 3 groups are also similar – all 3 conditions show a low level of activity during the night [*figures 1,3 and 5*] but the control participant has a much higher average activity reading in daylight hours.

Figure 1 Average activity of 3 females 40-44

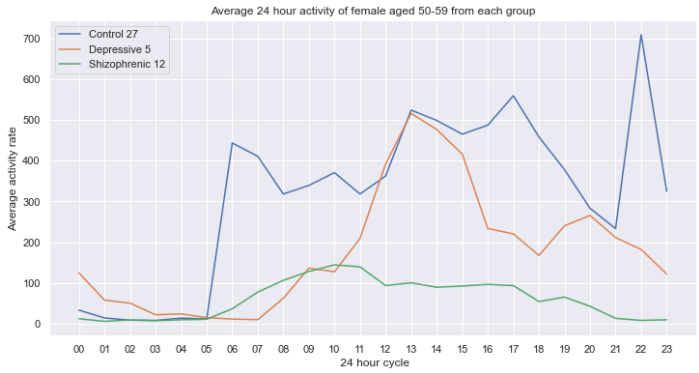


Figure 2 Boxplot of females 40-44

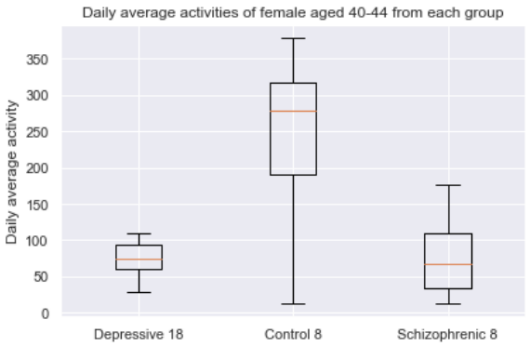


Figure 3 Average activity of females 50-59

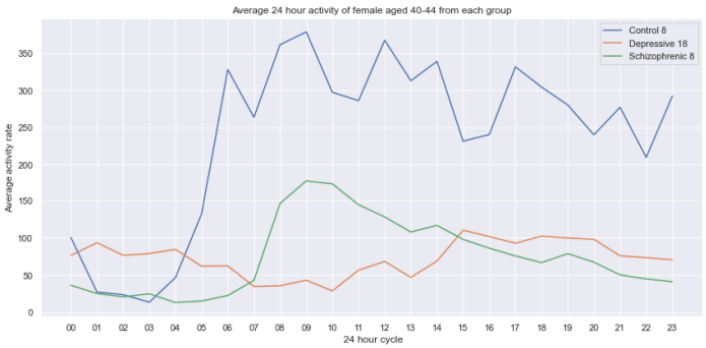


Figure 4 Boxplot of females 50-59

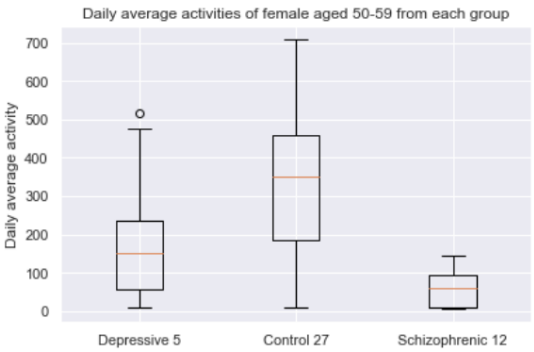


Figure 5 Average activity of males 30-34

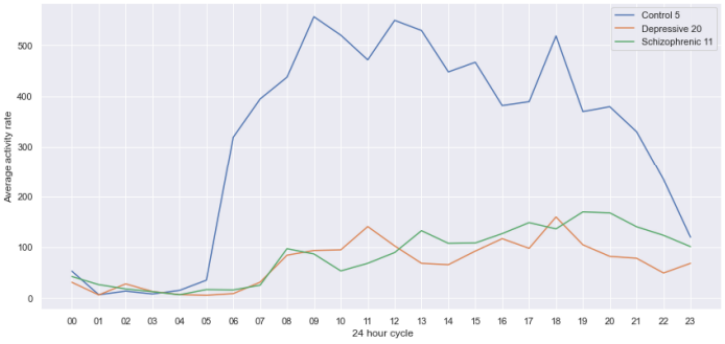
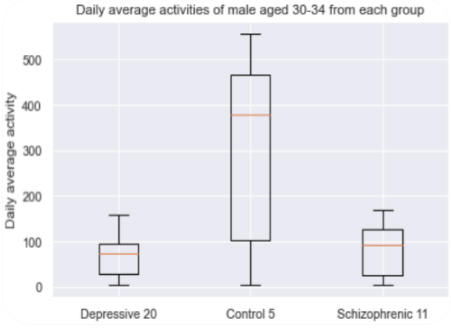


Figure 6 Boxplot of males 30-34

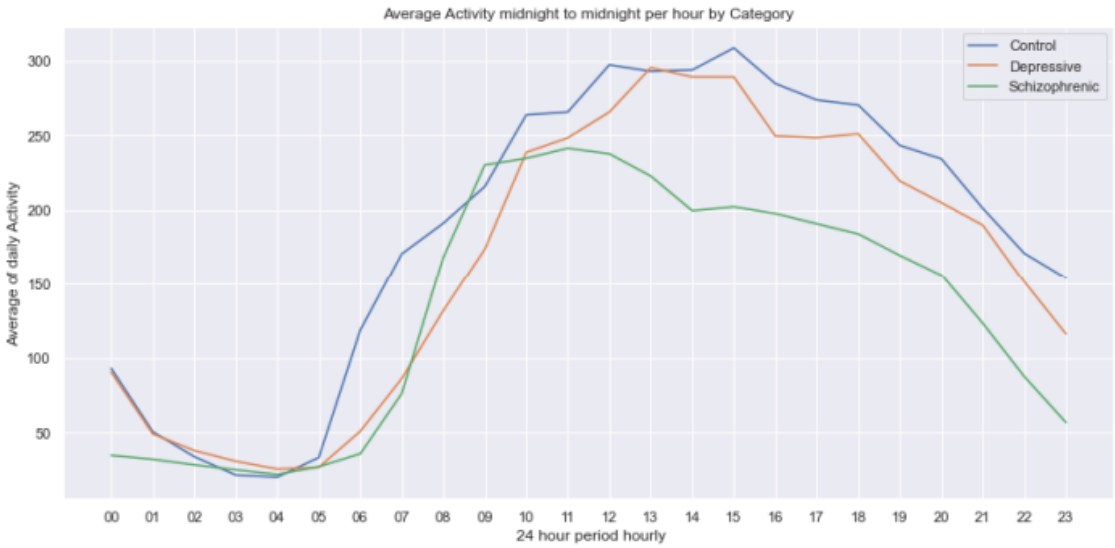


The three boxplots show low medians of activity from the depressive and schizophrenic participants – all show median readings of 100 or less (depression patient 5, outliers for this participant may have impacted the mean and median for this case). The three schizophrenic patients show median activity readings of 100 or less. However, the median reading for the three control participants is significantly higher than the patients, with median activity values of 300-400.

These results were encouraging and support the hypothesis that mobility patterns will be distinguishable between healthy patients and those with behavioural disorders.

Figure 7 shows the average activity for the 3 classes over a 24-hour period. Schizophrenic signal shows the lowest activity levels. Depression are closer to the control activity levels but do show less activity overall than the control group.

Figure 7 Mean activity levels over 24-hours



A further three datasets of features were also created: 4 hourly dataset containing *f.mean*, *f.sd* and *f.propZeros* features for every 4 hours of data (00:00-03:59, 04:00-07:59, 08:00-11:59, 12:00-15:59, 16:00-19:59, 20:00-23:59). This will have the effect of creating 6 times more feature data which may improve performance and accuracy based on higher volumes of testing and training data.

A daytime (08:00-11:59; 12:00-15:59; 16:00-19:59) and night-time dataset were created (20:00-23:59; 00:00-03:59; 04:00-07:59) to investigate activity patterns between the three classes based on previous analysis and results obtained using only night-time activity.

Figure 8 shows the average values of the *f.mean* feature at 4 hourly intervals over the 24 cycle. On average, the control group is consistently more active than the other two groups.

Figure 8 Average of mean activity every 4 hours

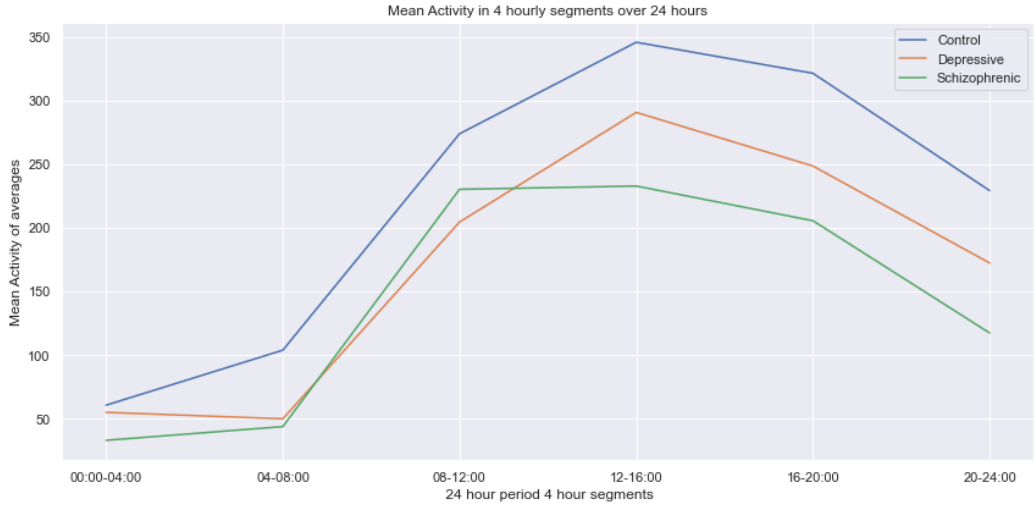
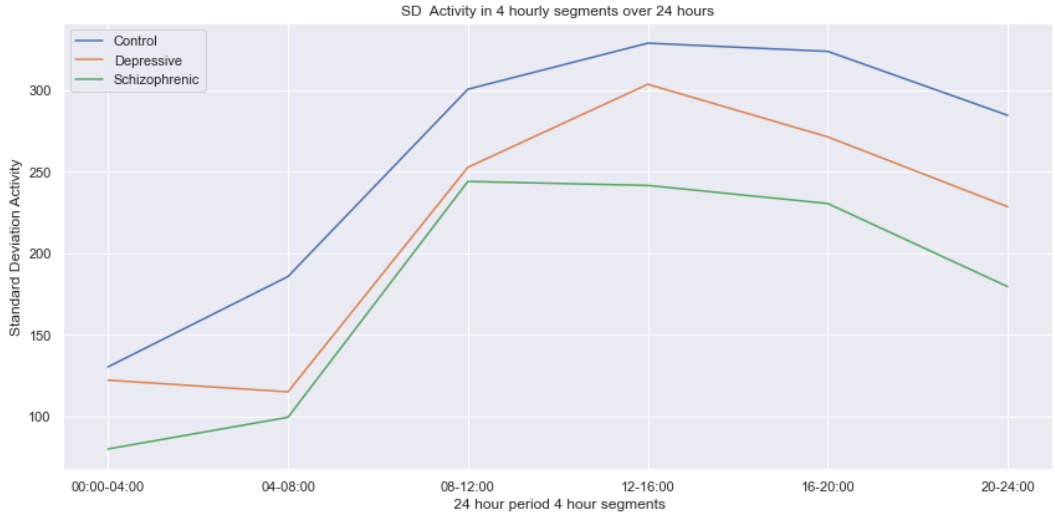
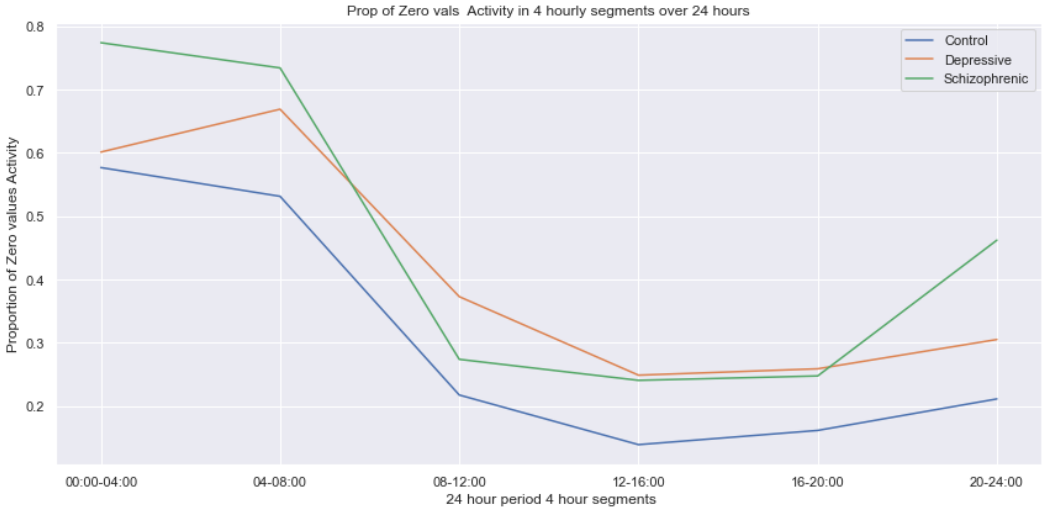


Figure 9 Average of Standard deviation every 4 hours



Figures 9 also shows the mean Standard deviation (*f.sd*) at 4 hourly intervals over 24 hours. Again, the controls group are consistently more active than the two condition groups. *f.mean* and *f.sd* are commonly used features in time series analysis and can provide useful information about the variability and central tendency of the data.

