

Department of Mathematics and Computer Science

Faculty of Data Analysis

UNIME

Data Mining and Analytics Project Report

POPANE Dataset Analysis

Student: Abdez Aibek (537907)

Student: Zhengis Sanat (537902)

Professor: Gennaro Tartarisco

Academic Year 2023/2024

**Table of Contents**

[1. Introduction 2](#_Toc177527727)

[1.1. Dataset description 2](#_Toc177527728)

[1.2. Objectives 3](#_Toc177527729)

[2. Data exploration and EDA 4](#_Toc177527730)

[3. Data visualization 10](#_Toc177527731)

[4. Data preprocessing 13](#_Toc177527732)

[5. Data correlation 15](#_Toc177527733)

[6. Principal Component Analysis 19](#_Toc177527734)

[7. Feature selection 23](#_Toc177527735)

[8. Model Development and Evaluation 26](#_Toc177527736)

[9. Conclusion 29](#_Toc177527737)

# Introduction

## Dataset description

The "Dataset\_Study3" comes from the publicly available POPANE dataset, which examines psychophysiological responses to positive and negative emotions in 1157 healthy young adults. This dataset spans seven different studies, with a specific focus on "Study 3," which investigates the cardiovascular responses of 147 participants. The goal of this study is to identify patterns that can predict daily behaviors by analyzing how cognitive processes—such as evaluations, motivations, and intentions—are linked to physiological responses during a controlled social task. The analysis centers on how these motivations, particularly in gratitude-related interventions, influence physiological reactions.

The study procedure is outlined below:

A screenshot of a phone

Description automatically generated

Participants first completed a 3-minute baseline measurement. They were then asked to send two text messages: one expressing gratitude and one neutral. The order of the messages was randomized using an algorithm to ensure balance. After each message, participants had a 3-minute recovery period to allow their physiological state to return to baseline.

The dataset contains 426 observations (142 participants, each in three conditions: baseline, gratitude, and neutral). Five participants were excluded due to missing ECG data. The data is labeled to indicate the condition: baseline (-1), gratitude (310), and neutral (109). The features in the dataset are extracted from the RR interval, which measures the time between consecutive R waves in an electrocardiogram's QRS complex. These features include time-domain, frequency-domain, and non-linear measurements.

## Objectives

The aim of this study is to better understand the connection between psychophysiological reactions and cognitive processes, with a particular focus on how motivations, especially those related to expressing gratitude, play a role. The dataset, which includes diverse participants and scenarios, provides a valuable opportunity to examine how emotional responses influence behavior. Through the use of advanced analytical methods, the study seeks to discover patterns that can help forecast everyday actions by examining the relationship between cognitive and physiological elements.

To analyze the dataset, a variety of data processing techniques will be employed, including data importation, cleaning, exploration, visualization, and normalization. Furthermore, techniques like feature selection, principal component analysis (PCA), clustering, and classification will be used to create a predictive model for emotional states. The ultimate aim is to find the most effective strategy for building an accurate and reliable model.

# Data exploration and EDA

In this project, the libraries serve various essential tasks necessary for data analysis and model building:

1. **pandas (pd)** – used for data manipulation, loading the dataset, cleaning, and performing preliminary data analysis.
2. **numpy (np)** – required for numerical operations and handling arrays, making it easier to perform mathematical computations.
3. **matplotlib.pyplot (plt)** – used for creating plots, such as histograms and line graphs, which help visualize data distributions.
4. **seaborn (sns)** – utilized for statistical data visualization, such as heatmaps or scatter plots, providing more advanced visualization tools for analyzing correlations.
5. **sklearn.decomposition.PCA** – performs Principal Component Analysis (PCA) to reduce the dimensionality of the dataset while retaining the most important information.
6. **sklearn.preprocessing.StandardScaler** – standardizes the data by scaling features to a common scale, which is important when applying machine learning models.
7. **mpl\_toolkits.mplot3d.Axes3D** – helps create 3D plots, which are useful for visualizing multidimensional data, such as PCA results.
8. **sklearn.ensemble.RandomForestClassifier** and **sklearn.tree.DecisionTreeClassifier** – these are used to build classification models with random forests and decision trees, allowing for feature importance analysis and data classification.
9. **sklearn.feature\_selection.SequentialFeatureSelector (SFS)** – used for stepwise feature selection to choose the most significant variables for model building.
10. **sklearn.feature\_selection.RFE** – applies Recursive Feature Elimination (RFE) to iteratively remove less important features, improving the model's accuracy.
11. **sklearn.linear\_model.LogisticRegression** – used to build logistic regression models for predicting categorical variables.
12. **sklearn.model\_selection.train\_test\_split** – splits the dataset into training and testing sets, which is crucial for evaluating the model's performance.
13. **sklearn.metrics** – provides metrics (accuracy, precision, recall, etc.) to assess the quality and effectiveness of the model.

