MULTIPLE DISEASE PREDICTION USING MACHINE LEARNING

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by
Ayman Sami, 221910320018
Swaroopa, 221910320028
Saketh Kanchi, 221910320033

Under the esteemed guidance of
Mr. A. B Pradeep
Asst. professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GITAM
(Deemed to be University)
HYDERABAD

March 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "MULTIPLE DISEASE PREDICTION USING MACHINE LEARNING" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. In Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s)	Name(s)	Signature(s)
221910320018	Ayman Sami	
221910320028	Swaroopa	
221910320033	Saketh Kanchi	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled **"MULTIPLE DISEASE PREDICTION USING MACHINE LEARNING"** is a bonafide record of work carried out by **Ayman Sami (221910320018), Swaroopa (221910320028), Saketh Kanchi (221910320033)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Project Coordinator**	**Head of Department**
Mr. A. B Pradeep Kumar	Dr. S. Aparna	Dr. K. Sudeep
Asst Professor	Professor	Associate Professor

ACKNOWLEDGEMENT

Our project would not have been successful without the help of several people. We would like to thank the personalities who were part of our project in numerous ways, those who gave us outstanding support from the birth of the project.

We are incredibly thankful to our honorable Pro-Vice Chancellor, **Prof. D Sambasiva Rao**, for providing the necessary infrastructure and resources to accomplish our project.

We are highly indebted **to Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the project's tenure.

We are very much obliged to our beloved **Prof. K. Sudeep**, Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this project and for encouragement in the completion of this project.

We would like to express our special thanks to our Project Co-ordinator **Dr. S. Aparna**, Assistant Professor, Department of Computer Science and Engineering, School of Technology, for her time and efforts she provided throughout the year.

We hereby wish to express our deep sense of gratitude to **Mr. A. B Pradeep Kumar**, Assistant Professor, Department of Computer Science and Engineering, School of Technology, for the esteemed guidance, moral support, and invaluable advice provided by him for the success of the project.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our project a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support directly or indirectly in our project work.

TABLE OF CONTENTS

1. ABSTRACT

Health is one of the important factors to be considered by an individual. With the increasing number of diseases and the population, medical practitioners find it hard to diagnose many numbers of diseases and predict whether the individual is suffering from the disease or not, over intensive population growth.

Here comes the need for dynamic Health care systems, which are established to meet the health requirements of the population. Such systems are built with technology, health care, and data, that have to be processed in a smart, efficient, and precise course of action.

Prediction system is the best technology that can meet the level of expectation in this field. There are many models proposed for single disease identification. However, very little is proposed concerning multiple disease identification. Many models are created by python pickling method and compared by applying it to multiple data sets and using different performance measures.

Accurate and on-time analysis of any health-related problem is important for the prevention and treatment of the illness. The traditional way of diagnosis may not be sufficient in the case of a serious ailment. Developing a medical diagnosis system based on machine learning (ML) algorithms for prediction of any disease can help in a more accurate diagnosis than the conventional method. We have designed a disease prediction system using multiple ML algorithms.

The main aim of the disease prediction model which identifies the multiple disease possibility by analyzing the health record of the patient. We consider the diseases such as Heart disease, Diabetes, and kidney disease using some of the basic parameters such as Pulse Rate, Cholesterol, Blood Pressure, Heart Rate, etc., and also the risk factors associated with the disease can be found using prediction model with good accuracy and Precision. Our diagnosis model can act as a doctor for the early diagnosis of a disease to ensure the treatment can take place on time and lives can be saved.

2. INTRODUCTION

If an organization needs to analyze their patient's health report, there arises a need to deploy many models for each disease. The approach which is made use in the existing systems is practicable for one disease but not for more than one disease. The mortality rate is increased since the precise disease is not diagnosed properly. Even though the patient recovered from one disease may also suffer from other diseases

The mortality rate is increased since the precise disease is not diagnosed properly. If we take diabetes, there is a probability of the diseases like hearing loss, heart disease, and dementia. In this model, we consider the diagnosis of heart disease, diabetes, and Parkinson's disease.

If the user needs to analyze the health condition of the patient, either they can predict a specific disease or utilizing the report which comprises of the relevant features considered to diagnose the d isease.

The machine learning model is trained on that record to get accurate results. Initially, algorithms of ML were designed and employed to observe medical data sets. Today, for efficient analysis of data, ML recommended various tools. Especially in the last few years, digital revolution has offered comparatively low- cost and obtainable means for collection and storage of data.

One of the many machine-learning applications is employed to build such classifier that can divide the data on the basis of their attributes. Data set is divided into two or more than two classes. Such classifiers are used for medical data analysis and disease detection. First the data is cleaned so as to avoid the missing values in the data set.

To predict the disease various input parameters like have been considered. For diabetes input parameters like Blood pressure, Glucose level are considered. For Heart disease input parameters like Cholesterol, Resting BP, Chest pain type are considered. For Parkinson's disease input parameters like DFA, NHR, HNR are considered. Our main objective of this research is to predict the Multiple Diseases like Diabetes, Heart Disease, and Parkinson's disease. To predict multiple diseases, the Random Forest algorithm is used.

Machine learning algorithms are being used in healthcare to help diagnose and treat a wide range of diseases. These algorithms are able to analyze large amounts of medical data quickly and accurately, making it possible to identify patterns and predict outcomes that might not be visible to human doctors.

One of the main advantages of machine learning in healthcare is its ability to analyze data from multiple sources, such as electronic health records, medical images, and genetic data. By integrating these different types of data, machine learning algorithms can provide a more comprehensive and accurate picture of a patient's health, helping doctors to make more informed decisions about diagnosis and treatment.

Another advantage of machine learning in healthcare is that it can help to identify and predict diseases at an early stage, before they become more serious or life-threatening. This is particularly important for chronic diseases such as diabetes and heart disease, which can be managed more effectively if they are detected early.

Random Forest is a machine learning algorithm that has been widely used in healthcare for disease prediction. It is an ensemble learning method that combines multiple decision trees to improve the accuracy and robustness of the predictions. Random Forest has been shown to be effective in predicting multiple diseases, including diabetes, heart disease, and Parkinson's disease, as mentioned in the input.

Overall, the use of machine learning algorithms in healthcare has the potential to revolutionize the way that diseases are diagnosed and treated. By providing more accurate and comprehensive insights into a patient's health, these algorithms can help to improve outcomes, reduce costs, and ultimately save lives.

3. LITERATURE REVIEW

3.1 DATASET EXTRACTION

Kaggle is one of the largest data source providers for the learning purpose. It allows users to find and publish data sets, explore and build models in a web-based data-science environment and hence the data is collected from Kaggle.

3.2 STUDY OF DATASETS

A Dataset is a set or collection of data. This set is normally presented in a tabular pattern. Every column describes a particular variable. And each row corresponds to a given member of the data set, as per the given user requirements, two data sets are considered, one for the training and another testing. The training dataset is used to train the model in which datasets are further divided into two parts such as 80:20 or 70:30 the major dataset is used for the training of the model and the minor dataset is used for the test model. Hence the accuracy of our developed model is calculated.

3.3 LITERATURE SURVEY

Iyer et al. has performed a work to predict diabetes disease by using decision tree and Naive Bayes. Diseases occur when production of insulin is insufficient or there is improper use of insulin. Data set used in this work is Pima Indian diabetes data set. Various tests were performed using WEKA data mining tool. In this data-set percentage split (70:30) predict better than cross validation. J48 shows 74.8698% and 76.9565% accuracy by using Cross Validation and Percentage Split Respectively. Naive Bayes presents 79.5652% correctness by using PS. Algorithms shows highest accuracy by utilizing percentage split test.

Meta learning algorithms for diabetes disease diagnosis has been discussed by Sen and Dash . The employed data set is Pima Indians diabetes that is received from UCI Machine Learning laboratory. WEKA is used for analysis. CART, Adaboost, Logiboost and grading learning algorithms are used to predict that patient has diabetes or not. Experimental results are compared on the behalf of correct or incorrect classification. CART offers 78.646% accuracy. The Adaboost obtains 77.864% exactness. Logiboost offers the correctness of 77.479%. Grading has correct classification rate of 66.406%. CART offers highest accuracy of 78.646% and misclassification Rate of 21.354%, which is smaller as compared to other techniques.

An experimental work to predict diabetes disease is done by the Kumari and Chitra. Machine learning technique that is used by the scientist in this experiment is SVM. RBF kernel is used in SVM for the purpose of classification. Pima Indian diabetes data set is provided by machine learning laboratory at University of California, Irvine. MATLAB 2010a are used to conduct experiment. SVM offers 78% accuracy.