Figure 10 Average of proportion of zeros every 4 hours



In the three graphs, the mean, standard deviation, and proportion of zeros in the activity signal appear to be significantly different between the three groups, with the patient group generally exhibiting lower activity levels compared to the control group. These results suggest that these features could be useful in identifying patterns of activity in patients with different conditions.

3.2 Modelling

3.2.1 Baseline Model Baseline modelling is a crucial step in machine learning because it helps establish a benchmark or starting point for evaluating the performance of future models. In other words, it sets a minimum level of performance that any future model should aim to exceed. Baseline models are typically simple models that are easy to implement and can be used to quickly evaluate the potential of the data set and the modelling approach. It also helps to identify whether there is any signal in the data that can be captured by even the simplest models. If a baseline model performs poorly, it may indicate that the data set is not well-suited for modelling or that more advanced modelling techniques may be necessary. On the other hand, if a baseline model performs well, it suggests that the data contains strong patterns or relationships that can be captured by more sophisticated models.

Furthermore, baseline models can provide a useful reference point for comparing the performance of different models. If a new model performs significantly better than the baseline model, it suggests that the new model is more effective at capturing the underlying patterns in the data. By comparing the performance of different models against a common baseline, it is possible to identify the most effective modelling approach for a given problem.

Initial baseline modelling was performed using Logistic Regression, Random Forest, Decision Tree, XG Boost and Light GBM algorithms. Basic models were created with no cross-validation techniques used. This was done to evaluate the potential of the dataset and to evaluate how the models perform.

Logistic Regression is used to predict the probability of a patient having a particular condition based on the features provided. The results suggest that the logistic regression model is not performing as well as other models, with an accuracy of 46% for the 24-hour features, 43% for the 4-hourly features, 42% for the daytime features, and 49% for the night-time features.

Random Forest is a popular ensemble learning algorithm that combines multiple decision trees to improve prediction accuracy. One of the key advantages of the random forest algorithm is that it can handle many features and can identify the most important features for making predictions. Random forests can also handle missing data and are relatively robust to outliers.

Random forests can be used for both classification and regression tasks. For classification, the final prediction is the most predicted class across all the trees, and for regression, the final prediction is the average of the predicted values across all the trees.

Overall, the random forest algorithm is a popular and effective approach for a wide range of machine learning tasks, and its ability to handle complex data and identify important features makes it a valuable tool for many real-world applications.

The results suggest that the random forest model is performing better than logistic regression, with an accuracy of 66% for the 24-hour features, 53% for the 4-hourly features, 58% for the daytime features, and 59% for the night-time features.

Decision trees are a simple, yet powerful, algorithm that can be used for both classification and regression tasks. A decision tree is a type of model that makes decisions by recursively partitioning the feature space into smaller and smaller regions. The model makes a prediction by traversing the tree from the root to a leaf node, where the leaf node represents the predicted value. The results suggest that the decision tree model is performing better than logistic regression but not as well as random forest, XGBoost, or Light GBM. The accuracy for the decision tree model was 54% for the 24-hour features, 48% for the 4-hourly features, 53% for the daytime features, and 53% for the night-time features.

XGBoost and Light GBM are both gradient boosting algorithms that use decision trees as their base learners. These methods iteratively add new models to an ensemble, with each new model attempting to correct the errors of the previous model. In other words, they combine multiple weak models to create a stronger and more accurate model.

Both XGBoost and Light GBM use decision trees as their base learners. However, there are some differences between the two algorithms. XGBoost uses a technique called gradient boosting, where each new model is trained to minimize the loss function of the previous model's residuals. This approach leads to a more accurate model, but it can be computationally expensive and prone to overfitting.

On the other hand, Light GBM uses a technique called gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB). GOSS is a down sampling method that keeps the important examples and randomly drops the unimportant ones. EFB is a feature bundling technique that reduces the number of feature vectors by merging features that share the same values in most of the samples. These techniques help to reduce overfitting and improve the speed of the algorithm. These models are known for their accuracy and ability to handle large datasets.

The results suggest that both models perform similarly, with an accuracy of 63% for XGBoost and 64% for Light GBM on the 24-hour features. For the 4-hourly features, both models had an accuracy of 53%. For the daytime features, XGBoost and Light GBM had an accuracy of 57% and 59%, respectively. For the night-time features, XGBoost had an accuracy of 54% while Light GBM had an accuracy of 55%.

These models are known for their accuracy and ability to handle large datasets. The results suggest that both models perform similarly, with an accuracy of 63% for XGBoost and 64% for Light GBM on the 24-hour features. For the 4-hourly features, both models had an accuracy of 53%. For the daytime features, XGBoost and Light GBM had an accuracy of 57% and 59%, respectively. For the night-time features, XGBoost had an accuracy of 54% while Light GBM had an accuracy of 55%.

3.2.2 Model Training Hyperparameter tuning is a crucial step in improving the performance of machine learning models. In the case of the 24-hour, 4-hourly, daytime, and night-time features, it was decided to focus on the XGBoost, Light GBM, Random Forest, Gradient Boosting and Decision Tree models for hyperparameter tuning.

The first step in hyperparameter tuning is to define the hyperparameters that will be tuned. These are different from the model parameters and are set before training the model. They determine the behaviour of the algorithm during training, such as the learning rate, the number of trees in the ensemble, the maximum depth of the trees, and the regularization parameters.

After defining the hyperparameters, we created the range of values that each hyperparameter can take. This range will be determined by conducting a grid search, which involves evaluating the performance of the model for a range of hyperparameter values. Grid search involves defining a grid of hyperparameter values and evaluating the performance of the model for each combination of hyperparameters.

Several classifiers were defined with their respective hyperparameters. These classifiers include Random Forest, Decision Tree, XGBoost, Gradient Boosting and Light GBM.

For each classifier, a parameter grid is defined for use in GridSearchCV, which is used to find the best hyperparameters for each classifier. The GridSearchCV object uses the f1 scoring metric and is fit to the training data. The best hyperparameters and corresponding scores are printed for each classifier. The best hyperparameters are determined based on the GridSearchCV object's results. Once the best hyperparameters for each model have been determined using the GridSearchCV object, they can be used to build the final models.

From this process, the combination of hyperparameters that yielded the best performance (e.g., highest accuracy or lowest mean squared error) was identified. These optimal hyperparameters were then used to train the model and make predictions on the 24-hour, 4-hourly, daytime, and night-time datasets.

Since there is a limited amount of data, dividing it into these three sets can result in the model being trained on a small amount of data, which can lead to overfitting. To mitigate overfitting a k-fold cross-validation technique was used, which involves dividing the data into k equal-sized subsets (or folds) and using k-1 of these folds as the training set and the remaining fold as the validation set. This process is repeated k times, each time using a different fold as the validation set. The performance of the model is then averaged over the k runs to give an estimate of its performance on new data.

KFold is a class in the sklearn.model_selection module that provides an implementation of k-fold cross-validation. The number of folds were defined (n_splits = 10), which determines how many times the data will be split into training and validation sets. An instance of KFold with this number of folds was created. A shuffle parameter was used, this indicates whether to shuffle the data before splitting it into folds, and the random_state (2018) parameter sets the random seed for reproducibility.

Initially, a pipeline was created consisting of two steps: imputing missing values and scaling the continuous variables. This is done using the SimpleImputer and StandardScaler classes from scikit-learn, respectively. The pipeline is fit to the training data and used to transform both the training and testing data. The target variable (class) was imbalanced, so the Synthetic Minority Over-sampling Technique (SMOTE) was used to balance it.

3.3 Evaluation

Performance metrics such as accuracy, precision, recall, and F1 score were applied to the models. Cross-validation techniques were used to ensure that the model is not overfitting the data.

To examine the features further, a correlation matrix was created to get an insight into the relationships between the three values. A positive correlation coefficient indicates that the variables are positively related, meaning that as one variable increases, the other variable also tends to increase. A negative correlation coefficient indicates that the variables are negatively related, meaning that as one variable increases, the other variable tends to decrease. A correlation coefficient of 0 indicates that there is no linear relationship between the variables.

Figure 11 Correlation matrix of features

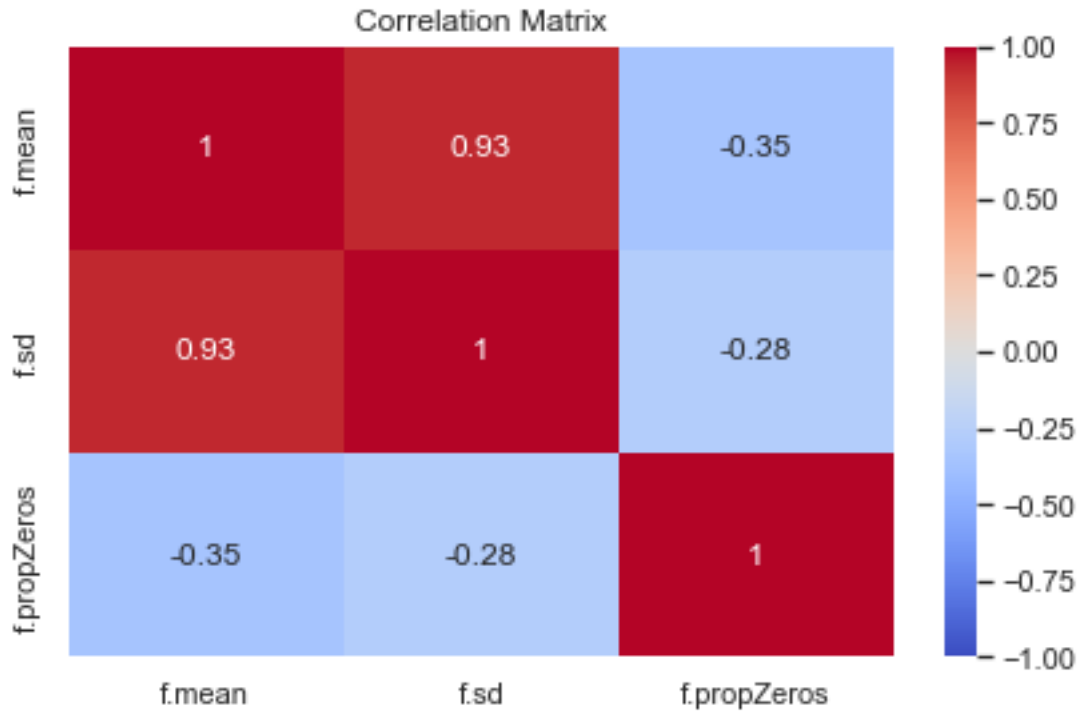
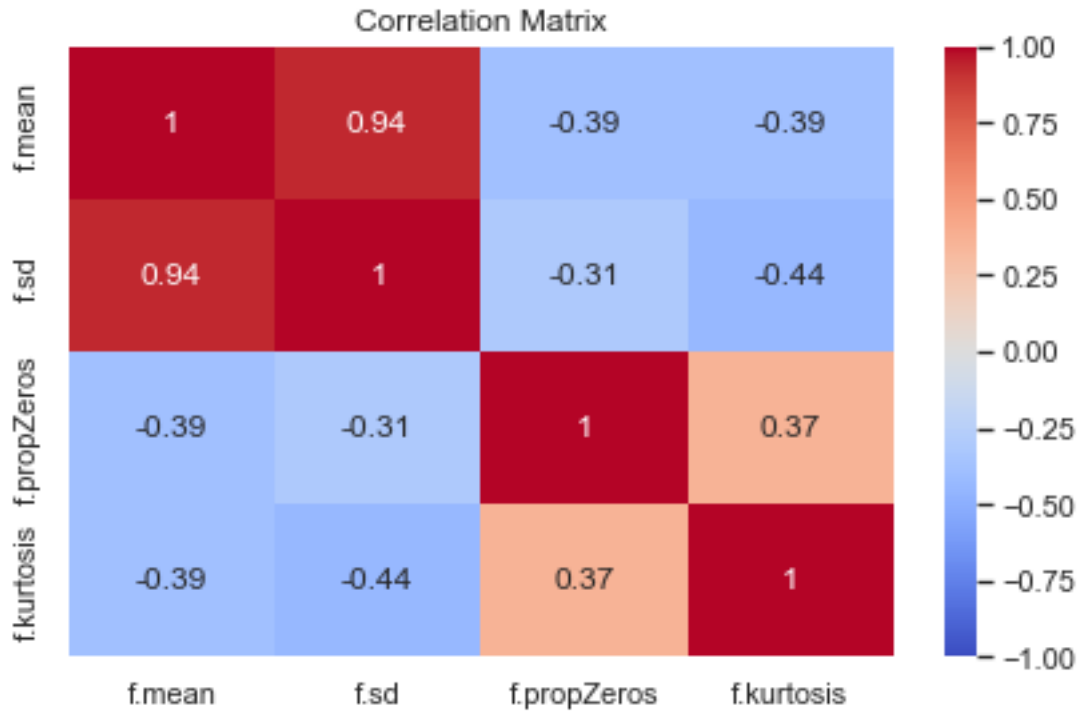


