Ain Shams University - Faculty of Engineering

CSE381 (UG2018) - Introduction to Machine Learning – Fall 24

Project – Phase 1

Mohamed Omar Adel ElBialy	21P0068
Abdelrahman Sherif Abdelaziz	21P0098
Ezz Eldin Alaa Eldin Omar	21P0100

Table of Contents

This page was intentionally left blank

Milestone 1

This milestone focuses on the key steps of the machine learning workflow, including Data Exploration and Visualization, Data Cleaning and Processing, as well as training and testing classifiers. Specifically, we will implement Naïve Bayes and Support Vector Machine (SVM) models to predict heart disease outcomes.

1.Introduction

Heart disease remains one of the leading causes of morbidity and mortality worldwide, necessitating the development of efficient diagnostic tools to aid in early detection and treatment. In this project, we leverage machine learning (ML) techniques to build a predictive model for identifying heart diseases based on patient data. The goal is to enhance diagnostic accuracy and provide timely interventions, ultimately improving patient outcomes. By utilizing various ML algorithms, this model aims to analyze patterns in health data, offering a reliable and scalable solution to support healthcare professionals in diagnosing and managing heart conditions effectively.

2.Libraries

These are the libraries to be used in **Milestone 1:**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score ,f1_score, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from decimal import Decimal, ROUND_HALF_UP
```

In this project, several libraries for data analysis, machine learning, and visualization were used. **Pandas** (imported as pd) are used for data manipulation and handling structured data with DataFrames and Series.

NumPy (np) supports numerical computing and array manipulation, while **Matplotlib** (plt) is used for creating static, animated, and interactive visualizations. **Seaborn** (sns) extends Matplotlib's capabilities for statistical graphics. **Scikit-learn** provides numerous tools for machine learning: train_test_split helps divide the data into training and testing sets, **PCA** is used for dimensionality reduction, and **StandardScaler** and **MinMaxScaler** are used for feature scaling and normalization. For model evaluation, **accuracy_score**, **precision_score**, **recall_score**, **f1_score**, and **confusion_matrix** provide various metrics to assess classification performance. The **Gaussian Naive Bayes** model (GaussianNB) is used for probabilistic classification, and **GridSearchCV** helps in hyperparameter tuning. The **Support Vector Classifier (SVC)** is another classification model in Scikit-learn, while **Decimal** and **ROUND_HALF_UP** from the decimal module are used for precise floating-point arithmetic and rounding. Each tool or module serves a distinct role in processing, modeling, and evaluating data efficiently.

3.Data preprocessing and cleaning

Kindly note that PCA is done but after data preprocessing.

3.1 Introduction

Data preprocessing and cleaning is a critical step in any machine learning pipeline, particularly in healthcare applications where data can be noisy, incomplete, and imbalanced. In this project, we focus on preparing the raw healthcare data for analysis by handling missing values, normalizing features, encoding categorical variables, and addressing class imbalances. These preprocessing steps are essential for ensuring that the model can effectively learn patterns and make accurate predictions related to heart disease diagnosis. Proper preprocessing improves the model's performance and reliability, ultimately leading to better predictive outcomes.

3.2 Process

Most of the data analysis steps and procedures are thoroughly explained in the notebook; however, we will briefly review them here to eliminate any ambiguity.

We started up by visualizing the data in form of rows and columns, to get more sense on what we are working on with; get familiar with the data. It seems that we have 918 rows of data, uncleaned and ready to be processed.

We will check for duplicates or missing values in rows, apparently, we don't have any:

```
[4]  df.isna().sum()

...  Age              0
     Sex              0
     ChestPainType    0
     RestingBP        0
     Cholesterol      0
     FastingBS        0
     RestingECG       0
     MaxHR            0
     ExerciseAngina   0
     Oldpeak          0
     ST_Slope         0
     HeartDisease     0
     dtype: int64
```

Checking for duplicates

```
[5]  df.duplicated().sum()