Sarwar and Sharma have suggested the work on Naive Bayes to predict diabetes Type-2. Diabetes disease has 3 types. First type is Type-1 diabetes, Type-2 diabetes is the second type and third type is gestational diabetes. Type-2 diabetes comes from the growth of Insulin resistance. Data set consists of 415 cases and for purpose of variety; data are gathered from dissimilar sectors of society in India. MATLAB with SQL server is used for development of model. 95% correct prediction is achieved by Naive Bayes.

Ephzibah has constructed a model for diabetes diagnosis. Proposed model joins the GA and fuzzy logic. It is used for the selection of best subset of features and also for the enhancement of classification accuracy. For experiment, dataset is picked up from UCI Machine learning laboratory that has 8 attributes and 769 cases. MATLAB is used for implementation. By using genetic algorithm only three best features/attributes are selected. These three attributes are used by fuzzy logic classifier and provide 87% accuracy. Around 50% cost is less than the original cost. Table 2 provides the Comprehensive view of Machine learning Techniques for diabetes disease diagnosis.

Machine Learning (ML) is basically that field of computer science with the help of which computer systems can provide sense to data in much the same way as human beings do. In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. The key focus of ML is to allow computer systems to learn from experience without being explicitly programmed or human intervention. The user must have basic knowledge of artificial intelligence. He/she should also be aware of Python, NumPy, Scikit-learn, SciPy, Matplotlib.

The mortality rate is increased since the precise disease is not diagnosed properly. If we take diabetes, there is a probability of the diseases like hearing loss, heart disease, and dementia. In this model, we consider the diagnosis of heart disease, diabetes, and Parkinson’s disease. If the user needs to analyze the health condition of the patient, either they can predict a specific disease or utilizing the report which comprises of the relevant features considered to diagnose the disease.

The vast majority of current studies focused on a common illness. When a user wants to analyse diabetes, they must use one model, and when they try to analyse heart disease, they must use another model. This is a lengthy procedure. Furthermore, if a user has several illnesses but the current method can only predict one of them, then there is a risk of death.

A number of experiments have been carried out to compare the performance of predictive data mining techniques on the same dataset, and the results show that Decision Tree outperforms, with Bayesian classification having comparable accuracy to Decision Tree in some cases, but other predictive approaches such as KNN, Neural Networks, and Classification based on Clustering underperform.

Different deep learning and machine learning techniques are employed to analyse diabetes, heart disease prediction, and Parkinson's. The patient's condition is determined using the random forest algorithm and a variety of other algorithms. The accuracy of training data for diabetes disease is 74 percent. Presently we have conducted only for diabetes, the training data for the remaining two diseases will be done later. Compared to all other algorithms Random Forest Algorithm has given the highest accuracy.

This system was tested with a reduced collection of features from the Parkinson's, Diabetes, and Heart disease datasets were examined to other machine learning strategies in R studio, including Decision tree, SVM-Polynomial, SVM-Linear, and Random forest. Accuracy, specificity, sensitivity, and misclassification rate have all been used to test the efficiency of these machine learning techniques. According to the findings of the trial, the enhanced Random Forest algorithm has better accuracy and for the remaining two diseases it will be predicted later.

4. PROBLEM IDENTIFICATION AND OBJECTIVES

4.1 TYPES OF ML MODELS

Depending on the situation, machine learning algorithm's function using more or less human intervention/reinforcement. The four major machine learning models are:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning reinforcement learning

4.1.1 Models Used

- Supervised Learning

With supervised learning, the computer is provided with a labeled set of data that enables it to learn how to do a human task. This is the least complex model, as it attempts to replicate human learning.

- Unsupervised Learning

With unsupervised learning, the computer is provided with unlabeled data and extracts previously unknown patterns/insights from it. There are many different ways machine learning algorithms do this, including:

Clustering, in which the computer finds similar data points within a data set and groups them accordingly (creating "clusters").

Density estimation, in which the computer discovers insights by looking at how a data set is distributed.

Anomaly detection, in which the computer identifies data points within a data set that are significantly different from the rest of the data.

Principal component analysis (PCA), in which the computer analyzes a data set and summarizes it so that it can be used to make accurate predictions.

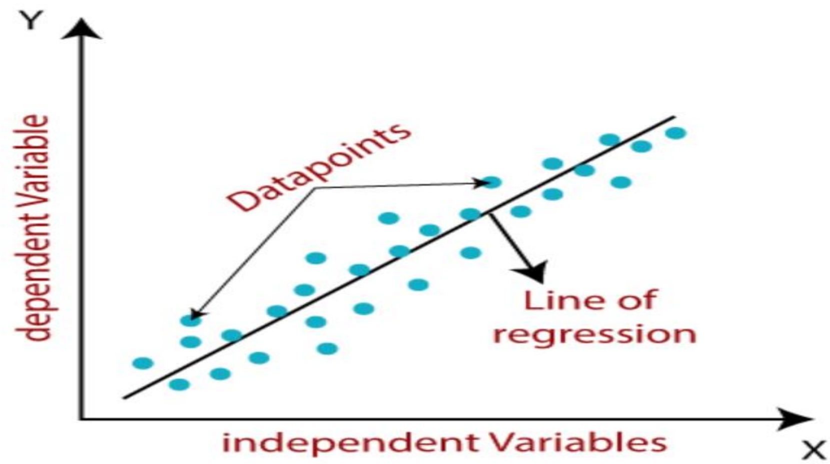
4.2 SUPERVISED LEARNING ALGORITHM

4.2.1 Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

● Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

● Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input)



and y (output).

● Hence, the name is Linear Regression.

Fig 4.2.1 Linear regression

4.2.2 Logistic Regression

Logistic Regression is a popular and very useful algorithm of machine learning for classification problems. The advantage of logistic regression is that it is a predictive analysis.

Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set. A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables.

● For example, a logistic regression could be used to predict whether a political candidate will win or lose an election or whether a high school student will be admitted or not to a particular college. These binary outcomes allow straightforward decisions between two alternatives.

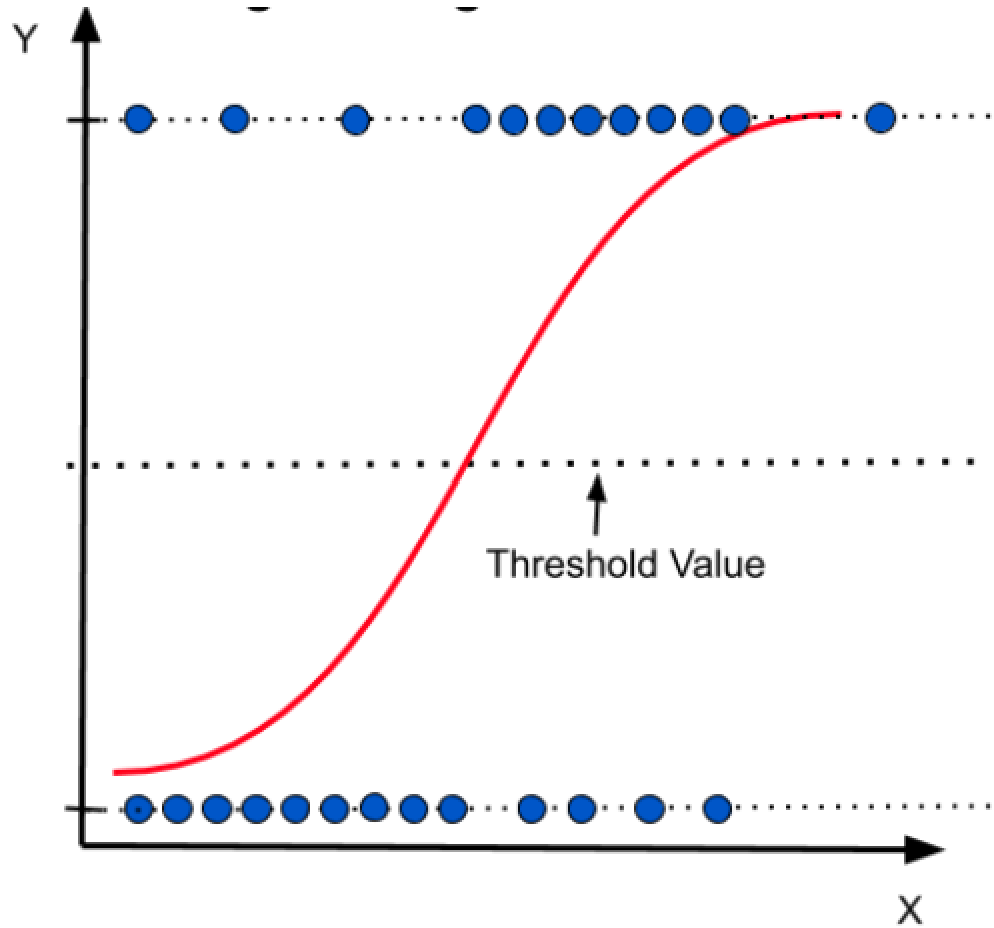


Fig 4.2.2 Logistic regression

4.2.3 Support Vector Machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

SVM algorithm can be used for Face detection, image classification, text categorization, etc.