Figure 11 shows the correlation matrix where we observe a high similarity of 92% between *F.mean* and *f.sd*. These two features appear to be highly correlated, which could potentially

lead to multicollinearity issues in our model. However, *F.propZeros* shows a negative correlation with *f.mean* (-35%) and f.sd (-28%). This suggests that *F.propZeros* is a distinguishing feature which should enhance the predictive powers of the model.

It was decided to add another feature to enhance the predictive power of the model. So a new dataset was created with an *f.kurtosis* feature. By adding this feature, it could provide valuable information to a machine learning model. For example, if the distribution of *f.kurtosis* is skewed or has high kurtosis, it could indicate that the feature is an important predictor for a certain outcome. Alternatively, if the distribution is relatively flat, it could indicate that the feature may not be as important in predicting the outcome. Overall, *f.kurtosis* could potentially improve the accuracy and performance of a machine learning model.

Figure 12 Correlation matrix of features (incl kurtosis)



Using the same modelling code, the baseline and hyperparameter code was run to evaluate the impact on the results of introducing the new feature.

4. Results

4.1 Baseline accuracy

Table 3 below describes the accuracy of modelling 3 features (f.mean,f.sd & f.propZeros) with no parameter tuning or cross-validation. These results suggested that Radom Forest Classifier, XGBoost and LightGBM were the most promising algorithms to pursue further for this task

Table 4 Baseline model results

Baseline Modelling results				
	Feature Data			
	24-Hour Data	4-hourly segments	Daytime	Night-time
Classification Models:	%	%	%	%
Logistic Regression	46	43	42	49
Random Forest Classifier	63	53	58	59
XGBoost	61	53	57	54
Light GBM	60	54	59	55
Decision Tree	54	48	53	53

4.2 Model Optimization

The approach of creating a new feature, *f.kurtosis*, and applying GridSearch to find the best hyperparameters will improve the accuracy of the models. The *'f.kurtosis'* feature represents the degree of peakedness or flatness of a probability distribution, which will be relevant making predictions. Using GridSearch to identify the best hyperparameters for the models can improve their performance by optimizing the algorithms' parameters to the specific problem.

4.2.1 Leave-one-group-out

The LeaveOneGroupOut cross-validation technique is a powerful method for evaluating the performance of a classifier in situations where the data is organized into groups or clusters. In this case, the groups are the patients, and the task is to predict their illness based on the features extracted from their daily activity. The LeaveOneGroupOut method works by leaving one group out at a time and using the remaining groups to train the classifier. The classifier is then tested on the left-out group, and the process is repeated until each group has been left out once. This technique ensures that the classifier is evaluated on data that is independent of the training data, making it an effective method for evaluating the classifier's generalization performance.

The models used were Random Forest, Decision Tree, and XG Boost. Light GBM was dropped from the experiment as it produced the same results as XGBoost. The classification was again performed on four different subsets of data: 4 Hourly, Daytime, Night-time, and 24 Hour. Additionally, a combination of all subsets was used for classification.

In Table 5, the Random Forest classifier achieved an accuracy of 54% on the combination of the datasets. The Decision Tree classifier achieved an accuracy of 51%. The XG Boost classifier achieved an accuracy of 53%. The precision, recall, and F1-score for each class are also provided in Table 5. The precision is the fraction of true positives among the predicted positives, the recall is the fraction of true positives among the actual positives, and the F1-score is the harmonic mean of precision and recall.

Overall, the results show that the classifiers were not able to achieve a high level of accuracy in predicting the patients' illness based on their daily activity features. The precision, recall, and F1-score for each class were also relatively low, indicating that the classifiers were not very effective in identifying true positives and true negatives. This may be due to the complexity of the data and the limited number of patients, which may have made it difficult for the classifiers to learn patterns that could reliably predict the patients' illness.

Table 5 Leave one group out results

Dataset	Classifier	% Accuracy	% Recall	% Precision	% F1
4 Hourly	Random Forest	48	49	50	49
	Decision Tree	47	47	49	47
	XG Boost	47	47	49	48
Daytime	Random Forest	53	53	53	53
	Decision Tree	50	50	50	50
	XG Boost	50	50	50	50
Night-time	Random Forest	55	55	56	56
	Decision Tree	52	52	54	53
	XG Boost	53	53	54	53
24 Hour	Random Forest	53	53	54	53
	Decision Tree	52	53	52	52
	XG Boost	53	53	53	53
Combination	Random Forest	54	54	55	53
	Decision Tree	51	52	53	52
	XG Boost	53	53	53	53

4.2.2 10-fold Cross validation

The use of 10 folds of k-fold cross-validation is a powerful method for assessing the generalization of the models. The use of k-fold cross-validation is a valuable approach for evaluating the models' generalization. By splitting the data into 10 folds and training and testing on different subsets, the models are tested on a variety of data and can be assessed for their ability to generalize to new data.

Table 6 shows the scores obtained by the different machine learning models in classifying people into the three classes

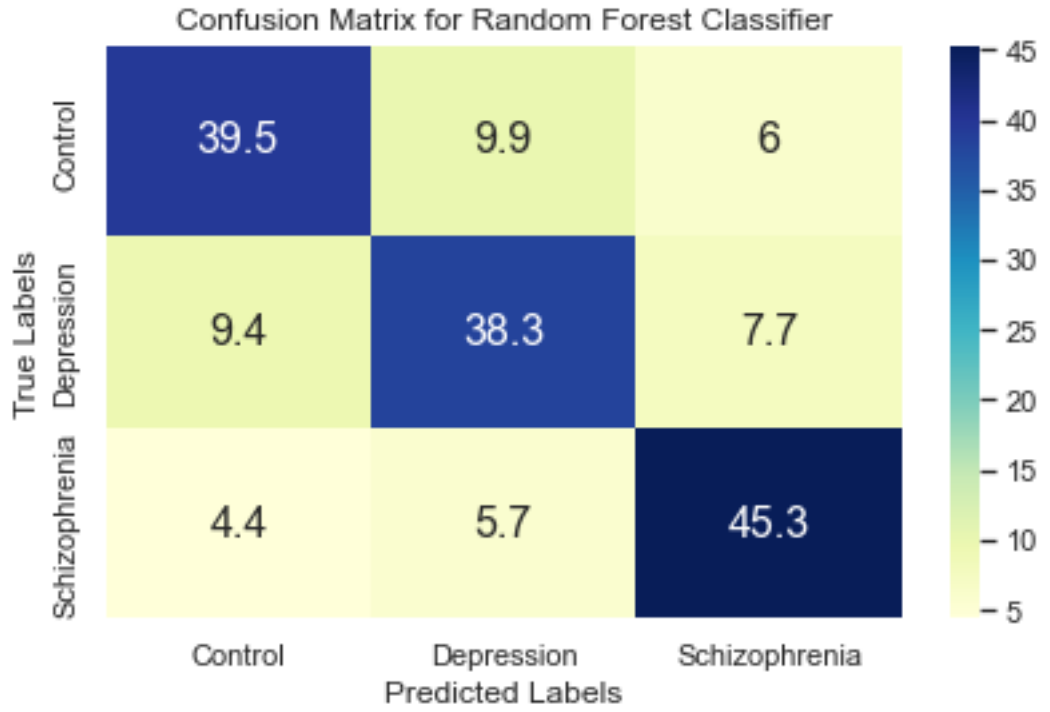
The models were trained using 10-fold cross-validation, and GridSearch was used to find optimal parameters for each model. The Random Forest model performed the best in the 24 Hour subset, with an accuracy score of 74%, an ROC AUC score of 89%, a precision score of 74%, and an F1 score of 74%. This means that, on average, the Random Forest model correctly predicted the class of 74% of the samples in the test set across all 10 folds.The Random Forest 24 hour average ROC AUC score over the 10 folds, is 89%. ROC AUC is a measure of the model's ability to distinguish between the positive and negative classes, with a value of 1 indicating perfect performance and a value of 0.5 indicating random performance. A value of 89% is a good score, suggesting that the model is able to make reasonably accurate predictions.

In the 4 Hourly and Night-time subsets, the Random Forest model also performed well, with an accuracy score of 64% and 69%, respectively. The XG Boost model showed similar performance to Random Forest in most subsets, with the highest accuracy and ROC AUC scores in the 24 Hour subset.

Table 6 10-fold cross validation Classification results

Dataset	Classifier	% Accuracy	% ROC AUC	% Precision	% F1
4 Hourly	Random Forest	64	81	64	64
	Decision Tree	57	68	57	57
	XG Boost	63	80	63	63
Daytime	Random Forest	69	85	69	69
	Decision Tree	61	73	62	61
	XG Boost	67	84	67	67
Night-time	Random Forest	69	86	69	68
	Decision Tree	64	73	64	64
	XG Boost	67	85	67	67
24 Hour	Random Forest Classifier	**74**	**89**	**74**	**74**
	Decision Tree	67	75	68	67
	XG Boost	73	89	74	73
Combination datasets	Random Forest	70	86	70	70
	Decision Tree	63	72	63	63
	XG Boost	70	86	70	70

Figure 13 Confusion matrix 24 hour data



In Figure 13, the confusion matrix shows the average results over 10 folds of cross-validation for the random forest classifier. The matrix is a 3x3 table representing the three classes in the classification problem: Control, Depression, and Schizophrenic.

The numbers in the diagonal from top-left to bottom-right show the number of correctly classified instances for each class. 39.5% of Control instances were correctly classified as Control, 38.3% of Depression instances were correctly classified as Depression, and 45.3% Schizophrenic instances were correctly classified as Schizophrenic.

The off-diagonal elements show the misclassified instances. For instance, 9.9% of Control instances were incorrectly classified as Depression, 9.4% of Depression instances were incorrectly classified as Control, 6% of Schizophrenic instances were incorrectly classified as Control, 7.7% of schizophrenic cases were incorrectly classified as Depression, 5.7% of Depression cases were incorrectly classified as schizophrenia and 4.4% of Control cases were misclassified as schizophrenia.

5. Discussion & Conclusion

Depression and Schizophrenia are difficult to diagnose because of the variation in the symptoms. This paper attempts to address two questions – can motor activity data accurately classify individuals with Depression and schizophrenia, and which Machine Learning algorithms most effective for classifying these disorders. It also aims to advance our understanding of the relationship between motor activity and mental illness and to provide a foundation for the development of novel and effective diagnostic tools.

The creation of baseline models helped to focus on the suitability of algorithms to solve the problem. Analysis was performed on the features in order to examine their effectivness for prediction.

