Heart Disease Prediction Using Machine Learning Techniques

Submitted by

G. KARTHIK

(AA.SC.P2MCA2301034)

in partial fulfilment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

07 February 2025

BONAFIDE CERTIFICATE

This is to certify that this dissertation titled "**Predicting Depression Among Students**," submitted in partial fulfilment of the requirements for the award of the Degree of **Master of Computer Applications**, by Karthik G (AA.SC.P2MCA2301034), is a bona fide record of the work carried out by him/her under my supervision during the academic year 2023-2025 and that it has not been submitted, to the best of my knowledge, in part or in full, for the award of any other degree or diploma.

Project Guide's name
Ms. Deepa Sreedhar

Coordinator's name

Reviewer
Ms. Prathibha KS

Date:08-02-2025

DECLARATION

I do hereby declare that this dissertation titled **"Heart Disease Prediction**", submitted in partial fulfilment of the requirements for the award of the degree of **Master of Computer Applications,** is a true record of work carried out by me and that all information contained herein, which do not arise directly from my work, have been properly acknowledged and cited, using acceptable international standards. Further, I declare that the contents of this thesis have not been submitted, in part or in full, for the award of any other degree or diploma.

Signature of the student

Date: 08.02.2025

G. Karthik

Acknowledgements

I would like to express my deepest gratitude to **Amrita Vishwa Vidyapeetham, Coimbatore**, for providing me with the opportunity and resources to successfully complete my postgraduate studies.

I extend my sincere appreciation to my **guide Ms. Deepa Sreedhar** for their invaluable guidance, constant encouragement, and insightful feedback throughout this research. Their expertise and support have been instrumental in shaping this work.

I would also like to thank my **department faculty members** for their constructive suggestions and academic support. My heartfelt gratitude goes to my **family and friends**, whose unwavering support and motivation have been a pillar of strength in this journey.

Finally, I acknowledge the efforts of all those who contributed directly or indirectly to the completion of this report. Their assistance and encouragement have been truly invaluable.

ABSTRACT

Heart disease remains one of the leading causes of mortality worldwide, emphasizing the need for early and accurate detection. This project leverages data science techniques to develop a predictive model for heart disease diagnosis. By analyzing clinical parameters such as age, blood pressure, cholesterol levels, and other health indicators, the project employs machine learning algorithms to identify patterns and risk factors associated with heart disease. The dataset, sourced from credible medical records, undergoes preprocessing to handle missing values and outliers, ensuring robust model performance. Feature selection techniques are used to prioritize key predictors, and various classification algorithms, including logistic regression, random forests, and neural networks, are evaluated to determine the most accurate model. The resulting system achieves a high predictive accuracy, providing a valuable tool for clinicians to support early diagnosis and personalized patient care. This project demonstrates the potential of data-driven approaches in advancing healthcare outcomes and reducing the global burden of heart disease.

CONTENTS

CHAPTER NO. **PAGE**

List of Tables

Table of Contents:

CHAPTER 1

INTRODUCTION

1.1 Course Overview

IBM Data Science Professional Certificate Course is twelve course series. The program gives opportunity to develop necessary skills and expertise in required tools for working as a data scientist at entrylevel. For this program contains 12 courses which make an individual to develop required skill-set, knowledge and practice to start a career in data science. Data science involves gathering, cleaning, organizing, and analysing data with the goal of extracting helpful insights and predicting expected outcomes.

The program starts with **the basic course i.e. What is Data Science?** It introduces about the work of data scientist and very fundamental information about the data science. It explains how data scientists follow certain processes to answer the question of concern with the data.

The second course (Tools for Data Science) gives understanding about different types and categories of tools that data scientists use such as programming languages-Python, R, SQL; Jupyter Notebook which allows a Data Scientist to record their data experiments and results that others can reuse.

Third Course (Data Science Methodology) explains about the data science methodology which is being followed to solve a particular problem in a domain. Data Science methodology is a structured approach to solving complex problems using data. The approach involves several steps in the given order-Business Understanding, Analytical Approach, Data Requirements, Data Collection, Data Understanding, Data Preparation, Modeling, Evaluation, Deployment and Feedback.

Forth Course (Python for Data Science, AI & Development) explains the basics of Python programming language which is widely used in data science. The course teaches basic concepts in python - data types, variables, data structures used - Lists, Tuples, Dictionaries; sets, Loops, functions, Reading & Writing files in Python, Pandas & NumPy Library.

Fifth Course (Python Project for Data Science) contains a capstone project for data science. The capstone requires to analyse the stock performance and building a dashboard. The stock data is obtained by performing web scrapping using python.

