PREDICTING HEART FAILURE OUTCOMES USING MACHINE LEARNING MODELS

TEAM MEMBERS

AKSHAY J P
FAHAD MUHAMMED M
GHANASHYAM T V
JIBIN K
MOHAMMED FAHIM P

PROJECT SUPERVISOR

DR. VIMALA MATHEW

CONTENTS

ABSTRACT

Heart failure is a leading cause of morbidity and mortality, and early prediction of patient survival is crucial for improving clinical outcomes. This project utilizes the Heart Failure Clinical Records dataset from the UCI Machine Learning Repository to develop predictive models using both supervised machine learning techniques. The primary objective is to predict patient survival status (DEATH_EVENT) based on clinical features such as Age, Anaemia, High blood pressure, Creatinine phosphokinase (CPK), Diabetes, Ejection fraction, Sex, Platelets, Serum creatinine, Serum sodium, Smoking, Time and Death event.

The methodology includes data preprocessing steps like handling missing values, feature scaling, and splitting the dataset into training and testing sets, etc., Several machine learning algorithms, including Logistic Regression, K Neighbors Classifier, Gaussian NB, Random Forest, Support Vector Machines, and Decision Tree Classifier, are applied to develop predictive models. The models are evaluated using performance metrics such as accuracy, precision, recall, F1-score, ROC-AUC, etc...

The analysis reveals that the Random Forest Classifier outperforms other models in terms of accuracy. Additionally, feature importance analysis identifies key factors contributing to survival predictions. The study concludes that machine learning models can significantly aid in clinical decision-making by providing accurate predictions and insights into risk factors, paving the way for personalized healthcare approaches.

1. INTRODUCTION

1.1.Background:

Heart failure (HF) is a chronic and progressive condition in which the heart is unable to pump blood effectively, leading to inadequate oxygen supply to the body's tissues and organs. It is one of the leading causes of hospitalization and death worldwide, with increasing incidence rates, particularly among older adults. Heart failure can arise from a variety of underlying conditions, including coronary artery disease, hypertension, diabetes, and previous heart attacks. It is characterized by symptoms such as shortness of breath, fatigue, fluid retention, and reduced exercise tolerance.

The management of heart failure is complex and involves medical interventions aimed at controlling symptoms, improving quality of life, and reducing the risk of complications. Early detection and accurate risk prediction are crucial in managing heart failure, as timely treatment can improve patient outcomes and prevent hospitalization. However, predicting the course of heart failure remains challenging, as it is influenced by numerous clinical factors that vary among individuals.

1.2.Significance in Healthcare:

Heart failure is not only a significant cause of death but also places a heavy burden on healthcare systems due to the high costs associated with hospitalization, long-term care, and medications. The economic strain caused by heart failure is exacerbated by frequent readmissions, particularly among elderly patients. Moreover, heart failure patients often experience poor quality of life, which makes it a major focus of healthcare improvement initiatives.

Given the complexities of heart failure management, there is a growing interest in developing predictive tools that can assist healthcare providers in making more informed clinical decisions. Accurate prediction models could help identify high-risk patients early on, enabling personalized interventions that could delay disease progression and reduce the need for emergency care.

1.3.Importance of Predictive Modelling:

Predictive modelling is an essential tool in modern healthcare. By using historical data and advanced machine learning techniques, predictive models can forecast patient outcomes based on their clinical attributes. In the case of heart failure, predictive models can help assess the likelihood of adverse events, such as hospitalization or death, enabling healthcare providers to tailor treatments more effectively.

Machine learning, in particular, offers great potential in heart failure prediction due to its ability to analyze complex and high-dimensional data. By leveraging various clinical variables, machine learning algorithms can uncover hidden patterns and relationships in the data that may not be immediately apparent to clinicians. This project aims to develop and evaluate machine learning models to predict the survival status of heart failure patients based on their clinical records, contributing to better clinical decision-making and improved patient care outcomes.

1.4.Problem Statement:

Heart failure (HF) is a complex and heterogeneous condition, influenced by a variety of clinical, demographic, and lifestyle factors. Predicting the outcomes of heart failure, particularly survival status and the risk of adverse events such as hospitalization and death,

presents significant challenges. Traditional clinical approaches to heart failure prognosis rely heavily on physician expertise, historical data, and standard diagnostic tests. However, these methods often lack the accuracy and precision needed to make timely, individualized treatment decisions, especially given the variability of the disease progression among patients.

One of the primary challenges in predicting heart failure outcomes is the large number of interrelated factors that influence the condition. These include clinical metrics such as ejection fraction, serum levels of creatinine, and comorbidities like diabetes and hypertension. Each patient's condition is influenced by a combination of these factors, which makes it difficult to predict the likelihood of adverse outcomes without sophisticated analytical methods.

Additionally, the traditional models for heart failure prognosis often rely on expert opinion or clinical scoring systems that may not fully capture the complexities of the condition. These models can sometimes be subjective and fail to incorporate the full range of variables that influence heart failure outcomes. Moreover, they do not leverage the vast amounts of data generated by modern healthcare systems, which could provide deeper insights into the factors that contribute to disease progression and patient survival.

Given these challenges, there is a growing need for more robust, data-driven approaches to predicting heart failure outcomes. Machine learning (ML) provides an opportunity to improve prediction accuracy by analyzing large datasets and uncovering hidden patterns and relationships within the clinical records of heart failure patients. Unlike traditional methods, ML models can handle complex, nonlinear relationships and adjust to new data, offering the potential to make more accurate and timely predictions.

This project aims to develop machine learning models that predict heart failure outcomes, specifically the survival status of patients, based on clinical data. By employing supervised learning techniques, the project seeks to address the challenges of heart failure prognosis and improve decision-making in clinical practice, ultimately leading to better patient care and resource management.

2. OBJECTIVE

The primary objective of this project is to apply machine learning techniques to the Heart Failure Clinical Records dataset with the goal of improving the prediction and understanding of heart failure outcomes. The specific objectives of the project are as follows:

- **Evaluate and Compare Different Machine Learning Algorithms**

A key objective is to compare the performance of multiple machine learning algorithms such as Logistic Regression, K Neighbors Classifier, Gaussian NB, Random Forest, Support Vector Machines, and Decision Tree Classifier and evaluate their effectiveness in predicting heart failure outcomes. By assessing various models, the project aims to identify the most accurate and reliable algorithms for heart failure prognosis. The performance of these models will be evaluated using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to ensure their suitability for clinical decision-making.

- **Develop Supervised Learning Models for Outcome Prediction**

The project aims to develop and evaluate supervised learning models to predict the survival status of heart failure patients, based on clinical features. By utilizing algorithms such as Logistic Regression, K Neighbors Classifier, Gaussian NB, Random Forest, Support Vector Machines, and Decision Tree Classifier, the goal is to accurately predict whether a patient will experience a heart failure event (DEATH_EVENT) or survive. This predictive capability can be instrumental in supporting healthcare providers in making more informed and timely treatment decisions.

- **Identify Key Features Influencing Predictions**

Another objective is to conduct feature importance analysis to identify the key clinical variables that significantly influence the prediction of survival outcomes. By understanding which features have the most impact, healthcare providers can gain insights into the factors that contribute to heart failure risk, leading to more effective treatment strategies and patient management.

In summary, this project aims to enhance clinical decision-making by developing machine learning models that not only predict heart failure outcomes but also identify key factors that influence disease progression. This can ultimately lead to more personalized care, improved patient outcomes, and more efficient resource utilization in healthcare systems.

3. METHODOLOGY

The methodology section outlines the various steps and techniques used to analyze the Heart Failure Clinical Records dataset and build machine learning models for predicting heart failure outcomes. This section covers the tools and libraries used, the data preprocessing steps, exploratory data analysis (EDA), feature selection, and the machine learning models applied.

3.1.Tools and Libraries Used

The analysis and machine learning tasks in this project are implemented using Python and its associated libraries. These libraries are well-suited for data manipulation, statistical analysis, and machine learning tasks.

- **Pandas:** Used for data manipulation and preprocessing, such as loading the dataset, handling missing values, and splitting the data into features and target variables.
- **NumPy:** Utilized for numerical operations and array manipulation. It is particularly useful for performing statistical operations, such as mean, median, and standard deviation calculations.
- **Scikit-learn:** The primary machine learning library used in this project. It provides implementations of various machine learning algorithms (Logistic Regression, Random Forest, Support Vector Machines, etc.), tools for data splitting, and model evaluation metrics (accuracy, precision, recall, etc.).
- **Matplotlib:** A visualization library used for creating static, interactive, and animated plots, such as histograms, scatter plots, and line graphs, to understand data distributions and model performance.
- **Seaborn:** Built on top of Matplotlib, Seaborn provides advanced visualization capabilities, particularly useful for creating correlation heatmaps, box plots, and other statistical visualizations.

3.2.Data Preprocessing

Data preprocessing is a crucial step in any machine learning workflow. It ensures that the dataset is clean, well-formatted, and ready for analysis.

- **Handling Missing Values:** The first step in preprocessing is checking for any missing values in the dataset. Missing values can be handled in several ways:
  - **Imputation:** For numerical features, missing values are replaced with the mean or median. For categorical features, missing values may be replaced with the mode.
  - **Deletion:** If the missing values are minimal and do not significantly affect the dataset, rows containing missing data can be removed.
- **Normalization:** Min-Max scaling or Standardization (Z-score normalization) can be applied to transform the features to a uniform scale, typically within the range of 0 to 1 or a standard normal distribution (mean = 0, standard deviation = 1).
- **Splitting the Dataset:** The dataset is split into training and testing sets to evaluate model performance on unseen data. A typical split ratio is 80% for training and 20% for testing.

3.3.Exploratory Data Analysis (EDA)

EDA is an essential step to understand the structure of the dataset, identify potential issues, and gain insights into the relationships between variables. The following EDA techniques are used:

- **Data Summary**: Initial exploration involves viewing basic information about the dataset, such as the number of rows, columns, data types, and basic statistics like mean, median, and standard deviation.
- **Correlation Analysis:** Understanding the relationships between numerical variables is critical. A correlation matrix is used to examine which features are strongly correlated with the target variable. Highly correlated features are potential candidates for inclusion in the model.
- **Visualizing Distributions:** Histograms, box plots, and density plots are used to understand the distribution of each feature and check for skewness, outliers, or unusual patterns in the data.
- **Categorical Variable Analysis:** For categorical variables (e.g., sex, diabetes, smoking), bar plots and pie charts are created to examine the distribution of each class.

