Department of Mathematics and Computer Science

Faculty of Data Analysis

UNIME

Data Mining and Analytics
Project Report
POPANE Dataset Analysis

Student: Alina Atayeva (533021)

Student: Alikhan Alashybay (536353)

Professor: Gennaro Tartarisco

Academic Year 2023/2024

Table of Contents

1. Introduction

1.1. Dataset description

The "Dataset_Study3" is derived from the publicly accessible POPANE dataset, centering its focus on unraveling the psychophysiological responses to both positive and negative emotions among 1157 healthy young adults. The dataset encapsulates a comprehensive examination spanning seven distinct studies, with a particular emphasis on "Study 3." This specific study zone delves into the intricate cardiovascular responses of 147 subjects. The overarching objective of this investigation is to detect and distinguish patterns that facilitate the prediction of daily actions. This is achieved by scrutinizing the interplay between cognitive processes, encompassing evaluation, motivations, and intentions, and the corresponding physiological responses elicited during a lab-based social task. The core emphasis of this analysis revolves around evaluating motivations, especially in the context of gratitude interventions.

The procedure followed by the study is illustrated in the next figure:

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)

After a 3-minute baseline, the participants were asked to send two text messages: one gratitude expression message and one neutral message. The order of sending messages was counterbalanced using a randomization algorithm. After sending the message individuals waited for 3 minutes as the time needed for physiological recovery.

The dataset is composed of 426 observations (142 subjects x 3 conditions: baseline, gratitude, and neutral). Indeed, 5 subjects were excluded because the ECG signal was missing. There are 3 labels indicating the different conditions from which the features were extracted. Specifically, baseline condition (-1), gratitude (310), and neutral state (109). Features are 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.

Overall, the dataset contains 19 features, of which 7 are time-domain features, 5 are frequency-domain features, and 7 are non-linear features.

1.2. Objectives

The primary goal of this study is to glean actionable insights into the complex interrelationship between psychophysiological responses and cognitive facets, with a specific lens on motivations tied to expressions of gratitude. The dataset's richness, drawn from a diverse array of individuals and scenarios, positions it as a valuable resource for understanding the dynamics that underlie emotional responses and their potential impact on subsequent behavior. By leveraging advanced analytical techniques, we aim to uncover meaningful patterns that can contribute to the predictive modeling of daily actions based on the intricate interplay of cognitive and physiological dimensions.

To analyze the dataset, various data mining and analysis methods must be used, such as import, cleaning, exploration, visualization, and normalization. Additionally, we aim to employ various techniques, including feature selection, principal component analysis, clustering, and classification, to build a robust model capable of predicting different emotional states. Ultimately, it is crucial to uncover the most efficient strategy to build the most accurate model.

2. Data exploration and EDA

Before delving into the analysis, all necessary libraries must be imported. The figure below only shows the necessary libraries for the data exploration and Exploratory Data Analysis (EDA) step. Other libraries that will be needed in the following steps are imported in each step respectively, and the screenshots of the scripts will be provided.

```
1  import pandas as pd
2  import numpy as np
3
4  import matplotlib.pyplot as plt
5  import seaborn as sns
```

After that, the dataset is loaded. Loading the dataframe requires the “Pandas” library and the path to the “.csv” file on the local PC.

```
1  df = pd.read_csv('/Users/alinaatayeva/Desktop/Data mining&analytics/Project/Dataset_Study3.csv')
```

Hence, the shape of the dataset was checked to see that the dataset we have coincides with the description and that the data frame was loaded correctly. The first 5 rows of the dataset are displayed to see all the information and the columns, which are the 19 features plus the labels.

```
1  df.shape
```

Python

```
(426, 20)
```

```
1  df.head()
```

Python

	HRV_MeanNN	HRV_SDNN	HRV_RMSSD	HRV_Prc20NN	HRV_Prc80NN	HRV_pNN50	HRV_HTI	HRV_VLF	HRV_LF	HRV_HF	HRV_TP	HRV_LFHF	HRV_SD1	HRV_SD2	HRV_...
0	838.708920	77.858966	42.966689	773.2	907.4	21.962617	15.285714	135.219052	1984.443232	941.576892	3083.350848	2.107574	30.452465	105.943750	0
1	899.929293	80.454862	50.910712	838.6	960.4	24.623116	15.307692	77.009053	1012.790954	712.205460	1835.167749	1.422049	36.089337	108.046467	0
2	827.319444	64.428505	40.450375	766.4	881.8	20.276498	15.500000	129.751167	1924.529470	761.893623	2848.723565	2.525982	28.668833	86.661784	0
3	768.630901	76.803474	43.512943	699.0	837.4	23.076923	16.714286	163.822696	4643.049228	806.486463	5625.889299	5.757132	30.832839	104.157364	0
4	807.432432	82.942629	49.326201	731.0	879.2	18.834081	13.937500	132.967712	2348.301339	567.338064	3066.742236	4.139157	34.955488	111.937167	0

For the statistical information df.describe() function is used, which returns a table with all the statistical information regarding the dataset.

```
1  df.describe()
```

Python

	HRV_MeanNN	HRV_SDNN	HRV_RMSSD	HRV_Prc20NN	HRV_Prc80NN	HRV_pNN50	HRV_HTI	HRV_VLF	HRV_LF	HRV_HF	HRV_TP	HRV_LFHF	HRV_SD1	H...
count	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426.000000	426
mean	760.770070	60.982323	38.275327	711.938967	807.670423	13.606149	12.476551	242.663063	1948.718982	885.662291	3123.250101	3.253009	27.124319	81
std	94.046679	31.201040	31.129139	87.268890	104.327001	13.232075	3.688746	1883.692791	10264.804862	2547.208263	14487.986145	2.977534	22.067763	3
min	555.599379	16.506371	5.812904	533.000000	568.600000	0.000000	4.625000	4.737018	49.902409	25.658985	194.346766	0.171808	4.116741	2
25%	692.535430	44.568471	22.849493	649.200000	733.100000	2.921488	9.815909	37.125944	548.847446	183.689415	942.674950	1.351391	16.188131	59
50%	757.405554	56.893008	32.411485	712.400000	809.800000	9.732418	12.114379	77.209885	954.991547	399.794185	1533.560411	2.250180	22.968826	7
75%	829.424839	71.232137	44.493817	772.850000	881.300000	20.544258	14.696429	143.858800	1592.353978	805.296999	2610.418292	3.922320	31.529098	9
max	1105.870370	389.188189	382.526115	1069.800000	1159.000000	64.814815	26.500000	33458.813731	177202.470783	32325.575897	243949.629472	19.146016	271.200496	48

Further, we come to one of the most essential steps of the analysis: checking for missing values, duplicated, and data types. This step is important because all of the next steps rely on it. In case we find some missing values and or duplicates of some values, it will affect our further analysis.

First, we check for data types. In some cases when we find that a dataset contains different types of data, all data must be treated, and the data types must be normalized. Usually, all data is either set to be in either of the two data types: integers or floating-point numbers. Checking the data types of our datasets using df.dtypes function shows us that all of the data is stored in floating-point numbers and the labels are integers.

```
1  df.dtypes
```

```
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
dtype: object
```

On the left side below we see the df.isnull().sum() function, which allows us to see whether or not the dataset contains null (or NaN) values for each column. As it is depicted in the figure, no null values are present, so all the values are in place. This means that we can proceed to analyze the dataset without filling in the missing values.

```
1  df.isnull().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
dtype: int64
```

```
1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426 entries, 0 to 425
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   HRV_MeanNN      426 non-null    float64
 1   HRV_SDNN        426 non-null    float64
 2   HRV_RMSSD       426 non-null    float64
 3   HRV_Prc20NN     426 non-null    float64
 4   HRV_Prc80NN     426 non-null    float64
 5   HRV_pNN50       426 non-null    float64
 6   HRV_HTI         426 non-null    float64
 7   HRV_VLF         426 non-null    float64
 8   HRV_LF          426 non-null    float64
 9   HRV_HF          426 non-null    float64
 10  HRV_TP          426 non-null    float64
 11  HRV_LFHF        426 non-null    float64
 12  HRV_SD1         426 non-null    float64
 13  HRV_SD2         426 non-null    float64
 14  HRV_SD1SD2      426 non-null    float64
 15  HRV_DFA_alpha1  426 non-null    float64
 16  HRV_DFA_alpha2  426 non-null    float64
 17  HRV_ApEn        426 non-null    float64
 18  HRV_SampEn      426 non-null    float64
 19  Label           426 non-null    int64
dtypes: float64(19), int64(1)
memory usage: 66.7 KB
```

Next to the df.isnull().sum() function, there is a df.info() function that displays overall information on the number of non-null values for each column and the data type. It is just yet another function to display the information discussed above.

```
1  df['Label'].value_counts()
```

```
Label
-1      142
 310    142
 109    142
Name: count, dtype: int64
```

Just to make sure that we have no values that are missed in some labels and are in excess in other ones, we may check the amount of data per label. After it was confirmed, we move on to check for duplicates.

```
1  df.duplicated().sum()
```

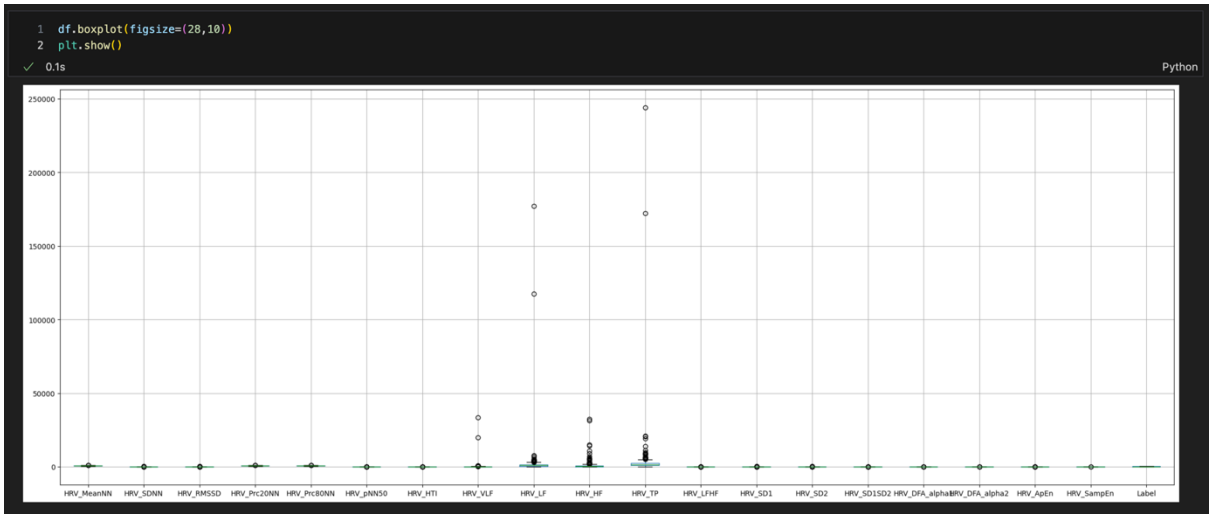
```
0
```

No duplicates of the values have been found. Overall, the dataset is put together and clean.

