A Project Report on

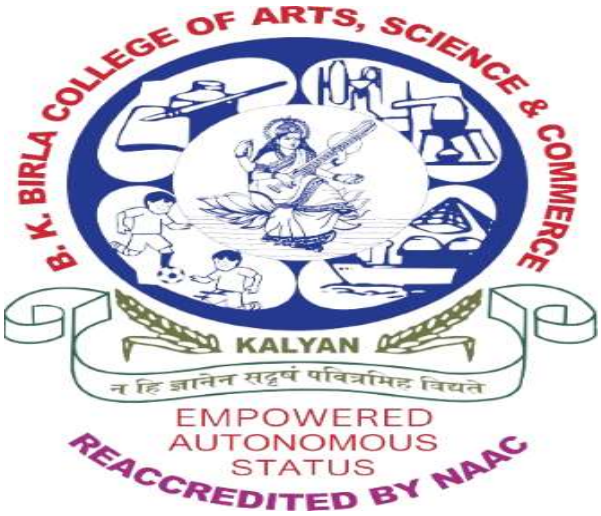
MULTIPLE DISEASE PREDICTION USING MACHINE LEARNING AND DEEP LEARNING

Submitted in partial fulfillment of the requirements for
the award of the degree of

Master of Science in Data Science and Big Data Analytics

BY:
AVADHUT DILIP VARVATKAR
3836385

Under the Guidance of
Miss Esmita Gupta



Department of Information Technology
B. K. Birla College of Arts, Science and Commerce (Autonomous), Kalyan
B. K. Birla College Road, Near RTO, Kalyan

UNIVERSITY OF MUMBAI

Academic Year 2023-2024

Acknowledgement

This Project Report entitled ***"MULTIPLE DISEASE PREDICTION USING MACHINE LEARNING AND DEEP LEARNING"*** Submitted by ***"AVADHUT DILIP VARVATKAR" (Student ID: 3836385)*** is approved for the partial fulfillment of the requirement for the award of the degree of ***Master of Science*** in ***Data Science and Big Data Analytics*** from ***University of Mumbai***.

(Prof. Esmita Gupta)
Guide

Prof. Esmita Gupta
Head, Department of Information Technology

Place: B. K. Birla College, Kalyan
Date:

CERTIFICATE

This is to certify that the project entitled ***"MULTIPLE DISEASE PREDICTION USING MACHINE LEARNING AND DEEP LEARNING"*** submitted by ***"AVADHUT DILIP VARVATKAR" (STUDENT ID: 3836385)*** for the partial fulfillment of the requirement for award of a degree ***Master of Science in Data Science and Big Data Analytics***, to the University of Mumbai, is a bonafide work carried out during academic year 2020-2021.

PLACE: KALYAN

Signature of External

DATE:

Signature of Guide

Signature of HOD

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Avadhut Varvatkar and STD ID: 3836385)

Date:

INDEX

1. ABSTRACT:

In recent years, the application of machine learning and deep learning techniques in healthcare has shown promising results in predicting and diagnosing various diseases. This project focuses on developing robust predictive models for diabetes and heart disease using advanced data-driven approaches. The primary objective is to enhance early detection and improve patient outcomes through accurate and timely predictions.

For diabetes prediction, we employed a range of machine learning algorithms, including logistic regression, decision trees, and random forests, to analyze patient data and identify key risk factors. The model's performance was evaluated using metrics such as accuracy, precision, recall, and the area under the receiver operating characteristic curve (AUC-ROC), demonstrating its effectiveness in distinguishing between diabetic and non-diabetic individuals.

In parallel, a deep learning approach was utilized for heart disease prediction. Leveraging neural networks, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), we processed complex datasets encompassing patient demographics, clinical history, and electrocardiogram (ECG) readings. The deep learning model exhibited superior predictive capabilities, achieving high accuracy and robustness in identifying individuals at risk of heart disease. The integration of both machine learning and deep learning models in this project underscores the potential of these technologies in predictive healthcare. By providing clinicians with reliable tools for early diagnosis, this project aims to contribute to better management and prevention strategies for diabetes and heart disease, ultimately enhancing patient care and reducing healthcare costs.

2. INTRODUCTION:

The advent of machine learning and deep learning technologies has significantly transformed the landscape of predictive analytics, particularly in healthcare. This project leverages these advanced techniques to develop predictive models for two prevalent health conditions: diabetes and heart disease. The early detection of these diseases can lead to timely interventions, improved patient outcomes, and reduced healthcare costs.

For the diabetes prediction model, a variety of machine learning algorithms are employed, including logistic regression, decision trees, random forests, support vector classifier (SVC), and Gaussian naïve Bayes. These algorithms help identify key risk factors and predict the likelihood of an individual developing diabetes. The model's performance is assessed using metrics such as accuracy, precision, recall, and the area under the receiver operating characteristic curve (AUC-ROC).

Similarly, the heart disease prediction model utilizes deep learning techniques, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs). These models process complex datasets that include patient demographics, clinical history, and electrocardiogram (ECG) readings. The deep learning approach demonstrates superior predictive capabilities, providing a robust tool for identifying individuals at risk of heart disease.

In parallel, this project involves the development of a flight price prediction system using machine learning techniques. The key steps in this development process include data preprocessing, exploratory data analysis (EDA), feature selection, model training, and hyper parameter tuning. The dataset is meticulously prepared to handle missing values and transform categorical variables into numerical representations suitable for analysis. EDA techniques are applied to uncover insights into the dataset and understand the relationships between various features and flight prices.

Feature selection plays a pivotal role in building an effective flight price prediction model. By identifying the most influential features, the system can concentrate on the essential variables, thereby enhancing the accuracy of the price predictions. The Random Forest algorithm, known for its ability to handle complex relationships and provide reliable predictions, is employed to train the model.

Additionally, the project focuses on analyzing a flights dataset to predict flight cancellations. The data preprocessing steps involve handling missing values, dropping irrelevant columns, and addressing multicollinearity. Missing values are imputed using strategies such as filling with median, mean, or specific calculations based on other columns. Irrelevant columns are dropped to focus on relevant features for analysis. Multicollinearity is addressed by removing highly correlated variables.

Exploratory data analysis is performed using visualizations, such as heat maps to visualize the correlation between variables and correlation matrices to analyze the relationships between features. The dataset is split into training and testing sets, and the features are standardized using 'Standard Scalar' from 'sklearn preprocessing'. Logistic regression is applied as a classification model to predict flight cancellations, and the model is evaluated using accuracy scores and a confusion matrix. Various other models, including random forest, linear regression, and decision tree, are also explored for predicting flight cancellations.

Through these diverse applications of machine learning and deep learning, this project demonstrates the versatility and power of these technologies in both healthcare and the airline industry. The integration of predictive models for diabetes, heart disease, flight prices, and flight cancellations underscores the potential of data-driven approaches to solve real-world problems and improve decision-making processes.

3. OBJECTIVE:

The primary objective of this project is to leverage machine learning and deep learning techniques to develop predictive models for diabetes and heart disease. These models aim to facilitate early detection and timely intervention, ultimately improving patient outcomes. The project also focuses on a comprehensive exploration and analysis of the datasets to uncover patterns and relationships that enhance the accuracy and reliability of the predictions.

1. Explore the Dataset:

Patterns and Data Types: Identify categorical and numerical variables, examine data quality, and detect outliers and duplicates. Distributions: Use frequency distributions, histograms, and box plots to understand the spread and central tendencies of the data. Relationships: Analyze correlations with a correlation matrix, scatter plots, bivariate analysis, and pair plots to uncover relationships between features and the target variables. Data Summary: Utilize descriptive statistics and analyze class balance to inform further data processing and modeling steps

2. Conduct Extensive Exploratory Data Analysis (EDA):

Exploratory Data Analysis (EDA) is a crucial step in understanding the dataset and uncovering relationships between variables. It involves examining the data through various statistical and visualization techniques to gain insights into its structure, distributions, and relationships. Here's how EDA can be conducted in detail, focusing on bivariate relationships against the target variables (diabetes and heart disease).By conducting extensive EDA, you can uncover meaningful insights about the dataset and understand how different variables relate to the target variables of interest (diabetes and heart disease). These insights serve as a foundation for further feature engineering, model selection, and ultimately, building accurate predictive models.

3. Pre-processing Steps:

Pre-processing is a critical phase in data preparation that ensures the data is clean, structured, and suitable for modeling. This step involves several techniques to enhance the quality and reliability of the data for building predictive models for both diabetes and heart disease datasets. Remove features that do not contribute significantly to the prediction of diabetes or heart disease. Irrelevant features can include identifiers, highly sparse variables, or variables with no discernible impact on the target variable. Pre-processing ensures that the dataset is cleaned, normalized, and structured in a way that maximizes the effectiveness of predictive modeling. By removing irrelevant features, addressing missing values, treating outliers, encoding categorical variables, and transforming skewed features, we prepare the data to be fed into machine learning algorithms. This structured approach enhances the model's ability to learn patterns and relationships, ultimately leading to more accurate predictions of diabetes and heart disease.

4. Model Building:

Model building is the stage where machine learning algorithms are implemented and tuned to predict outcomes based on pre-processed data. For both diabetes and heart disease prediction, establishing robust pipelines and selecting appropriate models are crucial steps in achieving accurate and reliable results.

Implement and tune various classification models, including K-Nearest Neighbours (KNN), Support Vector Machine (SVM), Decision Trees, and Random Forest for both diseases.

Model building involves implementing and optimizing machine learning algorithms to predict diabetes and heart disease based on pre-processed data. By establishing pipelines for consistent preprocessing, implementing and tuning various classification models, and emphasizing specific evaluation metrics tailored to each disease (high recall for heart disease and balanced approach for diabetes), the project aims to develop robust predictive models that can effectively identify individuals at risk and contribute to improved healthcare outcomes.

5. Evaluate and Compare Model Performance:

After building predictive models for diabetes and heart disease, it's crucial to assess their performance rigorously using appropriate metrics. This evaluation step helps in selecting the best-performing model and understanding its effectiveness in predicting the respective diseases. Utilize precision, recall, and F1-score to gauge the effectiveness of the models for both diabetes and heart disease. Compare the performance of different models to identify the most reliable and accurate predictors for each disease.

By systematically addressing these goals, the project aims to create robust and reliable predictive models for diabetes and heart disease. These models will aid healthcare professionals in identifying high-risk individuals early, enabling timely intervention and improving patient outcomes.

By accomplishing these objectives, the flight price prediction project aims to contribute to improved travel planning, budget optimization, and informed decision-making in the aviation industry. It strives to provide users with accurate and reliable predictions, empowering them to make well-informed choices and enhance their overall travel experience.

4. Tools and Technologies :

1. Python version 3.10 or Higher
2. Jupyter Notebook , VS-Code latest version
3. Python Data Processing , Visualization and Model building libraries
    - NumPy version 3.9.4
    - Pandas version 2.0.1
    - Scikit-Learn version 0.22.3
    - Plotly version 5.14
    - Seaborn version 0.12.2
    - Matplolib version 0.22.3
    - Keras and TensorFlow latest version 2.0
    - Flask version 2.3.2
4. OS Requirements
    - Windows 10 or higher version
    - Intel i5 8^{th} Gen Processor or higher
    - 8 GB RAM or Higher
    - Minimum 500 GB internal Storage or 200 or higher GB SSD
5. Parallel Computation/GPU Memory unit
    - NVIDIA GEFORCE 940 MX or higher for parallel computation
    - Intel iRISxe

5. DEPENDENCIES OF SYSTEM (Python Modules/Libraries):

OS: The OS (Operating System) module in Python provides a way of using operating system dependent functionality like reading or writing to the file system. In this project, it is used to access the files and data from internal storage.

Pandas: Pandas is an open-source library that is used for data manipulation and analysis. It provides data structures for efficiently storing and manipulating large datasets. In this project, Pandas is used to access and manipulate datasheets.

NumPy : NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices. In this project, NumPy is used for working on arrays and other data manipulation.

Streamlit: is a Python library designed for rapid development and deployment of interactive web applications. It simplifies the process of creating data-driven applications by allowing developers to write lightweight scripts that produce interactive user interfaces (UIs). Here are detailed aspects of Streamlit. Streamlit enables developers to create web applications using simple Python scripts, without requiring knowledge of HTML, CSS, or JavaScript. Streamlit empowers developers, data scientists, and engineers to create engaging and interactive web applications quickly and efficiently, making it a valuable tool for prototyping, data visualization, and deployment of data-driven solutions.

Pickle: The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

Matplotlib: matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of

things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).Sklearn: Scikit-learn is a Python library that is used for machine learning tasks like classification,regression, and clustering. In this project, we used Scikit-learn to make the model learn on variouscharacteristic values using logical regression. It provides a variety of tools for model selection, data pre-processing, model evaluation, and many other tasks related to machine learning.

These dependencies are used in the project to perform various tasks like data manipulation, GUI development, machine learning, and natural language processing. They make the project more efficient, scalable, and reliable. By using these dependencies, we can take advantage of the functionality they provide and build a better system.

6. METADATA:

6.1. Features

The dataset includes the following features:

Diabetes Disease:

1. **Pregnancies**: Number of times pregnant.
2. **Glucose**: Plasma glucose concentration after 2 hours in an oral glucose tolerance test.
3. **Blood Pressure**: Diastolic blood pressure (mm Hg).
4. **Skin Thickness**: Triceps skinfold thickness (mm).
5. **Insulin**: 2-Hour serum insulin (mu U/ml).
6. **BMI**: Body mass index (weight in kg/(height in m)^2).
7. **Diabetes Pedigree Function**: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history).
8. **Age**: Age of the patient in years.
9. **Outcome**: Diabetes outcome (0 = No diabetes, 1 = Diabetes present).

Heart Disease:

1. **Age**: Age of the patient in years.
2. **Sex**: Gender of the patient (0 = male, 1 = female).
3. **CP**: Chest pain type:
    a. 0: Typical angina
    b. 1: Atypical angina
    c. 2: Non-angina pain
    d. 3: Asymptomatic
4. **Trestbps**: Resting blood pressure in mm Hg.
5. **Chol**: Serum cholesterol in mg/dl.
6. **Fbs**: Fasting blood sugar level, categorized as above 120 mg/dl (1 = true, 0 = false).
7. **Restecg**: Resting electrocardiographic results:
    a. 0: Normal
    b. 1: Having ST-T wave abnormality
    c. 2: Showing probable or definite left ventricular hypertrophy
8. **Thalach**: Maximum heart rate achieved during a stress test.
9. **Exang**: Exercise-induced angina (1 = yes, 0 = no).
10. **Oldpeak**: ST depression induced by exercise relative to rest.
11. **slope**: Slope of the peak exercise ST segment:
    a. 0: Up sloping
    b. 1: Flat
    c. 2: Down sloping
12. **ca**: Number of major vessels (0-4) colored by fluoroscopy.