Sixth course (Databases and SQL for Data Science with) teaches to analyse the data obtained using Python and SQL. Further the course also familiarizes the basic concepts of using Python to connect to databases. Using Jupyter Notebook one can create tables, load data, query data using SQL magic and SQLite python library. The course also educates to analyse the data using SQL queries in Jupyter notebook.

Seventh Course (Data Analysis with Python) teaches for developing Python code for cleaning and preparing data for analysis - including handling missing values, formatting, normalizing, and binning data. It also includes lectures on how to manipulate data using data frames, summarize data, understand data distribution and perform exploratory data analysis and apply analytical techniques to real-word datasets using libraries such as Pandas, NumPy and SciPy. **Eighth Course (Data Visualization with Python)** give lessons in implementing Implement data visualization techniques and plots using Python libraries, such as Matplotlib, Seaborn, and Folium to tell a stimulating story regarding the data and its attributes.

Ninth Course (Machine Learning with Python) related to utilize Scikitlearn to build, test, and evaluate models. It covers to implement core machine learning algorithms, including linear regression, decision trees, and SVM, for classification and regression tasks. It also adds to evaluate model performance using metrics, cross-validation, and hyperparameter tuning to ensure accuracy and reliability.

Tenth Course is **Applied Data Science Project** which is related to demonstrating the skills in data science and machine learning using real world dataset problem. The same project is presented here in final report. **Eleventh Course** is related to **Generative AI in data science**. It teaches about the four common types of generative AI models and their impact and applications across diverse industries. It also covers how data scientists can leverage generative AI in the data science lifecycle.

Twelfth Course describe the role of a data scientist and some career path options as well as the prospective opportunities in the field. It further gives knowledge about how to prepare to appear in the interview and grab available opportunities. It also adds to build foundation to create portfolio and making an effective resume.

Conclusively, the program contains very practical elements and concepts to understand the data science and its implementation. The program starts with the basic introduction of data science and follows a theoretical and experimental learning path intermixed with labs and challenge to become proficient in the specified skill.

1.2 Project Overview

Objective:

The project aims to predict the likelihood of a person having heart disease using data science methodologies and machine learning models. The analysis is performed on openly available clinical datasets containing patient health metrics.

Methodology:

The project utilizes IBM Skill Labs integrated with JupyterLite for implementation. Python programming is the core language used, leveraging libraries such as Pandas, NumPy, Matplotlib, Seaborn, and Scikit-learn. The process involves key steps including Data Collection, Data Wrangling, Exploratory Data Analysis (EDA), Feature Selection, Model Training, and Model Comparison. Machine learning algorithms such as Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Decision Tree Classifier are employed for building predictive models.

Key Findings:

The trained models achieved an accuracy of approximately 85.67%, with Logistic Regression showing the most stable performance. However, minor overfitting was observed in some models, highlighting the need for additional data or regularization techniques to further enhance model performance and reliability.

Conclusion:

The prediction outcomes can assist medical professionals in identifying highrisk patients, enabling early interventions and personalized treatment plans. This project demonstrates how data-driven approaches can effectively contribute to better healthcare decision-making and outcomes.

CHAPTER 2

PROBLEM DEFINITION

Heart disease is a leading cause of death worldwide, making early detection and prevention critical. Traditional diagnostic methods often rely on extensive medical testing, which can be time-consuming and expensive. This project aims to address this issue by leveraging data science methodologies and machine learning models to predict the likelihood of heart disease based on a patient's clinical health metrics.

The key challenge lies in accurately identifying patterns in the data and differentiating between healthy and at-risk individuals. The dataset includes both numerical and categorical health parameters, such as age, blood pressure, cholesterol levels, and more. These features must be carefully analyzed, preprocessed, and modeled to ensure reliable predictions.

The ultimate goal is to develop a predictive system that is both accurate and interpretable, providing healthcare professionals with a valuable tool for early diagnosis and risk assessment, thereby improving patient outcomes and optimizing healthcare resources.

CHAPTER 3

REQUIREMENTS

3.1 Hardware:

A Computer with below mentioned requirements:
Processor: intel core i5.
RAM: 8 GB of RAM is the required for basic tasks, 8 GB of RAM is recommended. **Storage:** 5 GB of free disk space (to store datasets, libraries, and results)

3.2 Software:

Operating System: Windows 10 or newer, macOS 10.12 (Sierra) or later, Most modern Linux distributions (Ubuntu, Fedora, CentOS) **Python:** Python 3.7 or newer.