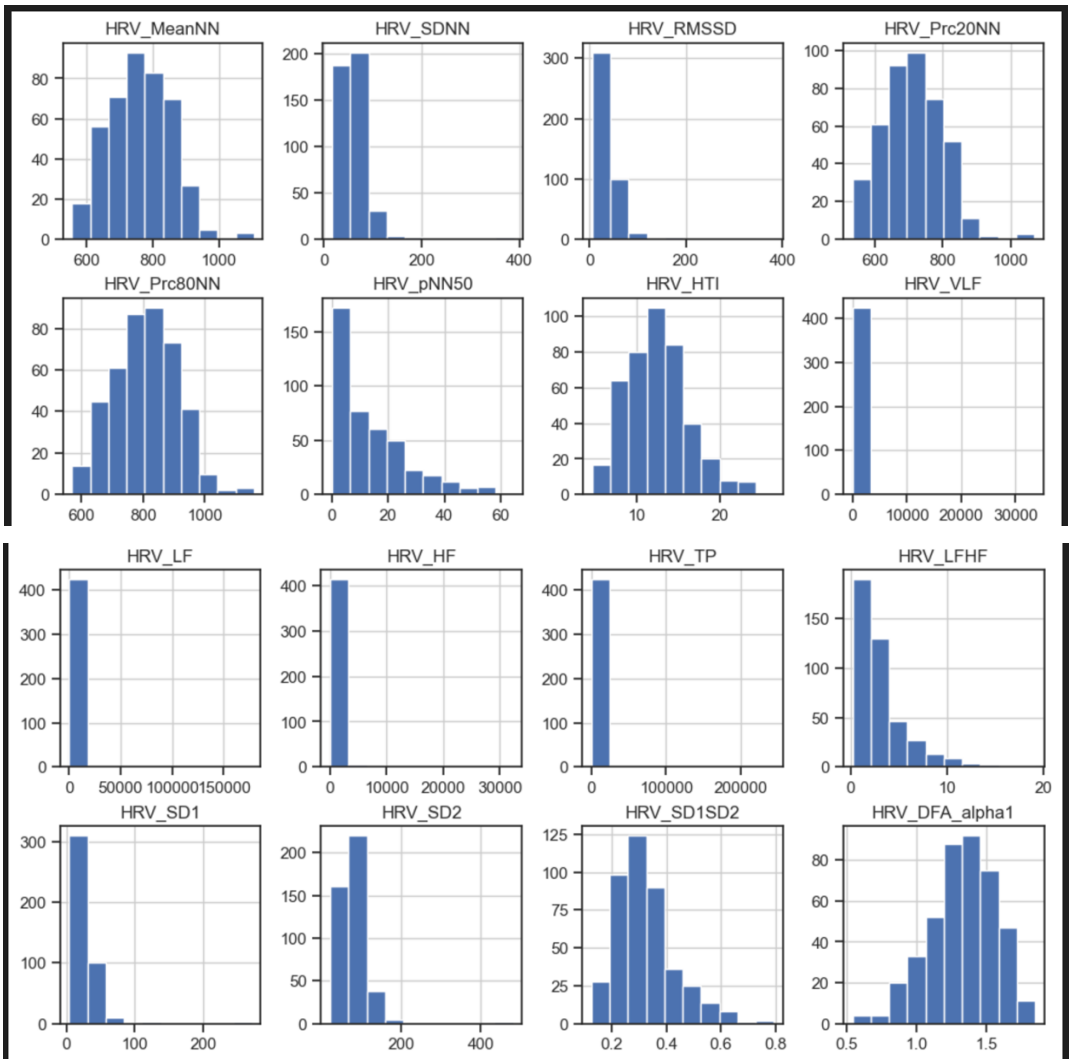
3. Data visualization

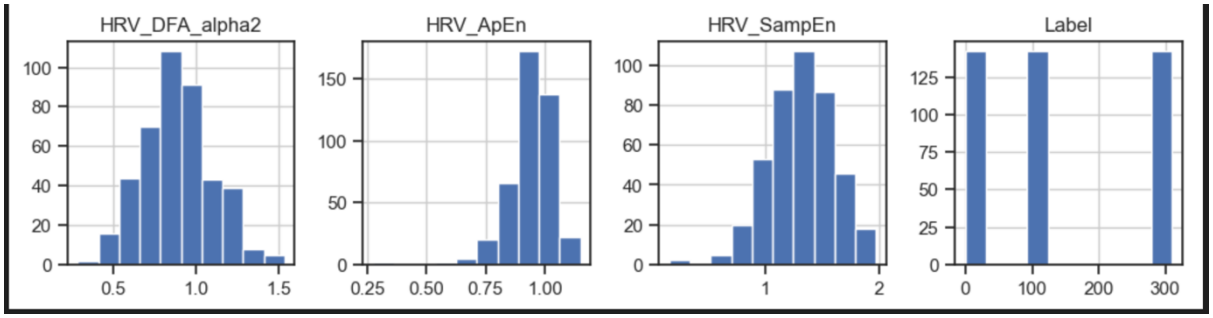
Moving on to the next part, the data needs to be properly visualized. In this step, we see how our data is distributed and whether or not it has some outliers.

A boxplot was built to display all of the features and their distribution. As it can be seen, there are some outliers, and the data is not normally distributed.



For a better look of each feature, a histogram can be built. However, we still see that the features with outliers are not properly represented.





4. Data preprocessing

Now, when it comes to treating outliers, we first have to consider a few factors. The first factor to consider is – the nature of the data. ECG signals often have inherent variability, and outliers might represent legitimate physiological information. The next factor is the impact on analysis. Assessing how outliers affect the analysis is crucial. We have to consider if they significantly skew statistical measures or adversely impact model performance. The third factor is the objective of analysis. We have to consider the goal of the analysis, and if outliers are likely to distort predictions or hinder pattern recognition. Finally, justification and documentation have to be taken into account. Regardless of the choice, provide a robust rationale and document the approach taken to handle or retain outliers.

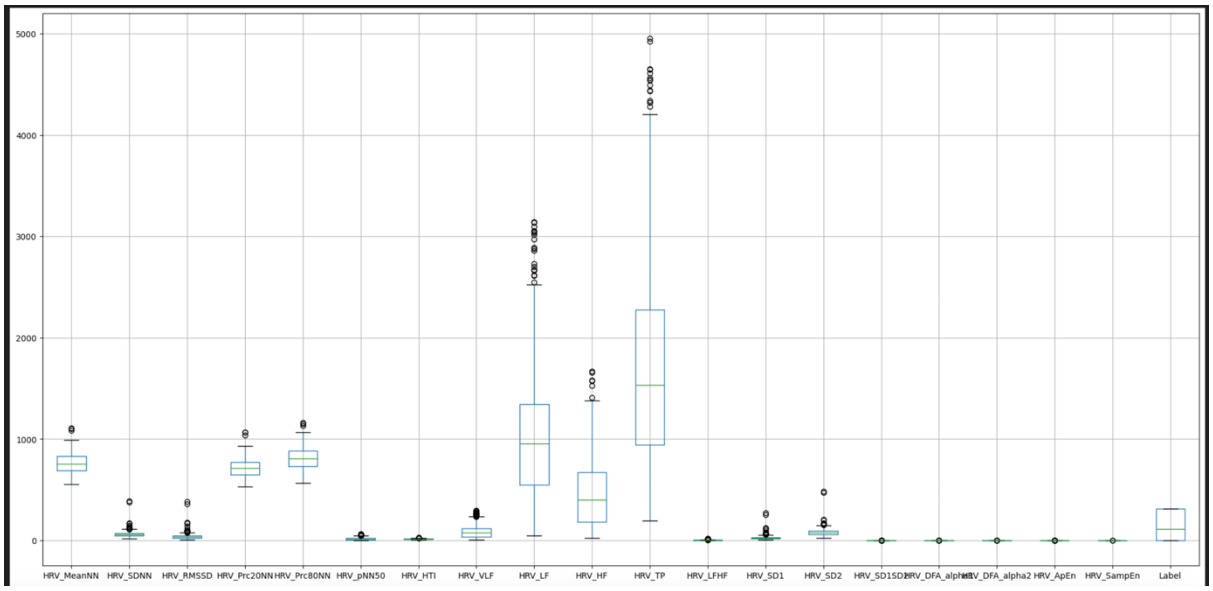
Since no documentation has been found regarding the outliers, the information about the nature of the data was considered. What are outliers in EEG data? Basically, any signal not originating from the brain can be called an outlier. Outliers can make the EEG signal completely unreadable. In the context of electrocardiogram (ECG) signals, outlier values typically refer to data points that deviate significantly from the expected or typical pattern. Outliers in ECG signals can be caused by various factors, including artifacts, noise, or abnormal physiological conditions. While some outliers may be the result of true physiological phenomena (such as ectopic beats or arrhythmias), the term "outlier" in ECG analysis often implies values that are not part of the normal or expected range. Identifying and handling outliers in ECG signals is crucial for accurate analysis and interpretation, as they can affect the quality of derived features and metrics. Hence it was decided to treat the most obvious outliers, that distort the visualization of the data, and are most probably not a part of the normal or expected range.