...  0
```

3.2.1 Data encoding

Before moving on and checking for outliers, we must **encode** all the string/text values in our dataset, to get a better visualization of the data.

We will be using label encoding in this part.

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

918 rows × 12 columns

Dataset before encoding

In the notebook, there is more explanation of what each number in the encoding represents and its past value.

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	1	140	289	0	1	172	0	0.0	2	0
1	49	0	2	160	180	0	1	156	0	1.0	1	1
2	37	1	1	130	283	0	2	98	0	0.0	2	0
3	48	0	0	138	214	0	1	108	1	1.5	1	1
4	54	1	2	150	195	0	1	122	0	0.0	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	1	3	110	264	0	1	132	0	1.2	1	1
914	68	1	0	144	193	1	1	141	0	3.4	1	1
915	57	1	0	130	131	0	1	115	1	1.2	1	1
916	57	0	1	130	236	0	0	174	0	0.0	1	1
917	38	1	2	138	175	0	1	173	0	0.0	2	0

918 rows × 12 columns

Dataset after encoding

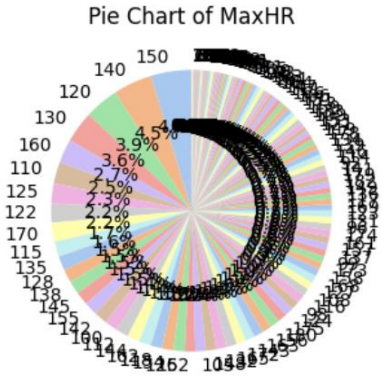
As we see, now we have a “numerical” dataset now which can be easily manipulated and visualized.

3.2.2 Outliers

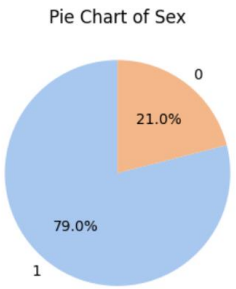
Now to identify outliers, we will be using the interquartile method, where we first determine the First Quartile (Q1) and Third Quartile (Q3). Q1 represents the 25th percentile, which means that 25% of the data lies below this value, while Q3 represents the 75th percentile, meaning 75% of the data lies below it. These quartiles are computed by finding the median of the lower half of the data for Q1 and the median of the upper half of the data for Q3. Once these quartiles are computed, we can

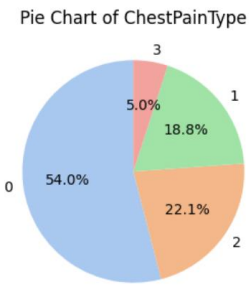
calculate the Interquartile Range (IQR), which is the difference between Q3 and Q1, and represents the range where the middle 50% of the data lies. The next step is to determine the outlier boundaries, which are calculated by applying a factor of 1.5 times the IQR to both the lower and upper quartiles. Specifically, the lower boundary is found by subtracting 1.5 times the IQR from Q1, while the upper boundary is determined by adding 1.5 times the IQR to Q3. Any data point that falls outside of these boundaries, either below the lower bound or above the upper bound, is considered an outlier. Once outliers are identified, they can be removed from the dataset to ensure a more accurate analysis or model-building process.

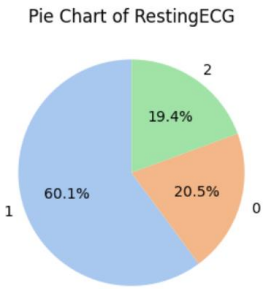
We will firstly be visualizing the data with boxplots, pie charts and histograms. Boxplots are very helpful in visualizing the IQRs. Note that the pie charts aren't that helpful in presenting features with lots of continues values in them, for example:



