NITTE EDUCATION TRUST

Department of Electronics and Communication Engineering

Nitte Education Trust

Nitte Meenakshi Institute of Technology

An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka
Accredited by NBA (Tier-1) and NAAC A+ Grade
Affiliated to Visvesvaraya Technological University, Belagavi-590018.
Post Box No. 6429, Yelahanka, Bengaluru-560064, Karnataka, India.

HEART DISEASE PREDICTION USING MACHINE LEARNING CLASSIFICATION MODELS

AML LA-1 [Mini-project]

4TH SEM 'A' SECTION

By

1.ANEESH G.B. 1NT22EC016
2.AUCHITYA JAIN 1NT22EC029

Under the Guidance of
Dr. Vishwanath V
Assistant professor, ECE Dept.
NMIT

Abstract: World Health Organization has estimated 12 million deaths occur worldwide, every year due to Heart diseases. Half the deaths all over are due to cardio vascular diseases. The early prognosis of cardiovascular diseases can aid in making decisions on lifestyle changes in high risk patients and in turn reduce the complications. Each individual has different values for Blood pressure, cholesterol and pulse rate. But according to medically proven results the normal values of Blood pressure is 120/90, Cholesterol is 100-129 mg/dL ,Pulse rate is 72, Fasting Blood Sugar level is 100 mg/dL ,Heart rate is 60-100 bpm, ECG is normal, Width of major vessels is 25 mm (1 inch) in the aorta to only 8 µm in the capillaries. This paper gives the survey about different classification techniques used for predicting the risk level of each person based on age, gender, Blood pressure, cholesterol, pulse rate.

This research intends to pinpoint the most relevant/risk factors of heart disease as well as predict the overall risk using 9 models([Logistic Regression, KNN, NB, SVM, Random Forest, Decision Tree).

This is a multivariate type of dataset which means providing or involving a variety of separate mathematical or statistical variables, multivariate numerical data analysis. It is composed of 14 attributes which are age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, oldpeak — ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels and Thalassemia. This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this dataset is to predict based on the given attributes of a patient that whether that particular person has heart disease or not and other is the experimental task to diagnose and find out various insights from this dataset which could help in understanding the problem more.

I. INTRODUCTION

In this project, we delve into a dataset encapsulating various health metrics from **heart patients**, including age, blood pressure, heart rate, and more. Our goal is to develop a predictive model capable of accurately identifying individuals with heart disease. Given the grave implications of missing a positive diagnosis, our primary emphasis is on ensuring that the model identifies all potential patients, making recall for the positive class a crucial metric.

The main aims of this research are as follows:

1. To make a machine learning model that can accurately classify patients as having the hepatitis C virus or not based on relevant medical information. The objective is to create a model that can correctly recognise Hepatitis C in patients.
2. Increased diagnostic accuracy: Compared to current diagnostic methods, the objective is to improve the accuracy of Hepatitis C detection. This may need testing with different machine learning algorithms and approaches to improve precision and recall rates.
3. Identify significant risk factors: The goal is to study the data and identify the key hepatitis C risk factors. This could involve feature selection or feature importance analysis to determine the components in the prediction model that have the most impact.
4. Giving advice to medical experts is goal of this project. The advice will be based on the findings of the machine learning model. For people who are atmost risk for Hepatitis C, this may suggest strategies for early detection, preventative measures, or specific medications.

This project objectives include:

- **Explore the Dataset**: Uncover patterns, distributions, and relationships within the data.
- **Conduct Extensive Exploratory Data Analysis (EDA)**: Dive deep into bivariate relationships against the target.
- **Preprocessing Steps**:
  - Remove irrelevant features
  - Address missing values
  - Treat outliers
  - Encode categorical variables
  - Transform skewed features to achieve normal-like distributions
- **Model Building**:
  - Establish pipelines for models that require scaling
  - Implement and tune classification models including KNN, SVM, Decision Trees, and Random Forest
  - Emphasize achieving high recall for class 1, ensuring comprehensive identification of heart patients
- **Evaluate and Compare Model Performance**: Utilize precision, recall, and F1-score to gauge models' effectiveness.

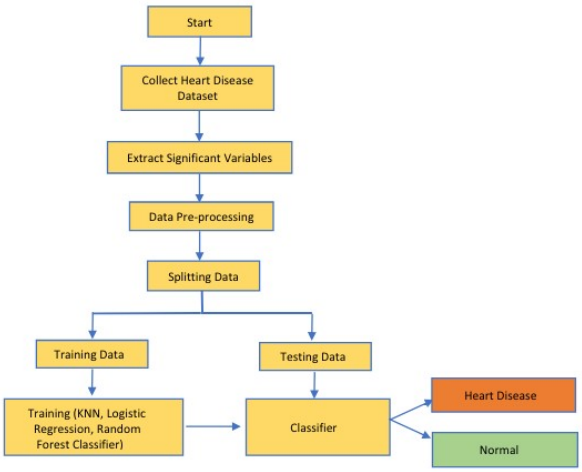


Figure 1. Proposed Model

Data Description:

This dataset is taken from the UCI machine learning website. This dataset contains medical information on patients and the diagnosis results of whether the patient has heart disease.

- Age - Patient Age in years
- Sex - Gender of Patient (0 - Male, 1 - Female)
- cp - Chest Pain type,0 = typical angina, 1 = atypical angina, 2 = non-anginal pain, 3 = asymptomatic
- trestbps - Rest Blood Pressure in mm Hg
- chol - Serum cholestoral (in mg/dl)
- fbs - Fasting blood sugar > 120 mg/dl
  0 = false, 1 = true
- restecg - Resting electrocardiographic results,
  0 = normal, 1 = having ST-T wave abnormality, 2 =

showing probable or definite left ventricular hypertrophy by Estes' criteria

- thalach - Maximum heart rate achieved
- exang - Exercise induced angina
  0 = no, 1 = yes
- oldpeak - ST depression induced by exercise relative to rest
- slope - The slope of the peak exercise ST segment,
  0 = upsloping, 1 = flat, 2 = downsloping
- ca - Number of major vessels (0-4) colored by flourosopy
- thal - Thalassemia, 3 = normal, 6 = fixed defect, 7 = reversable defect
- Target - Target column
  0 = not have heart disease 1 = have heart disease

2. METHODOLOGY

STEP 1: Import 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
from scipy.stats import boxcox
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

