Faculty of Computer Science, Data Analysis Program

University Of Messina

Data Mining and Analytics Project Report (DataSet_Study2)

Professor: **Gennaro Tartarisco** Student**: Yahia Ahmed (532118)**

Academic Year 2023/2024

Table of Contents:

1- Introduction
2- Dataset Description
3- Project and Code Architecture
4- Libraries used and importing phase
5- Dataset Traversing and Exploration
6- Dataset Visualization
7- Data Preprocessing
8- Data Correlation
9- Principal Component Analysis
10- Feature Selection:
  - Random Forest Classifier
  - Decision Tree Classifier
  - Forward Sequential Selection
11- Machine Learning (Supervised Learning)
12- Conclusion

Introduction

Understanding the intricate relationship between emotions and physiological responses is crucial in various fields, from psychology to healthcare. In this report, we delve into a comprehensive data mining project that explores how different emotional states influence cardiovascular, respiratory, and electrodermal reactions in healthy individuals. The dataset under scrutiny contains features extracted from the physiological responses of 186 subjects, providing a rich source of information to uncover patterns and insights into the interplay between emotions and bodily responses.

Objectives

The primary objective of this report is to elucidate the methodologies and findings of the data mining project focused on analyzing physiological responses to emotional stimuli. Through meticulous exploration of the dataset and application of data mining techniques, our goal is to:

1. Identify key physiological markers indicative of emotional states, particularly high and low approach positive emotions, anger, and threat.
2. Investigate how these emotional states influence cardiovascular, respiratory, and electrodermal responses, as reflected in the extracted features.
3. Provide insights into the potential buffering effects of positive emotions on stress responses and the interaction between high approach positive affect and reactivity to anger or threat.

4. Offer implications for future research in affective computing, psychophysiology, and healthcare, aiming to enhance our understanding of the complex relationship between emotions and physiological responses.

Dataset Description

The dataset under analysis comprises physiological responses extracted from 186 healthy subjects, encompassing cardiovascular, respiratory, and electrodermal measures in response to varied emotional stimuli. The dataset is structured as follows:

- **Observations:** The dataset consists of **558** observations, corresponding to three distinct phases experienced by each subject: baseline, affect manipulation, and task.
- **Features:** A total of **19** features were extracted from the physiological responses, capturing various aspects of heart rate variability (HRV) and related indices. These features are categorized into three domains:
    1. **Time Domain Features:**
        - MeannNN [ms]
        - SDNN [ms]
        - RMSSD [ms]
        - Prc20NN [ms]
        - Prc80NN [ms]
        - PNN50
        - HTI
    2. **Frequency Domain Features:**
        - VLF [ms^2/Hz]

        - LF [ms^2/Hz]
        - HF [ms^2/Hz]
        - TP [ms^2/Hz]
        - LFHF
    3. **Non-linear Features:**
        - SD1
        - SD2
        - SD1SD2
        - DFA_alpha1
        - DFA_alpha2
        - ApEn
        - SampEn
- **Labels:** Each observation is associated with a label indicating the condition under which the physiological responses were recorded. The labels and their corresponding conditions are as follows:
    1. Baseline condition: -1
    2. Threat: 209
    3. Anger: 208
    4. Low-approach positive emotions: 308
    5. High-approach positive emotions: 309
    6. Neutral State: 108

This rich dataset offers a nuanced glimpse into the complex interplay between emotional states and physiological responses, providing a fertile ground for exploratory data analysis and predictive modeling in the domain of affective computing and psychophysiological research

Project and Code Architecture

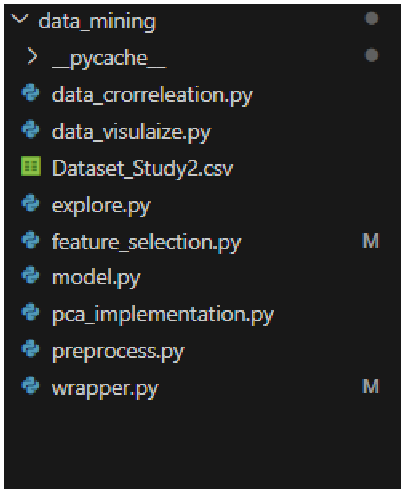
For this data science project, I've adopted a modular architecture to ensure code cleanliness, maintainability, and extensibility. Each distinct functionality is encapsulated within its own module, promoting clarity and ease of maintenance. The project consists of the following modules:

1. **wrapper.py**:
    - This serves as the main entry point for the project, orchestrating function calls to other modules.
2. **explore.py**:
    - Responsible for dataset exploration, providing insights into its structure and characteristics.
3. **data_visualize.py**:
    - Handles the visualization aspects of the dataset, aiding in data understanding and interpretation.
4. **preprocess.py**:
    - Manages data preprocessing tasks, ensuring data integrity and readiness for analysis.
5. **data_correlation.py**:
    - Focuses on generating correlation matrices and identifying relationships among dataset variables.
6. **pca_implementation.py**:
    - Implements Principal Component Analysis (PCA), reducing the dimensionality of the dataset while preserving its variance.
7. **feature_selection.py**:

- Executes various feature selection algorithms to identify the most relevant features for modeling.

8. **model.py**:
    - Facilitates data splitting into training and testing sets, as well as model training and evaluation, providing performance metrics and insights.

This modular design promotes code organization, facilitating seamless collaboration and future enhancements.



```
data_mining
  __pycache__
  data_crorreleation.py
  data_visulaize.py
  Dataset_Study2.csv
  explore.py
  feature_selection.py          M
  model.py
  pca_implementation.py
  preprocess.py
  wrapper.py                    M
```

Libraries used and importing phase

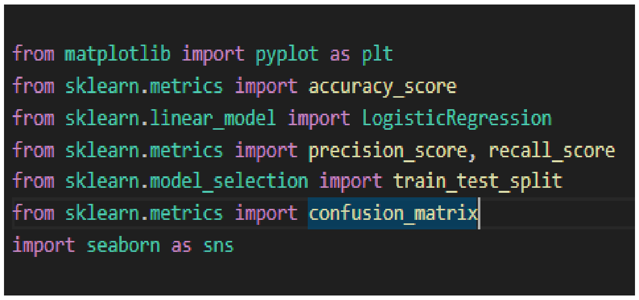
The main python libraries or frameworks I used were :

1- Pandas

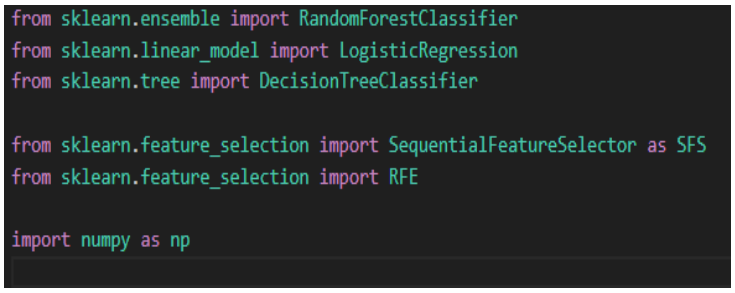
2- Numpy

3- Seaborn

4- Sklearn

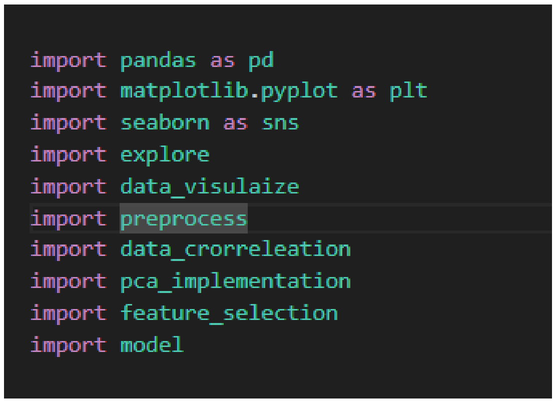


```
from matplotlib import pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
```