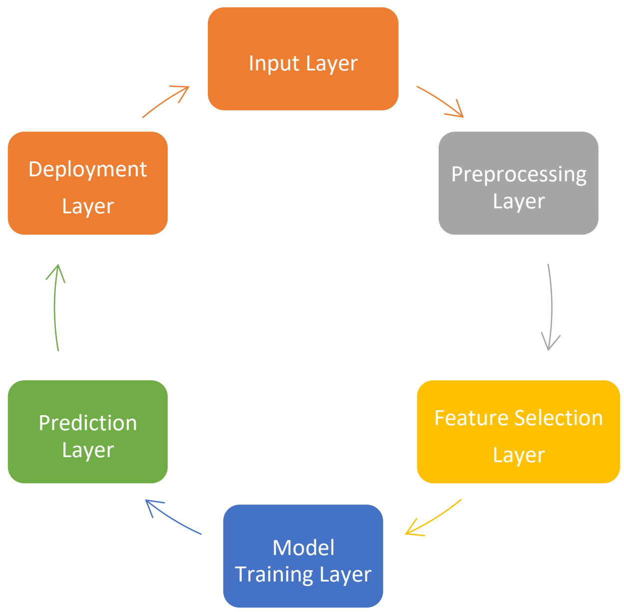
Python Libraries: Scikit-learn, Pandas, NumPy, SQLite3, Requests, Datetime, Beautifulsoup4, Matplotlib, Seaborn etc.

Web Browser: Chrome Browser, Microsoft Edge or Firefox etc.

Internet Connection: Reliable internet connection

CHAPTER 4
PROPOSED SYSTEM

The heart disease prediction system follows a systematic workflow divided into various stages: Data Collection, Preprocessing, Feature Selection, Model Development, Model Evaluation, and Deployment. This section elaborates on each stage with supporting diagrams and algorithms.



Algorithm 1: Data Preprocessing

1. **Input**: Raw patient data 2.
Process:
    - o Handle missing values by filling with mean/median. o Normalize numerical features (e.g., age, blood pressure).
    - o Encode categorical features like gender.
3. **Output**: Cleaned dataset ready for modeling.

Algorithm 2: Model Training

1. **Input**: Processed dataset 2. **Process**: o Split the dataset into training and test sets (80-20 split).
    - o Train multiple algorithms (Logistic Regression, Random Forest, SVM).
    - o Hyperparameter tuning for optimal performance.
3. **Output**: Trained model(s).

Algorithm 3: Prediction

1. **Input**: New patient data 2.
Process:
    - o Preprocess the input data.
    - o Pass the data to the trained model.
    - o Retrieve the prediction (e.g., heart disease or no heart disease).
3. **Output**: Prediction result.

4.4 Implementation Steps

1. Step 1 | Import Libraries

```
2]: import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
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

%matplotlib inline
```

2. Step 2 | Read Dataset

```
[14]: # Read dataset
df = pd.read_csv('./heart.csv')
df
```

[14]:

	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

-

3. Step 3 | Dataset Overview

```
import csv, sqlite3

con = sqlite3.connect("Heartpredictions.db")
cur = con.cursor()
```

```
%sql sqlite:///Heartpredictions.db
```

```
import pandas as pd
df = pd.read_csv("heart.csv")
```

```
cur.execute("SELECT * FROM People LIMIT 5")
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
(63, 1, 3, 145, 233, 1, 0, 150, 0, 2.3, 0, 0, 1, 1)
(37, 1, 2, 130, 250, 0, 1, 187, 0, 3.5, 0, 0, 2, 1)
(41, 0, 1, 130, 204, 0, 0, 172, 0, 1.4, 2, 0, 2, 1)
(56, 1, 1, 120, 236, 0, 1, 178, 0, 0.8, 2, 0, 2, 1)
(57, 0, 0, 120, 354, 0, 1, 163, 1, 0.6, 2, 0, 2, 1)
```

```
con.commit()
```

```
con.close()
```

4. [Step 3.1 | Dataset Basic Information]()

```
# 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
```

5. [Step 3.2 | Summary 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

6. Step 3.3 | Summary 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

7. Step 4 | EDA

For our **Exploratory Data Analysis (EDA)**, we'll take it in two main steps:

1. **Univariate Analysis**: Here, we'll focus on one feature at a time to understand its distribution and range.

2. **Bivariate Analysis**: In this step, we'll explore 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.

With these two steps, we aim to gain insights into the individual characteristics of the data and also how each feature relates to our main goal: **predicting the target variable**.

