University of Messina

Bachelor of Data Analysis

ACADEMIC YEAR - 2023/2024

DATA-MINING

(Project Report)

Supervisor :

Prof. Gennaro Tartarisco

Student :

SAHIL NAKRANI (533173)

Table of Content

INTRODUCTION

The objective of this data mining project is to analyze a dataset (loaded from "Dataset_Study3.csv") and perform various tasks such as data cleaning, exploration, feature selection, clustering, and classification. The project aims to uncover patterns, relationships, and insights within the data that can be used for predictive modelling.

DATASET

The dataset comprises features from cardiovascular responses of 147 young adults. The study aims to predict daily actions based on cognitive and physiological responses to a lab-based social task, particularly evaluating motivations towards gratitude interventions.

The procedure followed by the study :

Questionnaires	Continuous recording of physiological parameters × 2 (grateful and neutral)						Questionnaires	Follow up
Depressive symptoms and trait gratitude	Baseline (3 min)	Challenge/threat evaluations	Grateful/neutral SMS preparation (3 min)	SMS sending (grateful/neutral)	Recovery (3 min)	Invitation to continue the intervention in daily life	Motivation and behavioral intentions towards the intervention	Intervention completion rate (after three weeks)

Data composition :

425 observations (142 subjects x 3 conditions: baseline, gratitude and neutral). Indeed, 5 subjects were excluded because the ECG signal was missing.

3 labels indicating the different conditions from which the features were extracted. Baseline condition (-1), Gratitude (319), Neutral State (109).

Features extracted from the RR signal, which is the time elapsed between two successive R waves of the QRS signal on the electrocardiogram, including time and frequency domain features, as well as non-linear features.

We have time domain features [MeanNN, SDNN, RMSSD, Prc20NN, Prc80NN, PNN50, HTI], frequency domain features [VLF, LF, HF, TF, LFHF], and non-linear features [SD1, SD2, SD1SD2, DFA_alpha1, DFA_alpha2, ApEn, SampEn] in our dataset.

LIBRARIES

In this project, we begin by importing essential Python libraries necessary for exploring and analysing our dataset.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

- Pandas is used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

- Matplotlib is a 2D plotting library for Python. It allows you to create a variety of static, animated, and interactive visualizations in Python.

- Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.

- Scikit-learn is a machine learning library for Python. It provides simple and efficient tools for data analysis and modeling, including tools for classification, regression, clustering, dimensionality reduction, and more.

    - **model_selection**: Provides tool for model selection including functions for splitting datasets into train and test sets.
    - **Preprocessing**: Contains functions for preprocessing data, such as scaling features.
    - **feature_selection**: Offers methods for selecting relevant features in machine learning models.
    - **decomposition**: Implement dimensionality reduction techniques, such as Principal Component Analysis (PCA).
    - **cluster**: includes tools for clustering, such as the KMeans algorithm.
    - ensemble: Providing ensemble learning methods like RandomForestClassifier.
    - **metrics**: Contains evaluation metrics for assessing model performance.

- **linear_model**: Implements linear models for classification, like Logistic Regression.
- **RFE**: Recursive Feature Elimination (RFE) is a feature selection method. It works by recursively removing the least important features, training the model on the remaining features, and repeating the process until the desired number of features is reached.
- **LogisticRegression**: Logistic Regression is a classification algorithm used for binary and multiclass classification problems. It models the probability of an event occurring as a logistic function.

LOADING THE DATASET

Here, we are using pandas to load the csv file (read_csv()) and by describe() function, we are summarizing basic statistics of numerical features of the dataset including mean, std, min, 25%, 50%, 75%, and max.

```
data = pd.read_csv("Dataset_Study3.csv")

print(data.describe())
```

```
       HRV_MeanNN    HRV_SDNN   HRV_RMSSD  HRV_Prc20NN  HRV_Prc80NN  \
count  426.000000  426.000000  426.000000   426.000000   426.000000
mean   760.770070   60.982323   38.275327   711.938967   807.670423
std     94.046679   31.201040   31.129139    87.268890   104.327001
min    555.599379   16.506371    5.812904   533.000000   568.600000
25%    692.535430   44.568471   22.849493   649.200000   733.100000
50%    757.405554   56.893008   32.411485   712.400000   809.800000
75%    829.424839   71.232137   44.493817   772.850000   881.300000
max   1105.870370  389.188189  382.526115  1069.800000  1159.000000

       HRV_pNN50     HRV_HTI       HRV_VLF        HRV_LF        HRV_HF  \
count  426.000000  426.000000    426.000000    426.000000    426.000000
mean    13.606149   12.476551    242.663063   1948.718982    885.662291
std     13.232075    3.688746   1883.692791  10264.804862   2547.208263
min      0.000000    4.625000      4.737018     49.902409     25.658985
25%      2.921488    9.815909     37.125944    548.847446    183.689415
50%      9.732418   12.114379     77.209885    954.991547    399.794185
75%     20.544258   14.696429    143.858800   1592.353978    805.296999
max     64.814815   26.500000  33458.813731  177202.470783  32325.575897

             HRV_TP    HRV_LFHF     HRV_SD1     HRV_SD2  HRV_SD1SD2  \
count    426.000000  426.000000  426.000000  426.000000  426.000000
mean    3123.250101    3.253009   27.124319   81.490225    0.323573
std    14487.986145    2.977534   22.067763   39.291432    0.107316
min      194.346766    0.171808    4.116741   23.014532    0.125916
...
25%        1.177485        0.735552    0.897118    1.119527   -1.000000
50%        1.342512        0.879802    0.947427    1.317874  109.000000
75%        1.497038        1.014881    1.003580    1.501775  310.000000
max        1.851089        1.536499    1.151312    1.975763  310.000000
```

DATA CLEANING (PREPOCESSING)

NULL / NAN VALUES

The subsequent step involves checking for null values or Nan (Not a number) in our dataset. We utilize both the functions “isnull()” and “.isna()” to inspect null values. Fortunately, we don’t have any null values in dataset.

```
data.isnull().sum()
```

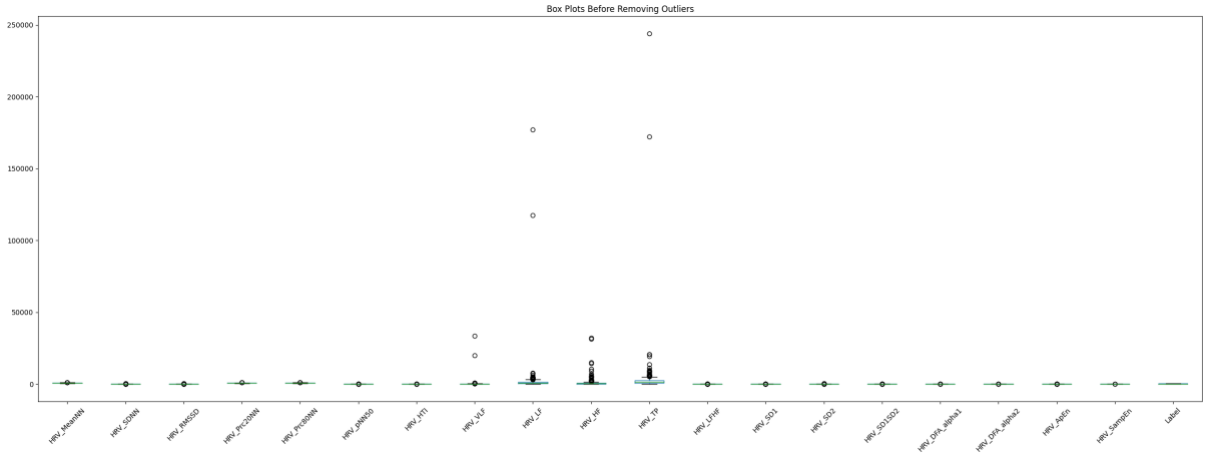
```
data.isna().sum()
```

OUTLIERS

Outliers are data points that deviate significantly from the rest of the dataset and can have a substantial impact on analysis.