To treat the outliers the following function was built. The function uses the interquartile ranges to detect the lower and upper bounds of the data and removes everything beyond those bands.

```python
def detect_outliers(df,feature_name):

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

    IQR = Q3 - Q1

    lwr_bound = Q1 -1.5 * IQR
    upp_bound = Q3 +1.5 * IQR

    ls = df.index[np.logical_or(df[feature_name]<lwr_bound,
                                df[feature_name]>upp_bound)]
    return ls
```

✓ 0.0s

The figure below shows the output result of the outlier treatment procedure.



5. Data correlation

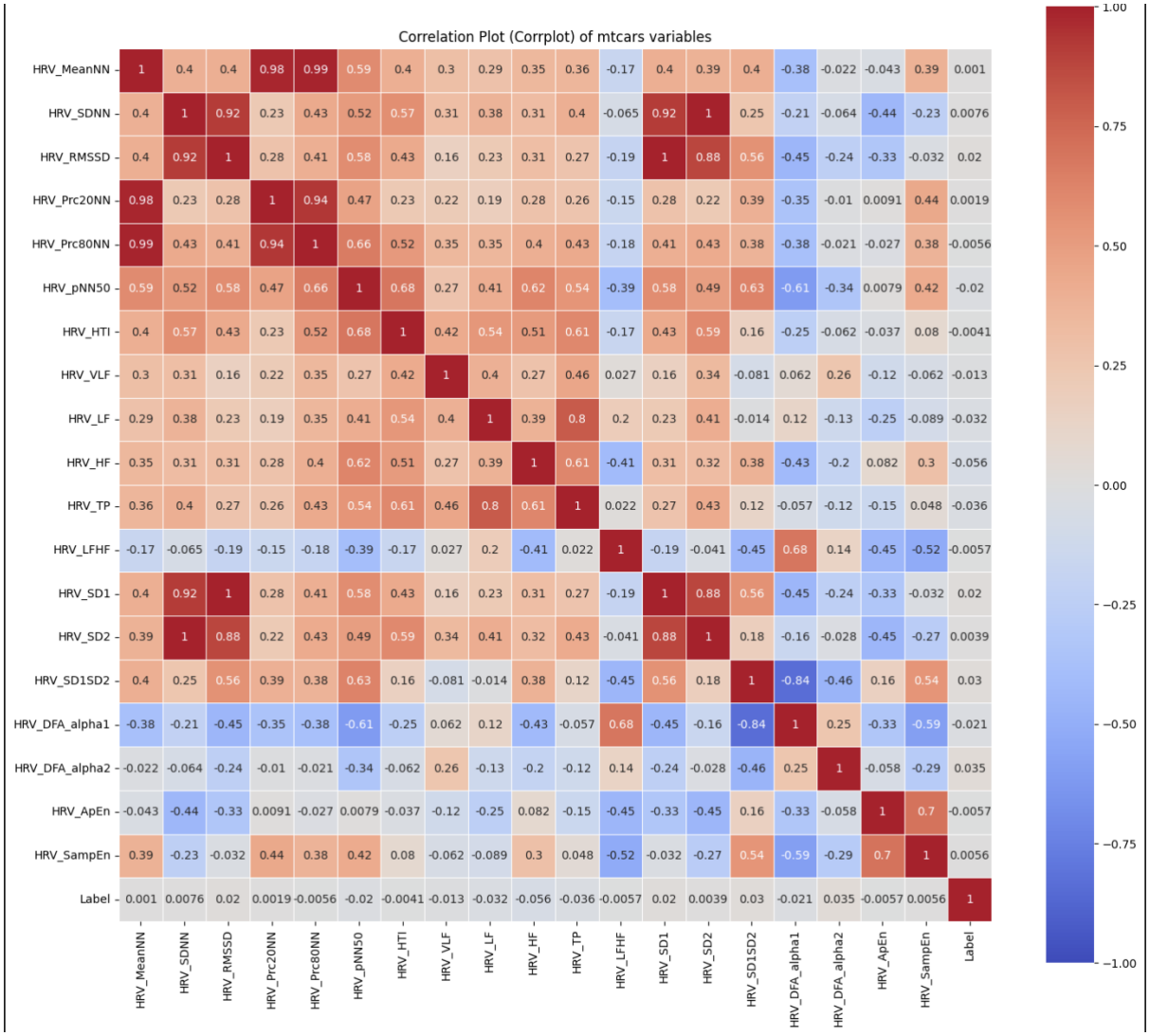
By constructing a correlation matrix, we aim to gain insights into the relationships between these features. The correlation matrix will reveal the strength and direction of connections, helping identify patterns and potential multicollinearity among variables. Detecting multicollinearity is crucial as it influences the stability of predictive models. Additionally, the matrix aids in variable selection, guiding decisions on the relevance of features for the analysis. Visualization through heatmaps facilitates the intuitive interpretation of complex relationships. Moreover, the matrix assists in identifying outliers, checking assumptions of

linearity, and addressing missing values—all essential steps in ensuring the dataset's quality and preparing it for accurate and insightful analysis.

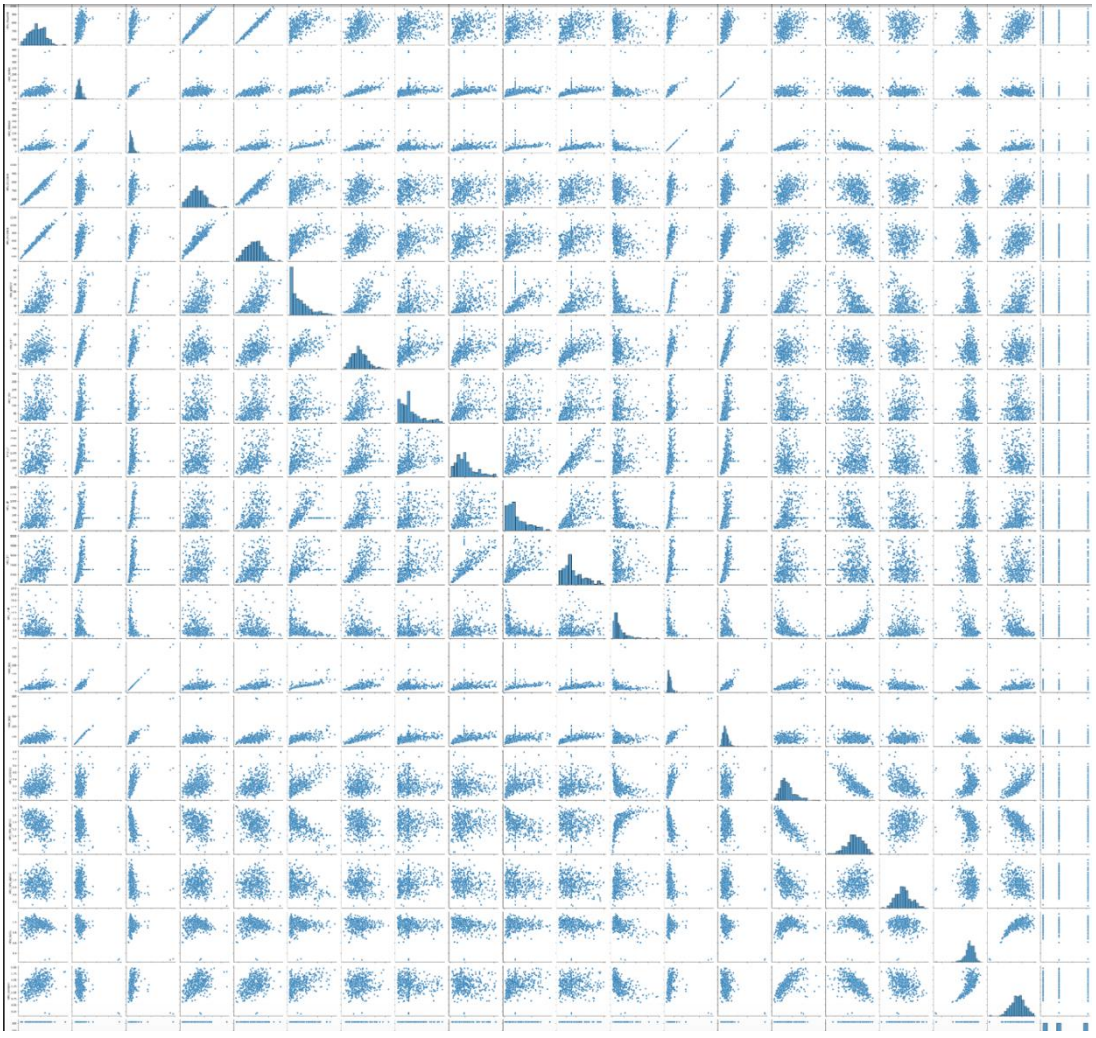
To construct a correlation matrix, the label column of the dataset must be deleted using the df.drop() function.

```
1  df_clean=df.drop('Label',axis=1)
✓  0.0s
```

