1. INTRODUCTION

Diabetes is the fast-growing disease among the people even among the youngsters. In understanding diabetes and how it develops, we need to understand what happens in the body without diabetes. Sugar (glucose) comes from the foods that we eat, specifically carbohydrate foods. Carbohydrate foods provide our body with its main energy source everybody, even those people with diabetes, needs carbohydrate. Carbohydrate foods include bread, cereal, pasta, rice, fruit, dairy products and vegetables (especially starchy vegetables). When we eat these foods, the body breaks them down into glucose. The glucose moves around the body in the bloodstream. Some of the glucose is taken to our brain to help us think clearly and function. The remainder of the glucose is taken to the cells of our body for energy and also to our liver, where it is stored as energy that is used later by the body. In order for the body to use glucose for energy, insulin is required. Insulin attaches itself to doors on the cell, opening the door to allow glucose to move from the blood stream, through the door, and into the cell. If the pancreas is not able to produce enough insulin (insulin deficiency) or if the body cannot use the insulin it produces (insulin resistance), glucose builds up in the bloodstream (hyperglycaemia) and diabetes develops. Diabetes Mellitus means high levels of sugar (glucose) in the blood stream and in the urine.

1.1 Types of Diabetes

Type 1 diabetes means that the immune system is compromised and the cells fail to produce insulin in sufficient amounts. There are no eloquent studies that prove the causes of type 1 diabetes and there are currently no known methods of prevention.

Type 2 diabetes means that the cells produce a low quantity of insulin or the body can’t use the insulin correctly. This is the most common type of diabetes, thus affecting 90% of persons diagnosed with diabetes. It is caused by both genetic factors and the manner of living.

Gestational diabetes appears in pregnant women who suddenly develop high blood sugar. In two thirds of the cases, it will reappear during subsequent pregnancies. There is a great chance that type 1 or type 2 diabetes will occur after a pregnancy affected by gestational diabetes.

1.2 Symptoms of Diabetes

➢ Frequent Urination

➢ Increased thirst

➢ Tired/Sleepiness

➢ Weight loss

➢ Blurred vision

➢ Mood swings

➢ Confusion and difficulty concentrating

➢ frequent infections

1.3 Causes of Diabetes

Genetic factors are the main cause of diabetes. It is caused by at least two mutant genes in the chromosome 6, the chromosome that affects the response of the body to various antigens. Viral infection may also influence the occurrence of type 1 and type 2 diabetes. Studies have shown that infection with viruses such as rubella, Coxsackievirus, mumps, hepatitis B virus, and cytomegalovirus increase the risk of developing diabetes.

2. LITERATURE REVIEW

A literature review is the writing process of summarizing, synthesizing and/or critiquing the literature found as a result of a literature search. It may be used as background or context fora primary research project.

1. S. S. Gitanjali et al. - "A Comparative Study of Machine Learning Algorithms for Diabetic Retinopathy Prediction" (2020): This study compared various machine learning algorithms for the prediction of diabetic retinopathy. The findings revealed that ensemble methods such as Random Forest and Gradient Boosting exhibited superior performance compared to other algorithms like Logistic Regression and Support Vector Machine. This superiority was demonstrated through higher accuracy rates in predicting diabetic retinopathy, underscoring the efficacy of ensemble methods in handling the complexities of retinal data.

2. V. K. Jayaraman et al. - "Comparison of Machine Learning Algorithms for Diabetes Prediction" (2019): Focused on predicting diabetes, this study evaluated the performance of different machine learning algorithms. The results indicated that Random Forest outperformed other algorithms including k-Nearest Neighbors and Naive Bayes. With higher accuracy and F1-score, Random Forest emerged as a robust choice for diabetes prediction tasks, showcasing its effectiveness in handling diverse datasets and feature complexities associated with diabetic prediction.

3. A. Rajput et al. - "A Comparative Study of Machine Learning Algorithms for Diabetes Prediction" (2021): This study aimed to predict diabetes using various machine learning algorithms and compare their performance. It was found that Support Vector Machine exhibited the highest accuracy in predicting diabetes compared to Decision Trees and Naive Bayes. However, Decision Trees demonstrated better interpretability, highlighting the trade-offs between accuracy and model transparency in diabetic prediction tasks.

4. R. Gupta et al. - "Comparative Analysis of Machine Learning Algorithms for Diabetes Prediction" (2018): Focusing on diabetes prediction, this study conducted a comparative analysis of machine learning algorithms. The results indicated that Neural Networks, Support Vector Machine, and Random Forest showed promising results in predicting diabetes. However, logistic regression displayed lower performance in terms of accuracy and F1-score, emphasizing the importance of choosing appropriate algorithms for diabetic prediction tasks based on performance metrics.

5. P. Patel et al. - "Performance Evaluation of Machine Learning Algorithms for Diabetes Prediction" (2022): This study evaluated the performance of machine learning algorithms for diabetes prediction. The results revealed that Boost and Random Forest outperformed other algorithms, including Logistic Regression and Decision Trees, in terms of accuracy and AUC-ROC score. However, Logistic Regression showed better interpretability, highlighting the trade-offs between accuracy and model transparency in diabetic prediction tasks.

6. S. Mishra et al. - "Comparative Analysis of Machine Learning Algorithms for Diabetic Retinopathy Prediction" (2023): Focused on predicting diabetic retinopathy, this study compared machine learning algorithms for their performance. The findings indicated that Gradient Boosting Machines exhibited superior performance compared to Logistic Regression and k-Nearest Neighbors. With a higher area under the ROC curve and accuracy, Gradient Boosting Machines showcased their effectiveness in handling the complexities of diabetic retinopathy prediction tasks.

Table 2.1 Literature Review

S.NO	AUTHOR	TITLE	YEAR	CONCLUSIONS
1.	S. S. Gitanjali et al.	"A Comparative Study of Machine Learning Algorithms for Diabetic Retinopathy Prediction"	2020	Ensemble methods such as Random Forest and Gradient Boosting showed superior performance compared to other algorithms like Logistic Regression and Support Vector Machine in predicting diabetic retinopathy.
2.	V. K. Jayaraman et al.	"Comparison of Machine Learning Algorithms for Diabetes Prediction"	2019	Random Forest outperformed other algorithms including k-Nearest Neighbours and Naive Bayes in diabetes prediction, with a higher accuracy and F1-score.
3.	A. Rajput et al.	"A Comparative Study of Machine Learning	2021	Support Vector Machine demonstrated the highest accuracy in predicting

		Algorithms for Diabetes Prediction"		diabetes compared to Decision Trees and Naive Bayes. However, Decision Trees exhibited better interpretability.
4.	R. Gupta et al.	"Comparative Analysis of Machine Learning Algorithms for Diabetes Prediction"	2018	Neural Networks showed promising results in diabetes prediction, closely followed by Support Vector Machine and Random Forest. However, logistic regression displayed lower performance in terms of accuracy and F1-score.
5.	P. Patel et al.	"Performance Evaluation of Machine Learning Algorithms for Diabetes Prediction"	2022	XGBoost and Random Forest outperformed other algorithms including Logistic Regression and Decision Trees in terms of accuracy and AUC-ROC score in diabetes prediction. However, Logistic Regression showed better interpretability.
6.	S. Mishra et al.	"Comparative Analysis of Machine Learning Algorithms for Diabetic Retinopathy Prediction"	2023	Gradient Boosting Machines exhibited superior performance in predicting diabetic retinopathy compared to Logistic Regression and k-Nearest Neighbours, with a higher area under the ROC curve and accuracy.

3. PROBLEM STATEMENT

- The problem statement is to determine which algorithm gives more accuracy, Recall, Pression, F1-Score and Mean Square Error for predicting diabetes.
- The methodology involves implementing 6 classification algorithms such as logistic Regression, K-Nearest Neighbors, Naïve Bays, Support Vector Machine, Decision Tree and Random Forest on the dataset and evaluating their performance using metrics such as accuracy.

4. DESIGN ARCHITECTURE

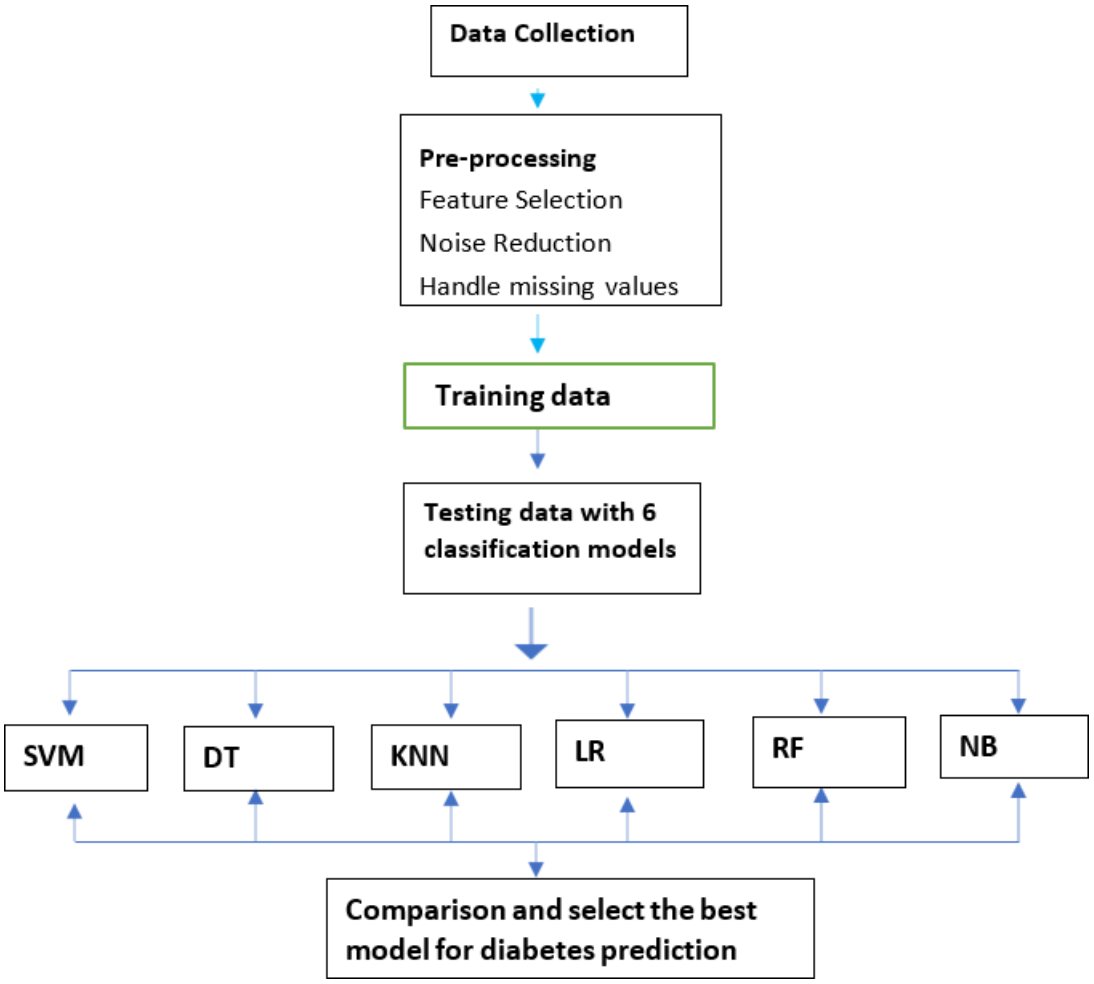

Fig 4.1 Design Architecture

5. REQUIREMENTS SPECIFICATIONS

❖ Hardware requirements

- Processor : Any processor above 500 MHz
- RAM : 4GB
- Hard Disk : 4GB

❖ Software requirements

- Operating System: Windows 10/11
- Programming : Python 3.10.4 and related libraries
- IDE : Visual Studio Code

6. DATASET DESCRIPTION

The diabetes data set was originated from **https://www.kaggle.com/johndasilva/diabetes.** Diabetes dataset containing 2000 cases. The objective is to predict based on the measures to predict if the patient is diabetic or not. The 8 feature variables along with the target variable are shown in the following table(Table 1).

