Activity Context reconition

report of implementation

Yedhu prasad

32072154

Table of Contents

[INTRODUCTION 2](#_Toc135659685)

[DATASET DESCRIPTION 2](#_Toc135659686)

[EXPLORATORY DATA ANALYSIS 2](#_Toc135659687)

[CLASSIFICATION MODELS 5](#_Toc135659688)

[PROGRAM EXECUTION 5](#_Toc135659689)

[REFLECTION 6](#_Toc135659690)

[APPENDIX 7](#_Toc135659691)

[REFERENCE 16](#_Toc135659692)

# INTRODUCTION

As a data engineer hired by a fitness company, my task is to develop an intelligent machine learning model for their mobile application. The objective is to enable the application to recognize people's actions using sensory data from their mobile phones. To accomplish this, I have been provided with a labelled historical activity context dataset captured from smartphone sensors. However, the dataset presents a challenge due to its low-level nature, making it difficult to generalize and extract meaningful insights. To address this challenge, I will utilize statistical feature extraction methods. These methods will help in extracting relevant and significant information from the dataset, which will then be used for implementing and evaluating the activity recognition model.

The ultimate goal of this project is to develop a robust and accurate activity recognition model that enhances the capabilities of the fitness application. To achieve this, we will implement three algorithms: Support Vector Machine (SVM), Random Forest, and Multilayer Perceptron (MLP). These algorithms have been chosen for their effectiveness in pattern recognition and classification tasks.

## DATASET DESCRIPTION

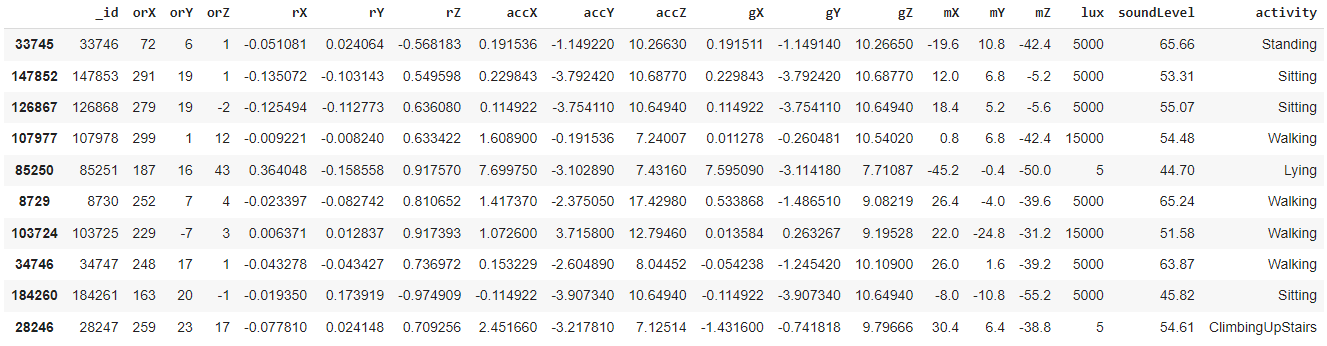
The provided dataset is a historical activity context data of individuals, captured from smartphone's built-in sensors. The dataset encompasses information from 7 different sensors. Among the sensors, 5 of them (orientation, rotation, accelerometer, gyroscope, and magnetic sensors) have 3 columns each, representing the x, y, and z axes of the sensors. On the other hand, the sound and light sensors have only one axis each, providing measurements related to sound intensity and ambient light levels. The dataset consists of 205,520 rows and 19 columns, including the '\_id' column and the 'activity' column. Each row represents a specific observation or data point, while the columns contain various attributes and measurements associated with the captured sensor data. Statistical feature extraction methods will be used to extract meaningful information for activity recognition.

# EXPLORATORY DATA ANALYSIS

EDA, or exploratory data analysis, is an important step in comprehending and learning from the dataset. EDA contributes to the informing of later modelling decisions by assessing the properties of the dataset, the interactions between variables, and the identification of patterns or anomalies.