13. **thal**: Thallium stress test result:
    a. 0: Normal
    b. 1: Fixed defect
    c. 2: Reversible defect
    d. 3: Not described

14. **target**: Heart disease status:
    a. 0 = No disease
    b. 1 = Presence of disease

These variables describe different health indicators and factors related to diabetes risk and diagnosis. Analyzing these variables can help in understanding the relationships between various factors and the likelihood of diabetes development, as well as in building predictive models to identify individuals at risk of diabetes based on their characteristics.

7. APPROACH:

This documentation approach for developing the predictive models for diabetes and heart disease involves a structured methodology encompassing data preprocessing, exploratory data analysis (EDA), feature selection, model fitting, and hyper parameter tuning. Below is a detailed outline of each step:

Code Structure

The code is divided into several sections, each performing a specific task. The sections are as follows:

1. Importing the necessary libraries and modules.
2. Loading the dataset using pandas from an Excel file.
3. Exploratory Data Analysis (EDA) to understand the dataset.
4. Data pre-processing, including converting date and time features into usable formats and handling categorical data.
5. Feature selection using correlation analysis and feature importance.
6. Model fitting using the Random Forest algorithm.
7. Model evaluation and analysis of the results.
8. Hyperparameter tuning using RandomizedSearchCV.

Python Libraries: Utilizes NumPy, Pandas, Scikit-Learn, Matplotlib, Seaborn, and potentially other libraries depending on specific tasks and requirements.

Data Pre-processing

Data Cleaning: Addresses missing values, outliers, and ensures data integrity. Handling Missing Data: Address missing values using appropriate techniques such as imputation (mean, median, mode) or advanced methods like iterative imputation.
Dealing with Outliers: Identify and treat outliers that could skew the model's performance or affect predictions.
Feature Scaling: Normalize or standardize numerical features to ensure all features contribute equally to model training.
Categorical Encoding: Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding.
Feature Scaling: Standardizes or normalizes numerical features as needed for modelling.

Exploratory Data Analysis (EDA)

Summary Statistics: Computes descriptive statistics (mean, median, range, etc.) to understand data distribution and characteristics

.

Data Visualization: Uses Matplotlib and Seaborne for visual exploration of data through histograms, scatter plots, box plots, and correlation matrices.

Feature Selection

Techniques: Applies feature selection methods such as correlation analysis, feature importance from tree-based models, and recursive feature elimination (RFE).

Goal: Identifies the most influential features that contribute significantly to the prediction task while reducing dimensionality and improving model performance.

Model Fitting

Selection of Models: Implements various machine learning models suitable for the task (e.g., logistic regression, decision trees, random forest, and support vector machines).

Model Training: Uses Scikit-Learn to train models on the pre-processed data.

Model Evaluation: Assesses model performance using metrics such as accuracy, precision, recall, and F1-score. Evaluating the model's performance using various metrics such as mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and R-squared score.

Hyper parameter Tuning

Grid Search / Random Search: Optimizes model performance by tuning hyperparameters through exhaustive grid search or randomized search.

Cross-Validation: Uses techniques like k-fold cross-validation to ensure robustness of model evaluation and parameter selection.

Using RandomizedSearchCV to randomly search for the best hyper parameters.

Obtaining the best parameter values and the best score achieved by the model.

8. DATA COLLECTION:

For both the diabetes and heart disease prediction projects, the data collection process involves obtaining relevant datasets that contain information crucial for training and evaluating predictive models. Here's how data collection is approached:

1. Source Identification:

Identify reputable sources for healthcare and medical datasets that contain variables pertinent to diabetes and heart disease prediction. Examples include publicly available repositories, healthcare institutions, research publications, and curated datasets from reliable sources.

2. Dataset Selection:

Choose datasets that align with project objectives and contain variables such as demographics (age, gender), clinical measurements (blood pressure, cholesterol levels), medical history (family diabetes history, exercise-induced symptoms), and diagnostic results (ECG readings, insulin levels).Ensure datasets are sufficiently large, representative, and diverse to train robust models and generalize predictions.

3. Data Privacy and Ethics:

Adhere to data privacy regulations (e.g., GDPR, HIPAA) and ethical guidelines when accessing and handling medical data. Obtain necessary permissions and approvals for using sensitive health information and ensure anonymization or de-identification of personal data to protect patient privacy.

4. Data Acquisition:

Access datasets through official repositories or collaborate with healthcare providers and researchers to acquire anonymized data. Use APIs (Application Programming Interfaces) for accessing real-time health data streams if applicable

5. Data Preprocessing:

Cleanse and preprocess raw data to handle missing values, outliers, and inconsistencies. Standardize formats and ensure data quality through validation and verification processes.

6. Documentation and Attribution:

Document dataset sources, collection methods, and any transformations applied during preprocessing. Attribute datasets to their original sources and acknowledge data contributors appropriately in project documentation and publications.

Example Approach:

Diabetes Dataset: Obtain a dataset containing variables such as pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI, pedigree function, age, and diabetes outcome (0 for no diabetes, 1 for diabetes).

Heart Disease Dataset: Obtain a dataset with variables including age, gender, chest pain type, blood pressure, cholesterol levels, fasting blood sugar, ECG results, maximum heart rate, exercise-induced angina, ST depression, slope of ST segment, number of major vessels colored by fluoroscopy, thalium stress test results, and heart disease status (0 for no disease, 1 for presence of disease).

Summary:

Effective data collection forms the foundation for building accurate predictive models for diabetes and heart disease. By sourcing relevant, high-quality datasets and adhering to ethical guidelines, the project ensures robust model training and validation, contributing to reliable healthcare predictions and applications

9. EDA (EXPLORATORY DATA ANALYSIS)

For both the diabetes and heart disease prediction models, Exploratory Data Analysis (EDA) plays a fundamental role in understanding the datasets and preparing them for modeling. Here's how EDA applies specifically to each:

Diabetes Prediction Model:

1. Descriptive Statistics:
Calculate summary statistics like mean, median, and standard deviation for features such as glucose levels, BMI, and age. Understand the distribution of the target variable (diabetes outcome) to assess class imbalance.

2. Data Visualization:
Plot histograms for numerical variables like glucose levels and BMI to observe their distributions.
Use box plots to identify potential outliers in features such as insulin levels or pregnancies.
Create bar charts to visualize categorical variables like outcome (0 = no diabetes, 1 = diabetes).

3. Correlation Analysis:
Compute correlation coefficients to understand relationships between variables (e.g., correlation between BMI and insulin levels).
Visualize correlations using heat maps to identify significant associations among features.

4. Identifying Patterns:
Explore how variables like age, pregnancies, and glucose levels vary with the diabetes outcome.
Detect any patterns or trends that suggest risk factors or predictive indicators of diabetes.

5. Data Quality Checks:
Assess and handle missing data, ensuring robustness in feature engineering and model training.

Heart Disease Prediction Model:

1. Descriptive Statistics:
Calculate statistics for features such as resting blood pressure, cholesterol levels, and maximum heart rate achieved during stress tests.
Analyze the distribution of the target variable (heart disease status: 0 = no disease, 1 = presence of disease).

2. Data Visualization:
Plot histograms to visualize distributions of numerical features like cholesterol levels and blood pressure across different heart disease outcomes.
Use scatter plots to explore relationships between variables such as age and maximum heart rate, differentiated by heart disease status.

3. Exploring ECG Results:
Interpret resting electrocardiographic results (restecg) through visualization techniques to understand their impact on heart disease prediction.

4. Feature Importance:
Assess the importance of features like chest pain type (cp), exercise-induced angina (exang), and ST depression (oldpeak) in predicting heart disease status.
Utilize box plots or violin plots to compare these features across different categories of heart disease outcomes.

5. Data Anomalies and Preprocessing:
Identify outliers in features such as serum cholesterol or ST depression and decide on appropriate handling strategies.
Transform skewed features like ST depression using mathematical transformations to achieve normality.

Conclusion:
EDA for both diabetes and heart disease prediction models involves thorough exploration of datasets to uncover insights, patterns, and relationships among variables. By leveraging descriptive statistics, visualization techniques, and correlation analyses, EDA guides the preprocessing steps and informs the selection of features and models for building accurate predictive models. This approach ensures that the models are based on a deep understanding of the data, leading to more reliable predictions and insights into health outcomes.

9.1 Importance of EDA in Data Science

In the realm of Data Science, thorough exploratory data analysis (EDA) holds pivotal significance. It enables businesses to derive actionable insights and make informed decisions from vast datasets. By comprehensively exploring data from various angles, EDA unveils meaningful patterns and influential features essential for strategic decision-making. Thus, EDA occupies an indispensable role in maximizing the utility of data in Data Science applications.

9.1.1 Data Collection

In today's data-driven era, vast volumes of data are generated across diverse sectors such as healthcare, sports, manufacturing, and tourism. The collection of relevant data from sources like surveys, social media, and customer feedback is critical for businesses aiming to harness data effectively. This initial step is foundational as it provides the necessary raw material for subsequent analysis and decision-making processes. Without adequate and pertinent data collection, businesses cannot embark on meaningful data-driven activities.

9.1.2 Finding All Variables and Understanding Them

At the outset of the analysis, understanding the available data is paramount. This involves examining various features or characteristics that provide insights into their changing values and potential impacts. Identifying key variables that influence outcomes is crucial for achieving meaningful insights and desired analysis outcomes.

9.1.3 Cleaning the Dataset

Subsequent to variable identification, cleaning the dataset becomes essential. This process involves eliminating null values and irrelevant information, ensuring that only pertinent data remains for analysis. Effective data cleaning enhances efficiency by reducing computational demands and streamlining subsequent preprocessing tasks such as outlier detection and anomaly handling.

9.1.4 Identifying Correlated Variables

Exploring correlations between variables is pivotal for understanding their relationships and impacts. Utilizing correlation matrix methods provides clear visual insights into how different variables are interrelated. This understanding aids in uncovering significant associations and dependencies among variables, thereby informing further analysis and decision-making processes.

9.1.5 Choosing the Right Statistical Methods

Selecting appropriate statistical methods is crucial and depends on factors such as data type (categorical or numerical), size, variable types, and analysis objectives. Statistical formulas provide quantitative insights, while graphical representations offer intuitive visualizations that are easier to interpret. By employing the right statistical tools tailored to the data characteristics, analysts can derive robust conclusions and actionable insights from their analyses.

9.2 EDA ON DIABETES DATA:

Exploratory Data Analysis (EDA) on a diabetes dataset involves comprehensive steps to understand and prepare the data for modeling. Initially, data is loaded and inspected to check for completeness and correctness. Missing values are handled through imputation or deletion, and duplicates are removed to ensure data integrity. Univariate analysis examines the distribution and summary statistics of each feature, identifying outliers and anomalies. Bivariate analysis investigates relationships between pairs of variables, highlighting correlations and potential dependencies.

Multivariate analysis delves deeper into complex interactions between multiple variables, aiding in feature engineering to derive new predictors. Visualizations such as histograms, scatter plots, and heatmaps provide graphical insights into data patterns and correlations. Addressing class imbalance and documenting decisions ensure the dataset is suitable for machine learning algorithms. EDA concludes with a summary of findings, guiding subsequent steps in data preprocessing and model development. Overall, EDA is pivotal in uncovering insights, validating assumptions, and preparing data for accurate and robust predictive modeling in diabetes research or healthcare applications.

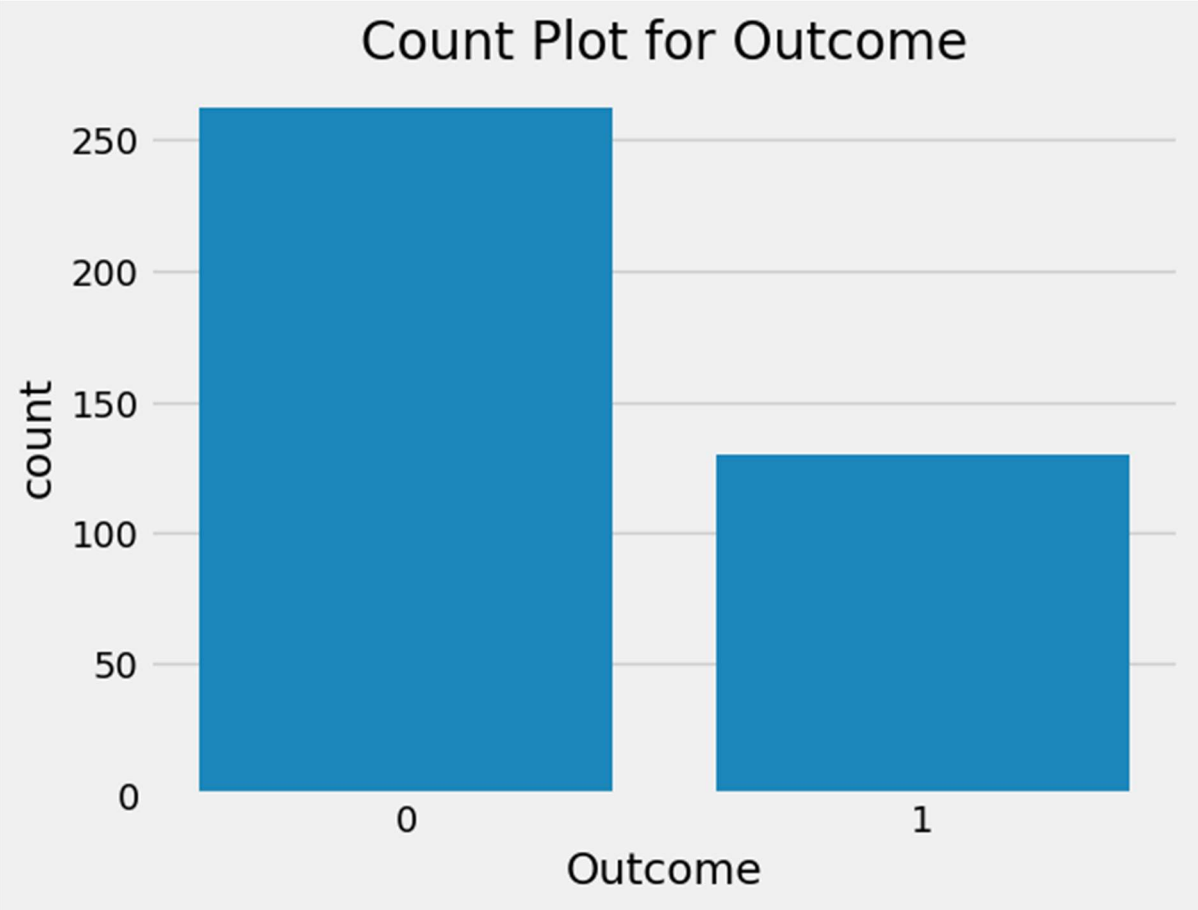


Fig.9.1. Count Plot for Outcome

- There are **66.8%** 1's (diabetic) and **33.1%** 0's (nondiabetic) in the data.