A screen shot of a computer

Description automatically generated

In this project, the use of various libraries is crucial to efficiently handle the data analysis process and build accurate models. Libraries like **pandas** and **numpy** are essential for data manipulation and performing numerical computations, which are the foundation of any data-driven analysis. Visualization libraries such as **matplotlib** and **seaborn** help us better understand the data through graphical representations, making patterns and insights more accessible.

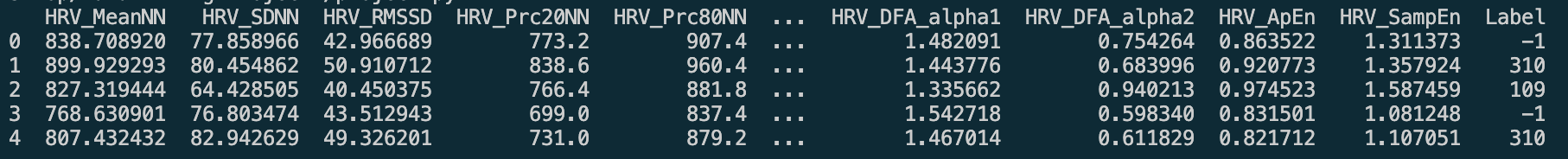
Furthermore, machine learning libraries from **scikit-learn** allow us to perform more advanced tasks, including feature selection, dimensionality reduction, and model building. Tools like **PCA** and **Sequential Feature Selection** are necessary for optimizing the dataset, while algorithms like **RandomForestClassifier** and **LogisticRegression** enable the creation of predictive models. These libraries streamline complex operations and provide standardized methods for evaluating the performance of the models, ensuring the analysis is both efficient and accurate.

This code is responsible for **loading the dataset** and **providing an initial overview** of its structure. It reads the data from a CSV file, displays a few sample rows, and gives information about the dataset's size and basic statistical properties. This step is crucial for understanding the dataset before any further analysis or preprocessing.

A computer screen shot of text

Description automatically generated

**print(df.head())** – This command outputs the first five rows of the dataset, allowing us to quickly inspect the structure of the data. It helps verify that the dataset was loaded correctly and gives a preview of the columns and data types.



**print(df.shape)** – This line prints the dimensions of the dataset in the form (rows, columns), providing an overview of how many records (rows) and features (columns) are in the dataset.

A number on a blue background

Description automatically generated

**print(df.describe())** – This generates summary statistics for the numerical columns in the dataset, such as mean, standard deviation, min, max, and quartiles. This information helps understand the distribution and range of the data, which is critical for identifying potential issues like outliers or skewed distributions.

A screen shot of a computer screen

Description automatically generated

The next stage of **Data Exploration and Exploratory Data Analysis** (EDA) is critical because it helps us understand the quality and structure of the dataset before diving into deeper analysis or building predictive models. Without this step, we run the risk of using incomplete, inconsistent, or erroneous data, which can severely impact the accuracy and reliability of any machine learning models we develop. By exploring the dataset, we can uncover hidden patterns, spot potential issues such as missing or duplicate data, and ensure that all data types are appropriate for the analysis methods we plan to use. This preliminary step sets the foundation for effective data processing and ensures that we have clean, well-structured data to work with throughout the project.

**print(df.dtypes)**   
This line prints the data types of each column in the DataFrame df. It helps to understand what kind of data is present (e.g., integers, floats, strings) and whether any columns need to be converted to a different type for further analysis.