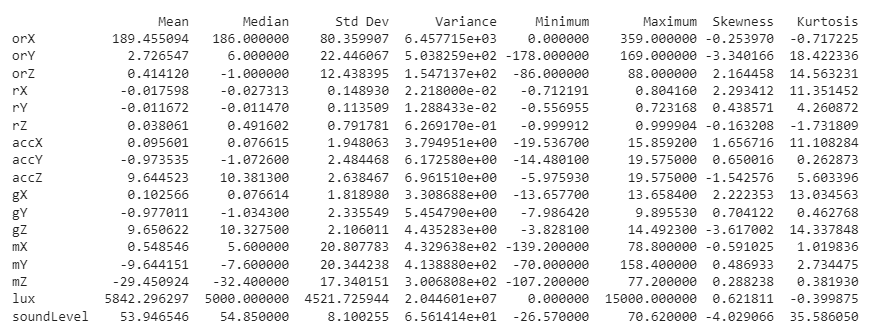
**Dataset Overview:**

Exploring a random sample from the data.



We have identified that the '\_id' column is not relevant for our analysis and can be dropped from the dataset. This column does not provide any meaningful information for the activity recognition task.

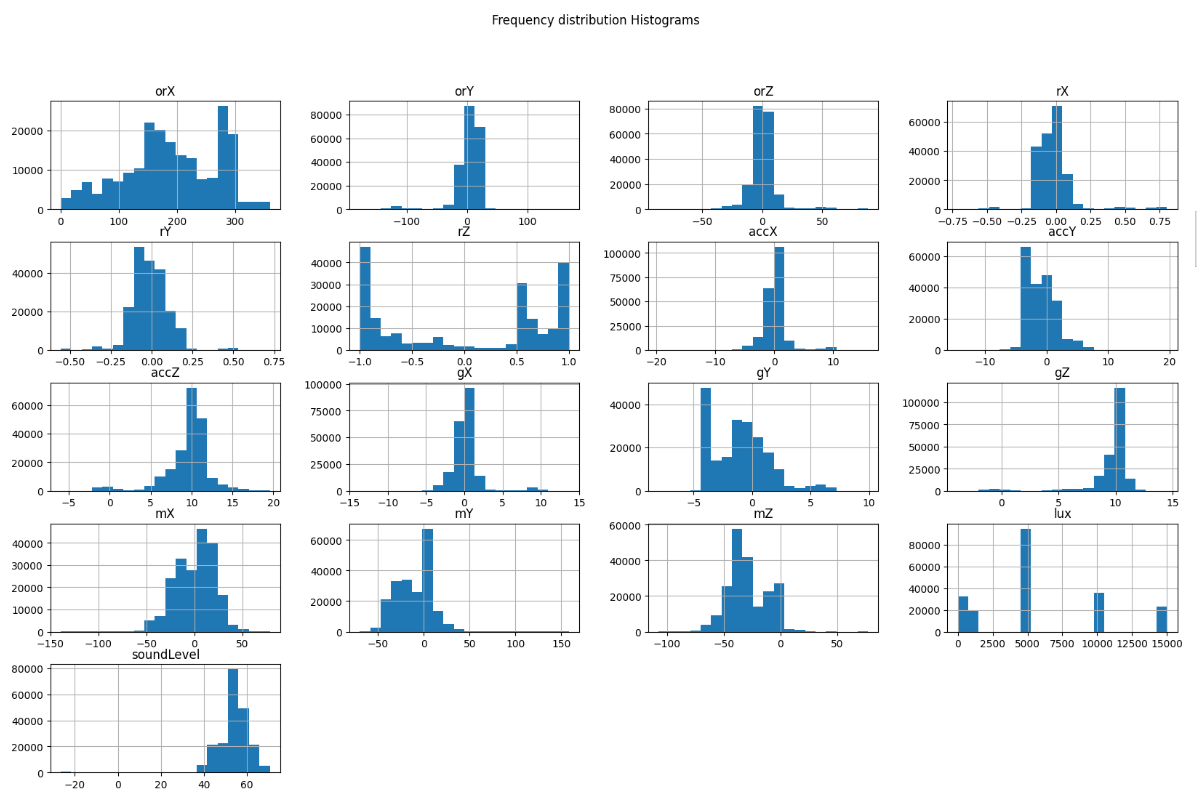
**Descriptive Statistics:**



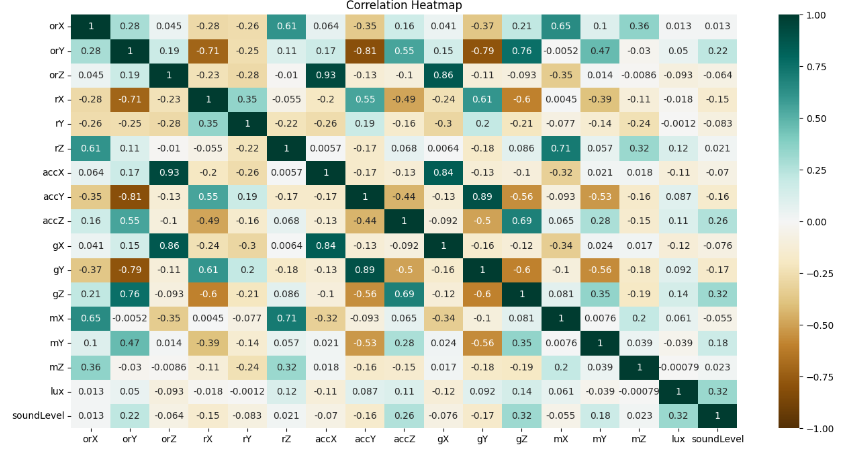
We can make the following inferences about the distributions of the variables:

* orX, orZ, rY, accY, mY, mZ, lux, and soundLevel have skewness values close to 0, indicating approximately symmetric distributions.