Here, we used boxplot to visually inspect distribution of each numerical variable. And then the IQR (interquartile range) method applied to detect the outliers by quantifying the degree of deviation of each data point from the mean or median.

```
# Visualize outliers using box plots
plt.figure(figsize=(30, 10))
data.boxplot()
plt.title(f'Box Plots Before Removing Outliers')
plt.xticks(rotation=45)
plt.grid(False)
plt.show()
```



```python
# making a list of interested features excludnig lable
intrested_features = data.columns

def iqr_outliers(dataset,feature_name, multiplier=4):

    Q1 = dataset[feature_name].quantile(0.25)
    Q3 = dataset[feature_name].quantile(0.75)

    IQR = Q3 - Q1

    lwr_bound = Q1 - multiplier * IQR
    upp_bound = Q3 + multiplier * IQR

    ls = dataset.index[np.logical_or(dataset[feature_name]<lwr_bound,
                                     dataset[feature_name]>upp_bound)]
    return ls #return the indexes

outliers_detected={}
for i in intrested_features:
    outliers = iqr_outliers(data,i)
    outliers_detected[i] = outliers

    print('Variable',i)
    print(outliers)
    print(data[i].iloc[outliers])
    print('\n')
```

From this function we have detected some outliers which is showed below.

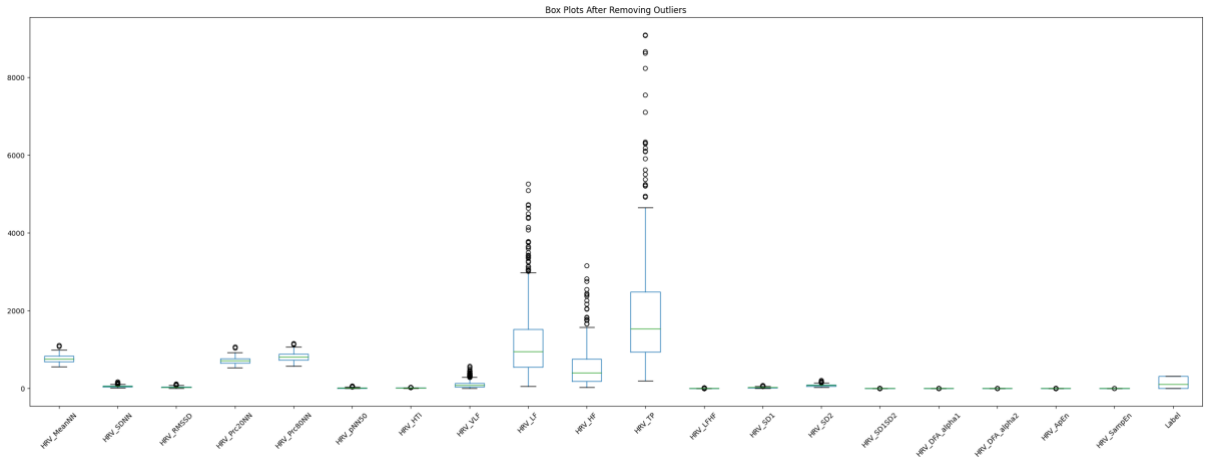
```
Variable HRV_SDNN
Index([130, 131], dtype='int64')
130    389.188189
131    379.291425
Name: HRV_SDNN, dtype: float64


Variable HRV_RMSSD
Index([72, 75, 76, 77, 130, 131], dtype='int64')
72     170.019078
75     177.096725
76     168.871906
77     143.089436
130    382.526115
131    358.175022
Name: HRV_RMSSD, dtype: float64
```

After detecting outliers, you can remove, transform, or replace the outliers. Here, we have replaced the outliers with the median. Because outliers can affect the result of our machine learning model. So, it is better to remove them or replace them with mean or median.

```python
for i in intrested_features:
    data[i] = data[i].replace(data[i].iloc[outliers_detected[i]].values,data[i].median())
```

We have also plotted the boxplot after replacing the outliers which is showed below.



DATA VISUALIZATION

HISTOGRAM FOR NUMERICAL FEATURES

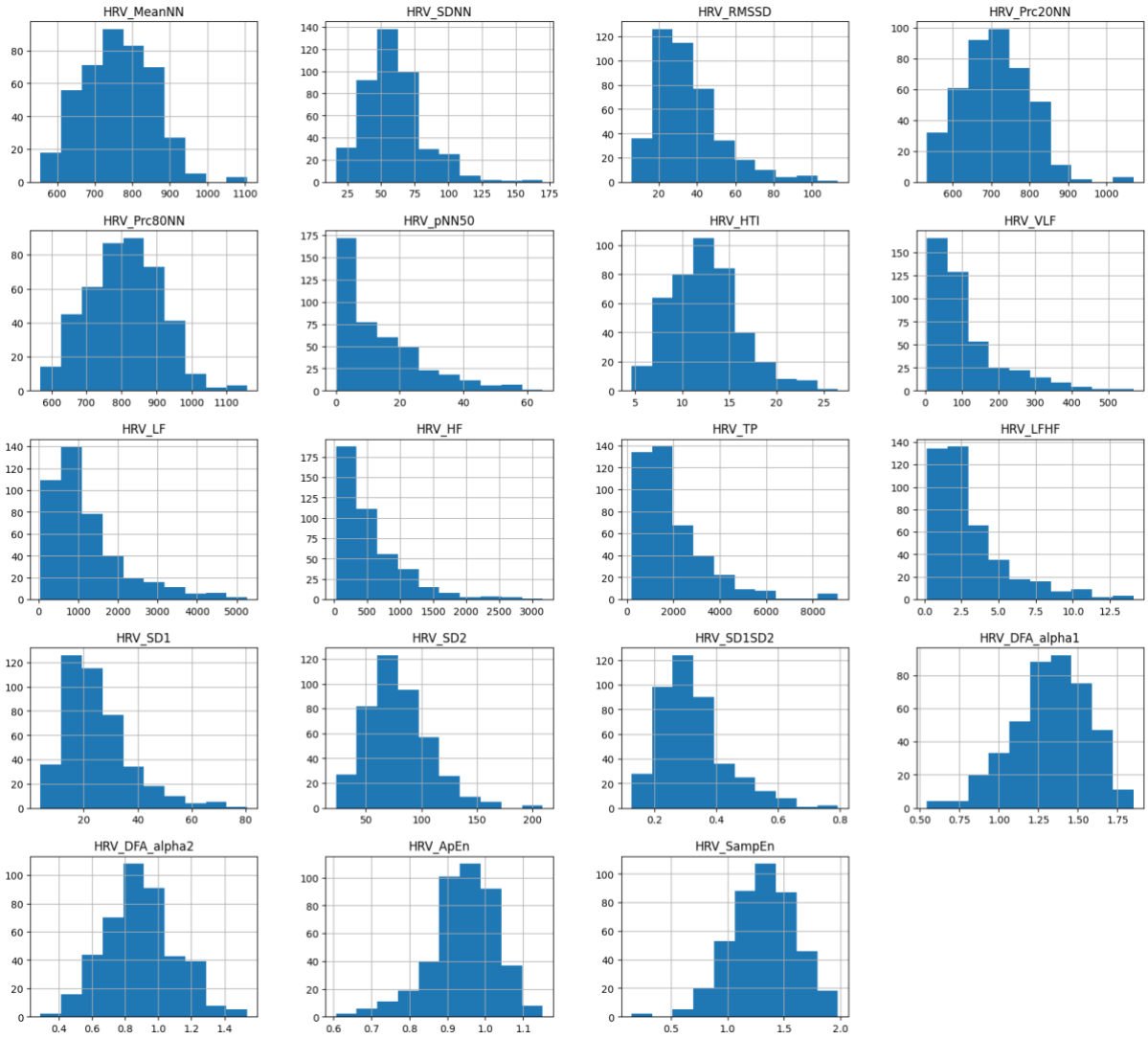
It provides a visual way to understand the underlying frequency or distribution of a set of continuous or discrete data.