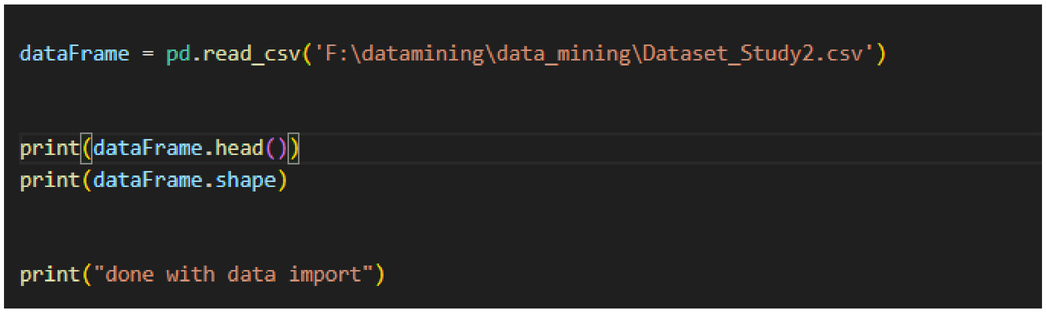
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.feature_selection import SequentialFeatureSelector as SFS
from sklearn.feature_selection import RFE

import numpy as np
```

And then in the wrapper file which is the entry point of the script. We import the other modules/files like:

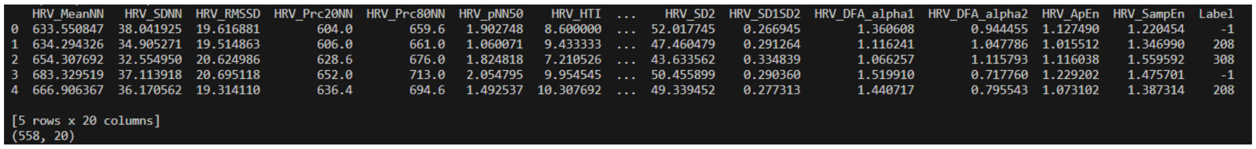


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import explore
import data_visulaize
import preprocess
import data_crorreleation
import pca_implementation
import feature_selection
import model
```

Then we start importing and reading the dataset, which is DataSet_Study2.csv and then we start printing some info about it like the first 5 rows and the size of the dataset:



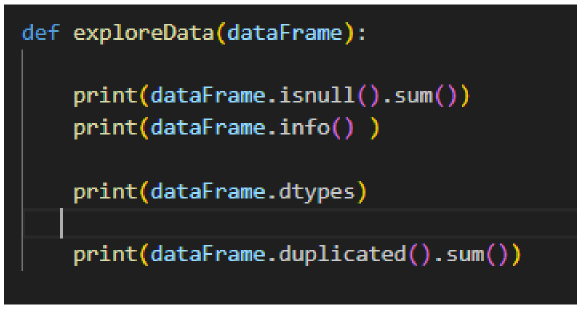
```
dataFrame = pd.read_csv('F:\datamining\data_mining\Dataset_Study2.csv')


print(dataFrame.head())
print(dataFrame.shape)


print("done with data import")
```

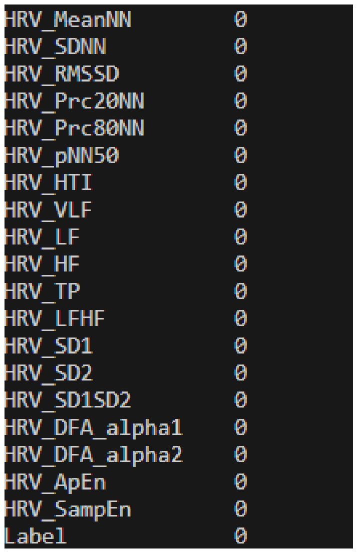


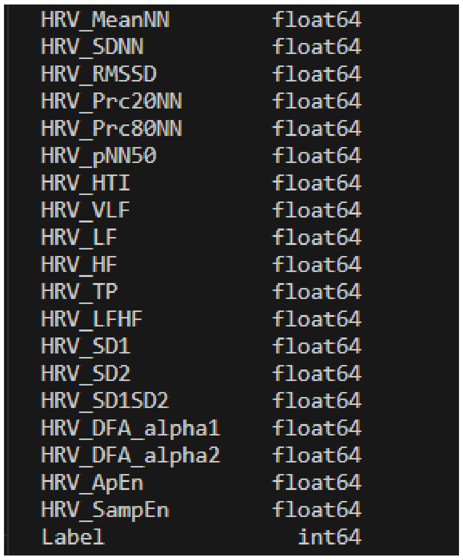
```
   HRV_MeanNN   HRV_SDNN  HRV_RMSSD  HRV_Prc20NN  HRV_Prc80NN  HRV_pNN50    HRV_HTI  ...    HRV_SD2  HRV_SD1SD2  HRV_DFA_alpha1  HRV_DFA_alpha2  HRV_ApEn  HRV_SampEn  Label
0  633.550847  38.041925  19.616881        604.0        659.6   1.902748   8.600000  ...  52.017745    0.266945        1.360608        0.944455  1.127490    1.220454     -1
1  634.294326  34.905271  19.514863        606.0        661.0   1.060071   9.433333  ...  47.460479    0.291264        1.116241        1.047786  1.015512    1.346990    208
2  654.307692  32.554950  20.624986        628.6        676.0   1.824818   7.210526  ...  43.633562    0.334839        1.066257        1.115793  1.116038    1.559592    308
3  683.329519  37.113918  20.695118        652.0        713.0   2.054795   9.954545  ...  50.455899    0.290360        1.519910        0.717760  1.229202    1.475701     -1
4  666.906367  36.170562  19.314110        636.4        694.6   1.492537  10.307692  ...  49.339452    0.277313        1.440717        0.795543  1.073102    1.387314    208

[5 rows x 20 columns]
(558, 20)
```

Dataset Traversing and Exploration

In the exploration phase I focused to check whether there are any missing data, null values or duplicated values which will may affect the efficiency of our calculations and analysis, so we used the functions isNUll to count the null values if any in each and every column, also we use the function duplicated() to check whether there are any duplicated values, and finally we used the dtypes attribute available in the dataframe variable to understand the type of the numerical data in our dataset.

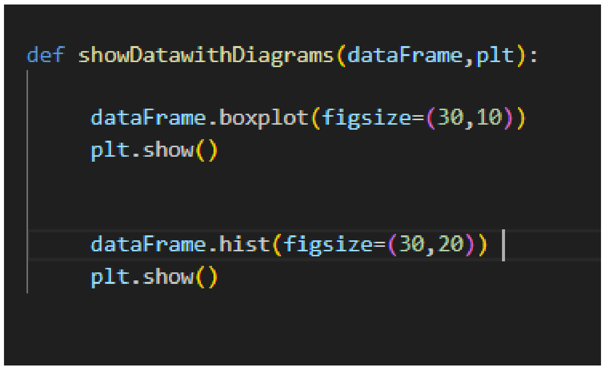

```
def exploreData(dataFrame):

    print(dataFrame.isnull().sum())
    print(dataFrame.info() )

    print(dataFrame.dtypes)

    print(dataFrame.duplicated().sum())
```


```
HRV_MeanNN          0
HRV_SDNN            0
HRV_RMSSD           0
HRV_Prc20NN         0
HRV_Prc80NN         0
HRV_pNN50           0
HRV_HTI             0
HRV_VLF             0
HRV_LF              0
HRV_HF              0
HRV_TP              0
HRV_LFHF            0
HRV_SD1             0
HRV_SD2             0
HRV_SD1SD2          0
HRV_DFA_alpha1      0
HRV_DFA_alpha2      0
HRV_ApEn            0
HRV_SampEn          0
Label               0
```