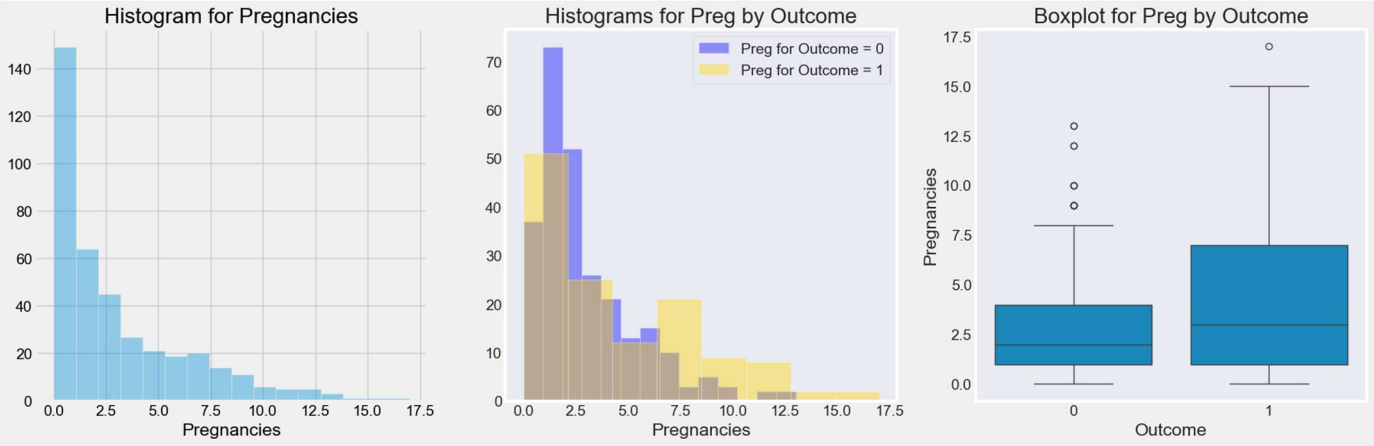


Fig.9.1. Count Plot for Outcome

From visuals we can say:

- Data is **right skewed.** For data of count of pregnancies.

- A large proportion of the participants are **zero count on pregnancy.** As the data set includes women > 21 yrs., it's likely that many are unmarried.
- When looking at the segmented histograms, **a hypothesis is the as pregnancies includes, women are more likely to be diabetic.**
- In the boxplots, we find few outliers in both subsets. **Some non-diabetic women have had many pregnancies.**
- To validate this hypothesis, need to **statistically test.**

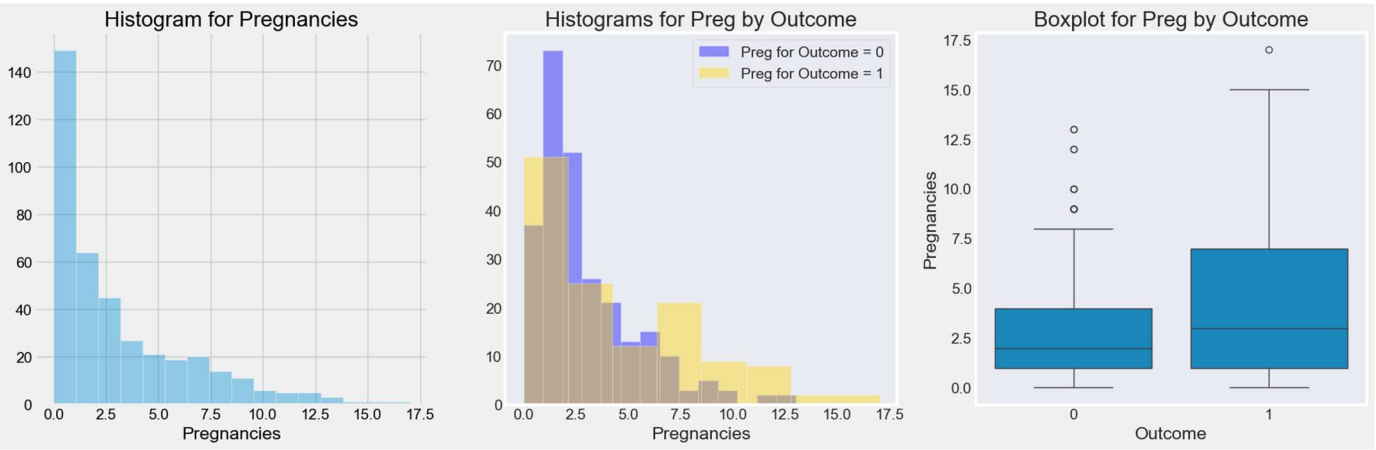


Fig.9.1. Count Plot for Outcome

From visuals we got:

- **1st graph** –-> Histogram of Glucose data is **slightly skewed to right.** Understandably, the data set contains over 60% who are diabetic and it's likely that their Glucose levels were higher.
- **2nd graph** –-> clearly diabetic group has **higher glucose** than non-diabetic.
- **3rd graph** –-> In the boxplot, visually skewness seems acceptable (<2) and it's also likely that confidence intervals of the means are not overlapping. So a hypothesis that Glucose is measure of outcome, is likely to be true. But needs to be statistically tested too.

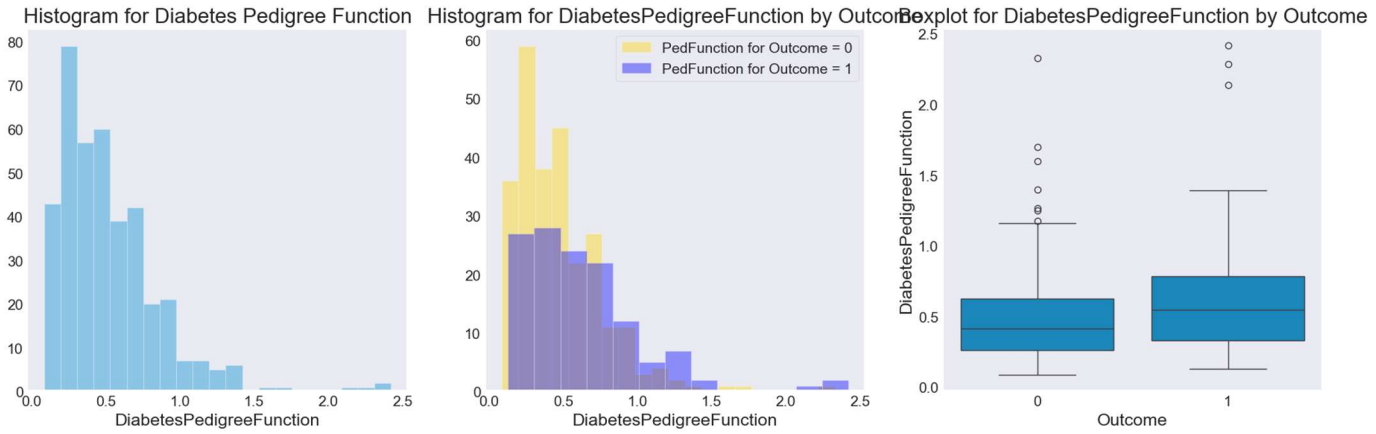


Fig.9.1. Count Plot for Outcome

From Visuals we got:

- This variable calculates diabetes likelihood depending on the subject's age her diabetic family history
- Data is skewed. There seems to be a likelihood of being diabetic, but needs statistical validation

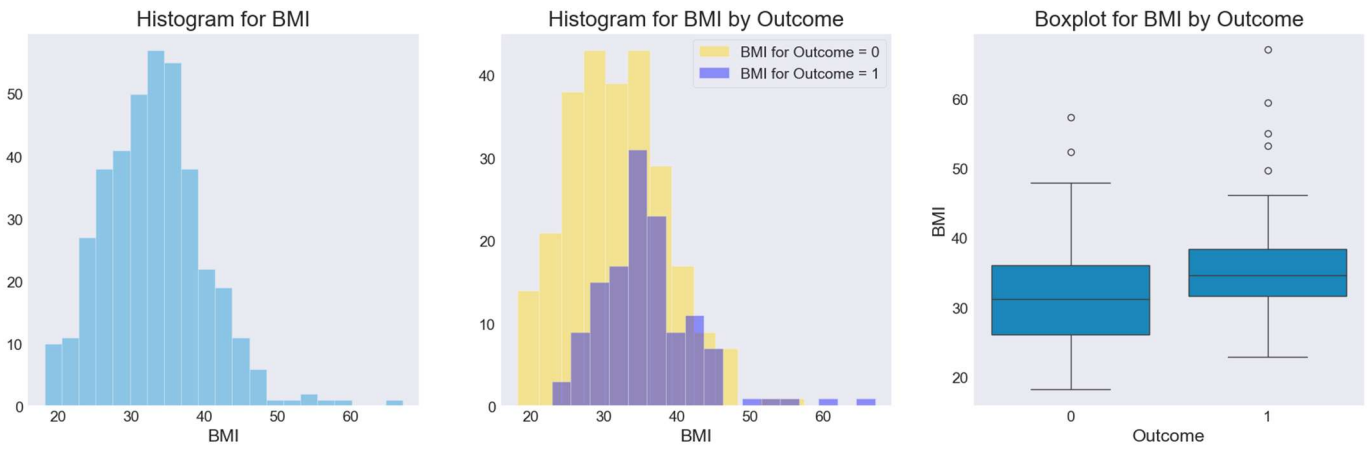


Fig.9.1. Count Plot for Outcome

From visuals we got:

- **1st graph** – There are few outliers. Few are obese in the dataset. Expected range is between 18 to 25. In general, people are obese
- **2nd graph** – Diabetic people seems to be only higher side of BMI. Also the contribute more for outliers
- **3rd graph** – Clearly there are Outliers in the data. These Outliers are concern for us and most of them with higher insulin values are also diabetic. So this is a suspect.

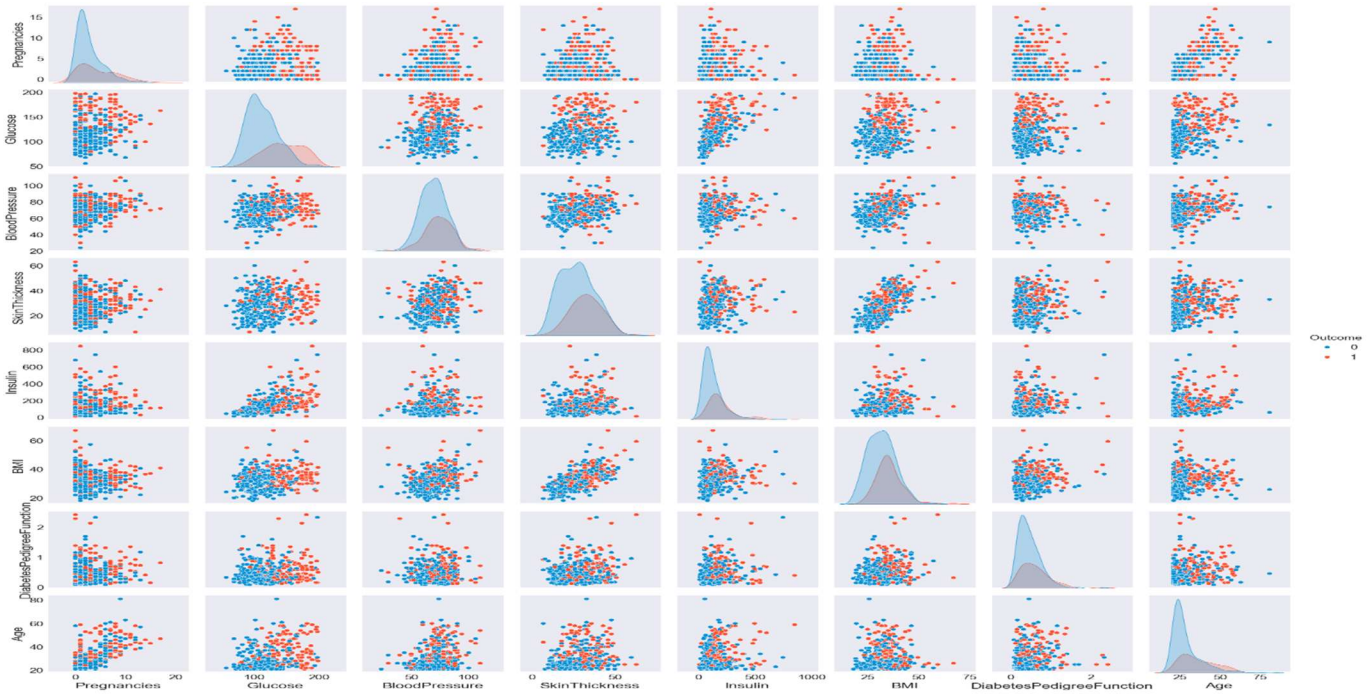


Fig.9.1. Count Plot for Outcome

From the pairplot we got:

- From scatter plots, only BMI, Skin Thickness, Pregnancies & Age seem to have positive linear relationships. Another likely suspect is Glucose and Insulin.
- There are no non-linear relationships
- We will check it out with Pearson Correlation and plot heat maps

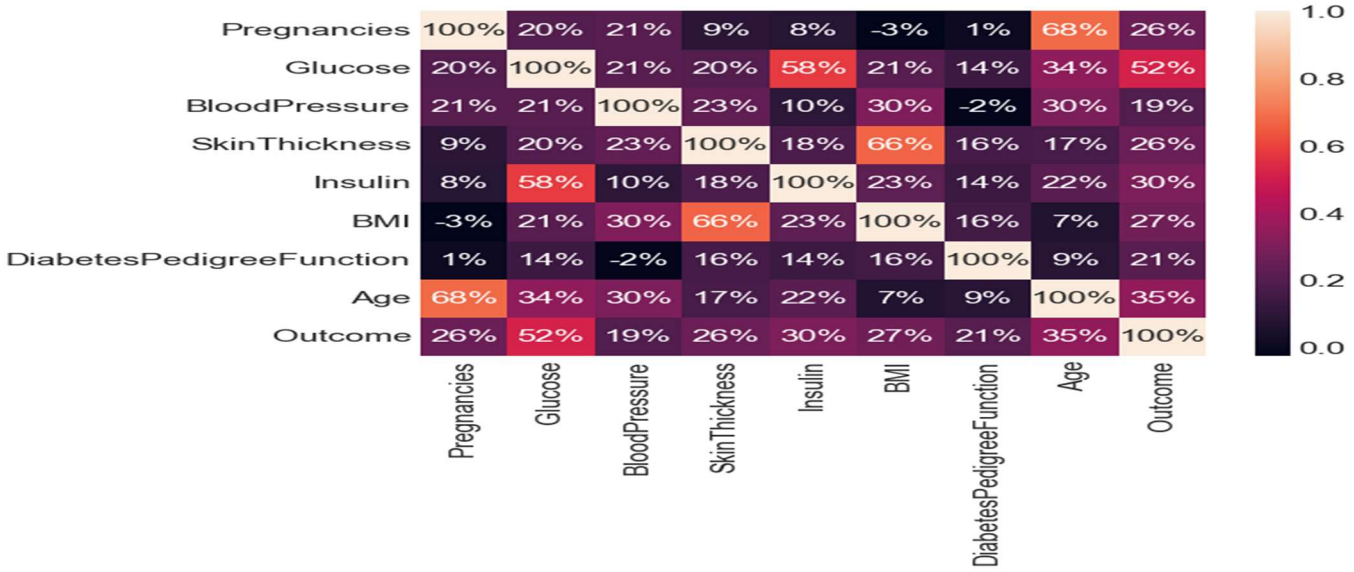


Fig.9.1. Count Plot for Outcome

Age & Pregnancies and BMI & Skin Thickness have moderate positive linear relationship Glucose & Insulin technically has low correlation but 0.58 is close to 0.6 so can be assumed as moderate correlation