The histograms below showcase the distribution of the numerical features , offering insights into potential skewness, outliers, and general patterns within the data.

```
# Visualize the distribution of numerical features
numerical_features = data.select_dtypes(include=['float64']).columns

data[numerical_features].hist(figsize=(20,18))
plt.show()
```

In the histogram showed below, you can see the distribution of the specific feature along with the skewness. For example, HRV_pNN50, HRV_VLF, HRV_LF, HRV_HF, and some others are showing the left skewness because most of the data is skewed left side of the graph.

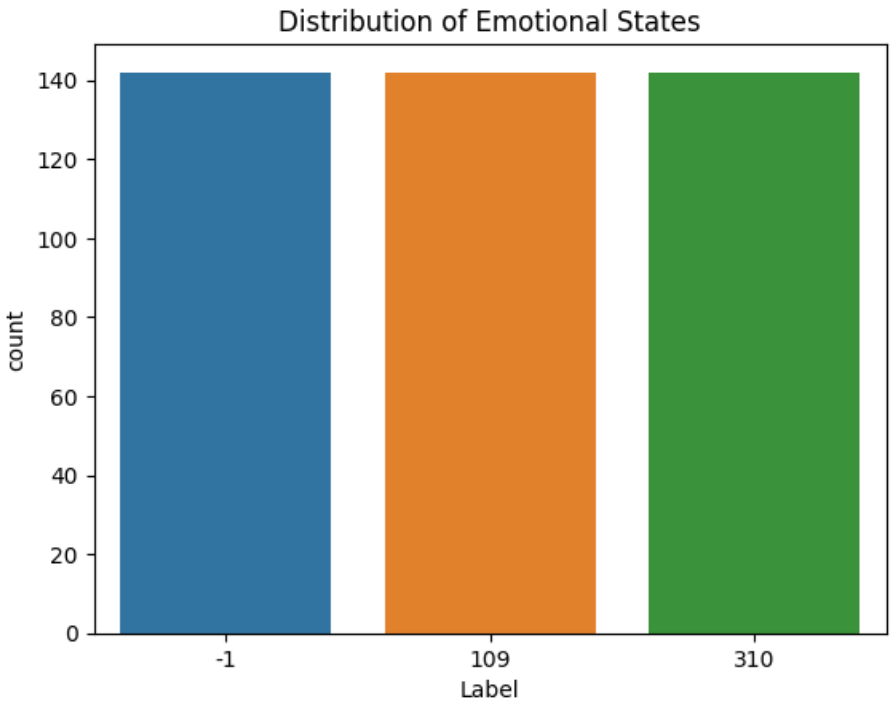


CATEGORICAL DISTRIBUTION

For categorical data we used bar chart. Because bar chart is representation of data that uses rectangular bars / columns to represent different categories / groups.

```
# Visualize the distribution of emotional states
sns.countplot(x='Label', data=data)
plt.title('Distribution of Emotional States')
plt.show()
```

Here, you can see bar chart provides clear overview of the target categorical feature. And as per bar chart, all the categories have same distribution. It means the data is equally distributed between these categories.



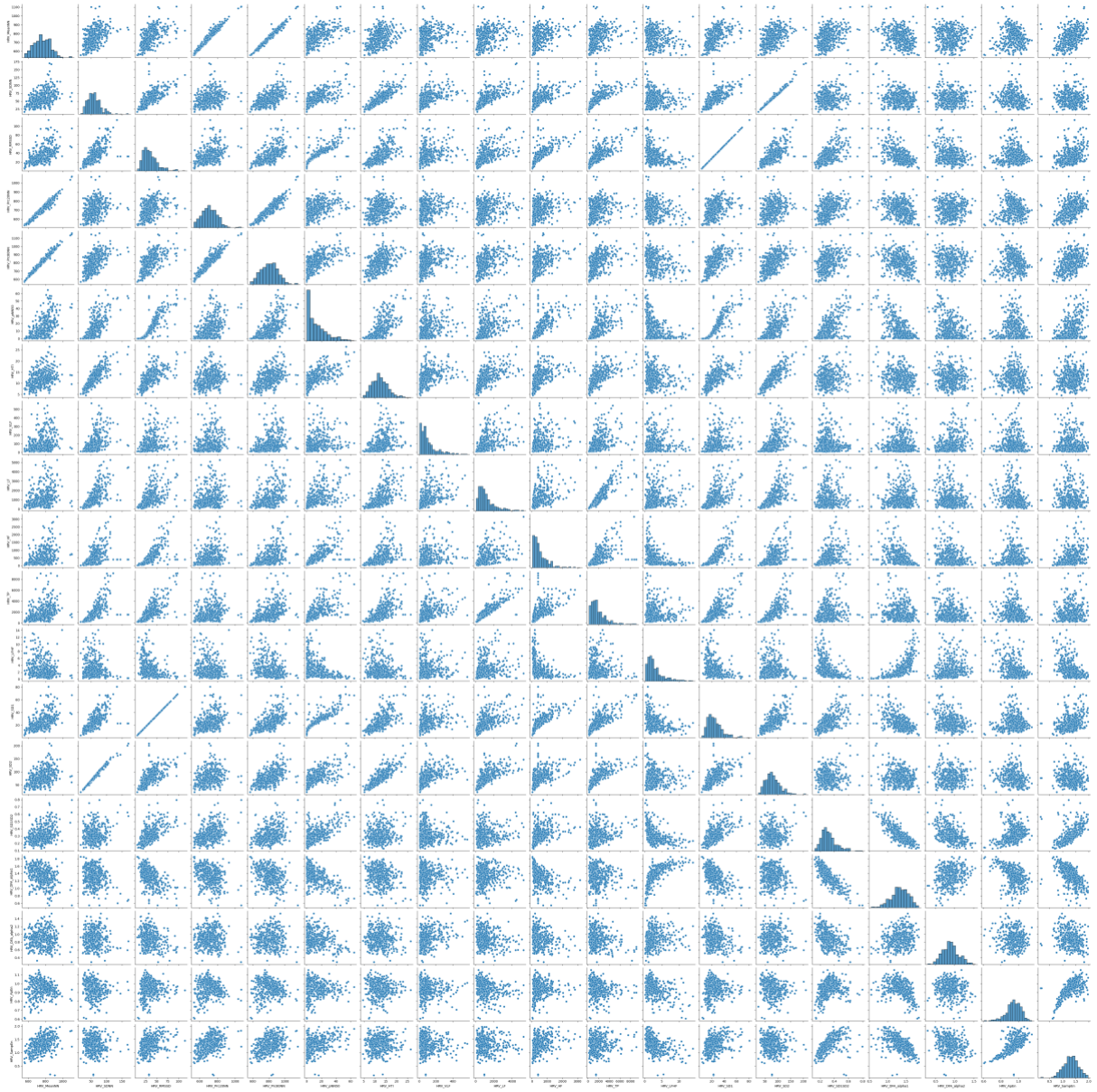
PAIRWISE SCATTER PLOT

It also known as a scatterplot matrix / scatterplot grid. It particularly useful when you have multiple variables and want to visualize the relationship between them. So here we are visualizing individual possible pairs of different variables of our dataset as a scatterplot matrix to see the relationship between that pair of variables.

```
sns.pairplot(data[numerical_features])
plt.show()
```

So, you can see below we mentioned a scatterplot matrix of all numerical features of our dataset. And it represents the relation of the pairs of features. Here, all the diagonal is showing histogram because it is relation with itself. By the scatters we can evaluate where features have positive correlation, negative correlation, or no relation.