8. Step 4.1 | Univariate Analysis

We undertake univariate analysis on the dataset's features, based on their datatype:

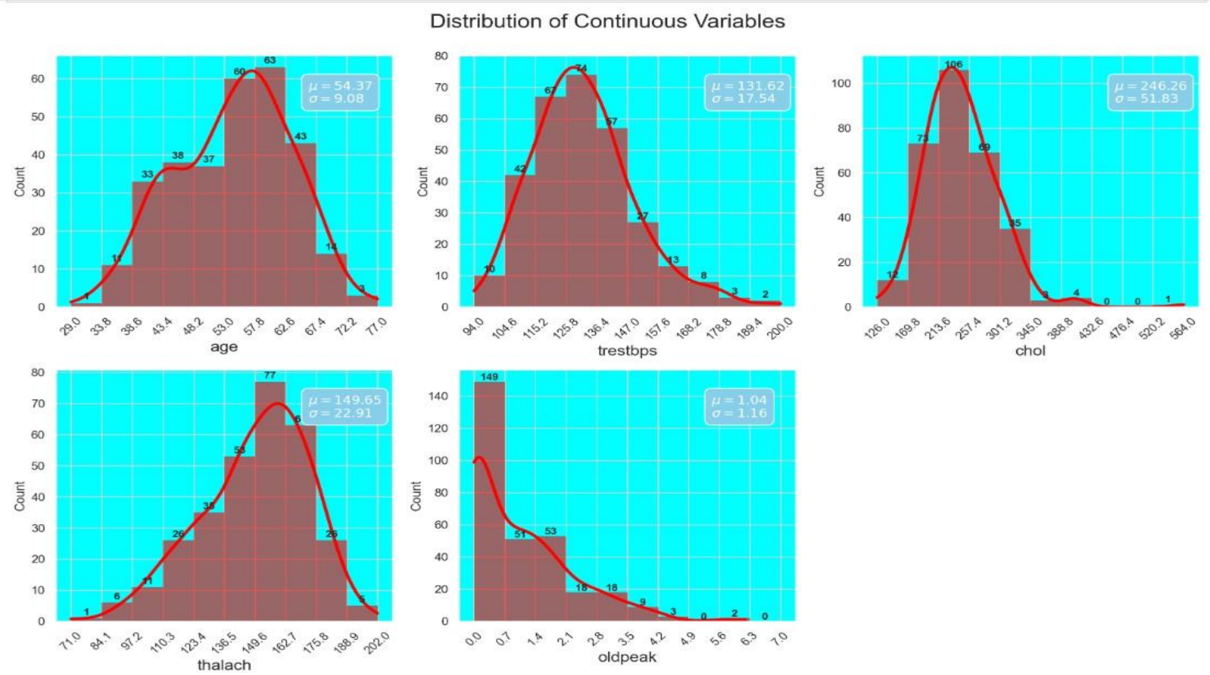
- For **continuous data**: We employ histograms to gain insight into the distribution of each feature. This allows us to understand the central tendency, spread, and shape of the dataset's distribution.

- 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.

By employing these visualization techniques, we're better positioned to understand the individual characteristics of each feature in the dataset.

9. [Step 4.1.1 | Numerical Variables Univariate Analysis]

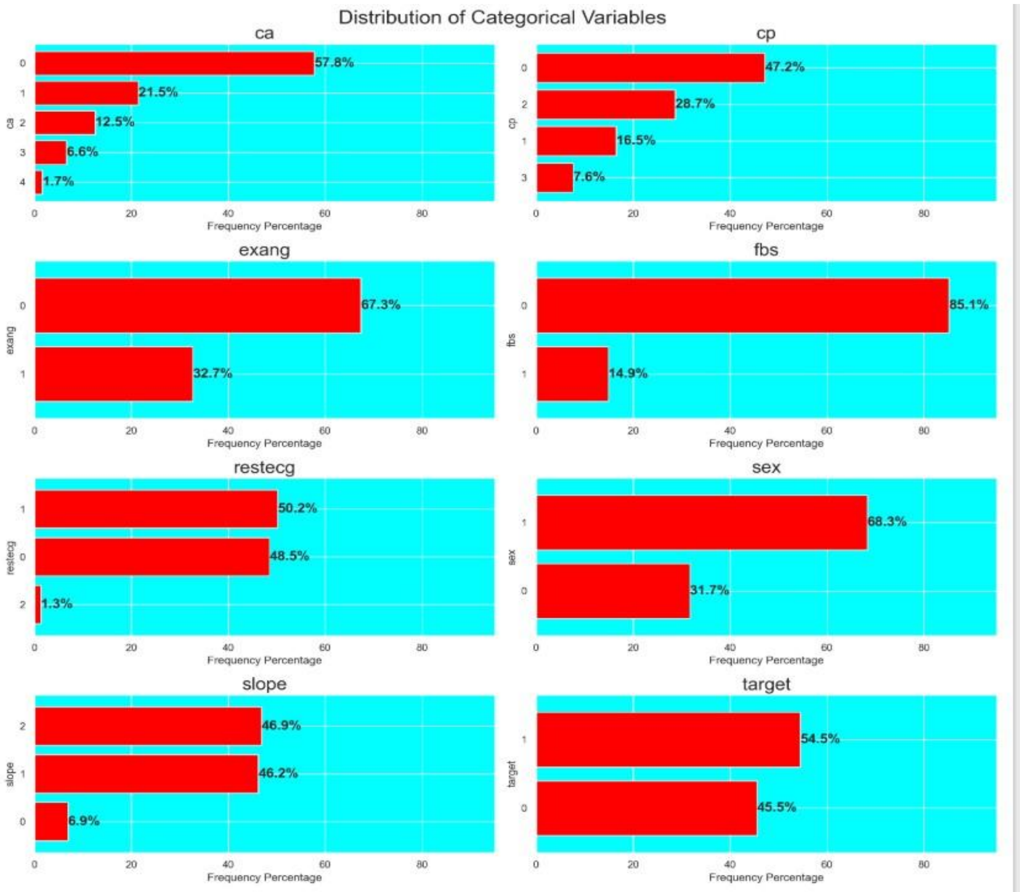
```
# Set up the subplot
fig, ax = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

# 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='skyblue', 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()
```