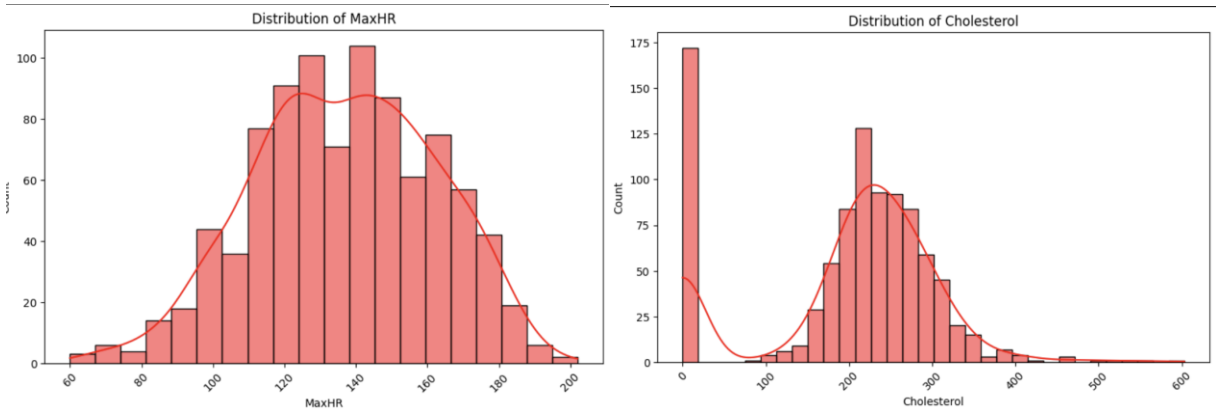
Below is an example for the visualization of the pie charts:





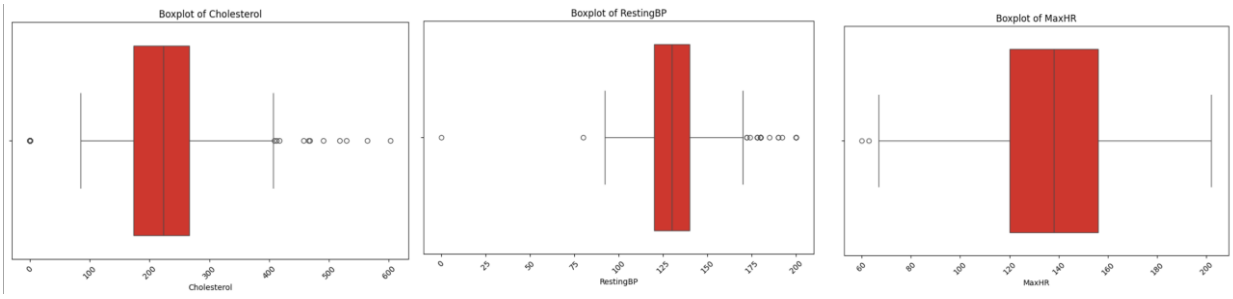


Below is an example for the visualization of the histograms:



Note that in a discrete data set, histograms are of no great use.

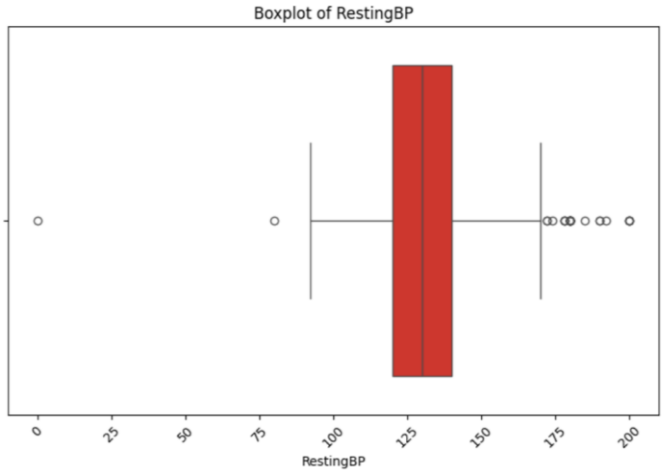
Below is an example for the visualization of boxplots:



After further analysis of the visualization of the data, we concluded that we had 4 features with possible outliers in them.

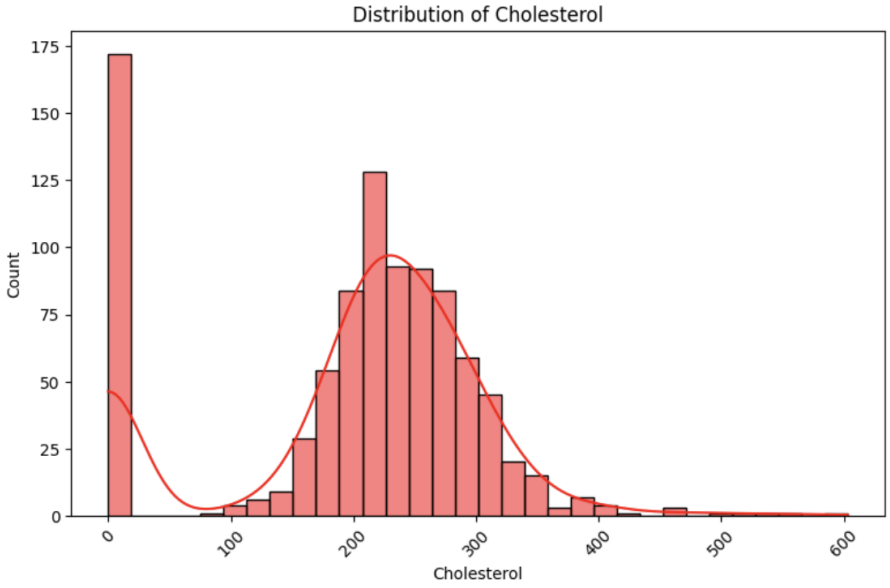
1. “RestingBp”

Because as we see here:



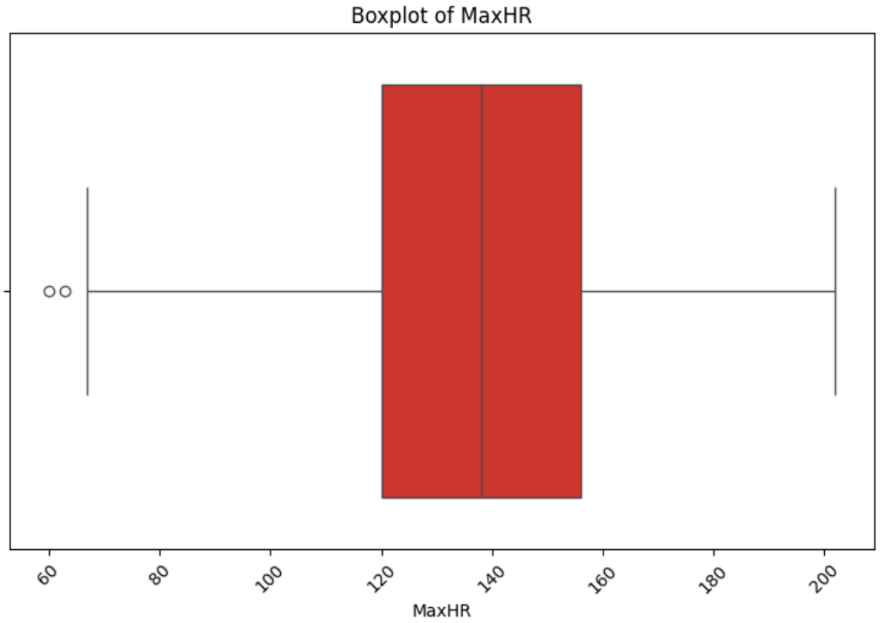
There’s plenty of outliers outside Q1 and Q3.

2. “Cholesterol”

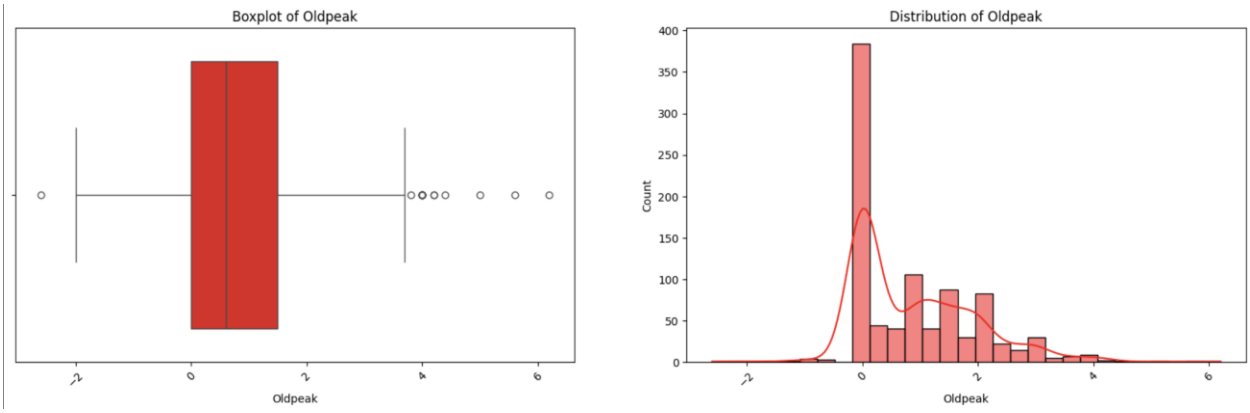


Same goes here + we have about 170 data rows with 0 cholesterol, which is nonsense.

3. “Maxhr”



4. “Old peak”



Now the new data set with no outliers:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	1	140	289	0	1	172	0	0.0	2	0
1	49	0	2	160	180	0	1	156	0	1.0	1	1
2	37	1	1	130	283	0	2	98	0	0.0	2	0
3	48	0	0	138	214	0	1	108	1	1.5	1	1
4	54	1	2	150	195	0	1	122	0	0.0	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...
697	45	1	3	110	264	0	1	132	0	1.2	1	1
698	68	1	0	144	193	1	1	141	0	3.4	1	1
699	57	1	0	130	131	0	1	115	1	1.2	1	1
700	57	0	1	130	236	0	0	174	0	0.0	1	1
701	38	1	2	138	175	0	1	173	0	0.0	2	0

702 rows × 12 columns

We now have 702 data rows.

Notice that we dropped 216 rows of data, most of them were the 0 cholesterol rows, and the rest are simple outliers.

In the notebook, you’ll find the new cleaned visualization of the data.

4 PCA

Now for the PCA, instead of simply running the PCA algorithm directly, we tried to do some classic “PCA” steps, that is to first explore the data to understand the features and their distributions. Next, check for collinearity by calculating the correlation to identify any highly correlated features, which we may want to remove or combine. It’s also crucial to standardize or normalize the data. This step will be implemented and discussed later

on but for now we will do the correlation step before standardizing and once again after standardizing.

Note that we weren't sure about whether we had to drop the target column or not, so we decided to run the PCA algorithm twice, once with no target and another time with the target.

4.1 Before Standardization

4.1.1 Correlated Data

As shown in the figure below, we focused on correlations greater than 0.5, as those below 0.5 indicate weak correlations and are not meaningfully related to each other. Therefore, the features with

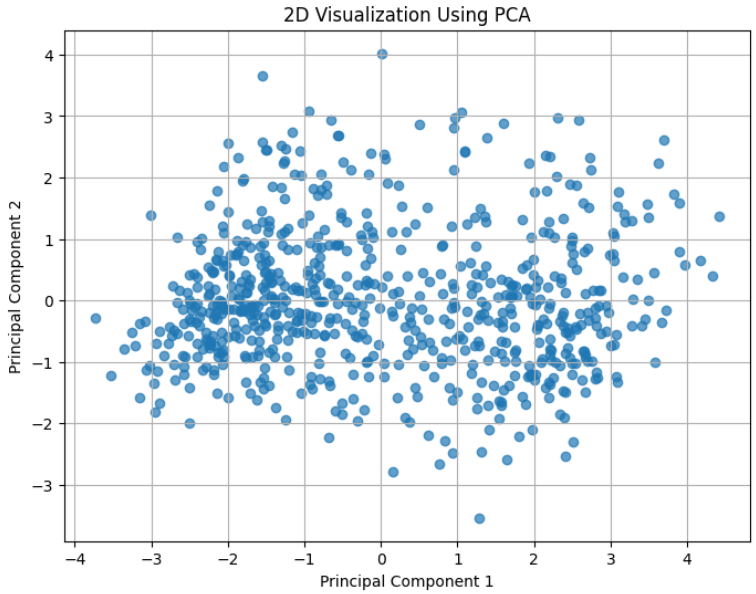
correlations below 0.5 were neglected.