- **import numpy as np**: Imports the numpy module and aliases it as np. NumPy is a library for numerical computing in Python, often used for handling arrays and matrices efficiently.
- **import pandas as pd:** imports the pandas module and aliases it as pd. It provides fast, flexible, and expensive data structures designed to make working with relational or labeled data both easy and intuitive.
- **import matplotlib.pyplot as plt:** It is a state-based interface to matplotlib. It provides an implicit, MATLAB-like, way of plotting. It also opens figures on your screen, and acts as the figure GUI manager.
- **import seaborn as sns:** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **from scipy.stats import boxcox:** Box-Cox transformtion is a statistical technique that transforms the target variable, so that it resembles a normal distribution.
- **from sklearn.preprocessing import StandardScaler:** It standardize features by removing the mean and scaling to unit variance.
- **from sklearn.model_selection import GridSearchCV :** GridSearchCV is the process of performing hyperparameter tuning in order to determine the optimal values for a given model.
- **from sklearn.model_selection import StratifiedKFold:** It is used in dealing with classification problem of imbalance class distribution. It ensures that each fold of dataset has the same proportion of observation with a guven target variable.
- **from sklearn.metrics import classification_report, accuracy_score:** Classification-report module is used to generate three common metrics Precision, Recall, F1-Score. Accuracy_score method compares the predicted value with the true values
- **from sklearn.neighbors import KNeighborsClassifier:** The K-Nearest Neighbors (KNN) algorithm is a non-parametric supervised machine learning method employed to tackle classification and regression problems.
- **from sklearn.svm import SVC:** Supervised classification algorithm. It seeks to find the hyperplane that best separates the data points into different classes
- **from sklearn.tree import DecisionTreeClassifier:** It is a tree-like structure where each internal node tests on attribute, each branch corresponds to attribute value and each leaf node represents the final decision or prediction.
- **from sklearn.ensemble import RandomForestClassifier:** It is a method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time
- **from sklearn.linear_model import LogisticRegression:** supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not.
- **from sklearn.naive_bayes import GaussianNB:** classification technique used in machine learning based on a probabilistic approach and Gaussian distribution
- **from sklearn.pipeline import Pipeline: A**llows to sequentially apply a list of transformers to preprocess the data and, if desired, conclude the sequence with a final predictor for predictive modeling.
- **from sklearn.model_selection import train_test_split:** Split arrays or matrices into random train and test subsets.

STEP 2: Explore Data

Read Dataset:

```
# Read dataset
df = pd.read_csv('heart.csv')
df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

Concise Summary of Dataset:

```
# Display a concise summary of the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Inferences:

i. Number of Entries: The dataset consists of 303 entries, ranging from index 0 to 302.

ii. Columns: There are 14 columns in the dataset corresponding to various attributes of the patients and results of tests.
iii. Data Types: Most of the columns (13 out of 14) are of the int64 data type. Only the oldpeak column is of the float64 data type.
iv. Missing Values: There don't appear to be any missing values in the dataset as each column has 303 non-null entries.

Identify and Convert continuous features to Object datatype:
We can see that 9 columns (sex, cp, fbs, restecg, exang, slope, ca, thal, and target) are indeed numerical in terms of data type, but categorical in terms of their semantics. These features should be converted to string (object) data type for proper analysis and interpretation.

```
# Define the continuous features
continuous_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

# Identify the features to be converted to object data type
features_to_convert = [feature for feature in df.columns if feature not in continuous_features]

# Convert the identified features to object data type
df[features_to_convert] = df[features_to_convert].astype('object')

df.dtypes
```

```
age          int64
sex         object
cp          object
trestbps     int64
chol         int64
fbs         object
restecg     object
thalach      int64
exang       object
oldpeak    float64
slope       object
ca          object
thal        object
target      object
dtype: object
```

Statistics for Numerical Variables:

```
# Get the summary statistics for numerical variables
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	303.0	54.366337	9.082101	29.0	47.5	55.0	61.0	77.0
trestbps	303.0	131.623762	17.538143	94.0	120.0	130.0	140.0	200.0
chol	303.0	246.264026	51.830751	126.0	211.0	240.0	274.5	564.0
thalach	303.0	149.646865	22.905161	71.0	133.5	153.0	166.0	202.0
oldpeak	303.0	1.039604	1.161075	0.0	0.0	0.8	1.6	6.2

Inferences:
i. age: The average age of the patients is approximately 54.4 years, with the youngest being 29 and the oldest 77 years.
ii. trestbps: The average resting blood pressure is about 131.62 mm Hg, ranging from 94 to 200 mm Hg.
iii. chol: The average cholesterol level is approximately 246.26 mg/dl, with a minimum of 126 and a maximum of 564 mg/dl.
iv. thalach: The average maximum heart rate achieved is around 149.65, with a range from 71 to 202.
v. oldpeak: The average ST depression induced by exercise relative to rest is about 1.04, with values ranging from 0 to 6.2.

Statistics for Categorical Variables:

```
# Get the summary statistics for categorical variables
df.describe(include='object')
```

	sex	cp	fbs	restecg	exang	slope	ca	thal	target
count	303	303	303	303	303	303	303	303	303
unique	2	4	2	3	2	3	5	4	2
top	1	0	0	1	0	2	0	2	1
freq	207	143	258	152	204	142	175	166	165

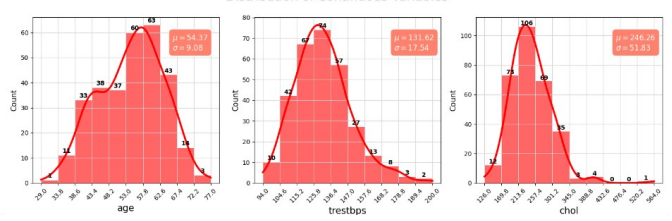
Inferences:
i. sex: There are two unique values, with males (denoted as 0) being the most frequent category, occurring 207 times out of 303 entries.
ii. cp: Four unique types of chest pain are present. The most common type is "0", occurring 143 times.
iii. fbs: There are two categories, and the most frequent one is "0" (indicating fasting blood sugar less than 120 mg/dl), which appears 258 times.
iv. restecg: Three unique results are present. The most common result is "1", appearing 152 times.
v. exang: There are two unique values. The most frequent value is "0" (indicating no exercise-induced angina), which is observed 204 times.
vi. slope: Three unique slopes are present. The most frequent slope type is "2", which occurs 142 times.
vii. ca: There are five unique values for the number of major vessels colored by fluoroscopy, with "0" being the most frequent, occurring 175 times.
viii. thal: Four unique results are available. The most common type is "2" (indicating a reversible defect), observed 166 times.
ix. target: Two unique values indicate the presence or absence of heart disease. The value "1" (indicating the presence of heart disease) is the most frequent, observed in 165 entries.

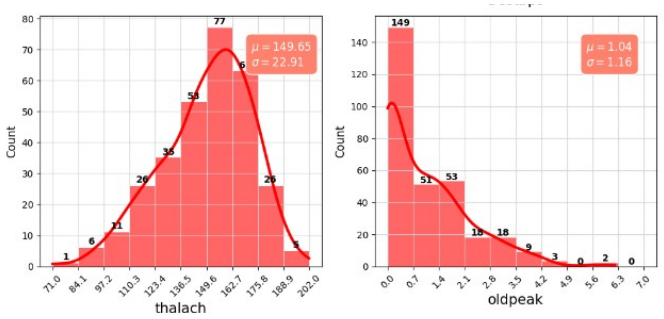
STEP 4: Exploratory Data Analysis (EDA)

Univariate Analysis: Focused on one feature at a time to understand its distribution and range.

- **For continuous data:** We employ histograms to gain insight into the distribution of each feature. This allows us to understand the central tendency, spread, and shape of the dataset's distribution.