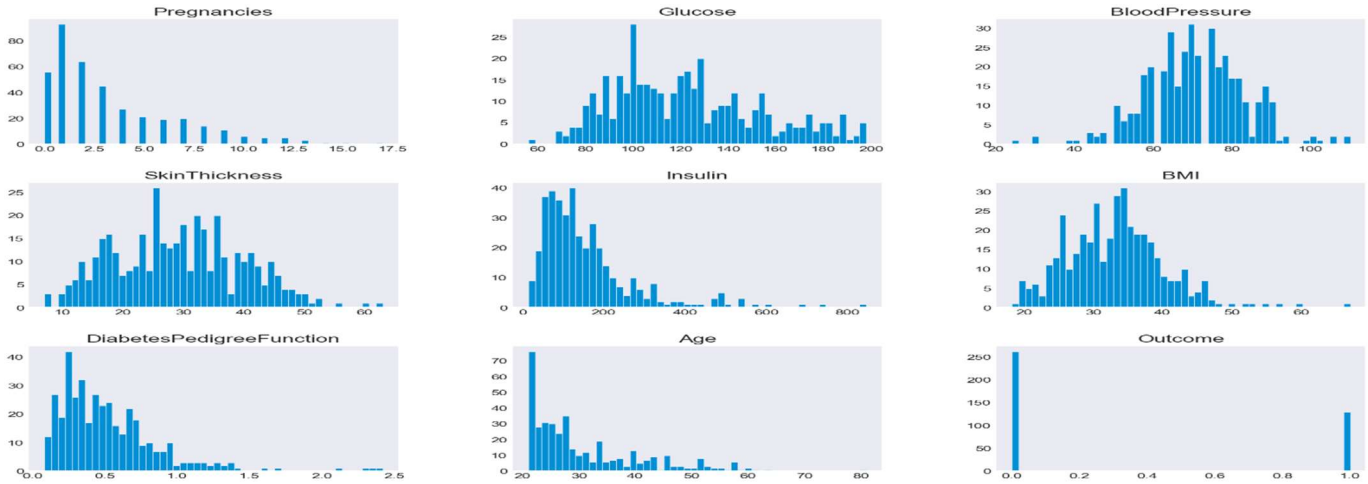


Fig.9.1. Count Plot for Outcome

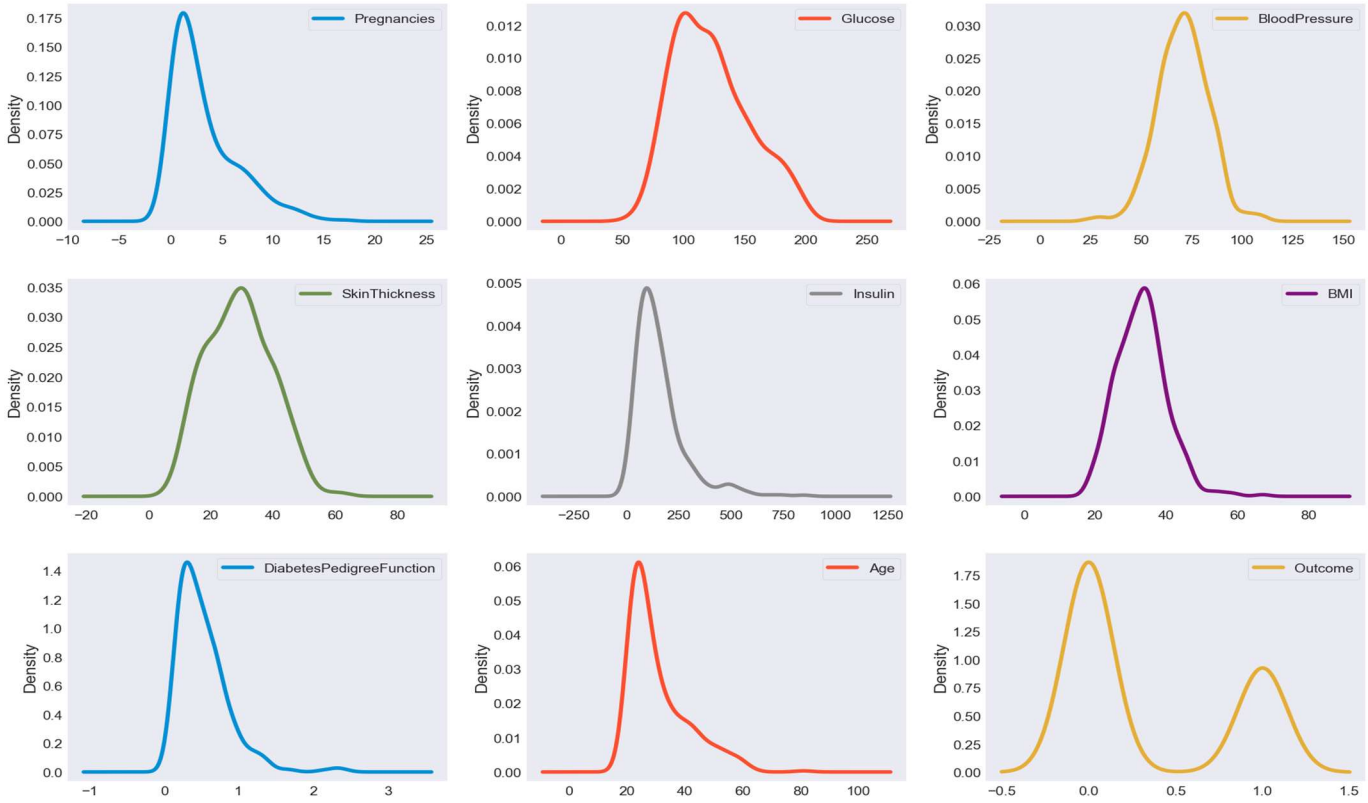


Fig.9.1. Count Plot for Outcome

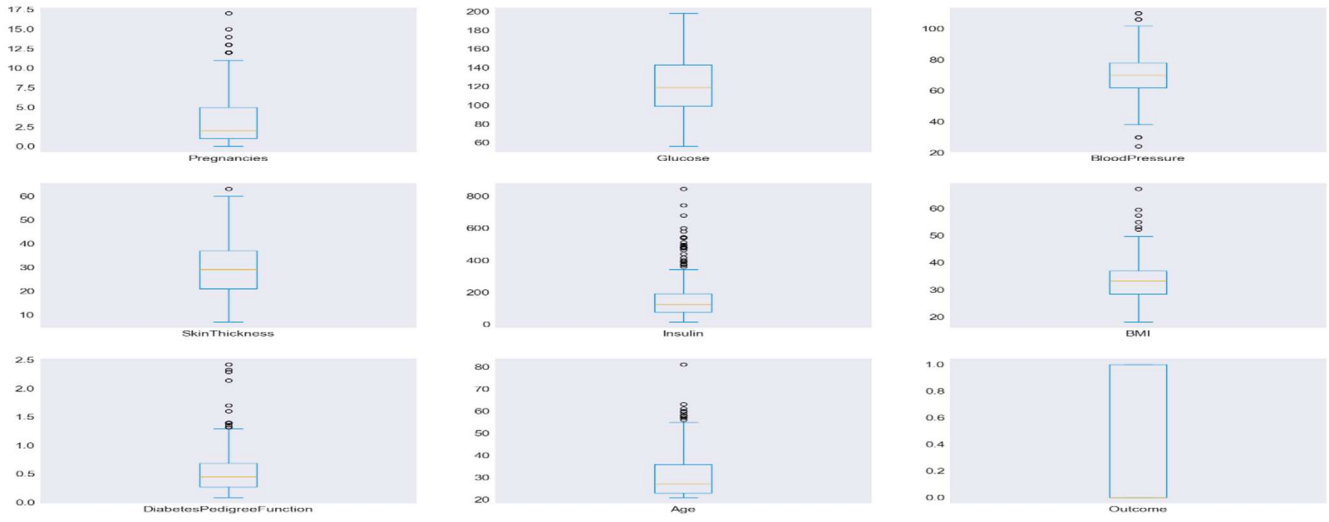


Fig.9.1. Count Plot for Outcome

Bell shape curve: Blood Pressure

- **Right-Skewed: Age, Insulin, Pregnancies, Diabetes Pedigree Function**
- **Short IQR: insulin, Diabetes Pedigree Function, Blood Pressure and BMI**
- **At least 75% of the women:**
- are 25 years old or older
- have BMI nearly 30 kg/m2
- have insulin level 100 or more
- have 1 or more pregnancies

- have glucose level of 100 mg/dL or more
- have blood pressure of 60 mmHg or more

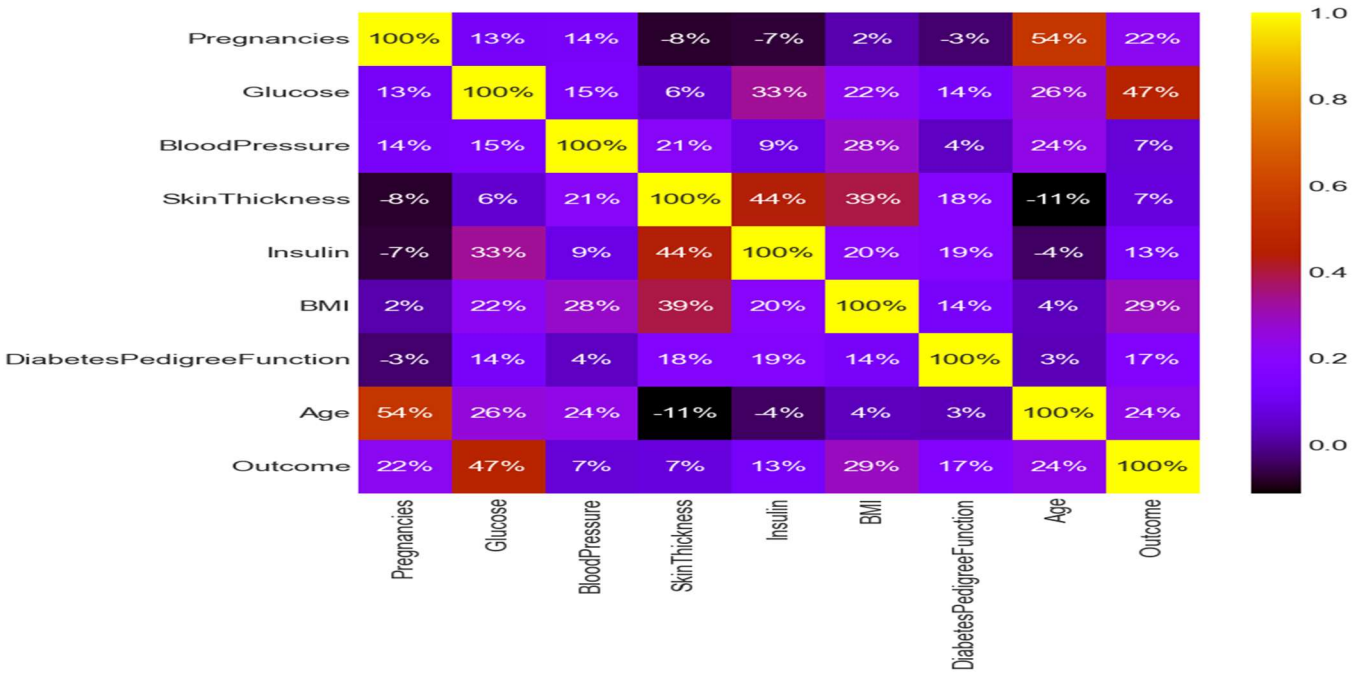


Fig.9.1. Count Plot for Outcome

There are no strong correlation between the features

The 'strongest' ones are the following (as expected):

- Age x pregnancies (0.68) - Older women tend to have higher number of pregnancies
- Glucose x insulin (0.58)
- Glucose x outcome (0.52) - Women that have higher level of glucose tend to have higher level of insulin and have DM
- Skin fold thickness x BMI (0.66) - Women with higher skin fold thickness value have higher BMI (and probably are overweight/obese)

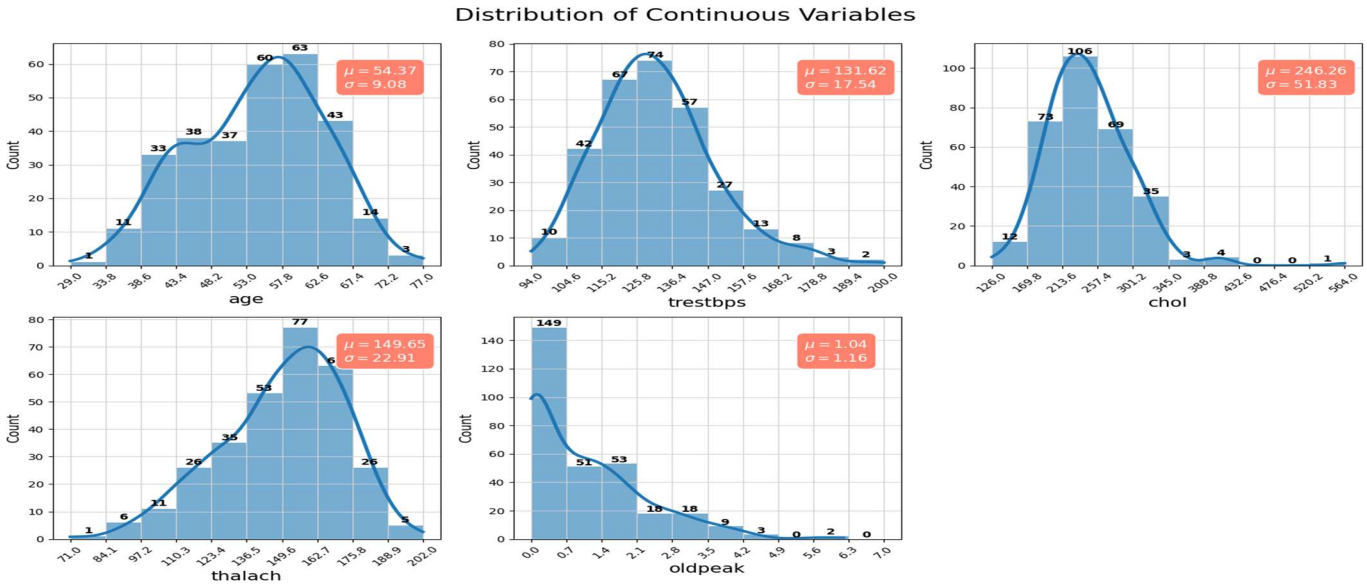
Negative correlation:

- BMI x Pregnancies (-0.025)
- Blood Pressure x Diabetes Pedigree Function (-0.016)

- **Diabetic women tend to show larger values of age, BMI, insulin, skin thickness, blood pressure, and pregnancies.**
- **The feature that it is possible to see 2 distinct groups (diabetic and non diabetic) is glucose.**

9.2 EDA ON HEART DISEASES DATA:

Exploratory Data Analysis (EDA) for heart disease data involves comprehensive steps to understand and prepare the dataset for modeling. Initially, the data is loaded, inspected for completeness, and cleaned by handling missing values and removing duplicates. Univariate analysis examines distributions and summary statistics of variables like cholesterol levels or blood pressure, identifying outliers and assessing data skewness. Bivariate analysis explores correlations between factors such as age and heart disease presence, using scatter plots and correlation matrices.
Multivariate analysis delves into interactions among multiple variables to uncover complex relationships, aiding in feature engineering for predictive modeling. Visualizations such as histograms, box plots, and heat maps offer graphical insights into data patterns and relationships. Addressing class imbalance ensures balanced representation of heart disease cases for robust model training. EDA concludes with summarizing key findings, guiding subsequent steps in data preprocessing and model development to enhance understanding and prediction of heart disease risks.