```
ST_Slope      HeartDisease   -0.558771
HeartDisease  ST_Slope       -0.558771
Oldpeak       ST_Slope       -0.501921
ST_Slope      Oldpeak        -0.501921
dtype: float64
```

A correlation of 55% and 50% is not considered strong enough to warrant combining the features, so these will be discarded. This leaves us with the remaining features that are not highly correlated. After standardizing the data, we hope to identify features with better correlations. For now, however, we will proceed by directly implementing the PCA algorithm.

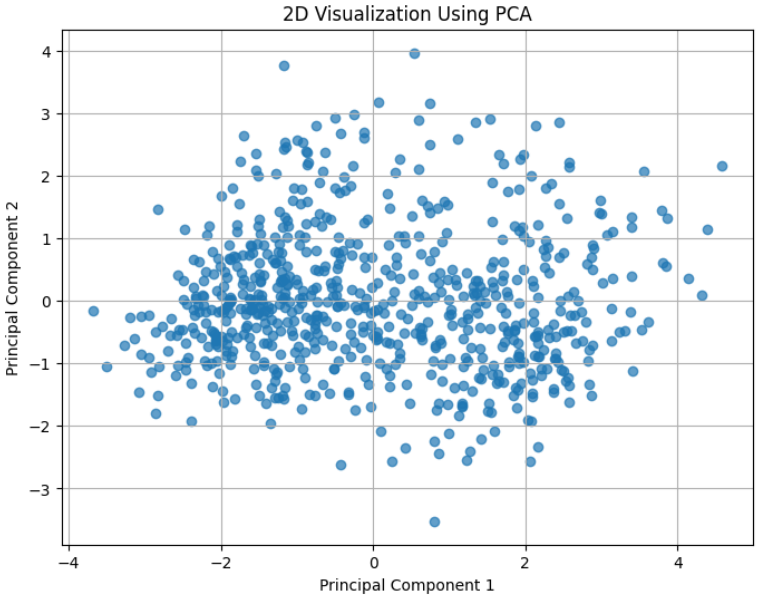
4.4.2 PCA Algorithm

In the notebook, you'll find the implementation of the PCA algorithm, nothing special about the code, below is a visualization of the output.



As we can see, the data are well spread all along the x and the y axis, although it's more spread on the x-axis than the y-axis.

Now without the target 'HeartDisease':



4.2 After Standardization

4.2.1 Correlated Data

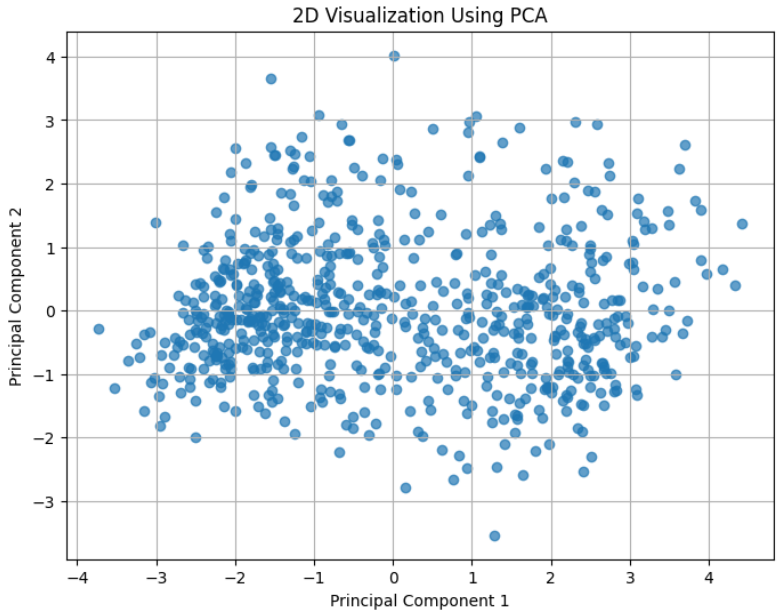
We will recheck the correlation after standardizing the data, as shown in the figure below.

ST_Slope	HeartDisease	-0.598696
HeartDisease	ST_Slope	-0.598696
Oldpeak	ST_Slope	-0.598051
ST_Slope	Oldpeak	-0.598051
Oldpeak	HeartDisease	0.502095
HeartDisease	Oldpeak	0.502095
ExerciseAngina	HeartDisease	0.550326
HeartDisease	ExerciseAngina	0.550326

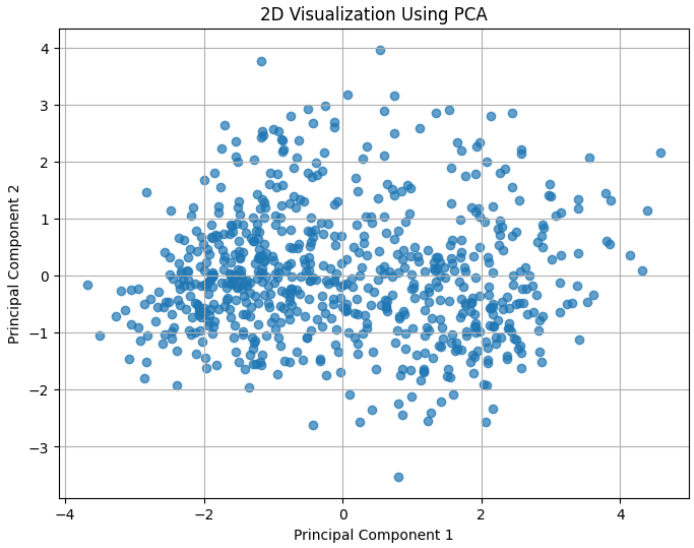
After standardization, we observe an improvement with a new correlation value of 59%. However, this is still not strong enough to warrant combining the features, so we will proceed with running the PCA algorithm as we did before standardizing the data.

4.2.2 PCA Algorithm

Below is the new PCA plot output, like the one before standardization.



Now without the target ‘HeartDisease’:



5 Data Standardization

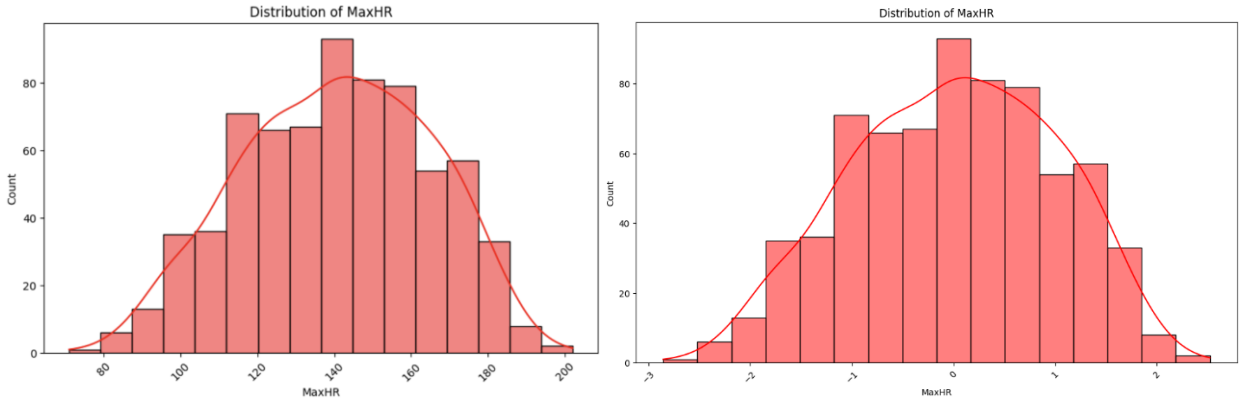
Data standardization is the process of scaling features to have a mean of zero and a standard deviation of one. This ensures that each feature contributes equally to the analysis, especially when the features have different units or scales. Standardization is crucial for data that are sensitive to the scale of the data, as it helps improve the performance and accuracy of the model by treating all features on the same scale.

In our data set, we have continuous features like 'Age', 'RestingBP', 'Cholesterol', 'MaxHR', and 'Oldpeak' often have different scales or units of measurement, which can impact the performance of certain algorithms. For example, Age might range from 20 to 80, Cholesterol might vary from 100 to 400 (in our case), and so on...

5.1 Standardizing-Normalizing

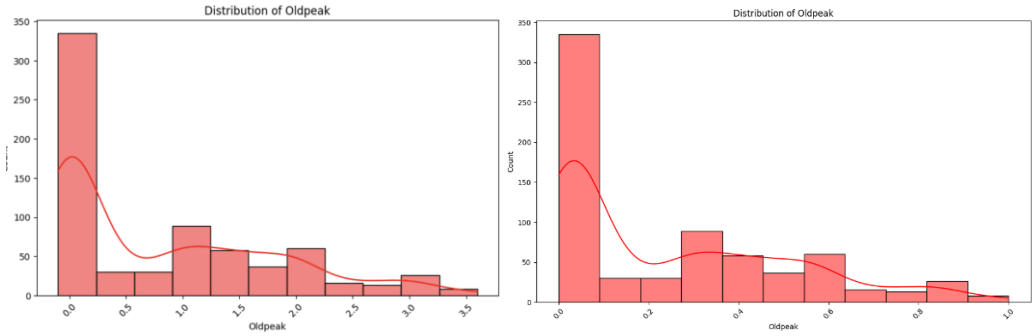
Standardization is applied to address this issue. It transforms each feature to have a similar scale, which helps to ensure that no feature dominates the learning process simply due to its scale. This is typically achieved through two main approaches, Z-score and Min-Max.

For ‘Age’, ‘RestingBP’, ‘Cholesterol’, ‘MaxHR’, since they are Gaussian distributed, we will use Z-score on them (standardized).



We can see above the ‘MaxHR’ before and after standardizing, as we can see the range have changed from 80-200 to -3-3. The other figures are available in the notebook.

Now for ‘Oldpeak’ will use Min-Max on it (normalized)



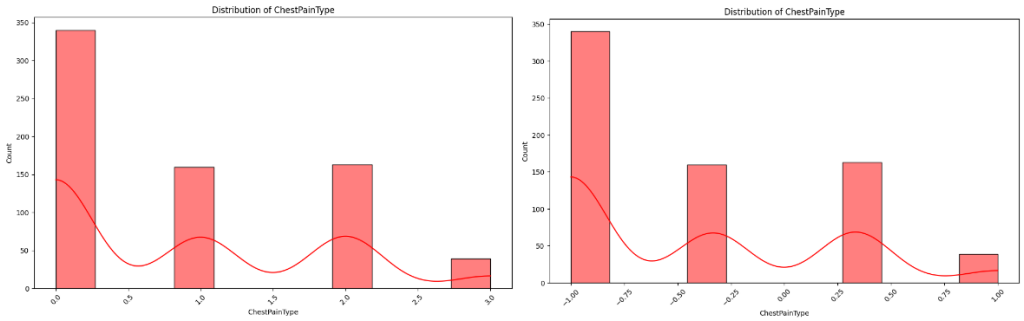
5.2 Scaling

Lastly, we will be scaling all our features to be ranging from -1 to 1.

Scaling the data to the range of -1 to 1 helps to ensure that all features contribute equally to the analysis and prevents features with larger ranges from dominating. This is particularly useful when the algorithm uses a distance-based or similarity-based approach, which could be distorted if features with large numerical ranges are not scaled appropriately. The range of -1 to 1 ensures that the data is centered around 0, which can be

beneficial for certain algorithms such as neural networks, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN).

Continuous features will range from -1 to 1 as expected, but discrete values that were binary such as sex, will be -1 and 1 instead of 0 and 1, others discrete features that have values like 0,1,2,3 will also be ranging from -1 to 1.



Below is the head of the final dataset that we will be using (in case of working with standardized data set).

	Age	RestingBP	Cholesterol	MaxHR	Oldpeak	Sex	ChestPainType	\
0	-0.510204	0.230769	0.278997	0.541985	-0.945946	1.0	-0.333333	