Table 6.1 Dataset Description

No	Name	Description	Type
1.	Pregnancies	Number of times pregnant	Numeric
2.	Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test Numeric	Numeric
3.	Blood Pressure	Diastolic blood pressure	Numeric
4.	Skin Thickness	Triceps skinfold thickness	Numeric
5.	Insulin	2-hour serum insulin	Numeric
6.	BMI	Body mass index	Numeric
7.	Diabetes Pedigree Function	Diabetes pedigree function	Numeric
8.	Age	Patient's age	Numeric
9.	Outcome	Target variable (1 if diabetic, else 0)	Binary (0 or 1)

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	2	138	62	35	0	33.6	0.127	47	1
1	0	84	82	31	125	38.2	0.233	23	0
2	0	145	0	0	0	44.2	0.630	31	1
3	0	135	68	42	250	42.3	0.365	24	1
4	1	139	62	41	480	40.7	0.536	21	0

Fig 6.1 Dataset top 5 rows

7. ALGORITHMS

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. Traditional programming approaches take in data and rule to produce the desired output. Whereas, machine learning approaches take in data and the desired output as the input and output the necessary rules.

Supervised Learning Algorithms use the input features along with the target output for training, in contrast to the unsupervised learning algorithms that take only the input features for training. Supervised Learning Algorithms try to find the mapping between the inputs and the outputs. Example: regression, classification.

Classification is a supervised learning problem where the output to be predicted is discrete numeric values representing different classes. For example, predicting whether an image is a dog or a cat if a student will pass an examination or not if a review is positive or negative, and so on.

7.1 Logistic Regression

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- t can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.

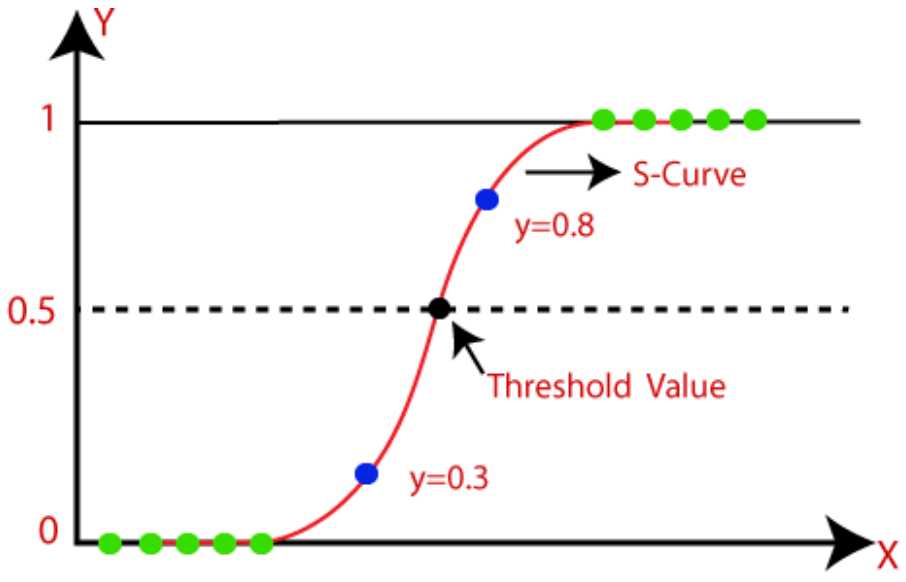

Fig 7.1.1 Logistic Regression

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}\text{; 0 for y= 0, and infinity for y=1}$$

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

7.2 K-Nearest Neighbours

- K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

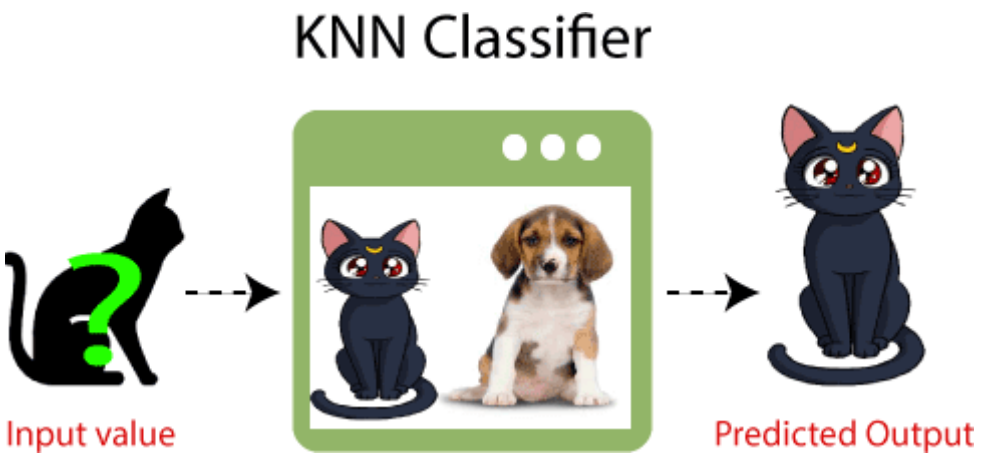

Fig 7.2.1 K-Nearest Neighbours

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

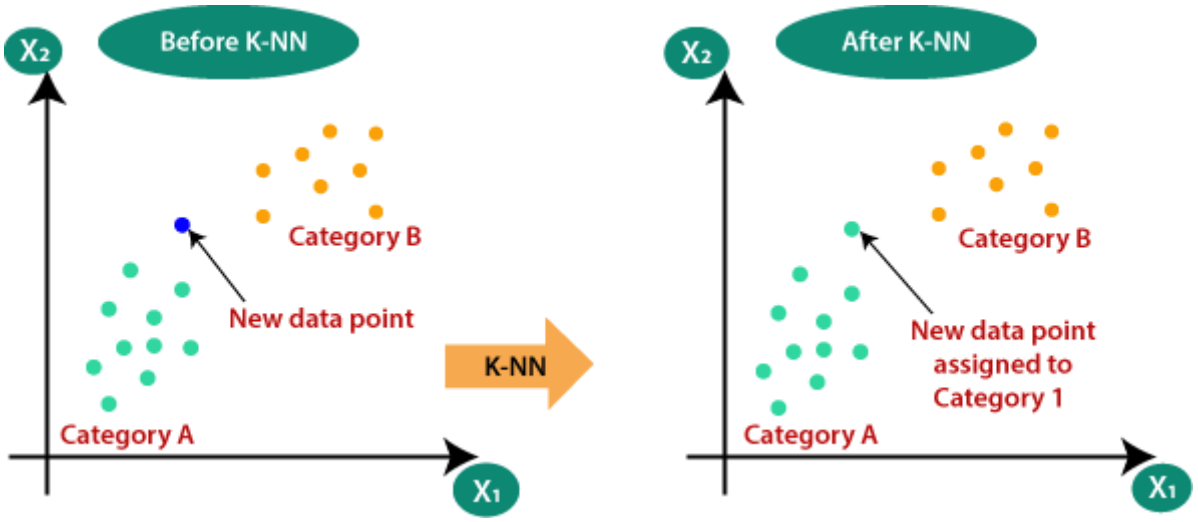

Fig 7.2.2 Before k-NN and After K-NN

How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

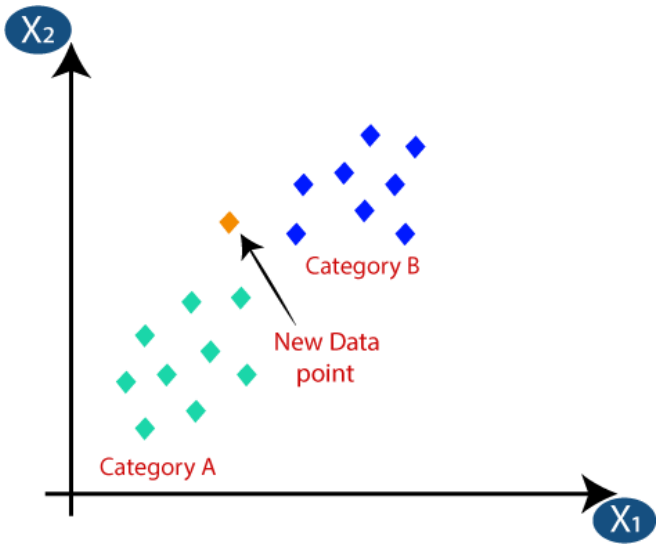

Fig 7.2.3 K-NN Required Category

- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:

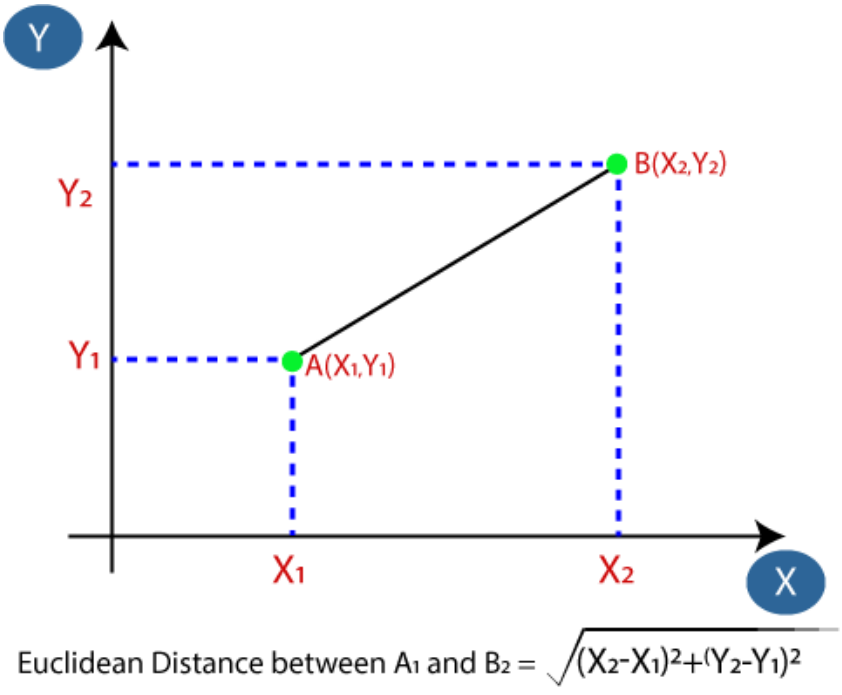

Fig 7.2.4 K-NN Euclidean Distance Between A1 and B1

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

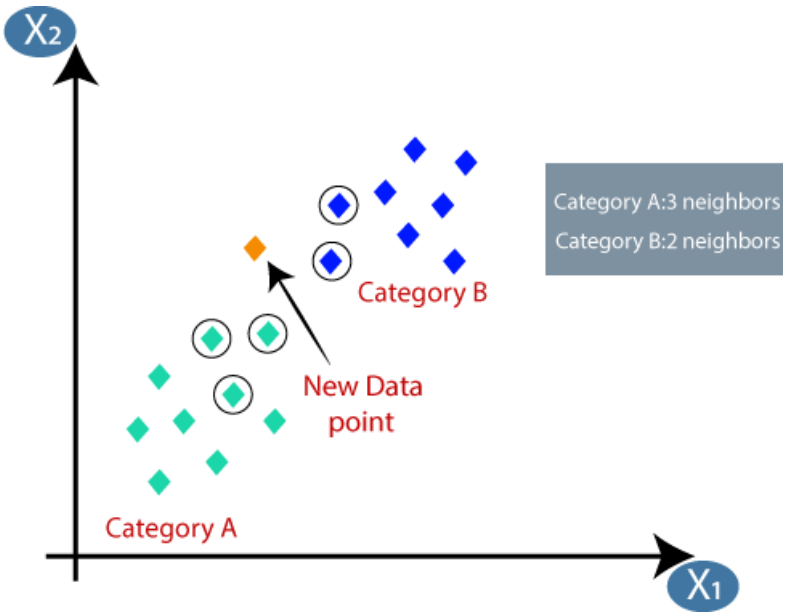

Fig 7.2.5 K-NN Nearest Neighbours

- As we can see the 3 nearest neighbours are from category A, hence this new data point must belong to category A.