The feature engineering approach of including the ‘f.kurtosis’ feature and using GridSearch for hyperparameter tuning was implemented to improve the models' accuracy. The LeaveOneGroupOut (LOGO) cross-validation technique was used to evaluate the performance of the models, with Random Forest, Decision Tree, and XG Boost models trained on four different subsets of data.

Overall, the LOGO results showed that the models were not able to achieve a high level of accuracy in predicting the patients' illness based on their daily activity features. The precision, recall, and F1-score for each class were relatively low, indicating that the models were not very effective in identifying true positives and true negatives. This could be due to the limited number of patients, making it difficult for the models to learn patterns that could reliably predict the patients' illness.

The 10-fold cross-validation technique was also employed to assess the models' generalization. The Random Forest model performed the best in the 24-Hour subset, with an accuracy score of 74%, an ROC AUC score of 89%, a precision score of 74%, and an F1 score of 74%.

In conclusion, while the study explored the potential of machine learning models in predicting patient illness using activity data, the results indicated that the models' performance was limited. The small sample size of the patient data could be a significant factor contributing to the limited success of the models. In future studies, increasing the sample size and including additional features could improve the accuracy of the models. Overall, this study contributes to the growing body of literature on the use of machine learning techniques in healthcare and highlights the importance of careful evaluation of model performance.

References

[1] "World Health Organization. (2013). Mental Health Action Plan 2013-2020." World Health Organization, 2021. [Online]. Available: https://www.who.int/publications/i/item/9789241506021

[2] Geneva, Switzerland: Author, "Depression and Other Common Mental Disorders: Global Health Estimate," *World Health Organ. 2017*.

[3] C. Arango and C. M. Díaz-Caneja, *Precision psychiatry in the 21st century*.

[4] J. Gill, G. N. Vythelingum, B. J. Whipp, A. P. Anilkumar, and A. Makdani, "Objective measurement of physical activity levels in people with schizophrenia and depression. Journal of Psychiatric Research, 135, 194-201," *J. Psychiatr. Res.*, no. 135, pp. 194–201, 2021, doi: 10.1016/j.jpsychires.2020.12.05.

[5] S. Chen, Y. Chen, X. Lu, L. Li, M. Li, and B. Li, "Association of schizophrenia and depression with daily physical activity and sleep in young adults: A pilot study," *J. Affect. Disord. Rep.*, vol. 4, no. 100137, doi: doi: 10.1016/j.jadr.2021.100137.

[6] R. K. Thelagathoti and H. H. Ali, "A Data-Driven Approach for the Analysis of Behavioral Disorders With a Focus on Classification and Severity Estimation," in *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2022, pp. 821–825. doi: 10.1109/BIBM55620.2022.9994876.

[7] J. G. Rodríguez-Ruiz *et al.*, "Classification of Depressive and Schizophrenic Episodes Using Night-Time Motor Activity Signal.," *Healthc. Basel Switz.*, vol. 10, no. 7, Jul. 2022, doi: 10.3390/healthcare10071256.

[8] E. Garcia Ceja *et al.*, *Depresjon: A Motor Activity Database of Depression Episodes in Unipolar and Bipolar Patients*. 2018. doi: 10.1145/3204949.3208125.

[9] P. Jakobsen *et al.*, "PSYKOSE: A Motor Activity Database of Patients with Schizophrenia," Jul. 2020, pp. 303–308. doi: 10.1109/CBMS49503.2020.00064.

[10] R. K. Thelagathoti and H. Ali, "A Population Analysis Approach using Mobility Data and Correlation Networks for Depression Episodes Detection," vol. 9, Jan. 2021.

[11] S. A. Montgomery and M. Åsberg, "A New Depression Scale Designed to be Sensitive to Change," *Br. J. Psychiatry*, vol. 134, no. 4, pp. 382–389, 1979, doi: 10.1192/bjp.134.4.382.

[12] J. E. Overall and D. R. Gorham, "The Brief Psychiatric Rating Scale," *Psychol. Rep.*, vol. 10, no. 3, pp. 799–812, 1962, doi: 10.2466/pr0.1962.10.3.799.

Appendix

1. Merge Datasets

```
"""
Created on Wed Feb 15 16:06:04 2023

@author: fitzgeraldj
"""
import pandas as pd
import os

def mergeLoop(files):
# Create a DataFrame to store the merged data
  mergedData = pd.DataFrame()
# Loop through all CSV files in the named directory and concatenate into the new DataFrame
# and apply the filename as a new variable
  for file in os.listdir(files):
    if file.endswith('.csv'):
       # Read the CSV file into a DataFrame and add a new column to identify the file
       data = pd.read_csv(os.path.join(files, file))
       data['id'] = file.split('.')[0]  # Add a new column with the file name as identifier

       # Append the DataFrame to the merged_data DataFrame
       mergedData = pd.concat([mergedData, data], ignore_index=True)
  return mergedData

def mergeDepression():
#merge the DEPRESSION files
   path = 'C:/mtu/project/depMerge/'
   mergedData = mergeLoop(path)
   mergedData.to_csv('C:/mtu/project/DepReadings.csv', index=False)

def mergeSchizophrenia():
#merge the SCHIZOPHRENIA files
   path = 'C:/mtu/project/schMerge/'
   mergedData = mergeLoop(path)
   mergedData.to_csv('C:/mtu/project/SchReadings.csv', index=False)

def mergeControl():
#merge the CONTROL files
   path = 'C:/mtu/project/conMerge/'
   mergedData = mergeLoop(path)
   mergedData.to_csv('C:/mtu/project/ConReadings.csv', index=False)

def merge3Files():
#merge the 3 .csv files
    Contro = 'C:/mtu/project/ConReadings.csv'
    Schizo = 'C:/mtu/project/SchReadings.csv'
    Depre = 'C:/mtu/project/DepReadings.csv'
    print("***    Merging multiple files into a single pandas dataframe  ***")
    allData= pd.concat(map(pd.read_csv, [Contro,Schizo,Depre]), ignore_index=True)
    allData.to_csv('C:/mtu/project/AllReadings.csv', index=False)

mergeSchizophrenia()
```

```
mergeDepression()
mergeControl()
merge3Files()
```