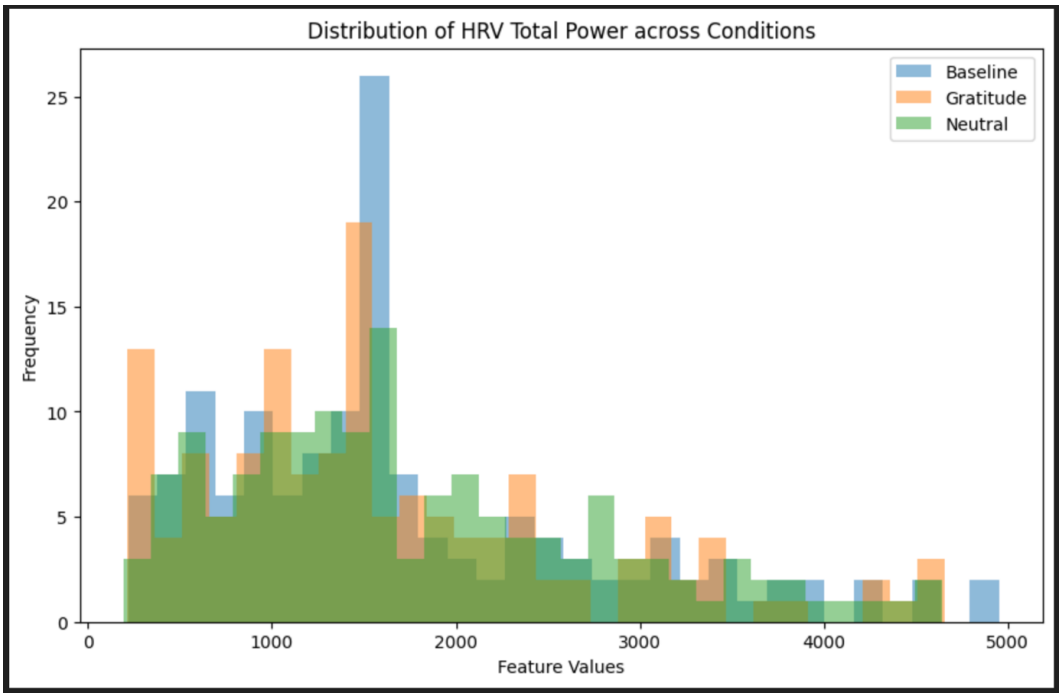
The correlation plot below depicts the correlation ratio of each feature. The diagonal shows the correlation of each feature with itself; hence the correlation factor is the highest possible, with a maximum of 1.

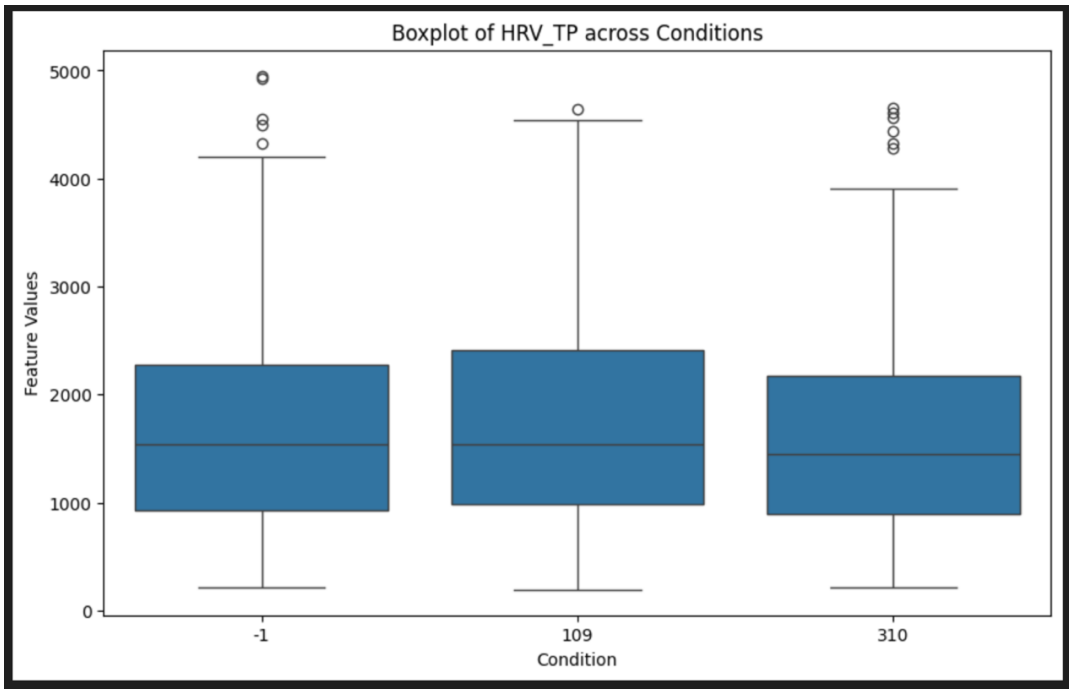


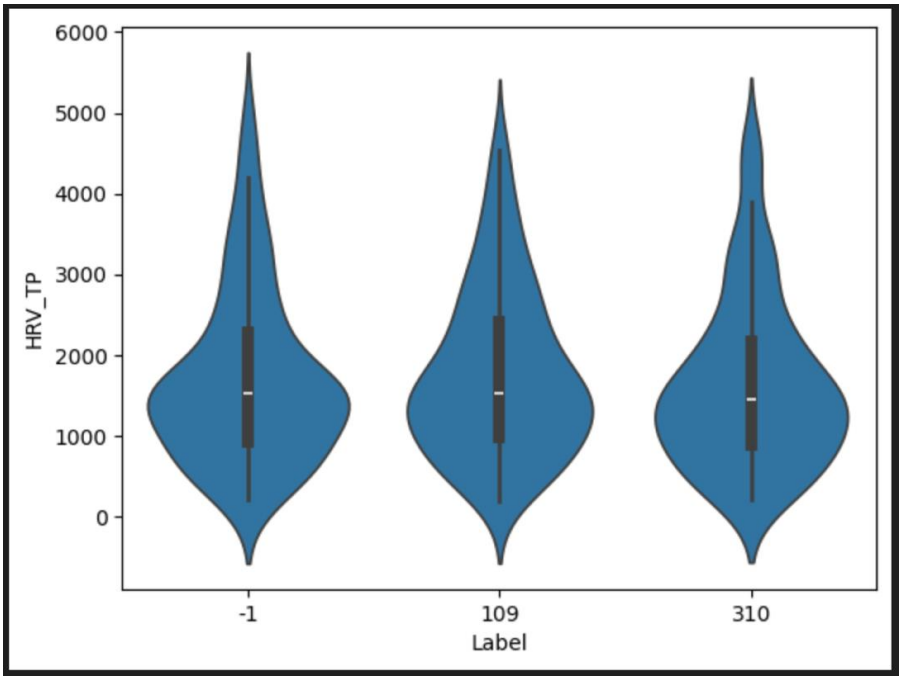
The correlation matrix allows us to explore the strength and direction of associations between different variables, revealing patterns, and dependencies within the dataset. A correlation coefficient close to 1 signifies a strong positive relationship, indicating that as one variable increases, the other tends to increase as well. Conversely, a coefficient near -1 denotes a strong negative correlation, implying an inverse relationship where one variable decreases as the other increases. A correlation matrix aids researchers in identifying potential multicollinearity, gauging the impact of one variable on another, and guiding feature selection processes. In the pursuit of predicting daily actions based on cognitive and physiological responses, a thorough examination of correlations provides valuable insights into the interconnectedness of psychophysiological features, contributing to the overall interpretability and effectiveness of subsequent analyses and predictive models.
Below is presented the graphical version of the correlation matrix, just to see the distribution of data.



Further, I decided to explore the total power of heart rate variability, using different representation methods.







6. Principal Component Analysis

Principal Component Analysis (PCA) is a statistical method used for dimensionality reduction in multivariate datasets. The primary goal of PCA is to transform the original features of a dataset into a new set of uncorrelated variables called principal components. These principal components are linear combinations of the original features and are ordered by the amount of variance they capture. PCA aims to simplify the complexity of the data by highlighting its most significant patterns and reducing the number of variables while retaining essential information.

PCA becomes particularly crucial when dealing with large datasets. In datasets with numerous features, the inherent correlations between variables can lead to redundancy and computational challenges. By applying PCA, one can effectively reduce the dimensionality of the dataset, eliminating redundant information and retaining the most critical aspects. This not only simplifies subsequent analyses but also speeds up computational processes. Additionally, PCA helps identify dominant patterns and trends within the data, contributing to a more concise and interpretable representation of the dataset.

Performing PCA before feature selection offers several advantages. Feature selection techniques, such as filtering or wrapper methods, are often computationally expensive on large datasets due to the sheer volume of features. PCA aids in reducing this computational burden by transforming 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. This sequential approach allows for a more streamlined and efficient feature selection process, leading to a refined subset of features that capture the essential information within the dataset.

To perform the PCA, we first need to import all the necessary libraries. These include sklearn decomposition and preprocessing libraries, from which we import PCA and StandardSclaer modules, respectively.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

Then we separate features from labels. Here we create two variables “X” and “y”, where X is the set of features and y is the label set.

```
# Separating features from labels
X = df.drop('Label', axis=1)
y = df['Label']
```

After that, we standardize the features as shown below.

```
# Standardize the features (recommended before PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

And we perform PCA.

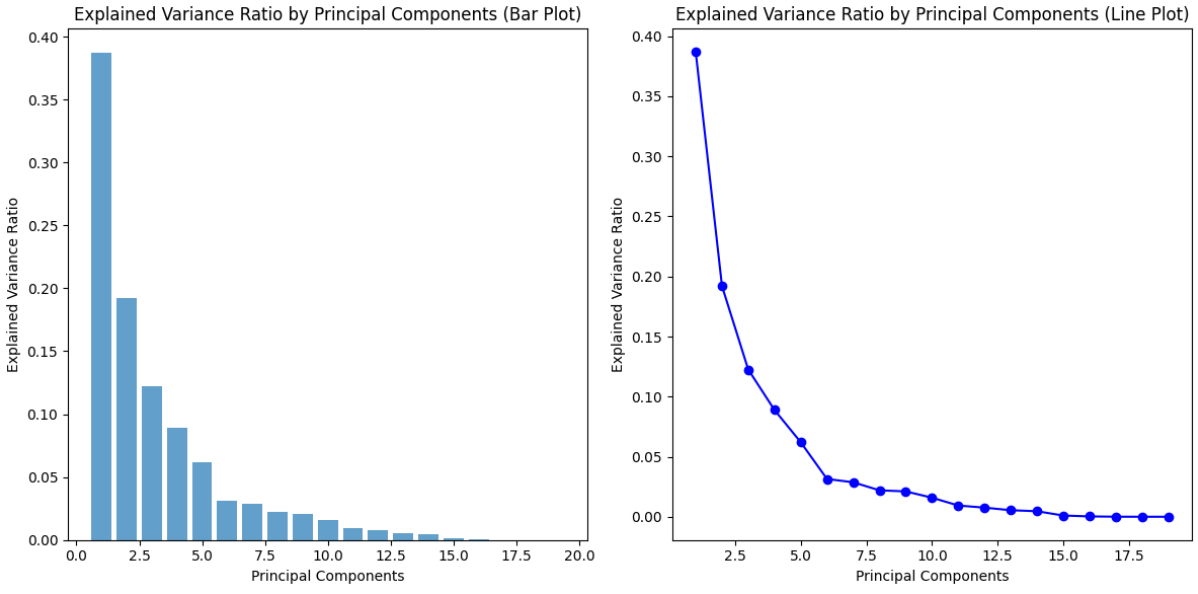
```
# Perform PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