3.4.Machine Learning Models

Several machine learning models are used to predict the survival status of heart failure patients:

a. Logistic Regression

- Type: Linear Model
- Description: Logistic Regression is a statistical model used for binary classification tasks, where the goal is to predict the probability of a binary outcome (e.g., survival or death). It estimates the relationship between the dependent variable (target) and the independent features using a linear equation. However, instead of predicting a continuous value, it applies a logistic function (sigmoid) to ensure the output is between 0 and 1, representing a probability. If the probability is greater than 0.5, the model predicts one class; otherwise, it predicts the other class.

b. K-Neighbors Classifier (KNN)

- Type: Instance-based Learning (Lazy Learner)
- Description: The K-Nearest Neighbors (KNN) classifier is a simple, non-parametric algorithm that makes predictions based on the majority class of the nearest neighbors to a given data point. KNN works by calculating the distance (usually Euclidean distance) between the query point and all other points in the training dataset. The algorithm then assigns the class label that is most common among the ‘k’ closest points. The value of 'k' (the number of neighbors) is a hyperparameter that must be optimized. KNN is easy to understand but can be computationally expensive for large datasets and is sensitive to irrelevant or redundant features.

c. Gaussian Naive Bayes (Gaussian NB)

- Type: Probabilistic Model (Naive Bayes)
- Description: Gaussian Naive Bayes is a probabilistic classification model that assumes the features are independent given the target variable (this is the "naive" assumption). It is based on Bayes' Theorem, which calculates the probability of each class given the features and selects the class with the highest posterior probability. In the case of

Gaussian Naive Bayes, it assumes that the continuous features follow a normal (Gaussian) distribution. It is computationally efficient and performs well with small datasets or when the feature independence assumption holds, though it may struggle with highly correlated features.

d. Random Forest

- Type: Ensemble Learning (Bagging)
- Description: Random Forest is an ensemble learning method that combines multiple decision trees to improve model performance. Each tree in the forest is trained on a random subset of the training data, using a technique called bootstrapping. Additionally, at each split in the tree, only a random subset of features is considered. This helps reduce the variance and prevents overfitting, making Random Forest a robust and powerful model. The final prediction is made by averaging the predictions of all individual trees (for regression) or by majority voting (for classification). Random Forest works well with both linear and non-linear relationships and can handle a large number of features.

e. Support Vector Machines (SVM)

- Type: Discriminative Classifier
- Description: Support Vector Machines (SVM) are powerful classifiers that find the optimal hyperplane in a high-dimensional space to separate classes. The objective of SVM is to maximize the margin (the distance between the hyperplane and the closest data points, known as support vectors) to improve generalization. SVM works well in cases where the data is non-linearly separable by transforming the data into a higher-dimensional space using a kernel function (e.g., radial basis function, polynomial kernel). SVM can be sensitive to the choice of the kernel and hyperparameters, but it is effective in high-dimensional spaces and is particularly useful for classification tasks with clear margins of separation.

f. Decision Tree Classifier

- Type: Supervised Learning (Tree-based model)
- Description: A Decision Tree is a supervised machine learning algorithm that splits the data into subsets based on the feature that results in the greatest information gain (or reduction in impurity, such as Gini index or entropy) at each node. The tree continues splitting until a stopping criterion is met, such as a maximum depth or minimum number of samples in a leaf. Decision trees are easy to interpret and visualize, as they provide a clear path from the root to the leaves that can be followed to make predictions. However, they can easily overfit, especially with deep trees, and are sensitive to small changes in the data.

3.5.Model Evaluation

- **Accuracy**:
  - Measures the proportion of correct predictions (both true positives and true negatives) to the total predictions.
  - **Formula**: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

- **Precision**:
  - Measures the proportion of true positive predictions to the total predicted positives (important for minimizing false positives).
  - **Formula**: $Precision = \frac{TP}{TP+FP}$
- **Recall (Sensitivity)**:
  - Measures the proportion of true positive predictions to the total actual positives (important for minimizing false negatives).
  - **Formula**: $Recall = \frac{TP}{TP+FN}$
- **F1-Score**:
  - The harmonic mean of precision and recall, providing a balance between the two metrics.
  - **Formula**: $F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$
- **ROC-AUC Score**:
  - The Area Under the Receiver Operating Characteristic Curve (ROC-AUC) evaluates the trade-off between true positive rate (recall) and false positive rate (1 - specificity). Higher AUC values indicate better model performance. Evaluates the model's ability to distinguish between classes based on predicted probabilities. Higher values (closer to 1) indicate better performance.
- **Classification Report**:
  - Provides a detailed summary of precision, recall, F1-score, and support for each class.
- **Confusion Matrix**:
  - A table summarizing the performance of the model by showing counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

4. DATASET OVERVIEW

The dataset used in this project is the Heart Failure Clinical Records dataset which is available from the UCI Machine Learning Repository. The dataset was collected to help predict the likelihood of heart failure events (DEATH_EVENT) based on various clinical features from patients diagnosed with heart failure. This dataset serves as for applying machine learning techniques to predict patient outcomes and identify key factors contributing to survival.

Dataset link: https://archive.ics.uci.edu/dataset/519/heart+failure+clinical+records

The Heart Failure Clinical Records dataset is a valuable resource for studying clinical data related to heart failure. This dataset contains the medical records of 299 patients who had heart failure, collected during their follow-up period, where each patient profile has 13 clinical features.

Feature	Explanation	Measurement	Range
Age	Age of the patient	Years	[40,..., 95]
Anaemia	Decrease of red blood cells or hemoglobin	Boolean	0, 1
High blood pressure	If a patient has hypertension	Boolean	0, 1
Creatinine phosphokinase (CPK)	Level of the CPK enzyme in the blood	mcg/L	[23,..., 7861]
Diabetes	If the patient has diabetes	Boolean	0, 1
Ejection fraction	Percentage of blood leaving	Percentage	[14,..., 80]
Sex	Woman or man	Binary	0, 1
Platelets	Platelets in the blood	kiloplatelets/mL	[25.01,..., 850.00]
Serum creatinine	Level of creatinine in the blood	mg/dL	[0.50,..., 9.40]
Serum sodium	Level of sodium in the blood	mEq/L	[114,..., 148]
Smoking	If the patient smokes	Boolean	0, 1
Time	Follow-up period	Days	[4,...,285]
(Target) Death event	If the patient died during the follow-up period	Boolean	0, 1

Table 1: Data Description

4.1.FEATURE DESCRIPTION:

The dataset consists of 12 features that describe various clinical, demographic, and lifestyle factors that may influence heart failure outcomes. These features are as follows:

1. Age

- Type: Numeric

- Description: The age of the patient. Age is a critical factor in predicting heart failure outcomes, as older individuals tend to have a higher risk of adverse health events.

2. Anaemia

- Type: Binary (0 = No, 1 = Yes)

- Description: Whether the patient has anaemia, a condition characterized by low levels of red blood cells or hemoglobin, which can exacerbate heart failure symptoms.

3. High Blood Pressure (Hypertension)

- Type: Binary (0 = No, 1 = Yes)

- Description: Whether the patient has high blood pressure, a major risk factor for heart failure.

4. Creatinine Phosphokinase (CPK)

- Type: Numeric

- Description: The level of creatinine phosphokinase in the blood. High CPK levels can indicate heart muscle damage, which is critical in diagnosing heart failure severity.

5. Diabetes

- Type: Binary (0 = No, 1 = Yes)

- Description: Whether the patient has diabetes, which is a known comorbidity that can worsen heart failure outcomes due to its impact on cardiovascular health.

6. Ejection Fraction

- Type: Numeric (Percentage)

- Description: The percentage of blood pumped out of the heart with each contraction. A lower ejection fraction is an indicator of heart dysfunction and is an important predictor of heart failure severity.

7. Sex

- Type: Binary (0 = Female, 1 = Male)

- Description: The gender of the patient, which can affect the risk of heart failure due to gender-specific risk factors.

8. Platelets

- Type: Numeric

- Description: The level of platelets in the blood. Low platelet levels can be associated with various health conditions, including heart failure, as they may indicate underlying issues such as inflammation or poor blood circulation.

9. Serum Creatinine

- Type: Numeric

- Description: The level of creatinine in the blood. Elevated serum creatinine levels can indicate kidney dysfunction, which often correlates with worsening heart failure.

10. Serum Sodium

- Type: Numeric

- Description: The level of sodium in the blood. It is a key indicator of fluid balance in the body, which is critical for heart failure patients, who often experience fluid retention.

11. Smoking

- Type: Binary (0 = No, 1 = Yes)

- Description: Whether the patient is a smoker. Smoking is a major risk factor for cardiovascular diseases, including heart failure.

12. Time

- Type: Numeric

- Description: The number of days the patient has been under observation. This feature helps track the duration of the patient's health status, which can influence survival.

4.2.TARGET VARIABLE:

1. Death Event (DEATH_EVENT)

- Type: Binary (0 = Survived, 1 = Death)

- Description: The target variable indicating whether the patient survived (0) or died (1) from heart failure. This is the primary variable to predict in this project.

5. ANALYSIS AND RESULTS

5.1. ENVIRONMENT SETUP

To start any machine learning project, the first and most crucial step is to set up the working environment. This involves installing the necessary libraries and tools required to perform data manipulation, visualization, and model building.

Installing the Libraries

To install libraries, Python's package manager pip is used. Below are the commands for installing the required libraries:

```
pip install pandas numpy matplotlib seaborn scikit-learn
```

Code Explanation

a. Data Handling and Manipulation:

- **Pandas**: Provides easy-to-use data structures and data manipulation tools.
- **NumPy**: Offers support for handling large, multi-dimensional arrays and matrices, along with mathematical functions.

b. Data Visualization:

- **Matplotlib**: A plotting library for creating static, interactive, and animated visualizations.
- **Seaborn**: Built on top of Matplotlib, it provides high-level functions for creating visually appealing and informative statistical graphics.

c. Machine Learning:

- **Scikit-learn**: A powerful library for implementing machine learning algorithms, preprocessing data, and evaluating models.

5.2. IMPORTING REQUIRED LIBRARIES

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, classification_report, confusion_matrix, roc_curve
```

Code Explanation

a. **Numpy**
   Purpose: A data manipulation and analysis library. Provides efficient array operations and mathematical computations.
   Use Case: Calculations for features, handling numerical data, and generating random values.

b. **pandasImport:**
   Purpose: Handles tabular data with rows and columns, offering tools for data loading, cleaning, manipulation, and analysis.
   Use Case: Reading datasets from CSV files, handling missing values, and preparing data for machine learning.

c. **matplotlib.pyplot**
   Purpose: A plotting library for visualizing data.
   Use Case: Generating graphs such as line plots, scatter plots, and histograms to understand data distribution and model performance. Visualizing model performance, comparing accuracy, and analyzing data distribution.

d. **Seaborn**
   Purpose: Extends Matplotlib for creating informative and visually appealing statistical graphics.
   Use Case: Creating heatmaps, pair plots, and box plots to identify patterns and correlations.

e. **sklearn.model_selection**
   train_test_split: Splits the dataset into training and testing subsets.

Use Case: Ensures the model is trained on one part of the data and tested on another to evaluate its performance.
StratifiedKFold: Ensures that each fold of the cross-validation process maintains the same class proportion as the original dataset, which is particularly useful for imbalanced datasets.
Use Case: Provides a robust way to evaluate model performance while preserving the distribution of target labels in each fold. Commonly used in classification problems where maintaining class balance is crucial for accurate model evaluation
cross_val_score: Computes cross-validation scores to assess model performance.
Use Case: Ensures fair comparison of different classification algorithms by testing them on multiple data splits.

f. sklearn.linear_model
Import: from sklearn.linear_model import LogisticRegression
Purpose: Logistic Regression is a linear model used for binary and multi-class classification problems.
Use Case: Acts as a baseline algorithm for comparison.

g. sklearn.neighbors
Import: from sklearn.neighbors import KNeighborsClassifier
Purpose: k-Nearest Neighbors (k-NN) is a non-parametric algorithm that classifies data based on proximity to training samples.
Use Case: Useful for datasets where similarity or distance between points is critical.

h. sklearn.tree
Import: from sklearn.tree import DecisionTreeClassifier
Purpose: A tree-based model that splits the dataset into subsets based on feature values.
Use Case: Works well for datasets with non-linear relationships.

i. sklearn.naive_bayes
Import: from sklearn.naive_bayes import GaussianNB
Purpose: Implements the Gaussian Naive Bayes algorithm, assuming features are normally distributed.
Use Case: Efficient for text classification and datasets with categorical features.

j. sklearn.ensemble
Import: from sklearn.ensemble import RandomForestClassifier
Purpose: Implements a robust ensemble learning method that combines multiple decision trees to make predictions.
Use Case: Used for classification tasks with high accuracy and feature importance analysis.

k. sklearn.svm
Import: from sklearn.svm import SVC
Purpose: Support Vector Machine (SVM) is a powerful algorithm for both linear and non-linear classification.
Use Case: Handles high-dimensional datasets effectively and finds optimal decision boundaries.

l. sklearn.discriminant_analysis
Import: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

Purpose: A dimensionality reduction technique that projects data into a lower-dimensional space for classification.
Use Case: Enhances classification accuracy when working with high-dimensional data.

m. sklearn.metrics

Purpose: Contains various evaluation metrics to assess the performance of classification models.
Functions Imported:

- **accuracy_score:** Measures the proportion of correctly predicted instances.
- **precision_score:** Evaluates the precision of positive predictions (true positives / predicted positives).
- **recall_score:** Evaluates the ability to identify all relevant instances (true positives / actual positives).
- **f1_score:** Harmonic mean of precision and recall, useful for imbalanced datasets.
- **roc_auc_score:** Measures the area under the Receiver Operating Characteristic (ROC) curve.
- **classification_report:** Provides a summary of precision, recall, F1-score, and support for each class.
- **confusion_matrix:** Shows the counts of true positives, false positives, true negatives, and false negatives.
- **roc_curve:** Plots the true positive rate (TPR) against the false positive rate (FPR) for various thresholds.

5.3. LOADING THE DATASET

The next step is to load the dataset into Python environment. This is a crucial step because it provides the foundation for all subsequent data processing, analysis, and model building tasks. Here, use the pandas library, which is specifically designed for handling and analyzing structured data.