1	-0.142857	0.743590	-0.404389	0.297710	-0.405405	-1.0	0.333333	
2	-0.632653	-0.025641	0.241379	-0.587786	-0.945946	1.0	-0.333333	
3	-0.183673	0.179487	-0.191223	-0.435115	-0.135135	-1.0	-1.000000	
4	0.061224	0.487179	-0.310345	-0.221374	-0.945946	1.0	0.333333	

	FastingBS	RestingECG	ExerciseAngina	ST_Slope	HeartDisease
0	-1.0	0.0	-1.0	1.0	-1.0
1	-1.0	0.0	-1.0	0.0	1.0
2	-1.0	1.0	-1.0	1.0	-1.0
3	-1.0	0.0	1.0	0.0	1.0
4	-1.0	0.0	-1.0	1.0	-1.0

6 Training, Testing and Validation Datasets

Splitting the data into training, testing, and validation sets is a fundamental step in machine learning to evaluate and improve model performance. Here's what each set is used for:

Training Set: This is the subset of the data used to train the model. The model learns patterns, relationships, and features from this data to make predictions. Typically, the majority of the data (60-80%) is allocated to the training set.

Testing Set: The testing set is used to evaluate the final performance of the model after it has been trained. This data is never used during training, ensuring an unbiased assessment of how well the model generalizes to new, unseen data. It typically represents 10% of the total dataset.

Validation Set: The validation set is used during the model development process to tune hyperparameters (like learning rate, depth of decision trees, etc.) and to choose the best model. It helps in assessing the model's performance while avoiding overfitting to the training set. In some cases, cross-validation is used instead of a separate validation set.

The goal of splitting the data is to ensure that the model performs well not just on the training data but also on new, unseen data, reflecting its true predictive power. Proper data splitting helps avoid overfitting and ensures that the model generalizes well to real-world scenarios.

In this notebook, we will split our data into training, testing, and validation sets in the standard way. However, we will create two separate sets: one before standardizing the data and one after standardization. This will allow us to run our models on both sets and compare the results to assess the impact of standardization on model performance.

7 Naïve Bayes

We will import the Naïve Bayes algorithm and use it directly to train our model, note that we will be using something called “var smoothing”.

In **Gaussian** Naive Bayes, we assume that the features follow a Gaussian distribution for each class. When the model calculates the likelihoods for each class, it uses the mean and variance of each feature to determine the probability of observing a given feature value. However, if any feature has

a variance of zero or a very small value, for example, a feature that has the same value for all data points in a class, it can cause problems, such as:

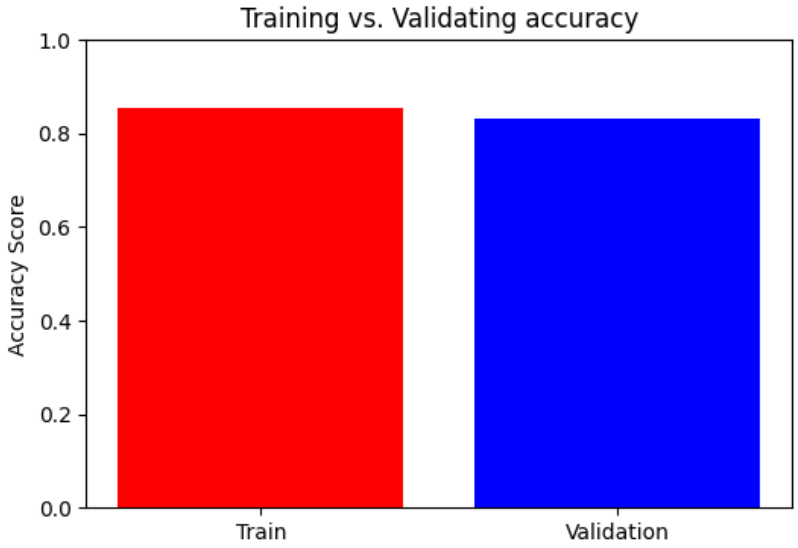
Zero variance: If the variance of a feature is zero, the model cannot compute a meaningful standard deviation. This would result in a division by zero in the probability calculation.

Very small variance: A very small variance can cause numerical instability or lead to overconfidence in the probability estimates.

To address this, variance smoothing adds a small value to the variance, preventing it from being exactly zero and making the model more stable.

7.1 Naïve Bayes without Standardization

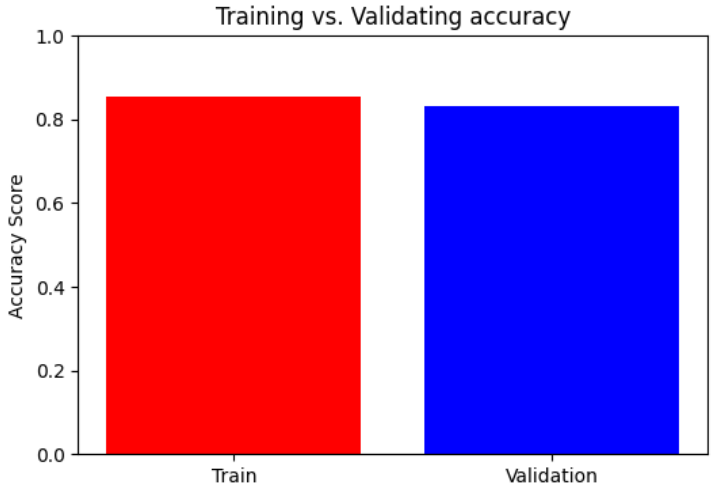
Below is the training accuracy compared to the validation one.



We achieved a training accuracy of 85.5% and a validation accuracy of 83.34%.

7.2 Naïve Bayes with Standardization

Below is the training accuracy compared to the validation one.



Note that we achieved the exact same results as without standardization, that's because Naïve Bayes algorithm doesn't really get affected by standardization.

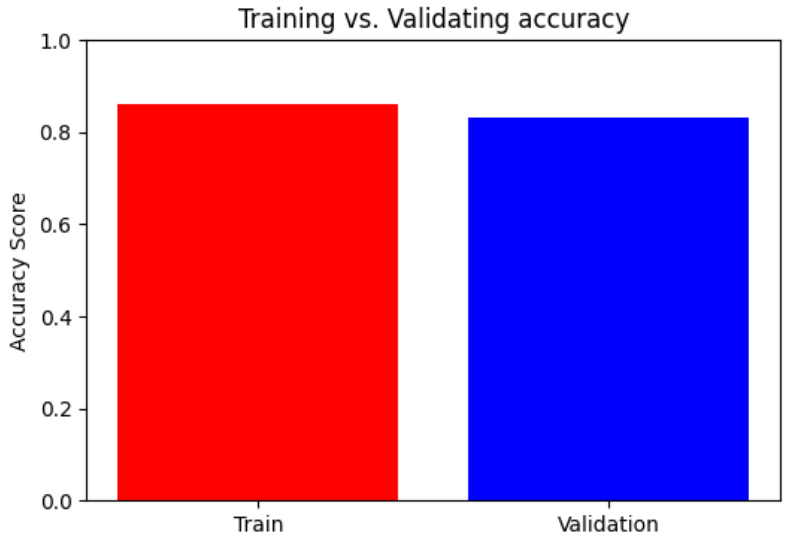
7.3 Grid Search

We will be performing the grid search on the var smoothing parameter, since the Naïve Bayes algorithm doesn't actually have that much of parameters, really "naïve" 😊.

We will see that we will get a var smoothing value of "1xe^{-5}", we previously started with "1xe-7".

Running it will give us a slight improvement in training accuracy, that will now be 85.98%, it was 85.5%.

Below is the new plot.



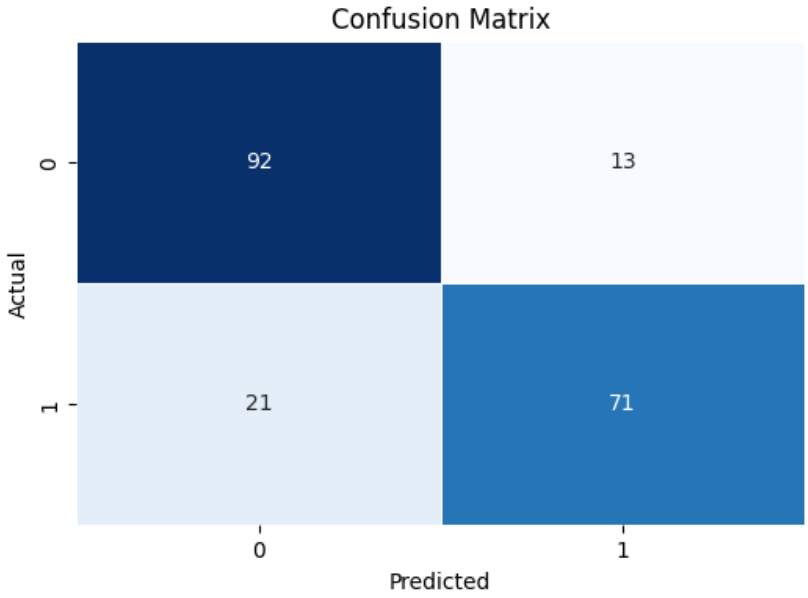
As we can see, it looks like the one before the grid search.

7.4 Performance Analysis

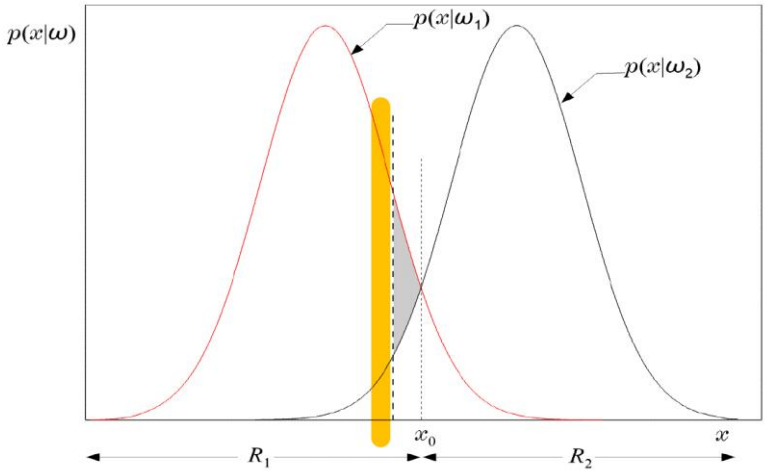
We will now go over the Accuracy, Precision, Recall, F1 Score and Confusion matrix.