Inferences:

- **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.

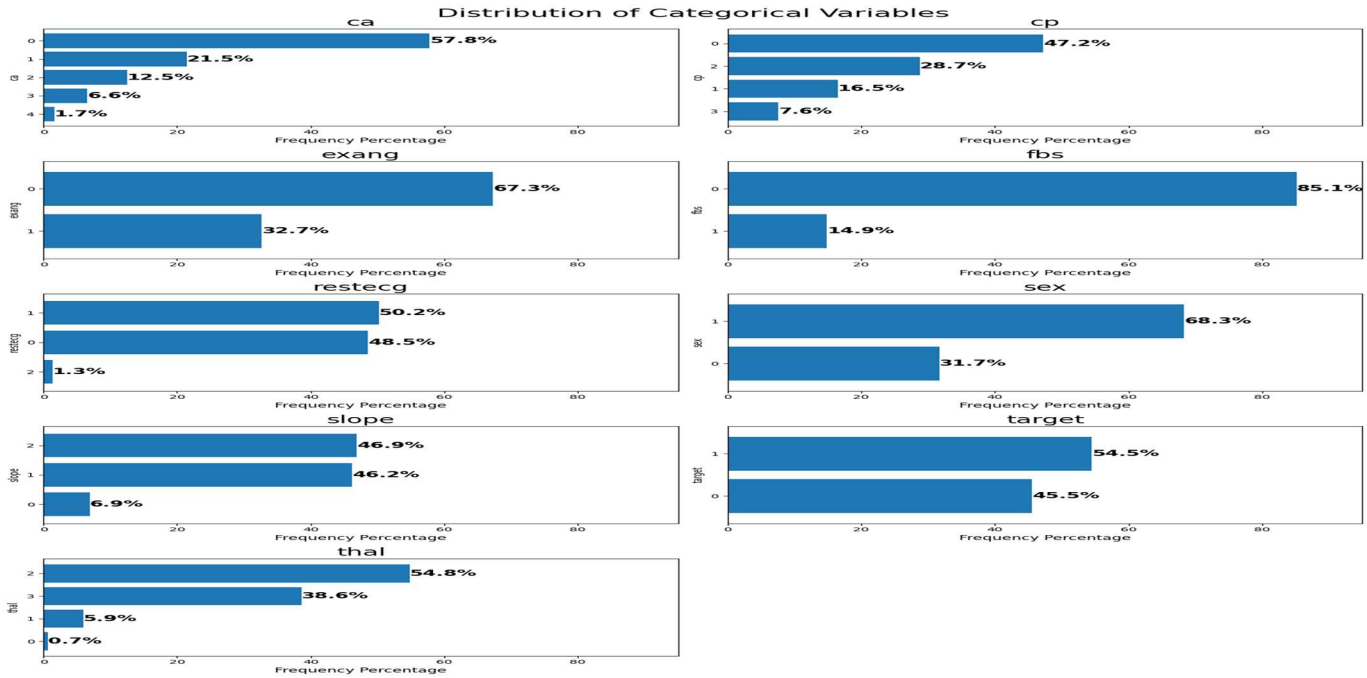
- **Resting Blood Pressure (`trestbps`)**: The resting blood pressure for most individuals is concentrated around 120-140 mm Hg, with a mean of approximately 131.62 mm Hg and a standard deviation of 17.54 mm Hg.

- **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.

- **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.

- **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. **There doesn't seem to be any noticeable noise or implausible values among the continuous variables.**



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

Type of Chest Pain (cp): There are varied types of chest pain among patients, with Type 0 (Typical angina) being the most prevalent, but the exact distribution among types can be inferred from bar plots.

Fasting Blood Sugar (fbs): A significant majority of patients have fasting blood sugar levels below 120 mg/dl, indicating low prevalence of high blood sugar in this dataset.

Resting Electrocardiographic Results (restecg): There are diverse resting electrocardiographic outcomes, with certain types more common than others. Detailed distribution can be understood from the plots.

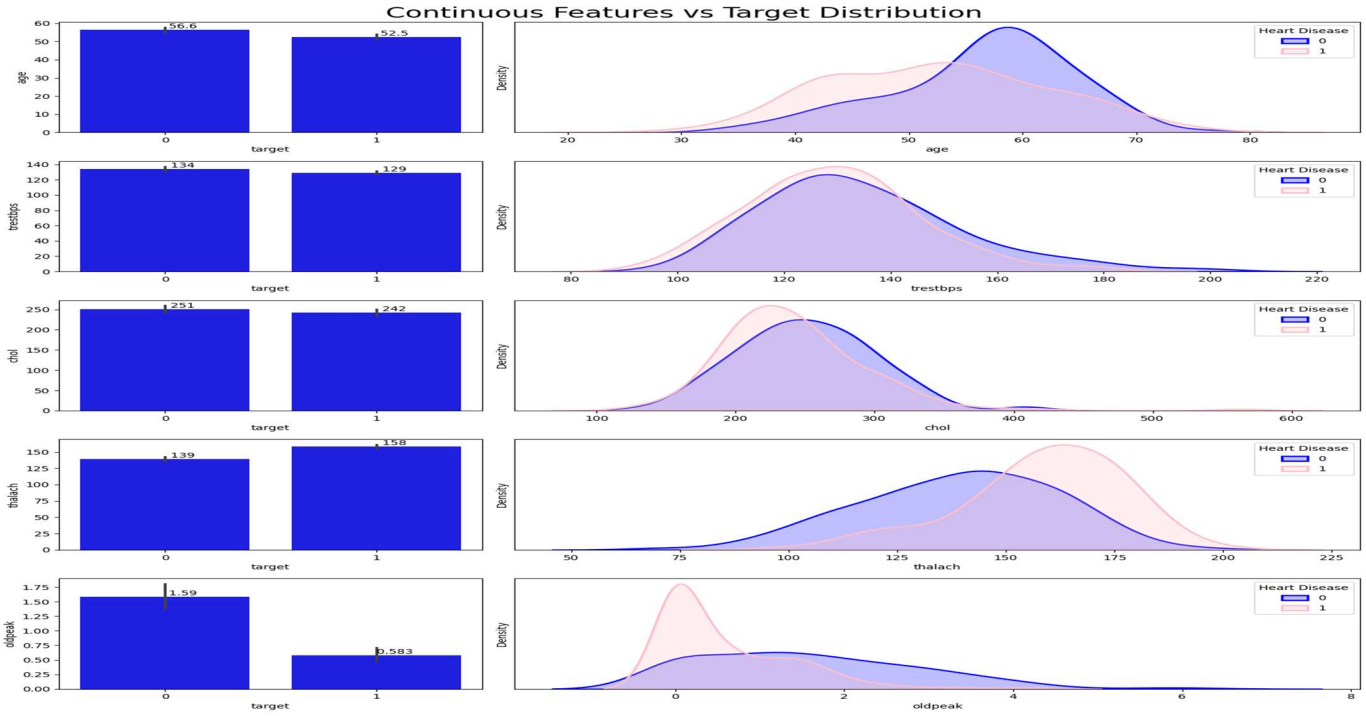
Exercise-Induced Angina (exang): Most patients do not experience exercise-induced angina, suggesting it's uncommon in this dataset.

Slope of the Peak Exercise ST Segment (slope): Different slopes of the peak exercise ST segment are observed, with specific types more prevalent, detailed in bar plots.

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

Thalium Stress Test Result (thal): There's a variety of thalium stress test results, with one type more prevalent, detailed in the plots.

Presence of Heart Disease (target): The dataset is nearly balanced regarding heart disease presence, with approximately 54.5% affected and 45.5% not affected.



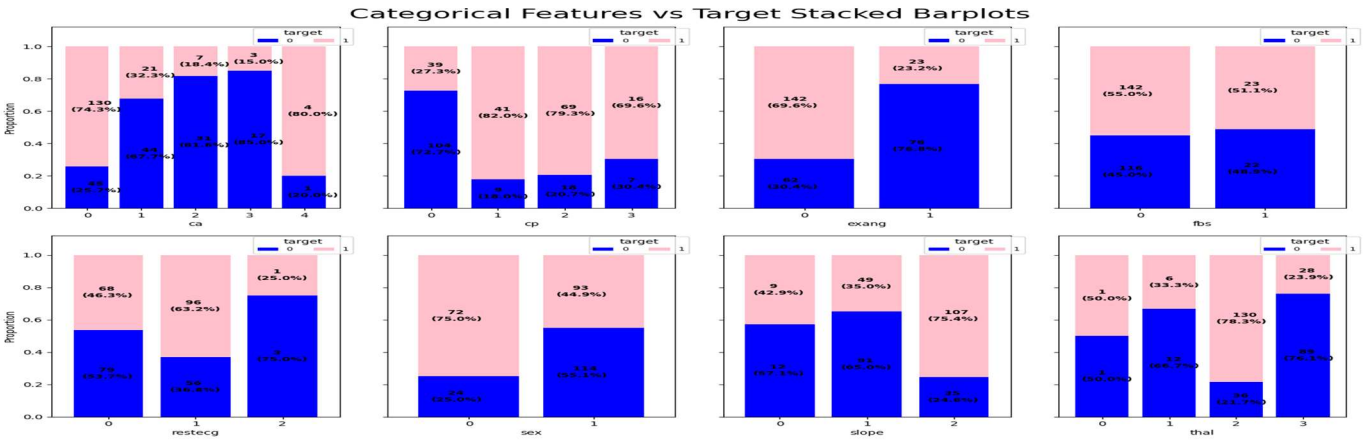
Inferences:

- **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.
- **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.
- **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.

- **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.

- **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`)**.



Inferences:

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,

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.

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).

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.

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.

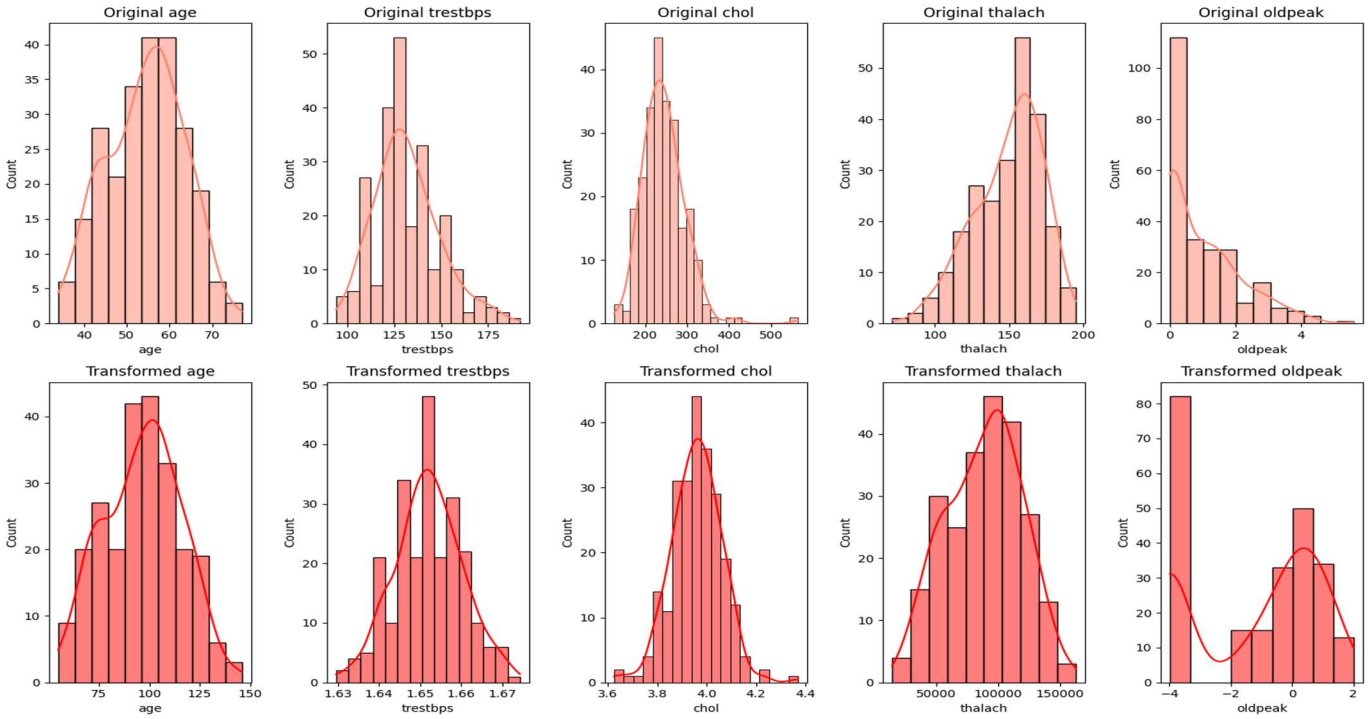
Sex (`sex`): Females (1) exhibit a lower proportion of heart disease presence compared to males (0).

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.

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**



Inference:

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

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

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

4- `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.

5- `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.

Conclusion:

Transforming features to be more normal-like primarily helps in mitigating the impact of outliers, which is particularly beneficial for distance-based algorithms like **SVM** and **KNN**. By reducing the influence of **outliers**, we ensure that

these algorithms can compute distances more effectively and produce more reliable results.