Patterns in the scatter plots may indicate subgroups within the data or suggest potential grouping for further analysis.



CORRELATION MATRIX

It describes the extent to which two variables change together. In other words, it quantifies the degree to which a change in one variable is associated with a change in another variable.

1 : perfect positive correlation (if one variable increases, another variable also increases proportionally).
0 : no correlation (variable do not have linear relationship).
-1 : perfect negative correlation (if one variable increases, another variable decreases proportionally).

Formula for calculating correlation coefficient between two variable is often expressed as person's correlation coefficient (r) :

$$r = \frac{\Sigma(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\Sigma(X_i - \bar{X})^2 \Sigma(Y_i - \bar{Y})^2}}$$

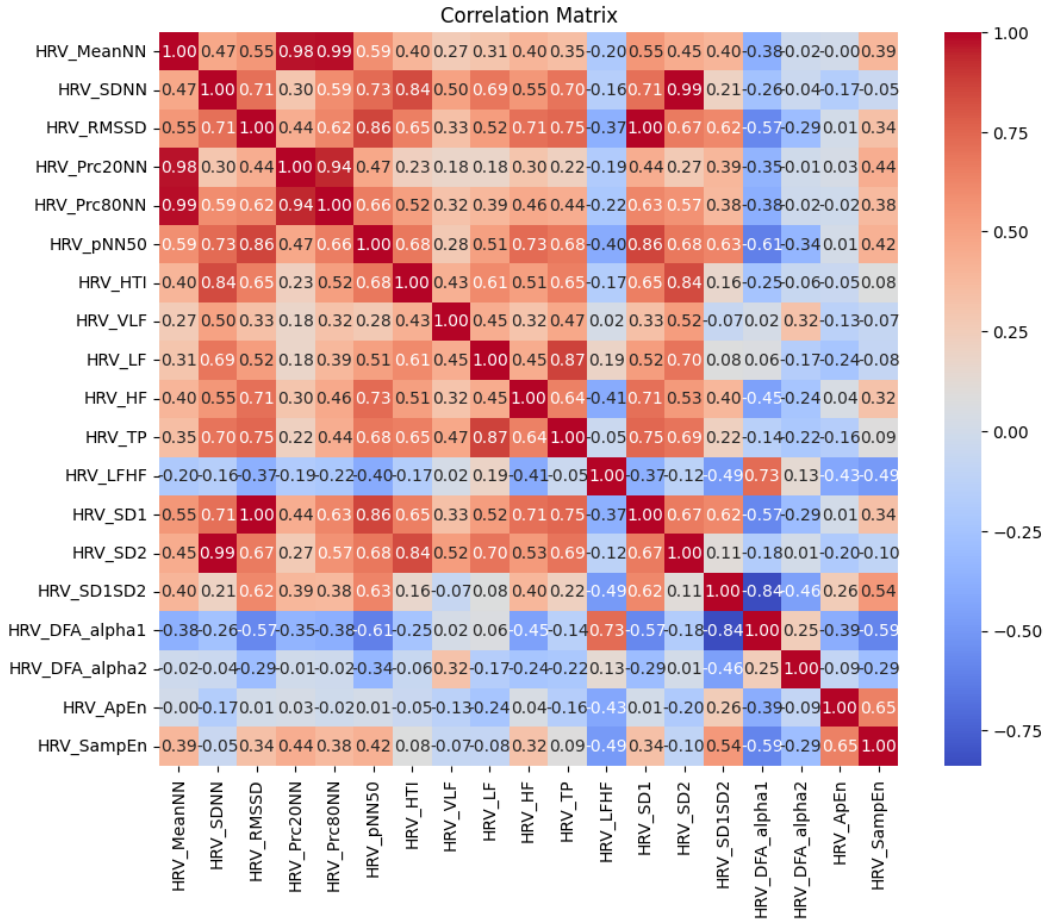
Here, we have mentioned a correlation matrix of a numerical features which is showing a correlation coefficient between all the possible pairs of variables (features) as a matrix form.

```
correlation_matrix = data[numerical_features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

"**annot=True**" : this parameter adds numeric annotation to each cell in the heatmap, displaying the correlation values.

"**cmap='coolwarm'**" : this parameter sets the color for heatmap (cool colors for low value and warm colors for high values).

"**fmt='.2f'**" : this parameter specifies the format for the annotation (floating number with two decimal position).



Here you can see some key points about this matrix.

- The correlation matrix is symmetric because correlation between x and y is same as the between y and x.
- The diagonal elements of the correlation matrix always have a correlation coefficient of 1, as they represent the correlation of a variable with itself.
- The correlation coefficient in the matrix ranges from -1 to 1.
- When visualize as heatmap, it helps to identify pattern and relationships between variables.
    - High positive correlation (shades of red)
    - High negative correlation (shades of blue)

HIGHLY CORRELATED FEATURES

we have also mentioned a correlation matrix for highly correlated features (for correlation coefficient more than 0.8).

Because if two features are highly correlated, we can choose to keep only one of them or we can do feature engineering by finding mean of them and deserve in the dataset instead of those two features. The goal is to retain most informative features while eliminating redundancy.

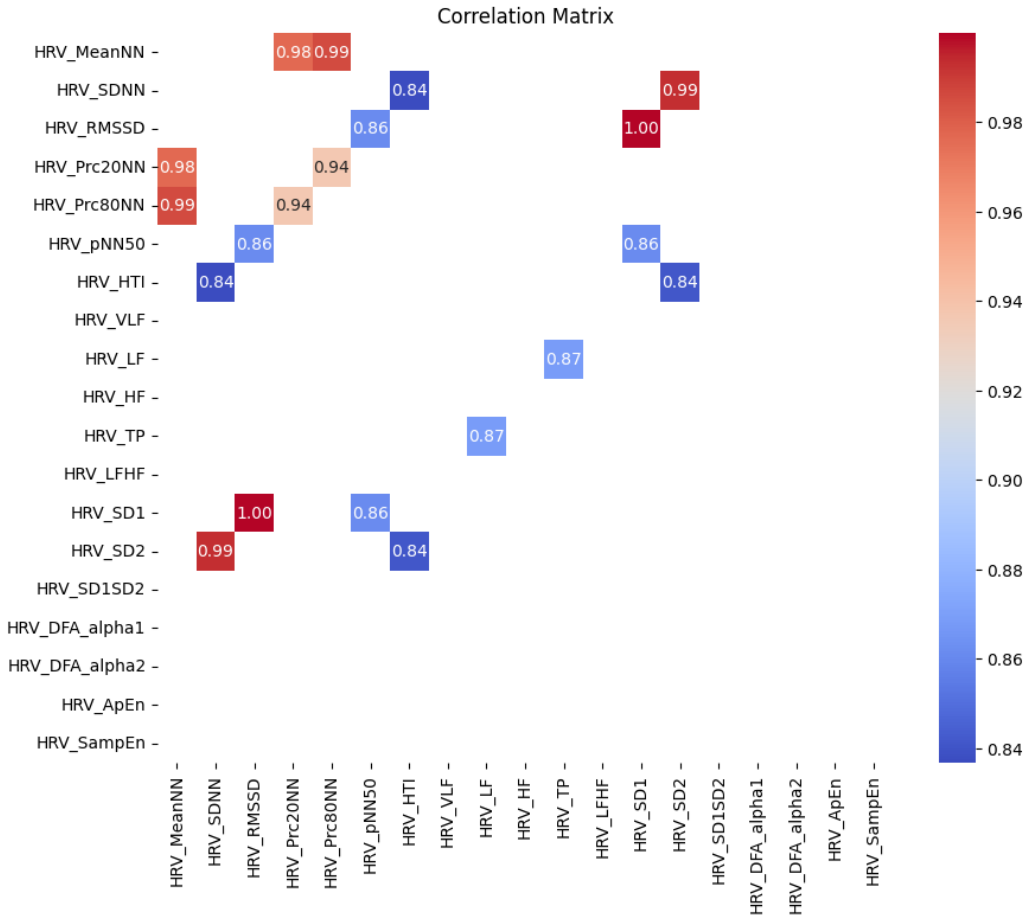
```
np.fill_diagonal(correlation_matrix.values,val=0)
positive = correlation_matrix[correlation_matrix > 0]

threshold = 0.8
correlation_hc_matrix = positive[positive>threshold]

# Visualize Correlation Matrix for highly correlated features
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_hc_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

Here, you can see first we have replaced diagonal with 0 not to get relation of feature with itself. Then, with threshold we have made a highly correlated matrix named “correlation_hc_matrix”.