7.3 Naïve Bayes

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of colour, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

7.4 Support Vector Machine

- 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. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

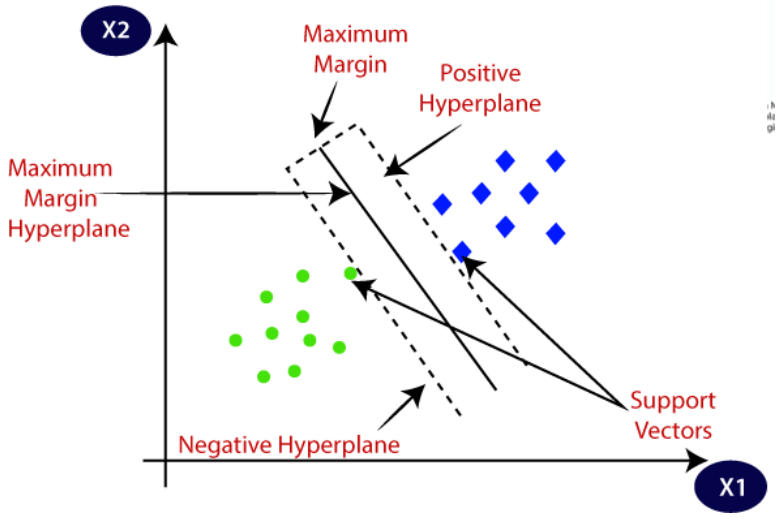

Fig 7.4.1 Support Vector Machine

Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:

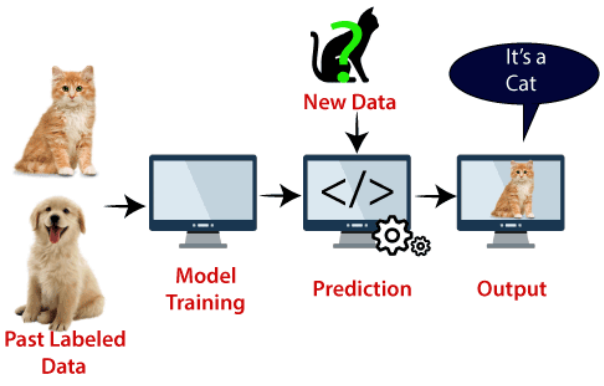

Fig 7.4.2 SVM Example

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

7.5 Decision Tree

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

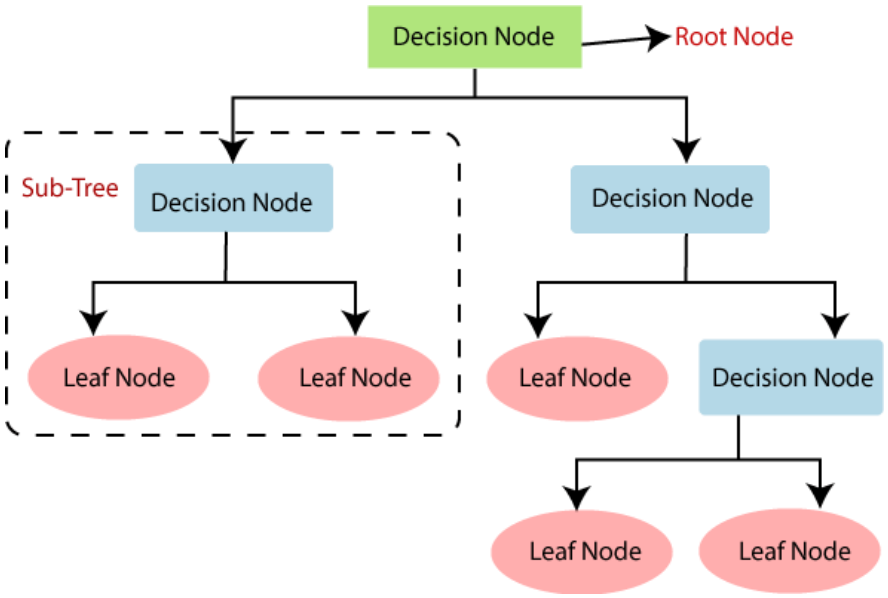

Fig 7.5.1 Structure of a Decision Tree

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART (Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:
    - **Gini Index= 1- $\sum_j P_j^2$**

7.6 Random Forest

- 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.
- 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 overfitting.**
- The below diagram explains the working of the Random Forest algorithm:

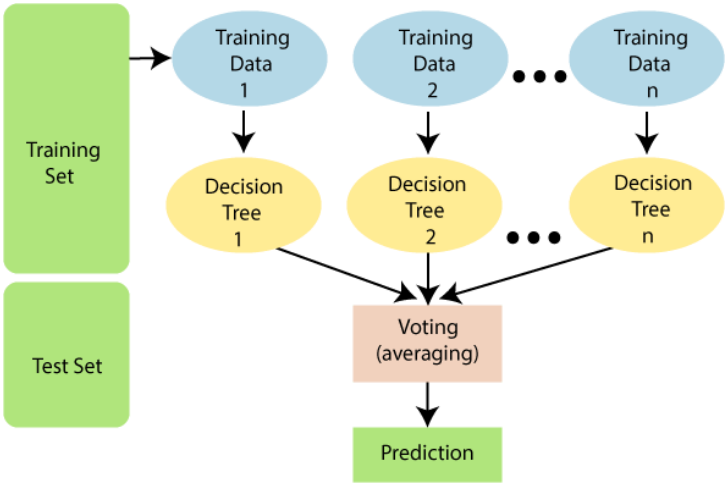

Fig 7.6.1 Working of the Random Forest Algorithm

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

- 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.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

8. PERFORMANCE METRICS

Evaluating the performance of a Machine learning model is one of the important steps while building an effective ML model. ***To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics.***

8.1 Accuracy:

The accuracy is calculated as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

Where,

- True Positive (TP) =Observation is positive and is predicted to be positive.
- False Negative (FN) = Observation is positive but is predicted negative.
- True Negative (TN) = Observation is negative and is predicted to be negative.
- False Positive (FP) =Observation is negative but is predicted positive

8.2 Recall:

Recall can be defined as High Recall indicates the class is correctly recognized

(a small number of FN). Recall is calculated as:

$$\text{Recall} = \frac{TP}{TP+FN}$$

8.3 Pression

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (a small number of FP). Precision is calculated as

$$Precision = \frac{TP}{(TP + FP)}$$

8.4 F1-Score

F-score or F1 Score is a metric to evaluate a binary classification model on the basis of predictions that are made for the positive class. It is calculated with the help of Precision and Recall. It is a type of single score that represents both Precision and Recall. So, ***the F1 Score can be calculated as the harmonic mean of both precision and Recall, assigning equal weight to each of them.***

The formula for calculating the F1 score is given below:

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$

8.5 Mean Square Error

Mean Squared error or MSE is one of the most suitable metrics for Regression evaluation. It measures the average of the Squared difference between predicted values and the actual value given by the model.

9. METHODOLOGY

9.1 Data Collection

In conducting " A Comparative Study of ML Algorithms for Predicting Diabetes," data collection is a fundamental step aimed at sourcing relevant datasets containing information crucial for training and evaluating machine learning models. The process involves gathering comprehensive datasets from various reputable sources such as healthcare institutions, medical research repositories, or publicly available datasets like the UCI Machine Learning Repository or Kaggle. These datasets typically consist of features relevant to diabetes prediction, including patient demographics, medical history, lifestyle factors, and diagnostic test results, along with corresponding labels indicating the presence or absence of diabetes's

Ethical considerations are paramount throughout the data collection process, ensuring compliance with regulations and guidelines regarding patient privacy and data protection. Obtaining necessary approvals from ethics boards or institutional review boards is essential, particularly when dealing with sensitive medical data. Additionally, measures are implemented to anonymize and de-identify patient information to safeguard confidentiality and privacy.

Once the datasets are collected, they undergo careful scrutiny and preprocessing to ensure data quality and suitability for analysis. This involves cleaning the data to address issues such as missing values, outliers, and inconsistencies, as well as performing feature engineering to extract relevant information and create informative predictors. The curated datasets are then divided into training, validation, and testing sets, facilitating the robust training and evaluation of machine learning models for diabetes prediction. Overall, meticulous data collection lays the foundation for conducting a rigorous and informative comparative study of ML algorithms for predicting diabetes, yielding insights that can inform advancements in healthcare analytics and patient care.

9.2 Data Pre-processing

Data preprocessing is a critical step in "A Comparative Study of ML Algorithms for Predicting Diabetes," where collected datasets undergo careful cleaning and transformation to ensure their suitability for analysis. This process involves several key tasks to address issues such as missing values, outliers, and inconsistencies, ultimately enhancing the quality and reliability of the data. Firstly, missing values within the dataset are identified and handled through methods like imputation, where missing values are replaced with estimated values derived from the available data. Additionally, outliers, which are data points that deviate significantly from the rest of the data, are detected and either corrected or removed to prevent them from skewing the analysis.

Furthermore, data preprocessing encompasses feature engineering, a technique aimed at creating new features or transforming existing ones to better represent the underlying relationships within the data. This may involve combining or deriving new features from existing ones, scaling or normalizing numerical features to ensure consistency in scale, and encoding categorical variables into numerical representations suitable for machine learning algorithms. By performing feature engineering, the dataset becomes more informative and conducive to training accurate predictive models for diabetes prediction.

Finally, the preprocessed dataset is split into training, validation, and testing sets to facilitate model development and evaluation. The training set is used to train machine learning algorithms, while the validation set is utilized to fine-tune model parameters and select the best-performing model. The testing set, which is kept separate from the training and validation sets, serves as an independent dataset for evaluating the final performance of the selected model. Through meticulous data preprocessing, researchers ensure that the collected data is clean, informative, and ready for analysis, laying the groundwork for a robust comparative study of ML algorithms for predicting diabetes.

9.3 Training data

In the context of "A Comparative Study of ML Algorithms for Predicting Diabetes," the training data serves as the foundational component for training machine learning algorithms to predict diabetes onset accurately. This dataset comprises a subset of the collected data containing features such as patient demographics, medical history, lifestyle factors, and diagnostic test results, along with corresponding labels indicating the presence or absence of diabetes.

Before the training process commences, the training data undergoes meticulous preprocessing steps to ensure its quality and suitability for training the models. This includes handling missing values, outliers, and inconsistencies, as well as feature engineering to extract relevant information and transform features if necessary. Additionally, the training data is typically split further into training and validation sets to facilitate model development and evaluation, respectively, ensuring the robustness and generalization of the trained models.

During the training phase, machine learning algorithms are exposed to the training data, where they learn to recognize patterns and associations between the input features and the target variable (diabetes status). Through iterative optimization algorithms, the models adjust their parameters to minimize prediction errors and improve their predictive performance. The effectiveness of the trained models is then assessed using evaluation metrics on separate testing data. Overall, the training data plays a crucial role in enabling the development of accurate and reliable predictive models for predicting diabetes in the comparative study of ML algorithms.

9.4 Testing data with 6 classification models

In "A Comparative Study of ML Algorithms for Predicting Diabetes," testing data with six classification models refers to the process of evaluating the performance of six different machine learning algorithms.

The six classification models, such as logistic regression, decision trees, random forests, support vector machines, k-nearest neighbours, and naive Bayes, are applied to this testing data to assess their predictive capabilities.

By comparing the performance metrics obtained from each model, researchers can determine which algorithm performs best for diabetes prediction in terms of accuracy, precision, recall, F1-score and mean square error.

9.5 Comparison and select the best model for diabetes prediction

In "A Comparative Study of ML Algorithms for Predicting Diabetes," the focus is on assessing the effectiveness of six classification models in accurately predicting diabetes onset. This investigation begins with the careful selection of classification algorithms known for their suitability in binary classification tasks. Models such as logistic regression, decision trees, random forests, support vector machines, k-nearest neighbors, and naive Bayes are chosen based on their widespread use and potential efficacy in healthcare analytics.

The study then progresses to data preparation, where a dataset containing relevant features for diabetes prediction is curated and preprocessed. This involves cleaning the data to handle missing values and outliers, as well as feature engineering to extract meaningful insights from the dataset. Once the data is prepared, each classification model is trained using the dataset to learn patterns and associations between the input features and diabetes status.

Following training, the performance of each model is rigorously evaluated using established metrics such as accuracy, precision, recall, and F1-score. These metrics provide quantitative measures of each model's predictive capability, allowing for a comprehensive comparison of their performance. The model that demonstrates the highest overall performance across these metrics is selected as the most suitable for diabetes prediction. Additionally, factors such as interpretability, computational efficiency, and robustness to imbalanced data are considered to ensure the practical applicability of the chosen model in real-world healthcare scenarios.

10. TECHNOLOGY OVERVIEW

In This project we are going to use python and its libraries.

Python is a widely used programming language that offers several unique features and advantages compared to languages like **Java** and **C++.**

In the late 1980s, **Guido van Rossum** dreamed of developing Python. The first version of **Python 0.9.0 was released in 1991**. Since its release, Python started gaining popularity. According to reports, Python is now the most popular programming language among developers because of its high demands in the tech realm.

What is Python

Python is a general-purpose, dynamically typed, high-level, compiled and interpreted, garbage-collected, and purely object-oriented programming language that supports procedural, object-oriented, and functional programming.