2. Clean and create features

```
"""
Created on Wed Mar  1 09:30:08 2023

@author: Jfitz
"""

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
# ================================================================================
#3 files merged
file = 'AllReadings.csv'
data = pd.read_csv(file)
# convert the "date" column to a datetime object
data['date'] = pd.to_datetime(data['date'])
data['timestamp'] = pd.to_datetime(data['timestamp'])
#examine data
print(data.dtypes)
print(f"no of records:  {data.shape[0]}")
print(f"no of variables: {data.shape[1]}")
print((data['id'].nunique()))
# want to drop cases that dont have 24 hours of returns
#aggregate date and hour and include data for 24 hour period only

#count the number of days recorded for each participant
# IE the number of unique dates for each id
date_counts = data.groupby("id")["date"].nunique()
# Write the results to a text file
with open("ParticipantDayCounts.txt", "w") as file:
    for index, value in date_counts.items():
        file.write(f"{index}: {value}\n")
# Print the results
#print(date_counts)


data['hour'] = data['timestamp'].dt.hour
data['minute'] = data['timestamp'].dt.minute + data['hour'] * 60
aggr = data.groupby(['date','hour']).agg({'activity': 'sum'}).reset_index()
aggr = aggr[(aggr['hour'] >= 0) & (aggr['hour'] <= 23)]
counted = aggr.groupby('date').agg({'hour' : 'count'}).reset_index()
counted = counted[counted['hour'] == 24]

final = pd.merge(data, counted[['date']], on='date', how='inner')

counts = final.groupby(['id', 'date']).count()
valid_groups = counts[counts['activity'] == 1440].reset_index()[['id', 'date']]
final = final.merge(valid_groups, on=['id', 'date'])


#print(final.head())
#####Create text categories - for visualisations
def newId(idVal):
    if idVal[:5] == 'condi':
        return 'Depressive'
    elif idVal[:5] == 'patie':
        return 'Schizophrenic'
    elif idVal[:5] == 'contr':
```

```
        return 'Control'
    else:
        return '*UNKNOWN*'

final['category'] = final['id'].apply(newId)

if '*UNKNOWN*' in final['category'].values:
    print("unknowns found")
else:
    print("All 24 hours have a category")

#add a counter for the 3 categories (visualisations)
final['counter'] = final.groupby('category').cumcount() + 1


# create a patient dictionary to map each unique ID
patient = {id:index + 1 for index, id in enumerate(data['id'].unique())}
# map the ID col to the pateientID using the dictionary values
final['patientID'] = final['id'].map(patient)

num_records = len(final)
print(f"Number of records in dataframe: {num_records}")

#Ensure 24 hours of entries for all particpants
if num_records % 1440 == 0:
   print("Number of records is divisible by 1440 with no remainder")
else:
   print("Number of records is NOT divisible by 1440")

#create a segment category based on the time
def seg(hr):
    if hr < 4:
        return '00:00-04:00'
    elif hr > 3 and hr < 8:
        return '04-08:00'
    elif hr > 7 and hr < 12:
        return '08-12:00'
    elif hr > 11 and hr < 16:
        return '12-16:00'
    elif hr > 15 and hr < 20:
        return '16-20:00'
    elif hr > 19 and hr < 24:
        return '20-24:00'
    else:
        return '*UNKNOWN*'

final['segment'] = final['hour'].apply(seg)

#all data is segmented?
if '*UNKNOWN*' in final['segment'].values:
    print("unknowns segments found")
else:
    print("All ids have segments")




######################create features for 24 hour data#########################

# create features for 24 hour data
grouped = final.groupby(['id','date'])
newData = grouped.agg({'activity': ['mean', 'std', lambda x: (x == 0).mean()]})
newData = newData.reset_index()
```

```
newData.columns = ['id','date','f.mean', 'f.sd', 'f.propZeros']

newData['class'] = newData['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
newData = newData[['id','date','f.mean','f.sd','f.propZeros','class']]
discard = newData.loc[((newData['f.mean'] == 0) & (newData['f.sd'] == 0))]
discard.to_csv('C:/mtu/project/removedCases.csv', index=False)
newData = newData.loc[~((newData['f.mean'] == 0) & (newData['f.sd'] == 0))]
print((newData['id'].nunique()))

#create features for 4 hourly data
grouped = final.groupby(['id','date','segment'])
segmented = grouped.agg({'activity': ['mean','std', lambda x: (x == 0).mean()]})
segmented = segmented.reset_index()
segmented.columns = ['id','date','segment','f.mean', 'f.sd', 'f.propZeros']

segmented['class'] = segmented['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
segmented = segmented[['id','date','segment','f.mean','f.sd','f.propZeros','class']]
segmented= segmented.loc[~((segmented['f.mean'] == 0) & (segmented['f.sd'] == 0))]
print((segmented['id'].nunique()))


# Create a daytime and nighttime dataframe
Daysegmented = final.loc[~(final['segment'].isin(['00:00-04:00','04-08:00','20-24:00']))]
Nightsegmented = final.loc[~(final['segment'].isin(['08-12:00','12-16:00','16-20:00']))]
#create features for daytime data 8am - 8pm hour data
grouped = Daysegmented.groupby(['id','date'])
dayData = grouped.agg({'activity': ['mean','std', lambda x: (x == 0).mean()]})
dayData = dayData.reset_index()
dayData.columns = ['id','date','f.mean', 'f.sd', 'f.propZeros']
dayData['class'] = dayData['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
dayData = dayData[['id','date','f.mean','f.sd','f.propZeros','class']]
dayData = dayData.loc[~((dayData['f.mean'] == 0) & (dayData['f.sd'] == 0))]
print((dayData['id'].nunique()))


#create features for night data 8pm - 8am hour data
grouped = Nightsegmented.groupby(['id','date'])
nightData = grouped.agg({'activity': ['mean','std', lambda x: (x == 0).mean()]})
nightData = nightData.reset_index()
nightData.columns = ['id','date','f.mean', 'f.sd', 'f.propZeros']
nightData['class'] = nightData['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
nightData = nightData[['id','date','f.mean','f.sd','f.propZeros','class']]
nightData = nightData.loc[~((nightData['f.mean'] == 0) & (nightData['f.sd'] == 0))]
print((nightData['id'].nunique()))

# ================================================================================
def newId(idVal):
    if idVal[:5] == 'condi':
        return 'Depressive'
    elif idVal[:5] == 'patie':
        return 'Schizophrenic'
    elif idVal[:5] == 'contr':
        return 'Control'
    else:
        return '*UNKNOWN*'

newData['category'] = newData['id'].apply(newId)
segmented['category'] = segmented['id'].apply(newId)
dayData['category'] = dayData['id'].apply(newId)
nightData['category'] = nightData['id'].apply(newId)

if '*UNKNOWN*' in newData['category'].values:
```

```
   print("unknowns found")
else:
   print("All 24 hours have a category")
if '*UNKNOWN*' in dayData['category'].values:
   print("unknowns found")
else:
   print("All daytime data have a category")
if '*UNKNOWN*' in nightData['category'].values:
   print("unknowns found")
else:
   print("All nighttime data have a category")
if '*UNKNOWN*' in segmented['category'].values:
   print("unknowns found")
else:
   print("All segment data has a category")
newData['counter'] = newData.groupby('category').cumcount() + 1
segmented['counter'] = segmented.groupby('category').cumcount() + 1
dayData['counter'] = dayData.groupby('category').cumcount() + 1
nightData['counter'] = nightData.groupby('category').cumcount() + 1
#
# # create a patient dictionary to map each unique ID
patient = {id:index + 1 for index, id in enumerate(newData['id'].unique())}
patient = {id:index + 1 for index, id in enumerate(dayData['id'].unique())}
patient = {id:index + 1 for index, id in enumerate(nightData['id'].unique())}
patient = {id:index + 1 for index, id in enumerate(segmented['id'].unique())}
# # map the ID col to the pateientID using the dictionary values
newData['patientID'] = newData['id'].map(patient)
segmented['patientID'] = segmented['id'].map(patient)
dayData['patientID'] = dayData['id'].map(patient)
nightData['patientID'] = nightData['id'].map(patient)
# ================================================================================

#print(newData)
#print(segmented)
#print(dayData)
#print(nightData)
print("***  All 3 groups Baseline input file created for 24 hr of data only ***")
print("***  Features created for 4 hourly segments/ Daytime and Nighttime ****")


#create output csvs for use in later scripts
newData.to_csv('C:/mtu/project/24HrFeatures.csv', index=False)
segmented.to_csv('C:/mtu/project/4HrFeatures.csv', index=False)
nightData.to_csv('C:/mtu/project/NightFeatures.csv', index=False)
dayData.to_csv('C:/mtu/project/DayFeatures.csv', index=False)

# ================================================================================
# Plots
#--------------------------------------------------------------------------------

# Plot with comaparison of particpants from each caetgory with similar demographics
###################### compare 3 females 40-44 ###################################
extract18 = final.query("id == 'condition_18'").head(18720)   #id 64 - 20160 rows 14 days
extract8 = final.query("id == 'control_8'").head(18720)       #id 31 - 27360 rows 19
extractp8 = final.query("id == 'patient_8'").head(18720)      #id 53 - 27360 rows 19

# Concatenate the data into a new dataframe
extract = pd.concat([extract18, extract8, extractp8], keys=['control_8','condition_18', 'patient_8'])
#print(extract18)
#print(extract8)
#print(extractp8)
# Compute the mean activity by category and hour
grouped = extract.groupby(['category', 'hour'])['activity'].mean().reset_index()
```

```
# Pivot the data to a wide format
pivoted = grouped.pivot(index='hour', columns='category', values='activity')

# Plot the data
plt.plot(pivoted.index, pivoted['Control'], label='Control 8')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive 18')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic 8')
plt.xticks(range(24), [f'{h:02d}' for h in range(24)])
plt.xlabel(' 24 hour cycle')
plt.ylabel('Average activity rate')
plt.title('Average 24 hour activity of female aged 40-44 from each group')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('females40-44.png', dpi=300)
plt.show()

#boxplot
Condition18 = extract18.groupby(['category', 'hour'])['activity'].mean()
Control8 = extract8.groupby(['category', 'hour'])['activity'].mean()
Patient8 = extractp8.groupby(['category', 'hour'])['activity'].mean()
activ = [Condition18,Control8,Patient8]
fig, ax = plt.subplots()
ax.boxplot(activ)
ax.set_xticklabels(['Depressive 18','Control 8','Schizophrenic 8'])
ax.set_ylabel('Daily average activity')
ax.set_title("Daily average activities of female aged 40-44 from each group")
plt.show()


# ======================3 females 50-59====================================
extract5 = final.query("id == 'condition_5'").head(18720)    #id73 - 20160 rows 14 days
extract27 = final.query("id == 'control_27'").head(18720)    #id20 - 18720 rows 13 days
extract12 = final.query("id == 'patient_12'").head(18720)    #id36 - 18720 rows 13 days

# Concatenate the data into a new dataframe
extract = pd.concat([extract5, extract27, extract12], keys=['control_27','condition_5', 'patient_12'])
#print(extract5)
#print(extract27)
#print(extract12)
# Compute the mean activity by category and hour
grouped = extract.groupby(['category', 'hour'])['activity'].mean().reset_index()

# Pivot the data to a wide format
pivoted = grouped.pivot(index='hour', columns='category', values='activity')

# Plot the data
plt.plot(pivoted.index, pivoted['Control'], label='Control 27')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive 5')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Shizophrenic 12')



plt.xticks(range(24), [f'{h:02d}' for h in range(24)])
plt.xlabel(' 24 hour cycle')
plt.ylabel('Average activity rate')
plt.title('Average 24 hour activity of female aged 50-59 from each group')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('females50-59.png', dpi=300)
plt.show()
```

```
#boxplot
Condition5 = extract5.groupby(['category', 'hour'])['activity'].mean()
Control27 = extract27.groupby(['category', 'hour'])['activity'].mean()
Patient12 = extract12.groupby(['category', 'hour'])['activity'].mean()
activ = [Condition5,Control27,Patient12]
fig, ax = plt.subplots()
ax.boxplot(activ)
ax.set_xticklabels(['Depressive 5','Control 27','Schizophrenic 12'])
ax.set_ylabel('Daily average activity')
#ax.set_xlabel('Patient 22 v Control 25')
ax.set_title("Daily average activities of female aged 50-59 from each group")
plt.show()


# ======================= 3 males aged 30-34 ====================================
extract20 = final.query("id == 'condition_20'").head(18720)  #id67 24480 17 days
extract5 = final.query("id == 'control_5'").head(18720)      #id28 46080  32 days
extract11 = final.query("id == 'patient_11'").head(18720)               #id35 18720 13 days

# Concatenate the data into a new dataframe
extract = pd.concat([extract20, extract5, extract11], keys=['control_5', 'condition_20', 'patient_11'])
#print(extract20)
#print(extract5)
#print(extract11)
# Compute the mean activity by category and hour
grouped = extract.groupby(['category', 'hour'])['activity'].mean().reset_index()

# Pivot the data to a wide format
pivoted = grouped.pivot(index='hour', columns='category', values='activity')

plt.plot(pivoted.index, pivoted['Control'], label='Control 5')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive 20')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic 11')

plt.xticks(range(24), [f'{h:02d}' for h in range(24)])
plt.xlabel(' 24 hour cycle')
plt.ylabel('Average activity rate')
plt.title('Average 24 hour activity of male aged 30-34 from each group')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('males30-34.png', dpi=300)
plt.show()

Condition20 = extract20.groupby(['category', 'hour'])['activity'].mean()
Control5    = extract5.groupby(['category', 'hour'])['activity'].mean()
Patient11   = extract11.groupby(['category', 'hour'])['activity'].mean()
activ = [Condition20,Control5,Patient11]
fig, ax = plt.subplots()

ax.boxplot(activ)



ax.set_xticklabels(['Depressive 20','Control 5','Schizophrenic 11'])
ax.set_ylabel('Daily average activity')
ax.set_title("Daily average activities of male aged 30-34 from each group")
plt.show()
# ========================================================================
############# plot of 24 hour averages ###############################
#plot of averages of activity from midnight to midnight by hour
grouped = final.groupby(['category', 'hour'])['activity'].mean().reset_index()
pivoted = grouped.pivot(index='hour', columns='category', values='activity')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
```

```
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')

plt.xticks(range(24), [f'{h:02d}' for h in range(24)])

plt.xlabel(' 24 hour period hourly')
plt.ylabel('Average of daily Activity')
plt.title('Average Activity midnight to midnight per hour by Category')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('AverageActivityPerHour.png', dpi=300)
plt.show()


#plot of 24 hours by minute
grouped = final.groupby(['category', 'minute'])['activity'].mean().reset_index()
pivoted = grouped.pivot(index='minute', columns='category', values='activity')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')

plt.xlabel(' 24 hour period in minutes')
plt.ylabel('Average Activity')
plt.title('Average Activity per minute by Category')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('AverageActivityPerMinute.png', dpi=300)
plt.show()
#################### Plot of averages 4 hour segments ########################
#plot of 4 hourly averages
grouped = segmented.groupby(['category', 'segment'])['f.mean'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.mean')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Mean Activity of averages')
plt.title('Mean Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('MeanActivityPer4hrSegement.png', dpi=300)
plt.show()

# plot of sd averages every 4 hours
grouped = segmented.groupby(['category', 'segment'])['f.sd'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.sd')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Standard Deviation Activity')
plt.title('SD  Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('SD-ActivityPer4hrSegement.png', dpi=300)
plt.show()

#plot of propZeros every 4 hours
grouped = segmented.groupby(['category', 'segment'])['f.propZeros'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.propZeros')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
```

```
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Proportion of Zero values Activity')
plt.title('Prop of Zero vals  Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('PropZeros-ActivityPer4hrSegement.png', dpi=300)
plt.show()
```

3. Baseline Models

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  6 12:12:29 2023
@author: Jfitz
"""
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import xgboost as xgb
import lightgbm as lgb
import pandas as pd
import seaborn as sns
color = sns.color_palette()
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

#file='allFeatureReadings.csv'
file = '24HrFeaturesk.csv'
#file = '4HrFeaturesk.csv'
#file='DayFeaturesk.csv'
#file='NightFeaturesk.csv'
df = pd.read_csv(file)


#Prepare the data for modeling
X = df.copy().drop(['id','class','date','category','counter','patientID'],axis=1)
y = df['class'].copy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=2018)


#Create the models:
# Create a pipeline to impute missing values and scale the continuous variables
pipeline = make_pipeline(SimpleImputer(strategy='median'), StandardScaler())

# Fit the pipeline to the training data and transform the training and testing data
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)

# calculate descriptive statistics for each feature
desc_stats = df[["f.mean", "f.sd", "f.propZeros","f.kurtosis"]].describe()
print(desc_stats)
```

```
from scipy.stats import ttest_ind


# separate the data into patient and control groups
patient_df = df[df["category"] != "Control"]
control_df = df[df["category"] == "Control"]

# loop through each feature and perform a t-test to compare the distributions between patient and control groups
for feature in ["f.mean", "f.sd", "f.propZeros","f.kurtosis"]:
   patient_data = patient_df[feature]
   control_data = control_df[feature]
   t, p = ttest_ind(patient_data, control_data)
   print("Feature:", feature)
   print("T-statistic:", t)
   print("P-value:", p)
   if p < 0.05:
      print("There is a significant difference in the distribution of this feature between patient and control groups.")
   else:
      print("There is no significant difference in the distribution of this feature between patient and control groups.")