10. [Step 4.1.2 | Categorical Variables Univariate Analysis]

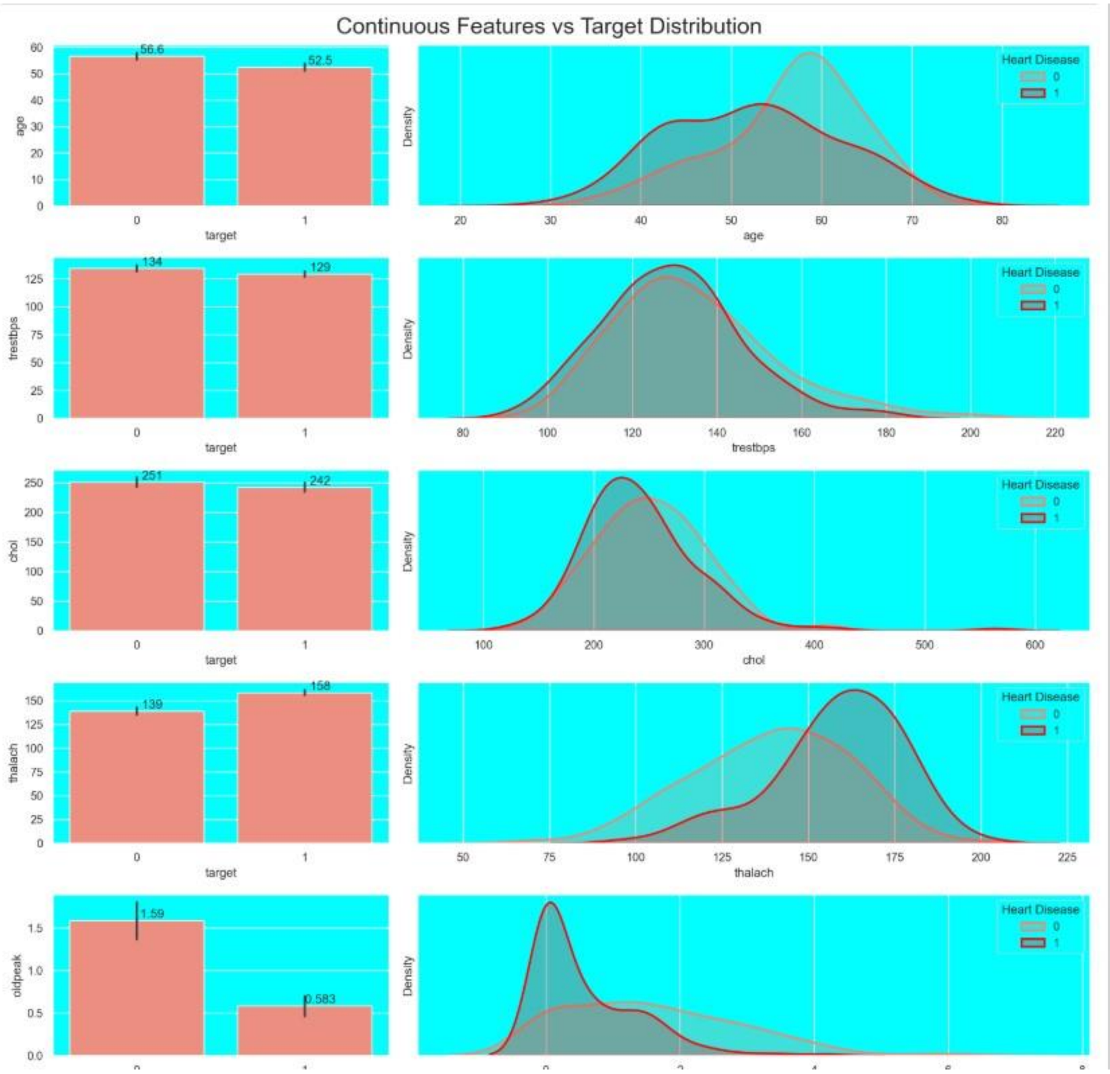
```
# Filter out categorical features for the univariate analysis
categorical_features = df.columns.difference(continuous_features)
df_categorical = df[categorical_features]
```

```
# 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()
```