SVM can be of two types:

- Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Advantages of Support Vector Machine:

- SVM works relatively well when there is a clear margin of separation between classes.
- SVM is more effective in high dimensional spaces.
- SVM is effective in cases where the number of dimensions is greater than the number of samples.
- SVM is relatively memory efficient

Disadvantages of Support Vector Machine:

- SVM algorithm is not suitable for large data sets.
- SVM does not perform very well when the data set has more noise i.e., target classes are overlapping.
- In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
- As the support vector classifier works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification.

4.2.4 Random Forest Classifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of over fitting.

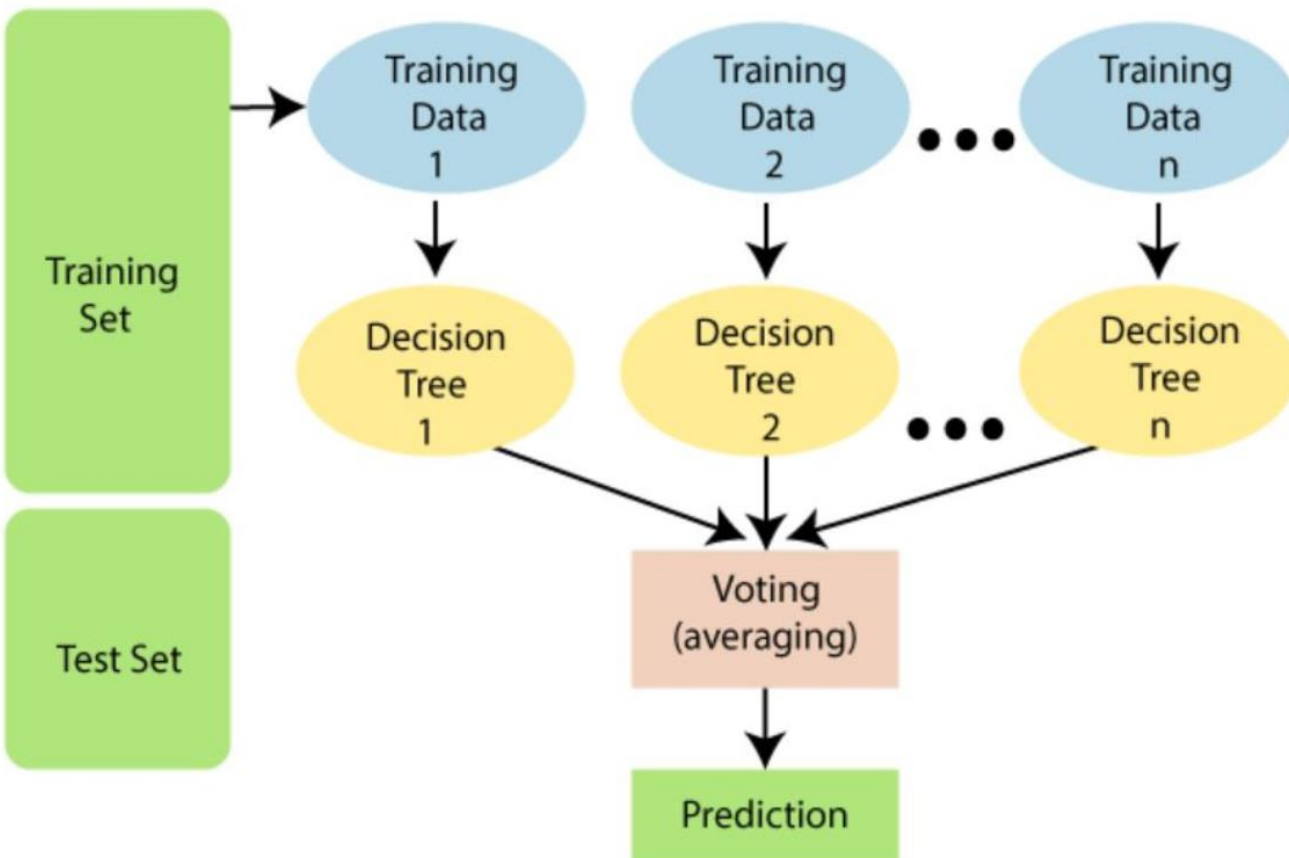


Fig 4.2.3 Working of the Random Forest algorithm

Advantages of Random Forest:

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.

- It enhances the accuracy of the model and prevents the over fitting issue.
- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

Disadvantages of Random Forest:

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

There are mainly four sectors where Random Forest mostly used:

- Banking: Banking sector mostly uses this algorithm for the identification of loan risk.
- Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified.
- Land Use: We can identify the areas of similar land use by this algorithm.
- Marketing: Marketing trends can be identified using this algorithm.

Implementation Steps are given below:

1) Data Pre-processing step
2) Fitting the Random Forest algorithm to the Training set
3) Predicting the test result
4) Test accuracy of the result (Creation of Confusion matrix)
5) Visualizing the test set result.

4.3 UNSUPERVISED LEARNING

4.3.1 K-means

● K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science.

● K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

● It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

● It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

● The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

● The k-means clustering algorithm mainly performs two tasks:

1) Determines the best value for K center points or centroids by an iterative process.

2) Assign each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster. Hence each cluster has data points with some commonalities, and it is away from other clusters.

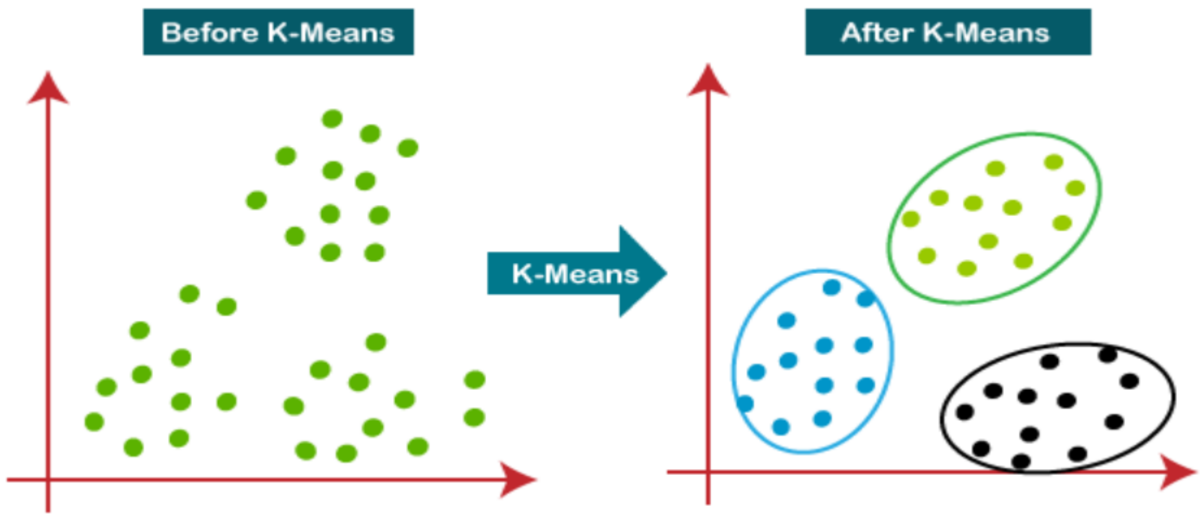


Fig 4.3.1 K-means clustering

4.4 Objectives

Our project aims to improve the current prediction models used by users both in terms of speed and accuracy. While bringing various types of diseases to predict in a single web application, using the most accurate prediction model for each separate disease. This will help the users get a general idea of their health regarding a particular disease. The previous system designed by the author of base paper only uses a single algorithm to solve the problem. Our system uses the most accurate model for each disease to get the best possible prediction for each kind of disease.