Here is the visualization of the correlation matrix of highly correlated matrix.



So, we can remove one of the correlated features. But here, we have adopted different technique to treat the highly correlated features.

```
# Remove one of the features from each highly correlated pair
features_to_remove = set()

high_corr_pairs = [(0,3),(0,4),(1,6),(1,13),(2,5),(2,12),(3,4),(5,12),(6,13),(8,10)]

for feature1, feature2 in high_corr_pairs:
    columns = data.columns
    colname1 = data.columns[feature1]
    colname2 = data.columns[feature2]

    if ( colname1+'_'+colname2 not in columns ) and ( colname2+'_'+colname1 not in columns ):
        data[f"{colname1}_{colname2}"] = data[[colname1, colname2]].mean(axis=1)
    if feature1 not in features_to_remove:
        features_to_remove.add(feature1)
    if feature2 not in features_to_remove:
        features_to_remove.add(feature2)
```

Here, we have extracted all the pair of highly correlated features in variable “high_corr_pairs”. Then, by for loop we have made a new feature which includes the mean of pair of highly correlated features. And at the end we have deleted the individual features as showed below.

```
# Drop the identified features
data_filtered = data.drop(columns=data.columns[list(features_to_remove)])

data = data_filtered
```

FEATURE SELECTION

SCALING THE DATA

Before moving to the feature selection, we have assigned the data into y variable which is target feature and X variable which contains other features.

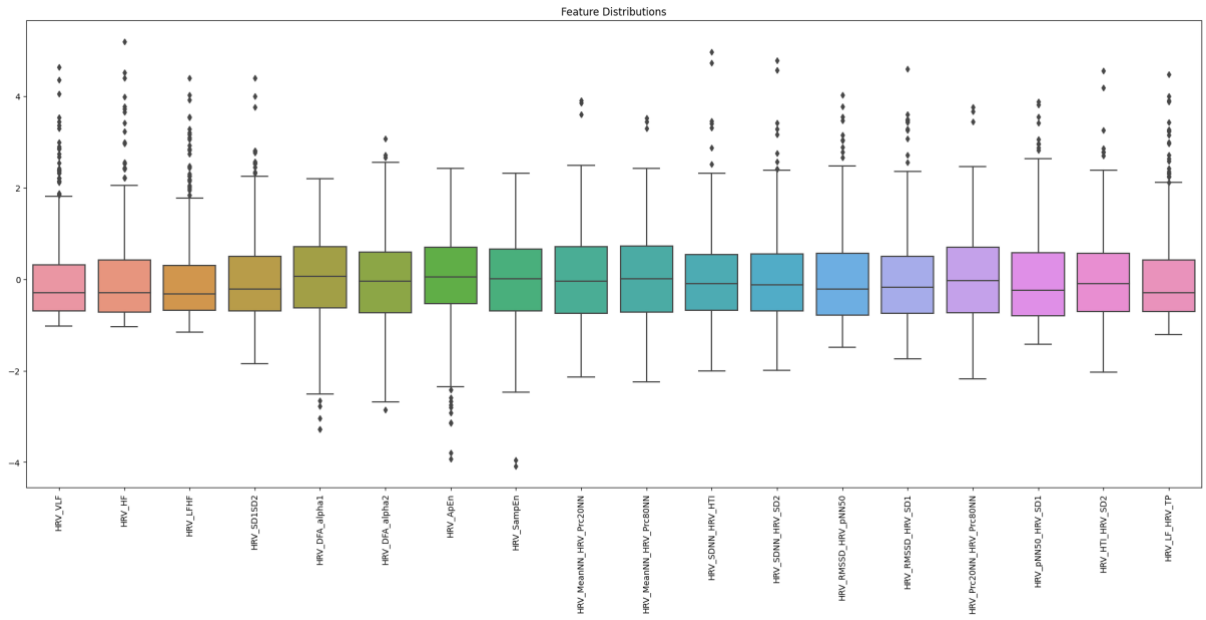
```
X = data.drop('Label', axis=1)
y = data['Label']
```

Then, we are creating an instance of the "StandardScaler" class from the "scikit-learn" library. It used for standardizing the features by removing the mean and scaling to unit variance.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled,columns=X.columns)
```

And then we are applying the standardization transformation to our original dataset "X". "fit_transforma" method calculates the mean and standard deviation from the data and then transforms the data accordingly. Then, we have made a new data frame according to original but scaled.

Now you can see in the below graph that our new data is scaled.



ANOVA TEST

Feature selection is a process of choosing a subset of the most important features from the original set of features. It aims to important model performance, reduce overfitting, and enhance interpretability.

So, to take most relevant features we used ANOVA test. ANOVA is a statistical method used to analyze the differences among group means in a sample.

ANOVA calculates an F-statistic, which is a ratio of two variance (the variance between group means and the variance within the group). It is working on hypothesis (null hypothesis (assumes that there is no significant difference between among the group means) and alternative hypothesis (states that at least one group mean is significantly different from the others).

So, ANOVA compares the calculated F-statistic to a critical value from an F-distribution table to obtain the P-value associated with the F-statistic. (if the P-value is less than a chosen significant, reject the null hypothesis which means at least one group mean is different). Here, we are using ANOVA F-statistic ('f_classif') as the scoring function.

```
# Feature Selection
# Select relevant features using ANOVA F-statistic
selector = SelectKBest(f_classif, k=5)
X_selected_fscore = selector.fit_transform(X_scaled, y)
X_fit = selector.fit(X_scaled, y)
```

Here, we used "SelectKBest" is algorithm which select the top 5 (k=5) features with the highest score based on F-statistic test. And then, by fitting we are transforming the dataset to including only the top features based on their ANOVA F-statistic scores.

```
# Display selected features
selected_features = X_scaled.columns[selector.get_support()]
print("\nSelected Features:")
print(selected_features)
```

✓ 0.1s