```
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')
print('Data:', data)
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75	0	582	0	20	1	265000	1.9	130	1	0	4	1
1	55	0	7861	0	38	0	263358	1.1	136	1	0	6	1
2	65	0	146	0	20	0	162000	1.3	129	1	1	7	1
3	50	1	111	0	20	0	210000	1.9	137	1	0	7	1
4	65	1	160	1	20	0	327000	2.7	116	0	0	8	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
294	62	0	61	1	38	1	155000	1.1	143	1	1	270	0
295	55	0	1820	0	38	0	270000	1.2	139	0	0	271	0
296	45	0	2060	1	60	0	742000	0.8	138	0	0	278	0
297	45	0	2413	0	38	0	140000	1.4	140	1	1	280	0
298	50	0	196	0	45	0	395000	1.6	136	1	1	285	0

Table 2. Dataset

Code Explanation

a. **pd.read_csv:**
   Function: Reads a CSV (Comma-Separated Values) file into a Pandas DataFrame.

b. **'heart_failure_clinical_records_dataset.csv':**
   The file path to dataset.

c. **Output:**
   Returns a DataFrame containing the data from the CSV file.

5.4. EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA) is a critical step in any machine learning project. This process helps to understand the dataset's structure, contents, and potential issues such as missing values or outliers. By exploring the data, identify patterns, relationships, and insights that inform data preprocessing and modelling strategies.

 # Display the first few rows of the dataset

print(data.head())

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75	0	582	0	20	1	265000	1.9	130	1	0	4	1
1	55	0	7861	0	38	0	263358	1.1	136	1	0	6	1
2	65	0	146	0	20	0	162000	1.3	129	1	1	7	1
3	50	1	111	0	20	0	210000	1.9	137	1	0	7	1
4	65	1	160	1	20	0	327000	2.7	116	0	0	8	1

Table 3. Data head

 # Display the last few rows of the dataset

print(data.tail())

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
294	62	0	61	1	38	1	155000	1.1	143	1	1	270	0
295	55	0	1820	0	38	0	270000	1.2	139	0	0	271	0
296	45	0	2060	1	60	0	742000	0.8	138	0	0	278	0
297	45	0	2413	0	38	0	140000	1.4	140	1	1	280	0
298	50	0	196	0	45	0	395000	1.6	136	1	1	285	0

Table 4. Data tail

 # Display an overview of the dataset (columns, data types, non-null counts)

print(data.info())

Data info:			
<class 'pandas.core.frame.DataFrame'>			
RangeIndex: 299 entries, 0 to 298			
Data columns (total 13 columns):			
#	Column	Non-Null Count	Dtype
0	age	299 non-null	float64
1	anaemia	299 non-null	int64
2	creatinine_phosphokinase	299 non-null	int64
3	diabetes	299 non-null	int64
4	ejection_fraction	299 non-null	int64
5	high_blood_pressure	299 non-null	int64
6	platelets	299 non-null	float64
7	serum_creatinine	299 non-null	float64
8	serum_sodium	299 non-null	int64
9	sex	299 non-null	int64
10	smoking	299 non-null	int64
11	time	299 non-null	int64
12	DEATH_EVENT	299 non-null	int64
dtypes: float64(3), int64(10)			
memory usage: 30.5 KB			

Table 5. Data info

 # Display summary statistics for numerical columns

print(data.describe())

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
coun	299	299	299	299	299	299	299	299	299	299	299	299	299
mea	61	0.43	581.8395	0.42	38.084	0.3512	263358	1.3939	136.6	0.6	0.32	130	0.321
std	12	0.5	970.2879	0.49	11.835	0.4781	97804.2	1.0345	4.412	0.5	0.47	78	0.468
min	40	0	23	0	14	0	25100	0.5	113	0	0	4	0
0.3	51	0	116.5	0	30	0	212500	0.9	134	0	0	73	0
0.5	60	0	250	0	38	0	262000	1.1	137	1	0	115	0
0.8	70	1	582	1	45	1	303500	1.4	140	1	1	203	1
max	95	1	7861	1	80	1	850000	9.4	148	1	1	285	1

Table 6. Data summary

 # Check for missing values in each column

print("Missing Values:\n", data.isnull().sum())

Missing values:	
age	0
anaemia	0
creatinine_phosphokinase	0
diabetes	0
ejection_fraction	0
high_blood_pressure	0
platelets	0
serum_creatinine	0
serum_sodium	0
sex	0
smoking	0
time	0
DEATH_EVENT	0
dtype: int64	

Table 7. Missing Values

```
# Compute and visualize correlation matrix
corrmat = data.corr()
# Create a heatmap for correlations
fig = plt.figure(figsize = (13, 13))
sns.heatmap(corrmat, vmax = .8, square = True, annot=True, fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

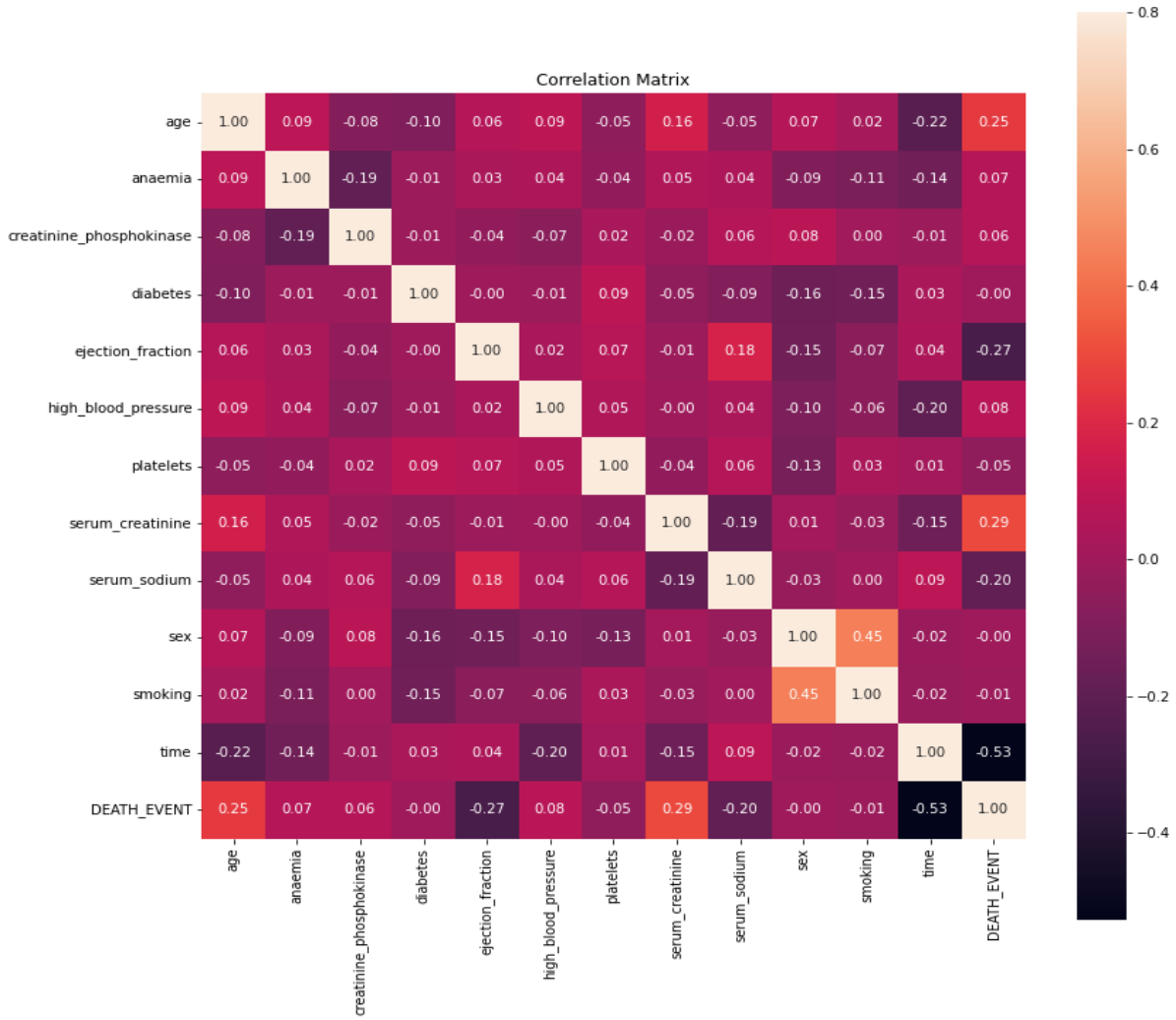

Figure 1. Correlation Matrix Heatmap

The correlation matrix visualizes the linear relationships between different variables in dataset.

- **DEATH_EVENT Correlations**:
  - The variable "time" has a strong negative correlation with "DEATH_EVENT" (-0.53), indicating that a shorter follow-up period is associated with higher mortality.
  - "Serum_creatinine" (0.29) and "age" (0.25) are positively correlated with "DEATH_EVENT," suggesting they are important risk factors for death.
  - "Ejection_fraction" (-0.27) is negatively correlated with "DEATH_EVENT," meaning lower ejection fraction is associated with higher mortality.

- **Inter-Variable Correlations**:
  - "Sex" and "smoking" have a strong positive correlation (0.45), which could suggest smoking is more prevalent in a particular gender within this dataset.
  - Other correlations are relatively weak, indicating low multicollinearity among most variables.
- **Implications for Modeling**:
  - Variables like "time," "serum_creatinine," "age," and "ejection_fraction" may be strong predictors for "DEATH_EVENT" and could be prioritized in predictive modeling.
  - The low correlations among most variables suggest the dataset doesn't suffer from severe multicollinearity, which is favorable for machine learning models.