5. SYSTEM METHODOLOGY

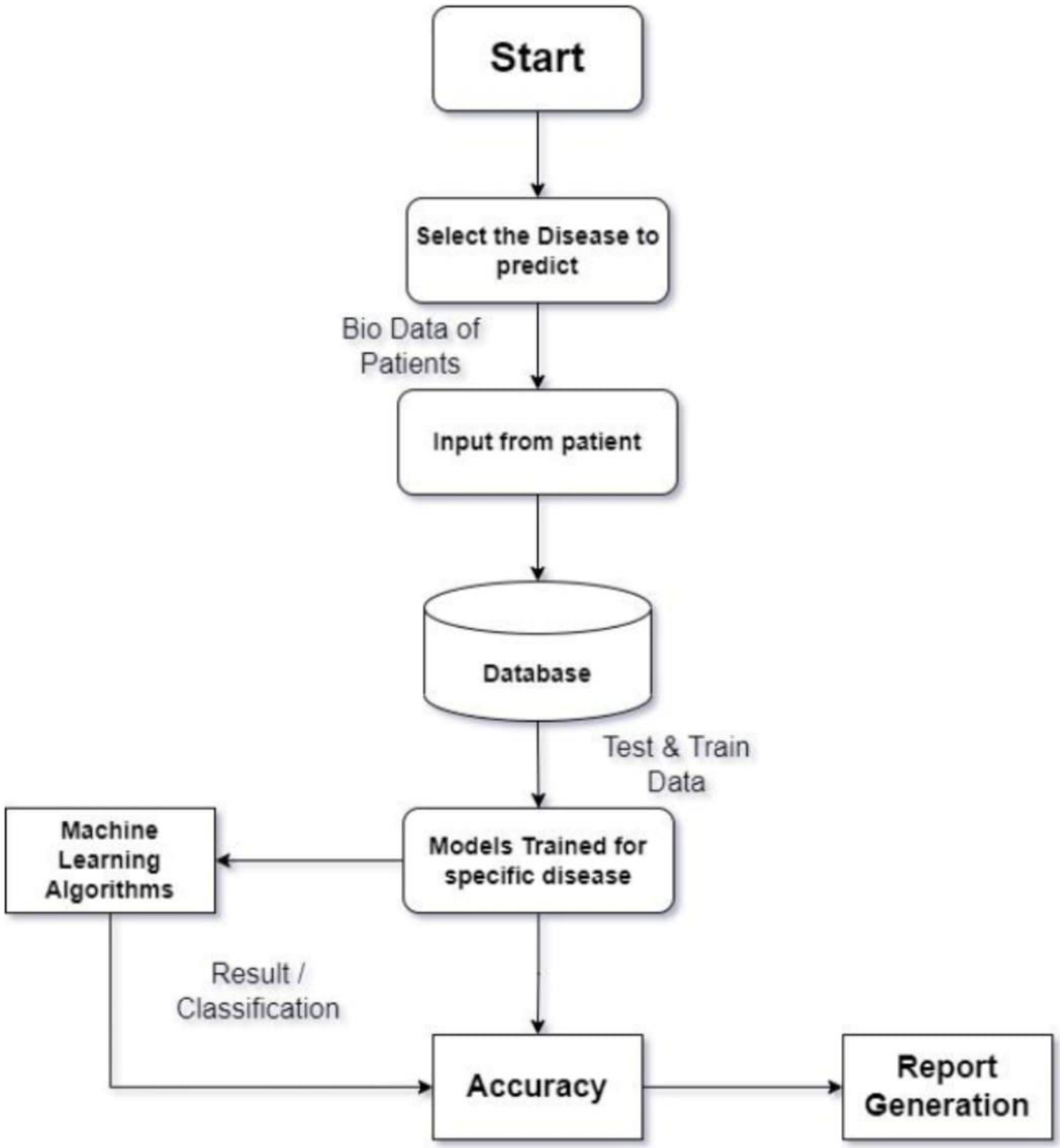


Fig 5.1 Flowchart

6. OVERVIEW OF TECHNOLOGIES

6.1 Introduction to Google Colab

Google Colab is a document that allows you to write, run, and share Python code within your browser. It is a version of the popular Jupyter Notebook within the Google suite of tools. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. It allows you to create a document containing executable code along with text, images, HTML, LaTeX, etc. which is then stored in your google drive and shareable to peers and colleagues for editing, commenting, and Viewing. Also it's a free Jupyter notebook interactive development environment.

Fig 6.1 Google Colab logo

6.2 Why Google Colab

As our project is completely based on python and should be done together, we've chosen google colab as the best working environment.

6.3 Python

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

6.4 Why Python

Benefits that make Python the best fit for machine learning and AI-based projects include simplicity and consistency, access to great libraries and frameworks for AI and machine learning (ML), flexibility, platform independence, and a wide community. These add to the overall popularity of the language.

6.5 Libraries Used

NumPy: NumPy is a basic Python scientific computing package that adds support for large multidimensional arrays and matrices and an extensive library of advanced mathematical functions for manipulating these arrays.

Matplotlib: Matplotlib is one of the plotting libraries in python which is however widely in use for machine learning applications with its numerical mathematics extension- NumPy to create static, animated and interactive visualizations.

Pandas: Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named NumPy, which provides support for multi-dimensional arrays.

Sklearn: Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Streamlit: Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers. It is a tool that is easier to learn and to use, as long as it can display data and collect needed parameters for modeling. Streamlit allows you to create a stunning-looking application with only a few lines of code.

7. IMPLEMENTATION

This project is divided into two phases, i.e., the Machine Learning Models and UI. And the Machine Learning Models is further divided into three modules, each module consisting of the training model for the disease selected

7.1) Diabetes Disease

Description

The dataset contains several medical predictor (Independent) variables and one target variable, (Outcome). Predictor variables include:

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

Dataset url: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Step 1: Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Load the Dataset

```
diabetes_dataset = pd.read_csv('diabetes.csv')
```

Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

3.1) Understanding Your Variables

3.1.1) Head of the dataset
3.1.2) The shape of the dataset

3.1.3) List types of columns
3.1.4) Info of the dataset
3.1.5) Summary of the dataset

3.1.1) Head of the Dataset

```
# Display first five records
diabetes_dataset.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
# Display last five records
diabetes_dataset.tail()
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

```
# Disply random records
diabetes_dataset.sample(5)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
582           12      121             78             17        0  26.5
518           13       76             60              0        0  32.8
501            3       84             72             32        0  37.2
701            6      125             78             31        0  27.6
653            2      120             54              0        0  26.8

     DiabetesPedigreeFunction  Age  Outcome
582                     0.259   62        0
518                     0.180   41        0
501                     0.267   28        0
701                     0.565   49        1
653                     0.455   27        0
```

3.1.2) The Shape of Dataset

```
# Numbers of rows and columns
diabetes_dataset.shape
```

```
(768, 9)
```

3.1.3) List types of columns

```
# List types of all columns
diabetes_dataset.dtypes
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

3.1.4) Info of the Dataset

```
# Checking for null values
diabetes_dataset.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
```

```
# Statistical Summary
diabetes_dataset.describe()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000   
mean      3.845052  120.894531      69.105469      20.536458   79.799479   
std       3.369578   31.972618      19.355807      15.952218  115.244002   
min       0.000000    0.000000       0.000000       0.000000    0.000000   
25%       1.000000   99.000000      62.000000       0.000000    0.000000   
50%       3.000000  117.000000      72.000000      23.000000   30.500000   
75%       6.000000  140.250000      80.000000      32.000000  127.250000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   
```

```
             BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Step 4: Split the data frame in X & Y

```
target_name = 'Outcome'

# Separate object for target feature
y = diabetes_dataset[target_name]

# Separate obhect for input feature
X = diabetes_dataset.drop(target_name,axis=1)

X.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
```

```
y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Step 5: Apply Feature Scaling

Various Data Scaling Techniques:

- Normalizer
- MinMax Scaler
- Binarizer
- Standard Scaler

```
# Apply Standard Scaler
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
SSX = scaler.fit_transform(X)
```

Step 6: Train Test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(SSX,y,test_size=0.2,random_state=2)
```

```
X_train.shape,y_train.shape
```

((614, 8), (614,))

Step 7: Build CLASSIFICATION Algorithms

8.1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,y_train)
```

LogisticRegression(multi_class='ovr', solver='liblinear')

8.2) K-Nearest Neighbors Classifier(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
```

KNeighborsClassifier()

8.3) Naive-Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
```

GaussianNB()

8.4) Support Vector Machine (SVM)

```
from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train,y_train)
```

SVC(kernel='linear')

8.5) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

DecisionTreeClassifier()

8.6) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='entropy')
rf.fit(X_train,y_train)
```

RandomForestClassifier(criterion='entropy')

Step 8: Making Prediction

9.1) Making Prediction using Logistic Regression

```
lr_pred = lr.predict(X_test)
```

9.2) Making Prediction using KNN

```
knn_pred = knn.predict(X_test)
```

9.3) Making Prediction using Naive Bayes

```
nb_pred = nb.predict(X_test)
```

9.4) Making Prediction using SVM

```
sv_pred = sv.predict(X_test)
```

9.5) Making Prediction using Decision Tree

```
dt_pred = dt.predict(X_test)
```

9.6) Making Prediciton using Random Forest

```
rf_pred = rf.predict(X_test)
```

Step 9: Model Evaluation

10.1) Train & Test Scores