```
Selected Features:
Index(['HRV_HF', 'HRV_LFHF', 'HRV_DFA_alpha1', 'HRV_SampEn', 'HRV_LF_HRV_TP'], dtype='object')
```

Here, get_support() returns Boolean mask indicating which feature were selected by the "SelectKBest" method. if the corresponding feature is selected, the mask is "True" else "False". And we are selecting features from X_scaled according to the mask. And it gave us the most 5 relevant features.

Now we are going to visualize the result of the “SelectKBest” algorithm by the bar chart mentioning all the features of the dataset.

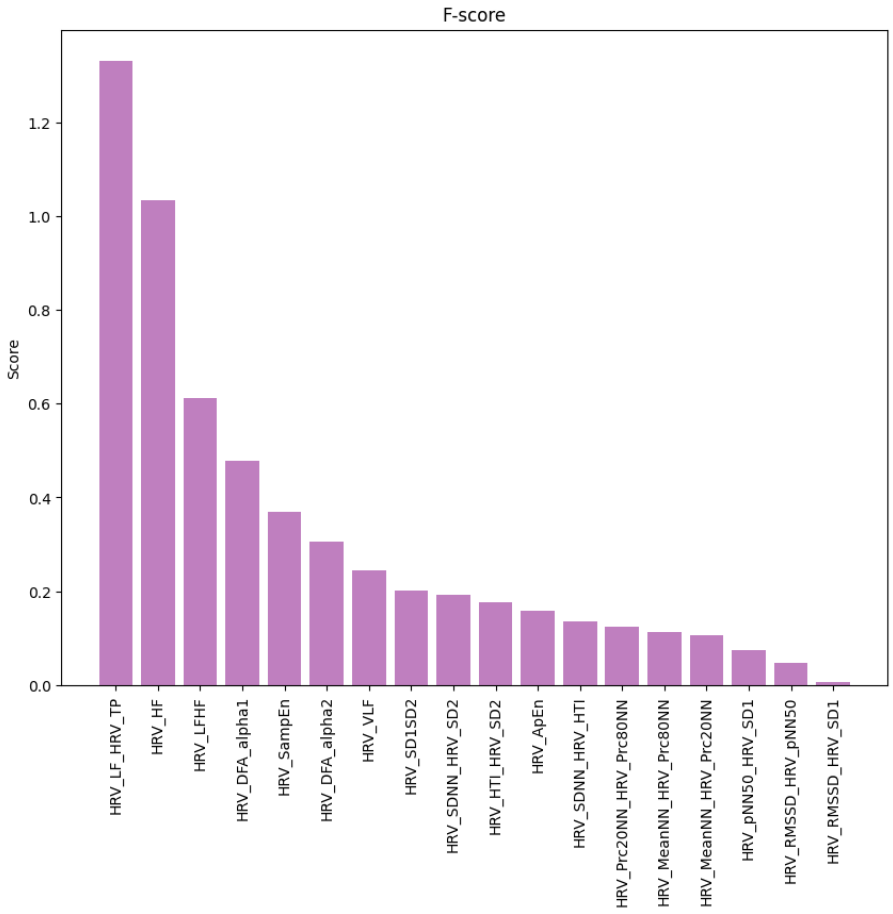
```
dfcolumns = pd.DataFrame(X_scaled.columns)
dfscores = pd.DataFrame(X_fit.scores_)

# Concatenate two dataframes for better visualization
featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Feature', 'Score']
featureScores = featureScores.sort_values(by='Score',ascending=False)

plt.figure(figsize=(10,8))
plt.bar(featureScores['Feature'], featureScores['Score'], color='purple',alpha=0.5)
plt.title('F-score')
plt.ylabel('Score')
plt.xticks(rotation=90)
plt.show()
```

First, we have created two data frames one for original data columns and another for fit data of the SelectKBest algorithm. Then, we have concatenated the two data frames along the columns (axis =1) in the variable “featureScores” which contains feature names and their corresponding F-score by function of pandas library “concat()”.

Then, for better clarity, we have given name to columns of “featureScore” as “Feature” and “Score”. After that, we have sorted that data to have good graphical representation.



RFE (Recursive Feature Elimination)

It is a feature selection technique that aims to improve the performance of the machine learning model by iteratively removing the least important features from the dataset.

Here, we are using Logistic regression model to train the entire set of features. Here, we have used logistic regression to have relation between numerical and categorical variable.

```
model = LogisticRegression()
```

In this section, we instantiate a logistic regression model using LogisticRegression from scikit-learn, assigning it to the variable model. This model will serve as the estimator used by the RFE algorithm to assess the importance of features.

```
rfe = RFE(model, n_features_to_select=5) # Selecting top 5 features
X_selected_rfe = rfe.fit(X_scaled, y)
```

Next, with the variable "rfe", we create an instance of the RFE (Recursive Feature Elimination) class from scikit-learn. And here we are selecting top 5 features (n_features_to_select=5). The logistic regression model (model) is specified as the estimator for evaluating feature importance.

Finally, using the variable "X_selected_rfe", we fit the RFE selector to our features (X) and target variable (y). During this process, RFE identifies and ranks the features based on their importance according to the logistic regression model.

```
# Print the selected features
selected_features2 = [f for f, s in zip(X_scaled.columns, X_selected_rfe.support_) if s]
print("Selected Features: ", selected_features2)
```

✓ 0.2s

```
Selected Features:  ['HRV_DFA_alpha1', 'HRV_RMSSD_HRV_SD1', 'HRV_pNN50_HRV_SD1', 'HRV_HTI_HRV_SD2', 'HRV_LF_HRV_TP']
```

Then, we are extracting the selected features after performing RFE. Here, "zip" combines the corresponding elements of two iterables. "Select_features2" list comprehension iterates through each pair of features, and it includes the feature name in the list if its support status is "True".

PCA (Principal Component Analysis)

Now the problem is overfitting. Because if we will give this data (scaled data) to machine learning model, then the model will be confused to make decision.

Principle component analysis try to reduce the problem of overfitting. Because PCA is trying to convert high-dimensionality into low-dimensionality which helps making visualize and analyze easier.

PCA aims to identify the directions in which the data varies the most. In simple words, it looks for the principal components or directions along which the data points are most spread out.

These principal components are linear combinations of original variables.

to achieve this, we have initiated the instance of PCA into "pca" variable. Then, "fit_transform" computes the principal components and performs dimensionality reduction on the input data which is scaled data (X_scaled).

```
# Apply PCA for dimensionality reduction
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

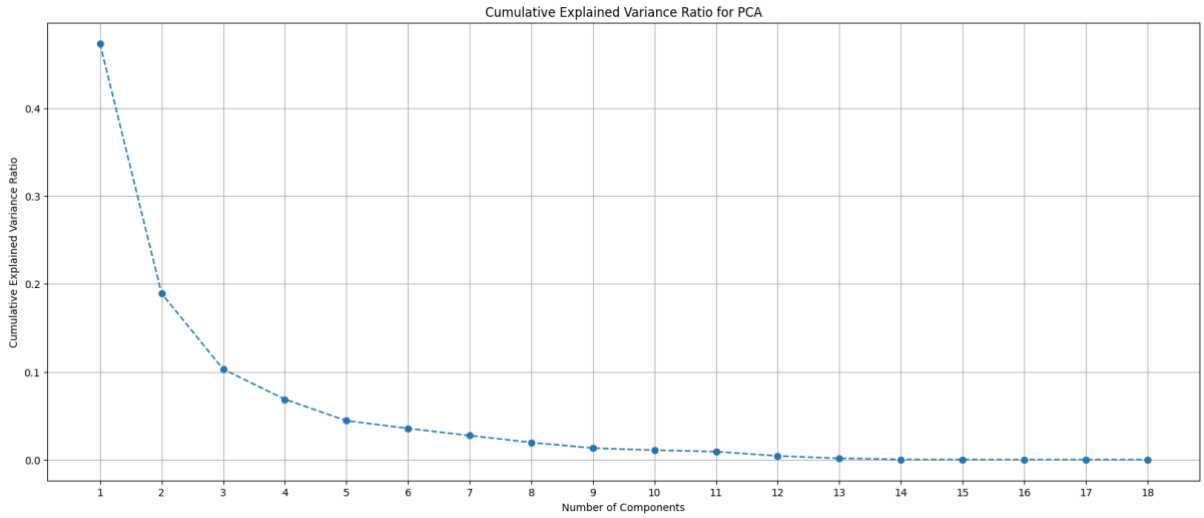
```
# Calculate the cumulative explained variance ratio
cumulative_explained_variance_ratio = pca.explained_variance_ratio_

# Plot the cumulative explained variance ratio
plt.figure(figsize=(20, 8))
plt.plot(range(1, len(cumulative_explained_variance_ratio) + 1), cumulative_explained_variance_ratio, marker='o', linestyle='--')
plt.xticks(range(1, len(cumulative_explained_variance_ratio) + 1))
plt.title('Cumulative Explained Variance Ratio for PCA')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
```

After fitting the PCA model, we have accessed the explained ratio for each principal component using "pca.explained_variance_ratio". Here, the variable "cumulative_explained_variance_raito" is a numpy array containing the cumulative explained variance ratio for each principal component. Cumulative explained variance represents the total amount of variance explained by including a certain number of principle component (in this case 2).