* orY, rX, accX, gX, accZ, and gZ have moderate to high skewness values, suggesting significant skewness in their distributions. orY, rX, and gZ are left-skewed, while accX, gX, and accZ are right-skewed.
* orX, rZ, mX, mY, and mZ have kurtosis values close to 0, indicating distributions that are close to a normal distribution.
* orY, orZ, rX, rY, accX, accZ, gX, accY, gY, gZ, lux, and soundLevel have kurtosis values greater than 0, indicating distributions with heavier tails and a more pronounced peak compared to a normal distribution. These distributions are leptokurtic.
* soundLevel stands out with a very high kurtosis value of 35.586050, indicating an extremely heavy-tailed distribution.

The frequency distribution plots:



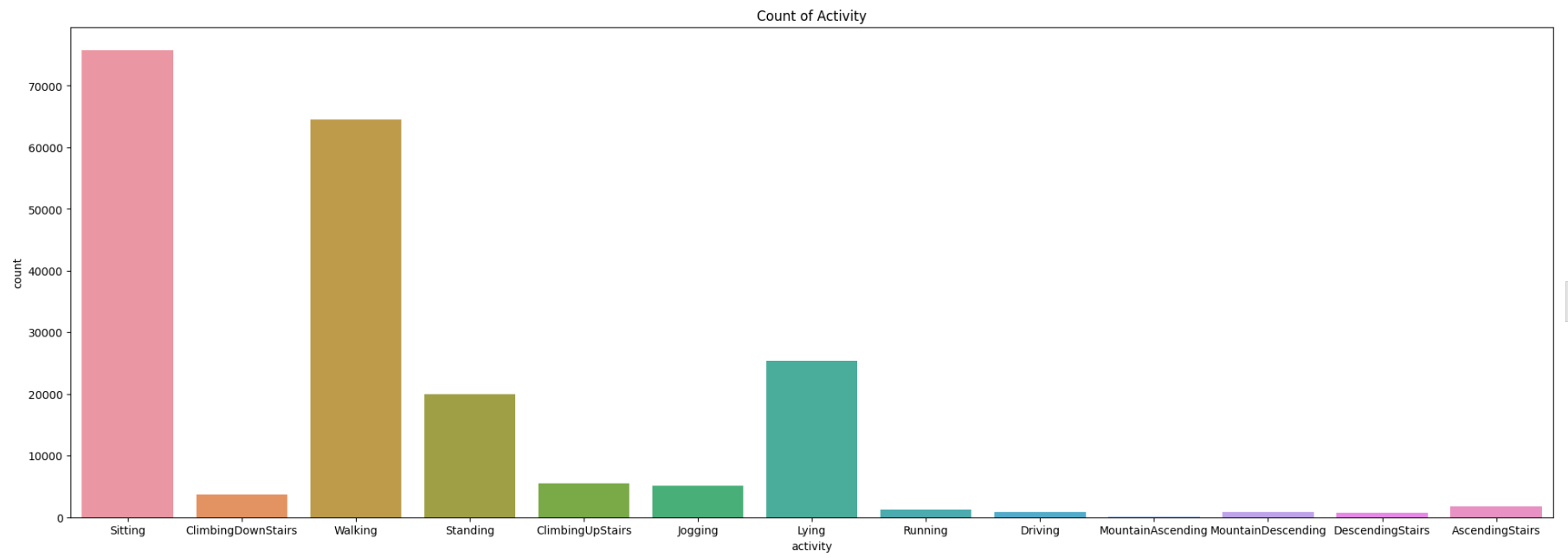
Correlation Heatmap:



Upon analysing the correlation heatmap, we observed that several variables in the dataset exhibit high correlation with each other. This suggests a strong linear relationship between these variables. Apart from these highly correlated variables, the correlation heatmap appears to follow the expected pattern, with no significant anomalies or irregularities.

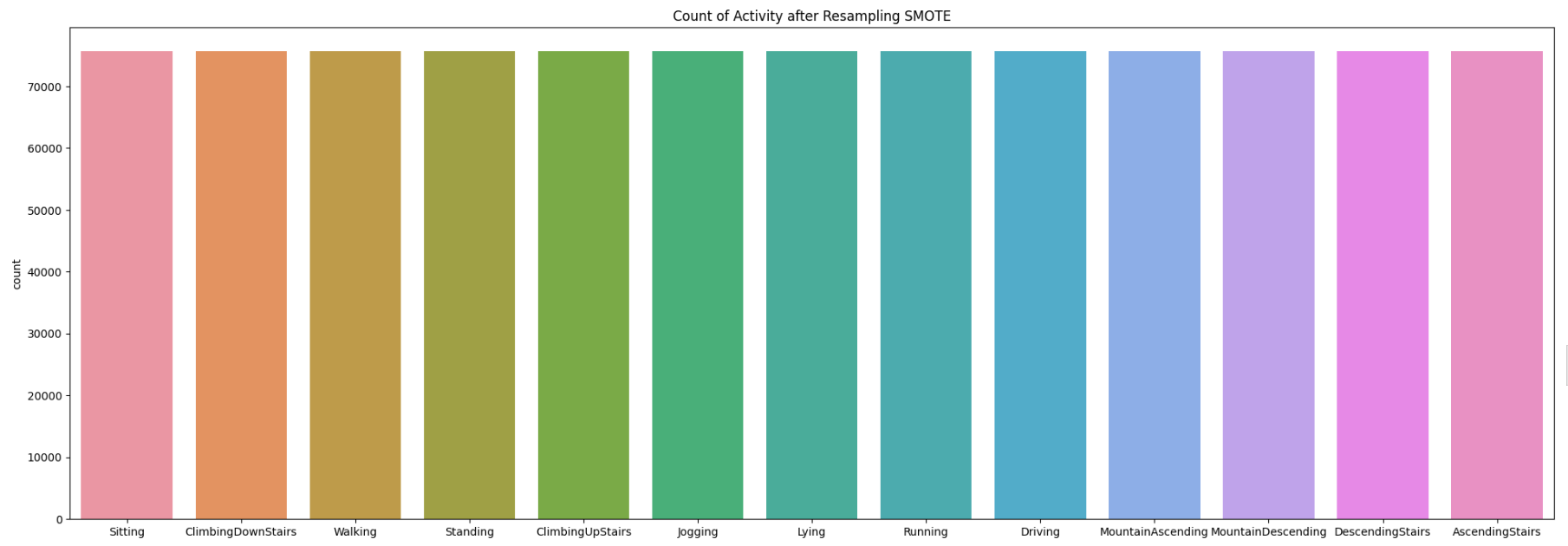
Check for the data balancing:

Frequency Distribution of Activity before balancing



During our analysis, we observed that the dataset is not balanced, meaning that there is an imbalance in the distribution of different activity classes. To address this issue, we will employ the SMOTE (Synthetic Minority Over-sampling Technique) oversampling technique.