```
Accuracy: 0.8274111675126904
Precision: 0.8452380952380952
Recall: 0.7717391304347826
F1 Score: 0.8068181818181818
```

As we can see the recall is lower than the precision, which isn't a good thing as this means that the model is good at avoiding false positives (which improves precision) but fails to identify all the true positives (which lowers recall). In our case this isn't good as we are currently identifying people who have heart disease as they don't, which is very critical and danger.

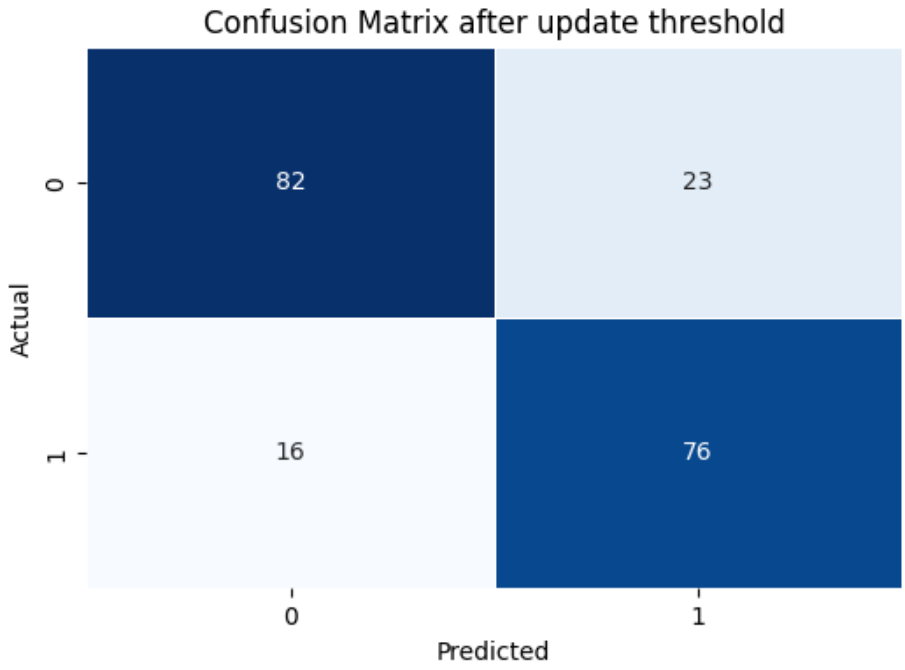


We have 21 false negative. To fix this problem, we will try to shift the threshold a bit to try to lower the false negatives, this will affect the other results but we will at least lower the false negatives which is crucial in our case.



We will choose the best threshold based on something similar to the grid search, but for threshold.

We will find that the best threshold is 0.1.



Now we have 16 false negatives instead of 21.

New scores:

```
Accuracy: 0.8020304568527918
Precision: 0.7676767676767676
Recall: 0.8260869565217391
F1 Score: 0.795811518324607 3
```

Now the recall is better than the precision, which is a good indication.

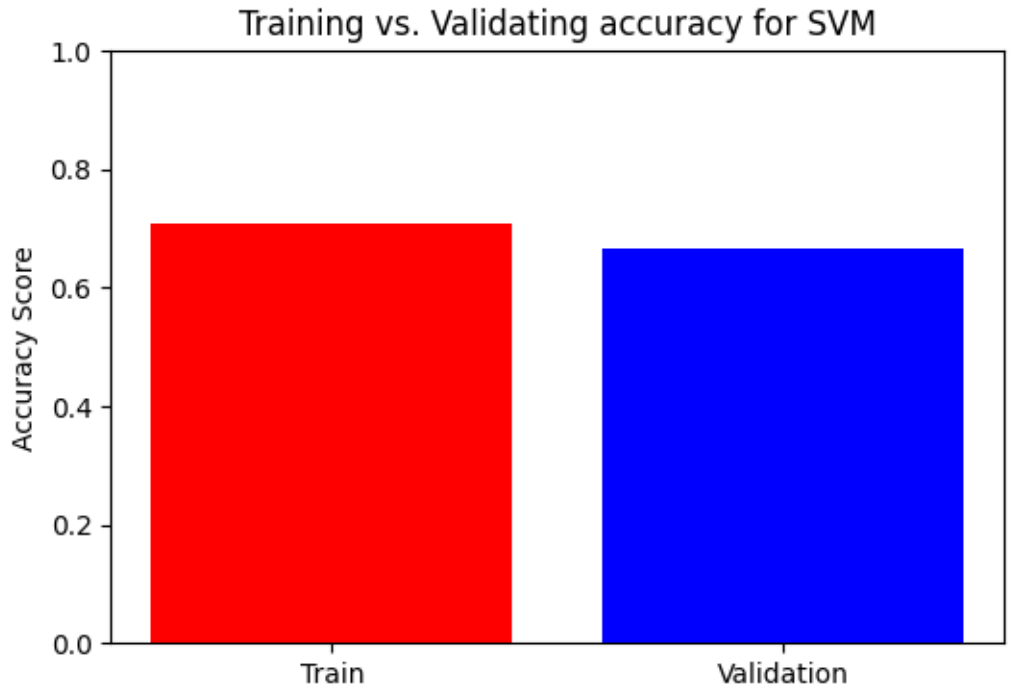
8 Support Vector Machine

Support Vector Machine (SVM) is a powerful supervised learning algorithm commonly used for classification tasks, although it can also be applied to regression. The main idea behind SVM is to find a hyperplane that best separates data points from different classes in a high-dimensional space.

In the case of binary classification, SVM works by identifying a hyperplane that maximizes the margin (distance) between the two classes. This hyperplane is determined by a subset of the training data points known as support vectors, which are the data points closest to the hyperplane. These support vectors are critical in defining the optimal decision boundary.

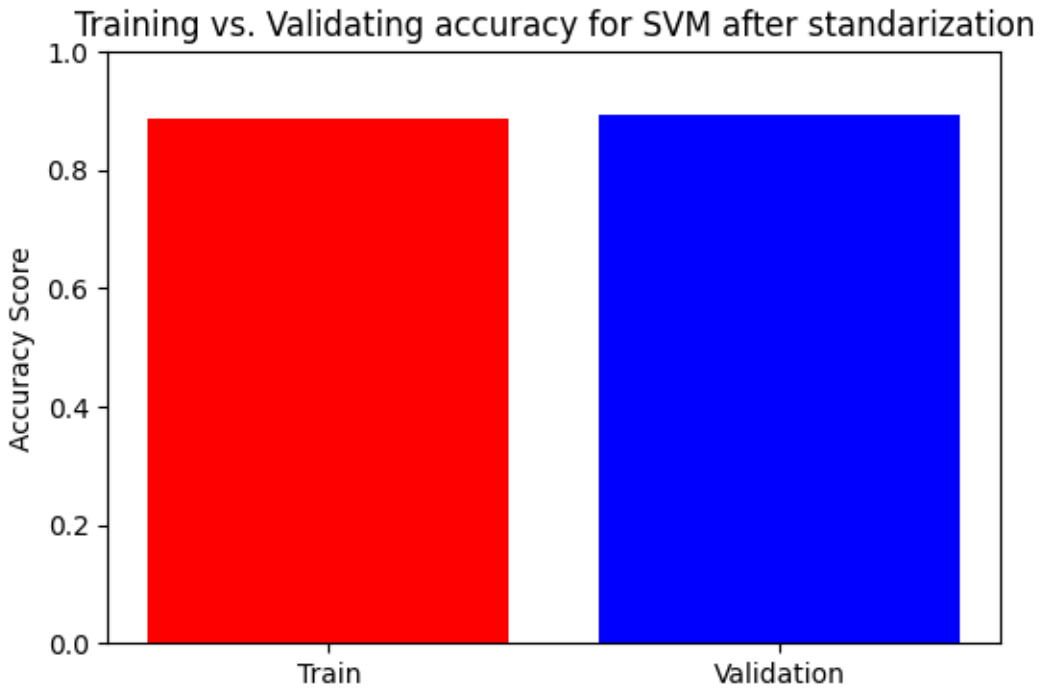
8.1 SVM without Standardization

Training SVM without Standardization is very bad, we will get training accuracy of 70% and validation accuracy of 66.6%.



8.2 SVM with Standardization

With standardization, we will begin to see a huge improvement in accuracy, with the training accuracy being 88.5% and the validation accuracy being 89.28%, the validation accuracy being better than the training accuracy isn't a good indication as we might be suffering from overfitting.



This problem will be solved later.

8.3 Grid Search

The two parameters that we will be tunning are the regularization parameter “C” and the kernel.

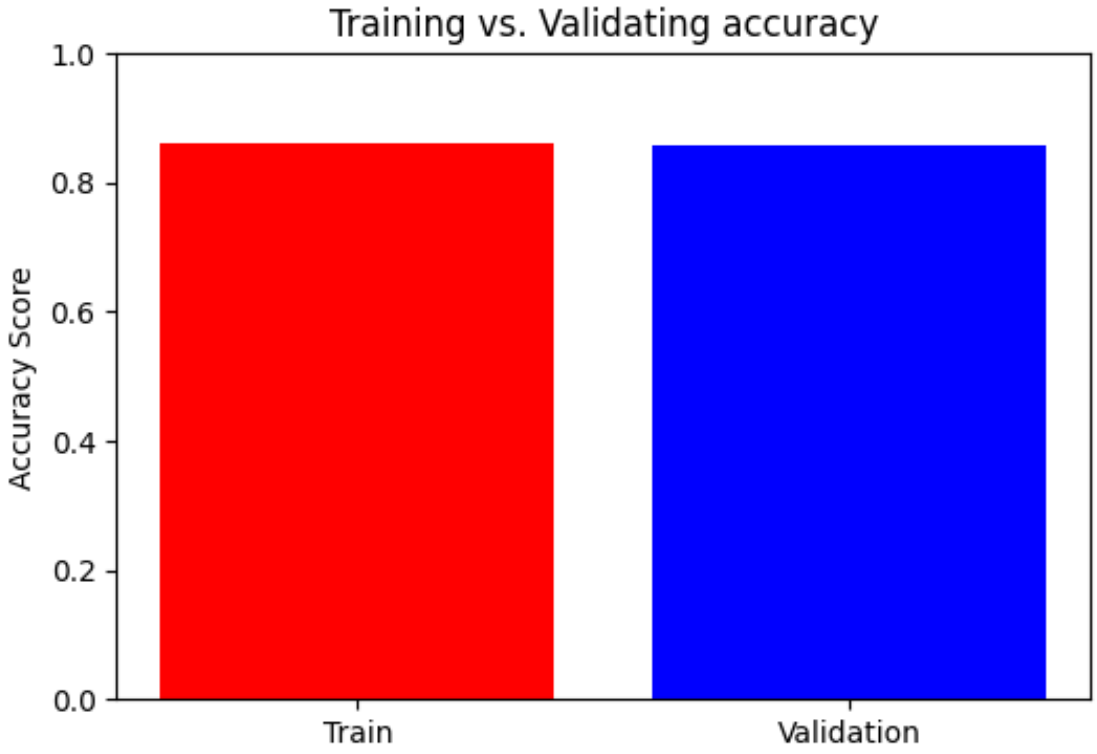
C: Controls the trade-off between training error and model complexity. Higher values of C lead to a more complex model that is sensitive to training data, and lower values of C make the model simpler and more general.

Kernel: Defines the type of decision boundary (linear or non-linear) by specifying a function to transform the data. Common choices are linear, RBF (Radial Basis Function), polynomial, and others.

8.3.1 Grid Search without Standardization

After running our grid search, the best C will be 26 and the best Kernel will be a linear one.

We will now see a huge improvement in the training and validation accuracy, which was previously 70% and 66%, it will now be 86% and 85%.



8.3.2 Grid Search with Standardization

In the notebook, you'll see that we had a small issue in the max value of the "x_train" as it is 1.0000000000000002, so it will be adjusted to be 1; so that our range stays perfect, -1 to 1.

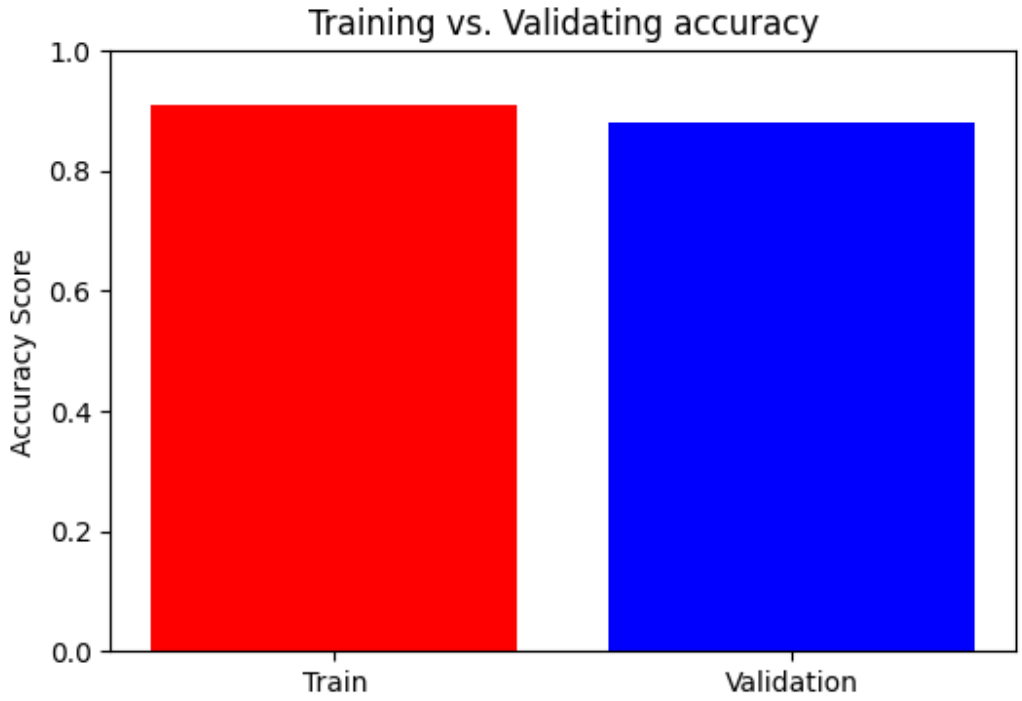
In this part, we will be adding the 'poly' parameter to the kernel's grid search, we didn't add it before because grid search without standardization takes a lot of time, also we will add a new hyperparameter called 'gamma' parameter, the gamma parameter defines the influence of each training sample and plays a key role in shaping the decision boundary of the model:

Low gamma makes the decision boundary smoother and simpler, which might result in underfitting.

High gamma makes the decision boundary more complex and detailed, which can result in overfitting if too high.

After running the grid search, we will end up with C of value 2 and Kernel of ‘poly’ and gamma of value 0.2.

Running the model with the new parameters will give us a training and validation accuracy of 90.0% and 88.09%.



As we can see now, the problem that we previously had, when we had validation accuracy higher than the training one, is now fixed.

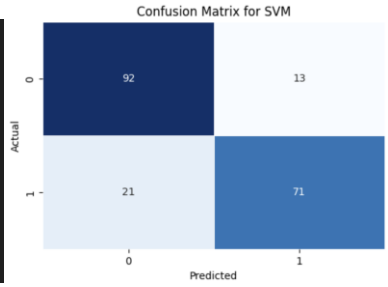
8.4 Performance Analysis

Since the output of the training, testing, and validation accuracies differ with and without standardization, we will check the performance analysis for both.

8.4.1 Performance Analysis without Standardization

Without Standardization, we will notice that the recall is lower than the precision.