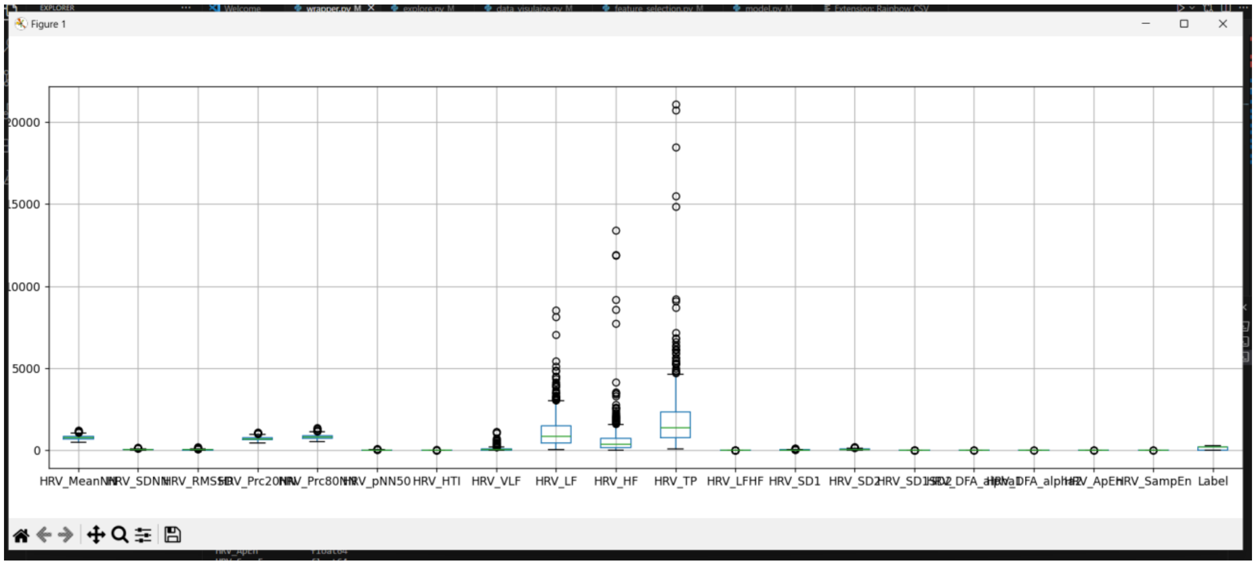

```
HRV_MeanNN          float64
HRV_SDNN            float64
HRV_RMSSD           float64
HRV_Prc20NN         float64
HRV_Prc80NN         float64
HRV_pNN50           float64
HRV_HTI             float64
HRV_VLF             float64
HRV_LF              float64
HRV_HF              float64
HRV_TP              float64
HRV_LFHF            float64
HRV_SD1             float64
HRV_SD2             float64
HRV_SD1SD2          float64
HRV_DFA_alpha1      float64
HRV_DFA_alpha2      float64
HRV_ApEn            float64
HRV_SampEn          float64
Label                 int64
```

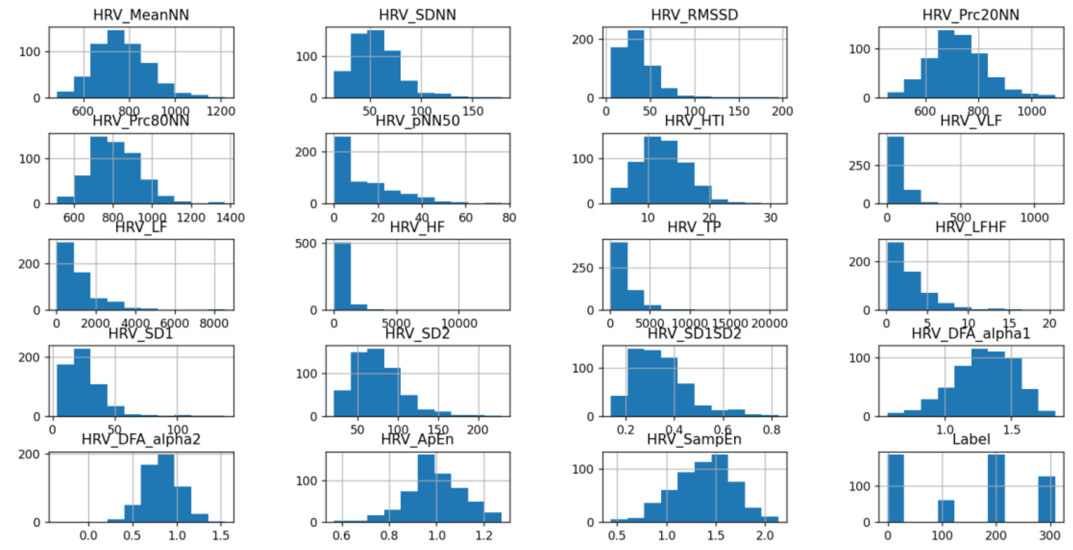
Data Visualization

We started the visualization phase by showing the whole data set features(columns) using the two famous plots which are the boxplot and the histograms:


```
def showDatawithDiagrams(dataFrame,plt):

    dataFrame.boxplot(figsize=(30,10))
    plt.show()


    dataFrame.hist(figsize=(30,20))
    plt.show()
```

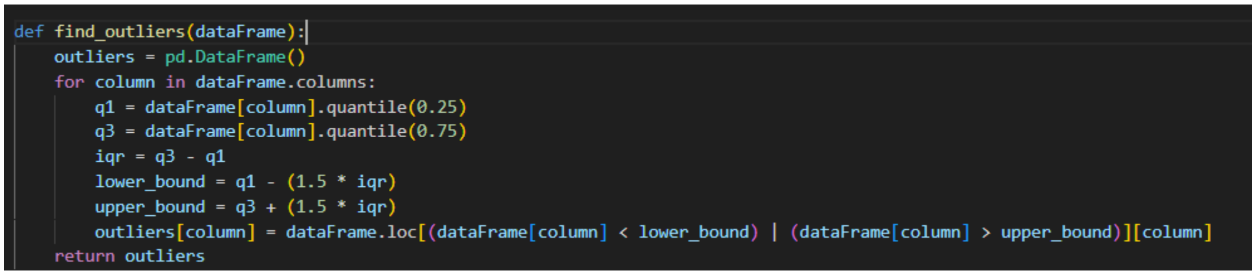
And the result is:



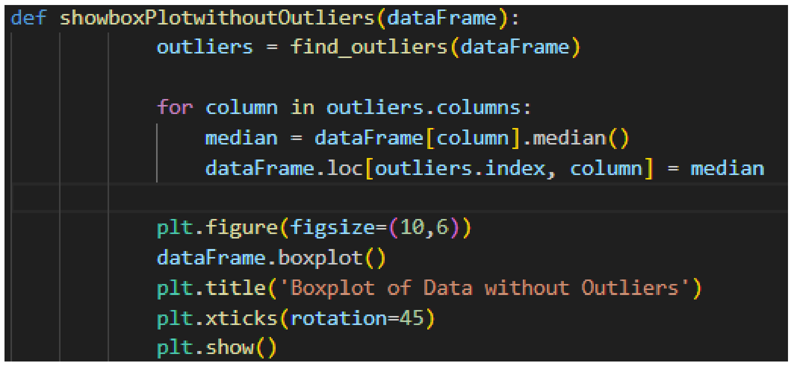


Data Preprocessing

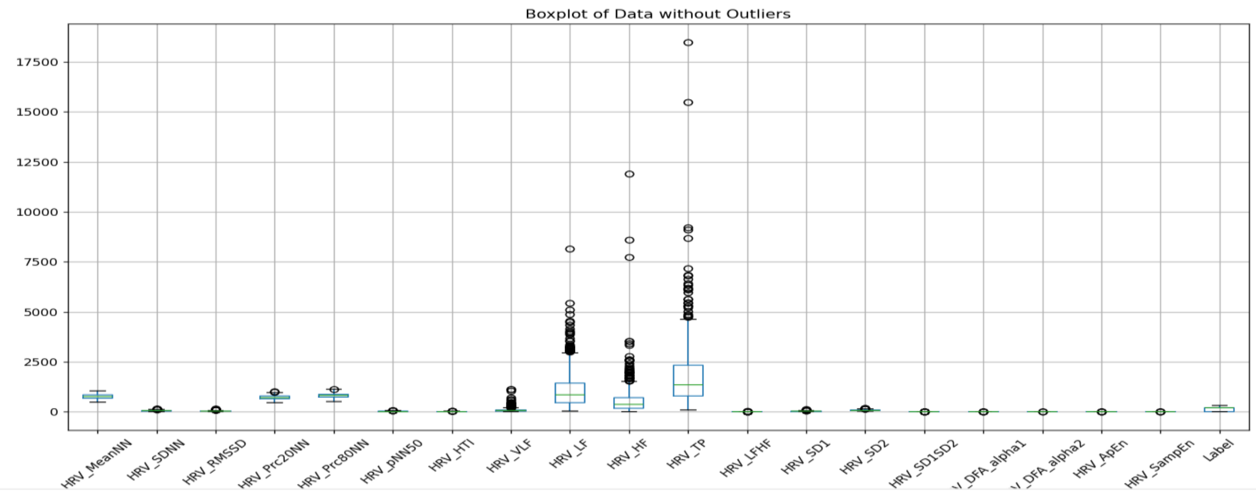
In the phase it is so important to calculate the outlier and do an outlier treatment, since that when handling outliers, we consider the data's nature, impact on analysis, analysis objectives, and documentation. In EEG data, outliers are nonbrain-originating signals, while in ECG signals, they deviate significantly from the expected pattern. Outliers affect analysis accuracy, so we address the most apparent ones likely outside the expected range. We developed a function using interquartile ranges to identify and remove outliers, enhancing analysis reliability.



```python
def find_outliers(dataFrame):
    outliers = pd.DataFrame()
    for column in dataFrame.columns:
        q1 = dataFrame[column].quantile(0.25)
        q3 = dataFrame[column].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - (1.5 * iqr)
        upper_bound = q3 + (1.5 * iqr)
        outliers[column] = dataFrame.loc[(dataFrame[column] < lower_bound) | (dataFrame[column] > upper_bound)][column]
    return outliers
```



```python
def showboxPlotwithoutOutliers(dataFrame):
            outliers = find_outliers(dataFrame)

            for column in outliers.columns:
                median = dataFrame[column].median()
                dataFrame.loc[outliers.index, column] = median

            plt.figure(figsize=(10,6))
            dataFrame.boxplot()
            plt.title('Boxplot of Data without Outliers')
            plt.xticks(rotation=45)
            plt.show()
```