Frequency Distribution of Activity before oversampling



After performing the data pre-processing steps, the next steps in our workflow involve splitting the dataset into training and testing data and applying feature scaling.

**Splitting the Data**: We will use the train\_test\_split technique to divide the training data and the testing data. The training data will be used to train the activity recognition model, while the testing data will be used to evaluate its performance. The data will be split in a stratified manner to ensure that the class distribution is maintained in both the training and testing sets.

**Feature Scaling**: To ensure that all features are on a similar scale and to avoid any bias that may be introduced by features with different magnitudes, we will apply feature scaling. In this case, we will use the StandardScaler, which standardizes the features by subtracting the mean and dividing by the standard deviation. This process ensures that all features have zero mean and unit variance.

# CLASSIFICATION MODELS

In our activity recognition model, we will implement three classification algorithms:

**Support Vector Machines (SVM):**

SVM is a powerful supervised learning algorithm used for both classification and regression tasks. It works by finding an optimal hyperplane that separates data points belonging to different classes in a high-dimensional space. SVM is known for its ability to handle complex decision boundaries and can handle both linear and non-linear classification problems.

**Random Forest:**

Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions. It creates a multitude of decision trees, each trained on a different subset of the data and using a random subset of features. The final prediction is made by aggregating the predictions of individual trees. Random Forest is known for its robustness, scalability, and ability to handle high-dimensional data with complex interactions between variables.

**Multilayer Perceptron (MLP):**

MLP is a type of artificial neural network that consists of multiple layers of interconnected nodes (neurons). It is a feedforward neural network, where information flows from the input layer through the hidden layers to the output layer. MLP is capable of learning complex relationships between inputs and outputs and can handle non-linear classification problems. It is trained using the backpropagation algorithm, which adjusts the weights of the connections to minimize the prediction error.

# PROGRAM EXECUTION

The program execution consists of the following files and notebooks:

**Files**

EDA.py: This file contains the functions related to exploratory data analysis (EDA). It includes essential EDA functions such as data pre-processing, visualization, and descriptive Statistics.

Optional\_EDA.py: This file contains additional optional statistical functions such as variance, median, mean, standard deviation, root mean square (RMS), zero crossing, sum of squares, and covariance.

**Notebooks**

test.ipynb: This notebook serves as the main program and is menu-driven. It imports the EDA.py file, Optional\_EDA.py file, and the three pickled models. It provides an interactive interface to execute different functionalities and perform tasks related to EDA and model evaluation.

EDA.ipynb: This notebook is used for creating and training the classification models. It includes the implementation of the SVM, Random Forest, and MLP algorithms. The trained models are then pickled and saved for later use.

**Pickled Models**

Three .pkl files(mlp\_model.pkl, rfc\_model.pkl, mlp\_model.pkl): These files contain the trained models that were created and saved in the EDA.ipynb notebook. The models correspond to the SVM, Random Forest, and MLP algorithms.

The test.ipynb notebook acts as the central module, importing the necessary modules and models to perform different tasks. It provides an interactive interface where users can choose to perform exploratory data analysis or evaluate the classification models. The EDA.py file contains the functions required for data preprocessing, visualization, and descriptive Statistics. The Optional\_EDA.py file includes additional statistical functions that users can choose to employ.

# REFLECTION

During the execution of this project, I encountered various challenges and learned valuable lessons. Initially, I performed data exploration and preprocessing, including handling null values, dropping unnecessary columns, and addressing data imbalance using SMOTE oversampling. However, I made a mistake by initially excluding important columns like 'lux' and 'soundlevel', but I quickly corrected it.

The next phase involved splitting the data and scaling it using StandardScaler. This step was relatively straightforward. However, the execution of the classification models proved to be more challenging. I first attempted to train an SVM model, but it took an unexpectedly long time, leading to multiple execution attempts and confusion. Eventually, I realized that training the model on a large dataset would naturally require significant time and computational resources. The same situation arose when training the Random Forest model, which initially appeared unresponsive before progressing after approximately 18 minutes. This experience taught me to be patient and understanding when working with computationally intensive tasks and large datasets. The entire process of training all three models and pickling them consumed approximately 7 hours, highlighting the importance of considering resource limitations and optimizing code execution. The optional statistics section was relatively easier to implement since pre-defined functions were readily available. With some research, I successfully incorporated the desired statistical operations into the code.