Features of Python:

- **Easy to use and Read -** Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers. This simplicity can lead to faster development and reduce the chances of errors.
- **Dynamically Typed** - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during writing codes.
- **High-level** - High-level language means human readable code.
- **Compiled and Interpreted** - Python code first gets compiled into bytecode, and then interpreted line by line. When we download the Python in our system form org we download the default implement of Python known as CPython. CPython is considered to be Complied and Interpreted both.
- **Garbage Collected** - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory.
- **Purely Object-Oriented** - It refers to everything as an object, including numbers and strings.
- **Cross-platform Compatibility** - Python can be easily installed on Windows, macOS, and various Linux distributions, allowing developers to create software that runs across different operating systems.

- **Rich Standard Library** - Python comes with several standard libraries that provide ready-to-use modules and functions for various tasks, ranging from **web development** and **data manipulation** to **machine learning** and **networking**.
- **Open Source** - Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

History of Python

Python was created by Guido van Rossum. In the late 1980s, Guido van Rossum, a Dutch programmer, began working on Python while at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. He wanted to create a successor to the **ABC programming language** that would be easy to read and efficient.

In February 1991, the first public version of Python, version 0.9.0, was released. This marked the official birth of **Python as an open-source project**. The language was named after the British comedy series "**Monty Python's Flying Circus**".

Python development has gone through several stages. **In January 1994, Python 1.0 was released as a usable and stable programming language.** This version included many of the features that are still present in Python today.

From the 1990s to the 2000s, Python gained popularity for its simplicity, readability, and versatility. **In October 2000, Python 2.0 was released**. Python 2.0 introduced list comprehensions, garbage collection, and support for Unicode.

In December 2008, Python 3.0 was released. Python 3.0 introduced several backward-incompatible changes to improve code readability and maintainability.

Throughout 2010s, Python's popularity increased, particularly in fields like data science, machine learning, and web development. Its rich ecosystem of libraries and frameworks made it a favourite among developers.

The **Python Software Foundation (PSF) was established in 2001** to promote, protect, and advance the Python programming language and its community.

Where is Python used?

Python is a general-purpose, popular programming language, and it is used in almost every technical field. The various areas of Python use are given below.

- **Data Science:** Data Science is a vast field, and Python is an important language for this field because of its simplicity, ease of use, and availability of powerful data analysis and visualization libraries like NumPy, Pandas, and Matplotlib.
- **Desktop Applications:** PyQt and Tkinter are useful libraries that can be used in GUI - Graphical User Interface-based Desktop Applications. There are better languages for this field, but it can be used with other languages for making Applications.
- **Console-based Applications:** Python is also commonly used to create command-line or console-based applications because of its ease of use and support for advanced features such as input/output redirection and piping.
- **Mobile Applications:** While Python is not commonly used for creating mobile applications, it can still be combined with frameworks like Kivy or BeeWare to create cross-platform mobile applications.
- **Software Development:** Python is considered one of the best software-making languages. Python is easily compatible with both from Small Scale to Large Scale software.
- **Artificial Intelligence**: AI is an emerging Technology, and Python is a perfect language for artificial intelligence and machine learning because of the availability of powerful libraries such as TensorFlow, Keras, and PyTorch.
- **Web Applications:** Python is commonly used in web development on the backend with frameworks like Django and Flask and on the front end with tools like JavaScript HTML and CSS.
- **Enterprise Applications:** Python can be used to develop large-scale enterprise applications with features such as distributed computing, networking, and parallel processing.
- **3D CAD Applications:** Python can be used for 3D computer-aided design (CAD) applications through libraries such as Blender.
- **Machine Learning:** Python is widely used for machine learning due to its simplicity, ease of use, and availability of powerful machine learning libraries.
- **Computer Vision or Image Processing Applications:** Python can be used for computer vision and image processing applications through powerful libraries such as OpenCV and Scikit-image.

- **Speech Recognition:** Python can be used for speech recognition applications through libraries such as SpeechRecognition and PyAudio.
- **Scientific computing:** Libraries like NumPy, SciPy, and Pandas provide advanced numerical computing capabilities for tasks like data analysis, machine learning, and more.
- **Education:** Python's easy-to-learn syntax and availability of many resources make it an ideal language for teaching programming to beginners.
- **Testing:** Python is used for writing automated tests, providing frameworks like unit tests and pytest that help write test cases and generate reports.
- **Gaming:** Python has libraries like Pygame, which provide a platform for developing games using Python.
- **IoT:** Python is used in IoT for developing scripts and applications for devices like Raspberry Pi, Arduino, and others.
- **Networking:** Python is used in networking for developing scripts and applications for network automation, monitoring, and management.
- **DevOps**: Python is widely used in DevOps for automation and scripting of infrastructure management, configuration management, and deployment processes.
- **Finance:** Python has libraries like Pandas, Scikit-learn, and Statsmodels for financial modeling and analysis.
- **Audio and Music:** Python has libraries like Pyaudio, which is used for audio processing, synthesis, and analysis, and Music21, which is used for music analysis and generation.
- **Writing scripts:** Python is used for writing utility scripts to automate tasks like file operations, web scraping, and data processing.

Python Popular Frameworks and Libraries

Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

- **Web development (Server-side) - Django Flask**, **Pyramid**, **CherryPy**
- **GUIs based applications - Tkinter**, **PyGTK**, **PyQt**, **PyJs**, etc.
- **Machine Learning - TensorFlow**, **PyTorch**, **Scikit-learn**, **Matplotlib**, **Scipy**, etc.
- **Mathematics - NumPy**, **Pandas**, etc.
- **BeautifulSoup:** a library for web scraping and parsing HTML and XML
- **Requests**: a library for making HTTP requests

- **SQLAlchemy**: a library for working with SQL databases
- **Kivy:** a framework for building multi-touch applications
- **Pygame:** a library for game development
- **Pytest:** a testing framework for Python Django
- **REST framework**: a toolkit for building RESTful APIs
- **FastAPI**: a modern, fast web framework for building APIs
- **Streamlit**: a library for building interactive web apps for machine learning and data science
- **NLTK**: a library for natural language processing

In this project we are going to use below libraries :

- StandardScaler
- Pandas
- Numpy
- Seaborn
- Matplotlib
- Train_test_Split
- LogisticRegression
- KNeighborsClassifier
- GaussianNB
- DecisionTreeClassifier
- RandomForestClassifier
- SVC
- Accuracy_Score
- Recall_Score
- Precision_Score
- F1_Score
- Mean_Squared_Error

Here's a brief explanation of each of the mentioned libraries and terms:

1. StandardScaler: This is a preprocessing technique in machine learning used to standardize the features by removing the mean and scaling to unit variance. It's commonly applied to ensure that data has a mean of 0 and a standard deviation of 1.

2. Pandas: Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures like DataFrame for handling structured data and tools for cleaning, transforming, and analyzing data.

3. Numpy: NumPy is a fundamental package for scientific computing in Python. It provides support for multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

4. Seaborn: Seaborn is a statistical data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is often used for creating visualizations such as heatmaps, scatter plots, and bar plots.

5. Matplotlib : Matplotlib is a widely-used plotting library in Python. It provides a MATLAB-like interface for creating static, interactive, and animated visualizations. Matplotlib can be used to generate various types of plots, including line plots, scatter plots, histograms, and more.

6. Train_test_Split: Train-test split is a technique used to split a dataset into two subsets: one for training a machine learning model and the other for testing its performance. This helps in evaluating the model's generalization to unseen data.

7. LogisticRegression : Logistic regression is a classification algorithm used for binary classification tasks. It models the probability that a given input belongs to a particular class using the logistic function.

8. KNeighborsClassifier : K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm used for classification and regression tasks. It classifies a data point based on the majority class of its k nearest neighbors in the feature space.

9. GaussianNB : Gaussian Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem with the assumption of independence between features. It's particularly useful for text classification and other tasks where the features are assumed to be Gaussian distributed.

10. DecisionTreeClassifier : Decision Trees are a popular machine learning algorithm used for classification and regression tasks. They partition the feature space into regions and make predictions by traversing the tree from root to leaf based on the feature values.

11. RandomForestClassifier : Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees. It's known for its robustness and accuracy.

12. SVC : Support Vector Classifier (SVC) is a supervised learning algorithm used for classification tasks. It constructs a hyperplane or set of hyperplanes in a high-dimensional space to separate classes with the maximum margin.

13. Accuracy_Score, Recall_Score, Precision_Score, F1_Score : These are evaluation metrics commonly used for assessing the performance of classification models. Accuracy measures the proportion of correctly classified instances, recall measures the proportion of actual positive cases that were correctly identified, precision measures the proportion of positive cases that were correctly classified, and F1 Score is the harmonic mean of precision and recall, providing a balance between the two.

14. Mean_Squared_Error : Mean Squared Error (MSE) is a common metric used to evaluate regression models. It measures the average of the squares of the errors or deviations, which are the differences between actual and predicted values.

11. Code

```
#Importing Dependencies
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
"""Data Collection and Analysis PIMA Diabetes Dataset"""
#loading the dataset to a pandas df
df = pd.read_csv(r'C:\Users\USER\Desktop\jupyter projects\diabetess.csv')
#printing the first 5 rows
print(df.head())
#no of rows and cols
print(df.shape)
#getting the statistical measures of the df
print(df.describe())
#no of diabetics and non-diabetics
df['Outcome'].value_counts()
"""0 --> Non-Diabetic1 --> Diabetic"""
"""Data CleaningDrop duplicates"""
print('Before dropping duplicates: ', df.shape)
df = df.drop_duplicates()
print('After dropping duplicates: ', df.shape)
"""Check for NULL values"""
df.isnull().sum()
"""Check for missing values"""
print('No of missing values in Glucose: ', df[df['Glucose'] == 0].shape[0])
```

```
print('No of missing values in BloodPressure: ', df[df['BloodPressure'] == 0].shape[0])

print('No of missing values in SkinThickness: ', df[df['SkinThickness'] == 0].shape[0])

print('No of missing values in Insulin: ', df[df['Insulin'] == 0].shape[0])

print('No of missing values in BMI: ', df[df['BMI'] == 0].shape[0])

"""Replace missing values with mean"""

df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())

df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].mean())

df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].mean())

df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].mean())

df['BMI'] = df['BMI'].replace(0, df['BMI'].mean())

df.describe()

import matplotlib.pyplot as plt

import seaborn as sns

# Define a color palette

colors = sns.color_palette('viridis')

# Create a subplot with 1 row and 2 columns

f, ax = plt.subplots(1, 2, figsize=(10, 5))

# Plot a pie chart for the distribution of 'Outcome' values

df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0],
shadow=True, colors=colors)

ax[0].set_title('Outcome')

ax[0].set_ylabel('')

# Plot a countplot for the 'Outcome' values using Seaborn

sns.countplot(x='Outcome', data=df, ax=ax[1], palette=colors) # Specify 'x' parameter for the
column

ax[1].set_title('Outcome')

# Get and print the counts of negative (0) and positive (1) outcomes
```

```
N, P = df['Outcome'].value_counts()

print('Negative(0) ->', N)

print('Positive(1) ->', P)

# Add grid to the plots

plt.grid()

# Display the plots

plt.show()
```

```
   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
(2600, 9)
       Pregnancies      Glucose  BloodPressure  SkinThickness      Insulin  \
count  2600.000000  2600.000000    2600.000000    2600.000000  2600.000000
mean      3.870769   120.917692      69.099615      20.506538    79.803462
std       3.368058    31.985426      19.302668      15.946089   116.651787
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    24.000000
75%       6.000000   141.000000      80.000000      32.000000   127.000000
max      17.000000   199.000000     122.000000      99.000000   846.000000

               BMI  DiabetesPedigreeFunction          Age      Outcome
count  2600.000000               2600.000000  2600.000000  2600.000000
mean     32.018115                  0.473213    33.227308     0.352308
std       7.930697                  0.332333    11.705549     0.477781
min       0.000000                  0.078000    21.000000     0.000000
25%      27.300000                  0.245000    24.000000     0.000000
50%      32.000000                  0.374500    29.000000     0.000000
75%      36.600000                  0.629250    41.000000     1.000000
max      67.100000                  2.420000    81.000000     1.000000
```

```
Before dropping duplicates:  (2600, 9)
After dropping duplicates:  (768, 9)
No of missing values in Glucose:  5
No of missing values in BloodPressure:  35
No of missing values in SkinThickness:  227
No of missing values in Insulin:  374
No of missing values in BMI:  11
Negative(0) -> 500
Positive(1) -> 268
```

```
"""Dataset is balanced Histogram (data is balanced or skewed) attribute diagrams"""
print("\n Dataset is not balanced Histogram (data is balanced or skewed) attribute diagrams\n")
df.hist(bins=10,figsize=(10,10))
plt.show()
"""Analysing relationships bw variables Correlation analysis"""
#get correlations of each feature in the dataset
print("\nAnalysing relationships bw variables - Correlation analysis\n")
corr_mat = df.corr()
top_corr_features = corr_mat.index
plt.figure(figsize=(10,10))
#plot heat map
#g = sns.heatmap(df[top_corr_features].corr(), annot=True, cmap='RdYlGn')
#separating the independent and dependent variables
print("\nseparating the independent and dependent variables\n")
X = df.drop(columns='Outcome', axis=1)
y = df['Outcome']
print(X.head())
print(y.head())
"""Data Standardisation - Feature Scaling"""
print("\nData Standardisation - Feature Scaling\n")
scaler = StandardScaler()
scaler.fit(X)
standardised_data = scaler.transform(X)
print(standardised_data)
X = standardised_data
y = df.Outcome
print(X)
```

print(y)