11. Step 4.2 | Bivariate Analysis
12. Step 4.2.1 | Numerical Features vs Target

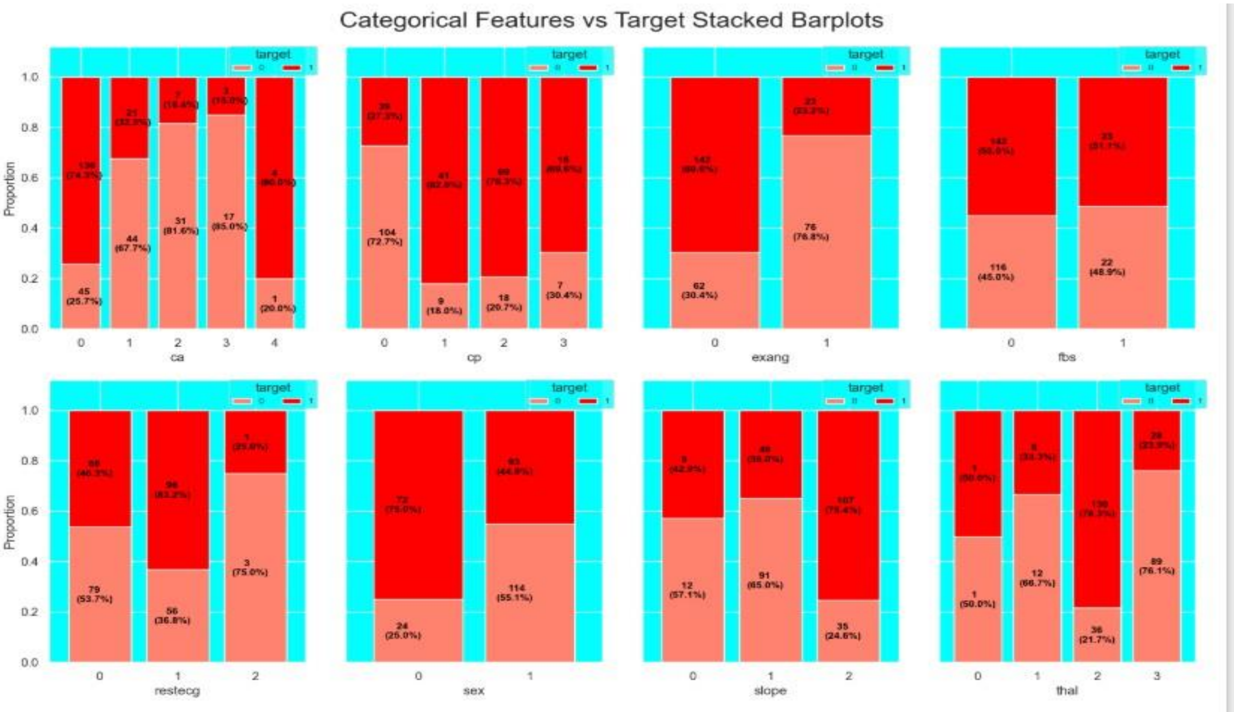
```
[26]: # Set color palette
sns.set_palette(['#ff826e', 'red'])

# 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()
```



13. [Step 4.2.2 | Categorical Features vs Target]

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

```python
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')

    # Define colormap
    cmp = ListedColormap(['#ff826e', 'red'])

    # 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, colormap=cmp,
                        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()
```



14. [Step 5 | Data Preprocessing]

```
[29]: # Check for missing values in the dataset
df.isnull().sum().sum()
```

```
[29]: 0
```

Upon our above inspection, it is obvious that there are no missing values in our dataset. This is ideal as it means we don't have to make decisions about imputation or removal, which can introduce bias or reduce our already limited dataset size.