One of the major challenges came when I started working on the test module, where I encountered an unexpected issue. When importing the EDA.py and program files, they automatically executed without being called explicitly. This confusion was due to the absence of functions in the initial EDA.py file. To rectify this, I had to rewrite the entire EDA module, ensuring that functions were properly defined.

If I were to redo this assignment, I would avoid the mistakes I made, such as excluding important columns initially and encountering issues with code execution order. Additionally, I would seize the opportunity to explore and implement additional algorithms on the dataset, further expanding my knowledge and understanding of machine learning models.

# APPENDIX

**EDA.py**

IMPORT LIBRARIES

FUNCTION READ\_DATA(file\_path):

READ data from CSV file

RETURN data frame

FUNCTION EXPLORE\_DATA(data\_frame):

SAMPLE = randomly select 10 rows from data\_frame

PRINT SAMPLE

DESCRIPTION = compute descriptive statistics for data\_frame

PRINT DESCRIPTION

SHAPE = get the shape (number of rows and columns) of data\_frame

PRINT SHAPE

NULL\_COUNTS = count the number of null values in data\_frame

PRINT NULL\_COUNTS

INFO = get information about data\_frame

PRINT INFO

FUNCTION DROP\_COLUMNS(data\_frame):

DROP "\_id" column from data\_frame

RETURN modified data\_frame

FUNCTION DUPLICATE\_ROWS(data\_frame):

DUPLICATES = count the number of duplicate rows in data\_frame

PRINT DUPLICATES

FUNCTION DESCRIPTIVE\_STATISTICS(data\_frame):

COMPUTE mean, median, standard deviation, variance, minimum, maximum, skewness, and kurtosis for numeric columns in data\_frame

CREATE a data frame to store the statistics

PRINT the statistics

FUNCTION HISTOGRAMS(data\_frame):

PLOT histograms for each column in data\_frame

DISPLAY the plots

FUNCTION CORRELATION\_HEATMAP(data\_frame):

COMPUTE the correlation matrix for data\_frame

PLOT a heatmap of the correlation matrix

DISPLAY the heatmap

FUNCTION VARIABLE\_DEPENDENCY(data\_frame):

PLOT a scatterplot of the "orZ" and "accX" variables, with different colors for each activity

DISPLAY the plot

FUNCTION ACTIVITY\_FREQUENCY\_DISTRIBUTION(data\_frame):

COMPUTE the frequency distribution of the "activity" column in data\_frame

PLOT a bar chart of the frequency distribution

DISPLAY the chart

FUNCTION OVERSAMPLING(data\_frame):

SEPARATE the features and target variables from data\_frame

PERFORM oversampling using SMOTE

RETURN the oversampled features and target variables

FUNCTION RESAMPLED\_ACTIVITY\_FREQ(oversampled\_target):

COUNT the frequency of each activity in the oversampled target

PLOT a bar chart of the activity frequencies

DISPLAY the chart

FUNCTION SPLIT\_DATA(features, target):

SPLIT the features and target into training and testing sets

RETURN the training and testing sets

FUNCTION SCALE\_DATA(training\_features, testing\_features):

INITIALIZE a scaler

SCALE the training features using the scaler

SCALE the testing features using the scaler

RETURN the scaled training and testing features

FUNCTION RANDOM\_FOREST\_CLASSIFIER(training\_features, training\_target):

INITIALIZE a random forest classifier

TRAIN the classifier using the training features and target

RETURN the trained classifier

FUNCTION SVM\_CLASSIFIER(training\_features, training\_target):

INITIALIZE an SVM classifier

TRAIN the classifier using the training features and target

RETURN the trained classifier

FUNCTION MLP\_CLASSIFIER(training\_features, training\_target):

INITIALIZE an MLP classifier

TRAIN the classifier using the training features and target

RETURN the trained classifier

FUNCTION EVALUATE\_CLASSIFIER(test\_target, predicted\_target):