A screenshot of a computer screen

Description automatically generated

**print(df.isnull().sum())**

This checks for missing values in the dataset. For each column, it calculates the total number of missing or NaN values and prints the result. This helps in identifying which columns may need imputation or removal before proceeding with analysis.

A screen shot of a computer

Description automatically generated

**df.info()**

This provides a detailed summary of the DataFrame, including the total number of rows, columns, data types, and non-null counts for each column. It gives an overview of the structure and completeness of the dataset.

A screenshot of a computer screen

Description automatically generated

**print(df.duplicated().sum())**

This checks for duplicate rows in the dataset. It counts how many rows have duplicate entries. Duplicate rows may need to be removed to avoid skewing the results of the analysis. It outputs 0, which means that there aro no duplicated values.

This code is a crucial step in **Exploratory Data Analysis (EDA)**. It allows us to:

* **Understand the structure of the dataset** by checking data types and basic metadata.
* **Identify potential issues** such as missing values or duplicate rows that could affect the quality of the analysis.
* **Prepare the dataset** for further steps, such as cleaning, preprocessing, and feature engineering.

By addressing these potential issues early, the dataset can be transformed into a cleaner and more reliable form, improving the accuracy and performance of the models we will build later in the project.

# Data visualization

Data visualization is a crucial step in data analysis that helps us **better understand the structure and distribution of the dataset.** By creating various plots, such as histograms, scatter plots, and boxplots, we gain valuable insights that can guide further data cleaning, feature selection, and model development.

**df.boxplot(figsize=(20, 10))**:

A blue background with white text

Description automatically generated

**df.boxplot()**: This method creates a **boxplot** for each numerical column in the DataFrame df.

A **boxplot** (also known as a box-and-whisker plot) is a statistical chart that displays the distribution of data based on a five-number summary:

* **Minimum**: The lowest data point excluding outliers.
* **Q1 (First quartile)**: The 25th percentile, which is the median of the lower half of the data.
* **Median (Q2)**: The 50th percentile, or the middle value of the dataset.
* **Q3 (Third quartile)**: The 75th percentile, or the median of the upper half of the data.
* **Maximum**: The highest data point excluding outliers.

The **whiskers** extend to the lowest and highest data points within 1.5 \* IQR (Interquartile Range), while outliers are represented by individual points outside this range.

The **figsize=(20, 10)** argument specifies the size of the plot (width = 20, height = 10), ensuring the boxplots are large and readable.

This code provides a visual representation of the distribution of each numerical column in the dataset. By looking at the boxplots, you can:

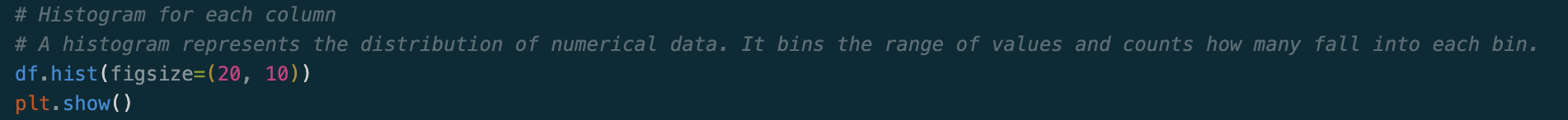
Identify outliers: Boxplots show points that fall outside the whiskers as potential outliers.

Understand data spread: It shows how the data is distributed across each feature, making it easier to identify skewed distributions, concentration around the median, and data variability.

Compare columns: You can compare the distributions of different features within the same DataFrame, helping in further analysis or feature engineering.

This is an essential step in exploratory data analysis (EDA) to understand the dataset's structure before applying machine learning models.

The histogram shows the **distribution of numerical values** by dividing the data into "bins" and counting how many values fall into each bin.



**df.hist(figsize=(20, 10))**:

**df.hist()**: This function generates a **histogram** for each numerical column in the DataFrame df. It groups the continuous values into "bins" (intervals) and counts how many values fall into each bin. This provides an understanding of how the data is distributed across the range of values.

The **histogram** helps visualize key aspects such as:

* **Skewness**: Whether the data is symmetric or skewed to the left or right.
* **Kurtosis**: The shape of the tails and how peaked the distribution is.
* **Outliers**: Unusual values far from the rest of the data.
* **Mode(s)**: The most frequent values in the dataset.