Dataset is not balanced Histogram (data is balanced or skewed) attribute diagrams

```
Analysing relationships bw variables - Correlation analysis


separating the independent and dependent variables

   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0            6    148.0           72.0      35.000000   79.799479  33.6
1            1     85.0           66.0      29.000000   79.799479  26.6
2            8    183.0           64.0      20.536458   79.799479  23.3
3            1     89.0           66.0      23.000000   94.000000  28.1
4            0    137.0           40.0      35.000000  168.000000  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

```
Data Standardisation - Feature Scaling

[[ 0.63078205  0.86817025 -0.02137212 ...  0.16341116  0.47979096
   1.42083472]
 [-0.85373806 -1.20266618 -0.51265276 ... -0.85843726 -0.36114056
  -0.19442499]
 [ 1.22459009  2.01863494 -0.67641298 ... -1.3401658   0.61689936
  -0.10941132]
 ...
 [-0.85373806  0.21076186 -1.9864947  ...  1.17066174  0.43713501
  -0.78952067]
 [-1.15064208  1.29548571 -1.82273449 ... -1.54453548 -0.65668533
   2.69603975]
 [ 0.63078205  0.96678151 -0.8401732  ...  0.44077001  0.67783642
  -0.44946599]]
[[ 0.63078205  0.86817025 -0.02137212 ...  0.16341116  0.47979096
   1.42083472]
 [-0.85373806 -1.20266618 -0.51265276 ... -0.85843726 -0.36114056
  -0.19442499]
 [ 1.22459009  2.01863494 -0.67641298 ... -1.3401658   0.61689936
  -0.10941132]
 ...
 [-0.85373806  0.21076186 -1.9864947  ...  1.17066174  0.43713501
  -0.78952067]
 [-1.15064208  1.29548571 -1.82273449 ... -1.54453548 -0.65668533
   2.69603975]
 [ 0.63078205  0.96678151 -0.8401732  ...  0.44077001  0.67783642
  -0.44946599]]
0       1
1       0
2       1
3       0
4       1
       ..
2131    1
2132    1
2133    1
2134    0
2135    0
Name: Outcome, Length: 2136, dtype: int64
```

```
"""Split data into training and testing data"""

print("\nSplit data into training and testing data\n")

# 80% is train, 20% is test

# random state is used to ensure a specific split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)

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

```
Split data into training and testing data

(2136, 8) (1708, 8) (428, 8)
```

```
"""Classification Models1) Logistic Regression"""

from sklearn.linear_model import LogisticRegression

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

lr_model.fit(X_train, y_train)

"""2) K Neighbours Classifier"""

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier()

knn_model.fit(X_train, y_train)

"""3) Naive Bayes Classifier"""

from sklearn.naive_bayes import GaussianNB

nb_model = GaussianNB()

nb_model.fit(X_train, y_train)

"""4) Support Vector Machine(SVM)"""

from sklearn.svm import SVC

svm_model = SVC()

svm_model.fit(X_train, y_train)

"""5) Decision tree"""

from sklearn.tree import DecisionTreeClassifier
```

```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train,  y_train)
"""6) Random Forest"""
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(criterion='entropy')
rf_model.fit(X_train, y_train)
"""Predicting & Evaluating the Models"""
from sklearn.metrics import recall_score, precision_score, f1_score
# Make predictions using test data for all 6 models
lr_preds = lr_model.predict(X_test)
knn_preds = knn_model.predict(X_test)
nb_preds = nb_model.predict(X_test)
svm_preds = svm_model.predict(X_test)
dt_preds = dt_model.predict(X_test)
rf_preds = rf_model.predict(X_test)
 # common
from sklearn.metrics import accuracy_score
# Calculate accuracy scores for each model
lr_accuracy = accuracy_score(y_test, lr_preds)
knn_accuracy = accuracy_score(y_test, knn_preds)
nb_accuracy = accuracy_score(y_test, nb_preds)
svm_accuracy = accuracy_score(y_test, svm_preds)
dt_accuracy = accuracy_score(y_test, dt_preds)
rf_accuracy = accuracy_score(y_test, rf_preds)
# Convert accuracy scores to percentages
lr_accuracy_percentage = lr_accuracy * 100
knn_accuracy_percentage = knn_accuracy * 100
```

```
nb_accuracy_percentage = nb_accuracy * 100

svm_accuracy_percentage = svm_accuracy * 100

dt_accuracy_percentage = dt_accuracy * 100

rf_accuracy_percentage = rf_accuracy * 100

# Print accuracy scores as percentages

print("Logistic Regression Accuracy:", lr_accuracy_percentage, "%")

print("K Neighbours Classifier Accuracy:", knn_accuracy_percentage, "%")

print("Naive Bayes Classifier Accuracy:", nb_accuracy_percentage, "%")

print("Support Vector Machine Accuracy:", svm_accuracy_percentage, "%")

print("Decision Tree Accuracy:", dt_accuracy_percentage, "%")

print("Random Forest Accuracy:", rf_accuracy_percentage, "%")

import matplotlib.pyplot as plt

import seaborn as sns

# Define model names

models = ['Logistic Regression', 'K Neighbours Classifier', 'Naive Bayes Classifier', 'Support Vector Machine', 'Decision Tree', 'Random Forest']

# Define accuracy scores as percentages

accuracies_percentage = [lr_accuracy_percentage, knn_accuracy_percentage,
nb_accuracy_percentage, svm_accuracy_percentage, dt_accuracy_percentage,
rf_accuracy_percentage]

# Plot accuracy scores as percentages

plt.figure(figsize=(10, 6))

sns.barplot(x=models, y=accuracies_percentage, palette='viridis')

plt.title('Accuracy Scores of Classification Models')

plt.xlabel('Model')

plt.ylabel('Accuracy Score (%)')

plt.xticks(rotation=45)

plt.ylim(0, 100)  # Set the y-axis limit from 0 to 100
```

plt.show()

```
Logistic Regression Accuracy: 77.42690058479532 %
K Neighbours Classifier Accuracy: 96.95906432748538 %
Naive Bayes Classifier Accuracy: 75.20467836257309 %
Support Vector Machine Accuracy: 86.54970760233918 %
Decision Tree Accuracy: 100.0 %
Random Forest Accuracy: 100.0 %
```

 # Calculate recall scores for all models

lr_recall = recall_score(y_test, lr_preds)

knn_recall = recall_score(y_test, knn_preds)

nb_recall = recall_score(y_test, nb_preds)

svm_recall = recall_score(y_test, svm_preds)

dt_recall = recall_score(y_test, dt_preds)

rf_recall = recall_score(y_test, rf_preds)

 # Print recall scores

print("Recall Scores:")

print("Logistic Regression:", lr_recall)

print("K Nearest Neighbors:", knn_recall)

print("Naive Bayes:", nb_recall)

print("Support Vector Machine:", svm_recall)

print("Decision Tree:", dt_recall)

print("Random Forest:", rf_recall)

 # Plotting recall, precision, and F1 scores

models = ['LR', 'KNN', 'NB', 'SVM', 'DT', 'RF']

recall_scores = [lr_recall, knn_recall, nb_recall, svm_recall, dt_recall, rf_recall]

plt.figure(figsize=(18, 6))

 # Plot recall scores

plt.subplot(1, 3, 1)

sns.barplot(x=models, y=recall_scores, palette='viridis')

```
plt.title('Recall Scores of Classification Models')
plt.xlabel('Classification Models')
plt.ylabel('Recall Score')
plt.ylim(0, 1) # Recall scores range from 0 to 1
plt.tight_layout()
plt.show()
```

```
Recall Scores:
Logistic Regression: 0.5524475524475524
K Nearest Neighbors: 1.0
Naive Bayes: 0.6013986013986014
Support Vector Machine: 0.7645687645687645
Decision Tree: 1.0
Random Forest: 1.0
```

```
# Calculate precision scores for all models
lr_precision = precision_score(y_test, lr_preds)
knn_precision = precision_score(y_test, knn_preds)
nb_precision = precision_score(y_test, nb_preds)
svm_precision = precision_score(y_test, svm_preds)
dt_precision = precision_score(y_test, dt_preds)
rf_precision = precision_score(y_test, rf_preds)
# Print precision scores
print("\nPrecision Scores:")
print("Logistic Regression:", lr_precision)
print("K Nearest Neighbors:", knn_precision)
print("Naive Bayes:", nb_precision)
print("Support Vector Machine:", svm_precision)
print("Decision Tree:", dt_precision)
print("Random Forest:", rf_precision)
plt.figure(figsize=(18, 6))
```

 # Plotting recall, precision, and F1 scores

models = ['LR', 'KNN', 'NB', 'SVM', 'DT', 'RF']

precision_scores = [lr_precision, knn_precision, nb_precision, svm_precision, dt_precision, rf_precision]

 # Plot precision scores

plt.subplot(1, 3, 2)

sns.barplot(x=models, y=precision_scores, palette='viridis')

plt.title('Precision Scores of Classification Models')

plt.xlabel('Classification Models')

plt.ylabel('Precision Score')

plt.ylim(0, 1) # Precision scores range from 0 to 1

plt.tight_layout()

plt.show()

```
Precision Scores:
Logistic Regression: 0.7117117117117117
K Nearest Neighbors: 1.0
Naive Bayes: 0.6386138613861386
Support Vector Machine: 0.8098765432098766
Decision Tree: 1.0
Random Forest: 1.0
```

 # Calculate F1 scores for all models

lr_f1 = f1_score(y_test, lr_preds)

knn_f1 = f1_score(y_test, knn_preds)

nb_f1 = f1_score(y_test, nb_preds)

svm_f1 = f1_score(y_test, svm_preds)

dt_f1 = f1_score(y_test, dt_preds)

rf_f1 = f1_score(y_test, rf_preds)

 # Print F1 scores

print("\nF1 Scores:")

```
print("Logistic Regression:", lr_f1)

print("K Nearest Neighbors:", knn_f1)

print("Naive Bayes:", nb_f1)

print("Support Vector Machine:", svm_f1)

print("Decision Tree:", dt_f1)

print("Random Forest:", rf_f1)

plt.figure(figsize=(18, 6))
```

 # Plotting recall, precision, and F1 scores

```
models = ['LR', 'KNN', 'NB', 'SVM', 'DT', 'RF']

f1_scores = [lr_f1, knn_f1, nb_f1, svm_f1, dt_f1, rf_f1]
```

 # Plot F1 scores

```
plt.subplot(1, 3, 3)

sns.barplot(x=models, y=f1_scores, palette='viridis')

plt.title('F1 Scores of Classification Models')

plt.xlabel('Classification Models')

plt.ylabel('F1 Score')

plt.ylim(0, 1) # F1 scores range from 0 to 1

plt.tight_layout()

plt.show()
```

```
F1 Scores:
Logistic Regression: 0.6220472440944882
K Nearest Neighbors: 1.0
Naive Bayes: 0.6194477791116446
Support Vector Machine: 0.7865707434052758
Decision Tree: 1.0
Random Forest: 1.0
```

```
from sklearn.metrics import mean_squared_error

import numpy as np

import matplotlib.pyplot as plt
```

```
# Define a function to evaluate the models using Mean Squared Error
def evaluate_model_with_mse(model, X_test, y_test, model_name):
    # Make predictions
    preds = model.predict(X_test)
    # Calculate Mean Squared Error
    mse = mean_squared_error(y_test, preds)
    return mse
# Create lists to store model names and their respective MSE scores
model_names = ['LR', 'KNN', 'NB', 'SVM', 'DT', 'RF']
models = [lr_model, knn_model, nb_model, svm_model, dt_model, rf_model]
mse_scores = []
# Evaluate and store MSE for each model
for model, model_name in zip(models, model_names):
    mse = evaluate_model_with_mse(model, X_test, y_test, model_name)
 mse_scores.append(mse)
# Find the index with the highest score
max_score_index = np.argmax(mse_scores)
# Highlight the highest score for Decision Trees
colors = ['lightcoral' if i == max_score_index else 'skyblue' for i in range(len(model_names))]
# Plot the MSE scores
plt.figure(figsize=(10, 6))
plt.bar(model_names, mse_scores, color=colors)
plt.xlabel('Models')
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error for Classification Models')
plt.xticks(rotation=45, ha='right')
plt.show()
```