```
# Filter out continuous features for the univariate analysis
df_continuous = df[continuous_features]

# Set up the subplot
fig, ax = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

# Loop to plot histograms for each continuous feature
for i, col in enumerate(df_continuous.columns):
    x = i // 3
    y = i % 3
    values, bin_edges = np.histogram(df_continuous[col],
                                     range=(np.floor(df_continuous[col].min()), np.ceil(df_continuous[col].max())))

    graph = sns.histplot(data=df_continuous, x=col, bins=bin_edges, kde=True, ax=ax[x, y],
                         edgecolor='none', color='red', alpha=0.6, line_kws={'lw': 3})
    ax[x, y].set_xlabel(col, fontsize=15)
    ax[x, y].set_ylabel('Count', fontsize=12)
    ax[x, y].set_xticks(np.round(bin_edges, 1))
    ax[x, y].set_xticklabels(ax[x, y].get_xticks(), rotation=45)
    ax[x, y].grid(color='lightgrey')

    for j, p in enumerate(graph.patches):
        ax[x, y].annotate('{}'.format(p.get_height()), (p.get_x() + p.get_width() / 2, p.get_height() + 1),
                          ha='center', fontsize=10, fontweight="bold")

    textstr = '\n'.join((
        r'$\mu=%.2f$' % df_continuous[col].mean(),
        r'$\sigma=%.2f$' % df_continuous[col].std()
    ))
    ax[x, y].text(0.75, 0.9, textstr, transform=ax[x, y].transAxes, fontsize=12, verticalalignment='top',
                  color='white', bbox=dict(boxstyle='round', facecolor='#ff826e', edgecolor='white', pad=0.5))

ax[1,2].axis('off')
plt.suptitle('Distribution of Continuous Variables', fontsize=20)
plt.tight_layout()
plt.subplots_adjust(top=0.92)
plt.show()
```





Inferences:

i. Age (age): The distribution is somewhat uniform, but there's a peak around the late 50s. The mean age is approximately 54.37 years with a standard deviation of 9.08 years.

ii. Resting Blood Pressure (trestbps): The resting blood pressure for most individuals is concentrated around 120-140 mm Hg, with a mean of approximately 131.62 mm Hg and a standard deviation of 17.54 mm Hg.

iii. Serum Cholesterol (chol): Most individuals have cholesterol levels between 200 and 300 mg/dl. The mean cholesterol level is around 246.26 mg/dl with a standard deviation of 51.83 mg/dl.

iv. Maximum Heart Rate Achieved (thalach): The majority of the individuals achieve a heart rate between 140 and 170 bpm during a stress test. The mean heart rate achieved is approximately 149.65 bpm with a standard deviation of 22.91 bpm.

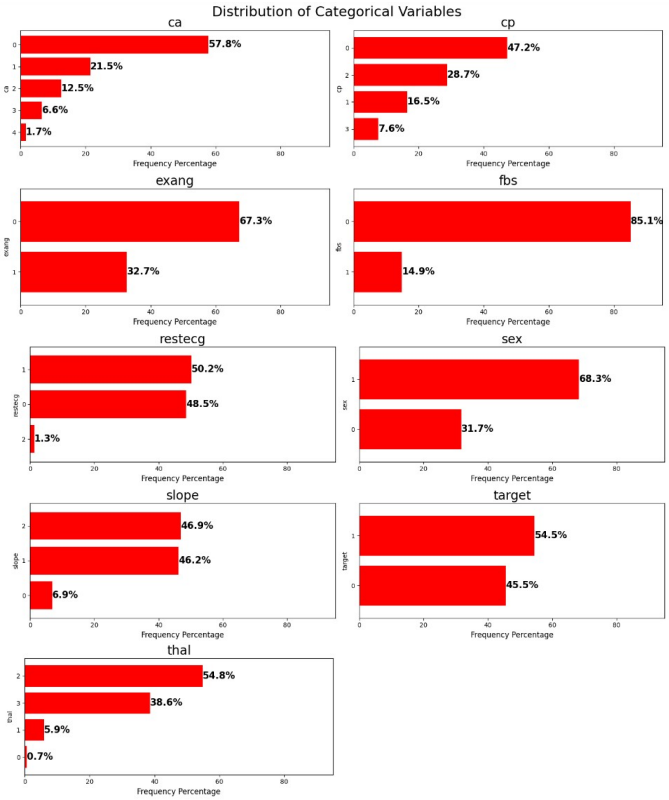
v. ST Depression Induced by Exercise (oldpeak): Most of the values are concentrated towards 0, indicating that many individuals did not experience significant ST depression during exercise. The mean ST depression value is 1.04 with a standard deviation of 1.16.

Upon reviewing the histograms of the continuous features and cross-referencing them with the provided feature descriptions, everything appears consistent and within expected ranges.

- **For categorical data:** Bar plots are utilized to visualize the frequency of each category. This provides a clear representation of the prominence of each category within the respective feature.