```
# Plot histograms for numerical features
data.hist(figsize=(10, 8), bins=20)
plt.tight_layout()
plt.show()
```

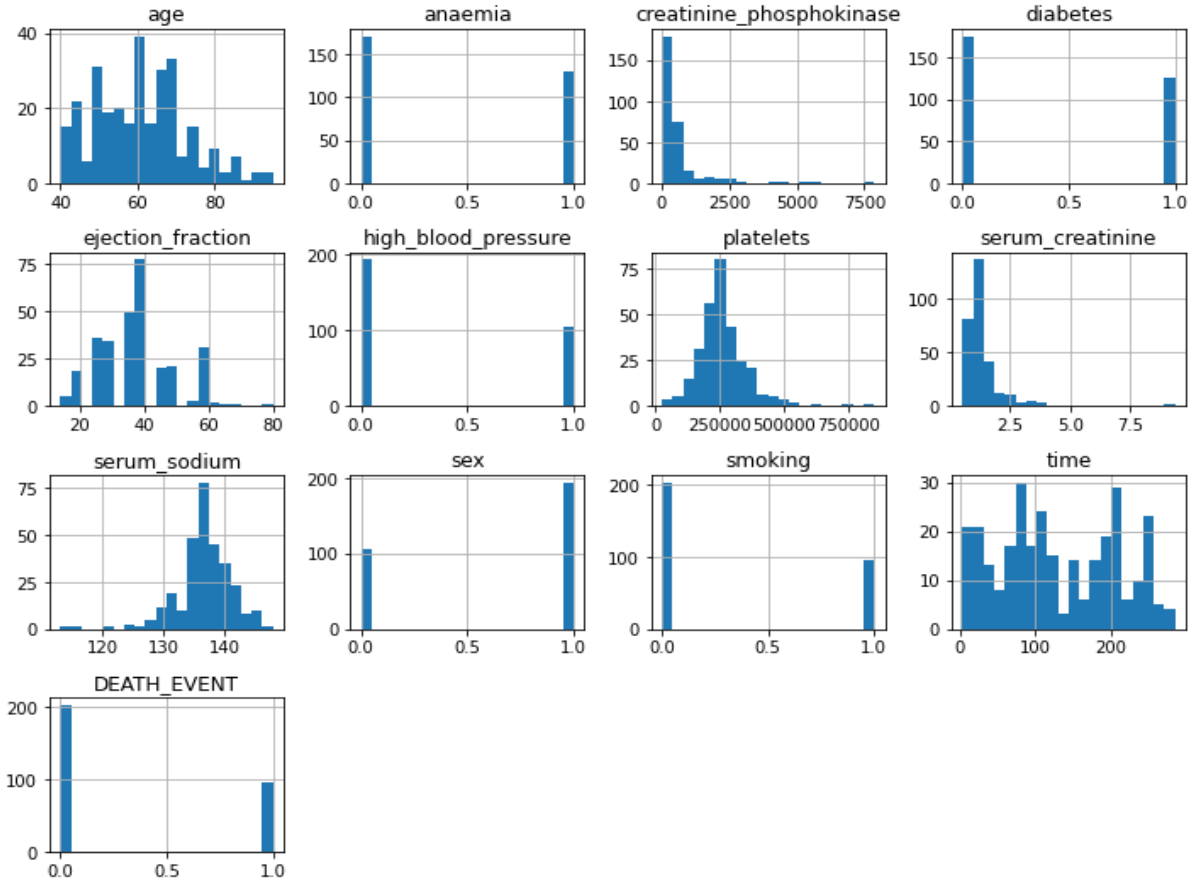

Figure 2. Histogram for numerical feature

This visualization represents the distribution of various features in the dataset.

- **Age**: The age distribution is skewed toward older individuals, with a higher frequency between 50 and 70 years.
- **Anaemia, Diabetes, High Blood Pressure, Sex, Smoking, and Death Event**: Most individuals do not have anaemia, diabetes, or high blood pressure. The "smoking" feature has more non-smokers than smokers. The "DEATH_EVENT" feature indicates fewer deaths (0) compared to survivals (1).
- **Creatinine Phosphokinase**: The distribution is highly right-skewed, indicating most individuals have low values, but a few have extremely high levels.
- **Ejection Fraction**: The values are fairly concentrated around 30–60%, which is expected in datasets related to heart failure patients.
- **Platelets**: The distribution appears roughly normal but with a long tail on the higher side.
- **Serum Creatinine**: Highly skewed to the right, with most values between 0.5 and 2.5, but outliers exist beyond 4.
- **Serum Sodium**: The distribution is normal and concentrated around 130–145, indicating typical physiological levels.
- **Time**: Time is distributed across a range, indicating diverse follow-up periods.

```
#Density Plots: Highlight the overall distribution shape
for col in data.select_dtypes(include=['int64', 'float64']).columns:
    sns.kdeplot(data[col], shade=True)
    plt.title(f" Plot for {col}")
    plt.show()
```

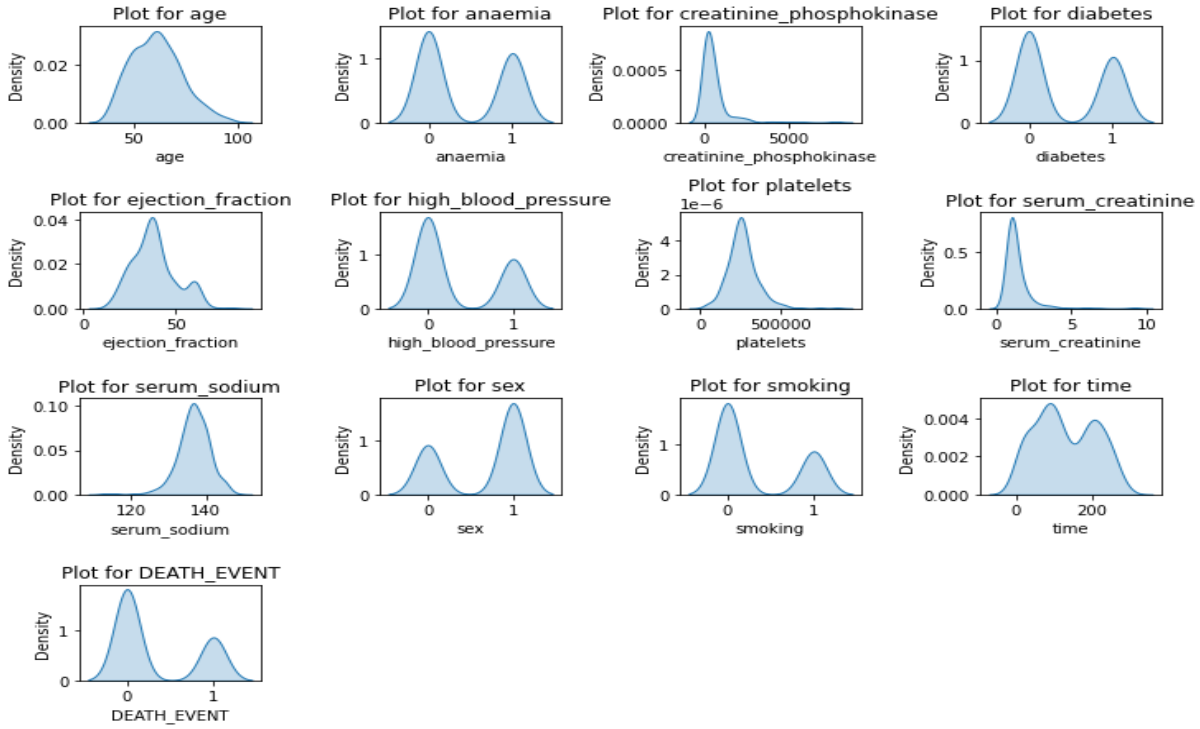

Figure 3. Density Plots: Highlight the overall distribution shape

This image displays kernel density plots (KDE) for the distribution of each variable in the dataset.

- **Age**: The age distribution shows a peak between 50 and 70 years, confirming most patients are middle-aged or older.
- **Anaemia, Diabetes, High Blood Pressure, Sex, Smoking, and Death Event**: These are binary variables, with peaks at 0 and 1. For "Death Event," the peak is higher at 0, indicating fewer deaths.
- **Creatinine Phosphokinase**: The density plot highlights a sharp peak near low values, with a long tail showing some extreme outliers.
- **Ejection Fraction**: A bimodal distribution with a peak near 30–40% and a smaller one at higher values, possibly indicating subgroups of patients with preserved and reduced ejection fractions.
- **Platelets**: The distribution is concentrated with a peak near the average value, with slight skewness.
- **Serum Creatinine**: Highly right-skewed, with most patients having values below 2.0 and a few outliers with much higher values.
- **Serum Sodium**: A normal distribution centered around 135–140, which aligns with expected physiological ranges.
- **Time**: Bimodal distribution indicating follow-up periods tend to cluster around two distinct time intervals.

```
# Boxplot for identifying outliers in numerical columns
for col in data.select_dtypes(include=['int64', 'float64']).columns:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=data[col])
    plt.title(f'Boxplot for {col}')
    plt.show()
```

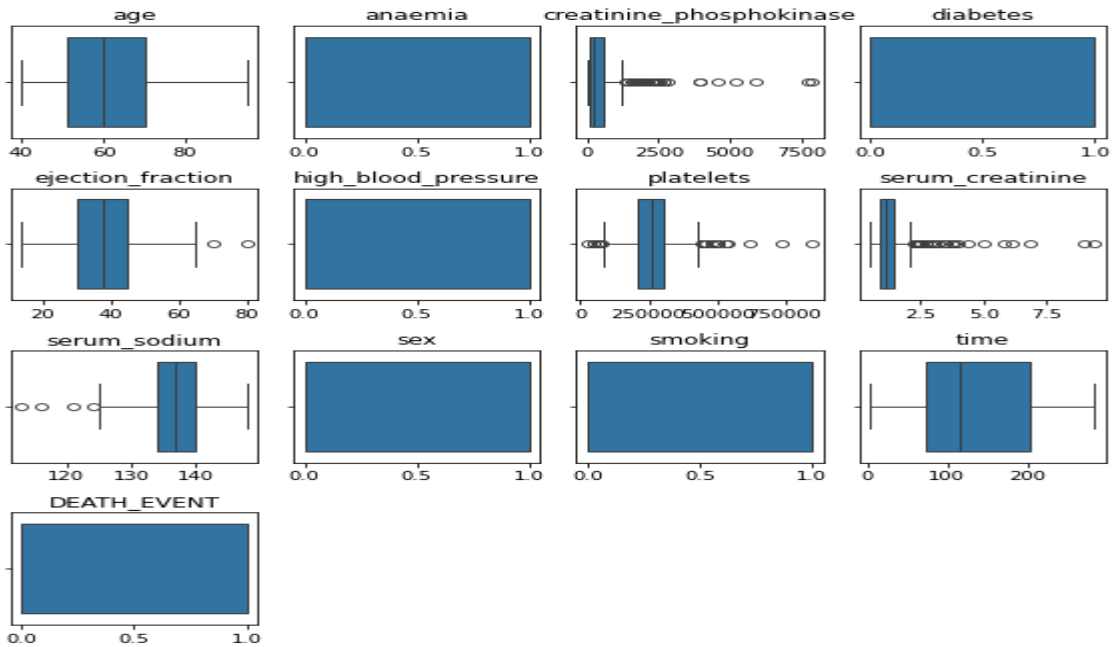

Figure 4. Boxplot for identifying outliers in numerical columns

The boxplots visualize the distribution of various features.

- **Age:** The median age is around 60, with the interquartile range (IQR) spanning from approximately 50 to 70. There are a few outliers above 90.
- **Anaemia, diabetes, high_blood_pressure, sex, smoking, and DEATH_EVENT:** These features are binary (0 or 1), as indicated by the boxplots showing values only at these two points.
- **Creatinine_phosphokinase:** This feature exhibits a wide range, with a median around 2000 and several outliers extending beyond 6000.
- **Ejection_fraction:** The median ejection fraction is around 40, with the IQR between roughly 30 and 50. There's at least one outlier above 70.
- **Platelets:** The median platelet count is around 400,000, with the IQR spanning from approximately 250,000 to 500,000. Several outliers exist above 600,000.
- **Serum_creatinine:** The median serum creatinine level is around 1.2, with the IQR between roughly 1 and 1.5. Outliers are present above 6.
- **Serum_sodium:** The median serum sodium level is approximately 137, with the IQR ranging from about 133 to 140. A few outliers are below 120.
- **Time:** The median time is around 150, with the IQR spanning from approximately 100 to 200. There are a few outliers above 250.