Step 5.3 | Outlier Treatment ¶

I am going to check for outliers using the **IQR method** for the continuous features:

```
[ ]: continuous_features
```

```
[30]: 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
```

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

```
[31]: # 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
```

```
[31]: 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
```

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

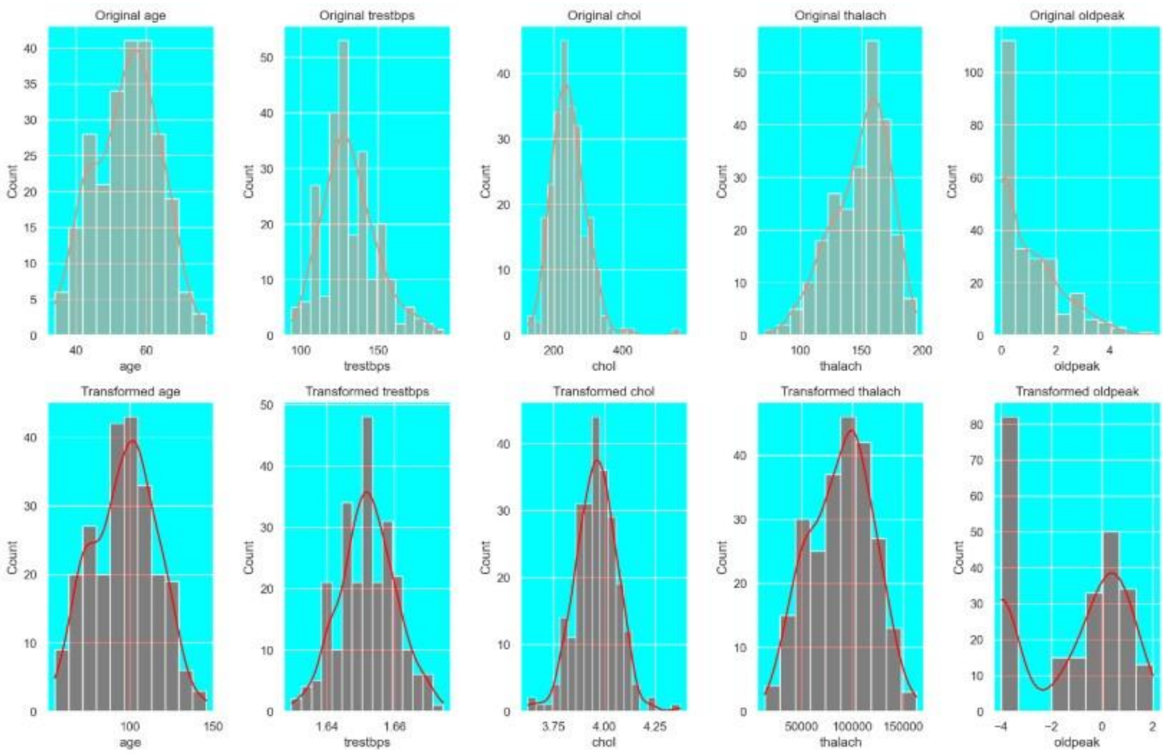
[32]:

	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

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

[34]: # 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)

[35]: continuous_features

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

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:

```
[36]: # 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

[37]: # 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()
```



[39]:
```python
# Define the base DT model
dt_base = DecisionTreeClassifier(random_state=0)
```

17. Step 6.2 | DT Hyper parameter Tuning

[43]:
```python
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 folds
    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
```

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

[44]:
```python
# 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]
}
```

[46]:
```python
# Call the function for hyperparameter tuning
best_dt, best_dt_hyperparams = tune_clf_hyperparameters(dt_base, param_grid_dt, X_train, y_train)
```

[47]:
```python
print('DT Optimal Hyperparameters: \n', best_dt_hyperparams)
```

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

18. Step 6.3 | DT Model Evaluation

[48]:
```python
# 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
```

[49]:
```python
# 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
```

```python
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
```

```python
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

19. Step 7 | Random Forest Model Building
20. Step 7.1 | RF Base Model Definition

Step 7.1 | RF Base Model Definition

First, let's define the base RF model:

```python
rf_base = RandomForestClassifier(random_state=0)
```

21. Step 7.2 | RF Hyper parameter Tuning

```python
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]
}
```

```python
# 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': True, 'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 30}
```

22. Step 7.3 | RF Model Evaluation

Finally, I am evaluating the model's performance on both the training and test datasets:

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

              precision    recall  f1-score   support

           0       0.84      0.79      0.81       110
           1       0.83      0.87      0.85       132

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

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

              precision    recall  f1-score   support

           0       0.85      0.79      0.81        28
           1       0.83      0.88      0.85        33

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

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

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

58]:

	precision_0	precision_1	recall_0	recall_1	f1_0	f1_1	macro_avg_precision	macro_avg_recall	macro_avg_f1	accuracy
RF	0.85	0.83	0.79	0.88	0.81	0.85	0.84	0.83	0.83	0.84

23. Step 8 | KNN Model Building
24. Step 8.1 | KNN Base Model Definition

Step 8.1 | KNN Base Model Definition

First of all, let's define the base KNN model and set up the pipeline with scaling:

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

25. Step 8.2 | KNN Hyperparameter Tuning