# compute the correlation between "f.mean", "f.sd", and "f.propZeros"
correlation = df[["f.mean", "f.sd", "f.propZeros","f.kurtosis"]].corr()
print(correlation)
# plot the correlation matrix as a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

# add a title to the plot
plt.title("Correlation Matrix")

# show the plot
plt.show()


#----------------------Logistic Regression
logReg = LogisticRegression()

model = logReg

model.fit(X_train, y_train)

# Evaluate the model's performance on the testing data
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)

print(f" Logistic Regression Accuracy: {accuracy}")


import matplotlib.pyplot as plt
import seaborn as sns
# Plot the confusion matrix as a heatmap
label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(confusion_mat, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
        xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for LR')
plt.show()

# Extract the classification report values
report = classification_report(y_test, y_pred, output_dict=True)
```

```
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
   if key in ['accuracy', 'macro avg', 'weighted avg']:
      continue
   data['precision'].append(value['precision'])
   data['recall'].append(value['recall'])
   data['f1-score'].append(value['f1-score'])
   data['support'].append(value['support'])

# Define label names
label_names = ['Control', 'Depression', 'Schizophrenia']
# Create a heatmap
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
        xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
        yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for Logistic Regression')
plt.show()

#---------- Gradient boost

gbm = GradientBoostingClassifier()
# fit the model on the training data
gbm.fit(X_train, y_train)

# predict on the test data
y_pred = gbm.predict(X_test)

# evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Gradient Boost Accuracy: {accuracy}")
confusionGB = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix as a heatmap
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(confusionGB, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
        xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Gradient Boosting')
plt.show()


# Extract the classification report values
report = classification_report(y_test, y_pred, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
   if key in ['accuracy', 'macro avg', 'weighted avg']:
      continue
   data['precision'].append(value['precision'])
   data['recall'].append(value['recall'])
   data['f1-score'].append(value['f1-score'])
   data['support'].append(value['support'])

# Create a heatmap
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
        xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
        yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for Gradient Boost')
plt.show()
```

```
#---------------------- Random Forest

# create a random forest classifier object
rfc = RandomForestClassifier()

# fit the model on the training data
rfc.fit(X_train, y_train)

# predict on the test data
y_pred = rfc.predict(X_test)

# evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Random Forest Accuracy: {accuracy}")
confusionRF = confusion_matrix(y_test, y_pred)


# Plot the confusion matrix as a heatmap
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(confusionRF, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
        xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for RF')
plt.show()


# Extract the classification report values
report = classification_report(y_test, y_pred, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
   if key in ['accuracy', 'macro avg', 'weighted avg']:
      continue
   data['precision'].append(value['precision'])
   data['recall'].append(value['recall'])
   data['f1-score'].append(value['f1-score'])
   data['support'].append(value['support'])

# Create a heatmap
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
        xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
        yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for Random Forest Classifier')
plt.show()
#----------------- XG Boost


# Define the model
xgb_model = xgb.XGBClassifier()

xgb_model.fit(X_train, y_train)

# Evaluate the model
y_pred = xgb_model.predict(X_test)

acc_score = accuracy_score(y_test, y_pred)
conf_mat = confusion_matrix(y_test, y_pred)

print("XGB Accuracy score:", acc_score)
```

```
# Plot the confusion matrix as a heatmap
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(conf_mat, annot=True, annot_kws={"size": 16}, cmap="Blues", fmt='g',
        xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for XGB')
plt.show()


# Extract the classification report values
report = classification_report(y_test, y_pred, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
   if key in ['accuracy', 'macro avg', 'weighted avg']:
      continue
   data['precision'].append(value['precision'])
   data['recall'].append(value['recall'])
   data['f1-score'].append(value['f1-score'])
   data['support'].append(value['support'])

# Create a heatmap
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
        xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
        yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for XGBoost')
plt.show()

#----------------------Light GBM


# Define the model
lgb_model = lgb.LGBMClassifier()

lgb_model.fit(X_train, y_train)

# Evaluate the model
y_pred = lgb_model.predict(X_test)

acc_score = accuracy_score(y_test, y_pred)
conf_mat = confusion_matrix(y_test, y_pred)

print("Light GBM Accuracy score:", acc_score)

# Plot the confusion matrix as a heatmap
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(conf_mat, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
        xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for LGBM')
plt.show()


# Extract the classification report values
report = classification_report(y_test, y_pred, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
   if key in ['accuracy', 'macro avg', 'weighted avg']:
      continue
   data['precision'].append(value['precision'])
```

```
    data['recall'].append(value['recall'])
    data['f1-score'].append(value['f1-score'])
    data['support'].append(value['support'])

# Create a heatmap
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
            xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
            yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for LightGBM')
plt.show()
#------------------Decision Tree


#  Define the decision tree model
dt_model = DecisionTreeClassifier()

dt_model.fit(X_train, y_train)

# Evaluate the model
y_pred = dt_model.predict(X_test)

acc_score = accuracy_score(y_test, y_pred)
conf_mat = confusion_matrix(y_test, y_pred)

print("Decision Tree Accuracy score:", acc_score)

# Plot the confusion matrix as a heatmap
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(conf_mat, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
            xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for RF')
plt.show()


# Extract the classification report values
report = classification_report(y_test, y_pred, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
    if key in ['accuracy', 'macro avg', 'weighted avg']:
        continue
    data['precision'].append(value['precision'])
    data['recall'].append(value['recall'])
    data['f1-score'].append(value['f1-score'])
    data['support'].append(value['support'])

# Create a heatmap
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
            xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
            yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for Decision Tree')
plt.show()
```

4. Kfold and Hyperparameters

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 26 22:20:04 2023

@author: Jfitz
"""
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, LeavePGroupsOut,
TimeSeriesSplit
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classification_report
from sklearn.model_selection import KFold
import xgboost as xgb
import lightgbm as lgb
from imblearn.over_sampling import SMOTE
from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
color = sns.color_palette()

#file='allFeatureReadings.csv'
#file = '4HrFeatures.csv'
#file='DayFeaturesk.csv'
#file='NightFeatures.csv'
file = '24HrFeaturesk.csv'
df = pd.read_csv(file)

# Prepare the data for modeling
X = df.drop(['id','class','date','category','counter','patientID'], axis=1)
y = df['class'].copy()

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=2018)

# Create a pipeline to impute missing values and scale the continuous variables
pipeline = make_pipeline(SimpleImputer(strategy='median'), StandardScaler())

# Fit the pipeline to the training data and transform the training and testing data
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)

# Balance the target variable using SMOTE
smote = SMOTE(random_state=2018)
X_train, y_train = smote.fit_resample(X_train, y_train)

# Define the models and their hyperparameters
rfc = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=2018)
dtc = DecisionTreeClassifier(max_depth=10, random_state=2018)
xgbc = xgb.XGBClassifier(n_estimators=200, max_depth=10, random_state=2018)
```

```
gbm = GradientBoostingClassifier(n_estimators=200, max_depth=10, random_state=2018)
lgbmc = lgb.LGBMClassifier(n_estimators=200, max_depth=10, random_state=2018)

# Define the parameter grids for GridSearchCV
rfc_param_grid = {'n_estimators': [100, 200, 500], 'max_depth': [5, 10, 20]}
dtc_param_grid = {'max_depth':[5, 10, 20]}
xgbc_param_grid = {'n_estimators': [100, 200, 500], 'max_depth': [5, 10, 20]}
gbm_param_grid = {'n_estimators': [100, 200, 500], 'max_depth': [5, 10, 20]}
lgbmc_param_grid = {'n_estimators': [100, 200, 500], 'max_depth': [5, 10, 20]}

# Define the GridSearchCV objects with f1 as the scoring metric
rfc_grid = GridSearchCV(estimator=rfc, param_grid=rfc_param_grid, scoring='accuracy', cv=5)
dtc_grid = GridSearchCV(estimator=dtc, param_grid=dtc_param_grid, scoring='accuracy', cv=5)
xgbc_grid = GridSearchCV(estimator=xgbc, param_grid=xgbc_param_grid, scoring='accuracy', cv=5)
gbm_grid = GridSearchCV(estimator=gbm, param_grid=gbm_param_grid, scoring='accuracy', cv=5)
lgbmc_grid = GridSearchCV(estimator=lgbmc, param_grid=lgbmc_param_grid, scoring='accuracy', cv=5)

# Fit the GridSearchCV objects
rfc_grid.fit(X_train, y_train)
dtc_grid.fit(X_train, y_train)
xgbc_grid.fit(X_train, y_train)
gbm_grid.fit(X_train, y_train)
lgbmc_grid.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding scores
print("Best hyperparameters for Random Forest: ", rfc_grid.best_params_)
print("Best score for Random Forest: ", rfc_grid.best_score_)
print("Best hyperparameters for Decision Tree: ", dtc_grid.best_params_)
print("Best score for Decision Tree: ", dtc_grid.best_score_)
print("Best hyperparameters for XGBoost: ", xgbc_grid.best_params_)
print("Best score for XGBoost: ", xgbc_grid.best_score_)
print("Best hyperparameters for Gradient Boosting: ", gbm_grid.best_params_)
print("Best score for Gradient Boosting: ", gbm_grid.best_score_)
print("Best hyperparameters for LightGBM: ", lgbmc_grid.best_params_)
print("Best score for LightGBM: ", lgbmc_grid.best_score_)

#Train the models with the best hyperparameters on the full training set
rfc_best = RandomForestClassifier(**rfc_grid.best_params_, random_state=2018)
#rfc_best.fit(X_train, y_train)

dtc_best = DecisionTreeClassifier(**dtc_grid.best_params_, random_state=2018)
#dtc_best.fit(X_train, y_train)

xgbc_best = xgb.XGBClassifier(**xgbc_grid.best_params_, random_state=2018)
#xgbc_best.fit(X_train, y_train)

gbm_best = GradientBoostingClassifier(**gbm_grid.best_params_, random_state=2018)
#gbm_best.fit(X_train, y_train)

lgbmc_best = lgb.LGBMClassifier(**lgbmc_grid.best_params_, random_state=2018)
#lgbmc_best.fit(X_train, y_train)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=2018)
# Create a pipeline to impute missing values and scale the continuous variables
pipeline = make_pipeline(SimpleImputer(strategy='median'), MinMaxScaler())

# Fit the pipeline to the training data and transform the training and testing data
X = pipeline.fit_transform(X)

# Detect and remove outliers using One-Class SVM
svm = OneClassSVM(nu=0.05)
outlier_mask = svm.fit_predict(X) == 1
```

```
X = X[outlier_mask]
y = y[outlier_mask]

# Balance the target variable using SMOTE
smote = SMOTE(random_state=2018)
X, y = smote.fit_resample(X, y)

# Create a pipeline to impute missing values and scale the continuous variables
pipeline = make_pipeline(SimpleImputer(strategy='median'), StandardScaler())


n_splits = 10

# Perform k-fold cross-validation
#kfold = KFold(n_splits=n_splits, shuffle=True, random_state=2018)
# Perform stratified k-fold cross-validation
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=2018)

models = {
   'Random Forest Classifier': RandomForestClassifier(**rfc_grid.best_params_, random_state=2018),
   'Decision Tree Classifier': DecisionTreeClassifier(**dtc_grid.best_params_, random_state=2018),
   'XGBoost Classifier': xgb.XGBClassifier(**xgbc_grid.best_params_, random_state=2018),
 #  'Gradient Boosting Classifier': GradientBoostingClassifier(**gbm_grid.best_params_, random_state=2018),
 #  'LightGBM Classifier': lgb.LGBMClassifier(**lgbmc_grid.best_params_, random_state=2018)
}
for model_name, model in models.items():
   print(f"Running {model_name}...")
   # lists to store metrics for each fold
   acc_scores = []
   auc_scores = []
   cm_list = []
   cr_list = []
   roc_auc_list = []  # list to store ROC AUC scores for each fold
   precision_list = []  # list to store precision scores for each fold
   f1_score_list = []  # list to store f1-score scores for each fold
   for fold, (train_index, test_index) in enumerate(skf.split(X, y)):
      X_train, X_test = X[train_index], X[test_index]
      y_train, y_test = y[train_index], y[test_index]

      # Fit the pipeline to the training data and transform the training and testing data
      X_train = pipeline.fit_transform(X_train)
      X_test = pipeline.transform(X_test)

      # Fit the model
      model.fit(X_train, y_train)

      y_pred = model.predict(X_test)
      y_pred_proba = model.predict_proba(X_test)

      # Compute metrics
      acc_score = accuracy_score(y_test, y_pred)
      auc_score = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')
      cm = confusion_matrix(y_test, y_pred)
      cr = classification_report(y_test, y_pred)
      precision = precision_score(y_test, y_pred, average='macro')
      f1 = f1_score(y_test, y_pred, average='macro')

      # Append metrics to lists
      acc_scores.append(acc_score)
      auc_scores.append(auc_score)
      cm_list.append(cm)
      cr_list.append(cr)
      roc_auc_list.append(auc_score)
```

```
    precision_list.append(precision)
    f1_score_list.append(f1)

# calculate the average metrics over all folds
avg_acc_score = np.mean(acc_scores)
avg_auc_score = np.mean(auc_scores)
avg_cm = np.mean(cm_list, axis=0)
avg_precision = np.mean(precision_list)
avg_f1_score = np.mean(f1_score_list)
print(f"{model_name}")
print("Average accuracy score over 10 folds : {:.2f}".format(avg_acc_score))
print("Average ROC AUC score over 10 folds : {:.2f}".format(avg_auc_score))
print("Average Confusion Matrix over 10 folds :\n{}".format(avg_cm))
print("Average Precision over 10 folds : {:.2f}".format(avg_precision))
print("Average F1-score over 10 folds : {:.2f}".format(avg_f1_score))


# Plot the confusion matrix as a heatmap
label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1) # Adjust to fit labels within the plot area
#sns.heatmap(confusion_mat, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g')
sns.heatmap(avg_cm, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
        xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
# plt.title(f"Confusion Matrix for  {model_name}")
plt.title(f"Confusion Matrix for {model_name}")
plt.show()
```