```
from sklearn.metrics import accuracy_score
```

```
# Train & Test Scores of Logistic Regression
print("Accuracy (Train) score of Logistic Regression ",lr.score(X_train,y_train)*100)
print("Accuracy (Test) score of Logistic Regression ", lr.score(X_test,y_test)*100)
print("Accuracy score of Logistic Regression ", accuracy_score(y_test,lr_pred)*100)
```

```
Accuracy (Train) score of Logistic Regression  77.85016286644951
Accuracy (Test) score of Logistic Regression  76.62337662337663
Accuracy score of Logistic Regression  76.62337662337663
```

```
# Train & Test Scores of KNN
print("Accuracy (Train) score of KNN ",knn.score(X_train,y_train)*100)
print("Accuracy (Test) score of KNN ", knn.score(X_test,y_test)*100)
print("Accuracy score of KNN ", accuracy_score(y_test,knn_pred)*100)
```

```
Accuracy (Train) score of KNN  83.55048859934854
Accuracy (Test) score of KNN  74.67532467532467
Accuracy score of KNN  74.67532467532467
```

```
# Train & Test Scores of Naive-Bayes
print("Accuracy (Train) score of Naive Bayes ",nb.score(X_train,y_train)*100)
print("Accuracy (Test) score of Naive Bayes ", nb.score(X_test,y_test)*100)
print("Accuracy score of Naive Bayes ", accuracy_score(y_test,nb_pred)*100)
```

```
Accuracy (Train) score of Naive Bayes  76.54723127035831
Accuracy (Test) score of Naive Bayes  75.97402597402598
Accuracy score of Naive Bayes  75.97402597402598
```

```
# Train & Test Scores of SVM
print("Accuracy (Train) score of SVM ",sv.score(X_train,y_train)*100)
print("Accuracy (Test) score of SVM ", sv.score(X_test,y_test)*100)
print("Accuracy score of SVM ", accuracy_score(y_test,sv_pred)*100)
```

```
Accuracy (Train) score of SVM  77.19869706840392
Accuracy (Test) score of SVM  76.62337662337663
Accuracy score of SVM  76.62337662337663
```

```python
# Train & Test Scores of Decision Tree
print("Accuracy (Train) score of Decision Tree ",dt.score(X_train,y_train)*100)
print("Accuracy (Test) score of Decision Tree ", dt.score(X_test,y_test)*100)
print("Accuracy score of Decision Tree ", accuracy_score(y_test,dt_pred)*100)
```

```
Accuracy (Train) score of Decision Tree  100.0
Accuracy (Test) score of Decision Tree  73.37662337662337
Accuracy score of Decision Tree  73.37662337662337
```

```python
# Train & Test Scores of Random Forest
print("Accuracy (Train) score of Random Forest ",rf.score(X_train,y_train)*100)
print("Accuracy (Test) score of Random Forest ", rf.score(X_test,y_test)*100)
print("Accuracy score of Random Forest ", accuracy_score(y_test,rf_pred)*100)
```

```
Accuracy (Train) score of Random Forest  100.0
Accuracy (Test) score of Random Forest  74.67532467532467
Accuracy score of Random Forest  77.92532467532467
```

Step 10: Making a prediction System

```python
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = lr.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
```

Step 11: Saving the trained Model

```python
import pickle
filename = 'diabetes_model.sav'
pickle.dump(lr,open(filename,'wb'))
```

7.2) Heart Disease

Description

The dataset contains several medical predictor (Independent) variables and one target variable, (Outcome). Predictor variables include:

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

Dataset url: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Step 1: Importing the Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Step 2: Load the dataset

```
# loading the csv data to a Pandas DataFrame
heart_data = pd.read_csv('/content/heart.csv')
```

Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

3.1) Understanding Your Variables

3.1.1) Head of the dataset
3.1.2) The shape of the dataset
3.1.3) List types of columns
3.1.4) Info of the dataset
3.1.5) Summary of the dataset

3.1.1) Head of the Dataset

```
# Display first five records
heart_data.head()
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

```
# Display last five records
heart_data.tail()
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

3.1.2)The Shape of Dataset

```
# number of rows and columns in the dataset
heart_data.shape
```

```
(303, 14)
```

3.1.3)List types of columns

```
heart_data.dtypes
```

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

3.1.4)Info of Dataset

```
# getting some info about the data
heart_data.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

# checking for missing values
heart_data.isnull().sum()

age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64

# Statistical Summary
heart_data.describe()

              age         sex          cp    trestbps        chol         fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000
```

```
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000
75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

```
# checking the distribution of Target Variable
heart_data['target'].value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

1 --> Defective Heart

0 --> Healthy Heart

Step 4: Split the data frame in X & Y

```
X = heart_data.drop(columns='target', axis=1)
Y = heart_data['target']
```

```
X.head()
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal
0   0     1
1   0     2
2   0     2
3   0     2
4   0     2
```

```
Y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

Step 5: Applying Feature Scaling

Various Data Scaling Techniques:

- Normalizer
- MinMax Scaler
- Binarizer
- Standard Scaler

```
# Apply Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
SSX = scaler.fit_transform(X)
```

Step 6: Splitting the Data into Training data & Test Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

print(X.shape, X_train.shape, X_test.shape)

(303, 13) (212, 13) (91, 13)
```

Step 7: Building Classification Algorithm

7.1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,Y_train)

LogisticRegression(multi_class='ovr', solver='liblinear')
```

7.2) K-Nearest Neighbors Classifier(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,Y_train)

KNeighborsClassifier()
```

7.3) Naive-Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, Y_train)

GaussianNB()
```

7.4) Support Vector Machine (SVM)

```
from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train,Y_train)
```

```
SVC(kernel='linear')
```

7.5) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
```

```
DecisionTreeClassifier()
```

7.6) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=6)
rf.fit(X_train,Y_train)
```

```
RandomForestClassifier(max_depth=6, n_estimators=20, random_state=12)
```

Step 8: Making Prediction

8.1) Making Prediction using Logistic Regression

```
print(f'Initial shape: {X_test.shape}')
lr_pred = lr.predict(X_test)
print(f'{lr_pred.shape}')
```

```
Initial shape: (91, 13)
(91,)
```

8.2) Making Prediction using KNN

```
knn_pred = knn.predict(X_test)
knn_pred.shape
```

```
(91,)
```

8.3) Making Prediction using Naive Bayes

```
nb_pred = nb.predict(X_test)
nb_pred.shape
```

```
(91,)
```

8.4) Making Prediction using SVM

```
sv_pred = sv.predict(X_test)
sv_pred.shape
```

```
(91,)
```

8.5) Making Prediction using Decision Tree

```
dt_pred = dt.predict(X_test)
```

8.6) Making Prediciton using Random Forest

```
rf_pred = rf.predict(X_test)
```

Step 9: Model Evaluation

```
from sklearn.metrics import accuracy_score
```

```
# Train & Test Scores of Logistic Regression
print("Accuracy (Train) score of Logistic Regression ",lr.score(X_train,Y_train)*100)
```

```python
print("Accuracy (Test) score of Logistic Regression ", lr.score(X_test,Y_test)*100)
print("Accuracy score of Logistic Regression ", accuracy_score(Y_test,lr_pred)*100)
```

```
Accuracy (Train) score of Logistic Regression  86.32075471698113
Accuracy (Test) score of Logistic Regression  80.21978021978022
Accuracy score of Logistic Regression  80.21978021978022
```

```python
# Train & Test Scores of KNN
print("Accuracy (Train) score of KNN ",knn.score(X_train,Y_train)*100)
print("Accuracy (Test) score of KNN ", knn.score(X_test,Y_test)*100)
print("Accuracy score of KNN ", accuracy_score(Y_test,knn_pred)*100)
```

```
Accuracy (Train) score of KNN  77.83018867924528
Accuracy (Test) score of KNN  67.03296703296702
Accuracy score of KNN  67.03296703296702
```

```python
# Train & Test Scores of Naive-Bayes
print("Accuracy (Train) score of Naive Bayes ",nb.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Naive Bayes ", nb.score(X_test,Y_test)*100)
print("Accuracy score of Naive Bayes ", accuracy_score(Y_test,nb_pred)*100)
```

```
Accuracy (Train) score of Naive Bayes  84.43396226415094
Accuracy (Test) score of Naive Bayes  80.21978021978022
Accuracy score of Naive Bayes  80.21978021978022
```

```python
# Train & Test Scores of SVM
print("Accuracy (Train) score of SVM ",sv.score(X_train,Y_train)*100)
print("Accuracy (Test) score of SVM ", sv.score(X_test,Y_test)*100)
print("Accuracy score of SVM ", accuracy_score(Y_test,sv_pred)*100)
```

```
Accuracy (Train) score of SVM  85.37735849056604
Accuracy (Test) score of SVM  81.31868131868131
Accuracy score of SVM  81.31868131868131
```

```python
# Train & Test Scores of Decision Tree
print("Accuracy (Train) score of Decision Tree ",dt.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Decision Tree ", dt.score(X_test,Y_test)*100)
print("Accuracy score of Decision Tree ", accuracy_score(Y_test,dt_pred)*100)
```

```
Accuracy (Train) score of Decision Tree  100.0
Accuracy (Test) score of Decision Tree  72.52747252747253
Accuracy score of Decision Tree  72.52747252747253
```

```python
# Train & Test Scores of Random Forest
print("Accuracy (Train) score of Random Forest ",rf.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Random Forest ", rf.score(X_test,Y_test)*100)
print("Accuracy score of Random Forest ", accuracy_score(Y_test,rf_pred)*100)
```

```
Accuracy (Train) score of Random Forest  98.11320754716981
Accuracy (Test) score of Random Forest  82.41758241758241
Accuracy score of Random Forest  82.41758241758241
```