```
#Bar Plots: Visualize class distributions
categorical_columns = ['sex', 'anaemia', 'diabetes', 'high_blood_pressure', 'smoking', 'DEATH_EVENT']
for col in categorical_columns:
    sns.countplot(x=col, data=data, palette="muted")
    plt.title(f'Distribution of {col}')
    plt.show()
```

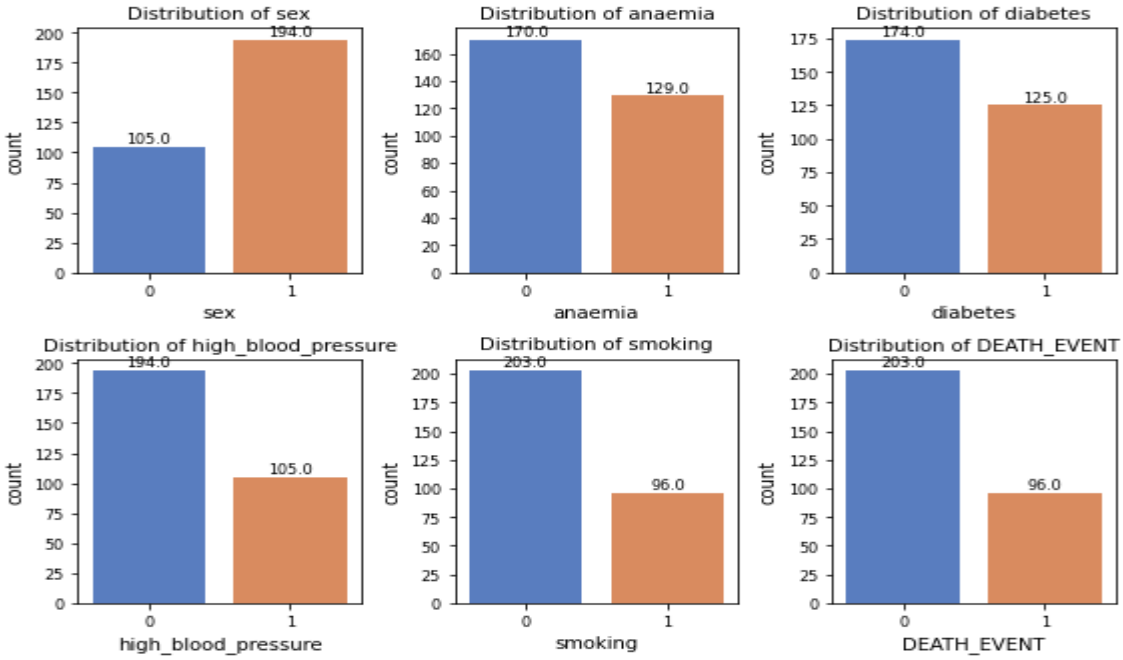

Figure 5. Bar Plots: Visualize class distributions

The count plots show the distribution of several binary features.

- **Sex:** There is a significant imbalance, with a higher representation of one sex (194) compared to the other (105). Without labels, it's impossible to determine which sex is more prevalent.
- **Anaemia:** More individuals do not have anaemia (170) than those who do (129).
- **Diabetes:** Similar to anaemia, more individuals do not have diabetes (174) than those who do (125).
- **High_blood_pressure:** A larger number of individuals do not have high blood pressure (194) compared to those who do (105).
- Smoking: More individuals do not smoke (203) than those who do (96).
- **DEATH_EVENT:** A greater number of individuals did not experience a death event (203) compared to those who did (96).

```
#Pie Charts: Show class proportions as percentages
categorical_columns = ['sex', 'anaemia', 'diabetes', 'high_blood_pressure', 'smoking', 'DEATH_EVENT']
for col in categorical_columns:
    data[col].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
    plt.title(f'{col} Distribution')
    plt.ylabel("")
    plt.show()
```

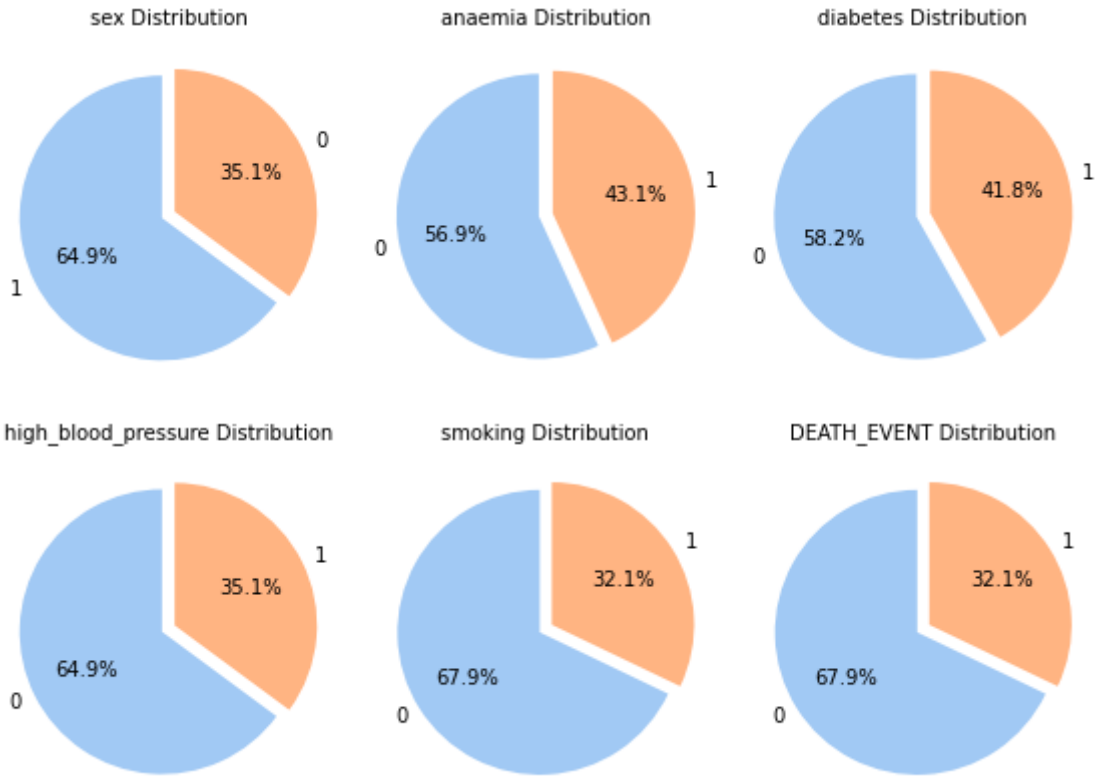

Figure 6. Pie Charts: Show class proportions as percentages

The pie charts illustrate the distribution of several binary variables.

- **Sex:** The distribution is approximately 65% (1) and 35% (0).
- **Anaemia**: The majority (57%) do not have anaemia (0), while 43% do (1).
- **Diabetes:** Similar to anaemia, a larger portion (58%) do not have diabetes (0), compared to 42% who do (1).
- **High_blood_pressure:** The distribution mirrors that of sex, with approximately 65% not having high blood pressure (0) and 35% having it (1).
- **Smoking:** About 68% do not smoke (0), while 32% do (1).
- **DEATH_EVENT:** The distribution is the same as smoking, with 68% not experiencing a death event (0) and 32% experiencing one (1).

Code Explanation

a. **data.head() & data.tail()**:

- Displays the first five rows of the dataset (data.head()) and last five rows of the dataset(data.tail()).
- Use this to get an initial look at the data and verify that it was loaded correctly.

b. **data.info()**:

- Provides metadata about the dataset, including:
  - Column names.
  - Data types (e.g., integer, float, object).
  - Number of non-null values in each column.
- Use this to identify missing data and understand the structure of your dataset.

c. **data.describe()**:

- Displays summary statistics for numerical columns, such as:
  - Count, mean, standard deviation, min, max, and quartiles (25%, 50%, 75%).
- Use this to identify data ranges, spot anomalies, and understand the distribution of numeric features.

d. **data.isnull().sum()**:

- Checks for missing values in each column.
- Use this to identify which columns have missing data and the extent of missingness.

e. **Correlation Matrix**:

- Identifies relationships between features:
  - Positive, negative, or no correlation.
- Use: Highlight potential multicollinearity or redundant features.
- **data.corr():**
  - Computes the correlation coefficients between all numerical features.
  - A value close to 1 indicates a strong positive correlation, -1 indicates a strong negative correlation, and 0 indicates no correlation.

f. **Histograms**:
   - Visualize distributions of numerical features:
     - Identify skewness, normality, or multimodal distributions.
   - Use: Detect features requiring transformation.
   - **data.hist():**
     - Automatically creates histograms for all numerical columns in the DataFrame.

g. **Density Plots:**
   - Visualize the overall distribution shape of numerical features to analyze data patterns.
   - **sns.kdeplot():**
     - Plots a **Kernel Density Estimate (KDE)** curve to show the probability density of a feature.
     - The curve smoothens the data points to identify trends such as skewness, peaks, and multimodality.
   - **shade=True:** Fills the area under the density curve to make it visually clearer.
   - **select_dtypes(include=['int64', 'float64'])**: Filters numerical columns (integer and float types) for plotting.

h. **Boxplots**:
   - Detect outliers in numerical features.
   - Use: Plan for outlier handling to improve model performance.
   - **data.select_dtypes(include=['int64', 'float64']):**
     - Filters numerical columns for boxplot visualization.

i. **Bar Plots:**
   - Visualize the class distributions of categorical variables to understand the frequency of each class.
   - **sns.countplot():**Plots the count of observations for each category in a given column.
   - **x=col:** Specifies the column to be plotted on the x-axis.
   - **palette="muted":** Applies a muted color palette for a clean and professional look.
   - **Categorical Columns:** Features like sex, anaemia, diabetes, high_blood_pressure, smoking, and DEATH_EVENT are considered because they contain discrete class values.

j. **Pie Charts:**
   - Represent the class proportions of categorical variables as percentages.
   - **value_counts():** Counts the number of occurrences for each class in the column.
   - **plot.pie():**Generates a pie chart from the counts of the classes.
   - **autopct='%1.1f%%':** Displays the percentage values for each slice with 1 decimal place.
   - **startangle=90:** Rotates the pie chart to start at a 90-degree angle for better alignment.
   - **ylabel(""):** Removes the default y-axis label for a cleaner chart.

5.5. DEFINING FEATURES AND TARGET VARIABLE

Separating the features and labels is a fundamental step in preparing dataset for a machine learning model. The **features** (X) are the input variables used to predict the output, and the **label** (y) is the target variable or the value to be predicted.

```
X = data.drop('DEATH_EVENT',axis=1) # Drop the target column to retain only features

y = data['DEATH_EVENT'] # Target variable

#Check Feature and Target Dimensions

print("Features shape (X):", X.shape)

print("Target shape (y):", y.shape)
```

Features shape (X):	(299, 12)
Target shape (y):	(299,)

Table 8. Feature and Target shape

Code Explanation

a. **X = data.drop('target_column', axis=1)**:
   - This removes the target column from the dataset to form the feature set X.
   - The axis=1 argument ensures the column is dropped (not a row).

b. **y = data['target_column']**:
   - This isolates the target column as the label y for predictions.
   - The target column should contain the output values (e.g., classes or regression targets).

c. **X.shape:**
   - X is the feature set containing all the input variables used to predict the target.
   - The .shape attribute of a DataFrame or NumPy array returns a tuple (n_samples, n_features):
     - **n_samples:** The number of rows (data points) in the dataset.
     - **n_features:** The number of columns (features) in the dataset.

d. **y.shape:**
   - y is the label or target variable, which contains the values the model is trying to predict.
   - The .shape for y returns (n_samples,):
     - **n_samples:** The number of rows in y should match the number of rows in X, ensuring every feature set has a corresponding label.

5.6. EVALUATE AND COMPARE DIFFERENT MACHINE LEARNING ALGORITHMS

This section of the code initializes multiple machine learning models, compares their performance, and stores the results for evaluation.