```
[60]: # 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
}
```

```
[61]: # 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'}
```

26. Step 8.3 | KNN Model Evaluation

Let's evaluate the model's performance on both the training and test datasets:

```
[62]: # 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
```

```
[63]: # 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.

```
[64]: knn_evaluation = evaluate_model(best_knn, X_test, y_test, 'KNN')
knn_evaluation
```

[64]:

	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

27. Step 9 | SVM Model Building
28. Step 9.1 | SVM Base Model Definition

First, let's define the base SVM model and set up the pipeline with scaling:

```
[65]: svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(probability=True))
])
```

29. Step 9.2 | SVM Hyperparameter Tuning

```
[66]: 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]
}
```

```
[67]: # 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'}
```

30. Step 9.3 | SVM 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
```

```
[70]: svm_evaluation = evaluate_model(best_svm, X_test, y_test, 'SVM')
svm_evaluation
```

[70]:

	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

31. Step 10 | Conclusion

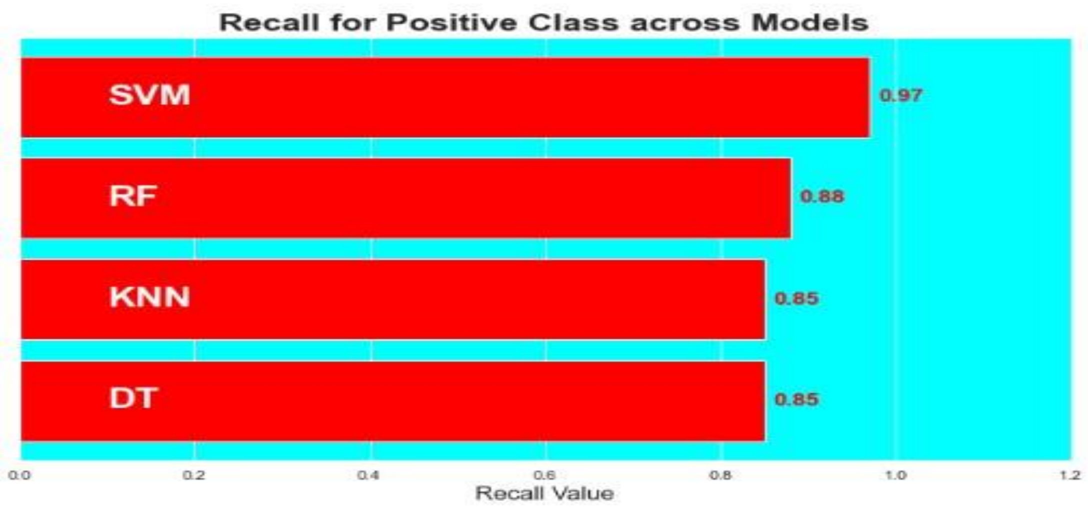
In the critical context of diagnosing heart disease, our primary objective is **to ensure a high recall for the positive class**. It's imperative to accurately identify every potential heart disease case, as even one missed diagnosis could have dire implications. However, while striving for this high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

[71]:
```python
# Concatenate the dataframes
all_evaluations = [dt_evaluation, rf_evaluation, knn_evaluation, svm_evaluation]
results = pd.concat(all_evaluations)

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

[71]:

	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
RF	0.85	0.83	0.79	0.88	0.81	0.85	0.84	0.83	0.83	0.84
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

[72]:
```python
# 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='red')

# 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='red', 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()
```



Model accuracy:

Random forest modelling : 0.88

Support Vector Machine: 0.97

Decision Tree Classifier: 0.85

K-Nearest neighbor: 0.85

CHAPTER 5 RESULT AND ANALYSIS

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.

REFERENCES

[1] IBM Data Science Professional Certificate Course on Coursera

[2] GeeksforGeeks url: https://www.geeksforgeeks.org/python-webscrapingtutorial/

[3] Data Analysis url: https://www.geeksforgeeks.org/what-is-data-analysis/

[4] Libraries and Tools: **Python Libraries** (Pandas, NumPy, Matplotlib, Seaborn, Scikitlearn, requests, Plotly)

Tools: IBM Skill**-Lab ; Jupyter notebook** for working with python;

APPENDIX A

Dataset Source:

url: https://github.com/K1rthik/DataScience.git

Source code

Github Url: https://github.com/K1rthik/DataScience.git