COMPUTE classification report for the predicted target compared to the test target

COMPUTE confusion matrix for the predicted target compared to the test target

PLOT the confusion matrix

DISPLAY the plot

FUNCTION SAVE\_MODEL(model, file\_path):

SAVE the model to a file using pickle

FUNCTION LOAD\_MODEL(file\_path):

LOAD a model from a file using pickle

RETURN the loaded model

**test.ipynb**

IMPORT LIBRARIES

class TestProgram:

def \_\_init\_\_(self):

self.df = None

FUNCTION READ\_DATA(self, file\_path):

self.df = READ\_CSV(file\_path)

PRINT "Data loaded successfully."

FUNCTION DISPLAY\_MENU(self):

WHILE True:

PRINT "\n--- Main Menu ---"

PRINT "1. Explore Data"

PRINT "2. Descriptive Statistics"

PRINT "3. Frequency distribution Histograms"

PRINT "4. Correlation Heatmap"

PRINT "5. Variable Dependency"

PRINT "6. Activity Frequency Distribution Before Balancing"

PRINT "7. Activity Frequency Distribution After Balancing"

PRINT "8. Optional Statistics"

PRINT "9. Models"

PRINT "10. Quit"

choice = INPUT("Enter your choice: ")

IF choice == "1":

CALL READ\_DATA()

CALL EXPLORE\_DATA(self.df)

ELSE IF choice == "2":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

CALL DESCRIPTIVE\_STATISTICS(self.df)

ELSE IF choice == "3":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

CALL HISTOGRAMS(self.df)

ELSE IF choice == "4":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

CALL CORRELATION\_HEATMAP(self.df)

ELSE IF choice == "5":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

CALL VARIABLE\_DEPENDENCY(self.df)

ELSE IF choice == "6":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

CALL ACTIVITY\_FREQUENCY\_DISTRIBUTION(self.df)

ELSE IF choice == "7":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

PRINT "Please wait Loading ..."

x\_resampled, y\_resampled = CALL OVERSAMPLING(self.df)

PRINT "Please wait Loading ..."

CALL RESAMPLED\_ACTIVITY\_FREQ(y\_resampled)

ELSE IF choice == "8":

CALL OPTIONAL\_STATISTICS\_MENU()

ELSE IF choice == "9":

CALL MENU\_MODEL()

ELSE IF choice == "10":

BREAK

ELSE:

PRINT "Invalid choice. Please try again."

FUNCTION MENU\_MODEL(self):

WHILE True:

PRINT "\n--- Choose a Model ---"

PRINT "1. Random Forest Classifier "

PRINT "2. Support Vector Machine Classifier"

PRINT "3. MLP Classifier"

PRINT "4. Back to Main Menu"

choice = INPUT("Enter your choice: ")

IF choice == "1":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

PRINT "Please wait Loading ..."

x\_resampled, y\_resampled = CALL OVERSAMPLING(self.df)

PRINT "Please wait splitting ..."

x\_train, x\_test, y\_train, y\_test = CALL SPLIT\_DATA(x\_resampled, y\_resampled)

PRINT "Please wait Scaling ..."

x\_train\_scaled, x\_test\_scaled = CALL SCALE\_DATA(x\_train, x\_test)

PRINT "Please wait predicting ..."

pickled\_rfc\_model = CALL LOAD\_MODEL('rfc\_model.pkl')

CALL EVALUATE\_CLASSIFIER(y\_test, pickled\_rfc\_model.predict(x\_test\_scaled))

ELSE IF choice == "2":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

PRINT "Please wait Loading ..."

x\_resampled, y\_resampled = CALL OVERSAMPLING(self.df)

PRINT "Please wait splitting ..."

x\_train, x\_test, y\_train, y\_test = CALL SPLIT\_DATA(x\_resampled, y\_resampled)

PRINT "Please wait Scaling ..."

x\_train\_scaled, x\_test\_scaled = CALL SCALE\_DATA(x\_train, x\_test)

PRINT "Please wait predicting ..."

pickled\_svm\_model = CALL LOAD\_MODEL('svm\_model.pkl')