```
models = []   # Initialize list to store models

# Add different classifiers to the list with shorthand names

models.append(('LR', LogisticRegression(max_iter=100)))  # Logistic Regression

models.append(('KNN', KNeighborsClassifier()))              # K-Nearest Neighbors

models.append(('DT', DecisionTreeClassifier()))             # Decision Tree

models.append(('GNB', GaussianNB()))                       # Gaussian Naive Bayes

models.append(('RFC', RandomForestClassifier()))           # Random Forest

models.append(('SVC', SVC()))                               # Support Vector Classifier

models.append(('LDA', LinearDiscriminantAnalysis()))     # Linear Discriminant Analysis

# Initialize empty lists to store results, names, and average scores

results = [],

resultsf = []

names = []

avgs = [],

avgsf = []
```

Code Explanation

a. **models List**:
   - This list is used to store tuples of model names and their corresponding initialized classifiers.
   - Example: ('LR', LogisticRegression(max_iter=100)) pairs the shorthand name "LR" with the LogisticRegression model.

b. **Appended Classifiers**:
   - Logistic Regression (LR):
     - max_iter=100: Sets the maximum number of iterations for convergence.
   - K-Nearest Neighbors (KNN)
   - Decision Tree (DT)

- Gaussian Naive Bayes (GNB)
- Random Forest (RFC)
- Support Vector Classifier (SVC)
- Linear Discriminant Analysis (LDA)

c. **results and resultsf List**:
   - This list will store the accuracy and f1-score cross-validation scores for each model.

d. **names List**:
   - Stores the shorthand names of the models for easy reference.

e. **avgs and avgsf List**:
   - Used to store the average performance metrics (accuracy and f1-score) of each model.

5.6.1. Cross-Validation and Model Comparison

This performs **cross-validation** on a set of machine learning models to evaluate their accuracy and f1-score.

```
# Initialize StratifiedKFold Cross-Validation

kfold = StratifiedKFold (n_splits=10, shuffle=True, random_state=42)  # 10 folds, shuffled, and reproducible with random_state

# Loop through the models to evaluate each one

for name, model in models:

   # Perform cross-validation

   cv_results = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')

   cv_scores = cross_val_score(model, X, y, cv=kfold, scoring='f1')

      # Append results

   results.append(cv_results)      # Append all accuracy scores for the model

   resultsf.append(cv_scores)      # Append all f1 scores for the model

   names.append(name)              # Append the model name

   avgs.append(cv_results.mean())  # Append the mean accuracy score

   avgsf.append(cv_scores.mean())  # Append the mean f1 score

   print(name, 'accuracy', cv_results.round(2))

   print(name, 'f1 score', cv_scores.round(2))

   print(f"{name}: Mean Accuracy = {cv_results.mean():.2f}, Mean F1-Score = {cv_scores.mean():.2f}\n")
```

LR accuracy [0.8 0.8 0.77 0.87 0.9 0.73 0.93 0.8 0.87 0.79]
LR f1 score [0.67 0.73 0.59 0.82 0.84 0.56 0.88 0.62 0.75 0.57]
LR: Mean Accuracy = 0.83, Mean F1-Score = 0.70
KNN accuracy [0.7 0.53 0.63 0.67 0.53 0.53 0.7 0.73 0.6 0.59]
KNN f1 score [0.31 0.12 0.15 0.38 0.12 0.3 0.4 0.33 0.   0.14]
KNN: Mean Accuracy = 0.62, Mean F1-Score = 0.23
DT accuracy [0.8 0.87 0.73 0.9 0.77 0.7 0.83 0.67 0.73 0.79]
DT f1 score [0.7 0.78 0.67 0.87 0.67 0.58 0.62 0.5 0.56 0.67]
DT: Mean Accuracy = 0.78, Mean F1-Score = 0.66
GNB accuracy [0.8 0.77 0.73 0.77 0.77 0.77 0.87 0.8 0.8 0.79]
GNB f1 score [0.62 0.59 0.5 0.63 0.59 0.59 0.71 0.62 0.62 0.57]
GNB: Mean Accuracy = 0.79, Mean F1-Score = 0.61
RFC accuracy [0.83 0.97 0.8 0.87 0.93 0.8 0.83 0.8 0.73 0.86]
RFC f1 score [0.74 0.95 0.67 0.83 0.9 0.62 0.67 0.67 0.56 0.75]
RFC: Mean Accuracy = 0.84, Mean F1-Score = 0.73
SVC accuracy [0.67 0.67 0.67 0.67 0.67 0.67 0.7 0.7 0.7 0.69]
SVC f1 score [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
SVC: Mean Accuracy = 0.68, Mean F1-Score = 0.00
LDA accuracy [0.77 0.87 0.8 0.93 0.87 0.8 0.93 0.87 0.77 0.76]
LDA f1 score [0.59 0.82 0.62 0.91 0.8 0.7 0.88 0.75 0.59 0.53]
LDA: Mean Accuracy = 0.84, Mean F1-Score = 0.72

Table 9. Accuracy and f1-score for Stratified KFold

Code Explanation

a. StratifiedKFold Initialization:

- **n_splits=10**:
  - Divides the dataset into 10 equally-sized subsets (folds).
  - Each fold is used once as the validation set while the remaining 9 are used as the training set.
- **shuffle=True**:
  - Randomly shuffles the dataset before splitting into folds to ensure randomness.
- **random_state=42**:
  - Sets the seed for reproducibility, ensuring consistent splits across runs.

b. Cross-Validation for Each Model:

- **cross_val_score**:
  - Evaluates the model's performance using cross-validation.

- Parameters:
  - **model**: The current machine learning model.
  - **X and y**: The features and target variable.
  - **cv=kfold**: Specifies the KFold cross-validation object.
  - **scoring**: Metric to evaluate the model's performance.
    - '**accuracy**': Proportion of correctly classified samples.
    - '**f1**': Harmonic mean of precision and recall, useful for imbalanced datasets.

c. Results Storage:

- **results**: A list to store accuracy scores for all folds for each model.
- **resultsf**: A list to store F1 scores for all folds for each model.
- **names**: A list to store the names of the models.
- **avgs**: A list to store the mean accuracy score for each model.
- **avgsf**: A list to store the mean F1 score for each model.

d. Print Results:

- **name:** Prints the name of the model.
- **cv_results.round(2):** Displays the accuracy scores for each fold rounded to two decimal places.
- **cv_scores.round(2):** Displays the F1 scores for each fold rounded to two decimal places.
- **cv_results.mean():** Calculates the mean accuracy score across all folds.

5.6.2. Visualizing Model Performance

Generates a **boxplot** to compare the performance of different machine learning models based on their accuracy and f1-score.

```
fig = plt.figure(figsize=(13, 5))          # Create a figure with a specific size
fig.suptitle("Model Comparison")         # Add a title to the figure
ax = fig.add_subplot(111)         # Add a single subplot to the figure (1 row, 1 column, 1st plot)
plt.boxplot(results)              # Create a boxplot using the cross-validation results
ax.set_xticklabels(names)         # Set the x-axis labels to model names
ax.set_ylabel("Accuracy", fontsize=12)
plt.show()          # Display the plot
```

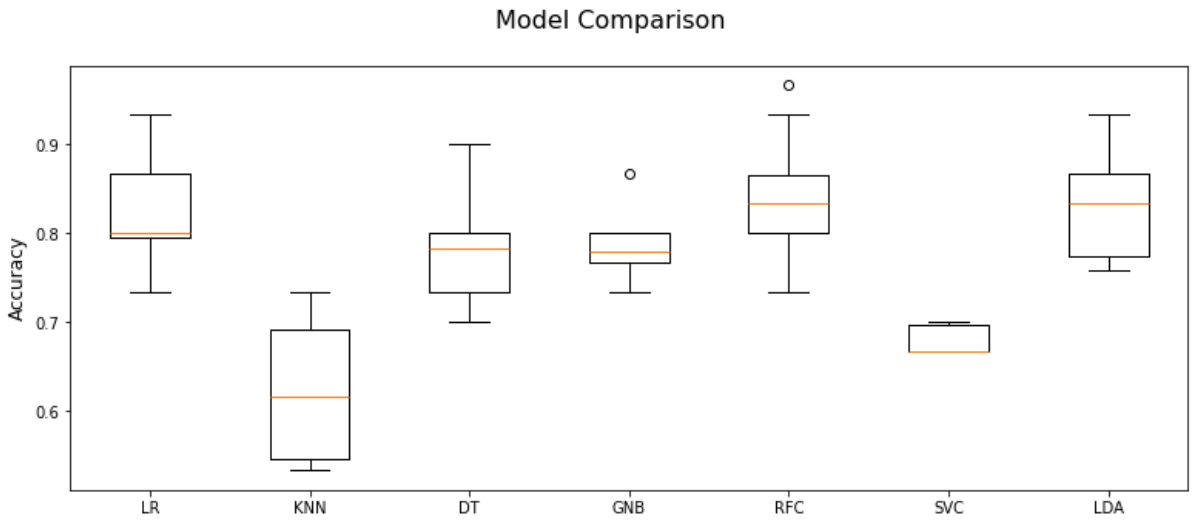

Figure 7. Model comparison -accuracy- boxplot

- Random Forest (RFC), and LDA are the most consistent and accurate models in this comparison. Their high and stable performance suggests they may generalize well to unseen data.
- SVC and KNN perform poorly, with SVC being the weakest. SVC's low accuracy may be due to hyperparameter choices or the characteristics of the dataset. KNN's wide range of performance indicates it is sensitive to experimental conditions or dataset splits.

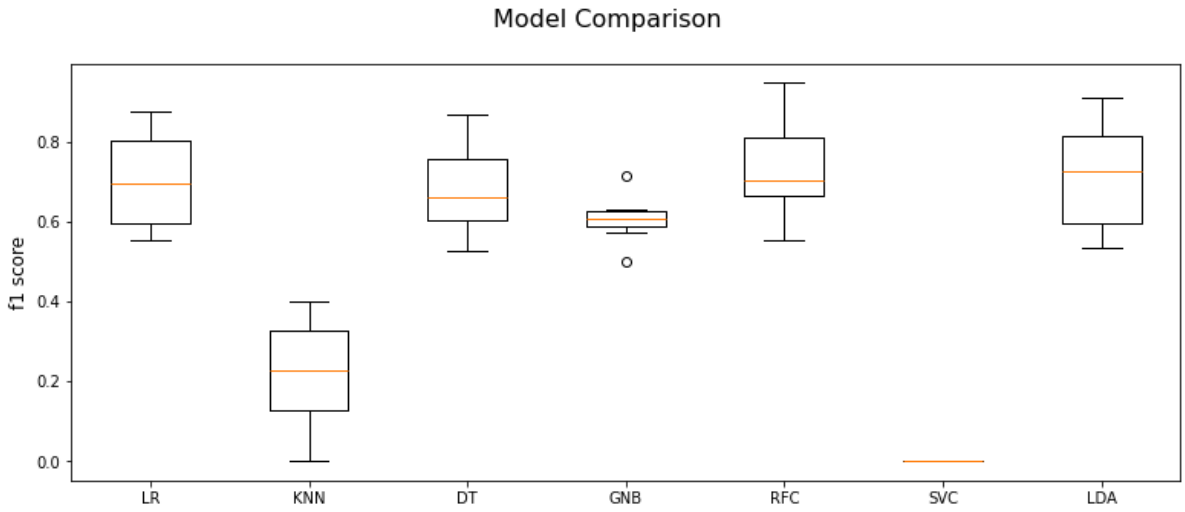

Figure 8. Model comparison -f1-score- boxplot

- Random Forest (RFC), LDA, and LR are the best models based on this comparison.
- RFC provides excellent predictive capacity and consistency.
- Similar outcomes with LDA make it another great choice, especially since it's computationally lighter.
- LR serves as a robust baseline with high and consistent F1-scores.

Code Explanation

a. **fig = plt.figure(figsize=(13, 5))**:

- Creates a new figure object with a width of 13 inches and height of 5 inches.

- The larger size ensures all model names and boxplots fit nicely in the visualization.

b. **fig.suptitle("Model Comparison")**:

- Adds a title ("Model Comparison") to the overall figure for context.

c. **ax = fig.add_subplot(111)**:

- Adds a subplot to the figure. The parameters (111) mean:
  - 1: Number of rows in the figure.
  - 1: Number of columns in the figure.
  - 1: Position of the subplot (1st and only one in this case).

d. **plt.boxplot(results)**:

- Plots a boxplot for each model using their cross-validation results (results list).
- Each boxplot represents the distribution of accuracy scores from Stratified K-Fold cross-validation.

e. **plt.boxplot(resultsf)**:

- Plots a boxplot for each model using their cross-validation score (resultsf list).
- Each boxplot represents the distribution of f1-scores from Stratified K-Fold cross-validation.

f. **ax.set_xticklabels(names)**:

- Sets the x-axis labels to the corresponding model names (names list).
- Without this, the x-axis would display default numeric ticks.

g. **plt.show()**:

- Displays the generated plot.

5.6.3. Identifying the Best Model

Creates a panda Data Frame to summarize the performance of various machine learning models and identifies the model with the highest average accuracy and f1-score.