Step 10: Building a Predictive System

```python
input_data = (63,1,3,145,233,1,0,150,0,2.3,0,0,1)

# change the input data to a numpy array
```

```
input_data_as_numpy_array= np.asarray(input_data)

# reshape the numpy array as we are predicting for only on instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = dt.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```

```
[1]
The Person has Heart Disease
```

Step 11 : Saving the trained model

```
import pickle
import pickle
filename = 'heart_disease_model.sav'
pickle.dump(lr,open(filename,'wb'))
```

7.3) Parkinsons Disease

Description

The dataset was created by Max Little of the University of Oxford, in collaboration with the National Centre for Voice and Speech, Denver, Colorado, who recorded the speech signals. The original study published the feature extraction methods for general voice disorders.

- MDVP:Fo(Hz) - Average vocal fundamental frequency
- MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
- MDVP:Flo(Hz) - Minimum vocal fundamental frequency
- MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several
- measures of variation in fundamental frequency
- MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude
- NHR,HNR - Two measures of ratio of noise to tonal components in the voice
- status - Health status of the subject (one) - Parkinson's, (zero) - healthy
- RPDE,D2 - Two nonlinear dynamical complexity measures
- DFA - Signal fractal scaling exponent
- spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

https://www.kaggle.com/datasets/thecansin/parkinsons-data-set

Step 1 : Importing the Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Step 2: Loading the dataset

```
# loading the csv data to a Pandas DataFrame
parkinsons_data = pd.read_csv('/content/parkinsons.csv')
```

Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

3.1) Understanding Your Variables

3.1.1) Head of the dataset
3.1.2) The shape of the dataset
3.1.3) List types of columns
3.1.4) Info of the dataset
3.1.5) Summary of the dataset

3.1.1) Head of the Dataset

```
# Display first five records
parkinsons_data.head()
```

```
           name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
0  phon_R01_S01_1      119.992       157.302        74.997         0.00784
1  phon_R01_S01_2      122.400       148.650       113.819         0.00968
2  phon_R01_S01_3      116.682       131.111       111.555         0.01050
3  phon_R01_S01_4      116.676       137.871       111.366         0.00997
4  phon_R01_S01_5      116.014       141.781       110.655         0.01284

   MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  ...  \
0           0.00007   0.00370   0.00554     0.01109       0.04374  ...
1           0.00008   0.00465   0.00696     0.01394       0.06134  ...
2           0.00009   0.00544   0.00781     0.01633       0.05233  ...
3           0.00009   0.00502   0.00698     0.01505       0.05492  ...
4           0.00011   0.00655   0.00908     0.01966       0.06425  ...

   Shimmer:DDA      NHR     HNR  status      RPDE       DFA   spread1  \
0      0.06545  0.02211  21.033       1  0.414783  0.815285 -4.813031
1      0.09403  0.01929  19.085       1  0.458359  0.819521 -4.075192
2      0.08270  0.01309  20.651       1  0.429895  0.825288 -4.443179
3      0.08771  0.01353  20.644       1  0.434969  0.819235 -4.117501
4      0.10470  0.01767  19.649       1  0.417356  0.823484 -3.747787

    spread2        D2       PPE
0  0.266482  2.301442  0.284654
1  0.335590  2.486855  0.368674
2  0.311173  2.342259  0.332634
3  0.334147  2.405554  0.368975
4  0.234513  2.332180  0.410335
```

```
[5 rows x 24 columns]
```

```
# Display last five records
parkinsons_data.tail()
```

```
           name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
190  phon_R01_S50_2      174.188       230.978        94.261         0.00459
191  phon_R01_S50_3      209.516       253.017        89.488         0.00564
192  phon_R01_S50_4      174.688       240.005        74.287         0.01360
193  phon_R01_S50_5      198.764       396.961        74.904         0.00740
194  phon_R01_S50_6      214.289       260.277        77.973         0.00567

     MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  ...  \
190           0.00003   0.00263   0.00259     0.00790       0.04087  ...
191           0.00003   0.00331   0.00292     0.00994       0.02751  ...
192           0.00008   0.00624   0.00564     0.01873       0.02308  ...
193           0.00004   0.00370   0.00390     0.01109       0.02296  ...
194           0.00003   0.00295   0.00317     0.00885       0.01884  ...

     Shimmer:DDA      NHR     HNR  status      RPDE       DFA   spread1  \
190      0.07008  0.02764  19.517       0  0.448439  0.657899 -6.538586
191      0.04812  0.01810  19.147       0  0.431674  0.683244 -6.195325
192      0.03804  0.10715  17.883       0  0.407567  0.655683 -6.787197
193      0.03794  0.07223  19.020       0  0.451221  0.643956 -6.744577
194      0.03078  0.04398  21.209       0  0.462803  0.664357 -5.724056

      spread2        D2       PPE
190  0.121952  2.657476  0.133050
191  0.129303  2.784312  0.168895
192  0.158453  2.679772  0.131728
193  0.207454  2.138608  0.123306
194  0.190667  2.555477  0.148569

[5 rows x 24 columns]
```

3.1.2) The Shape of Dataset

```
parkinsons_data.shape
```

```
(195, 24)
```

3.1.3)List types of columns

```
parkinsons_data.dtypes
```

```
name                 object
MDVP:Fo(Hz)         float64
MDVP:Fhi(Hz)        float64
MDVP:Flo(Hz)        float64
MDVP:Jitter(%)      float64
MDVP:Jitter(Abs)    float64
MDVP:RAP            float64
MDVP:PPQ            float64
Jitter:DDP          float64
MDVP:Shimmer        float64
MDVP:Shimmer(dB)    float64
```

```
Shimmer:APQ3       float64
Shimmer:APQ5       float64
MDVP:APQ           float64
Shimmer:DDA        float64
NHR                float64
HNR                float64
status             int64
RPDE               float64
DFA                float64
spread1            float64
spread2            float64
D2                 float64
PPE                float64
dtype: object
```

3.1.4)Info of Dataset

```
# getting some info about the data
parkinsons_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              195 non-null    object
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
 20  spread1           195 non-null    float64
 21  spread2           195 non-null    float64
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

```
# checking for missing values
parkinsons_data.isnull().sum()
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

```
# Statistical Summary
parkinsons_data.describe()
```

```
       MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
count   195.000000    195.000000    195.000000      195.000000
mean    154.228641    197.104918    116.324631        0.006220
std      41.390065     91.491548     43.521413        0.004848
min      88.333000    102.145000     65.476000        0.001680
25%     117.572000    134.862500     84.291000        0.003460
50%     148.790000    175.829000    104.315000        0.004940
75%     182.769000    224.205500    140.018500        0.007365
max     260.105000    592.030000    239.170000        0.033160

       MDVP:Jitter(Abs)    MDVP:RAP    MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
count        195.000000  195.000000  195.000000  195.000000    195.000000
mean           0.000044    0.003306    0.003446    0.009920      0.029709
std            0.000035    0.002968    0.002759    0.008903      0.018857
min            0.000007    0.000680    0.000920    0.002040      0.009540
25%            0.000020    0.001660    0.001860    0.004985      0.016505
50%            0.000030    0.002500    0.002690    0.007490      0.022970
75%            0.000060    0.003835    0.003955    0.011505      0.037885
max            0.000260    0.021440    0.019580    0.064330      0.119080

       MDVP:Shimmer(dB)  ...  Shimmer:DDA         NHR         HNR      status  \
count        195.000000  ...   195.000000  195.000000  195.000000  195.000000
mean           0.282251  ...     0.046993    0.024847   21.885974    0.753846
std            0.194877  ...     0.030459    0.040418    4.425764    0.431878
min            0.085000  ...     0.013640    0.000650    8.441000    0.000000
```