10. MACHINE LEARNING

1. Machine Learning Model:

Machine learning models are algorithms designed to identify patterns and make decisions based on data. They learn from historical data and can generalize to make predictions on new, unseen data. These models can be broadly categorized into supervised learning (where the model is trained on labeled data), unsupervised learning (where the model identifies patterns without labeled responses), and reinforcement learning (where the model learns through trial and error).

2. Decision Tree:

A decision tree is a simple yet powerful supervised learning algorithm used for both classification and regression tasks. It splits the data into subsets based on the value of input features, creating a tree-like structure of decisions. Each node represents a feature, each branch represents a decision rule, and each leaf represents an outcome. Decision trees are easy to interpret but can be prone to overfitting.

3. Random Forest:

Random Forest is an ensemble learning method that combines multiple decision trees to improve the model's accuracy and robustness. Each tree in the forest is trained on a random subset of the data and a random subset of features. The final prediction is made by aggregating the predictions of all the trees, typically through majority voting for classification or averaging for regression. This reduces the risk of overfitting and enhances generalization.

4. K-Nearest Neighbors (KNN):

KNN is a non-parametric, instance-based learning algorithm used for classification and regression. It works by identifying the k-nearest neighbors of a new data point based on a distance metric (e.g., Euclidean distance) and making predictions based on the majority class (for classification) or the average value (for regression) of these neighbors. KNN is simple and effective but can be computationally expensive and sensitive to the choice of k and the distance metric.

5. Support Vector Machine (SVM):

SVM is a powerful supervised learning algorithm primarily used for classification tasks. It works by finding the hyperplane that best separates the data into different classes, maximizing the margin between the closest points (support vectors) of each class. SVM can handle both linear and non-linear classification through the use of kernel functions, which transform the input data into a higher-dimensional space where a linear separator can be found.

6. Logistic Regression:

Logistic regression is a statistical model used for binary classification tasks. It models the probability that a given input belongs to a particular class using a logistic function. The output is a probability value between 0 and 1, which is then thresholded to assign a class label. Logistic regression is simple, interpretable, and effective for linearly separable data but may struggle with complex non-linear relationships.

7. Gradient Boosting Model:

Gradient Boosting is an ensemble learning technique that builds models sequentially, each new model correcting the errors of the previous ones. It combines weak learners, typically decision trees, to form a strong predictive model. The algorithm optimizes a loss function by adding models that reduce the residual errors of the previous models. Gradient Boosting is highly effective for both classification and regression tasks but can be computationally intensive and prone to overfitting if not properly regularized.

8. Multilayer Perceptron (MLP) Classifier:

An MLP is a type of artificial neural network used for supervised learning tasks. It consists of multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer. Each neuron applies a weighted sum of its inputs followed by a non-linear activation function. MLPs are capable of capturing complex non-linear relationships in the data. Training an MLP involves optimizing the weights using backpropagation. MLPs are versatile and powerful but require careful tuning of hyperparameters and can be computationally demanding.

11. MACHINE LEARNING MODEL

11.1. MACHINE LEARNING MODEL OF DIABETES DATA

11.1.1 KNN Algorithm

5.1 : K-Nearest Neighbors (KNN)

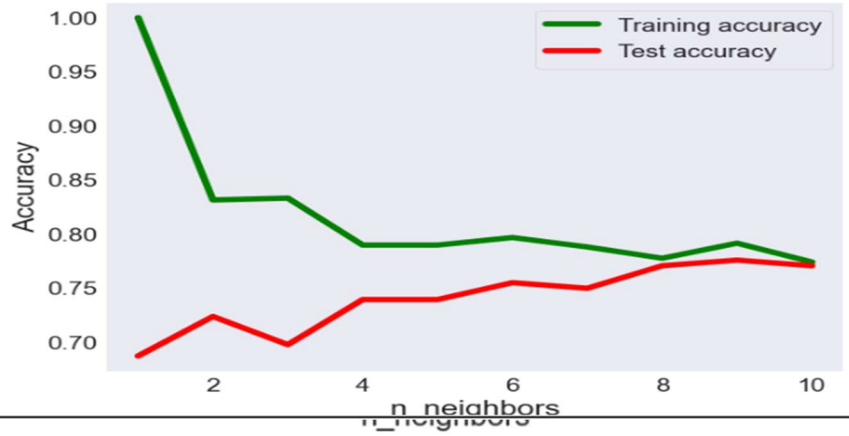
The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its "nearest neighbors."

In [77]:
```
cols = ["Pregnancies", "Glucose","BloodPressure","SkinThickness","Insulin", "BMI","DiabetesPedigreeFunction", "Age"]
X = df[cols]
y = df.Outcome
```

In [78]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.loc[:, df.columns != 'Outcome'],
                                                    df['Outcome'], stratify = df['Outcome'], random_state=66)
```

In [79]:
```
from sklearn.neighbors import KNeighborsClassifier

training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    # build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # record test set accuracy
```

In [80]:
```
# Draw Plot for Training data Accuracy and test Data Accuracy
plt.plot(neighbors_settings, training_accuracy, label="Training accuracy",color = "green")
plt.plot(neighbors_settings, test_accuracy, label="Test accuracy",color = "r")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```



- If we choose one single nearest neighbor, the prediction on the training set is perfect.
- But when more neighbors are considered, the training accuracy drops, indicating that using the single nearest neighbor leads to a model that is too complex.
- The best performance is somewhere around 9 neighbors.

> The above plot suggests that we should choose n_neighbors = 9

In [81]:
```
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train, y_train)

print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.79
Accuracy of K-NN classifier on test set: 0.78
```

In [82]:
```
pickle.dump(knn, open("../model/knn_Model","wb"))
```

11.1.1 Logistic Regression Model

5.2: Logistic Regression Model

Logistic regression is one of the most common classification algorithms.

In [83]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression().fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set accuracy: 0.786
Test set accuracy: 0.771
```

The default value of C = 1 provides with 78% accuracy on training and 77% accuracy on test set.

> The C parameter controls the trade-off between achieving a low training error and a low testing error. A smaller value of C creates a wider margin hyperplane that may lead to more training errors but better generalization to the test set, while a larger value of C creates a narrower margin hyperplane that may result in fewer training errors but worse performance on the test set.

In [84]:
```
logreg001 = LogisticRegression(C = 0.01).fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
Training set accuracy: 0.762
Test set accuracy: 0.760
```

Using C = 0.01 results in lower accuracy on both the training and the test sets.

In [85]:
```
logreg100 = LogisticRegression(C = 100).fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg100.score(X_test, y_test)))
```

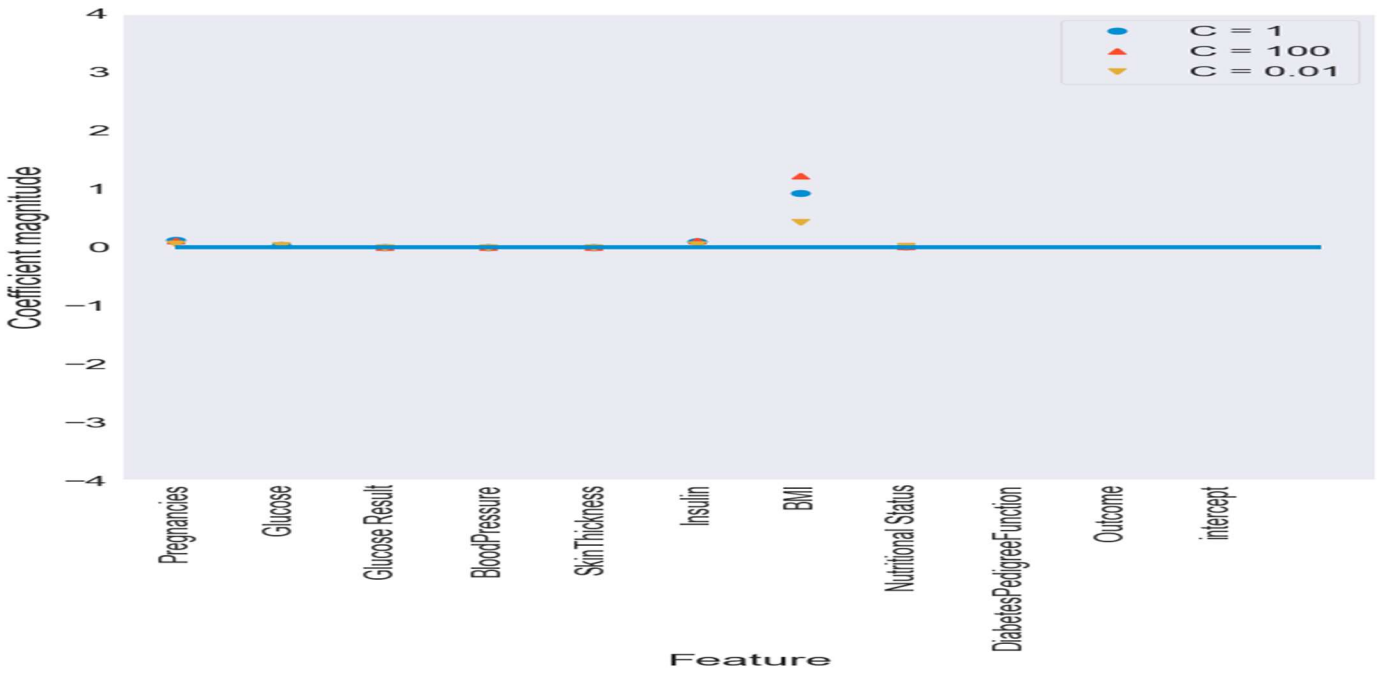
```
Training set accuracy: 0.783
Test set accuracy: 0.781
```

Using C = 100 results in a little bit higher accuracy on training set little bit lower accuracy on test set, confirming that less regularization and a more complex model may not generalize better than default setting.

> **Therefore, we should choose default value C = 1**

Finally, let's look at the coefficients learned by the models with the three different settings of the regularization parameter C.

Stronger regularization (C = 0.01) pushes coefficients more and more toward zero. Inspecting the plot more closely, we can also see that feature "DiabetesPedigreeFunction", for C = 100, C = 1 and C = 0.01, the coefficient is positive. This indicates that high "DiabetesPedigreeFunction" feature is related to a sample being "diabetes", regardless which model we look at.



11.1.1 DECISION TREE MODEL

5.3: Decision Tree Model

In [87]:
```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(random_state = 0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.714
```

The accuracy on the training set is 100%, while the test set accuracy is much worse.

> This is an indicative that the tree is overfitting and not generalizing well to new data. Therefore, we need to apply pre-pruning to the tree

We set max_depth = 3, limiting the depth of the tree decreases overfitting.

> This leads to a lower accuracy on the training set, but an improvement on the test set.

In [88]:
```
tree = DecisionTreeClassifier(max_depth = 3, random_state = 0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.773
Accuracy on test set: 0.740
```

5.3.1 Feature importance in Decision trees

Feature importance rates how important each feature is for the decision a tree makes.
It is a number between 0 and 1 for each feature, where 0 means "not used at all" and 1 means "perfectly predicts the target." The feature importances always sum to 1:

]:
```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

```
Feature importances:
[0.04554275 0.6830362  0.         0.         0.         0.27142106
 0.         0.        ]
```

11.1.1 RANDOM FOREST MODEL

5.4: Random Forest Model:

We will apply a random forest consisting of 100 trees on the diabetes dataset:

In [91]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 100, random_state = 0)
rf.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(rf.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(rf.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.786
```

The random forest model gives us an accuracy of 78.6%, better than the logistic regression model or a single decision tree, without tuning any parameters.

> we can adjust the max_features setting, to see whether the above result can be improved or not.

In [92]:
```
rf1 = RandomForestClassifier(max_depth = 3, n_estimators = 100, random_state = 0)
rf1.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(rf1.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(rf1.score(X_test, y_test)))
```

```
Accuracy on training set: 0.800
Accuracy on test set: 0.755
```

11.1.1 GRADIENT BOOSTING MODEL

5.5 : Gradient Boosting Model

In [94]:
```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(random_state=0)
gb.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gb.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gb.score(X_test, y_test)))
```

```
Accuracy on training set: 0.917
Accuracy on test set: 0.792
```

This is an indicative that the Gradient Boosting is overfittingTo reduce overfitting, we could either apply stronger pre-pruning by limiting the maximum depth or lower the learning rate:

Method 1 : by limiting the maximum depth:

In [95]:
```
gb1 = GradientBoostingClassifier(random_state=0, max_depth=1)
gb1.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gb1.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gb1.score(X_test, y_test)))
```

```
Accuracy on training set: 0.804
Accuracy on test set: 0.781
```

Method 2 : by lower the learning rate:

[96]:
```
gb2 = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gb2.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gb2.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gb2.score(X_test, y_test)))
```

```
Accuracy on training set: 0.802
Accuracy on test set: 0.776
```

Both methods of decreasing the model complexity reduced the training set accuracy, as expected. In this case, none of these methods increased the generalization performance of the test set.

Now Lets visualize the feature importances to get more insight into our model even though we are not really happy with the model.

11.1.1 SUPPORT VECTOR MACHINE MODEL

5.6 : Support Vector Machine Model

In [98]:
```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

```
Accuracy on training set: 0.77
Accuracy on test set: 0.76
```

The model overfits quite substantially, with a perfect score on the training set and only 65% accuracy on the test set.

> SVM requires all the features to vary on a similar scale. We will need to rescale our data that all the features are approximately on the same scale

Rescale All Features

In [99]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

svc = SVC()
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.79
Accuracy on test set: 0.80
```

Scaling the data made a huge difference!

> Now we are actually in an underfitting, where training and test set performance are quite similar but less close to 100% accuracy. From here, we can try increasing either C or gamma to fit a more complex model.

In [100]:
```
# Lets Try to increasing C and check the result.
svc = SVC(C = 1000)
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.944
Accuracy on test set: 0.724
```

Here, increasing C allows us to improve the model, resulting in 79.7% test set accuracy.

In [108]:
```
pickle.dump(svc, open("../model/svc","wb"))
```

11.1.1 NEURAL NETWORKS MODEL

5.7: Neural Networks Model

In [101]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(mlp.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(mlp.score(X_test, y_test)))
```

```
Accuracy on training set: 0.73
Accuracy on test set: 0.72
```

The accuracy of the MLP is not as good as the other models at all, this is likely due to scaling of the data. Neural networks also expect all input features to vary in a similar way, and ideally to have a mean of 0, and a variance of 1.

We must rescale our data so that it fulfills these requirements.