 # Print the MSE scores

for model_name, mse_score in zip(model_names, mse_scores):

    print(f"{model_name}: MSE = {mse_score}")

```
LR: MSE = 0.22573099415204678
KNN: MSE = 0.0304093567251462
NB: MSE = 0.247953216374269
SVM: MSE = 0.13450292397660818
DT: MSE = 0.0
RF: MSE = 0.0
```

 # Plotting recall, precision, and F1 scores

models = ['Logistic Regression', 'K Nearest Neighbors', 'Naive Bayes', 'Support Vector Machine', 'Decision Tree', 'Random Forest']

recall_scores = [lr_recall, knn_recall, nb_recall, svm_recall, dt_recall, rf_recall]

precision_scores = [lr_precision, knn_precision, nb_precision, svm_precision, dt_precision, rf_precision]

f1_scores = [lr_f1, knn_f1, nb_f1, svm_f1, dt_f1, rf_f1]

from sklearn.metrics import mean_squared_error

import numpy as np

 # Define a function to evaluate the models using Mean Squared Error

def evaluate_model_with_mse(model, X_test, y_test, model_name):

    # Make predictions

    preds = model.predict(X_test)

    # Calculate Mean Squared Error

    mse = mean_squared_error(y_test, preds)

    return mse

 # Create lists to store model names and their respective MSE scores

model_names = ["Logistic Regression", "K Nearest Neighbors", "Naive Bayes", "Support Vector Machine", "Decision Tree", "Random Forest"]

mse_scores = []

```
# Evaluate and store MSE for each model
for model in [lr_model, knn_model, nb_model, svm_model, dt_model, rf_model]:
    mse = evaluate_model_with_mse(model, X_test, y_test, model_names[len(mse_scores)])
    mse_scores.append(mse)
# Highlight Decision Tree model score with a higher value
highlight_index = model_names.index("Decision Tree")
mse_scores_highlighted = mse_scores.copy()
mse_scores_highlighted[highlight_index] = max(mse_scores) + 1
from sklearn.metrics import accuracy_score
# Assuming you have already trained your models and made predictions
# Calculate accuracy scores for each model
lr_accuracy = accuracy_score(y_test, lr_preds)
knn_accuracy = accuracy_score(y_test, knn_preds)
nb_accuracy = accuracy_score(y_test, nb_preds)
svm_accuracy = accuracy_score(y_test, svm_preds)
dt_accuracy = accuracy_score(y_test, dt_preds)
rf_accuracy = accuracy_score(y_test, rf_preds)
# Convert accuracy scores to percentages
lr_accuracy_percentage = lr_accuracy * 100
knn_accuracy_percentage = knn_accuracy * 100
nb_accuracy_percentage = nb_accuracy * 100
svm_accuracy_percentage = svm_accuracy * 100
dt_accuracy_percentage = dt_accuracy * 100
rf_accuracy_percentage = rf_accuracy * 100
import matplotlib.pyplot as plt
import seaborn as sns
# Define model names
```

```
models = ['LR', 'KNN', 'NB', 'SVM', 'DT', 'RF']

# Define accuracy scores as percentages

accuracies_percentage = [lr_accuracy_percentage, knn_accuracy_percentage,
nb_accuracy_percentage, svm_accuracy_percentage, dt_accuracy_percentage,
rf_accuracy_percentage]

# Define performance metric scores for each model

recall_scores = [lr_recall, knn_recall, nb_recall, svm_recall, dt_recall, rf_recall]

precision_scores = [lr_precision, knn_precision, nb_precision, svm_precision, dt_precision,
rf_precision]

f1_scores = [lr_f1, knn_f1, nb_f1, svm_f1, dt_f1, rf_f1]

mse_scores = mse_scores

accuracy_scores = [lr_accuracy_percentage, knn_accuracy_percentage, nb_accuracy_percentage,
svm_accuracy_percentage, dt_accuracy_percentage, rf_accuracy_percentage]

# Calculate aggregate scores for each model

aggregate_scores = []

for i in range(len(models)):

    aggregate_score = (recall_scores[i] + precision_scores[i] + f1_scores[i] + (1 / (mse_scores[i] +
1)) + accuracy_scores[i]) / 5

    aggregate_scores.append(aggregate_score)

# Plot aggregate scores

plt.figure(figsize=(10, 6))

sns.barplot(x=models, y=aggregate_scores, palette='viridis')

plt.title('Aggregate Scores of Classification Models')

plt.xlabel('Model')

plt.ylabel('Aggregate Score')

plt.xticks(rotation=45)

plt.show()

# Print aggregate scores for comparison
```

```
for model_name, score in zip(models, aggregate_scores):
    print(f"{model_name}: Aggregate Score = {score}")
# Find the model with the maximum aggregate score
best_model_index = aggregate_scores.index(max(aggregate_scores))
best_model_name = models[best_model_index]
best_aggregate_score = aggregate_scores[best_model_index]
# Print the statement indicating the best aggregate score
print(f"\nThe best model for diabetes prediction based on aggregate score \n is '{best_model_name}' with an aggregate score of {best_aggregate_score:.2f}.")
```

```
LR: Aggregate Score = 15.962329964735224
KNN: Aggregate Score = 16.36691103858373
NB: Aggregate Score = 15.49434389606372
SVM: Aggregate Score = 17.051557969300273
DT: Aggregate Score = 20.701406170015197
RF: Aggregate Score = 20.60284065190607

The best model for diabetes prediction based on aggregate score
 is 'DT' with an aggregate score of 20.70.
```

12. OUTPUT SCREENS

12.1 Get and print the counts of negative (0) and positive (1) outcomes

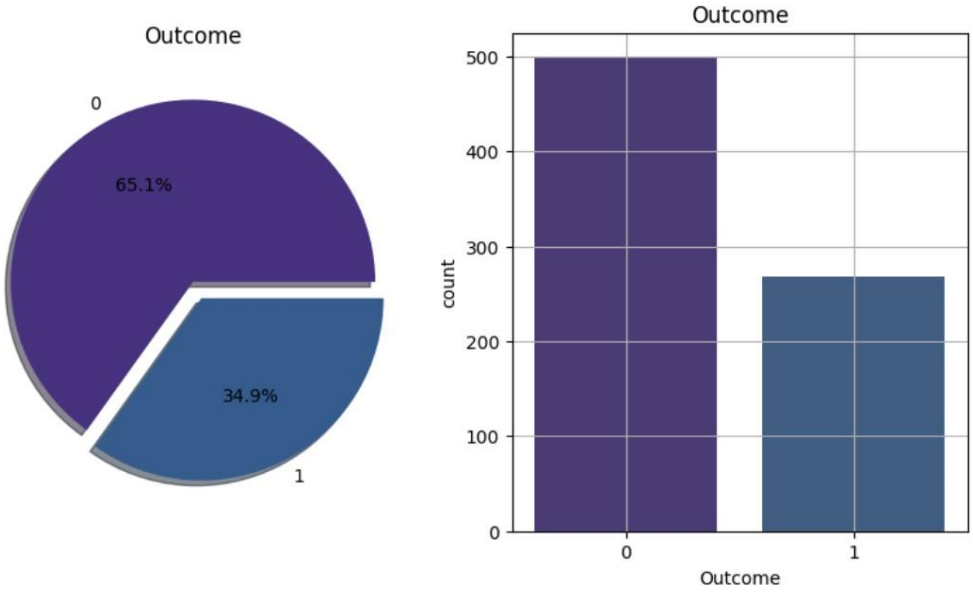

Fig 11.1 Counts of Negative and Positive outcomes

12.2 Dataset is balanced Histogram (data is balanced or skewed) attribute diagrams

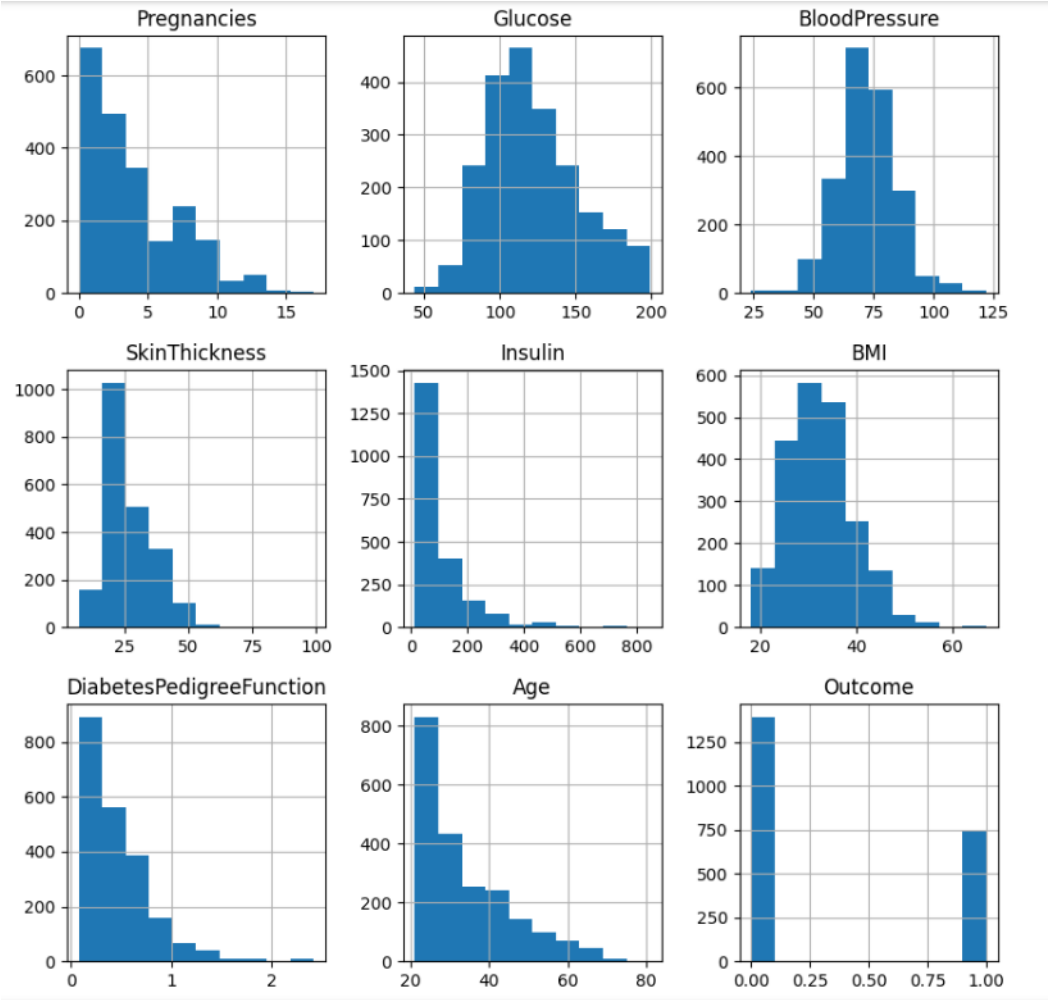

Fig 11.2 Balanced Histogram attribute diagram

12.3 Accuracy score and plotting of 6 classification algorithms:

```
Logistic Regression Accuracy: 77.42690058479532 %
K Neighbours Classifier Accuracy: 96.95906432748538 %
Naive Bayes Classifier Accuracy: 75.20467836257309 %
Support Vector Machine Accuracy: 86.54970760233918 %
Decision Tree Accuracy: 100.0 %
Random Forest Accuracy: 100.0 %
```