Hence, we create a dataframe to store PCA results, and by adding the labels back to the result dataframe, we graph the data.

```python
# Creating a DataFrame to store PCA results
pca_df = pd.DataFrame(data=X_pca, columns=[f'PC{i}' for i in range(1, pca.n_components_ + 1)])

# Concatenate PCA results with labels if needed
pca_df_with_labels = pd.concat([pca_df, y.reset_index(drop=True)], axis=1)

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"Explained Variance Ratio: {explained_variance}")

# Visualize explained variance ratio
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Bar plot
axes[0].bar(range(1, len(explained_variance) + 1), explained_variance, alpha=0.7)
axes[0].set_xlabel('Principal Components')
axes[0].set_ylabel('Explained Variance Ratio')
axes[0].set_title('Explained Variance Ratio by Principal Components (Bar Plot)')

# Line plot
axes[1].plot(range(1, len(explained_variance) + 1), explained_variance, marker='o', linestyle='-', color='b')
axes[1].set_xlabel('Principal Components')
axes[1].set_ylabel('Explained Variance Ratio')
axes[1].set_title('Explained Variance Ratio by Principal Components (Line Plot)')

plt.tight_layout()
plt.show()
```

```
Explained Variance Ratio: [4.32794736e-01 2.43440223e-01 1.13789568e-01 6.84959384e-02
 5.34676660e-02 3.22042087e-02 2.14006647e-02 1.20699518e-02
 9.41297382e-03 4.76048910e-03 4.38006899e-03 2.69065240e-03
 5.96526966e-04 2.62385998e-04 1.75150578e-04 5.25387903e-05
 5.07966155e-06 1.17643771e-06 7.75166038e-10]
```

We get a graph of the explained variance ratio by Principal Components. From this point on the most valuable principal components can be chosen for further analysis.



Using the elbow point method we could choose the 8th component point, leaving us with 8 principal components. However, from the bar graph, we see that it will be more efficient for

us to choose the first 5 components of the PCA. Other components are negligible in comparison to the first five.

```
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
```

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

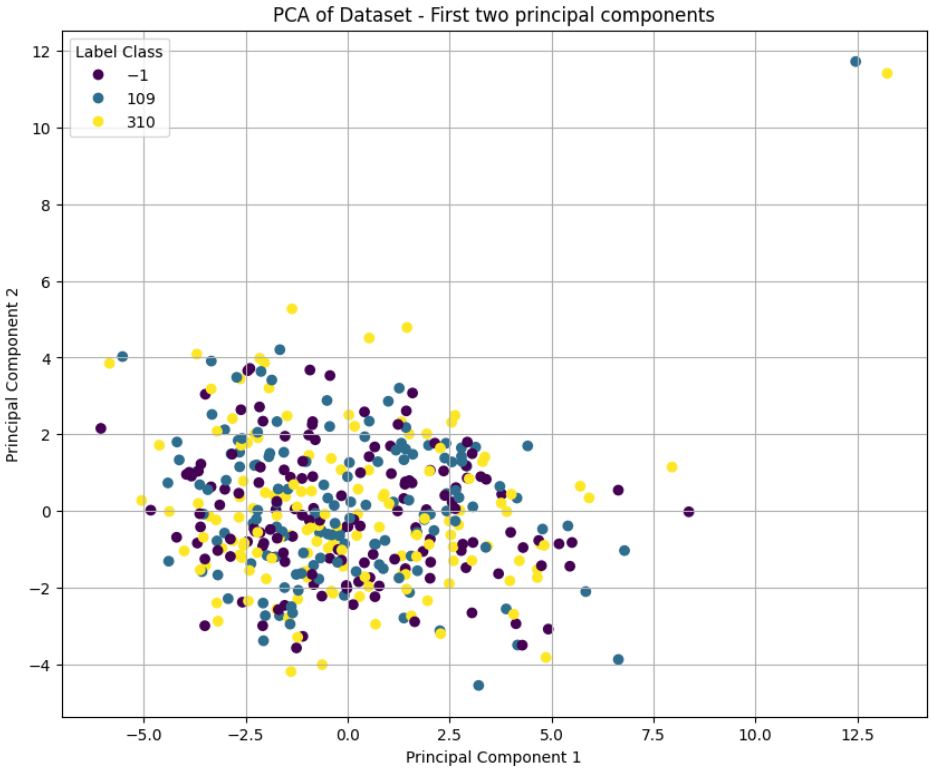
```
{'0': 'PC 1 (43.3%)',
 '1': 'PC 2 (24.3%)',
 '2': 'PC 3 (11.4%)',
 '3': 'PC 4 (6.8%)',
 '4': 'PC 5 (5.3%)'}
```

In the screenshot above we see the percentages of λ of each component.

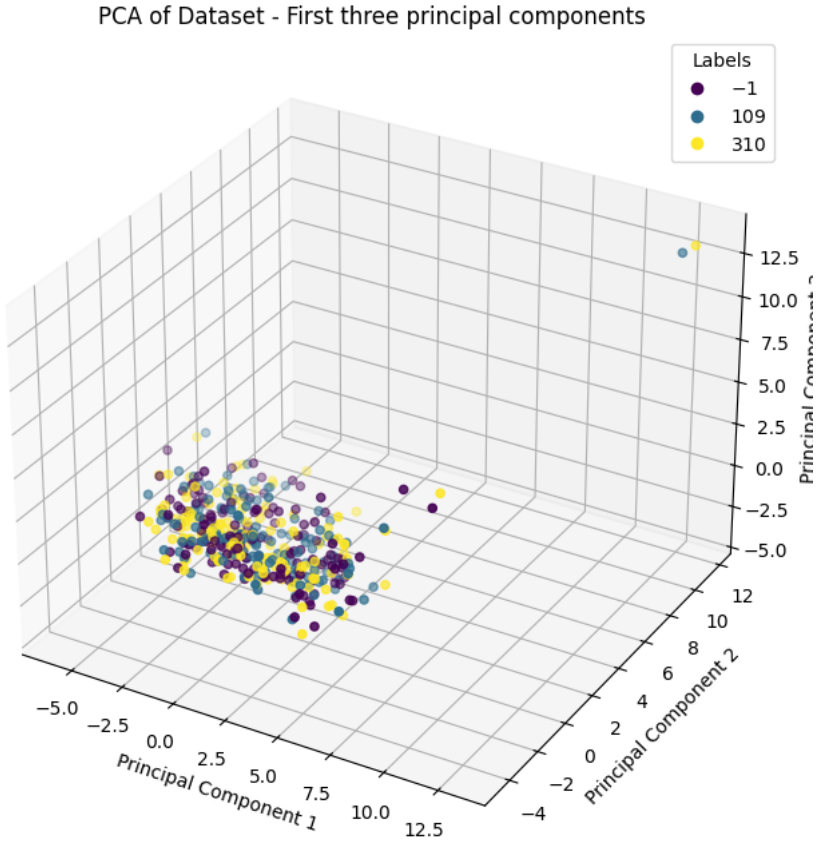
```
# Create a DataFrame with the principal components and the target variable
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5'])
pca_df['Label'] = y
```

```
plt.figure(figsize=(10, 8))
scatter_label = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['Label'], cmap='viridis')
plt.title('PCA of Dataset - First two principal components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(*scatter_label.legend_elements(), title="Label Class")
plt.grid(True)
plt.show()
```

A dataframe with the principal components, and the target variable - labels, is created. Now we plot the first two principal components with the highest explained variance ratio.



Below we see the same graph in 3D.