In [102]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.823
Accuracy on test set: 0.802
```

In [109]:
```
pickle.dump(mlp, open("../model/best_mlp","wb"))
```

The results are much better after scaling. As at matter of fact, we have obtained the highest test accuracy so far.

11.2. MACHINE LEARNING MODEL ON HEART DISEASE

11.2.1 DECISION TREE MODEL

Step 6 | Decision Tree Model Building

Tabel of Contents

Step 6.1 | DT Base Model Definition

First, let's define the base DT model:

In [25]:
```
# Define the base DT model
dt_base = DecisionTreeClassifier(random_state=0)
```

Step 6.2 | DT Hyperparameter Tuning

Note: In medical scenarios, especially in the context of diagnosing illnesses, it's often more important **to have a high recall (sensitivity) for the positive class (patients with the condition)**. A high recall ensures that most of the actual positive cases are correctly identified, even if it means some false positives (cases where healthy individuals are misclassified as having the condition). The rationale is that it's generally better to have a few false alarms than to miss out on diagnosing a patient with a potential illness.

I am establishing a function 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:

In [26]:
```
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:

In [27]:
```
# 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]
}
```

In [28]:
```
# Call the function for hyperparameter tuning
best_dt, best_dt_hyperparams = tune_clf_hyperparameters(dt_base, param_grid_dt, X_train, y_train)
```

In [29]:
```
print('DT Optimal Hyperparameters: \n', best_dt_hyperparams)
```

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

Step 6.3 | DT Model Evaluation

Now let's evaluate our DT model performance on both the training and test datasets:

In [30]:
```
# 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
```

In [31]:
```
# 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 doesn't appear to be overfitting.

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

In [32]:
```
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
```

In [33]:
```
dt_evaluation = evaluate_model(best_dt, X_test, y_test, 'DT')
dt_evaluation
```

Out[33]:

	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

In [34]:
```
import pickle
```

In [35]:
```
pickle.dump(best_dt, open("../model/best_dt","wb"))
```

11.2.2. RANDOM FOREST MODEL

Step 7.1 |< RF Base Model Definition

First, let's define the base RF model:

In [36]:
```
rf_base = RandomForestClassifier(random_state=0)
```

Step 7.2 | RF Hyperparameter Tuning

Afterward, I am setting up the hyperparameters grid and utilize the **tune_clf_hyperparameters** function to pinpoint the optimal hyperparameters for our RF model:

In [37]:
```
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]
}
```

In [38]:
```
# 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}
```

Step 7.3 | RF Model Evaluation

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

In [39]:
```
# 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
```

In [40]:
```
# 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.

In [41]:
```
rf_evaluation = evaluate_model(best_rf, X_test, y_test, 'RF')
rf_evaluation
```

Out[41]:

	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

In [42]:
```
pickle.dump(best_rf, open("../model/best_rf","wb"))
```

11.2.3. KNN MODEL

KNN Base Model Definition

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

In [43]:
```
# Define the base KNN model and set up the pipeline with scaling
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])
```

KNN Hyperparameter Tuning

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

In [44]:
```
# 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
}
```

In [45]:
```
# 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'}
```

KNN Model Evaluation

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

In [46]:
```
# 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
```

In [47]:
```
# 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.

In [48]:
```
knn_evaluation = evaluate_model(best_knn, X_test, y_test, 'KNN')
knn_evaluation
```

Out[48]:

	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

In [49]:
```
pickle.dump(best_knn, open("../model/best_knn","wb"))
```

11.2.3.SVM MODEL

SVM Base Model Definition

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

In [50]:
```
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(probability=True))
])
```

SVM Hyperparameter Tuning

Let's configure the hyperparameters grid and employ the **tune_clf_hyperparameters** function to determine the best hyperparameters for our SVM pipeline:

In [51]:
```
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]
}
```

In [52]:
```
# 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'}
```

SVM Model Evaluation

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

In [53]:
```
# 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
```

In [54]:
```
# 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
```

✅ Inference

The **recall of 0.87 for class 1** indicates that almost all the **true positive cases (i.e., patients with heart disease)** are correctly identified. This high recall is of utmost importance in a medical context, where missing a patient with potential heart disease could have dire consequences.

However, it's also worth noting the balanced performance of the model. With an **F1-score of 0.83 for class 1**, it's evident that the model doesn't merely focus on maximizing recall at the expense of precision. This means the reduction in False Negatives hasn't significantly increased the False Positives, ensuring that the cost and effort of examining healthy individuals are not unnecessarily high.

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.

In [55]:
```
svm_evaluation = evaluate_model(best_svm, X_test, y_test, 'SVM')
svm_evaluation
```

Out[55]:

	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

In [56]:
```
pickle.dump(best_svm, open("../model/best_svm","wb"))
```

Step 10 | Conclusion

Tabel of Contents

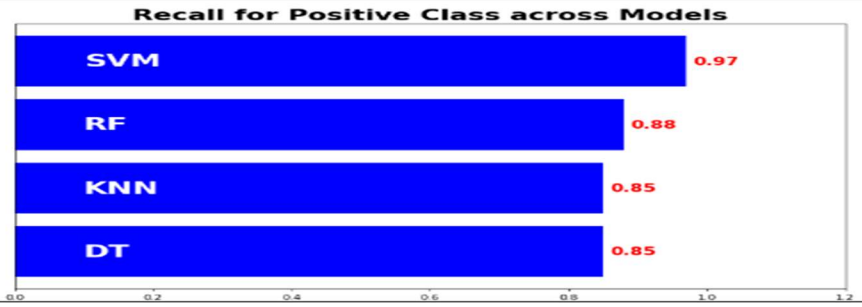
In the critical context of diagnosing heart disease, our primary objective is **to ensure a high recall for the positive class**. It's imperative to accurately identify every potential heart disease case, as even one missed diagnosis could have dire implications. However, while striving for this high 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.

In [57]:
```
# 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
```

Out[57]:

	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

In [58]:
```
# 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='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()
```



11.3. DEPLOYMENT USING STREAMLIT

project.py > ...

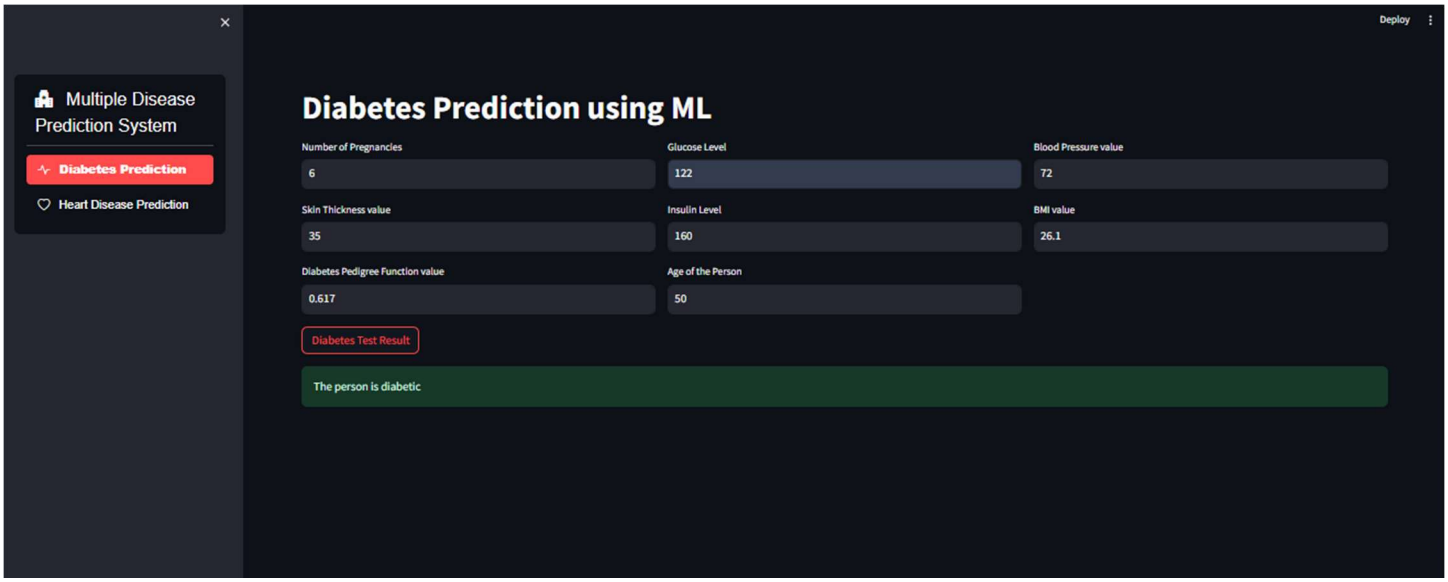
```python
import os
import pickle
import streamlit as st
from streamlit_option_menu import option_menu


# Set page configuration
st.set_page_config(page_title="Health Assistant",
                   layout="wide",
                   page_icon="🧑‍⚕️")


# loading the saved models

diabetes_model =  pickle.load(open('./model/dia/best_mlp','rb'))

heart_disease_model = pickle.load(open('./model/heart/best_svm','rb'))



# sidebar for navigation
with st.sidebar:
    selected = option_menu('Multiple Disease Prediction System',

                           ['Diabetes Prediction',
                            'Heart Disease Prediction'],
                           menu_icon='hospital-fill',
                           icons=['activity', 'heart'],
                           default_index=0)

# Diabetes Prediction Page
if selected == 'Diabetes Prediction':

    # page title
    st.title('Diabetes Prediction using ML')

    # getting the input data from the user
    col1, col2, col3 = st.columns(3)

    with col1:
        Pregnancies = st.text_input('Number of Pregnancies')

    with col2:
        Glucose = st.text_input('Glucose Level')

    with col3:
        BloodPressure = st.text_input('Blood Pressure value')

    with col1:
        SkinThickness = st.text_input('Skin Thickness value')

    with col2:
        Insulin = st.text_input('Insulin Level')

    with col3:
        BMI = st.text_input('BMI value')

    with col1:
        DiabetesPedigreeFunction = st.text_input('Diabetes Pedigree Function value')

    with col2:
        Age = st.text_input('Age of the Person')

    # code for Prediction
    diab_diagnosis = ''

    # creating a button for Prediction

    if st.button('Diabetes Test Result'):

        user_input = [Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin,
                      BMI, DiabetesPedigreeFunction, Age]

        user_input = [float(x) for x in user_input]

        diab_prediction = diabetes_model.predict([user_input])

        if diab_prediction[0] == 1:
            diab_diagnosis = 'The person is diabetic'
        else:
            diab_diagnosis = 'The person is not diabetic'

    st.success(diab_diagnosis)
```

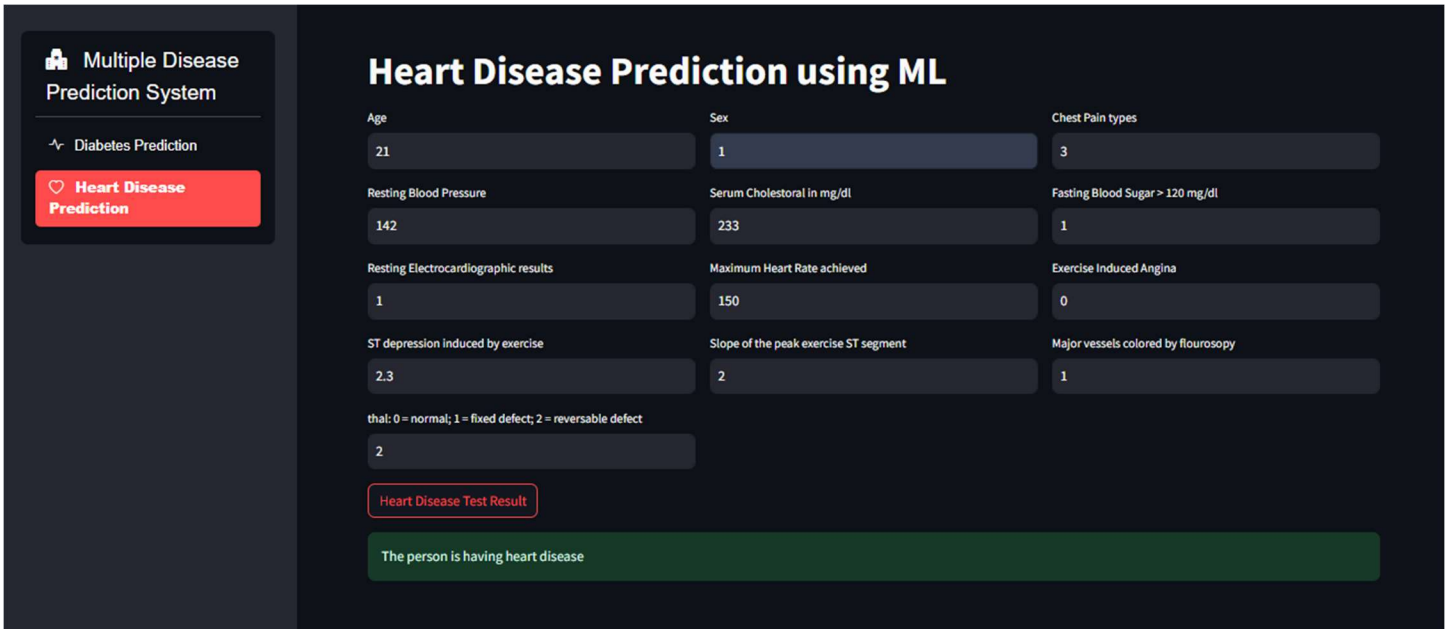
```python
    # Heart Disease Prediction Page
    if selected == 'Heart Disease Prediction':

        # page title
        st.title('Heart Disease Prediction using ML')

        col1, col2, col3 = st.columns(3)

        with col1:
            age = st.text_input('Age')

        with col2:
            sex = st.text_input('Sex')

        with col3:
            cp = st.text_input('Chest Pain types')

        with col1:
            trestbps = st.text_input('Resting Blood Pressure')

        with col2:
            chol = st.text_input('Serum Cholestoral in mg/dl')

        with col3:
            fbs = st.text_input('Fasting Blood Sugar > 120 mg/dl')

        with col1:
            restecg = st.text_input('Resting Electrocardiographic results')

        with col2:
            thalach = st.text_input('Maximum Heart Rate achieved')

        with col3:
            exang = st.text_input('Exercise Induced Angina')

        with col1:
            oldpeak = st.text_input('ST depression induced by exercise')

        with col2:
            slope = st.text_input('Slope of the peak exercise ST segment')

```

```python
    with col3:
        ca = st.text_input('Major vessels colored by flourosopy')

    with col1:
        thal = st.text_input('thal: 0 = normal; 1 = fixed defect; 2 = reversable defect')