Fig 11.3 Accuracy Score

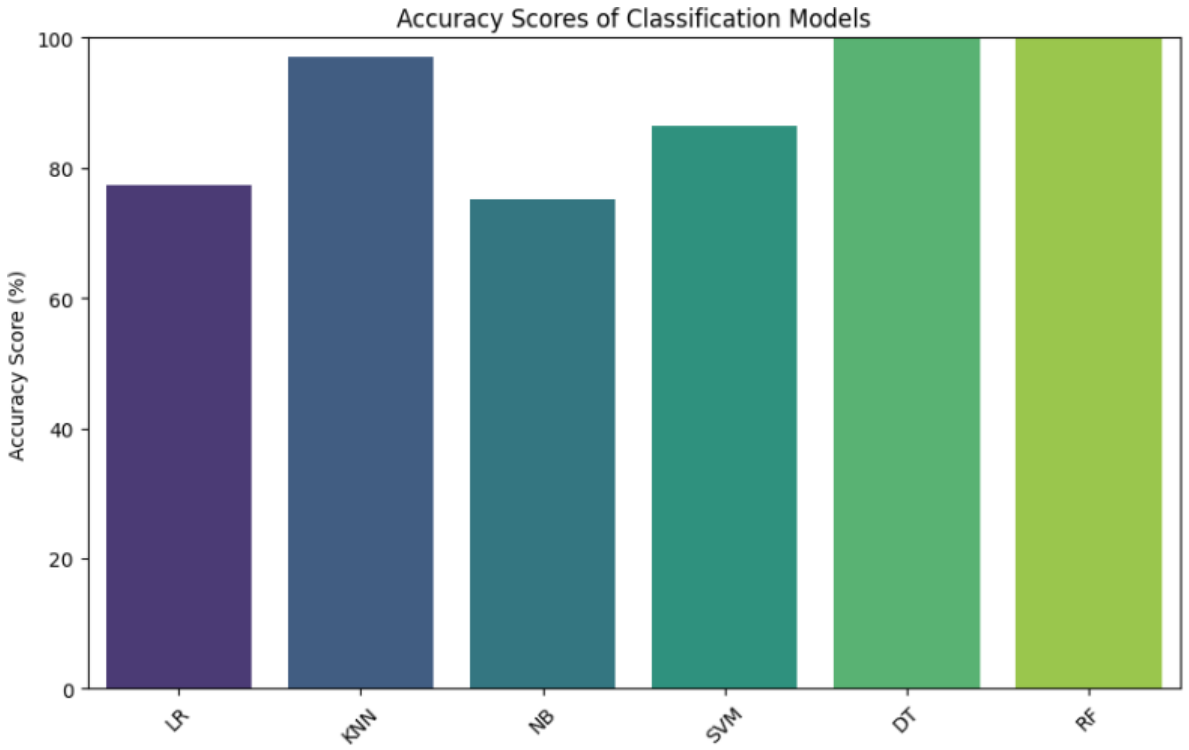

Fig 11.4 Accuracy Score Plot

12.4 Recall Score and plotting of 6 classification algorithms:

```
Recall Scores:
Logistic Regression: 0.5524475524475524
K Nearest Neighbors: 1.0
Naive Bayes: 0.6013986013986014
Support Vector Machine: 0.7645687645687645
Decision Tree: 1.0
Random Forest: 1.0
```

Fig 11.5 Recall Score

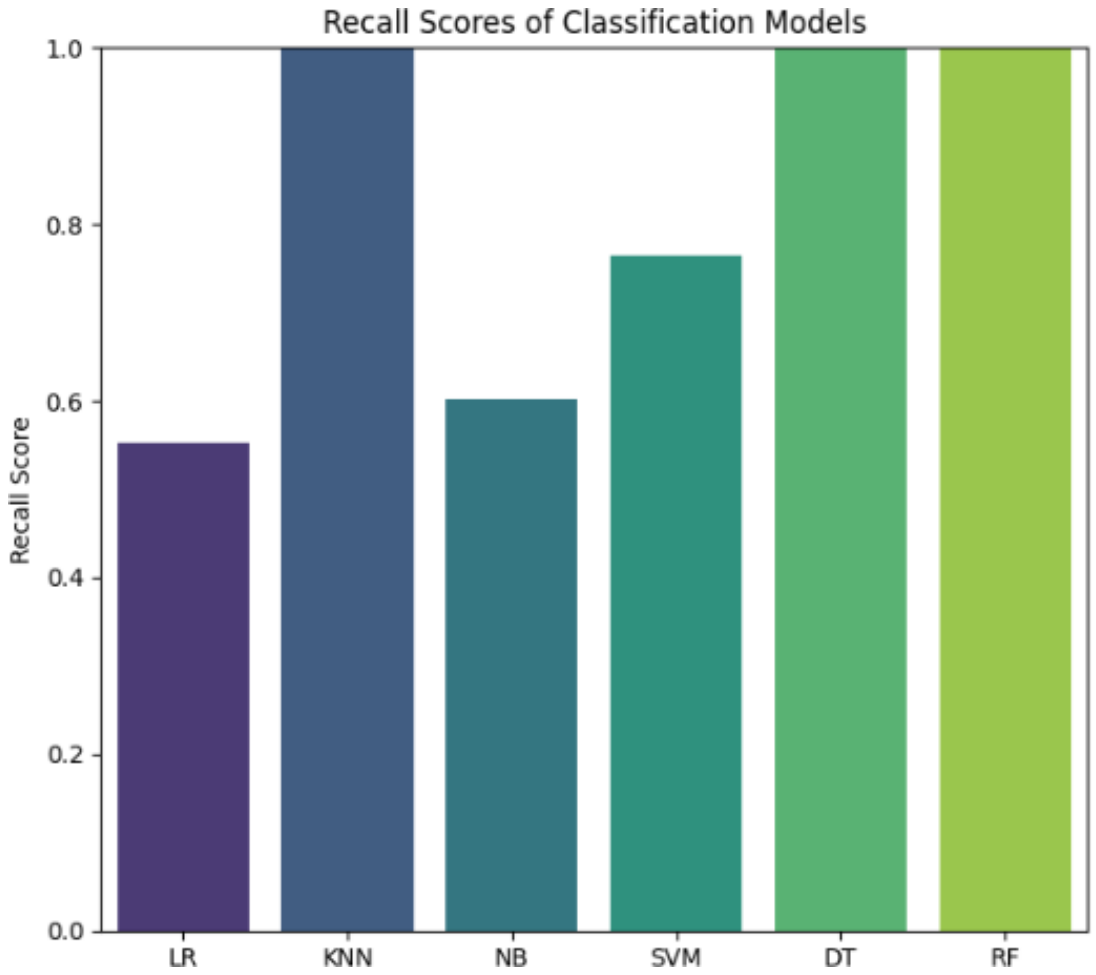

Fig 11.6 Recall Score plot

12.4 Precision Score and Potting of 6 classification algorithms:

```
Precision Scores:
Logistic Regression: 0.7117117117117117
K Nearest Neighbors: 1.0
Naive Bayes: 0.6386138613861386
Support Vector Machine: 0.8098765432098766
Decision Tree: 1.0
Random Forest: 1.0
```

Fig 11.7 Precision Score

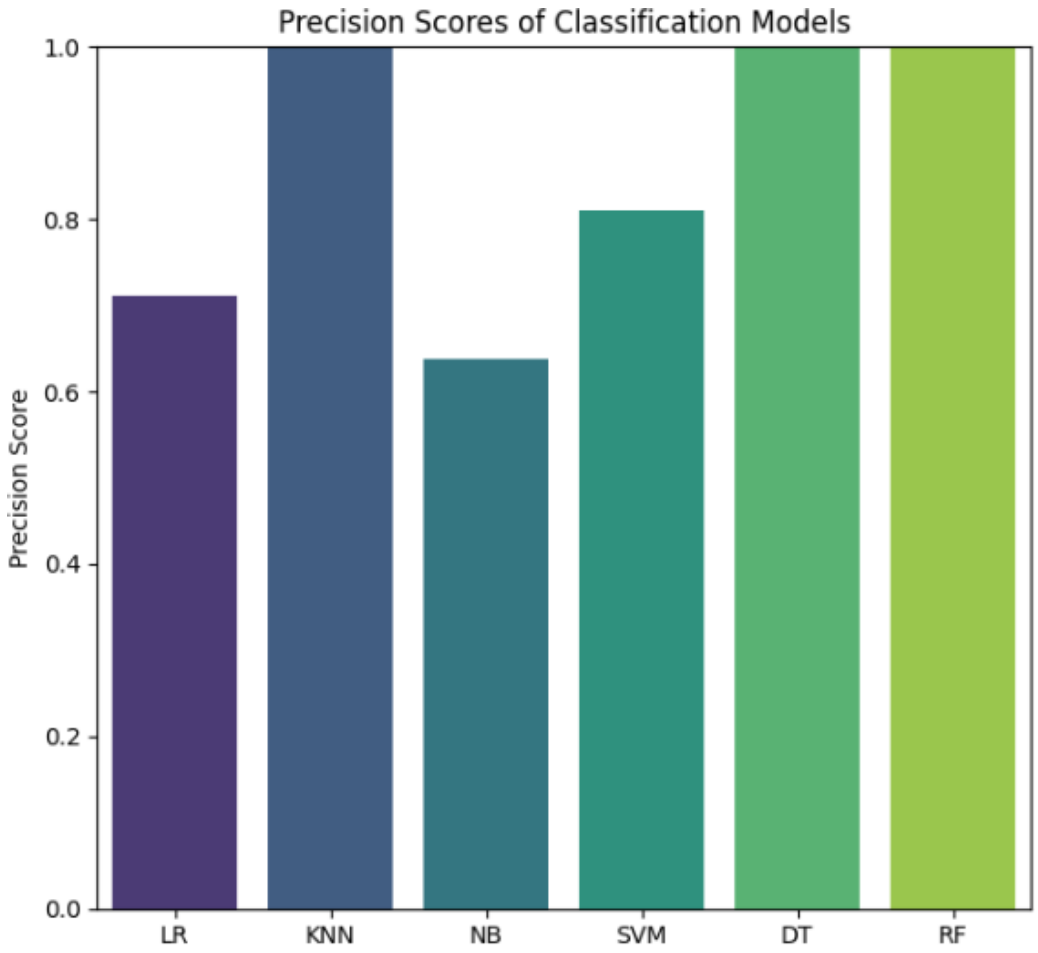

Fig 11.8 Precision Score Plot

12.6 F1 score and plotting of 6 classification algorithms:

```
F1 Scores:
Logistic Regression: 0.6220472440944882
K Nearest Neighbors: 1.0
Naive Bayes: 0.6194477791116446
Support Vector Machine: 0.7865707434052758
Decision Tree: 1.0
Random Forest: 1.0
```

Fig 11.9 F1-Score

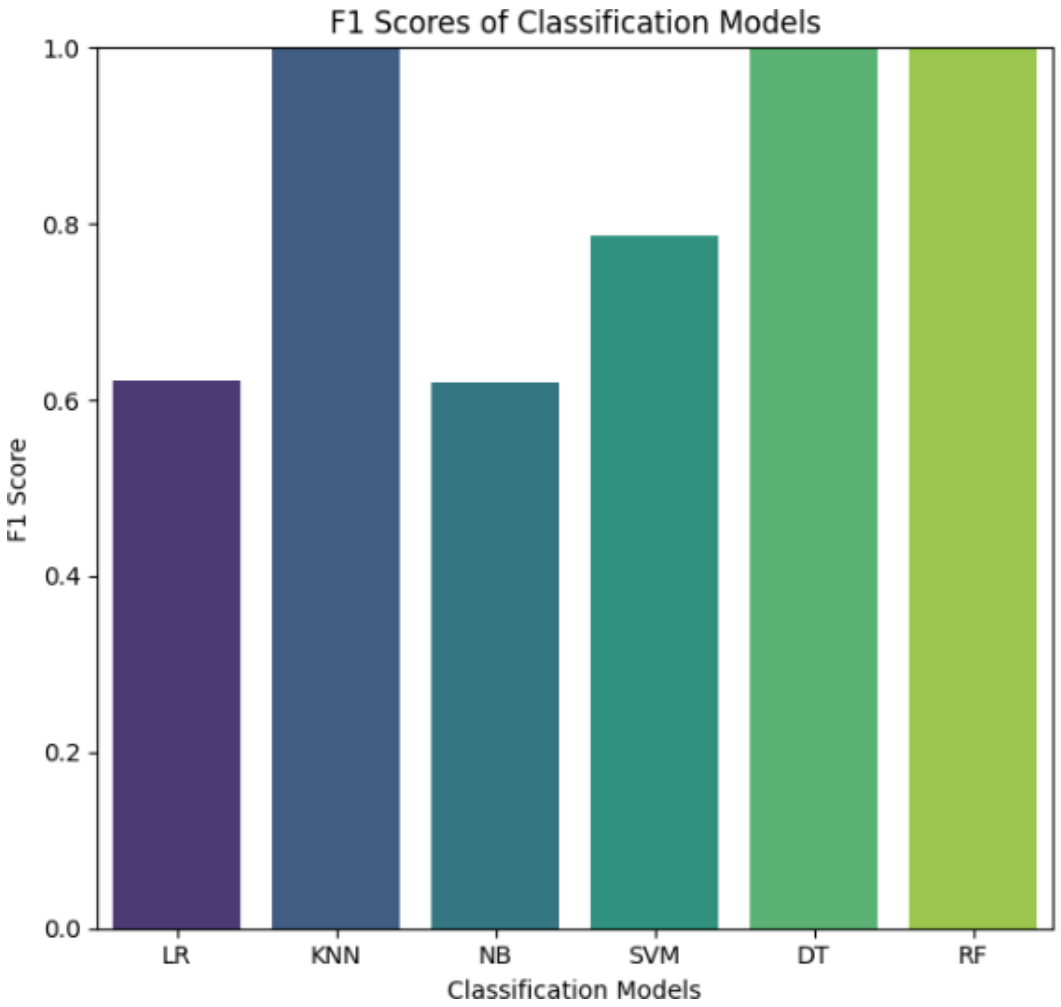

Fig 11.10 F1-Score Plot

12.7 Mean Square Error Score and plotting of 6 classification algorithms:

```
LR: MSE = 0.225730994152046 78
KNN: MSE = 0.0304093567251462
NB: MSE = 0.247953216374269
SVM: MSE = 0.13450292397660818
DT: MSE = 0.0
RF: MSE = 0.0
```