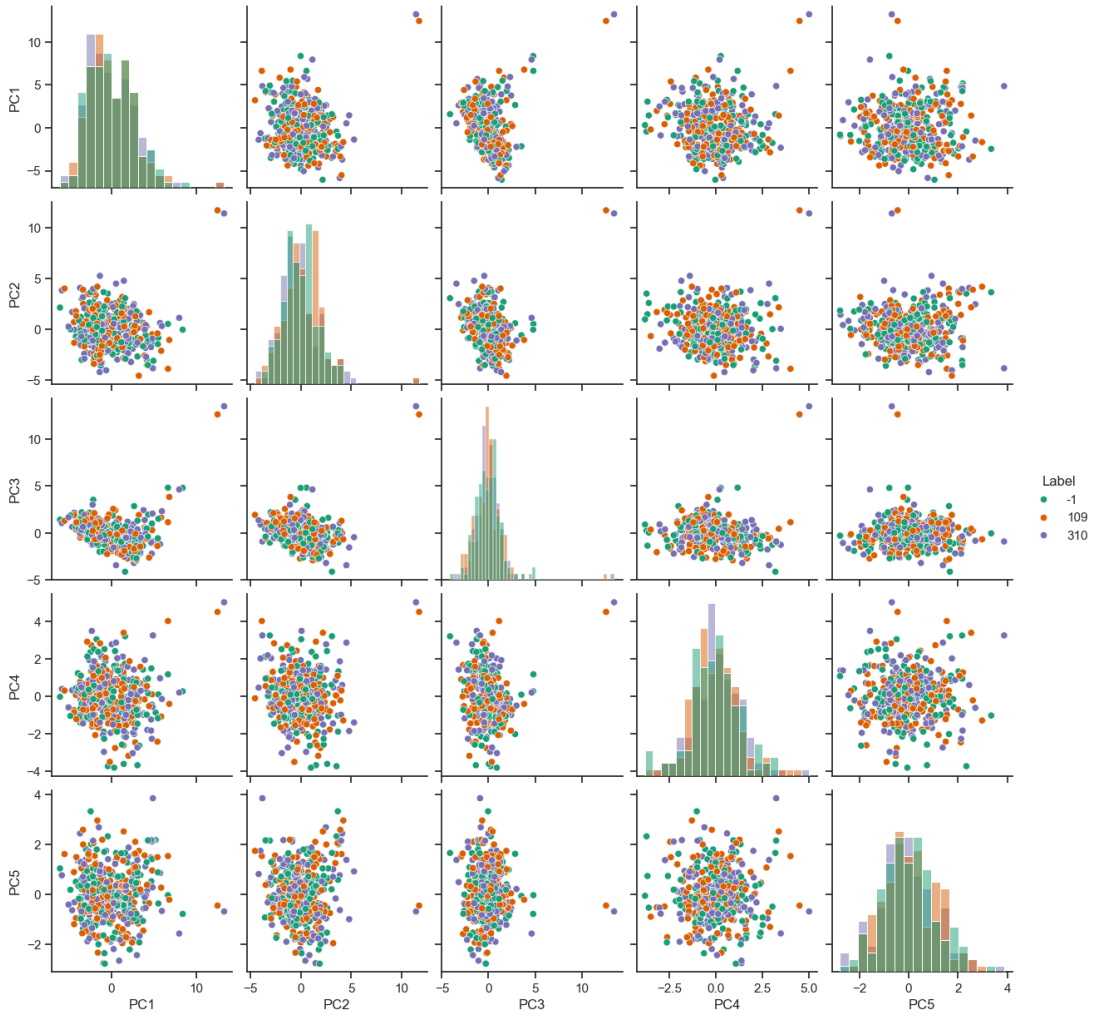
The scatter plot depicting the first two principal components in the PCA analysis reveals that the majority of data points from all three labels are concentrated within a confined region, with limited separation along both axes. This observation suggests challenges in discriminating between the labels in the reduced-dimensional space formed by these principal components. Several factors may contribute to this outcome. First, it is possible that the original features lack inherent discriminatory power among the labels, leading to a similar limitation in the principal components. Additionally, the first two principal components might not sufficiently capture the variability needed for effective label separation. Complex relationships between features and labels, not easily represented in a linear combination of features, could also contribute to this scenario. To address this, exploring scatter plots for other principal component pairs, evaluating classification techniques, considering feature engineering, and exploring nonlinear dimensionality reduction methods may offer insights and potential improvements for achieving clearer label separation.
Thus, we explore scatter plots of other principal component pairs.

```
# Create DataFrame with 5 PC components
df_pca = pd.DataFrame(PCA(n_components=5).fit_transform(X_scaled), columns=[f'PC{i+1}' for i in range(5)])
df_pca['Label'] = df['Label'].values

# Scatter Pairplot
sns.set_theme(style="ticks")
sns.pairplot(df_pca, hue="Label", diag_kind='hist', palette='Dark2')
plt.show()
```

✓ 1.7s

The exploration of scatter plots for various principal component pairs has revealed diverse patterns in the distribution of data points. Some pairs exhibit clear alignment along either the vertical or horizontal axis, suggesting strong linear relationships between certain features. This alignment may indicate potential redundancies or collinearities among those specific features, warranting further investigation. On the other hand, pairs with scattered distributions across the plot signify complex and non-linear relationships, which may pose challenges for linear techniques to effectively capture the underlying structure.
The observed variations in scatter plot patterns highlight the intricate nature of the dataset and emphasize the importance of selecting appropriate dimensionality reduction techniques. The presence of both linear and non-linear relationships underscores the need for a nuanced approach to data analysis. Considering domain-specific knowledge for feature engineering or

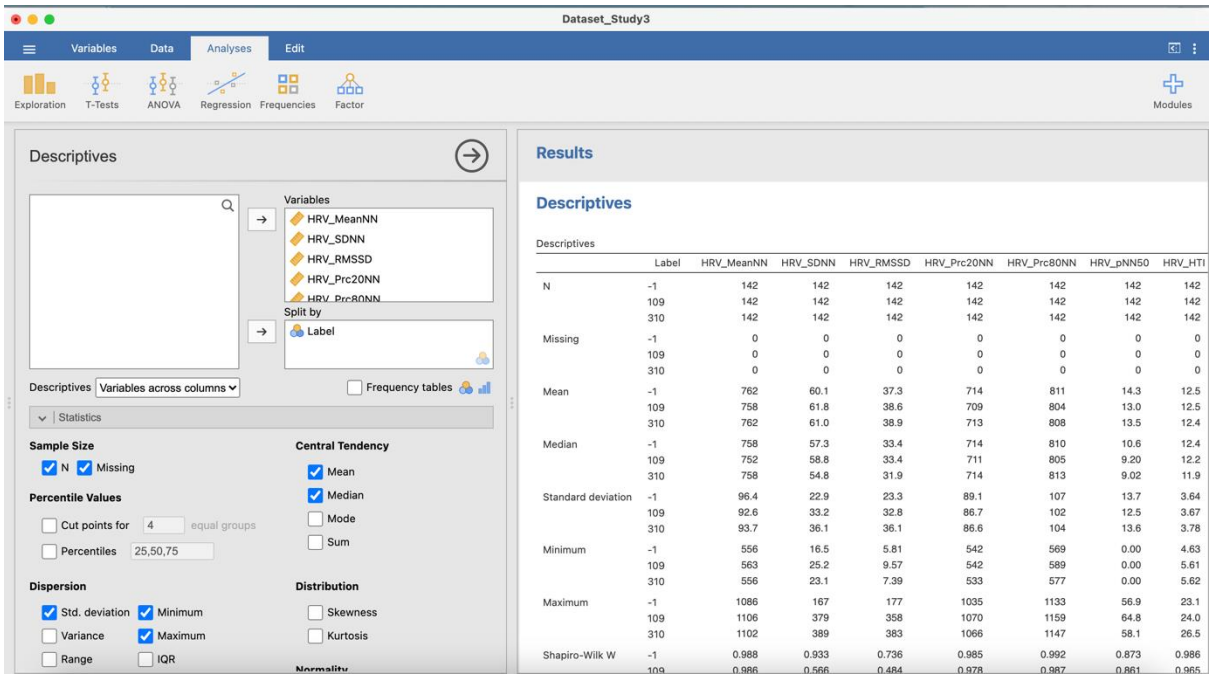
employing ensemble techniques may contribute to a more comprehensive understanding and improved representation of the dataset.



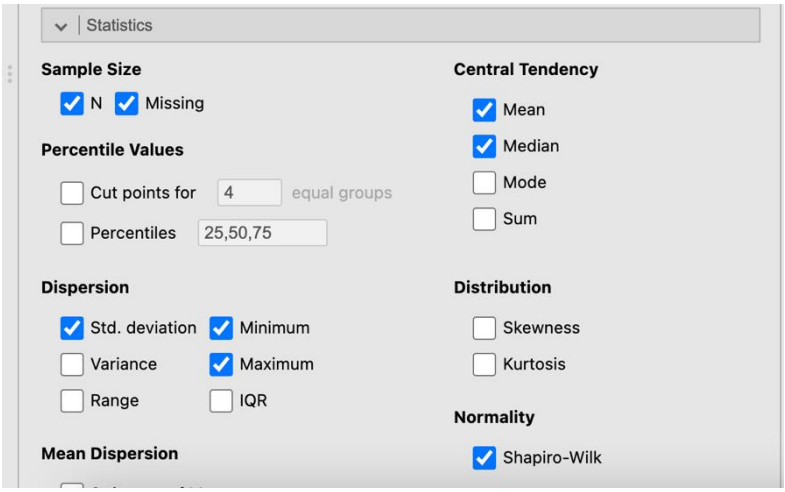
7. Feature selection

The next step in our analysis is the feature selection. Feature engineering is an important step in the analysis of our dataset. Feature selection is a process in machine learning and statistics where the most relevant and significant features (attributes or variables) are chosen from a larger set of features. The goal is to reduce the dimensionality of the dataset by selecting a subset of features that contribute the most to the predictive model's performance. The process involves evaluating and ranking features based on their importance or relevance to the task at hand. Feature selection aims to enhance model efficiency, reduce overfitting, improve interpretability, and potentially mitigate the impact of irrelevant or redundant features, especially in datasets with high dimensionality. There are various methods for feature

selection, including filter methods, wrapper methods, and embedded methods, each with its own advantages and use cases. Choosing between these methods of feature selection depends on the dataset. First, we need to check if the dataset is normally distributed or not. Previously, from the histograms, we saw that the dataset is not normally distributed. However, to be precise, we shall check it using Jamovi.



In the Jamovi application, the dataset was uploaded. Hence, in the analysis section, we explore the data. We choose all features as our variables, and we split the data by the labels.



In the statistics section, we choose the Shapiro-Wilk test for normality. The results are below.