```
# Create a DataFrame to summarize model names and their average accuracies and f1-score

df=pd.DataFrame([names,avgs,avgsf],index=['Name', 'Average Accuracy', 'Average f1score'],columns=np.arange(1,8)).T

# Display the DataFrame

print(df)
```

	Name	Average Accuracy	Average f1score
1	LR	0.825977	0.701945
2	KNN	0.621954	0.226273
3	DT	0.776092	0.676291
4	GNB	0.785977	0.6057
5	RFC	0.842874	0.73481
6	SVC	0.678966	0
7	LDA	0.835862	0.718708

Table 10. Model name, average accuracy and f1-score

```
# Find the model with the highest average accuracy
print(df[df['Average Accuracy'] == df["Average Accuracy"].max()])
```

	Name	Average Accuracy	Average f1score
5	RFC	0.842874	0.73481

Table 11. Best model based on highest average accuracy

```
# Find the model with the highest f1-score
print(df[df['Average f1score'] == df["Average f1score"].max()])
```

	Name	Average Accuracy	Average f1score
5	RFC	0.842874	0.73481

Table 12. Best model based on highest f1-score

Code Explanation

a. **df = pd.DataFrame([names, avgs], index=['Name', 'Average Accuracy'], columns=np.arange(1, 8)).T:**
- **names:** This list contains the names of the machine learning models (e.g., 'LR', 'KNN', etc.).
- **avgs:** This list contains the average accuracy scores for each model.
- **index=['Name', 'Average Accuracy']:** This defines two row labels: one for the model names and the other for the average accuracy scores.
- **columns=np.arange(1, 8):** This creates column labels as numbers from 1 to 7 (assuming there are 7 models).
- **.T:** This transposes the DataFrame so that the rows represent the model names and average accuracies.

b. **df**:
- This displays the DataFrame with the model names and their corresponding average accuracy values.

c. **df[df['Average Accuracy'] == df["Average Accuracy"].max()]**:
- This line finds the model with the highest average accuracy.
- **df["Average Accuracy"].max()**: Returns the maximum value of the average accuracy column.

- **df[df['Average Accuracy'] == *df["Average Accuracy"].max()*]**: Filters the DataFrame to return the row(s) where the average accuracy is equal to the maximum accuracy.

d. **df[df['Average f1score] == df["Average f1score"].max()]**:

- This line finds the model with the highest average f1score.
- **df["Average f1score"].max()**: Returns the maximum value of the average f1score column.
- **df[df['Average f1score] == *df["Average f1score"].max()*]**: Filters the DataFrame to return the row(s) where the average f1score is equal to the maximum f1score.

5.7. RANDOM FOREST CLASSIFIER: DETAILED ANALYSIS

i. Import Libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report, confusion_matrix, roc_curve
```

The first step in any machine learning project is to import the necessary libraries. Each library serves a specific purpose and is essential for different parts of the workflow.

ii. Load and Explore the Dataset

```
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')

# Overview of the dataset print(data.head()) # First few rows

print(data.info()) # Column types and non-null counts

print(data.describe()) # Summary statistics

# Check for missing values

print("Missing values per column:\n", data.isnull().sum())
```

The second step involves loading the dataset into a pandas DataFrame and performing an initial exploration to understand the structure, types of data, and potential issues.

iii. Separate features and target

```
X = data.drop("DEATH_EVENT", axis=1)

y = data["DEATH_EVENT"]
```

The dataset is divided into features (X) and the target variable (y). This separation is essential for training machine learning models, where features are used to make predictions about the target variable.

iv. Split the Dataset into Training and Testing Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Print shapes of the splits

print("\nX_train Shape: ",X_train.shape)

print("\nX_test Shape: ",X_test.shape)

print("\ny_train Shape: ",y_train.shape)

print("\ny_test Shape: ",y_test.shape)
```

X_train Shape:	(239, 12)
X_test Shape:	(60, 12)
y_train Shape:	(239,)
y_test Shape:	(60,)

Table 13. Shape after train-test split

Splitting the dataset ensures that the model is trained on one part of the data (training set) and evaluated on another (testing set). This prevents overfitting and helps to assess how the model generalizes to unseen data.

Code Explanation

a. **train_test_split**:

- Divides the data into:
  - **Training set (X_train, y_train)**: Used to train the model.
  - **Testing set (X_test, y_test)**: Used to evaluate the model's performance.

b. **Parameters**:

- **X and y**: Features and target.
- **test_size=0.2**: 20% of the data is allocated for testing, and the remaining 80% is used for training.
- **random_state=42**: Ensures reproducibility by setting a fixed seed for the random number generator.

c. **Shapes**:

- X_train: The feature matrix for training data.
- X_test: The feature matrix for testing data.
- y_train: The target labels for training data.
- y_test: The target labels for testing data.

d. **Shape Outputs**:

- Printing the shapes helps verify that the data has been split correctly.

v. Train the Random Forest Classifier

model = RandomForestClassifier(random_state=42)

model.fit(X_train, y_train)

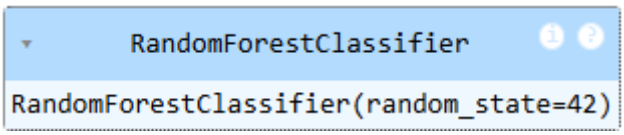

Table 14. Model selected

Training the model involves fitting the chosen machine learning algorithm to the training data. Here, we use the **Random Forest Classifier**, which is a robust ensemble learning method that combines multiple decision trees to improve accuracy and reduce overfitting.

Code Explanation

a. **Parameters**:

- **random_state=42**: Ensures that the results are reproducible by fixing the random seed.
- Other hyperparameters (like n_estimators, max_depth, etc.) can be tuned for better performance.

b. **fit(X_train, y_train)**:

- Trains the model using the training features (X_train) and their corresponding labels (y_train).

vi. Make Predictions

 # Predict on the testing set

y_pred = model.predict(X_test)

 # Predict probabilities for ROC curve

y_proba = model.predict_proba(X_test)[:, 1]

Once the model is trained, it is used to predict the outcomes for the test data (X_test). Additionally, the probability predictions are obtained to evaluate the model's performance using metrics like the ROC curve and AUC score.

Code Explanation

a. **model.predict(X_test)**:

- Predicts the class labels for the testing data.
- Output: y_pred, a list of predicted labels (0 or 1, depending on the dataset).

b. **model.predict_proba(X_test)**:

- Returns the predicted probabilities for each class.

- Example: For binary classification, it outputs a 2D array where each row contains two probabilities (for class 0 and class 1).
- **[:, 1]**: Extracts the probabilities of the positive class (class 1) for ROC curve analysis.

c. **Probabilities**:

- Useful for evaluating model performance using metrics that require probability scores, such as the **ROC curve** and **AUC score**.

vii. Evaluate the Model

```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: ', accuracy)
precision = precision_score(y_test, y_pred)
print('Precision: ', precision)
recall = recall_score(y_test, y_pred)
print('Recall: ', recall)
f1score = f1_score(y_test, y_pred)
print('F1 score: ',f1score)
roc_auc = roc_auc_score(y_test, y_proba)
print('ROC-AUC Score: ',roc_auc)
class_report = classification_report(y_test, y_pred)
print('Classification Report: \n', class_report)
confusionmatrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix: \n', confusionmatrix)
```

```
Accuracy:  0.8333333333333334
Precision:  0.8
Recall:  0.631578947368421
F1 score:  0.7058823529411765
ROC-AUC Score:  0.8915275994865212
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.93      0.88        41
           1       0.80      0.63      0.71        19

    accuracy                           0.83        60
   macro avg       0.82      0.78      0.79        60
weighted avg       0.83      0.83      0.83        60

Confusion Matrix:
 [[38  3]
 [ 7 12]]
```

Table 15. Model evaluation results

viii. visualizing confusion matrix

```
LABELS = ['Survive', 'Death']

plt.figure(figsize =(10, 10))

sns.heatmap(confusionmatrix, xticklabels = LABELS, yticklabels = LABELS, annot = True, fmt ="d");

plt.title("Confusion matrix")

plt.ylabel('Actualclass')

plt.xlabel('Predicted class')
```

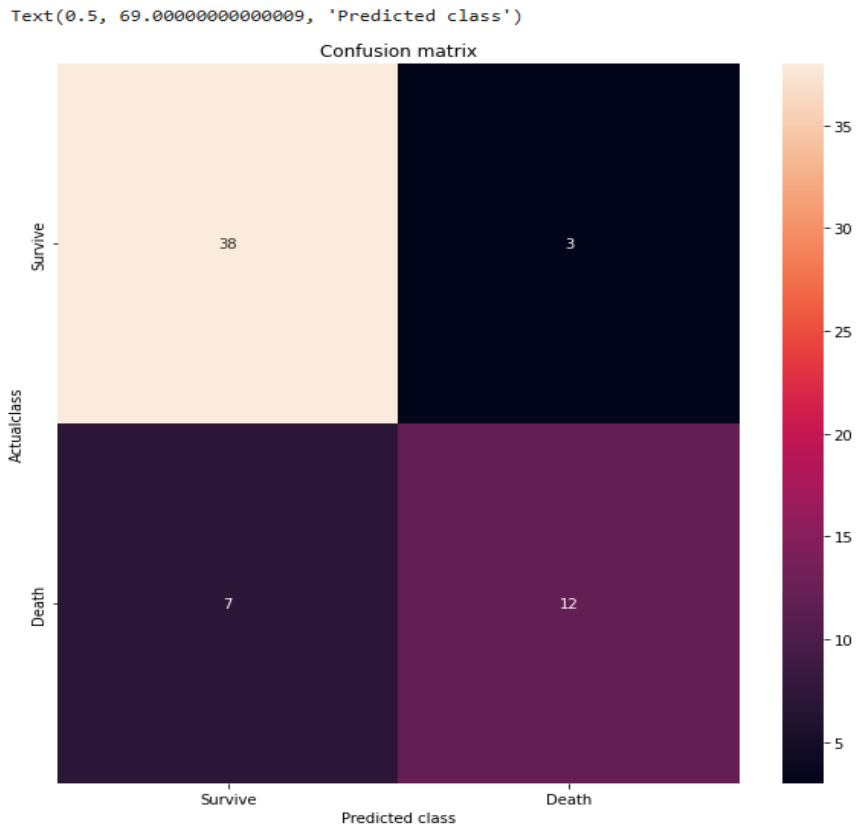

Figure 9. Confusion matrix heatmap

Visualizing the confusion matrix helps in understanding the model's performance at a glance.

- The confusion matrix shows the performance of a binary classification model predicting survival (Survive/Death).
- The model correctly predicted 38 survival cases and 12 death cases. However, it incorrectly predicted 7 survival cases as deaths and 3 death cases as survivals.
- The diagonal elements (38 and 12) represent true positives and true negatives, respectively.
- The off-diagonal elements (7 and 3) represent false negatives and false positives, respectively.
- The matrix suggests a relatively good performance, with a higher number of correct predictions than incorrect ones

Code Explanation

a. **LABELS**: Provides names for the actual and predicted classes (e.g., Survive and Death).

b. **sns.heatmap**: Visualizes the confusion matrix using a heatmap.

- **Parameters**:
  - **xticklabels and yticklabels**: Set the labels for the x and y axes.
  - **annot=True**: Displays the counts in each cell of the heatmap.
  - **fmt="d"**: Ensures integer formatting for the annotations.
  - **cmap='Blues'**: Specifies the color palette for the heatmap.

c. **Titles and Labels**: The title, x-axis, and y-axis labels make the visualization clear.

ix. Plot the ROC Curve