    # code for Prediction
    heart_diagnosis = ''

    # creating a button for Prediction

    if st.button('Heart Disease Test Result'):

        user_input = [age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal]

        user_input = [float(x) for x in user_input]

        heart_prediction = heart_disease_model.predict([user_input])

        if heart_prediction[0] == 1:
            heart_diagnosis = 'The person is having heart disease'
        else:
            heart_diagnosis = 'The person does not have any heart disease'

    st.success(heart_diagnosis)

```

11.4.Model Deployment :

Model Deployment is one of the last stages of any machine learning project. Here, we will design a user interface. we used a flask to make an HTML file for flight price prediction. this will take the input value for each feature and calculate the price for a flight as shown in the image below.





12. PYTHON CODE:
12.1. Diabetes Diagnosis

Diabetes Diagnosis using Machine Learning

1. Load libraries:

In [1]:
```
# Python Libraries
# Classic,data manipulation and linear algebra
import pandas as pd
import numpy as np

# Plots
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
plt.style.use('fivethirtyeight')

#ignore warning messages
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
diabetes_data = pd.read_csv('../Dataset/diabetes.csv')
print(diabetes_data.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

2. Data Cleaning

In [3]:
```
diabetes_data.head(5)
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

The datasets consist of several medical predictor (independent) variables and one target (dependent) variable --> Outcome. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

In [4]:
```
print("dimension of diabetes data: {}".format(diabetes_data.shape))
```

```
dimension of diabetes data: (768, 9)
```

```
*The diabetes dataset consists of 768 data points, with 9 features*
```

Checking for missing values

In [5]:
```
diabetes_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:
```
# Check missing values
diabetes_data.isnull().sum()
```

Out[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [7]:
```
# Check the type of 'data'
type(diabetes_data)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
# Get row indices
diabetes_data_row_idx = diabetes_data.index
diabetes_data_row_idx
```

Out[8]:
```
RangeIndex(start=0, stop=768, step=1)
```

In [9]:
```
# Get data type for each attribute
diabetes_data.dtypes
```

Out[9]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [10]:
```
diabetes_data.describe()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

- **Minimum values for many variables are 0.**
- **As biological parameters like Glucose, BP, Skin thickness,Insulin & BMI cannot have zero values, looks like null values have been coded as zeros.**
- **So we need to find out how many Zero values are included in each variable.**

In [11]:
```
## Counting cells with 0 Values for each variable and publishing the counts below
(diabetes_data.Pregnancies == 0).sum(),(diabetes_data.Glucose==0).sum(),(diabetes_data.BloodPressure==0).sum(),(diabetes_data.Sk
```

Out[11]:
```
(111, 5, 35, 227, 374, 11, 0, 0)
```

- As Zero Counts of some the variables are as high as 374 and 227, in a 768 data set, it is better to remove the Zeros uniformly for 5 variables (except Pregnancies & Outcome)
- We'll drop 0 values and create a our new dataset which can be used for further analysis

In [12]:
```
drop_Glu = diabetes_data.index[diabetes_data.Glucose == 0].tolist()
drop_BP = diabetes_data.index[diabetes_data.BloodPressure == 0].tolist()
drop_Skin = diabetes_data.index[diabetes_data.SkinThickness == 0].tolist()
drop_Ins = diabetes_data.index[diabetes_data.Insulin == 0].tolist()
drop_BMI = diabetes_data.index[diabetes_data.BMI == 0].tolist()
```

In [13]:
```
# Creating a dataset called 'dia' from original dataset 'diabetes_data'
# with excludes all rows with have zeros only for Glucose, BP, Skinthickness, Insulin and BMI, as other columns can contain Zero

c = drop_Glu + drop_BP + drop_Skin + drop_Ins + drop_BMI
dia = diabetes_data.drop(diabetes_data.index[c])
```

In [14]:
```
dia.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 3 to 765
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               392 non-null    int64
 1   Glucose                   392 non-null    int64
 2   BloodPressure             392 non-null    int64
 3   SkinThickness             392 non-null    int64
 4   Insulin                   392 non-null    int64
 5   BMI                       392 non-null    float64
 6   DiabetesPedigreeFunction  392 non-null    float64
 7   Age                       392 non-null    int64
 8   Outcome                   392 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 30.6 KB
```

- **As in above, we created a cleaned up list titled "dia" which has 392 rows of data instead of 768 from original list.**
- **It looks like we lost nearly 50% of data but our data set is now cleaner than before.**
- **In fact, the removed values can be used for Testing during modeling. So actually we haven't really lost them completly.**

2.1 Performing Preliminary Descriptive Stats on the Data set

In [15]:
```
dia.describe()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000
mean	3.301020	122.627551	70.663265	29.145408	156.056122	33.086224	0.523046	30.864796	0.331633
std	3.211424	30.860781	12.496092	10.516424	118.841690	7.027659	0.345488	10.200777	0.471401
min	0.000000	56.000000	24.000000	7.000000	14.000000	18.200000	0.085000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	21.000000	76.750000	28.400000	0.269750	23.000000	0.000000
50%	2.000000	119.000000	70.000000	29.000000	125.500000	33.200000	0.449500	27.000000	0.000000
75%	5.000000	143.000000	78.000000	37.000000	190.000000	37.100000	0.687000	36.000000	1.000000
max	17.000000	198.000000	110.000000	63.000000	846.000000	67.100000	2.420000	81.000000	1.000000

2.2 Split the data frame into two sub sets for convenience of analysis

> As we wish to study the influence of each variable on Outcome (Diabetic or not), we can subset the data by Outcome
>
> - dia1 Subset : All samples with 1 values of Outcome
> - dia0 Subset : All samples with 0 values of Outcome

In [16]:
```
dia1 = dia[dia.Outcome == 1]
dia0 = dia[dia.Outcome == 0]
```

12.2. Heart diseases

In [1]:
```
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
```

Step 2 | Read Dataset

Tabel of Contents

In [3]:
```
# Read dataset
df = pd.read_csv('../Dataset/heart.csv')
df
```

Out[3]:

	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

Dataset Description

Variable	Description
age	Age of the patient in years
sex	Gender of the patient (0 = male, 1 = female)
cp	Chest pain type: 0: Typical angina 1: Atypical angina 2: Non-anginal pain 3: Asymptomatic
trestbps	Resting blood pressure in mm Hg
chol	Serum cholesterol in mg/dl
fbs	Fasting blood sugar level, categorized as above 120 mg/dl (1 = true, 0 = false)
restecg	Resting electrocardiographic results: 0: Normal 1: Having ST-T wave abnormality 2: Showing probable or definite left ventricular hypertrophy
thalach	Maximum heart rate achieved during a stress test
exang	Exercise-induced angina (1 = yes, 0 = no)
oldpeak	ST depression induced by exercise relative to rest
slope	Slope of the peak exercise ST segment: 0: Upsloping 1: Flat 2: Downsloping
ca	Number of major vessels (0-4) colored by fluoroscopy
thal	Thalium stress test result: 0: Normal 1: Fixed defect 2: Reversible defect 3: Not described
target	Heart disease status (0 = no disease, 1 = presence of disease)

Step 3.1 | Dataset Basic Information

In [4]:
```
# 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:

- **Number of Entries**: The dataset consists of **303 entries**, ranging from index 0 to 302.
- **Columns**: There are **14 columns** in the dataset corresponding to various attributes of the patients and results of tests.
- **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.
- **Missing Values**: There don't appear to be any missing values in the dataset as each column has 303 non-null entries.

Note: Based on the data types and the feature explanations we had earlier, 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:

In [5]:
```
# 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
```

Out[5]:
```
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
```

Step 3.2 |Summary Statistics for Numerical Variables

In [6]:
```
# Get the summary statistics for numerical variables
df.describe().T
```

Out[6]:

	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

Numerical Features:

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

Step 3.3 | Summary Statistics for Categorical Variables

In [7]:
```
# Get the summary statistics for categorical variables
df.describe(include='object')
```

Out[7]:

	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

Categorical Features (object data type

- `sex` : There are two unique values, with males (denoted as 0) being the most frequent category, occurring 207 times out of 303 entries.
- `cp` : Four unique types of chest pain are present. The most common type is "**0**", occurring 143 times.
- `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.
- `restecg` : Three unique results are present. The most common result is "**1**", appearing 152 times.
- `exang` : There are two unique values. The most frequent value is "**0**" (indicating no exercise-induced angina), which is observed 204 times.
- `slope` : Three unique slopes are present. The most frequent slope type is "**2**", which occurs 142 times.
- `ca` : There are five unique values for the number of major vessels colored by fluoroscopy, with "**0**" being the most frequent, occurring 175 times.
- `thal` : Four unique results are available. The most common type is "**2**" (indicating a reversible defect), observed 166 times.
- `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.

13. SWOT Analysis

13.1. Strengths:

1. Valuable Insights:

The project provides actionable insights for early detection and prevention of heart diseases and diabetes, potentially saving lives and reducing healthcare costs.

Data-driven insights can help healthcare providers tailor personalized treatment plans.

2. Machine Learning Techniques:

Utilizes advanced machine learning algorithms that can accurately predict the likelihood of heart diseases and diabetes, offering higher precision than traditional methods.

Continuous improvement of models through retraining with new data.

3. Dataset Availability:

Access to extensive and diverse datasets, including patient health records, lifestyle data, and genetic information, which enhance model training and prediction accuracy.

Availability of public datasets and potential partnerships for data sharing.

4. Feature Engineering:

Advanced feature engineering techniques to extract meaningful patterns and relationships from raw data.

Use of domain expertise to identify relevant features, improving model performance.

5. Evaluation Metrics:

Comprehensive evaluation metrics (accuracy, precision, recall, F1-score, ROC-AUC) to assess model performance, ensuring robustness and reliability.

Use of cross-validation techniques to validate model generalizability.

13.2. Weaknesses:

1. Limited Dataset Scope:

The dataset may not cover all demographic groups or geographic regions, leading to potential biases in the prediction model.

Insufficient longitudinal data to fully capture the progression of diseases over time.

2. Data Quality Issues:

Inconsistent, incomplete, or erroneous data can negatively impact model training and prediction accuracy.

Challenges in standardizing data from multiple sources.

3. Model Generalization:

Risk of overfitting to the training data, resulting in poor generalization to new, unseen data.

Need for extensive validation on diverse datasets to ensure model applicability in various settings.

13.3. Opportunities:

1. Real-time Data Integration:

Integration of real-time health data from wearable devices and IoT sensors for continuous monitoring and timely intervention.

Potential to leverage big data analytics for more dynamic and adaptive prediction models.

2. User Interface Development:

Development of user-friendly interfaces and mobile applications to facilitate easy access to prediction results and health recommendations for patients and healthcare providers.

Enhanced visualization tools for better interpretation of prediction outcomes.

3. Collaboration with Hospitals:

Partnering with hospitals to gain access to extensive patient data, enhancing model accuracy and reliability.
Implementation of predictive models in hospital settings to assist clinicians in early diagnosis and treatment planning.

13.4. Threats:

1. Competing Solutions:

Emergence of competing predictive models and solutions in the healthcare market, potentially reducing the project's market share.

Rapid advancements in machine learning techniques by competitors.

2. Regulatory Factors:

Stringent regulatory requirements for healthcare data usage and predictive model deployment, which may delay project implementation.

Compliance with data privacy laws (e.g., GDPR, HIPAA) and ethical considerations.

3. Technological Advancements:

Rapid technological advancements may render the current predictive models obsolete, necessitating continuous updates and improvements.

Dependency on high computational resources and infrastructure, posing challenges for scalability and cost-effectiveness.

14. CONCLUSION:

The heart disease and diabetes prediction project represents a significant advancement in healthcare by leveraging machine learning techniques to provide valuable insights for early detection and personalized treatment. By utilizing extensive datasets, including patient health records, lifestyle data, and genetic information, the project enhances the accuracy and reliability of its predictions. Advanced feature engineering and robust evaluation metrics further strengthen the model's performance, ensuring that the predictions are both precise and actionable.

However, the project faces several challenges that need to be addressed. The limited scope of datasets can lead to potential biases, and data quality issues, such as inconsistencies and incomplete information, can negatively impact the model's accuracy. Additionally, there is a risk of overfitting, which affects the model's generalizability to new, unseen data.

Despite these weaknesses, the project offers significant opportunities. Real-time data integration from wearable devices and IoT sensors can provide continuous monitoring and timely interventions. Developing user-friendly interfaces and mobile applications can facilitate easy access to prediction results and health recommendations. Collaborations with hospitals can enhance data access, improve model accuracy, and support joint research initiatives.

However, the project must navigate threats such as competing solutions, regulatory factors, and rapid technological advancements. Addressing these challenges and seizing the opportunities can lead to transformative impacts on healthcare, improving patient outcomes and healthcare efficiency through predictive analytics.

15. FUTURE WORK:

1. Data Expansion and Quality Improvement:

Collect more diverse and comprehensive datasets to cover various demographic groups and geographic regions. Implement advanced data cleaning and preprocessing techniques to address inconsistencies, missing values, and errors in the data.

2. Model Enhancement:

Explore and integrate more sophisticated machine learning algorithms and techniques, such as deep learning and ensemble methods, to improve prediction accuracy.

Regularly update the model with new data to maintain its relevance and accuracy over time.

3. Real-time Data Integration:

Develop systems to integrate real-time health data from wearable devices, mobile apps, and IoT sensors.

Implement real-time monitoring and alert systems to provide timely health interventions.

4. User Interface and Experience:

Design and develop intuitive and user-friendly interfaces for both healthcare providers and patients.

Include features for personalized health recommendations and easy interpretation of prediction results.

5. Collaboration and Partnerships:

Strengthen collaborations with hospitals and healthcare institutions to enhance data sharing and validation of models. Partner with research institutions for continuous improvement of predictive models and to stay updated with the latest advancements in the field.

6. Regulatory Compliance and Ethical Considerations:

Ensure compliance with all relevant data privacy regulations (e.g., GDPR, HIPAA) and maintain high ethical standards in data usage and model deployment.

Develop transparent model interpretability and explain ability to gain trust from healthcare providers and patients.

7. Scalability and Deployment:

Focus on scalable solutions that can be deployed in various healthcare settings, from small clinics to large hospitals. Develop cloud-based infrastructure to support widespread and efficient use of the predictive models.

8. Patient Engagement and Education:

Create educational materials and programs to help patients understand the importance of early detection and how to use the predictive tools.

Encourage patient engagement through interactive features and feedback mechanisms in the application.

9. Cost-effectiveness Analysis:

Conduct comprehensive cost-benefit analyses to demonstrate the economic value of the predictive models in reducing healthcare costs and improving patient outcomes.

Explore funding opportunities and business models to sustain and expand the project.

10. Continuous Evaluation and Feedback Loop:

Establish a continuous evaluation process to gather feedback from users and healthcare providers.

Use this feedback to make iterative improvements to the predictive models and user interfaces.

16.REFERENCES

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8843464&isnumber=8843429

https://www.researchgate.net/publication/337821411_heart diseases prediction-in_India/citation/download

https://towardsdatascience.com/a-practical-guide-for-exploratory-data-analysis-diabetes-prediction-f8a713ef7121

https://medium.com/geekculture/heart-prediction-93da3958eb95

https://www.analyticsvidhya.com/blog/2022/01/diabetes-diseases-prediction-using-machine-learning/