Descriptives

	Label	HRV_MeanNN	HRV_SDNN	HRV_RMSSD	HRV_Prc20NN	HRV_Prc80NN	HRV_pNN50	HRV_HTI
N	-1	142	142	142	142	142	142	142
	109	142	142	142	142	142	142	142
	310	142	142	142	142	142	142	142
Missing	-1	0	0	0	0	0	0	0
	109	0	0	0	0	0	0	0
	310	0	0	0	0	0	0	0
Mean	-1	762	60.1	37.3	714	811	14.3	12.5
	109	758	61.8	38.6	709	804	13.0	12.5
	310	762	61.0	38.9	713	808	13.5	12.4
Median	-1	758	57.3	33.4	714	810	10.6	12.4
	109	752	58.8	33.4	711	805	9.20	12.2
	310	758	54.8	31.9	714	813	9.02	11.9
Standard deviation	-1	96.4	22.9	23.3	89.1	107	13.7	3.64
	109	92.6	33.2	32.8	86.7	102	12.5	3.67
	310	93.7	36.1	36.1	86.6	104	13.6	3.78
Minimum	-1	556	16.5	5.81	542	569	0.00	4.63
	109	563	25.2	9.57	542	589	0.00	5.61
	310	556	23.1	7.39	533	577	0.00	5.62
Maximum	-1	1086	167	177	1035	1133	56.9	23.1
	109	1106	379	358	1070	1159	64.8	24.0
	310	1102	389	383	1066	1147	58.1	26.5
Shapiro-Wilk W	-1	0.988	0.933	0.736	0.985	0.992	0.873	0.986
	109	0.986	0.566	0.484	0.978	0.987	0.861	0.965
	310	0.988	0.599	0.490	0.981	0.991	0.856	0.967
Shapiro-Wilk p	-1	0.284	<.001	<.001	0.118	0.625	<.001	0.148
	109	0.163	<.001	<.001	0.019	0.216	<.001	<.001
	310	0.261	<.001	<.001	0.042	0.515	<.001	0.002

HRV_HTI	HRV_VLF	HRV_LF	HRV_HF	HRV_TP	HRV_LFHF	HRV_SD1	HRV_SD2	HRV_SD1SD2	HRV_DFA_alpha1
142	142	142	142	142	142	142	142	142	142
142	142	142	142	142	142	142	142	142	142
142	142	142	142	142	142	142	142	142	142
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
12.5	108	1351	862	2371	3.10	26.4	80.5	0.321	1.32
12.5	363	2507	842	3747	3.52	27.4	82.7	0.322	1.34
12.4	258	1988	953	3251	3.14	27.6	81.3	0.328	1.32
12.4	80.2	937	446	1559	2.15	23.7	76.8	0.299	1.35
12.2	74.7	1035	398	1649	2.64	23.7	79.0	0.299	1.35
11.9	77.5	866	377	1453	2.08	22.6	74.2	0.308	1.32
3.64	101	1198	1648	2752	2.78	16.5	29.1	0.103	0.238
3.67	2801	14799	2820	20366	3.09	23.2	41.9	0.110	0.239
3.78	1676	9822	2981	14469	3.06	25.6	45.2	0.109	0.240
4.63	7.36	49.9	25.7	217	0.309	4.12	23.0	0.133	0.665
5.61	4.74	114	39.0	194	0.172	6.78	32.3	0.141	0.545
5.62	6.56	93.4	26.0	215	0.301	5.23	29.9	0.126	0.695
23.1	573	7245	14259	20820	17.2	126	201	0.727	1.85
24.0	33459	177202	32326	243950	18.6	254	474	0.795	1.81
26.5	20027	117579	31459	172140	19.1	271	480	0.625	1.85
0.986	0.816	0.794	0.416	0.599	0.780	0.736	0.964	0.930	0.982
0.965	0.0793	0.0886	0.187	0.0940	0.803	0.484	0.601	0.881	0.959
0.967	0.0903	0.107	0.237	0.125	0.734	0.490	0.638	0.965	0.991
0.148	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	0.065
<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001	<.001
0.002	<.001	<.001	<.001	<.001	<.001	<.001	<.001	0.001	0.523

HRV_SD1SD2	HRV_DFA_alpha1	HRV_DFA_alpha2	HRV_ApEn	HRV_SampEn
142	142	142	142	142
142	142	142	142	142
142	142	142	142	142
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0.321	1.32	0.878	0.940	1.32
0.322	1.34	0.892	0.941	1.30
0.328	1.32	0.897	0.939	1.32
0.299	1.35	0.872	0.945	1.31
0.299	1.35	0.869	0.951	1.31
0.308	1.32	0.896	0.944	1.34
0.103	0.238	0.206	0.0840	0.280
0.110	0.239	0.206	0.101	0.281
0.109	0.240	0.222	0.106	0.295
0.133	0.665	0.323	0.673	0.678
0.141	0.545	0.481	0.276	0.191
0.126	0.695	0.289	0.292	0.153
0.727	1.85	1.43	1.13	1.98
0.795	1.81	1.45	1.15	1.97
0.625	1.85	1.54	1.12	1.95
0.930	0.982	0.988	0.985	0.991
0.881	0.959	0.984	0.841	0.985
0.965	0.991	0.997	0.870	0.982
<.001	0.065	0.232	0.122	0.538
<.001	<.001	0.094	<.001	0.120
0.001	0.523	0.990	<.001	0.052

The screenshots above display the p values of the Shapiro-Wilk test. Almost all of the values are the p-values obtained from the test for many features are less than 0.001, suggesting that those features are not normally distributed.

Since the dataset features are not normally distributed, several suitable were used, including decision tree-based, sequential feature selection, and wrapper methods.

7.1. Decision tree-based methods

Decision tree-based methods are a powerful approach for feature selection, especially when dealing with datasets like ours, which involve psychophysiological responses to emotions and likely exhibit complex and non-linear relationships. Decision trees recursively partition the feature space based on the most informative features, making them robust to the distribution of data and capable of capturing intricate patterns. In the context of our dataset from the POPANE study, where emotions are associated with a variety of physiological responses, decision trees can effectively identify relevant features that contribute to predicting daily actions. The non-parametric nature of decision trees allows them to handle diverse and non-normally distributed features, making them well-suited for uncovering the nuanced relationships between cognitive and physiological variables. Additionally, decision tree-based

methods provide feature importance scores, offering valuable insights into which features contribute significantly to the predictive task. The interpretability of decision trees makes them particularly advantageous for understanding the complex interplay between psychophysiological responses and emotional states in our dataset.

7.1.1. Random Forest Classifier

The Random Forest classifier is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. It works by creating a "forest" of decision trees, each trained on a different subset of the data and incorporating random feature subsets. This randomness and diversity among the trees enhance the model's robustness and reduce overfitting. During classification, each tree in the forest independently makes a prediction, and the final output is determined by a majority vote. This approach improves the model's accuracy, generalization, and resilience to outliers, making Random Forest a powerful and versatile tool for various machine-learning tasks. In the context of our dataset from the POPANE study, where psychophysiological responses are associated with emotions, a Random Forest classifier could effectively capture complex relationships and enhance predictive performance.

```python
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X, y)
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]
print("Feature ranking:")

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

✓ 0.1s

```
Feature ranking:
1. Feature HRV_DFA_alpha2 (0.065538)
2. Feature HRV_VLF (0.062682)
3. Feature HRV_DFA_alpha1 (0.062374)
4. Feature HRV_LFHF (0.060656)
5. Feature HRV_SD1SD2 (0.060456)
6. Feature HRV_SampEn (0.057640)
7. Feature HRV_HTI (0.055609)
8. Feature HRV_ApEn (0.053515)
9. Feature HRV_SD2 (0.053233)
10. Feature HRV_LF (0.051174)
11. Feature HRV_HF (0.050231)
12. Feature HRV_Prc80NN (0.050185)
13. Feature HRV_TP (0.049360)
14. Feature HRV_SDNN (0.046905)
15. Feature HRV_Prc20NN (0.046831)
16. Feature HRV_pNN50 (0.045720)
17. Feature HRV_MeanNN (0.043072)
18. Feature HRV_SD1 (0.042942)
19. Feature HRV_RMSSD (0.041875)
```

The model ranks the features by importance and returns the ranked list with the percentages next to it. It is up to the data analyst to choose the most important features.

```
selected_features_RFC = X[['HRV_DFA_alpha1', 'HRV_VLF', 'HRV_SD1SD2', 'HRV_DFA_alpha2', 'HRV_ApEn']]
selected_features_RFC.head()
```

✓ 0.0s

	HRV_DFA_alpha1	HRV_VLF	HRV_SD1SD2	HRV_DFA_alpha2	HRV_ApEn
0	1.482091	135.219052	0.287440	0.754264	0.863522
1	1.443776	77.009053	0.334017	0.683996	0.920773
2	1.335662	129.751167	0.330813	0.940213	0.974523
3	1.542718	163.822696	0.296022	0.598340	0.831501
4	1.467014	132.967712	0.312278	0.611829	0.821712

These are the features that were chosen for the model.

7.1.2. Decision Tree Classifier

The Decision Tree classifier is a popular machine learning algorithm that makes decisions based on a tree-like model. It recursively splits the dataset into subsets by identifying the most informative features at each node, leading to a hierarchical structure resembling a tree. The decision-making process involves traversing the tree from the root to the leaves, with each internal node representing a decision based on a feature, and each leaf node representing the predicted class or outcome. Decision Trees are transparent, easy to understand, and interpret, making them valuable for gaining insights into the model's decision-making process. However, they are prone to overfitting and capturing noise in the data. Despite this limitation, Decision Trees are suitable for our dataset from the POPANE study, as they can reveal important features influencing psychophysiological responses to emotions, contributing to the interpretability of the results.