```
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], "--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

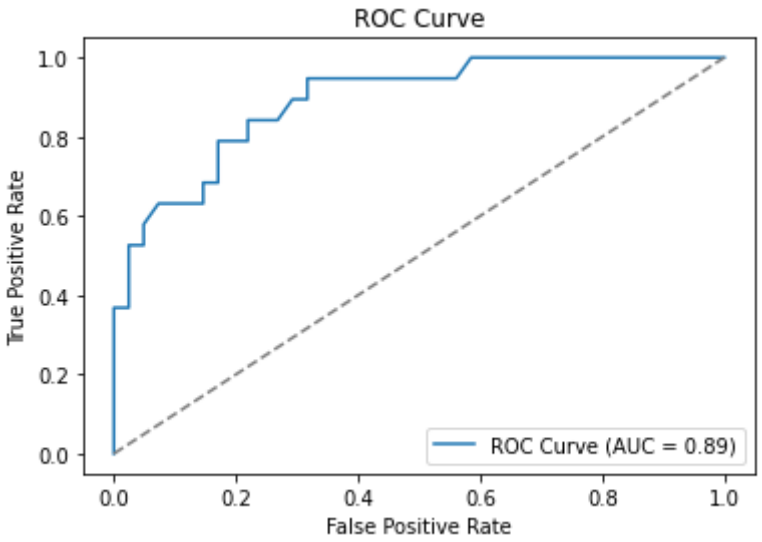

Figure 10. ROC Curve

The ROC (Receiver Operating Characteristic) curve visualizes the trade-off between sensitivity (True Positive Rate) and specificity (False Positive Rate) at various classification thresholds. The Area Under the Curve (AUC) quantifies the model's ability to distinguish between classes.

Code Explanation

a. **roc_curve(y_test, y_proba)**:

- Calculates the False Positive Rate (FPR) and True Positive Rate (TPR) for varying thresholds using the actual and predicted probabilities.
- Outputs:
  - fpr: Array of false positive rates.
  - tpr: Array of true positive rates.
  - _: Thresholds (not plotted but computed for the curve).

b. **Plot Elements**:

- **Main ROC Curve**: Plots fpr vs. tpr.
- **Reference Line**: Represents random guessing, plotted as a diagonal line from (0, 0) to (1, 1).
- **AUC**: Displayed in the legend to quantify performance. Higher values (closer to 1) indicate better performance.

c. **Styling**:

- Gridlines and clear labels ensure readability.
- The line width (lw) makes the curve visually distinct.

x. Feature importance

```
#Feature importance bar plot
feature_importances = pd.DataFrame({
    "Feature": X.columns,
    "Importance": model.feature_importances_
}).sort_values(by="Importance", ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x="Importance", y="Feature", data=feature_importances, palette="viridis", hue="Feature", dodge=False)
plt.title("Feature Importances")
plt.show()
```

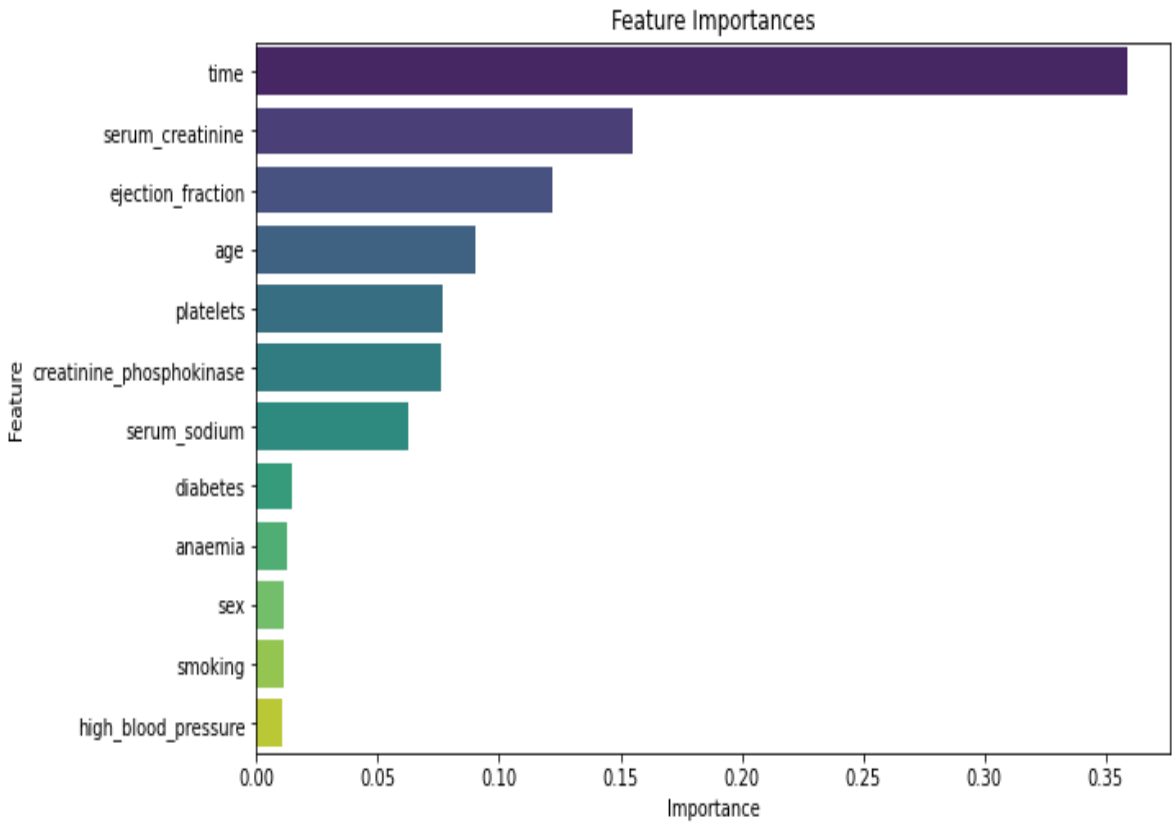

Figure 11. Feature importance bar plot

The bar chart shows the feature importance for a model.

- "Time" is the most important feature, followed by "serum creatinine," then "ejection fraction."
- The least important features are "high blood pressure," "smoking," and "sex".
- The features in the middle of the importance range include blood-related metrics like platelets, creatinine phosphokinase, and serum sodium, as well as medical conditions like diabetes and anemia.

Code Explanation

a. **model.feature_importances_**: Returns an array of importance scores for each feature. Features with higher scores are more influential in predicting the target variable.
b. **Data Preparation**: Combines feature names (X.columns) and their respective importances into a DataFrame. Sorted in descending order for better visualization.
c. **Visualization**:
   - **Barplot**: Depicts the importance of each feature.
   - **Palette**: The viridis palette ensures visually appealing and distinct color gradients.
d. **Labels and Titles**: The chart includes clear titles and axis labels to explain what the chart represents.

xi. Predicting for Specific Values

```
test1 =  pd.DataFrame([[75, 0, 582, 0, 20, 1, 265000, 1.9, 130, 1, 0, 4]], columns=X.columns)

test2 = pd.DataFrame([[53,0,63,1,60,0,368000,0.8,135,1,0,22]], columns=X.columns)

print(model.predict(test1))

print(model.predict(test2))
```

[1]
[O]

Table 16. Prediction result

Once the model is trained and evaluated, we can use it to make predictions on new data points. Here, we create two test examples (test1 and test2) and use the trained Random Forest model to predict their outcomes.

Code Explanation

a. **test1 and test2**:

- These are new observations formatted as DataFrames with the same structure as the training features (X).
- The column names match the features of the training set, ensuring compatibility with the model.

b. **model.predict()**:

- Predicts the target class (DEATH_EVENT) for the given test inputs.
- Outputs a NumPy array containing the predicted class (0 for "Survive" and 1 for "Death").

c. **Predicted Results**:

- The predictions for test1 and test2 are displayed using print.

6. CONCLUSION

This project aimed to predict heart failure events (DEATH_EVENT) using the Heart Failure Clinical Records dataset from the UCI repository. The primary goal was to develop and evaluate supervised machine learning models for clinical decision-making. The analysis included comparing several classification algorithms and identifying the most effective model for predicting patient outcomes. Key findings are summarized below:

- **Explore the Data (Exploratory Data Analysis):**

  The exploratory data analysis provides valuable insights into the dataset, helping to identify key trends, distributions, and relationships among variables that are critical in understanding heart failure. Variables such as age, ejection fraction, and serum creatinine emerge as significant contributors to the likelihood of heart failure. Older individuals and patients with lower ejection fraction or higher serum creatinine levels tend to exhibit a higher risk of heart failure, indicating their potential as predictive features. The analysis also reveals the presence of class imbalance in the target variable, suggesting that more patients belong to the "non-heart-failure" group compared to the "heart-failure" group.

- **Model Comparison:**

  Among the models tested, the Random Forest Classifier emerged as the most effective, achieving the highest accuracy (~84%) and f1-score (~74%). These results highlight the strength of ensemble learning methods in handling both linear and non-linear relationships.

  Logistic Regression, and Linear Discriminant Analysis also performed well, but they did not surpass the Random Forest model in overall predictive performance.

  Models such as Support Vector Classifier, and K-Nearest Neighbors were comparatively less effective, likely due to the dataset's characteristics and the models' sensitivity to feature scaling and correlations.

- **Insights from random forest classifier model:**

  The classification model achieves an accuracy of 83.33%, with a precision of 80% and an F1-score of 70.59%, indicating a solid overall performance. It demonstrates strong results for Class 0, with high recall (93%) and F1-score (88%), but struggles with Class 1, where recall drops to 63%. The ROC-AUC score of 89.15% reflects the model’s ability to distinguish between classes effectively, though the confusion matrix reveals difficulty in capturing all positive instances for Class 1.

  The high recall score (~93%) of the Random Forest model ensures a lower likelihood of false negatives, which is critical for identifying high-risk patients.

- **Feature Importance:**

  Feature importance analysis revealed that clinical variables such as ejection fraction, serum creatinine, and age were the most significant predictors of heart failure survival.

  These findings align with medical literature, emphasizing the importance of heart functionality (ejection fraction) and kidney performance (serum creatinine) in determining patient outcomes.

- **Challenges and Limitations:**

  The dataset consists of 299 samples, limiting the generalizability of the findings. Future work should validate the model on larger, more diverse datasets.

  The imbalance in the target variable (more survivors than deaths) could have influenced the models' performance, despite using techniques like stratified cross-validation.

In conclusion, the study demonstrates that machine learning models, particularly Random Forest, can effectively predict heart failure events and provide actionable insights for personalized healthcare. By identifying key risk factors, these models pave the way for proactive patient management and resource optimization in clinical settings.