```
# Filter out categorical features for the univariate analysis
categorical_features = df.columns.difference(continuous_features)
df_categorical = df[categorical_features]
```

```
# Set up the subplot for a 4x2 layout
fig, ax = plt.subplots(nrows=5, ncols=2, figsize=(15, 18))

# Loop to plot bar charts for each categorical feature in the 4x2 layout
for i, col in enumerate(categorical_features):
    row = i // 2
    col_idx = i % 2

    # Calculate frequency percentages
    value_counts = df[col].value_counts(normalize=True).mul(100).sort_values()

    # Plot bar chart
    value_counts.plot(kind='barh', ax=ax[row, col_idx], width=0.8, color='red')

    # Add frequency percentages to the bars
    for index, value in enumerate(value_counts):
        ax[row, col_idx].text(value, index, str(round(value, 1)) + '%', fontsize=15, weight='bold', va='center')

    ax[row, col_idx].set_xlim([0, 95])
    ax[row, col_idx].set_xlabel('Frequency Percentage', fontsize=12)
    ax[row, col_idx].set_title(f'{col}', fontsize=20)

ax[4,1].axis('off')
plt.suptitle('Distribution of Categorical Variables', fontsize=22)
plt.tight_layout()
plt.subplots_adjust(top=0.95)
plt.show()
```



Inferences:

i. Gender (sex): The dataset is predominantly female, constituting a significant majority.

ii. Type of Chest Pain (cp): The dataset shows varied chest pain types among patients. Type 0 (Typical angina) seems to be the most prevalent, but an exact distribution among the types can be inferred from the bar plots.

iii. Fasting Blood Sugar (fbs): A significant majority of the patients have their fasting blood sugar level below 120 mg/dl, indicating that high blood sugar is not a common condition in this dataset.

iv. Resting Electrocardiographic Results (restecg):The results show varied resting electrocardiographic outcomes, with certain types being more common than others. The exact distribution can be gauged from the plots.

v. Exercise-Induced Angina (exang): A majority of the patients do not experience exercise-induced angina, suggesting that it might not be a common symptom among the patients in this dataset.

vi. Slope of the Peak Exercise ST Segment (slope): The dataset shows different slopes of the peak exercise ST segment. A specific type might be more common, and its distribution can be inferred from the bar plots.

vii. Number of Major Vessels Colored by Fluoroscopy (ca): Most patients have fewer major vessels colored by fluoroscopy, with '0' being the most frequent.

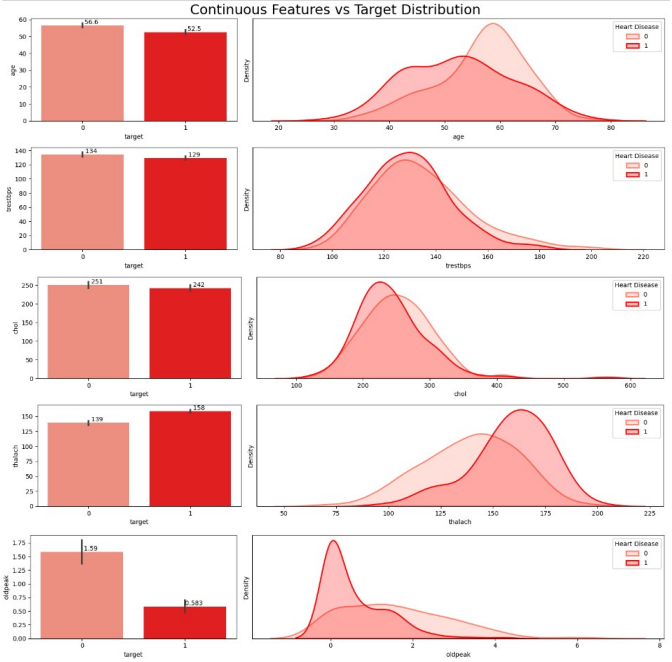
viii. Thalium Stress Test Result (thal): The dataset displays a variety of thalium stress test results. One particular type seems to be more prevalent, but the exact distribution can be seen in the plots.

ix. Presence of Heart Disease (target): The dataset is nearly balanced in terms of heart disease presence, with about 54.5% having it and 45.5% not having it.

Bivariate Analysis: Explored the relationship between each feature and the target variable. This helps us figure out the importance and influence of each feature on the target outcome.

- **For continuous data:** Bar plots to showcase the average value of each feature for the different target classes, and KDE plots to understand the distribution of each feature across the target classes. This aids in discerning how each feature varies between the two target outcomes.

```
# Set color palette
sns.set_palette(['#ff826e', 'red'])

# Create the subplots
fig, ax = plt.subplots(len(continuous_features), 2, figsize=(15,15), gridspec_kw={'width_ratios': [1, 2]})

# Loop through each continuous feature to create barplots and kde plots
for i, col in enumerate(continuous_features):
    # Barplot showing the mean value of the feature for each target category
    graph = sns.barplot(data=df, x="target", y=col, ax=ax[i,0])

    # KDE plot showing the distribution of the feature for each target category
    sns.kdeplot(data=df[df["target"]==0], x=col, fill=True, linewidth=2, ax=ax[i,1], label='0')
    sns.kdeplot(data=df[df["target"]==1], x=col, fill=True, linewidth=2, ax=ax[i,1], label='1')
    ax[i,1].set_yticks([])
    ax[i,1].legend(title='Heart Disease', loc='upper right')

    # Add mean values to the barplot
    for cont in graph.containers:
        graph.bar_label(cont, fmt='         %.3g')

# Set the title for the entire figure
plt.suptitle('Continuous Features vs Target Distribution', fontsize=22)
plt.tight_layout()
plt.show()
```



Inferences:

i. Age (age): The distributions show a slight shift with patients having heart disease being a bit younger on average than those without. The mean age for patients without heart disease is higher.

ii. Resting Blood Pressure (trestbps): Both categories display overlapping distributions in the KDE plot, with nearly identical mean values, indicating limited differentiating power for this feature.

iii. Serum Cholesterol (chol): The distributions of cholesterol levels for both categories are quite close, but the mean cholesterol level for patients with heart disease is slightly lower.

iv. Maximum Heart Rate Achieved (thalach): There's a noticeable difference in distributions. Patients with heart disease tend to achieve a higher maximum heart rate during stress tests compared to those without.

v. ST Depression (oldpeak): The ST depression induced by exercise relative to rest is notably lower for patients with heart disease. Their distribution peaks near zero, whereas the non-disease category has a wider spread.

Based on the visual difference in distributions and mean values, Maximum Heart Rate (thalach) seems to have the most impact on the heart disease status, followed by ST Depression (oldpeak) and Age (age).

- **For categorical data:** Stacked bar plots to depict the proportion of each category across the target classes. This offers a comprehensive view of how different categories within a feature relate to the target.