```
from sklearn.tree import DecisionTreeClassifier
# Decision Tree Classifier for feature importance
dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X, y)

# Extracting feature importances
dtree_importances = dtree.feature_importances_
indices = np.argsort(dtree_importances)[::-1]
for f in range(X.shape[1]):
    print("%d. Feature %s (%f)" % (f + 1, X.columns[indices[f]], dtree_importances[indices[f]]))
```

✓ 0.0s

```
1. Feature HRV_VLF (0.132946)
2. Feature HRV_DFA_alpha2 (0.093646)
3. Feature HRV_SampEn (0.077037)
4. Feature HRV_SD1SD2 (0.063798)
5. Feature HRV_SD2 (0.062771)
6. Feature HRV_ApEn (0.059822)
7. Feature HRV_HTI (0.059717)
8. Feature HRV_LFHF (0.057143)
9. Feature HRV_HF (0.049322)
10. Feature HRV_DFA_alpha1 (0.045242)
11. Feature HRV_SDNN (0.045189)
12. Feature HRV_Prc20NN (0.044009)
13. Feature HRV_Prc80NN (0.038152)
14. Feature HRV_LF (0.033611)
15. Feature HRV_RMSSD (0.029955)
16. Feature HRV_MeanNN (0.029577)
17. Feature HRV_TP (0.029542)
18. Feature HRV_pNN50 (0.029101)
19. Feature HRV_SD1 (0.019420)
```

The ranked list is more or less the same as the list obtained from the Random Forest Classifier.

```
1  selected_features_DTC = X[['HRV_DFA_alpha2', 'HRV_LF', 'HRV_HTI', 'HRV_LFHF', 'HRV_Prc20NN']]
2  selected_features_DTC.head()
```

✓ 0.0s

	HRV_DFA_alpha2	HRV_LF	HRV_HTI	HRV_LFHF	HRV_Prc20NN
0	0.754264	1984.443232	15.285714	2.107574	773.2
1	0.683996	1012.790954	15.307692	1.422049	838.6
2	0.940213	1924.529470	15.500000	2.525982	766.4
3	0.598340	954.991547	16.714286	5.757132	699.0
4	0.611829	2348.301339	13.937500	4.139157	731.0

These are the features that were chosen for the model.

7.2. Sequential feature selection

Sequential Feature Selection (SFS) is a method that iteratively selects or eliminates features to improve the performance of a machine learning model. In the context of our dataset from the POPANE study, SFS can be a valuable tool. It operates by starting with an empty set of

features and gradually adding or removing features based on a predefined criterion, such as maximizing model accuracy or minimizing error.

SFS proceeds through different subsets of features, evaluating their impact on the model's performance at each step. The process can follow either a forward or backward direction. In the forward direction, SFS starts with an empty set and incrementally adds features, while in the backward direction, it begins with all features and removes them one by one.

For our study, SFS could help identify a subset of features that significantly contribute to predicting psychophysiological responses to emotions. This can enhance the interpretability of the model and potentially improve its performance. It's essential to consider the specific objectives of our analysis and the nature of our dataset to determine the most suitable SFS strategy, whether forward or backward, and the performance metric for feature evaluation.

```
1  from sklearn.feature_selection import SequentialFeatureSelector as SFS
✓  0.0s
```

From the sklearn.feature_selection library we import the sequential feature selection module.

7.2.1. Forward selection

```
1  # Forward Selection using SFS from sklearn
2  sfs_forward = SFS(estimator=model,
3                    n_features_to_select=5,
4                    direction='forward',
5                    scoring='accuracy',
6                    cv=5)
7  sfs_forward = sfs_forward.fit(X, y)
✓  24.4s
```

The sequential forward selector module is built and applied to our dataset.

```
1  X.columns[sfs_forward.get_support()==True]
✓  0.0s
Index(['HRV_MeanNN', 'HRV_Prc20NN', 'HRV_Prc80NN', 'HRV_SD1SD2', 'HRV_ApEn'], dtype='object')
```

These are the selected features, which are then saved to the selected_features_SFSF variable.

```
selected_features_SFSF = X[['HRV_MeanNN', 'HRV_pNN50', 'HRV_LFHF', 'HRV_SD1', 'HRV_ApEn']]
selected_features_SFSF.head()
```

✓ 0.0s

	HRV_MeanNN	HRV_pNN50	HRV_LFHF	HRV_SD1	HRV_ApEn
0	838.708920	21.962617	2.107574	30.452465	0.863522
1	899.929293	24.623116	1.422049	36.089337	0.920773
2	827.319444	20.276498	2.525982	28.668833	0.974523
3	768.630901	23.076923	5.757132	30.832839	0.831501
4	807.432432	18.834081	4.139157	34.955488	0.821712

Above we see the chosen five features and the corresponding data to it.

7.2.2. Backwards selection

```
# Backward Elimination using SFS from sklearn
sfs_backward = SFS(estimator=model,
                   n_features_to_select=5,
                   direction='backward',
                   scoring='accuracy',
                   cv=5)
sfs_backward = sfs_backward.fit(X, y)
```

✓ 1m 7.2s

```
X.columns[sfs_backward.get_support()==True]
```

✓ 0.0s

```
Index(['HRV_Prc80NN', 'HRV_VLF', 'HRV_LFHF', 'HRV_SD1SD2', 'HRV_DFA_alpha2'], dtype='object')
```

```
selected_features_SFSB = X[['HRV_MeanNN', 'HRV_RMSSD', 'HRV_Prc80NN', 'HRV_HF', 'HRV_TP']]
selected_features_SFSB.head()
```

✓ 0.0s

	HRV_MeanNN	HRV_RMSSD	HRV_Prc80NN	HRV_HF	HRV_TP
0	838.708920	42.966689	907.4	941.576892	3083.350848
1	899.929293	50.910712	960.4	712.205460	1835.167749
2	827.319444	40.450375	881.8	761.893623	2848.723565
3	768.630901	43.512943	837.4	806.486463	1533.560411
4	807.432432	49.326201	879.2	567.338064	3066.742236

Respectively, these are the chosen features of the sequential backward selection model.

7.3. Wrapper method

Recursive Feature Elimination (RFE) is a powerful feature selection technique employed in machine learning to identify the most relevant variables for predictive modeling. In the context of the POPANE dataset derived from Study 3, RFE becomes particularly valuable due to its ability to navigate through a plethora of features related to psychophysiological responses to emotions. By iteratively training a model, assessing feature importance, and eliminating the least significant features, RFE hones in on a subset of variables that contribute significantly to the predictive capacity of the model. This method not only aids in mitigating the curse of dimensionality but also enhances model interpretability by highlighting the most influential features. Applied judiciously with an appropriate classification algorithm, RFE becomes an instrumental tool for distilling complex datasets into concise and impactful sets of features, aligning with the overarching objective of predicting daily actions based on cognitive and physiological responses to emotions in the POPANE study.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs',max_iter=2000)

# Initializing RFE model
rfe = RFE(estimator = model,n_features_to_select= 6)  # selecting 6 features
fit = rfe.fit(X_scaled, y)

# summarize the selection of the attributes
print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
```

✓ 0.0s

```
Num Features: 6
Selected Features: [ True False False  True  True False False False  True False False False
 False False False False False  True  True]
Feature Ranking: [ 1  7  6  1  1  2  4 13  1  9 14 12  8  5  3 10 11  1  1]
```

```
X.columns[fit.get_support()==True]
```

✓ 0.0s

```
Index(['HRV_MeanNN', 'HRV_Prc20NN', 'HRV_Prc80NN', 'HRV_LF', 'HRV_ApEn',
       'HRV_SampEn'],
      dtype='object')
```

```
X_rfe = X[X.columns[fit.get_support()==True]]
X_rfe.head()
```

✓ 0.0s

	HRV_MeanNN	HRV_Prc20NN	HRV_Prc80NN	HRV_LF	HRV_ApEn	HRV_SampEn
0	838.708920	773.2	907.4	1984.443232	0.863522	1.311373
1	899.929293	838.6	960.4	1012.790954	0.920773	1.357924
2	827.319444	766.4	881.8	1924.529470	0.974523	1.587459
3	768.630901	699.0	837.4	954.991547	0.831501	1.081248
4	807.432432	731.0	879.2	2348.301339	0.821712	1.107051

From this model, we have obtained important features different from the previous ones. The model evaluations of each feature selection method will be presented below.

8. Model development – Supervised learning

8.1. Model development

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Assuming X is your feature set, y is your target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=45)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

✓ 0.0s

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Function to build model and evaluate
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)

    accuracy = accuracy_score(y_test, y_pred)

    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')

    return accuracy, precision, recall
```

✓ 0.0s

To build a model we split the data into training and testing data sets. Before training the model we first standardize the data. Hence, we use the logistic regression model, returning the results of accuracy score, precision score, and recall score.

8.2. Model evaluation

```
1  evaluate_model(X_pca,y,0.3)
✓ 0.0s
(0.2734375, 0.2646796994623082, 0.2955026455026455)
```

```
1  evaluate_model(selected_features_RFC,y,0.3)
✓ 0.0s
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/skle
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

(0.2890625, 0.30854017435726583, 0.32162698412698415)
```

```
1  evaluate_model(selected_features_DTC,y,0.3)
✓ 0.0s
(0.2890625, 0.2933057280883367, 0.3103174603174603)
```

```
1  evaluate_model(selected_features_SFSF,y,0.3)
✓ 0.0s
(0.2890625, 0.21347248576850095, 0.32486772486772486)
```

```
1  evaluate_model(selected_features_SFSB,y,0.3)
✓ 0.0s
(0.28125, 0.2984437901820585, 0.3146825396825397)
```

```
1  evaluate_model(X_rfe,y,0.3)
✓  0.0s
(0.3203125, 0.339810982668125554, 0.34160052910052907)
```

Overall, all models show approximately the same accuracy, precision, and recall scores. The recursive feature elimination method is found to be the most accurate.

9. Conclusion

In conclusion, despite the meticulous exploration and application of various data analysis techniques on the POPANE dataset from Study 3, the achieved model accuracy plateaued at a maximum of 32%. This modest accuracy could stem from several factors within the dataset and the modeling approach. One plausible reason is the inherent complexity of psychophysiological responses to emotions, which might require more sophisticated modeling techniques or inclusion of additional contextual features for improved predictions. Moreover, the dataset's limited size or the presence of subtle patterns and outliers could contribute to the challenge of accurately capturing and generalizing the underlying relationships. It is imperative to critically evaluate and potentially expand the dataset, experiment with alternative algorithms, and explore feature engineering strategies to enhance the model's predictive performance in future analyses.