And then by line graph, we have made a graph of PC to identify how many components to choose with elbow technique.

Here is the line graph which show elbow at 2nd principal component.



And then we have iterated all the principal component to see how much variance the principal component is covering from the total variance, which is the explained variance (in percentage).

```
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}
labels
```

✓ 0.2s

```
{'0': 'PC 1 (47.4%)',
 '1': 'PC 2 (18.9%)',
 '2': 'PC 3 (10.3%)',
 '3': 'PC 4 (6.9%)',
 '4': 'PC 5 (4.4%)',
 '5': 'PC 6 (3.6%)',
 '6': 'PC 7 (2.7%)',
 '7': 'PC 8 (2.0%)',
 '8': 'PC 9 (1.3%)',
 '9': 'PC 10 (1.1%)',
 '10': 'PC 11 (0.9%)',
 '11': 'PC 12 (0.4%)',
 '12': 'PC 13 (0.1%)',
 '13': 'PC 14 (0.0%)',
 '14': 'PC 15 (0.0%)',
 '15': 'PC 16 (0.0%)',
 '16': 'PC 17 (0.0%)',
 '17': 'PC 18 (0.0%)'}
```

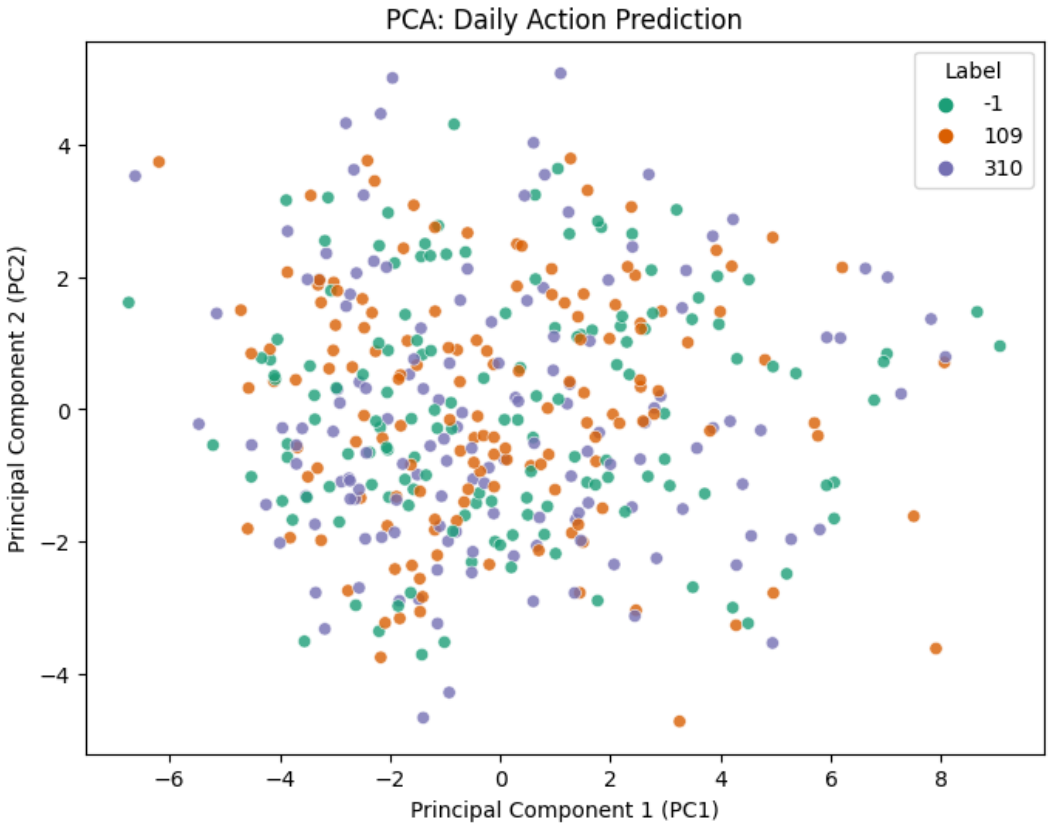
Here, “enumerate()” function is used to iterate over the elements of the explained variance ratio array along with their corresponding indices.

```
X_pca = pd.DataFrame(PCA(n_components=2).fit_transform(X_scaled),columns=['PC1','PC2'])
```

After seeing the line graph of principal component, we can say that the majority of variance is covered by 2 principal components. So, here we have selected 2 principal components (in the above code).

Then, we have used the scatter plot of that 2 principal components to see the distribution of the data points in the reduced dimensional space.

```
# Plot the data points in the reduced-dimensional space
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue='Label', data=X_pca, palette='Dark2', alpha=0.8)
plt.title('PCA: Daily Action Prediction')
plt.xlabel('Principal Component 1 (PC1)')
plt.ylabel('Principal Component 2 (PC2)')
plt.legend(title='Label', loc='upper right')
plt.show()
```



Here, the X-axis represents the values of the first-principal components, and Y-axis represents the value of the second principal components. Each point in the scatter plot corresponds to a sample in the dataset, and the colour of the points indicates the class label of the sample.

Here, we can observe that the data points are spread out across the two-dimensional space, indicating that there is some variance in the data.

However, there is no clear separation between the three classes, the plot suggests that there may be some relationship between the principal components and the emotional states.

CLASSIFICATION

Classification is a type of supervised learning in a machine learning where the goal is to predict the category or class of an input based on its features. It is like teaching a computer to categorize things into different groups or classes.

SPLITTING THE DATA INTO TRAINING AND TESTING SETS

First step in the classification process is to split the data into training and testing sets. This is done using the "train_test_split" function from scikit-learn. The data is split into 70% training and 30% testing. So, we have gave enough data (70%) to model to learn from, while also having a separate set of data (30%) to test the model's performance.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.3, random_state=42)
```

Here, "random_state" is fixed as a 42. Which ensure that the same split of the data into training and testing sets is obtained every time the code is run.

CLASSIFIER MODEL

Here, we have used the Random Forest Classifier. It is based on the concept of ensemble learning, which combines multiple decision tree to create a strong model.

The algorithm works by creating a set of decision trees, where each tree is trained on a random subset of the training data. At each node of the tree, a random subset of the feature is selected to determine the best split. After repeating this process multiple time, the final prediction is determined by taking majority vote of all the decision tree.

```
model = RandomForestClassifier()
```

CROSS-VALIDATION

The next step is to use cross-validation to tune the parameters of the model for better performance. Which is a technique used to evaluate the performance of the machine learning model by splitting the data into multiple folds and training and testing the model on each fold which is also known as K-fold cross validation.

This helps to ensure that the model is not overfitting or underfitting the data. In this case, the "GridSearchCV" function from scikit-learn is used to perform a grid search over a range of parameter values.

After tunning the parameter, the model is trained on the training data using the "randomForestClassifier" class from scikit-learn. The model is trained using the best parameters found during cross-validation.

```
# GridSearchCV for hyperparameter tuning
param_grid = { 'n_estimators': [50, 100, 200],
               'max_depth': [None, 10, 20, 30],
               'min_samples_split': [2, 5, 10],
               'min_samples_leaf': [1, 2, 4] }

model = RandomForestClassifier()
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters
best_parameters = grid_search.best_params_
print("Best Parameters:", best_parameters)

# Training the model with the best parameters
best_model = grid_search.best_estimator_
```

The "param_grid" dictionary specifies the set of hyperparameters to search over and their corresponding values.

- "n_estimators" : means the number of trees in the random forest.
- "max_depth" : the maximum depth of the trees.
- "min_sample_split" : the minimum number of samples required to split an internal node.
- "min_sample_leaf" : the minimum number of samples required to be a leaf node.

The "GridSearchCV" function is used to perform a grid search over these hyperparameters. The "cv" parameter specifies the number of cross-validation folds to.

Here, the "best_parameters" stores the best set of hyperparameters found by the grid search and the "best_model" stores the random forest classifier with the best set of hyperparameter.

PREDICTION

The model's performance is evaluated by making prediction on the testing data and comparing them to the actual values. The "predict" function from scikit-learn is used to make prediction on the testing data.