5. Create Kurtosis Feature

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 26 11:45:54 2023

@author: Jfitz
"""


import pandas as pd
#import numpy as np
from scipy.stats import kurtosis
#import math
#import os
import matplotlib.pyplot as plt
#import matplotlib.ticker as ticker
import seaborn as sns
from datetime import datetime
# ============================================================================
#3 files merged
file = 'AllReadings.csv'
data = pd.read_csv(file)
# convert the "date" column to a datetime object
data['date'] = pd.to_datetime(data['date'])
data['timestamp'] = pd.to_datetime(data['timestamp'])
#examine data
print(data.dtypes)
print(f"no of records:  {data.shape[0]}")
print(f"no of variables: {data.shape[1]}")
print((data['id'].nunique()))
# want to drop cases that dont have 24 hours of returns
#aggregate date and hour and include data for 24 hour period only
data['hour'] = data['timestamp'].dt.hour
data['minute'] = data['timestamp'].dt.minute + data['hour'] * 60
aggr = data.groupby(['date','hour']).agg({'activity': 'sum'}).reset_index()
aggr = aggr[(aggr['hour'] >= 0) & (aggr['hour'] <= 23)]
counted = aggr.groupby('date').agg({'hour' : 'count'}).reset_index()
counted = counted[counted['hour'] == 24]

final = pd.merge(data, counted[['date']], on='date', how='inner')

counts = final.groupby(['id', 'date']).count()
valid_groups = counts[counts['activity'] == 1440].reset_index()[['id', 'date']]
final = final.merge(valid_groups, on=['id', 'date'])


#print(final.head())
#examine the data#####################################################################
def newId(idVal):
    if idVal[:5] == 'condi':
        return 'Depressive'
    elif idVal[:5] == 'patie':
        return 'Schizophrenic'
    elif idVal[:5] == 'contr':
        return 'Control'
    else:
        return '*UNKNOWN*'

final['category'] = final['id'].apply(newId)

if '*UNKNOWN*' in final['category'].values:
```

```
    print("unknowns found")
else:
    print("All 24 hours have a category")

#add a counter for the 3 categories (visualisations)
final['counter'] = final.groupby('category').cumcount() + 1


# create a patient dictionary to map each unique ID
patient = {id:index + 1 for index, id in enumerate(data['id'].unique())}
# map the ID col to the pateientID using the dictionary values
final['patientID'] = final['id'].map(patient)

num_records = len(final)
print(f"Number of records in dataframe: {num_records}")

#Ensure 24 hours of entries for all particpants
if num_records % 1440 == 0:
  print("Number of records is divisible by 1440 with no remainder")
else:
  print("Number of records is NOT divisible by 1440")

#create a segment category based on the time
def seg(hr):
    if hr < 4:
        return '00:00-04:00'
    elif hr > 3 and hr < 8:
        return '04-08:00'
    elif hr > 7 and hr < 12:
        return '08-12:00'
    elif hr > 11 and hr < 16:
        return '12-16:00'
    elif hr > 15 and hr < 20:
        return '16-20:00'
    elif hr > 19 and hr < 24:
        return '20-24:00'
    else:
        return '*UNKNOWN*'

final['segment'] = final['hour'].apply(seg)

#all data is segmented?
if '*UNKNOWN*' in final['segment'].values:
    print("unknowns segments found")
else:
    print("All ids have segments")




# ============================================================================

# create features for 24 hour data
grouped = final.groupby(['id','date'])
newData = grouped.agg({'activity': ['mean', 'std', lambda x: (x == 0).mean(), kurtosis]})
newData = newData.reset_index()

# create a new dataframe with only the first 13 dates for each id
#newData = newData.groupby('id').head(13)
#remainder = len(newData) % 13

# Check if there is a person not 13 days
#if remainder != 0:
```

```
#    print(f"There are {remainder} records that do not have exactly 13 dates for each patient ID.")
#else:
#    print("All records have exactly 13 dates for each patient ID.")

newData.columns = ['id','date','f.mean', 'f.sd', 'f.propZeros', 'f.kurtosis']

newData['class'] = newData['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
newData = newData[['id','date','f.mean','f.sd','f.propZeros','f.kurtosis','class']]
newData = newData.loc[~((newData['f.mean'] == 0) or (newData['f.sd'] == 0))]
#newData = newData.loc[~( (newData['f.kurtosis'] == 0))]
newData = newData.loc[~((newData['f.mean'] == 0) & (newData['f.sd'] == 0))]
newData.dropna(inplace=True)
print((newData['id'].nunique()))


#create features for 4 hourly data
grouped = final.groupby(['id','date','segment'])
segmented = grouped.agg({'activity': ['mean', 'std', lambda x: (x == 0).mean(), kurtosis]})
segmented = segmented.reset_index()

segmented.columns = ['id','date','segment','f.mean', 'f.sd', 'f.propZeros', 'f.kurtosis']

segmented['class'] = segmented['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
segmented = segmented[['id','date','segment','f.mean','f.sd','f.propZeros','f.kurtosis','class']]
segmented= segmented.loc[~((segmented['f.mean'] == 0) & (segmented['f.sd'] == 0))]
#segmented = segmented.loc[~((segmented['f.propZeros'] == 0) & (segmented['f.kurtosis'] == 0))]
segmented.dropna(inplace=True)
print((segmented['id'].nunique()))


# Create a daytime and nighttime dataframe
Daysegmented = final.loc[~(final['segment'].isin(['00:00-04:00','04-08:00','20-24:00']))]
Nightsegmented = final.loc[~(final['segment'].isin(['08-12:00','12-16:00','16-20:00']))]



#create features for daytime data 8am - 8pm hour data
grouped = Daysegmented.groupby(['id','date'])
dayData = grouped.agg({'activity': ['mean', 'std', lambda x: (x == 0).mean(), kurtosis]})
dayData = dayData.reset_index()

dayData.columns = ['id','date','f.mean', 'f.sd', 'f.propZeros', 'f.kurtosis']

dayData['class'] = dayData['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
dayData = dayData[['id','date','f.mean','f.sd','f.propZeros','f.kurtosis','class']]
dayData = dayData.loc[~((dayData['f.mean'] == 0) & (dayData['f.sd'] == 0))]
dayData.dropna(inplace=True)
#dayData = newData.loc[~((dayData['f.propZeros'] == 0)  & (dayData['f.kurtosis'] == 0))]
print((dayData['id'].nunique()))


#create features for night data 8pm - 8am hour data
grouped = Nightsegmented.groupby(['id','date'])
nightData = grouped.agg({'activity': ['mean', 'std', lambda x: (x == 0).mean(), kurtosis]})
nightData = nightData.reset_index()

nightData.columns = ['id','date','f.mean', 'f.sd', 'f.propZeros', 'f.kurtosis']

nightData['class'] = nightData['id'].str[:5].apply(lambda x: 1 if x == 'condi' else (0 if x == 'contr' else 2))
nightData = nightData[['id','date','f.mean','f.sd','f.propZeros','f.kurtosis','class']]
nightData = nightData.loc[~((nightData['f.mean'] == 0) & (nightData['f.sd'] == 0))]
#nightData = nightData.loc[~((nightData['f.propZeros'] == 0) & (nightData['f.kurtosis'] == 0))]
nightData.dropna(inplace=True)
```

```
print((nightData['id'].nunique()))

# ============================================================================
def newId(idVal):
    if idVal[:5] == 'condi':
        return 'Depressive'
    elif idVal[:5] == 'patie':
        return 'Schizophrenic'
    elif idVal[:5] == 'contr':
        return 'Control'
    else:
        return '*UNKNOWN*'

newData['category'] = newData['id'].apply(newId)
segmented['category'] = segmented['id'].apply(newId)
dayData['category'] = dayData['id'].apply(newId)
nightData['category'] = nightData['id'].apply(newId)

if '*UNKNOWN*' in newData['category'].values:
   print("unknowns found")
else:
   print("All 24 hours have a category")
if '*UNKNOWN*' in dayData['category'].values:
   print("unknowns found")
else:
   print("All daytime data have a category")
if '*UNKNOWN*' in nightData['category'].values:
   print("unknowns found")
else:
   print("All nighttime data have a category")
if '*UNKNOWN*' in segmented['category'].values:
   print("unknowns found")
else:
   print("All segment data has a category")
newData['counter'] = newData.groupby('category').cumcount() + 1
segmented['counter'] = segmented.groupby('category').cumcount() + 1
dayData['counter'] = dayData.groupby('category').cumcount() + 1
nightData['counter'] = nightData.groupby('category').cumcount() + 1
#
# # create a patient dictionary to map each unique ID
patient = {id:index + 1 for index, id in enumerate(newData['id'].unique())}
patient = {id:index + 1 for index, id in enumerate(dayData['id'].unique())}
patient = {id:index + 1 for index, id in enumerate(nightData['id'].unique())}
patient = {id:index + 1 for index, id in enumerate(segmented['id'].unique())}
# # map the ID col to the pateientID using the dictionary values
newData['patientID'] = newData['id'].map(patient)
segmented['patientID'] = segmented['id'].map(patient)
dayData['patientID'] = dayData['id'].map(patient)
nightData['patientID'] = nightData['id'].map(patient)
# ============================================================================

#print(newData)
#print(segmented)
#print(dayData)
#print(nightData)
print("***  All 3 groups Baseline input file created for 24 hr of data only ***")
print("***  Features created for 4 hourly segments/ Daytime and Nighttime ****")


#create output csvs for use in later scripts
newData.to_csv('C:/mtu/project/24HrFeaturesk.csv', index=False)
segmented.to_csv('C:/mtu/project/4HrFeaturesk.csv', index=False)
nightData.to_csv('C:/mtu/project/NightFeaturesk.csv', index=False)
```

```
dayData.to_csv('C:/mtu/project/DayFeaturesk.csv', index=False)


#creating a file of all feature sets
hr = 'C:/mtu/project/24HrFeaturesk.csv'
ni = 'C:/mtu/project/NightFeaturesk.csv'
dy = 'C:/mtu/project/DayFeaturesk.csv'
print("***    Merging features into a single pandas dataframe  ***")
newSet= pd.concat(map(pd.read_csv, [hr,ni,dy]), ignore_index=True)
newSet.to_csv('C:/mtu/project/AllFeatureReadings.csv', index=False)

# ============================================================================
# Plots
#----------------------------------------------------------------------------

# Plot with comaparison of particpants from each caetgory with similar demographics
###################### comapre 3 females 40-44 ###################################
extract18 = final.query("id == 'condition_18'").head(18720)   #id 64 - 20160 rows 14 days
extract8 = final.query("id == 'control_8'").head(18720)      #id 31 - 27360 rows 19
extractp8 = final.query("id == 'patient_8'").head(18720)     #id 53 - 27360 rows 19

# Concatenate the data into a new dataframe
extract = pd.concat([extract18, extract8, extractp8], keys=['control_8','condition_18', 'patient_8'])
#print(extract18)
#print(extract8)
#print(extractp8)
# Compute the mean activity by category and hour
grouped = extract.groupby(['category', 'hour'])['activity'].mean().reset_index()

# Pivot the data to a wide format
pivoted = grouped.pivot(index='hour', columns='category', values='activity')

# Plot the data
plt.plot(pivoted.index, pivoted['Control'], label='Control 8')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive 18')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic 8')
plt.xticks(range(24), [f'{h:02d}' for h in range(24)])
plt.xlabel(' 24 hour cycle')
plt.ylabel('Average activity rate')
plt.title('Average 24 hour activity of female aged 40-44 from each group')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('females40-44.png', dpi=300)
plt.show()

#boxplot
Condition18 = extract18.groupby(['category', 'hour'])['activity'].mean()
Control8 = extract8.groupby(['category', 'hour'])['activity'].mean()
Patient8 = extractp8.groupby(['category', 'hour'])['activity'].mean()
activ = [Condition18,Control8,Patient8]
fig, ax = plt.subplots()
ax.boxplot(activ)
ax.set_xticklabels(['Depressive 18','Control 8','Schizophrenic 8'])
ax.set_ylabel('Daily average activity')
ax.set_title("Daily average activities of female aged 40-44 from each group")
plt.show()