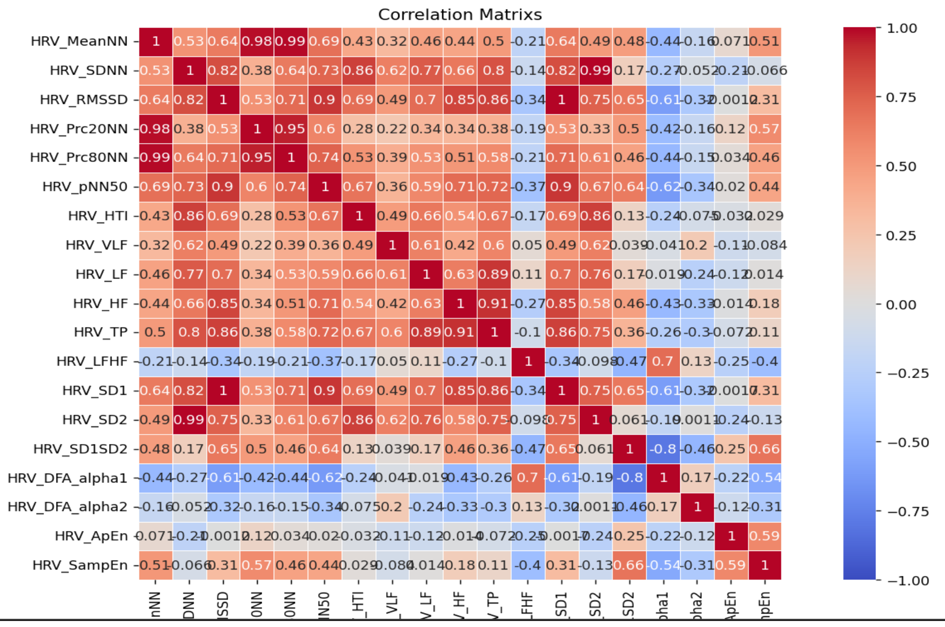


<u>Data Correlation</u>

Constructing a correlation matrix serves to uncover the intricate relationships among features within our dataset. By examining the correlation coefficients, we discern both the strength and direction of these connections, thereby shedding light on underlying patterns and potential multicollinearity issues. Identifying multicollinearity is paramount, given its impact on the stability of predictive models. Moreover, the correlation matrix aids in the selection of relevant features for analysis, guiding decisions regarding variable inclusion. Visualizing the matrix through heatmaps offers a user-friendly approach to grasping complex interrelationships intuitively.

Furthermore, the correlation matrix plays a pivotal role in several data quality assurance tasks. It assists in outlier detection, validates assumptions of linearity, and facilitates the handling of missing values. These steps are fundamental in ensuring the dataset's integrity and readiness for precise and insightful analysis.

Before proceeding with the correlation matrix, we had to cancel the label column which it doesn’t have a context among the data and of course we will use it again in the classification in the supervised machine learning that will be used later in the report



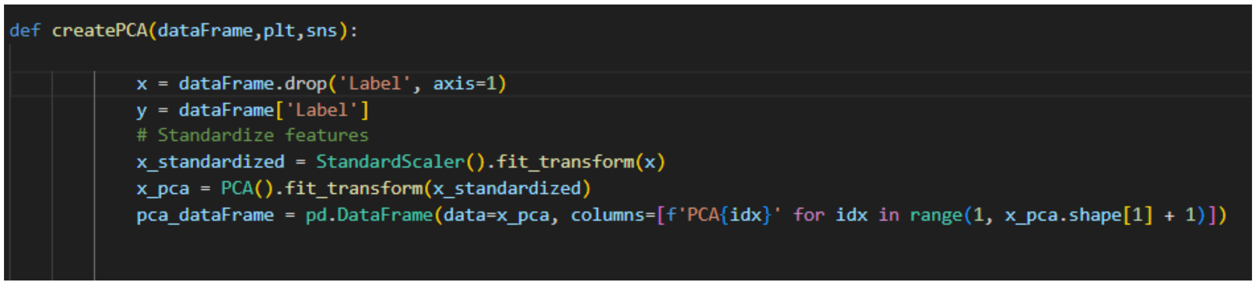
Principal Component Analysis

Principal Component Analysis (PCA) is a powerful statistical technique used to simplify and extract essential information from complex, multivariate datasets. Its primary objective is to transform a set of correlated variables, known as features, into a new set of uncorrelated variables called principal components. These principal components are linear combinations of the original features and are arranged in order of the variance they capture.

PCA is particularly valuable when dealing with large datasets containing numerous features. In such datasets, the interrelationships between variables often result in redundancy and computational challenges. By applying PCA, redundant information is minimized, and the most significant patterns within the data are highlighted. This process not only simplifies subsequent analyses but also accelerates computational processes.

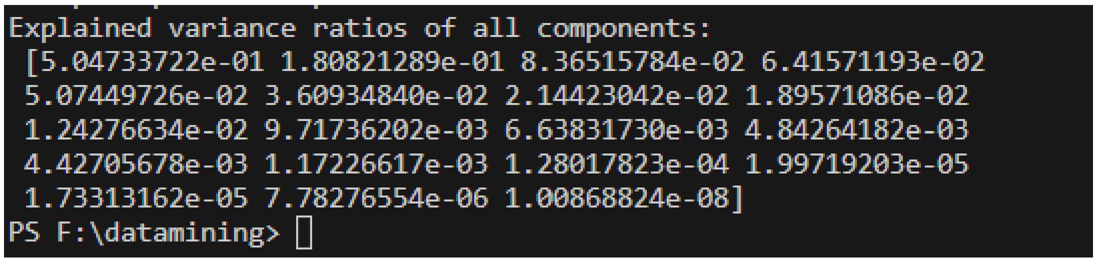
Furthermore, PCA aids in identifying dominant patterns and trends within the dataset, leading to a more concise and interpretable representation. Performing PCA before feature selection offers several benefits. Feature selection techniques, which are typically computationally intensive on large datasets, are streamlined by PCA as it transforms the original features into a smaller set of principal components. Moreover, PCA decorrelates the features, making subsequent feature selection more effective as it operates on uncorrelated variables.