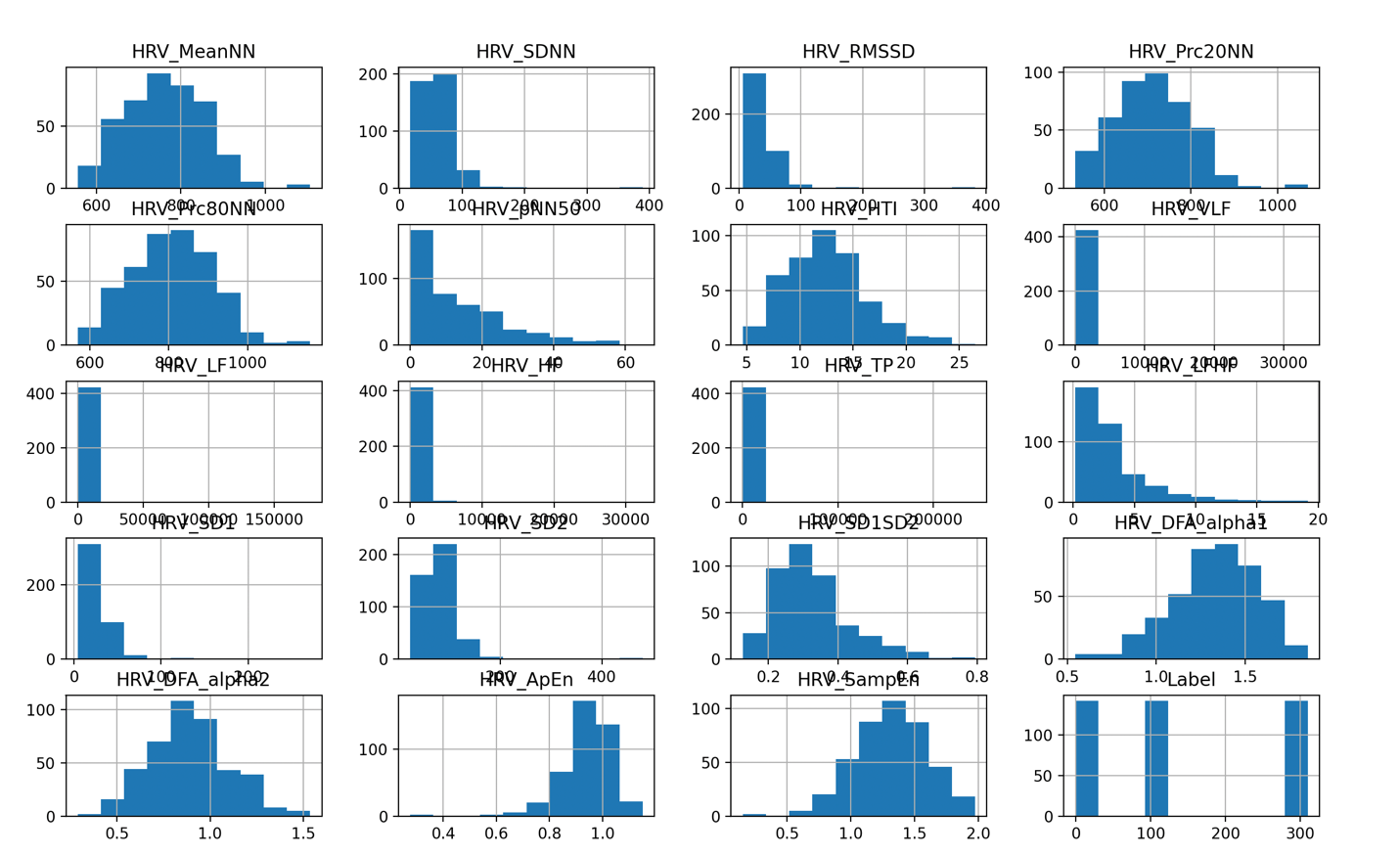
The **figsize=(20, 10)** argument defines the size of the overall figure (width = 20, height = 10), ensuring that the histograms are large enough for clear visualization.