```
# Remove 'target' from the categorical_features
categorical_features = [feature for feature in categorical_features if feature != 'target']
```

```
fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(15,10))

for i,col in enumerate(categorical_features):

    # Create a cross tabulation showing the proportion of purchased and non-purchased loans for each category of the feature
    cross_tab = pd.crosstab(index=df[col], columns=df['target'])

    # Using the normalize=True argument gives us the index-wise proportion of the data
    cross_tab_prop = pd.crosstab(index=df[col], columns=df['target'], normalize='index')

    # Plot stacked bar charts
    x, y = i//4, i%4
    cross_tab_prop.plot(kind='bar', ax=ax[x,y], stacked=True, width=0.8,
                        legend=False, ylabel='Proportion', sharey=True)

    # Add the proportions and counts of the individual bars to our plot
    for idx, val in enumerate([*cross_tab.index.values]):
        for (proportion, count, y_location) in zip(cross_tab_prop.loc[val],cross_tab.loc[val],cross_tab_prop.loc[val].cumsum(
            ax[x,y].text(x=idx-0.3, y=(y_location-proportion)+(proportion/2)-0.03,
                         s = f'    {count}\n({np.round(proportion * 100, 1)}%)',
                         color = "black", fontsize=9, fontweight="bold")

    # Add legend
    ax[x,y].legend(title='target', loc=(0.7,0.9), fontsize=8, ncol=2)
    # Set y Limit
    ax[x,y].set_ylim([0,1.12])
    # Rotate xticks
    ax[x,y].set_xticklabels(ax[x,y].get_xticklabels(), rotation=0)


plt.suptitle('Categorical Features vs Target Stacked Barplots', fontsize=22)
plt.tight_layout()
plt.show()
```

Inferences:

i. Number of Major Vessels (ca): The majority of patients with heart disease have fewer major vessels colored by fluoroscopy. As the number of colored vessels increases, the proportion of patients with heart disease tends to decrease. Especially, patients with 0 vessels colored have a higher proportion of heart disease presence.

ii. Chest Pain Type (cp): Different types of chest pain present varied proportions of heart disease. Notably, types 1, 2, and 3 have a higher proportion of heart disease presence compared to type 0. This suggests the type of chest pain can be influential in predicting the disease.

iii. Exercise Induced Angina (exang): Patients who did not experience exercise-induced angina (0) show a higher proportion of heart disease presence compared to those who did (1). This feature seems to have a significant impact on the target.

iv. Fasting Blood Sugar (fbs): The distribution between those with fasting blood sugar > 120 mg/dl (1) and those without (0) is relatively similar, suggesting fbs might have limited impact on heart disease prediction.

v. Resting Electrocardiographic Results (restecg): Type 1 displays a higher proportion of heart disease presence, indicating that this feature might have some influence on the outcome.

vi. Sex (sex): Females (1) exhibit a lower proportion of heart disease presence compared to males (0). This indicates gender as an influential factor in predicting heart disease.

vii. Slope of the Peak Exercise ST Segment (slope): The slope type 2 has a notably higher proportion of heart disease presence, indicating its potential as a significant predictor.

viii. Thalium Stress Test Result (thal): The reversible defect category (2) has a higher proportion of heart disease presence compared to the other categories, emphasizing its importance in prediction.

In summary, based on the visual representation:
Higher Impact on Target: ca, cp, exang, sex, slope, and thal
Moderate Impact on Target: restecg
Lower Impact on Target: fbs

STEP 5: Data Preprocessing

- Irrelevant Features Removal: No columns seem redundant or irrelevant.

- Missing Value Treatment:

```
# Check for missing values in the dataset
df.isnull().sum().sum()
```

```
0
```

Above result shows that there are no missing values in our dataset.

- Outlier Treatment: Check for outliers using the IQR method for the continuous features.

```
continuous_features
```

```
['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

```
Q1 = df[continuous_features].quantile(0.25)
Q3 = df[continuous_features].quantile(0.75)
IQR = Q3 - Q1
outliers_count_specified = ((df[continuous_features] < (Q1 - 1.5 * IQR)) | (df[continuous_features] > (Q3 + 1.5 * IQR))).sum(

outliers_count_specified
```

```
age       0
trestbps  9
chol      5
thalach   1
oldpeak   5
dtype: int64
```

Inference: Given the nature of the algorithms (especially SVM and KNN) and the small size of our dataset, direct removal of outliers might not be the best approach. Instead, we'll focus on applying transformations like Box-Cox in the subsequent steps to reduce the impact of outliers and make the data more suitable for modeling.

- Categorical features encoding: One-hot encoding is done. Nominal variables with no inherent order should be one-hot encoded, as numbers might introduce an unintended order to the model. Ordinal variables have inherent order hence need not be encoded.

  Need One-Hot Encoding: cp, restecg, thal

  Donot Need One-Hot Encoding: sex, fbs, exang, slope,ca

```
# Implementing one-hot encoding on the specified categorical features
df_encoded = pd.get_dummies(df, columns=['cp', 'restecg', 'thal'], drop_first=True)

# Convert the rest of the categorical variables that don't need one-hot encoding to integer data type
features_to_convert = ['sex', 'fbs', 'exang', 'slope', 'ca', 'target']
for feature in features_to_convert:
    df_encoded[feature] = df_encoded[feature].astype(int)

df_encoded.dtypes
```

```
age              int64
sex              int32
trestbps         int64
chol             int64
fbs              int32
thalach          int64
exang            int32
oldpeak        float64
slope            int32
ca               int32
target           int32
cp_1              bool
cp_2              bool
cp_3              bool
restecg_1         bool
restecg_2         bool
thal_1            bool
thal_2            bool
thal_3            bool
dtype: object
```

```
# Displaying the resulting DataFrame after one-hot encoding
df_encoded.head()
```

	age	sex	trestbps	chol	fbs	thalach	exang	oldpeak	slope	ca	target	cp_1	cp_2	cp_3	restecg_1	restecg_2	thal_1	thal_2	thal_3
0	63	1	145	233	1	150	0	2.3	0	0	1	False	False	True	False	False	True	False	False
1	37	1	130	250	0	187	0	3.5	0	0	1	False	True	False	True	False	False	True	False
2	41	0	130	204	0	172	0	1.4	2	0	1	True	False	False	False	False	False	True	False
3	56	1	120	236	0	178	0	0.8	2	0	1	True	False	False	True	False	False	True	False
4	57	0	120	354	0	163	1	0.6	2	0	1	False	False	False	True	False	False	True	False

- Transforming Skewed Features: Box-Cox transformation is a powerful method to stabilize variance and make the data more normal-distribution-like. It is particularly useful when we are unsure about the exact nature of the distribution we are dealing with, as it can adapt itself to the best power transformation. However, the Box-Cox transformation only works for positive data, so one must be cautious when applying it to features that contain zeros or negative values.