```
25%          0.148500  ...     0.024735    0.005925   19.198000    1.000000
50%          0.221000  ...     0.038360    0.011660   22.085000    1.000000
75%          0.350000  ...     0.060795    0.025640   25.075500    1.000000
max          1.302000  ...     0.169420    0.314820   33.047000    1.000000

            RPDE         DFA     spread1     spread2         D2         PPE
count  195.000000  195.000000  195.000000  195.000000  195.000000  195.000000
mean     0.498536    0.718099   -5.684397    0.226510    2.381826    0.206552
std      0.103942    0.055336    1.090208    0.083406    0.382799    0.090119
min      0.256570    0.574282   -7.964984    0.006274    1.423287    0.044539
25%      0.421306    0.674758   -6.450096    0.174351    2.099125    0.137451
50%      0.495954    0.722254   -5.720868    0.218885    2.361532    0.194052
75%      0.587562    0.761881   -5.046192    0.279234    2.636456    0.252980
max      0.685151    0.825288   -2.434031    0.450493    3.671155    0.527367

[8 rows x 23 columns]
```

```
# checking the distribution of target Variable
parkinsons_data['status'].value_counts()
```

```
1    147
0     48
Name: status, dtype: int64
```

1 --> Parkinson's Positive

2 --> Healthy

```
# grouping the data based on the target variable
parkinsons_data.groupby('status').mean()
```

```
        MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
status
0        181.937771    223.636750    145.207292        0.003866
1        145.180762    188.441463    106.893558        0.006989

        MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
status
0               0.000023  0.001925  0.002056    0.005776      0.017615
1               0.000051  0.003757  0.003900    0.011273      0.033658

        MDVP:Shimmer(dB)  ...  MDVP:APQ  Shimmer:DDA       NHR        HNR  \
status                    ...
0               0.162958  ...  0.013305     0.028511  0.011483  24.678750
1               0.321204  ...  0.027600     0.053027  0.029211  20.974048

            RPDE       DFA   spread1   spread2        D2       PPE
status
0       0.442552  0.695716 -6.759264  0.160292  2.154491  0.123017
1       0.516816  0.725408 -5.333420  0.248133  2.456058  0.233828

[2 rows x 22 columns]
```

Step 4: Split the data frame in X & Y

```
X = parkinsons_data.drop(columns=['name','status'], axis=1)
Y = parkinsons_data['status']
```

```
X.head()
```

```
   MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  MDVP:Jitter(Abs)  \
0      119.992       157.302        74.997         0.00784           0.00007
1      122.400       148.650       113.819         0.00968           0.00008
2      116.682       131.111       111.555         0.01050           0.00009
3      116.676       137.871       111.366         0.00997           0.00009
4      116.014       141.781       110.655         0.01284           0.00011

   MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  MDVP:Shimmer(dB)  ...  \
0   0.00370   0.00554     0.01109       0.04374             0.426  ...
1   0.00465   0.00696     0.01394       0.06134             0.626  ...
2   0.00544   0.00781     0.01633       0.05233             0.482  ...
3   0.00502   0.00698     0.01505       0.05492             0.517  ...
4   0.00655   0.00908     0.01966       0.06425             0.584  ...

   MDVP:APQ  Shimmer:DDA      NHR     HNR      RPDE       DFA   spread1  \
0   0.02971      0.06545  0.02211  21.033  0.414783  0.815285 -4.813031
1   0.04368      0.09403  0.01929  19.085  0.458359  0.819521 -4.075192
2   0.03590      0.08270  0.01309  20.651  0.429895  0.825288 -4.443179
3   0.03772      0.08771  0.01353  20.644  0.434969  0.819235 -4.117501
4   0.04465      0.10470  0.01767  19.649  0.417356  0.823484 -3.747787

    spread2        D2       PPE
0  0.266482  2.301442  0.284654
1  0.335590  2.486855  0.368674
2  0.311173  2.342259  0.332634
3  0.334147  2.405554  0.368975
4  0.234513  2.332180  0.410335

[5 rows x 22 columns]
```

```
Y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: status, dtype: int64
```

Step 5: Splitting the Data into Training data & Test Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(195, 22) (156, 22) (39, 22)
```

Step 6: Building Classification Algorithm

6.1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,Y_train)
```

```
LogisticRegression(multi_class='ovr', solver='liblinear')
```

6.2) K-Nearest Neighbors Classifier(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,Y_train)
```

```
KNeighborsClassifier()
```

6.3) Naive-Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, Y_train)
```

```
GaussianNB()
```

6.4) Support Vector Machine (SVM)

```
from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train,Y_train)
```

```
SVC(kernel='linear')
```

6.5) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
```

```
DecisionTreeClassifier()
```

6.6) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='entropy')
rf.fit(X_train,Y_train)
```

```
RandomForestClassifier(criterion='entropy')
```

Step 7: Making Prediction

7.1) Making Prediction using Logistic Regression

```
print(f'Initial shape: {X_test.shape}')
lr_pred = lr.predict(X_test)
print(f'{lr_pred.shape}')
```

```
Initial shape: (39, 22)
(39,)
```

7.2) Making Prediction using KNN

```
knn_pred = knn.predict(X_test)
knn_pred.shape
```

```
(39,)
```

7.3) Making Prediction using Naive Bayes

```
nb_pred = nb.predict(X_test)
nb_pred.shape
```

```
(39,)
```

7.4) Making Prediction using SVM

```
sv_pred = sv.predict(X_test)
sv_pred.shape
```

```
(39,)
```

7.5) Making Prediction using Decision Tree

```
dt_pred = dt.predict(X_test)
```

7.6) Making Prediciton using Random Forest

```
rf_pred = rf.predict(X_test)
```

Step 8: Model Evaluation

```
from sklearn.metrics import accuracy_score
```

```
# Train & Test Scores of Logistic Regression
print("Accuracy (Train) score of Logistic Regression ",lr.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Logistic Regression ", lr.score(X_test,Y_test)*100)
print("Accuracy score of Logistic Regression ", accuracy_score(Y_test,lr_pred)*100)
```

```
Accuracy (Train) score of Logistic Regression  87.17948717948718
Accuracy (Test) score of Logistic Regression  87.17948717948718
Accuracy score of Logistic Regression  87.17948717948718
```

```
# Train & Test Scores of KNN
print("Accuracy (Train) score of KNN ",knn.score(X_train,Y_train)*100)
print("Accuracy (Test) score of KNN ", knn.score(X_test,Y_test)*100)
print("Accuracy score of KNN ", accuracy_score(Y_test,knn_pred)*100)
```

```
Accuracy (Train) score of KNN  87.17948717948718
Accuracy (Test) score of KNN  74.35897435897436
Accuracy score of KNN  74.35897435897436
```

```
# Train & Test Scores of Naive-Bayes
print("Accuracy (Train) score of Naive Bayes ",nb.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Naive Bayes ", nb.score(X_test,Y_test)*100)
print("Accuracy score of Naive Bayes ", accuracy_score(Y_test,nb_pred)*100)
```

```
Accuracy (Train) score of Naive Bayes  73.71794871794873
Accuracy (Test) score of Naive Bayes  58.97435897435898
Accuracy score of Naive Bayes  58.97435897435898
```

```
# Train & Test Scores of SVM
print("Accuracy (Train) score of SVM ",sv.score(X_train,Y_train)*100)
print("Accuracy (Test) score of SVM ", sv.score(X_test,Y_test)*100)
print("Accuracy score of SVM ", accuracy_score(Y_test,sv_pred)*100)
```

```
Accuracy (Train) score of SVM  87.17948717948718
Accuracy (Test) score of SVM  87.17948717948718
Accuracy score of SVM  87.17948717948718
```

```
# Train & Test Scores of Decision Tree
print("Accuracy (Train) score of Decision Tree ",dt.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Decision Tree ", dt.score(X_test,Y_test)*100)
print("Accuracy score of Decision Tree ", accuracy_score(Y_test,dt_pred)*100)
```

```
Accuracy (Train) score of Decision Tree  100.0
Accuracy (Test) score of Decision Tree  74.35897435897436
Accuracy score of Decision Tree  74.35897435897436
```

```
# Train & Test Scores of Random Forest
print("Accuracy (Train) score of Random Forest ",rf.score(X_train,Y_train)*100)
print("Accuracy (Test) score of Random Forest ", rf.score(X_test,Y_test)*100)
print("Accuracy score of Random Forest ", accuracy_score(Y_test,rf_pred)*100)
```

```
Accuracy (Train) score of Random Forest  100.0
Accuracy (Test) score of Random Forest  82.05128205128204
Accuracy score of Random Forest  82.05128205128204
```

Step 9: Building a Predictive System

```
input_data = (197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.01098,0.09700,0.00563,0.00680,0.00802,0.01689,0.00339,26.77500,0.422229,0.741367,-7.348300,0.177551,1.743867,0.085569)