**plt.show()**:

This function from matplotlib.pyplot is used to **display the histograms** on the screen. It ensures that the generated plots are rendered.

A graph with numbers and lines

Description automatically generated



# Data preprocessing

This step is essential for **detecting outliers** in the dataset. Outliers are data points that are significantly different from most observations and can skew statistical analyses and negatively impact the performance of machine learning models. This code uses the **Interquartile Range (IQR)** method to systematically identify outliers in any numerical column of the dataset. The IQR method is a common technique for detecting outliers because it focuses on the central distribution of the data and filters out extreme values.

**Breakdown of Each Step:**

1. **First Quartile (Q1)**: The 25th percentile, below which 25% of the data points fall.
2. **Third Quartile (Q3)**: The 75th percentile, below which 75% of the data points fall.
3. **Interquartile Range (IQR)**: The range between Q1 and Q3, which captures the middle 50% of the data. It's a robust measure of statistical dispersion.
4. **Lower Bound**: Any data points below this value are considered outliers.
5. **Upper Bound**: Any data points above this value are considered outliers.
6. A computer screen shot of text

   Description automatically generated**Outliers**: The code filters and returns rows where the values are outside the bounds, indicating potential outliers.

**def find\_outliers(df, column):**

* This function identifies outliers in a specific column of a DataFrame.

**q1 = df[column].quantile(0.25)**

* Finds the **first quartile (Q1)**, the value below which 25% of data falls.

**q3 = df[column].quantile(0.75)**

* Finds the **third quartile (Q3)**, the value below which 75% of data falls.

**iqr = q3 - q1**

* Calculates the **interquartile range (IQR)**, which is the range between Q1 and Q3 (middle 50% of data).

**lower\_bound = q1 - (1.5 \* iqr)**

* Defines the **lower limit** for outliers.

**upper\_bound = q3 + (1.5 \* iqr)**

* Defines the **upper limit** for outliers.

**outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]**

* Filters and finds rows where values fall outside the lower or upper bounds (outliers).

**return outliers**

* Returns a DataFrame containing only the outliers.

A graph with lines and dots

Description automatically generated

This step of **detecting and addressing outliers** is a vital part of the **data preprocessing** phase. Ensuring that outliers are appropriately handled helps to create a cleaner, more reliable dataset, which leads to better-performing models and more accurate insights during analysis.

# Data correlation

This step is important because it helps understand the relationships between features and how the data behaves across different conditions. The correlation matrix and heatmap show how features relate to each other, revealing any redundant or independent features. Visual tools like histograms, boxplots, and violin plots allow us to see how key metrics, such as **HRV\_TP**, vary across conditions like baseline, gratitude, and natural. Pairplots help identify patterns or outliers between features, giving a better sense of the data. Overall, this analysis guides the next steps by providing insights that help with feature selection and modeling decisions, ensuring more accurate and informed results.

A screenshot of a computer program

Description automatically generated

**Heatmap**: A heatmap visually represents the correlation between features in the dataset. It helps identify which features are strongly related or independent. Strong correlations may indicate redundant features, while weak correlations suggest feature independence.

**Pairplot**: Pairplots show the relationships between pairs of features through scatter plots, while histograms on the diagonal display individual feature distributions. It helps detect linear relationships, clusters, and patterns between different features.

**Histograms**: Histograms show the distribution of data for a specific feature by binning values into ranges. They help in understanding the shape, spread, and skewness of data, allowing detection of outliers or unusual distributions.

**Boxplot**: Boxplots summarize data using the minimum, first quartile (Q1), median, third quartile (Q3), and maximum values, highlighting the central tendency and spread. They also reveal outliers, making it easy to spot deviations from normal ranges.

**Violin Plot**: Violin plots combine features of a boxplot and a density plot. They show the distribution, density, and probability of data across different categories while also providing summary statistics. This makes it useful for comparing the full distribution of a feature across groups or conditions.

Each plot type provides different insights into the data, helping to identify relationships, distributions, outliers, and patterns. Together, they give a comprehensive view of the

A screenshot of a computer screen

Description automatically generateddataset.