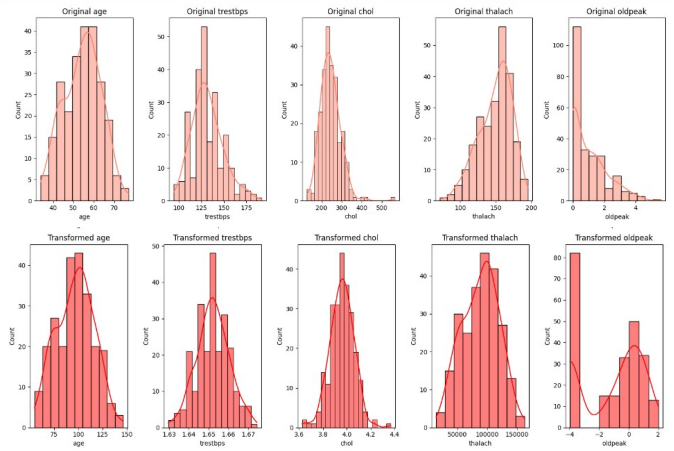
```
# Define the features (X) and the output labels (y)
X = df_encoded.drop('target', axis=1)
y = df_encoded['target']
```

```
# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)
```

The Box-Cox transformation requires all data to be strictly positive. To transform the oldpeak feature using Box-Cox, we can add a small constant (e.g., 0.001) to ensure all values are positive:

```
# Adding a small constant to 'oldpeak' to make all values positive
X_train['oldpeak'] = X_train['oldpeak'] + 0.001
X_test['oldpeak'] = X_test['oldpeak'] + 0.001
```

```
# Checking the distribution of the continuous features
fig, ax = plt.subplots(2, 5, figsize=(15,10))

# Original Distributions
for i, col in enumerate(continuous_features):
    sns.histplot(X_train[col], kde=True, ax=ax[0,i], color='#ff826e').set_title(f'Original {col}')


# Applying Box-Cox Transformation
# Dictionary to store lambda values for each feature
lambdas = {}

for i, col in enumerate(continuous_features):
    # Only apply box-cox for positive values
    if X_train[col].min() > 0:
        X_train[col], lambdas[col] = boxcox(X_train[col])
        # Applying the same lambda to test data
        X_test[col] = boxcox(X_test[col], lmbda=lambdas[col])
        sns.histplot(X_train[col], kde=True, ax=ax[1,i], color='red').set_title(f'Transformed {col}')
    else:
        sns.histplot(X_train[col], kde=True, ax=ax[1,i], color='green').set_title(f'{col} (Not Transformed)')

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



Inference:

i. age: The transformation has made the age distribution more symmetric, bringing it closer to a normal distribution.

ii. Trestbps: The distribution of trestbps post-transformation appears to be more normal-like, with reduced skewness.

iii. Chol: After applying the Box-Cox transformation, chol exhibits a shape that's more aligned with a normal distribution.

iv. Thalach: The thalach feature was already fairly symmetric before the transformation, and post-transformation, it continues to show a similar shape, indicating its original distribution was close to normal.

v. Oldpeak: The transformation improved the oldpeak distribution, but it still doesn't perfectly resemble a normal distribution. This could be due to the inherent

nature of the data or the presence of outliers. To enhance its normality, we could consider utilizing advanced transformations such as the Yeo-Johnson transformation, which can handle zero and negative values directly.

STEP 6: Model Building

- **Decision Tree Model**

Base model definition:

```
# Define the base DT model
dt_base = DecisionTreeClassifier(random_state=42)
```

Hyperparameter tuning: A function is defined to determine the optimal set of hyperparameters that yield the highest recall for the model. This approach ensures a reusable framework for hyperparameter tuning of subsequent models.

```
def tune_clf_hyperparameters(clf, param_grid, X_train, y_train, scoring='recall', n_splits=3):
    '''
    This function optimizes the hyperparameters for a classifier by searching over a specified hyperparameter grid.
    It uses GridSearchCV and cross-validation (StratifiedKFold) to evaluate different combinations of hyperparameters.
    The combination with the highest recall for class 1 is selected as the default scoring metric.
    The function returns the classifier with the optimal hyperparameters.
    '''

    # Create the cross-validation object using StratifiedKFold to ensure the class distribution is the same across all the fo
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=0)

    # Create the GridSearchCV object
    clf_grid = GridSearchCV(clf, param_grid, cv=cv, scoring=scoring, n_jobs=-1)

    # Fit the GridSearchCV object to the training data
    clf_grid.fit(X_train, y_train)

    # Get the best hyperparameters
    best_hyperparameters = clf_grid.best_params_

    # Return best_estimator_ attribute which gives us the best model that has been fitted to the training data
    return clf_grid.best_estimator_, best_hyperparameters
```

set up the hyperparameters grid and utilize the tune_clf_hyperparameters function to pinpoint the optimal hyperparameters for our DT model:

```
# Hyperparameter grid for DT
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2,3],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2]
}
```

```
# Call the function for hyperparameter tuning
best_dt, best_dt_hyperparams = tune_clf_hyperparameters(dt_base, param_grid_dt, X_train, y_train)
```

```
print('DT Optimal Hyperparameters: \n', best_dt_hyperparams)
```

```
DT Optimal Hyperparameters:
 {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

Model evaluation:

```
# Evaluate the optimized model on the train data
print(classification_report(y_train, best_dt.predict(X_train)))
```

```
              precision    recall  f1-score   support

           0       0.73      0.75      0.74       110
           1       0.78      0.77      0.78       132

    accuracy                           0.76       242
   macro avg       0.76      0.76      0.76       242
weighted avg       0.76      0.76      0.76       242
```

```
# Evaluate the optimized model on the test data
print(classification_report(y_test, best_dt.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.80      0.71      0.75        28
           1       0.78      0.85      0.81        33

    accuracy                           0.79        61
   macro avg       0.79      0.78      0.78        61
weighted avg       0.79      0.79      0.79        61
```

Given that the metric values for both the training and test datasets are closely aligned and not significantly different, the model doesnot appear to be overfitting.