```
# Predictions
predictions = best_model.predict(X_test)
```

Here, we trained our model on the training set. Now on the X_test set, we are predicting or testing the model's performance.

RESULTS

CLASSIFICATION REPORT

The classification report is used to evaluate the performance of the model. The classification report provides metrics such as precision, recall, F1-score, and support for each class in the data. The "classification_report" function from scikit-learn is used to generate the classification report.

Precision : is the ratio of true positive prediction to the total predicted positive instances. It measures the proportion of correct positive prediction out of all positive predictions made.

- Recall : the ratio of true positive to the total actual positive instances.
- F1-score : is the harmonic mean of precision and recall.
- Support : is the number of instances for each class in the test set.

We also have macro average which is unweighted mean of the metrics for each class and weighted average which is calculated as the weighted mean of the metrics for each class.

```
print("Classification Report:")
print(classification_report(y_test, predictions))
```

So after feature selection, we have perform a machine learning on the output of some feature selection techniques and here is the classification report for those techniques.

Classification on ANOVA F-score result :

Classification Report:

	precision	recall	f1-score	support
-1	0.32	0.21	0.25	48
109	0.33	0.49	0.40	35
310	0.35	0.36	0.35	45
accuracy			0.34	128
macro avg	0.33	0.35	0.33	128
weighted avg	0.33	0.34	0.33	128

Here, you can see the total accuracy is 34%. But the weighted average is changing according to weight.

Classification on RFE (with Logistic Regression) result :

Classification Report:

	precision	recall	f1-score	support
-1	0.30	0.23	0.26	48
109	0.28	0.34	0.31	35
310	0.38	0.40	0.39	45
accuracy			0.32	128
macro avg	0.32	0.32	0.32	128
weighted avg	0.32	0.32	0.32	128

Here, the total accuracy for the RFE feature selection technique is 32%.

Classification on PCA result :

Classification Report:

	precision	recall	f1-score	support
-1	0.37	0.33	0.35	48
109	0.21	0.29	0.24	35
310	0.34	0.29	0.31	45
accuracy			0.30	128
macro avg	0.31	0.30	0.30	128
weighted avg	0.32	0.30	0.31	128

Here, the total accuracy is 30%.

We have also measured the classification report for the data with all features. Which is showed below.

```
Classification Report:
              precision    recall  f1-score   support

          -1       0.28      0.17      0.21        48
         109       0.24      0.43      0.31        35
         310       0.33      0.27      0.30        45

    accuracy                           0.27       128
   macro avg       0.28      0.29      0.27       128
weighted avg       0.29      0.27      0.27       128
```

Here, you can see the accuracy is 27%. But after performing some feature selection techniques we can improve the accuracy of our model by choosing most relevant features. Like ANOVA F-score (34%), RFE (32%), and PCA (30%).

CONFUSION MATRIX

The confusion matrix is used to visualize the performance of the model. The confusion matrix provides a visual representation of the true positive, true negative, false positive ,and false negative values. The "confusion_matrix" function is used to generate the confusion matrix. Which includes a detailed breakdown of the model's prediction, showing how many instances of each class were correctly or incorrectly classified (true positive, true negative, false positive, false negative).

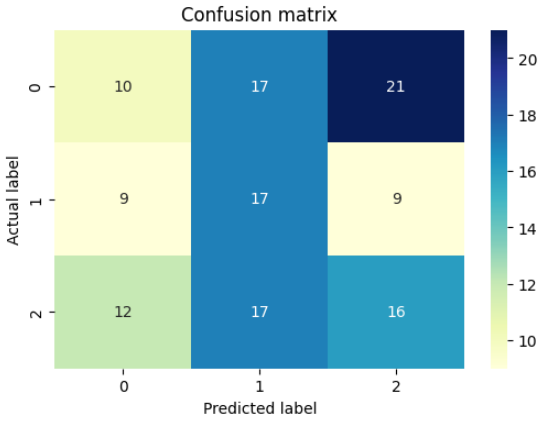
```python
# Evaluation
conf_matrix = confusion_matrix(y_test, predictions)

# Visualization of Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(pd.DataFrame(conf_matrix), annot=True, cmap="YlGnBu", fmt='g')
plt.title('Confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```

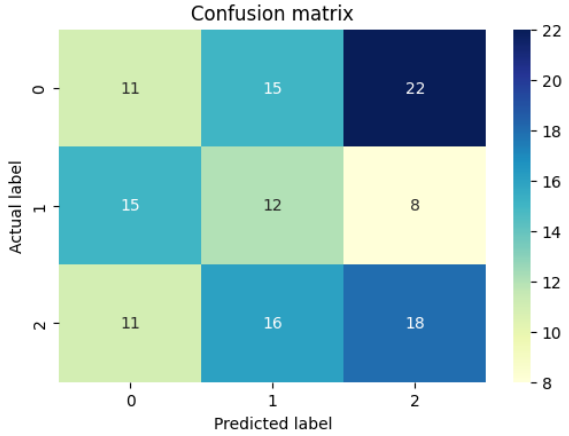
Here, the "confusion_matrix" function takes two arguments: the true labels ('y_test') and the predicted labels ('prediction'). And then, it will provide a table that summarizes the performance of a classifier by comparing the true labels with the predicted labels.

Below, we have mentioned the confusion matrix for the some different subset of the dataset which is taken from feature selection techniques same as showed in the classification report

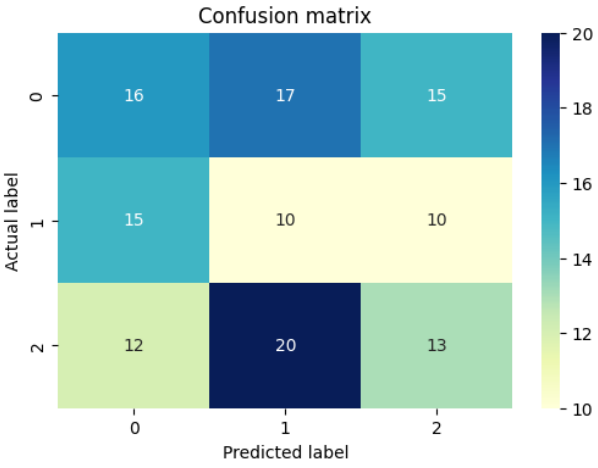
For ANOVA F-score test :



For RFE test :



For PCA test :



Here, in this matrix, diagonal represents the true positive means correctly predicted and others are not.

CONCLUSION

In conclusion, this data mining project aimed to analyse a dataset using various techniques such as data cleaning, exploration, feature selection, and classification. We started by importing essential Python libraries and loading the dataset. The data cleaning process ensured there were no missing values or outliers, enhancing the reliability of our analysis.

The exploration phase involved visualizing numerical and categorical features, creating pairwise scatter plots, and generating a correlation matrix to understand relationships between variables and also replacing the highly correlated features by their average. Feature selection techniques, including ANOVA F-statistic, Recursive Feature Elimination (RFE), and Principal Component Analysis (PCA), helped identify the most relevant features for our analysis.

Finally, we applied a Random Forest Classifier for classification, evaluating its performance with accuracy metrics and a confusion matrix.

The project demonstrated the importance of through data exploration and pre-processing in enhancing the effectiveness of machine learning models.