A graph of a graph of a person

Description automatically generated with medium confidence

A diagram of a graph

Description automatically generated with medium confidence

A blue and black leaf shapes

Description automatically generated with medium confidence

# Principal Component Analysis

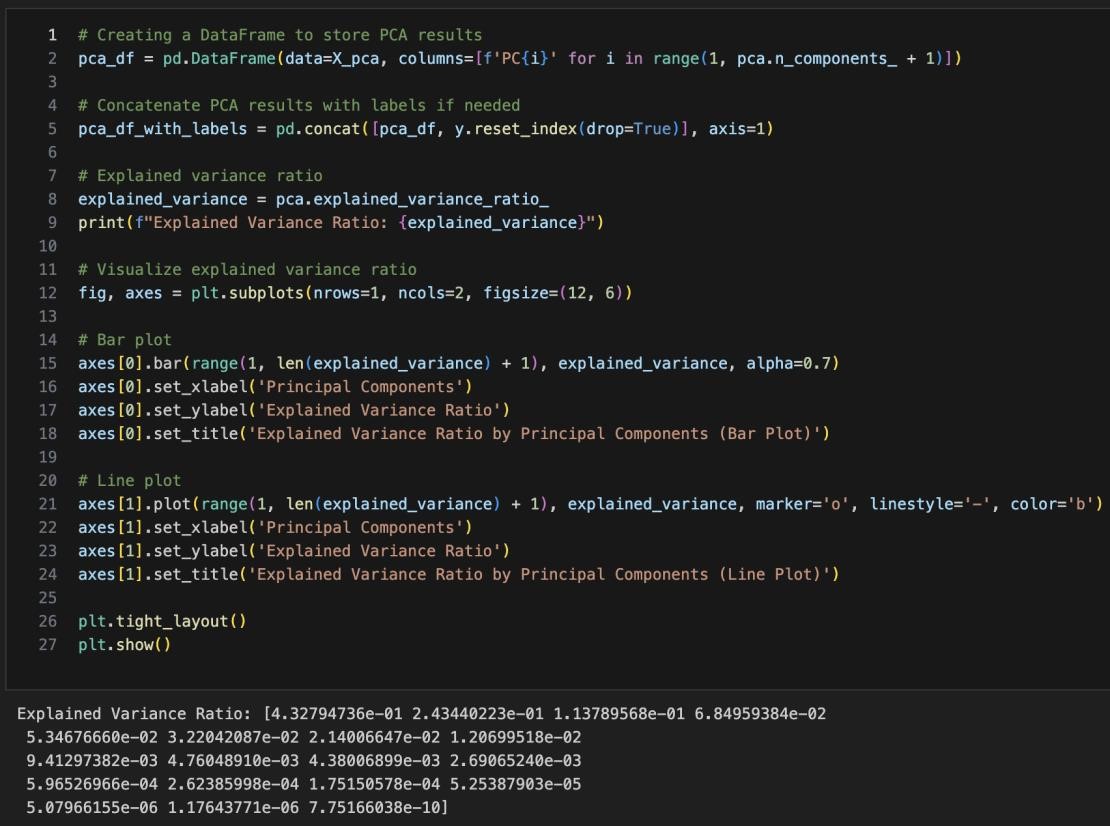
This step performs **Principal Component Analysis (PCA)**, a powerful technique for reducing the dimensionality of a dataset while retaining most of the information. PCA is useful when dealing with datasets that have many features, as it transforms the data into a new set of uncorrelated variables called **principal components**. These components capture the maximum variance in the data, helping to simplify the dataset while minimizing information loss.

A screen shot of a computer program

Description automatically generated

**Purpose of PCA in Data Analysis:**

* **Dimensionality Reduction**: PCA simplifies the dataset by reducing the number of features, making the data easier to work with, while still retaining most of the variance.
* **Avoid Overfitting**: By reducing the number of features, PCA helps to avoid overfitting in models, particularly when the dataset has many features compared to the number of observations.
* **Improves Model Efficiency**: PCA speeds up machine learning algorithms by reducing the feature space, making training and predictions more efficient.
* **Data Visualization**: By reducing the dataset to a few components, it becomes possible to visualize high-dimensional data in 2D or 3D, which can aid in understanding patterns or clusters in the data.