A function that consolidates each model's metrics into a dataframe, facilitating an end-to-end comparison of all models later:

```
def evaluate_model(model, X_test, y_test, model_name):
    """
    Evaluates the performance of a trained model on test data using various metrics.
    """
    # Make predictions
    y_pred = model.predict(X_test)

    # Get classification report
    report = classification_report(y_test, y_pred, output_dict=True)

    # Extracting metrics
    metrics = {
        "precision_0": report["0"]["precision"],
        "precision_1": report["1"]["precision"],
        "recall_0": report["0"]["recall"],
        "recall_1": report["1"]["recall"],
        "f1_0": report["0"]["f1-score"],
        "f1_1": report["1"]["f1-score"],
        "macro_avg_precision": report["macro avg"]["precision"],
        "macro_avg_recall": report["macro avg"]["recall"],
        "macro_avg_f1": report["macro avg"]["f1-score"],
        "accuracy": accuracy_score(y_test, y_pred)
    }

    # Convert dictionary to dataframe
    df = pd.DataFrame(metrics, index=[model_name]).round(2)

    return df
```

```
dt_evaluation = evaluate_model(best_dt, X_test, y_test, 'DT')
dt_evaluation
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
DT	0.8	0.78	0.71	0.85	0.75	0.81	0.79	0.78	0.78	0.79

- **Logistic Regression Model**

Base model definition:

```
# Define the base LR model and set up the pipeline with scaling
lr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('lr', LogisticRegression(random_state=42))
])
```

Hyperparameter Tuning:

```
# --- Logistic Regression Parameters ---
param_grid_lr = {"lr__solver": ["lbfgs", "saga", "newton-cg"], 'lr__C': [0.1, 0.2, 0.5, 0.8]
                 }
```

```
# Using the tune_clf_hyperparameters function to get the best estimator
best_lr, best_lr_hyperparams = tune_clf_hyperparameters(lr_pipeline, param_grid_lr, X_train, y_train)
print('LR Optimal Hyperparameters: \n', best_lr_hyperparams)
```

```
LR Optimal Hyperparameters:
 {'lr__C': 0.1, 'lr__solver': 'lbfgs'}
```

Model Evaluation:

```
# Evaluate the optimized model on the train data
print(classification_report(y_train, best_lr.predict(X_train)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.80      0.82       110
           1       0.84      0.88      0.86       132

    accuracy                           0.84       242
   macro avg       0.84      0.84      0.84       242
weighted avg       0.84      0.84      0.84       242
```

```
# Evaluate the optimized model on the test data
print(classification_report(y_test, best_lr.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.84        28
           1       0.85      0.88      0.87        33

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

The LR model's similar performance on both training and test data suggests it is not overfitting

```
lr_evaluation = evaluate_model(best_lr, X_test, y_test, 'LR')
lr_evaluation
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
LR	0.85	0.85	0.82	0.88	0.84	0.87	0.85	0.85	0.85	0.85

- **Random Forest Model**

Base Model Definition

```
rf_base = RandomForestClassifier(random_state=42)
```

Hyperparameter Tunig:

```
param_grid_rf = {
    'n_estimators': [10, 30, 50, 70, 100],
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4],
    'min_samples_split': [2, 3, 4, 5],
    'min_samples_leaf': [1, 2, 3],
    'bootstrap': [True, False]
}
```

```
# Using the tune_clf_hyperparameters function to get the best estimator
best_rf, best_rf_hyperparams = tune_clf_hyperparameters(rf_base, param_grid_rf, X_train, y_train)
print('RF Optimal Hyperparameters: \n', best_rf_hyperparams)
```

```
RF Optimal Hyperparameters:
 {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
100}
```

Model Evaluation:

```
# Evaluate the optimized model on the train data
print(classification_report(y_train, best_rf.predict(X_train)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.78      0.82       110
           1       0.83      0.89      0.86       132

    accuracy                           0.84       242
   macro avg       0.84      0.83      0.84       242
weighted avg       0.84      0.84      0.84       242
```

```
# Evaluate the optimized model on the test data
print(classification_report(y_test, best_rf.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.83      0.71      0.77        28
           1       0.78      0.88      0.83        33

    accuracy                           0.80        61
   macro avg       0.81      0.80      0.80        61
weighted avg       0.81      0.80      0.80        61
```

The RF model's similar performance on both training and test data suggests it is not overfitting.

```
rf_evaluation = evaluate_model(best_rf, X_test, y_test, 'RF')
rf_evaluation
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
RF	0.83	0.78	0.71	0.88	0.77	0.83	0.81	0.8	0.8	0.8

- **KNN Model**

Base model definition:

```
# Define the base KNN model and set up the pipeline with scaling
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])
```

Hyperparameter Tuning:

```
# Hyperparameter grid for KNN
knn_param_grid = {
    'knn__n_neighbors': list(range(1, 12)),
    'knn__weights': ['uniform', 'distance'],
    'knn__p': [1, 2]  # 1: Manhattan distance, 2: Euclidean distance
}
```

```
# Hyperparameter tuning for KNN
best_knn, best_knn_hyperparams = tune_clf_hyperparameters(knn_pipeline, knn_param_grid, X_train, y_train)
print('KNN Optimal Hyperparameters: \n', best_knn_hyperparams)
```

```
KNN Optimal Hyperparameters:
 {'knn__n_neighbors': 9, 'knn__p': 1, 'knn__weights': 'uniform'}
```

Model Evaluation:

```
# Evaluate the optimized model on the train data
print(classification_report(y_train, best_knn.predict(X_train)))
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.79       110
           1       0.83      0.83      0.83       132

    accuracy                           0.81       242
   macro avg       0.81      0.81      0.81       242
weighted avg       0.81      0.81      0.81       242
```

```
# Evaluate the optimized model on the test data
print(classification_report(y_test, best_knn.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        28
           1       0.85      0.85      0.85        33

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

The KNN model's consistent scores across training and test sets indicate no overfitting.

```
knn_evaluation = evaluate_model(best_knn, X_test, y_test, 'KNN')
knn_evaluation
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
KNN	0.82	0.85	0.82	0.85	0.82	0.85	0.83	0.83	0.83	0.84

- **SVM Model**

Base model definition:

```
# define the base SVM model and set up the pipeline with scaling
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(probability=True))
])
```

Hyperparameter Tuning:

```
param_grid_svm = {
    'svm__C': [0.0011, 0.005, 0.01, 0.05, 0.1, 1, 10, 20],
    'svm__kernel': ['linear', 'rbf', 'poly'],
    'svm__gamma': ['scale', 'auto', 0.1, 0.5, 1, 5],
    'svm__degree': [2, 3, 4]
}
```

```
# Call the function for hyperparameter tuning
best_svm, best_svm_hyperparams = tune_clf_hyperparameters(svm_pipeline, param_grid_svm, X_train, y_train)
print('SVM Optimal Hyperparameters: \n', best_svm_hyperparams)
```

```
SVM Optimal Hyperparameters:
 {'svm__C': 0.0011, 'svm__degree': 2, 'svm__gamma': 'scale', 'svm__kernel': 'linear'}
```

Model Evaluation:

```
# Evaluate the optimized model on the train data
print(classification_report(y_train, best_svm.predict(X_train)))
```

```
              precision    recall  f1-score   support

           0       0.92      0.54      0.68       110
           1       0.71      0.96      0.82       132

    accuracy                           0.77       242
   macro avg       0.82      0.75      0.75       242
weighted avg       0.81      0.77      0.76       242
```

```
# Evaluate the optimized model on the test data
print(classification_report(y_test, best_svm.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.94      0.57      0.71        28
           1       0.73      0.97      0.83        33

    accuracy                           0.79        61
   macro avg       0.83      0.77      0.77        61
weighted avg       0.83      0.79      0.78        61
```