First of all, we had to drop the labels columns since the PCA will not be applied on it, and we also saved its value in a variable for a future use, so we had two variables. X which is the dataset that will undergoes the PCA process and then we did a standardization for the data so that it has a zero mean value and the 1 standard deviation, we did that to make the PCA more efficient then we created a totally new data frame which contains the principal components generated after the PCA execution as illustrated in the code:

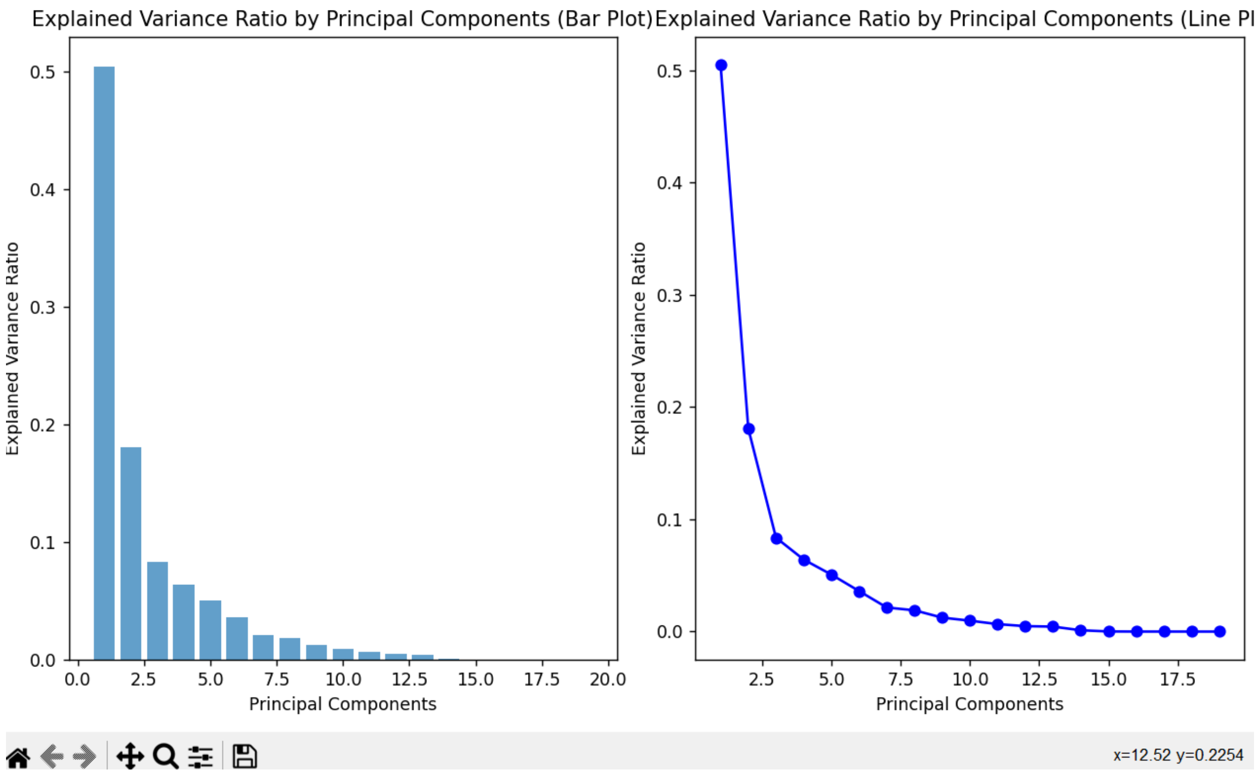


```
def createPCA(dataFrame,plt,sns):

        x = dataFrame.drop('Label', axis=1)
        y = dataFrame['Label']
        # Standardize features
        x_standardized = StandardScaler().fit_transform(x)
        x_pca = PCA().fit_transform(x_standardized)
        pca_dataFrame = pd.DataFrame(data=x_pca, columns=[f'PCA{idx}' for idx in range(1, x_pca.shape[1] + 1)])
```

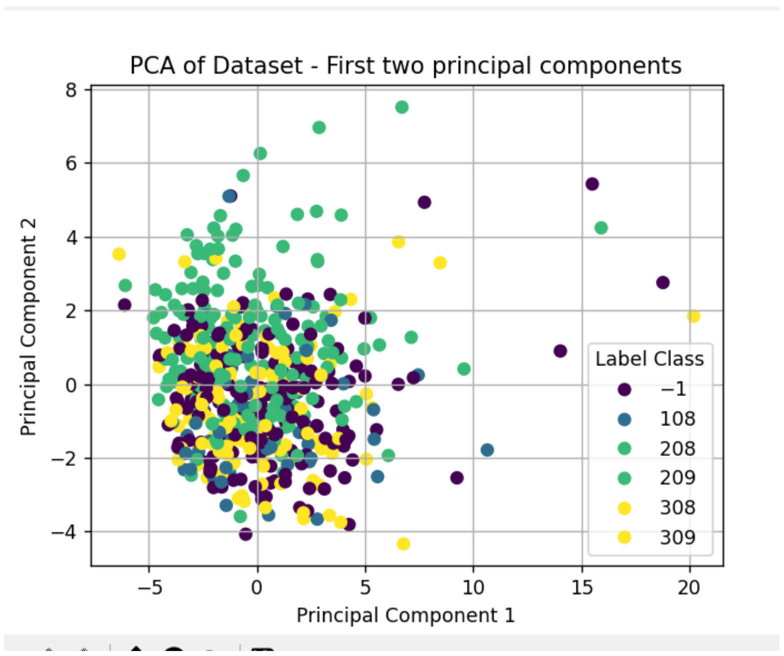
Then we computed the explained variance by each and every principal component and we printed it and plotted it in a bar and a line plot so we can visualize it:

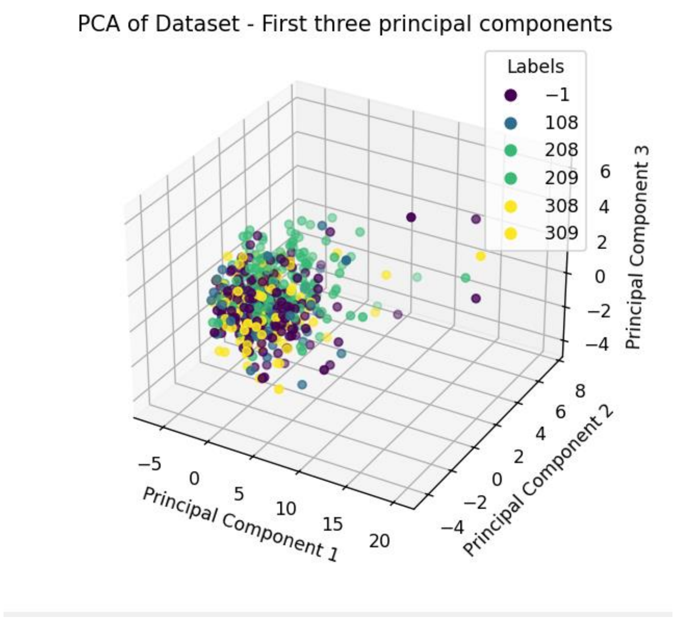


```
Explained variance ratios of all components:
 [5.04733722e-01 1.80821289e-01 8.36515784e-02 6.41571193e-02
 5.07449726e-02 3.60934840e-02 2.14423042e-02 1.89571086e-02
 1.24276634e-02 9.71736202e-03 6.63831730e-03 4.84264182e-03
 4.42705678e-03 1.17226617e-03 1.28017823e-04 1.99719203e-05
 1.73313162e-05 7.78276554e-06 1.00868824e-08]
PS F:\datamining> 
```



And then I visualized the the 1st two principle components (with bigger variances of course and this how the PCA works) in 2D space and I also plotted the 1st three principle components in 3D space, highlighting the different labels with different colors for elaboration:





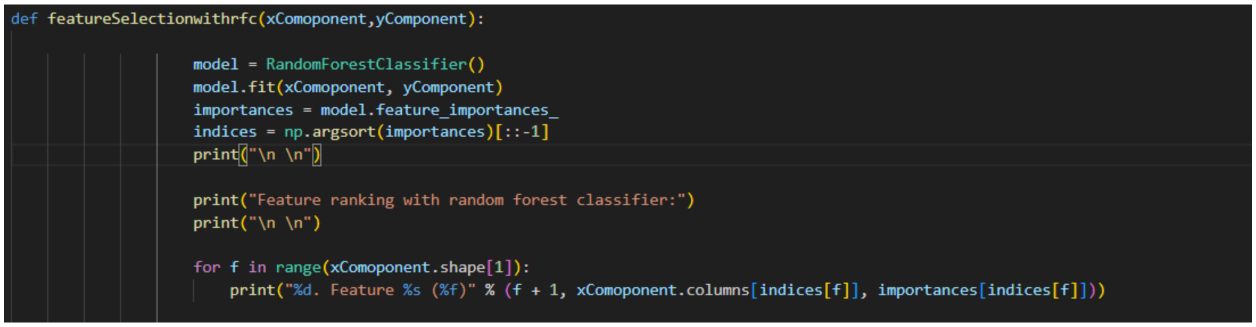
<u>Feature Selection</u>

Feature selection is an essential part of analyzing data, where we focus on picking out the most important attributes or variables. The goal is to simplify the dataset by keeping only the features that really matter for making predictions accurately. We do this by evaluating and ranking features based on how relevant they are to the task we're working on.