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.

A graph of a graph and a graph of a graph

Description automatically generated

**Detailed Steps:**

1. **Separating Features and Labels**:
   * **x = df.drop('Label', axis=1)**: The features (independent variables) are separated from the 'Label' column, which is the target (dependent variable).
   * **y = df['Label']**: The 'Label' column is stored in a separate variable for later use. This is the variable the model will try to predict.
2. **Standardizing the Features**:
   * **x\_scaled = StandardScaler().fit\_transform(x)**: Before applying PCA, the features are standardized so that each has a mean of 0 and a variance of 1. This ensures that PCA isn't biased by features with larger scales, as it is sensitive to variances.
3. **Performing PCA**:
   * **x\_pca = PCA().fit\_transform(x\_scaled)**: PCA is applied to the standardized features. It reduces the data into a new set of variables, known as principal components, which are linear combinations of the original features.
4. **Creating a DataFrame for Principal Components**:
   * **pca\_df = pd.DataFrame(x\_pca, columns=[f'PCA{i}' for i in range(1, x.shape[1] + 1)])**: The transformed data (principal components) is stored in a new DataFrame with columns labeled as PCA1, PCA2, etc., based on the number of original features.
5. **Adding Labels Back**:
   * **pca\_df\_labels = pd.concat([pca\_df, y.reset\_index(drop=True)], axis=1)**: The original target labels (y) are added back to the transformed dataset, creating a dataset that includes both the principal components and the target labels. This can now be used for further modeling or analysis.
6. **Calculating Explained Variance Ratio**:
   * **explained\_variance = PCA().fit(x\_scaled).explained\_variance\_ratio\_**: This calculates the percentage of variance explained by each principal component. It helps determine how much information each component retains from the original data. A few components can often explain most of the variance, reducing the need to use all features.
7. **Plotting Explained Variance**:
   * The code plots the **explained variance** using two plots:
     + **Bar Plot**: Shows how much variance is explained by each individual principal component.
     + **Line Plot**: Displays the cumulative explained variance, helping to visualize how many components are needed to retain most of the information.
   * These plots help decide how many principal components to keep for further analysis or model training.

# Feature selection

Feature selection is a critical step in data analysis because:

1. **Improves Model Accuracy**: By removing irrelevant or redundant features, the model can focus on the most useful data, leading to better performance.
2. **Reduces Overfitting**: Too many features can lead to overfitting, where the model performs well on training data but poorly on new, unseen data. Feature selection reduces the risk of overfitting.
3. **Simplifies the Model**: A simpler model with fewer features is easier to interpret, faster to train, and often performs just as well as, or even better than, a model with many features.
4. **Reduces Computational Cost**: With fewer features, less computational power is required for training, making the process more efficient, especially for large datasets.

A computer screen shot of a program code

Description automatically generated

**Random Forest Classifier for Feature Importance**:

* **What it does**: The Random Forest Classifier is trained on the dataset, and it automatically computes the importance of each feature by looking at how much each one contributes to making better decisions in the model.
* **Why we need it**: Feature importance helps identify which features (or variables) have the most predictive power. This is useful for simplifying the model by focusing on the most important features, improving both performance and interpretability.
* **Output**: It prints the ranking of features by their importance in the Random Forest model.

**Decision Tree Classifier for Feature Importance**:

* **What it does**: The Decision Tree Classifier is trained on the dataset. Like Random Forest, it identifies which features are the most relevant for making accurate predictions.
* **Why we need it**: It provides an additional method for evaluating feature importance. Using different models (like Random Forest and Decision Trees) ensures robustness in selecting important features.
* **Output**: This step does not print results directly but fits the Decision Tree model to be used for analysis later.

**Sequential Feature Selection (SFS)**:

* **What it does**: SFS is a method for selecting the best subset of features by adding one feature at a time and evaluating the model's performance (in this case, accuracy). It uses the Random Forest model to select 5 features in a forward direction (adding features one by one).
* **Why we need it**: SFS helps in selecting the best combination of features that maximizes model performance. It is especially useful when working with high-dimensional data where many features might be irrelevant or redundant.
* **Output**: The most important 5 features selected by the Random Forest model.