# ======================3 females 50-59====================================
extract5 = final.query("id == 'condition_5'").head(18720)    #id73 - 20160 rows 14 days
extract27 = final.query("id == 'control_27'")    #id20 - 18720 rows 13 days
extract12 = final.query("id == 'patient_12'")    #id36 - 18720 rows 13 days
```

```
# Concatenate the data into a new dataframe
extract = pd.concat([extract5, extract27, extract12], keys=['control_27','condition_5', 'patient_12'])
#print(extract5)
#print(extract27)
#print(extract12)
# Compute the mean activity by category and hour
grouped = extract.groupby(['category', 'hour'])['activity'].mean().reset_index()

# Pivot the data to a wide format
pivoted = grouped.pivot(index='hour', columns='category', values='activity')

# Plot the data
plt.plot(pivoted.index, pivoted['Control'], label='Control 27')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive 5')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Shizophrenic 12')



plt.xticks(range(24), [f'{h:02d}' for h in range(24)])
plt.xlabel(' 24 hour cycle')
plt.ylabel('Average activity rate')
plt.title('Average 24 hour activity of female aged 50-59 from each group')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('females50-59.png', dpi=300)
plt.show()

#boxplot
Condition5 = extract5.groupby(['category', 'hour'])['activity'].mean()
Control27 = extract27.groupby(['category', 'hour'])['activity'].mean()
Patient12 = extract12.groupby(['category', 'hour'])['activity'].mean()
activ = [Condition5,Control27,Patient12]
fig, ax = plt.subplots()
ax.boxplot(activ)
ax.set_xticklabels(['Depressive 5','Control 27','Schizophrenic 12'])
ax.set_ylabel('Daily average activity')
#ax.set_xlabel('Patient 22 v Control 25')
ax.set_title("Daily average activities of female aged 50-59 from each group")
plt.show()


# ======================= 3 males aged 30-34 ======================================
extract20 = final.query("id == 'condition_20'").head(18720)  #id67 24480 17 days
extract5 = final.query("id == 'control_5'").head(18720)      #id28 46080  32 days
extract11 = final.query("id == 'patient_11'")                #id35 18720 13 days

# Concatenate the data into a new dataframe
extract = pd.concat([extract20, extract5, extract11], keys=['control_5', 'condition_20', 'patient_11'])
#print(extract20)
#print(extract5)
#print(extract11)
# Compute the mean activity by category and hour
grouped = extract.groupby(['category', 'hour'])['activity'].mean().reset_index()

# Pivot the data to a wide format
pivoted = grouped.pivot(index='hour', columns='category', values='activity')

plt.plot(pivoted.index, pivoted['Control'], label='Control 5')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive 20')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic 11')

plt.xticks(range(24), [f'{h:02d}' for h in range(24)])
plt.xlabel(' 24 hour cycle')
```

```
plt.ylabel('Average activity rate')
plt.title('Average 24 hour activity of male aged 30-34 from each group')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('males30-34.png', dpi=300)
plt.show()

Condition20 = extract20.groupby(['category', 'hour'])['activity'].mean()
Control5    = extract5.groupby(['category', 'hour'])['activity'].mean()
Patient11   = extract11.groupby(['category', 'hour'])['activity'].mean()
activ = [Condition20,Control5,Patient11]
fig, ax = plt.subplots()

ax.boxplot(activ)



ax.set_xticklabels(['Depressive 20','Control 5','Schizophrenic 11'])
ax.set_ylabel('Daily average activity')
ax.set_title("Daily average activities of male aged 30-34 from each group")
plt.show()
# ============================================================================
############# plot of 24 hour averages ###############################
#plot of averages of activity from midnight to midnight by hour
grouped = final.groupby(['category', 'hour'])['activity'].mean().reset_index()
pivoted = grouped.pivot(index='hour', columns='category', values='activity')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')

plt.xticks(range(24), [f'{h:02d}' for h in range(24)])

plt.xlabel(' 24 hour period hourly')
plt.ylabel('Average of daily Activity')
plt.title('Average Activity midnight to midnight per hour by Category')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('AverageActivityPerHour.png', dpi=300)
plt.show()


#plot of 24 hours by minute
grouped = final.groupby(['category', 'minute'])['activity'].mean().reset_index()
pivoted = grouped.pivot(index='minute', columns='category', values='activity')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')

plt.xlabel(' 24 hour period in minutes')
plt.ylabel('Average Activity')
plt.title('Average Activity per minute by Category')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('AverageActivityPerMinute.png', dpi=300)
plt.show()
#################### Plot of averages 4 hour segments ########################
#plot of 4 hourly averages
grouped = segmented.groupby(['category', 'segment'])['f.mean'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.mean')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
```

```
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Mean Activity of averages')
plt.title('Mean Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('MeanActivityPer4hrSegement.png', dpi=300)
plt.show()

# plot of sd averages every 4 hours
grouped = segmented.groupby(['category', 'segment'])['f.sd'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.sd')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Standard Deviation Activity')
plt.title('SD  Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('SD-ActivityPer4hrSegement.png', dpi=300)
plt.show()

#plot of propZeros every 4 hours
grouped = segmented.groupby(['category', 'segment'])['f.propZeros'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.propZeros')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Proportion of Zero values Activity')
plt.title('Prop of Zero vals  Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('PropZeros-ActivityPer4hrSegement.png', dpi=300)
plt.show()

#plot of kurtosis every 4 hours
grouped = segmented.groupby(['category', 'segment'])['f.kurtosis'].mean().reset_index()
pivoted = grouped.pivot(index='segment', columns='category', values='f.kurtosis')
plt.plot(pivoted.index, pivoted['Control'], label='Control')
plt.plot(pivoted.index, pivoted['Depressive'], label='Depressive')
plt.plot(pivoted.index, pivoted['Schizophrenic'], label='Schizophrenic')
plt.rcParams["figure.autolayout"] = True
plt.xlabel(' 24 hour period 4 hour segments')
plt.ylabel('Kurtosis Activity')
plt.title('Kurtosis vals  Activity in 4 hourly segments over 24 hours')
plt.legend()
plt.gcf().set_size_inches(12, 6)
plt.savefig('Kurtosis-ActivityPer4hrSegement.png', dpi=300)
plt.show()
```

6. Leave one group out

```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, LeavePGroupsOut, TimeSeriesSplit
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classification_report, precision_score,
f1_score
import itertools
from sklearn.model_selection import LeaveOneGroupOut
import pandas as pd
import xgboost as xgb
import lightgbm as lgb
from imblearn.over_sampling import SMOTE
from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')
color = sns.color_palette()

#file='allFeatureReadings.csv'
#file = '4HrFeaturesk.csv'
#file='DayFeaturesk.csv'
#file='NightFeaturesk.csv'
file = '24HrFeaturesk.csv'
df = pd.read_csv(file)



X = df[['f.mean', 'f.sd', 'f.propZeros', 'f.kurtosis']]
y = df['class'].values
groups = df['patientID'].values

#  classifiers
rfc = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=2018)
dtc = DecisionTreeClassifier(max_depth=10, random_state=2018)
xgbc = xgb.XGBClassifier(n_estimators=200, max_depth=10, random_state=2018)
# pipeline to impute missing values and scale the continuous variables
pipeline = make_pipeline(SimpleImputer(strategy='median'), StandardScaler())


logo = LeaveOneGroupOut()

#predicted and actual labels
predicted_labels = np.array([])
true_labels = np.array([])


for train_index, test_index in logo.split(X, y, groups):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
    # Fit the pipeline a
    X_train = pipeline.fit_transform(X_train)
    X_test = pipeline.transform(X_test)

    # Balance the target
    smote = SMOTE(random_state=2018)
    X_train, y_train = smote.fit_resample(X_train, y_train)

    rfc.fit(X_train, y_train)


    predicted_labels = np.append(predicted_labels, rfc.predict(X_test))
    true_labels = np.append(true_labels, y_test)


accuracy = accuracy_score(true_labels, predicted_labels)
print("Random Forest Accuracy:", accuracy)
cmrf = confusion_matrix(true_labels, predicted_labels)
print("Confusion Matrix Random Forest:")
print(cmrf)
crrf = classification_report(true_labels, predicted_labels)
print("Classification Report Random Forest:")
print(crrf)

cmrf = confusion_matrix(true_labels, predicted_labels)
#print(cmdt)
#print("Decision Tree - Classification Report:")
#print(classification_report(y_test, dtc_pred))
cmrf = np.array(cmrf)
label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(cmrf, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
            xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Random Forest')
plt.show()

report = classification_report(true_labels, predicted_labels, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
    if key in ['accuracy', 'macro avg', 'weighted avg']:
        continue
    data['precision'].append(value['precision'])
    data['recall'].append(value['recall'])
    data['f1-score'].append(value['f1-score'])
    data['support'].append(value['support'])


label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
            xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
            yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for Random Forest')
plt.show()



for train_index, test_index in logo.split(X, y, groups):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
    X_train = pipeline.fit_transform(X_train)
    X_test = pipeline.transform(X_test)

    # Balance the target
    smote = SMOTE(random_state=2018)
    X_train, y_train = smote.fit_resample(X_train, y_train)

    dtc.fit(X_train, y_train)


    predicted_labels = np.append(predicted_labels, dtc.predict(X_test))
    true_labels = np.append(true_labels, y_test)


accuracy = accuracy_score(true_labels, predicted_labels)
print("Decision Tree Accuracy:", accuracy)
cmdt = confusion_matrix(true_labels, predicted_labels)
print("Confusion Matrix Decision Tree:")
print(cmdt)
cr = classification_report(true_labels, predicted_labels)
print("Classification Report Decision Tree:")
print(cr)

cmdt = confusion_matrix(true_labels, predicted_labels)
cmdt = np.array(cmrf)
label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1) # Adjust to fit labels within the plot area
sns.heatmap(cmdt, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
            xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Decision Tree')
plt.show()

report = classification_report(true_labels, predicted_labels, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
    if key in ['accuracy', 'macro avg', 'weighted avg']:
        continue
    data['precision'].append(value['precision'])
    data['recall'].append(value['recall'])
    data['f1-score'].append(value['f1-score'])
    data['support'].append(value['support'])


label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
            xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
            yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for Decision Tree')
plt.show()

for train_index, test_index in logo.split(X, y, groups):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    X_train = pipeline.fit_transform(X_train)
    X_test = pipeline.transform(X_test)

    smote = SMOTE(random_state=2018)
```

```
    X_train, y_train = smote.fit_resample(X_train, y_train)

    xgbc.fit(X_train, y_train)

    predicted_labels = np.append(predicted_labels, xgbc.predict(X_test))
    true_labels = np.append(true_labels, y_test)

accuracy = accuracy_score(true_labels, predicted_labels)
print("XG Boost Accuracy:", accuracy)
cmxg = confusion_matrix(true_labels, predicted_labels)
print("Confusion Matrix XG Boost:")
print(cmxg)
cr = classification_report(true_labels, predicted_labels)
print("Classification Report XG Boost:")
print(cr)

cmxg = confusion_matrix(true_labels, predicted_labels)
cmxg = np.array(cmxg)
label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1)
sns.heatmap(cmxg, annot=True, annot_kws={"size": 16}, cmap="YlGnBu", fmt='g',
            xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for XG Boost')
plt.show()


report = classification_report(true_labels, predicted_labels, output_dict=True)
data = {'precision':[], 'recall':[], 'f1-score':[], 'support':[]}
for key, value in report.items():
    if key in ['accuracy', 'macro avg', 'weighted avg']:
        continue
    data['precision'].append(value['precision'])
    data['recall'].append(value['recall'])
    data['f1-score'].append(value['f1-score'])
    data['support'].append(value['support'])

label_names = ['Control', 'Depression', 'Schizophrenia']
sns.set(font_scale=1)
fig, ax = plt.subplots(figsize=(8, 4))
sns.heatmap(pd.DataFrame(data), annot=True, cmap='Blues', fmt='.2f',
            xticklabels=['Precision', 'Recall', 'F1-score', 'Support'],
            yticklabels=['Control', 'Depression', 'Schizophrenia'], ax=ax)
plt.title('Classification Report for XG Boost')
plt.show()
```