By narrowing down the features we use, we not only make our predictive models work better but also reduce the chance of them learning from noise or irrelevant information. This is especially important when dealing with lots of different features, where some might not actually help our model make better predictions.

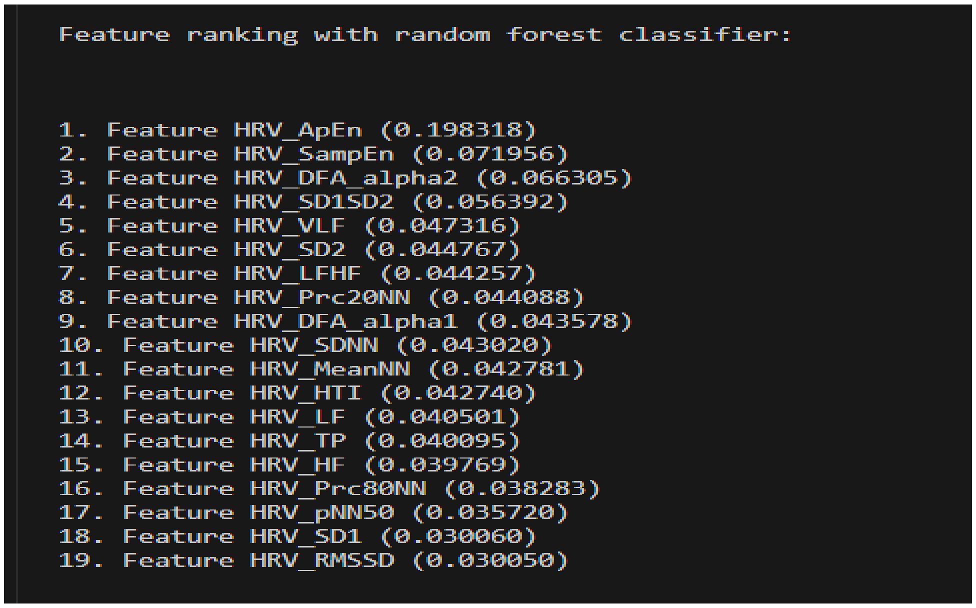
There are different ways to do feature selection, like using filter methods, wrapper methods. Each method has its own strengths and is useful in different situations. We decide which method to use based on the specific characteristics of our dataset. In this analysis we used 4 ways which are:

1- Feature Selection using Random forest classifier:



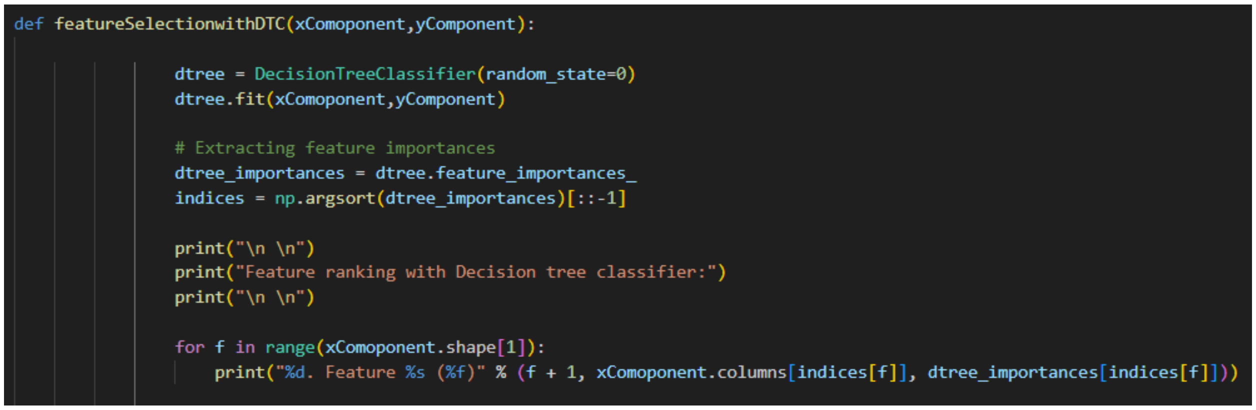
```
def featureSelectionwithrfc(xComoponent,yComponent):

                model = RandomForestClassifier()
                model.fit(xComoponent, yComponent)
                importances = model.feature_importances_
                indices = np.argsort(importances)[::-1]
                print("\n \n")

                print("Feature ranking with random forest classifier:")
                print("\n \n")

                for f in range(xComoponent.shape[1]):
                    print("%d. Feature %s (%f)" % (f + 1, xComoponent.columns[indices[f]], importances[indices[f]]))
```

And the features were ranked like:

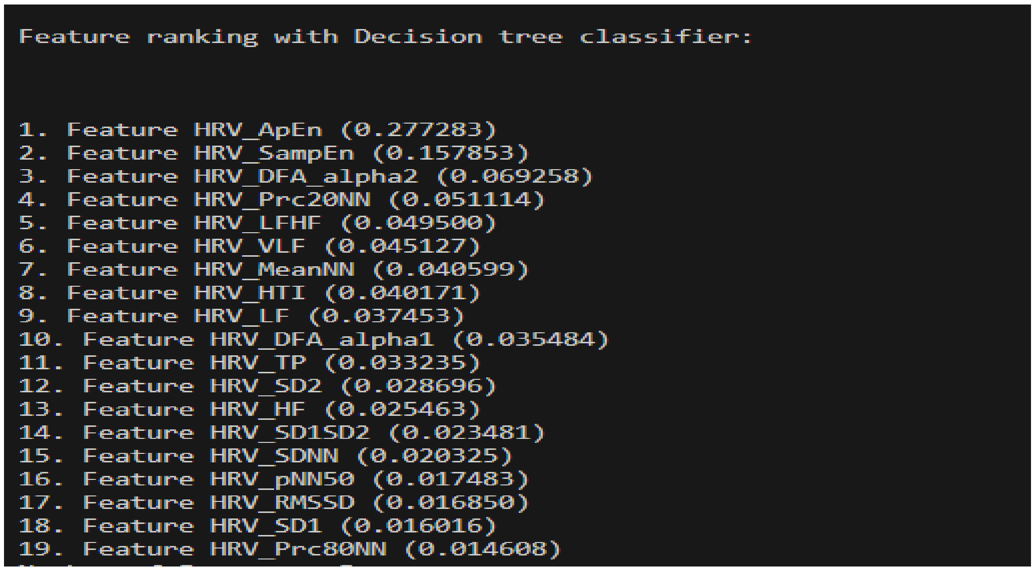


```
Feature ranking with random forest classifier:


1. Feature HRV_ApEn (0.198318)
2. Feature HRV_SampEn (0.071956)
3. Feature HRV_DFA_alpha2 (0.066305)
4. Feature HRV_SD1SD2 (0.056392)
5. Feature HRV_VLF (0.047316)
6. Feature HRV_SD2 (0.044767)
7. Feature HRV_LFHF (0.044257)
8. Feature HRV_Prc20NN (0.044088)
9. Feature HRV_DFA_alpha1 (0.043578)
10. Feature HRV_SDNN (0.043020)
11. Feature HRV_MeanNN (0.042781)
12. Feature HRV_HTI (0.042740)
13. Feature HRV_LF (0.040501)
14. Feature HRV_TP (0.040095)
15. Feature HRV_HF (0.039769)
16. Feature HRV_Prc80NN (0.038283)
17. Feature HRV_pNN50 (0.035720)
18. Feature HRV_SD1 (0.030060)
19. Feature HRV_RMSSD (0.030050)
```

2- Feature Selection using Decision Tree classifier:

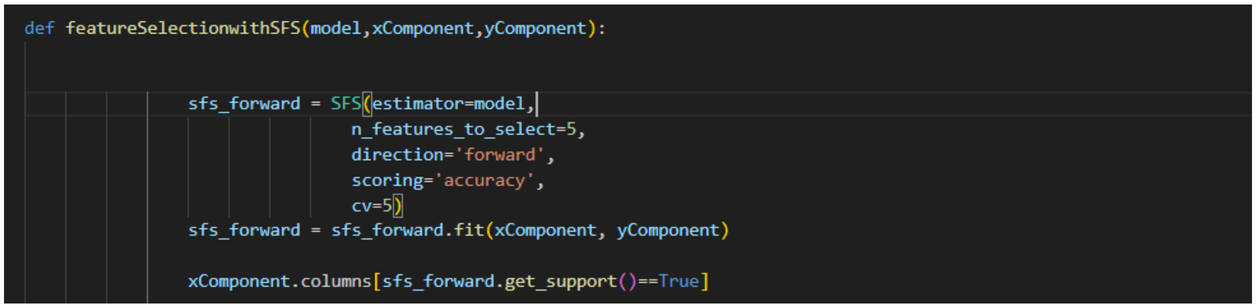


```python
def featureSelectionwithDTC(xComoponent,yComponent):

            dtree = DecisionTreeClassifier(random_state=0)
            dtree.fit(xComoponent,yComponent)

            # Extracting feature importances
            dtree_importances = dtree.feature_importances_
            indices = np.argsort(dtree_importances)[::-1]

            print("\n \n")
            print("Feature ranking with Decision tree classifier:")
            print("\n \n")

            for f in range(xComoponent.shape[1]):
                print("%d. Feature %s (%f)" % (f + 1, xComoponent.columns[indices[f]], dtree_importances[indices[f]]))
```

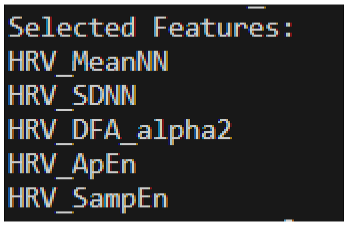


```
Feature ranking with Decision tree classifier:


1. Feature HRV_ApEn (0.277283)
2. Feature HRV_SampEn (0.157853)
3. Feature HRV_DFA_alpha2 (0.069258)
4. Feature HRV_Prc20NN (0.051114)
5. Feature HRV_LFHF (0.049500)
6. Feature HRV_VLF (0.045127)
7. Feature HRV_MeanNN (0.040599)
8. Feature HRV_HTI (0.040171)
9. Feature HRV_LF (0.037453)
10. Feature HRV_DFA_alpha1 (0.035484)
11. Feature HRV_TP (0.033235)
12. Feature HRV_SD2 (0.028696)
13. Feature HRV_HF (0.025463)
14. Feature HRV_SD1SD2 (0.023481)
15. Feature HRV_SDNN (0.020325)
16. Feature HRV_pNN50 (0.017483)
17. Feature HRV_RMSSD (0.016850)
18. Feature HRV_SD1 (0.016016)
19. Feature HRV_Prc80NN (0.014608)
```

3- Feature Selection using Forward Sequential Selection:

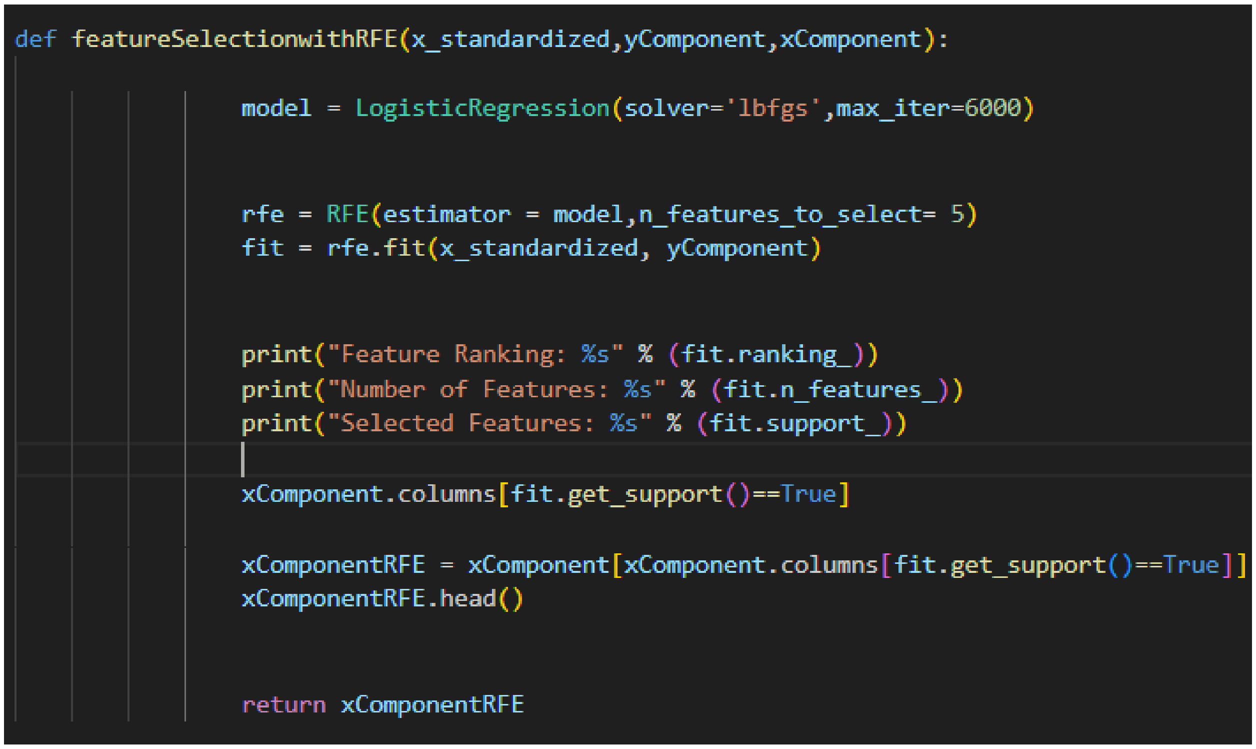


```
def featureSelectionwithSFS(model,xComponent,yComponent):


        sfs_forward = SFS(estimator=model,
                          n_features_to_select=5,
                          direction='forward',
                          scoring='accuracy',
                          cv=5)
        sfs_forward = sfs_forward.fit(xComponent, yComponent)

        xComponent.columns[sfs_forward.get_support()==True]
```

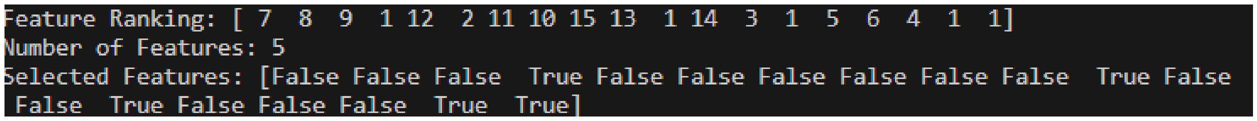


```
Selected Features:
HRV_MeanNN
HRV_SDNN
HRV_DFA_alpha2
HRV_ApEn
HRV_SampEn
```

4- Feature Selection using Recursive Feature Elimination:



```
def featureSelectionwithRFE(x_standardized,yComponent,xComponent):

        model = LogisticRegression(solver='lbfgs',max_iter=6000)


        rfe = RFE(estimator = model,n_features_to_select= 5)
        fit = rfe.fit(x_standardized, yComponent)


        print("Feature Ranking: %s" % (fit.ranking_))
        print("Number of Features: %s" % (fit.n_features_))
        print("Selected Features: %s" % (fit.support_))

        xComponent.columns[fit.get_support()==True]

        xComponentRFE = xComponent[xComponent.columns[fit.get_support()==True]]
        xComponentRFE.head()


        return xComponentRFE
```