```
Accuracy: 0.8274111675126904
Precision: 0.8452380952380952
Recall: 0.7717391304347826
F1 Score: 0.8068181818181818
```

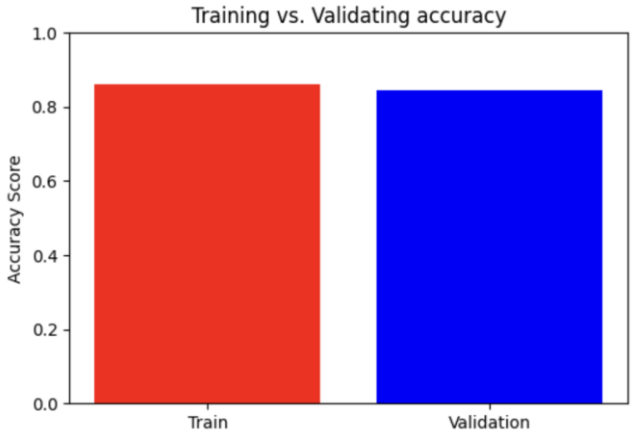


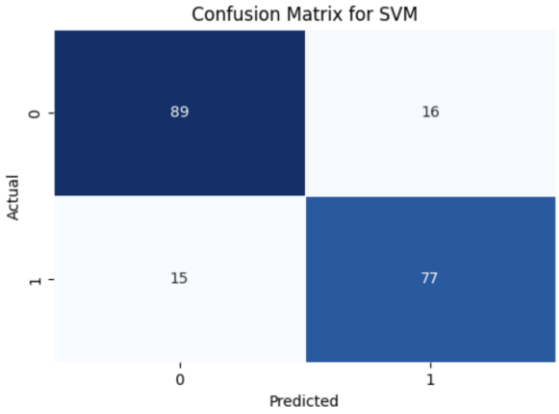
Since the grid search without Standardization for SVM takes a lot of time (20-30 minutes), we will try to train a new model and give more bias to the '1' class.

More explanation to what we are trying to do; class weight parameter explicitly assigns a higher weight to class 1. This means that the model will treat class 1 as more important and try to classify it more accurately, which can improve recall for class 1.

We will find the best weight to give to the '1' class is 1.3 times more important than class '0', below is the new precision and recall and the new plot of the training and validation accuracy.

The new scores:

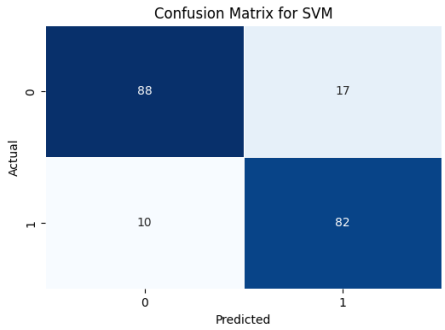





```
Accuracy: 0.8426395939086294
Precision: 0.8279569892473119
Recall: 0.8369565217391305
F1 Score: 0.8324324324324325
```

8.4.2 Performance Analysis with Standardization

And finally, we will find that the scores of the standardized model after grid search and after trying many times to find the best weights for class '1' will be 2.55 times more important than the '0' class, this will result in the following confusion matrix and scores:




```
Accuracy: 0.8629441624365483
Precision: 0.8282828282828283
Recall: 0.8913043478260869
F1 Score: 0.8586387434554974
```