Overall, the model's performance is promising for medical diagnostics, especially when prioritizing the accurate identification of patients with heart disease without overburdening the system with false alarms.

```
svm_evaluation = evaluate_model(best_svm, X_test, y_test, 'SVM')
svm_evaluation
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
SVM	0.94	0.73	0.57	0.97	0.71	0.83	0.83	0.77	0.77	0.79

- **Naive Bayes Model**

Base model definition:

```
# define the base SVM model and set up the pipeline with scaling
nb_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('nb', GaussianNB())
])
```

Hyperparameter Tuning:

```
param_grid_nb = {'nb__var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]}
```

```
# Call the function for hyperparameter tuning
best_nb, best_nb_hyperparams = tune_clf_hyperparameters(nb_pipeline, param_grid_nb, X_train, y_train)
print('NB Optimal Hyperparameters: \n', best_nb_hyperparams)
```

```
NB Optimal Hyperparameters:
 {'nb__var_smoothing': 1e-09}
```

Model Evaluation:

```
# Evaluate the optimized model on the train data
print(classification_report(y_train, best_nb.predict(X_train)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.80      0.81       110
           1       0.84      0.85      0.84       132

    accuracy                           0.83       242
   macro avg       0.83      0.82      0.82       242
weighted avg       0.83      0.83      0.83       242
```

```
# Evaluate the optimized model on the test data
print(classification_report(y_test, best_nb.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        28
           1       0.88      0.88      0.88        33

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

The Naive Bayes model's consistent scores across training and test sets indicate no overfitting.

```
nb_evaluation = evaluate_model(best_nb, X_test, y_test, 'NB')
nb_evaluation
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
NB	0.86	0.88	0.86	0.88	0.86	0.88	0.87	0.87	0.87	0.87

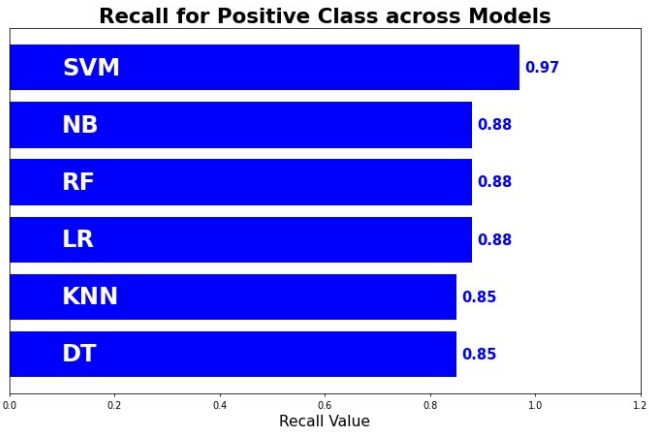
3. CONCLUSION

In the critical context of diagnosing heart disease, our primary objective is to ensure a high recall for the positive class. It's imperative to accurately identify every potential heart disease case, as even one missed diagnosis could have dire implications. However, while striving for this high recall, it's essential to maintain a balanced performance to avoid unnecessary medical interventions for healthy individuals. Evaluate our models against these crucial medical benchmarks.

```
# Concatenate the dataframes
all_evaluations = [lr_evaluation,dt_evaluation, rf_evaluation, knn_evaluation, svm_evaluation,nb_evaluation]
results = pd.concat(all_evaluations)

# Sort by 'recall_1'
results = results.sort_values(by='recall_1', ascending=False).round(2)
results
```

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
SVM	0.94	0.73	0.57	0.97	0.71	0.83	0.83	0.77	0.77	0.79
LR	0.85	0.85	0.82	0.88	0.84	0.87	0.85	0.85	0.85	0.85
RF	0.83	0.78	0.71	0.88	0.77	0.83	0.81	0.80	0.80	0.80
NB	0.86	0.88	0.86	0.88	0.86	0.88	0.87	0.87	0.87	0.87
DT	0.80	0.78	0.71	0.85	0.75	0.81	0.79	0.78	0.78	0.79
KNN	0.82	0.85	0.82	0.85	0.82	0.85	0.83	0.83	0.83	0.84

```
# Sort values based on 'recall_1'
results.sort_values(by='recall_1', ascending=True, inplace=True)
recall_1_scores = results['recall_1']

# Plot the horizontal bar chart
fig, ax = plt.subplots(figsize=(12, 7), dpi=70)
ax.barh(results.index, recall_1_scores, color='blue')

# Annotate the values and indexes
for i, (value, name) in enumerate(zip(recall_1_scores, results.index)):
    ax.text(value + 0.01, i, f"{value:.2f}", ha='left', va='center', fontweight='bold', color='blue', fontsize=15)
    ax.text(0.1, i, name, ha='left', va='center', fontweight='bold', color='white', fontsize=25)

# Remove yticks
ax.set_yticks([])

# Set x-axis limit
ax.set_xlim([0, 1.2])

# Add title and xlabel
plt.title("Recall for Positive Class across Models", fontweight='bold', fontsize=22)
plt.xlabel('Recall Value', fontsize=16)
plt.show()
```



The SVM model demonstrates a commendable capability in recognizing potential heart patients. With a recall of 0.97 for class 1, it's evident that almost all patients with heart disease are correctly identified. This is of paramount importance in a medical setting. However, the model's balanced performance ensures that while aiming for high recall, it doesn't compromise on precision, thereby not overburdening the system with unnecessary alerts

4. REFERENCES

[1] Soni J, Ansari U, Sharma D & Soni S (2011). Predictive data mining for medical diagnosis: an overview of heart disease prediction. International Journal of Computer Applications, 17(8), 43-8 [2] Dangare C S & Apte S S (2012). Improved study of heart disease prediction system using data mining classification techniques. International Journal of Computer Applications, 47(10), 44-8. [3] Ordonez C (2006). Association rule discovery with the train and test approach for heart disease prediction. IEEE Transactions on Information Technology in Biomedicine, 10(2), 334-43. [4] Shinde R, Arjun S, Patil P & Waghmare J (2015). An intelligent heart disease prediction system using k-means clustering and Naïve Bayes algorithm. International Journal of Computer Science and Information Technologies, 6(1), 637-9. [5] Bashir S, Qamar U & Javed M Y (2014, November). An ensemble-based decision support framework for intelligent heart disease diagnosis. In International Conf