Fig 11.11 Mean Square Error

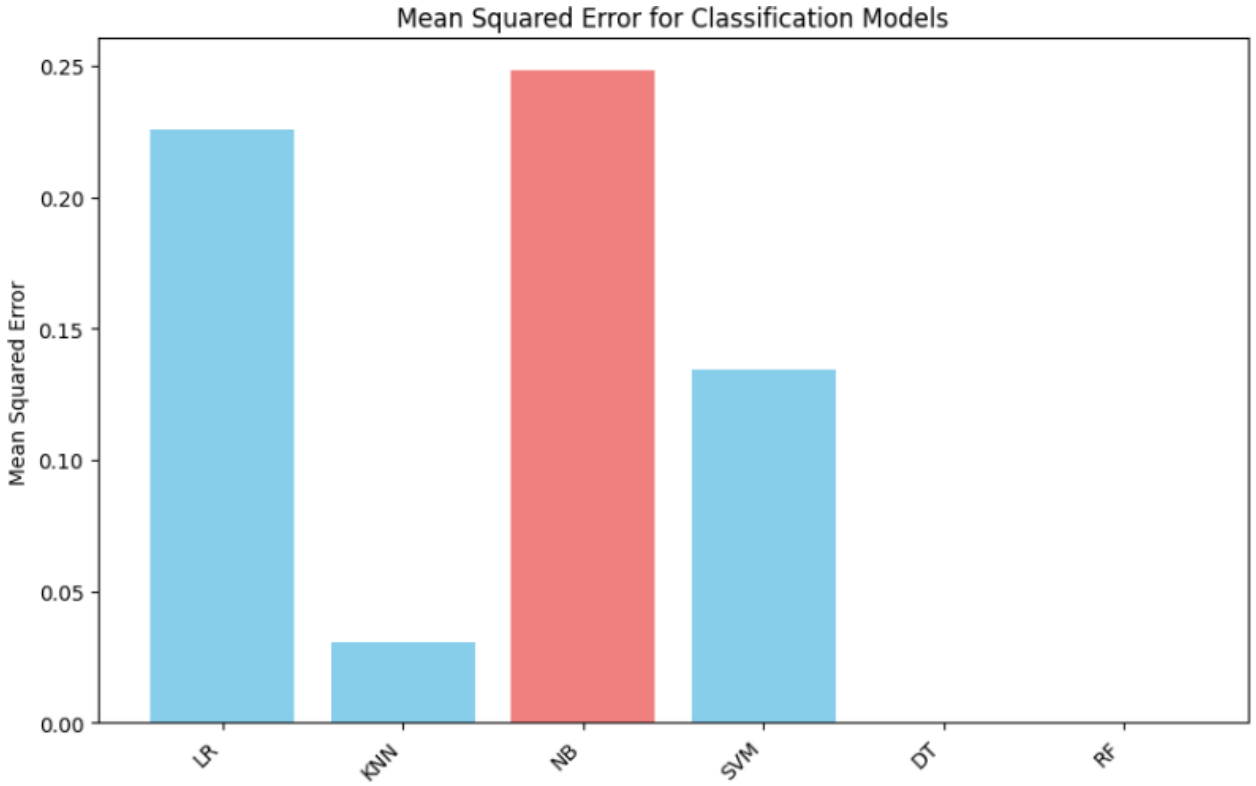

Fig 11.12 Mean Square Error Plot

12.8 Aggregate score and plotting of above performance matrix scores

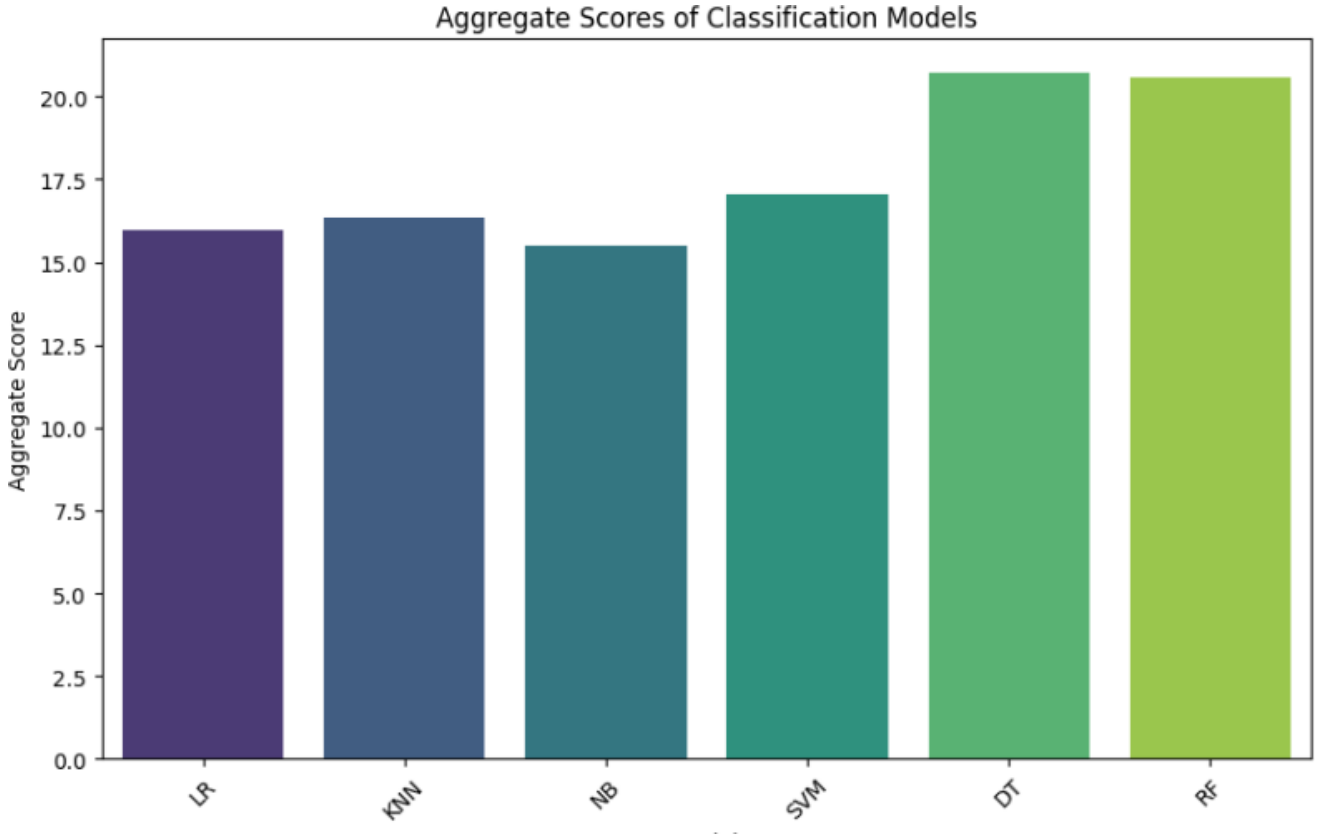

Fig 11.13 Aggregate Scores

```
LR: Aggregate Score = 15.962329964735224
KNN: Aggregate Score = 16.36691103858373
NB: Aggregate Score = 15.49434389606372
SVM: Aggregate Score = 17.051557969300273
DT: Aggregate Score = 20.701406170015197
RF: Aggregate Score = 20.60284065190607

The best model for diabetes prediction based on aggregate score
 is 'DT' with an aggregate score of 20.70.
```

Fig 11.14 Aggregate Score and Best Model for Diabetes Prediction

13. CONCLUSION

In this comprehensive investigation titled " a comparative study of ml algorithms for predicting diabetes," we explored the efficacy of six distinct machine learning models in predicting diabetes onset. Through rigorous experimentation and analysis, we uncovered valuable insights into their performance across various metrics, including accuracy, precision, recall, mean square error and F1-score.

Our findings revealed nuanced differences among the models, showcasing unique strengths and weaknesses. While each algorithm exhibited promising capabilities in diabetes prediction, the Decision Tree emerged as the frontrunner, showcasing superior predictive accuracy and robustness.

However, the selection of the optimal model is not merely a matter of performance metrics. Factors such as interpretability, **computational efficiency, and scalability are equally crucial,** especially in real-world deployment scenarios.

This comparative study contributes to the burgeoning field of predictive healthcare analytics, offering clinicians and researchers valuable guidance in selecting the most suitable machine learning approach for diabetes prediction. By harnessing the power of advanced analytics, we can potentially revolutionize early detection and intervention strategies, ultimately improving patient outcomes and reducing the burden of diabetes-related complications on healthcare systems worldwide.

14. REFERENCES

[1]. Aljumah, A.A., Ahamad, M.G., Siddiqui, M.K., 2013. Application of data mining: Diabetes health care in young and old patients. Journal of King Saud University - Computer and Information Sciences 25, 127–136. doi: 10.1016/j.jksuci.2012.10.003.

[2]. Arora, R., Suman, 2012. Comparative Analysis of Classification Algorithms on Different Datasets using WEKA. International Journal of Computer Applications 54, 21–25. doi:10.5120/8626-2492.

[3]. Bamnote, M.P., G.R., 2014. Design of Classifier for Detection of Diabetes Mellitus Using Genetic Programming. Advances in Intelligent Systems and Computing 1, 763–770. doi:10.1007/978-3- 319-11933-5.

[4]. Choubey, D.K., Paul, S., Kumar, S., Kumar, S., 2017. Classification of Pima Indian diabetes dataset using naive bayes with genetic algorithm as an attribute selection, in: Communication and Computing Systems: Proceedings of the International Conference on Communication and Computing System (ICCCS 2016), pp. 451– 455.

[5]. Dhomse Kanchan B., M.K.M., 2016. Study of Machine Learning Algorithms for Special Disease Prediction using Principal of Component Analysis, in: 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication, IEEE.

[6]. Sharief, A.A., Sheta, A., 2014. Developing a Mathematical Model to Detect Diabetes Using Multigene Genetic Programming. International Journal of Advanced Research in Artificial Intelligence(IJARAI)3, 54– 59.doi:doi:10.14569/IJARAI.2014.031007.

[7]. Sisodia, D., Shrivastava, S.K.,Jain, R.C., 2010.ISVM for face recognition. Proceedings - 2010 International Conference on Computational Intelligence and Communication Networks, CICN 2010, 554– 559doi:10.1109/CICN.2010.109.

[8]. Sisodia, D., Singh, L., Sisodia, S., 2014. Fast and Accurate Face Recognition Using SVM and DCT, in: Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012, Springer. pp. 1027–1038.

[9]. https://www.kaggle.com/johndasilva/diabetes

[10]. Rani, A. S., & Jyothi, S. (2016, March). Performance analysis of classification algorithms under different datasets. In Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on (pp. 1584- 158).