**Recursive Feature Elimination (RFE) with Logistic Regression**:

* **What it does**: RFE recursively removes less important features and refits the model, ranking the features based on their contribution to the Logistic Regression model. It selects 5 features that are most significant.
* **Why we need it**: RFE helps identify the most important features for a model by gradually eliminating the weakest ones. This ensures that only the most predictive features remain, which simplifies the model and may improve performance.
* **Output**: The top 5 features selected based on their importance in the Logistic Regression model.

**Conclusion: What This Step Accomplishes**

By applying different feature selection techniques (Random Forest, Decision Tree, SFS, and RFE), this code helps identify the most important features for prediction. The result is a streamlined dataset where only the most significant features are retained. This reduces complexity, improves model accuracy, and prevents overfitting. With selected important features, the next steps in model development and evaluation become more efficient and effective, ensuring better overall model performance.

A screen shot of a computer

Description automatically generated

# Model Development and Evaluation

This step focuses on model development and evaluation, where the goal is to build a logistic regression model and assess its performance using different feature sets.

A screen shot of a computer program

Description automatically generated

**Purpose**: This function builds and evaluates a **Logistic Regression model** using **train-test split**. It helps us understand how well the model is performing by using three key metrics:

* **Accuracy**: The proportion of correctly predicted instances out of the total.
* **Precision**: The ability of the model to correctly predict the positive class out of all positive predictions.
* **Recall**: The ability of the model to capture all relevant instances of the positive class from the dataset.

**Step-by-Step Breakdown:**

1. **Train-Test Split**:
   * The dataset is split into **training** (70%) and **testing** (30%) sets using train\_test\_split(). This helps assess model performance on unseen data.
2. **Training the Model**:
   * A **Logistic Regression model** is initialized and trained on the training data using the fit() method.
3. **Predictions**:
   * The model makes predictions on the **test data** using the predict() function.
4. **Evaluation**:
   * The performance of the model is evaluated using three metrics: **accuracy**, **precision**, and **recall**. These metrics provide insights into how well the model is performing and whether it is correctly predicting the target variable.
5. **Return**:
   * A screen shot of a computer code

     Description automatically generatedThe function returns the computed **accuracy, precision, and recall** values for further comparison.

This step evaluates how well the **Logistic Regression model** performs with two different feature sets—one created using **PCA** and the other created using **RFE**. The results provide valuable insights:

* It helps decide which feature selection or dimensionality reduction technique is better suited for this dataset.
* The evaluation process ensures that the selected features or components improve model accuracy without overfitting, leading to a more efficient and robust predictive model.

By running these evaluations, we ensure that the final model is not only accurate but also efficient, using the best combination of features.

**PCA Components Evaluation:**

* **Accuracy: 0.2891**: The model correctly predicted the outcome around 28.91% of the time, which is quite low. This suggests that the model is not very effective at classifying the data using the principal components derived from the PCA.
* **Precision: 0.3011**: Precision measures how many of the positive predictions were actually correct. A precision of 0.3011 means that when the model predicts a certain class as positive, it's correct about 30.11% of the time.
* **Recall: 0.3038**: Recall measures how well the model captures all the true positive instances in the dataset. A recall of 0.3038 means the model identifies only about 30.38% of the actual positive cases.

**RFE-Selected Features Evaluation:**

* **Accuracy: 0.2969**: Similar to PCA, the accuracy is low at 29.69%, indicating that the model struggles to classify the data with the features selected by RFE.
* **Precision: 0.2506**: The precision dropped further to 25.06%, meaning the model is even less reliable at predicting the correct positive class.
* **Recall: 0.3255**: The recall is slightly higher (32.55%), meaning that this model is slightly better at capturing true positive cases than the PCA model but still underwhelming.

# Conclusion

**Overall Model Performance**: Both PCA and RFE feature sets lead to poor performance. The accuracy is barely above random guessing (which would be around 33.33% for a 3-class problem), indicating that the model is struggling to find meaningful patterns in the data with the selected features.

**Low Precision and Recall**: Both metrics are also low, meaning the model is neither making accurate positive predictions nor capturing all the positive instances. This suggests that the model might be misclassifying instances frequently.