CALL EVALUATE\_CLASSIFIER(y\_test, pickled\_svm\_model.predict(x\_test\_scaled))

ELSE IF choice == "3":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

PRINT "Please wait Loading ..."

x\_resampled, y\_resampled = CALL OVERSAMPLING(self.df)

PRINT "Please wait splitting ..."

x\_train, x\_test, y\_train, y\_test = CALL SPLIT\_DATA(x\_resampled, y\_resampled)

PRINT "Please wait Scaling ..."

x\_train\_scaled, x\_test\_scaled = CALL SCALE\_DATA(x\_train, x\_test)

PRINT "Please wait predicting ..."

pickled\_mlp\_model = CALL LOAD\_MODEL('mlp\_model.pkl')

CALL EVALUATE\_CLASSIFIER(y\_test, pickled\_mlp\_model.predict(x\_test\_scaled))

ELSE IF choice == "4":

BREAK

ELSE:

PRINT "Invalid choice. Please try again."

FUNCTION OPTIONAL\_STATISTICS\_MENU(self):

WHILE True:

PRINT "\n--- Optional Statistics Menu ---"

PRINT "1. Compute Variance "

PRINT "2. Compute Median"

PRINT "3. Compute Mean"

PRINT "4. Compute Standard Deviation"

PRINT "5. Compute Root Mean Square"

PRINT "6. Compute Zero Crossings"

PRINT "7. Compute Sum of Squares"

PRINT "8. Compute Covariance"

PRINT "9. Back to Main Menu"

choice = INPUT("Enter your choice: ")

IF choice == "1":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL VARIANCE(self.df)

ELSE IF choice == "2":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL MEDIAN(self.df)

ELSE IF choice == "3":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL MEAN(self.df)

ELSE IF choice == "4":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL STD(self.df)

ELSE IF choice == "5":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL RMS(self.df)

ELSE IF choice == "6":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL ZERO\_CROSSING(self.df)

ELSE IF choice == "7":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL SUM\_OF\_SQUARES(self.df)

ELSE IF choice == "8":

CALL READ\_DATA()

CALL DROP\_COLUMNS(self.df)

self.df = self.df.DROP('activity', axis=1)

PRINT CALL COVARIANCE(self.df)

ELSE IF choice == "9":

BREAK

ELSE:

PRINT "Invalid choice. Please try again."

IF \_\_name\_\_ == "\_\_main\_\_":

program = TestProgram()

program.DISPLAY\_MENU()

**Optional\_EDA.py**

IMPORT LIBRARIES

FUNCTION variance(data):

RETURN data.var()

FUNCTION median(data):

RETURN data.median()

FUNCTION mean(data):

RETURN data.mean(axis=0)

FUNCTION std(data):

RETURN data.std()

FUNCTION rms(data):

RETURN np.sqrt(np.mean(np.square(data)))

FUNCTION zero\_crossing(data):

zc = (np.diff(np.signbit(data), axis=0) != 0).sum()

RETURN zc

FUNCTION sum\_of\_squares(data):

RETURN np.sum(np.square(data))

FUNCTION covariance(data):

RETURN data.cov()

# REFERENCE

* “Imbalanced-Learn Documentation — Version 0.8.1.” *Imbalanced-Learn.org*, 2021, [imbalanced-learn.org/stable/](https://imbalanced-learn.org/stable/).
* Matplotlib. “Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation.” *Matplotlib.org*, 2012, [matplotlib.org/](https://matplotlib.org/).
* Numpy. “NumPy.” *Numpy.org*, 2009, [numpy.org/](https://numpy.org/).
* Pandas. “Python Data Analysis Library — Pandas: Python Data Analysis Library.” *Pydata.org*, 2018, [pandas.pydata.org/](https://pandas.pydata.org/).
* scikit-learn. “Scikit-Learn: Machine Learning in Python.” *Scikit-Learn.org*, 2019, [scikit-learn.org/stable/](https://scikit-learn.org/stable/).
* seaborn. “Seaborn: Statistical Data Visualization — Seaborn 0.9.0 Documentation.” *Pydata.org*, 2012, [seaborn.pydata.org/](https://seaborn.pydata.org/).