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = sv.predict(input_data_reshaped)
print(prediction)


if (prediction[0] == 0):
  print("The Person does not have Parkinsons Disease")

else:
  print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease
```

Step 10: Saving the trained model

```
import pickle
filename = 'parkinsons_model.sav'
pickle.dump(sv, open(filename, 'wb'))
```

7.4) Front-End

```
import pickle
import streamlit as st
from streamlit_option_menu import option_menu
from sklearn import datasets
import requests
```

```python
from streamlit_lottie import st_lottie

# loading the saved models

diabetes_model = pickle.load(open('./diabetes_model.sav', 'rb'))

heart_disease_model = pickle.load(open('./heart_disease_model.sav', 'rb'))

parkinsons_model = pickle.load(open('./parkinsons_model.sav', 'rb'))


# sidebar for navigation
with st.sidebar:

    selected = st.selectbox('Multiple Disease Prediction System',
                            ['Home',
                             'Diabetes Prediction',
                             'Heart Disease Prediction',
                             'Parkinsons Prediction'],)


if (selected == 'Home'):
    st.title("Multiple Disease Prediction using Machine Learning")

    def load_lottieurl(url: str):
        r = requests.get(url)
        if r.status_code != 200:
            return None
        return r.json()
    lottie_url_hello =
"https://assets3.lottiefiles.com/packages/lf20_0ssane8p.json"
    lottie_hello = load_lottieurl(lottie_url_hello)
    st_lottie(lottie_hello, key="hello")

# 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'):
        diab_prediction = diabetes_model.predict(
            [[Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI,
DiabetesPedigreeFunction, Age]])

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

    st.success(diab_diagnosis)


# 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.number_input('Age', step=1, max_value=110)

    with col2:
        sex = st.selectbox('Sex (female=0, male=1)', (0, 1))

    with col3:
        cp = st.selectbox('Chest pain type', (0, 1, 2, 3))

    with col1:
        trestbps = st.number_input('Resting Blood Pressure', step=1)

    with col2:
        chol = st.number_input('Serum Cholestoral in mg/dl', step=1)

    with col3:
        fbs = st.selectbox('Fasting blood sugar', (0, 1))

    with col1:
        restecg = st.number_input(
```

```
            'Resting Electrocardiographic results', step=1)

    with col2:
        thalach = st.number_input('Maximum Heart Rate achieved', step=1)

    with col3:
        exang = st.number_input('Exercise Induced Angina', step=1)

    with col1:
        oldpeak = st.number_input(
            'ST depression induced by exercise', step=0.1)

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

    with col3:
        ca = st.selectbox('Major vessels colored by flourosopy', (0, 1, 2, 3))

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

    # code for Prediction
    heart_diagnosis = ''

    # creating a button for Prediction

    if st.button('Heart Disease Test Result'):
        heart_prediction = heart_disease_model.predict(
            [[age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak,
slope, ca, thal]])
        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)


# Parkinson's Prediction Page
if (selected == "Parkinsons Prediction"):

    # page title
    st.title("Parkinson's Disease Prediction using ML")

    col1, col2, col3, col4, col5 = st.columns(5)

    with col1:
        fo = st.text_input('MDVP:Fo(Hz)')

    with col2:
        fhi = st.text_input('MDVP:Fhi(Hz)')

    with col3:
        flo = st.text_input('MDVP:Flo(Hz)')

    with col4:
```

```python
    Jitter_percent = st.text_input('MDVP:Jitter(%)')

with col5:
    Jitter_Abs = st.text_input('MDVP:Jitter(Abs)')

with col1:
    RAP = st.text_input('MDVP:RAP')

with col2:
    PPQ = st.text_input('MDVP:PPQ')

with col3:
    DDP = st.text_input('Jitter:DDP')

with col4:
    Shimmer = st.text_input('MDVP:Shimmer')

with col5:
    Shimmer_dB = st.text_input('MDVP:Shimmer(dB)')

with col1:
    APQ3 = st.text_input('Shimmer:APQ3')

with col2:
    APQ5 = st.text_input('Shimmer:APQ5')

with col3:
    APQ = st.text_input('MDVP:APQ')

with col4:
    DDA = st.text_input('Shimmer:DDA')

with col5:
    NHR = st.text_input('NHR')

with col1:
    HNR = st.text_input('HNR')

with col2:
    RPDE = st.text_input('RPDE')

with col3:
    DFA = st.text_input('DFA')

with col4:
    spread1 = st.text_input('spread1')

with col5:
    spread2 = st.text_input('spread2')

with col1:
    D2 = st.text_input('D2')

with col2:
    PPE = st.text_input('PPE')
```

```
    # code for Prediction
    parkinsons_diagnosis = ''

    # creating a button for Prediction
    if st.button("Parkinson's Test Result"):
        parkinsons_prediction = parkinsons_model.predict(
            [[fo, fhi, flo, Jitter_percent, Jitter_Abs, RAP, PPQ, DDP, Shimmer,
Shimmer_dB, APQ3, APQ5, APQ, DDA, NHR, HNR, RPDE, DFA, spread1, spread2, D2, PPE]])

        if (parkinsons_prediction[0] == 1):
            parkinsons_diagnosis = "The person has Parkinson's disease"
        else:
            parkinsons_diagnosis = "The person does not have Parkinson's disease"

    st.success(parkinsons_diagnosis)
```

8. CONCLUSION AND FUTURE SCOPE

Multiple disease prediction model is the system in which the users are allowed to input their health records that they have obtained from various prescribed tests' reports. Disease predictions with this model can be considered as an intermediate step between taking up the prescribed health check-up test and consultation with the medical practitioners.

The benefit of multiple disease prediction model is that it can predict the occurrence probability of various diseases in advance, thereby reducing the mortality ratio. The process of prediction starts from cleaning and processing of data, imputation of missing values, experimental analysis of data set and then model building to evaluation of model and testing on test data.

In future, the developers can add any type and number of diseases with better scalability on datasets. If the dataset, which is going to be collected in the future, has records of patients from various birth places all over the world, then its trained model will be more efficient than the proposed one. Moreover, the larger the dataset, more will be the benefit to be gained. As a result of that, the model may be recommended to more users irrespective of geographical locations and its conditions.

The accuracy scores of algorithms for specific disease are given below

Algorithms	**Accuracy**
RFA	79
SVM	75
LR	76
KNN	75
DT	71

Table. i. Accuracy Scores for Diabetes

Algorithms	**Accuracy**
RFA	82
SVM	81
LR	80
KNN	67
NB	80
DT	72

Table. ii. Accuracy Scores for Heart Disease

Algorithms	**Accuracy**
RFA	82
SVM	86
LR	87
KNN	74
NB	58
DT	74

Table. iii. Accuracy Scores for Parkinsons

9. REREFENCES

[1] Multi Disease Prediction System using Random Forest Algorithm in Healthcare System
By R. Shanthakumari, C. Nalini; S. Vinothkumar, E.M. Roopadevi, B. Govindaraj
Year of Production: 14 April 2022

[2] Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques
By Senthilkumar Mohan, Chandrasegar Thirumalai, Gautam Srivastava
Year of Production: 19 June 2019

[3] Designing Disease Prediction Model Using Machine Learning Approach
By Dhiraj Dahiwade, Gajanan Patle, Ektaa Meshram
Year of Production: 29 March 2019

[4] Diabetes Disease Prediction Using Machine Learning
By Preetha, Chandan, Darshan N, Gowrav P
Year of Production: 5 May 2020

[5] Survey on data mining techniques for disease prediction
By Kinge, Durga, and S. K. Gaikwad.

[6] Heart disease prediction using data mining
By Patel, Ajad, et al.

[7] Iyer, A., Jeyalatha, S. and Sumbaly, R. (2015) Diagnosis of Diabetes Using Classification Mining Techniques. International Journal of Data Mining & Knowledge Management Process (IJDKP), 5, 1-14. *Link:* ***https://doi.org/10.5121/ijdkp.2015.5101***

[8] Kumari, V.A. and Chitra, R. (2013) Classification of Diabetes Disease Using Support Vector Machine. International Journal of Engineering Research and Applications (IJERA), 3, 1797-1801.

[9] Sarwar, A. and Sharma, V. (2012) Intelligent Naive Bayes Approach to Diagnose Diabetes Type-2. Special Issue of International Journal of Computer Applications (0975-8887) on Issues and Challenges in Networking, Intelligence and Computing Technologies-ICNICT 2012, 3, 14-16.

[10] Ephzibah, E.P. (2011) Cost Effective Approach on Feature Selection using Genetic Algorithms and Fuzzy Logic for Diabetes Diagnosis. International Journal on Soft Computing (IJSC), 2, 1-10.
Link: ***https://doi.org/10.5121/ijsc.2011.2101***