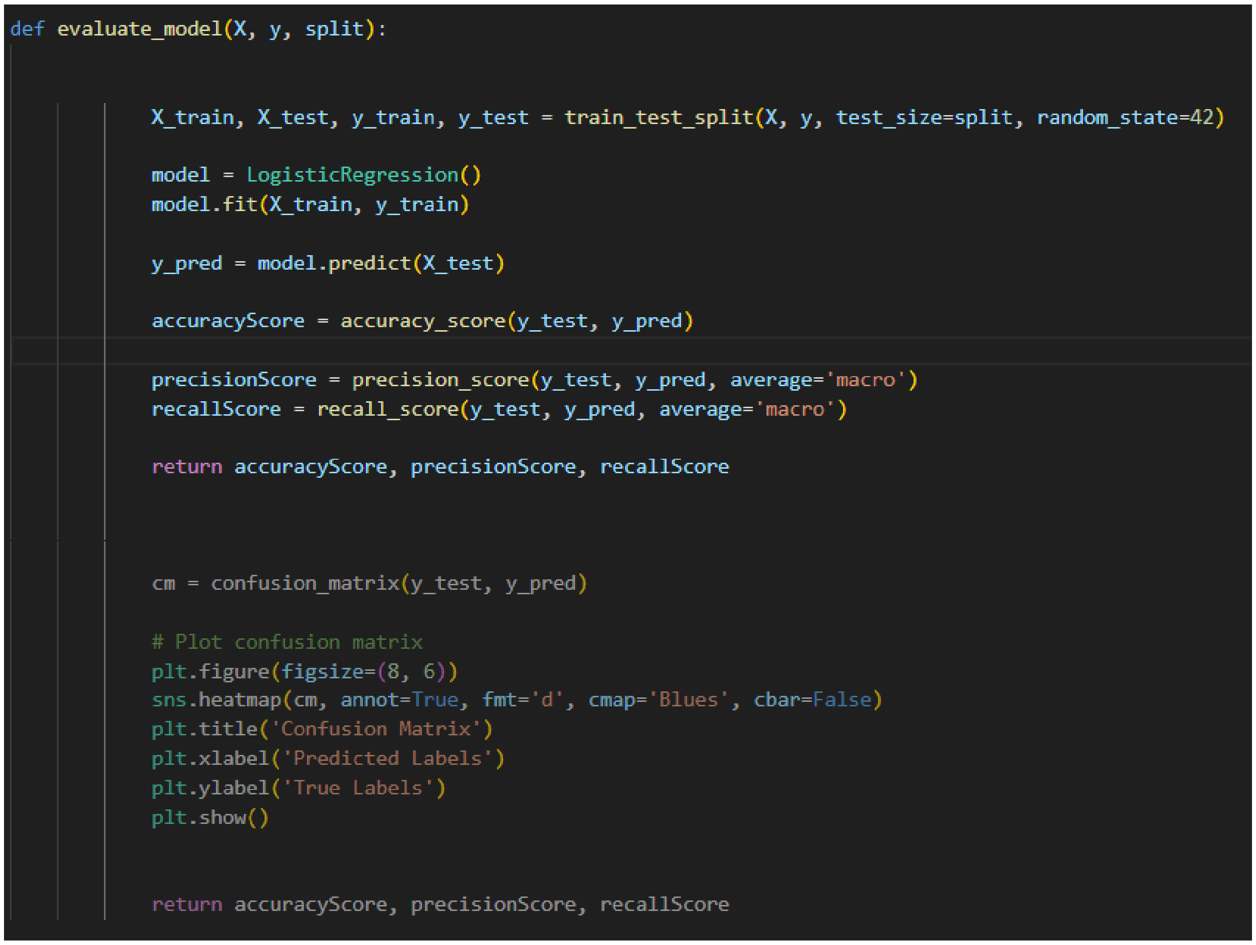
```
eature Ranking: [ 7  8  9  1 12  2 11 10 15 13  1 14  3  1  5  6  4  1  1]
Number of Features: 5
Selected Features: [False False False  True False False False False False False  True False
 False  True False False False  True  True]
```

Machine Learning

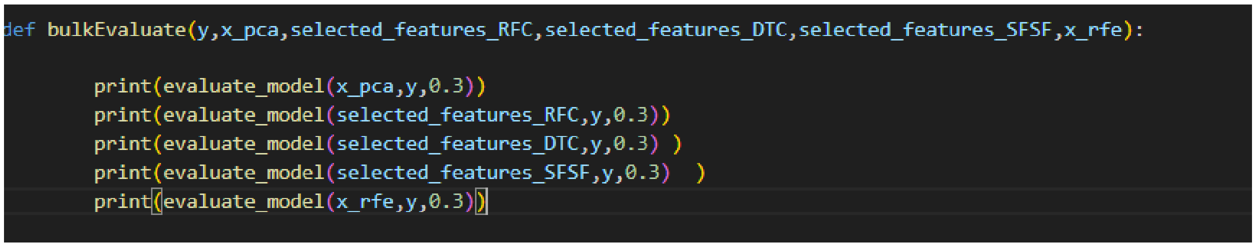
Since we have the labels for different observation, so of course the Supervised learning is the type we will use in this model implementation.

As known there are many algorithms, but I decided to use the Logistic regression after using already some other algorithms like SVC but the metrics were not satisfying.

We splited the data into two portions which are for testing and training 30% and 70% respectively, and then we used the model Logistic Rgression to do its work, after fitting the data, we used the predict function to know how is the efficiency of the model, please pay close attention that the I used the algorithm, on 4 different data lists of features that we obtained in the feature extraction phase, and then using the evaluate_model function , I could do retrieve the metrics and calculations of how the model is efficient through ***accuracy***,***precsison*** and ***recall*** scores, then we generated a confusion matrix for each trial.

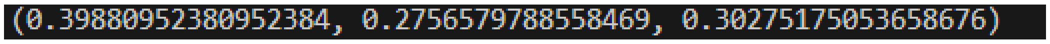


```
def evaluate_model(X, y, split):


        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)

        model = LogisticRegression()
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)

        accuracyScore = accuracy_score(y_test, y_pred)

        precisionScore = precision_score(y_test, y_pred, average='macro')
        recallScore = recall_score(y_test, y_pred, average='macro')

        return accuracyScore, precisionScore, recallScore



        cm = confusion_matrix(y_test, y_pred)

        # Plot confusion matrix
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
        plt.title('Confusion Matrix')
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.show()


        return accuracyScore, precisionScore, recallScore
```

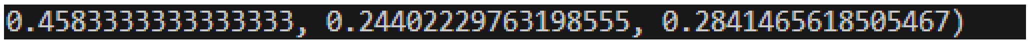


```
def bulkEvaluate(y,x_pca,selected_features_RFC,selected_features_DTC,selected_features_SFSF,x_rfe):

    print(evaluate_model(x_pca,y,0.3))
    print(evaluate_model(selected_features_RFC,y,0.3))
    print(evaluate_model(selected_features_DTC,y,0.3) )
    print(evaluate_model(selected_features_SFSF,y,0.3)  )
    print(evaluate_model(x_rfe,y,0.3))
```

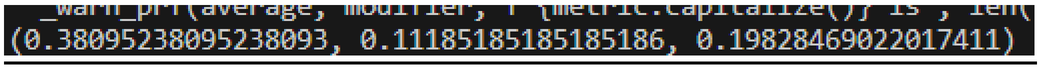
And these are evaluation function for the different data feeded to the model, respictevely:



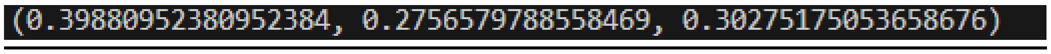
```
(0.39880952380952384, 0.2756579788558469, 0.30275175053658676)
```



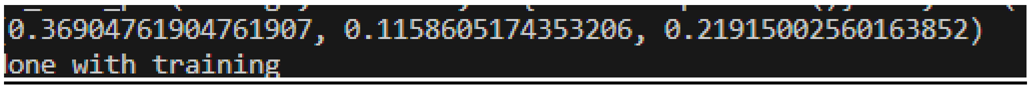
```
0.4583333333333333, 0.24402229763198555, 0.2841465618505467)
```



```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(
(0.38095238095238093, 0.11185185185185186, 0.19828469022017411)
```



```
(0.39880952380952384, 0.2756579788558469, 0.30275175053658676)
```



```
0.36904761904761907, 0.1158605174353206, 0.21915002560163852)
one with training
```

Conclusion

In summary, despite thorough exploration and application of various data analysis techniques on the dataset from Study 2, the model's accuracy reached a maximum of only 45%. And that achieved after using other algorithms but in vain This limited accuracy could be attributed to multiple factors within the dataset and the modeling approach. One possible explanation is the intricate nature of psychophysiological responses to emotions, which might necessitate more advanced modeling techniques or inclusion of additional contextual features for better predictions. Furthermore, the dataset's small size or the presence of subtle patterns and outliers could pose challenges in accurately capturing and generalizing the underlying relationships. To improve predictive performance in future analyses, it's crucial to critically assess and potentially expand the dataset, experiment with alternative algorithms, and explore feature engineering strategies.