|  |  |  |
| --- | --- | --- |
|  |  |  |

# Diabetes Disease Prediction Using Machine Learning

**ABSTRACT**

**ABSTRACT**

This paper explores the prediction of Diabetes Disease utilizing an analysis of five supervised machine learning algorithms: K-Nearest Neighbors (KNN), Naïve Bayes, Decision Tree Classifier, Random Forest, and Support Vector Machine (SVM). By incorporating all pertinent risk factors from the dataset, stable accuracies were achieved post-classification and cross-validation. The study attained a robust accuracy of 76% with the KNN classifier, while other classifiers also exhibited stable accuracies surpassing 70%. Through visualization of training and testing accuracies and examination of model overfitting and underfitting, insights were gained into why specific classifiers failed to yield stable and optimal accuracies. The primary objective is to ascertain the most optimal results concerning accuracy and computational time for Diabetes disease prediction, considering the performance of all aforementioned algorithms. This research contributes to the advancement of predictive modeling in healthcare, offering insights into the comparative effectiveness of various machine learning techniques for disease prognosis.

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S.NO | CONTENT | PGNO |
| 1 | Introduction   * 1. Objective   2. Problem Statement   3. Software requirements   4. Hardware requirements |  |
| 2 | Feasibility study |  |
| 3 | Literature survey |  |
| 4 | System analysis  4.1 Existing system  4.1.1 Disadvantages of existing system  4.2 Proposed system  4.2.1 Advantages of proposed system  4.3 Functional requirements  4.4 Non-Functional requirements |  |
| 5 | System design  5.1 System architecture  5.2 UML diagrams |  |
| 6 | Implementation  6.1 Modules  6.2 Sample code |  |
| 7 | Software environment |  |
| 8 | System testing  8.1 Testing strategies  8.2 Test cases |  |
| 9 | Screens |  |
| 10 | Conclusion |  |
| 11 | References |  |

**INTRODUCTION**

**1. INTRODUCTION**

In this day and age, one of the most notorious diseases to have taken the world by storm is Diabetes, which is a disease which causes an increase in blood glucose levels as a result of the absence or low levels of insulin. Due to the many criterion to be taken into consideration for an individual to harbour this disease, it’s detection and prediction might be tedious or sometimes inconclusive. Nevertheless, it isn’t impossible to detect it, even at an early stage. In the year 2019, approx. 463 million adults between the age of 20-79 years had diabetes (International Diabetes Federation- IDF). 79% of the adult population were living in the countries with the low and middle-income groups. It is estimated that by the year 2045 approx. 700 million people will have diabetes (IDF). Diabetes is increasing day by day in the world because of environmental, genetic factors. The numbers are rising rapidly due to several factors which includes unhealthy foods, physical inactivity and many more. Diabetes is a hormonal disorder in which the inability of the body to produce insulin causes the metabolism of sugar in the body to be abnormal, thereby, raising the blood glucose levels in the body of a particular individual. Intense hunger, thirst and frequent urination are some of the observable characteristics. Certain risk factors such as age, BMI, Glucose Levels, Blood Pressure, etc., play an important role to the contribution of the disease.

**1.1 Objective:**

This paper aims to explore the prediction of Diabetes Disease utilizing five supervised machine learning algorithms: K-Nearest Neighbors, Naïve Bayes, Decision Tree Classifier, Random Forest, and Support Vector Machine. It seeks to evaluate their efficacy in terms of accuracy and computational efficiency.

**1.2 Problem Statement:**

1. Diabetes prevalence is rising due to lifestyle changes, genetic predisposition, and inadequate healthcare access.
2. Existing diagnostic methods lack efficiency, necessitating advanced predictive models for early detection and management.
3. Individuals globally, especially those with predisposing factors, face heightened risk and health complications.
4. Increased healthcare burden, compromised quality of life, and economic implications accompany undiagnosed or poorly managed diabetes.
5. Employing machine learning algorithms to develop accurate and timely predictive models for diabetes diagnosis, facilitating proactive intervention and improved healthcare outcomes.

**1.3 SOFTWARE REQUIREMENTS**

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

**Platform –** In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer’s architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the converse is not always true. Similarly, software designed using newer features of Linux Kernel v2.6 generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

**APIs and drivers –** Software making extensive use of special hardware devices, like high-end display adapters, needs special API or newer device drivers. A good example is DirectX, which is a collection of APIs for handling tasks related to multimedia, especially game programming, on Microsoft platforms.

**Web browser –** Most web applications and software depending heavily on Internet technologies make use of the default browser installed on system. Microsoft Internet Explorer is a frequent choice of software running on Microsoft Windows, which makes use of ActiveX controls, despite their vulnerabilities.

**1) Software: Anaconda**

**2) Primary Language: Python**

**3) Frontend Framework: Flask**

**4) Back-end Framework: Jupyter Notebook**

**5) Database: Sqlite3**

**6) Front-End Technologies: HTML, CSS, JavaScript and Bootstrap4**

**1.4 HARDWARE REQUIREMENTS**

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

**Architecture –** All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems and their supporting architectures.

**Processing power –** The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity, and are often mentioned in this category.

**Memory –** All software, when run, resides in the random access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system, supporting software and files, and other running processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

**Secondary storage –** Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

**Display adapter –** Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the system requirements.

**Peripherals –** Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

**1) Operating System: Windows Only**

**2) Processor: i5 and above**

**3) Ram: 8gb and above**

**4) Hard Disk: 25 GB in local drive**

**FEASIBILITY STUDY**

**2. FEASIBILITY STUDY**

**Feasibility Study**

A feasibility study evaluates a project's or system's practicality. As part of a feasibility study, the objective and rational analysis of a potential business or venture is conducted to determine its strengths and weaknesses, potential opportunities and threats, resources required to carry out, and ultimate success prospects. Two criteria should be considered when judging feasibility: the required cost and expected value.

**Types Of Feasibility Study**

A feasibility analysis evaluates the project’s potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lending institutions. There are five types of feasibility study—separate areas that a feasibility study examines, described below.

**1. Technical Feasibility**

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system. As an exaggerated example, an organization wouldn’t want to try to put Star Trek’s transporters in their building—currently, this project is not technically feasible.

**2. Economic Feasibility**

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent [project assessment](https://www.simplilearn.com/risk-assessment-project-management-article) and enhances project credibility—helping decision-makers determine the positive economic benefits to the organization that the proposed project will provide.

### **3. Legal Feasibility**

This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws,[data protection](https://www.simplilearn.com/understanding-data-security-rar30-article) acts or social media laws. Let’s say an organization wants to construct a new office building in a specific location. A feasibility study might reveal the organization’s ideal location isn’t zoned for that type of business. That organization has just saved considerable time and effort by learning that their project was not feasible right from the beginning.

### **4. Operational Feasibility**

This assessment involves undertaking a study to analyze and determine whether—and how well—the organization’s needs can be met by completing the project. Operational feasibility studies also examine how a [project plan](https://www.simplilearn.com/project-management-plans-in-project-environment-rar79-article) satisfies the requirements identified in the requirements analysis phase of system development.

### **5. Scheduling Feasibility**

This assessment is the most important for [project success](https://www.simplilearn.com/how-to-make-a-project-successful-article); after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete.

When these areas have all been examined, the feasibility analysis helps identify any constraints the proposed project may face, including:

* Internal Project Constraints: Technical, Technology, Budget, Resource, etc.
* Internal Corporate Constraints: Financial, Marketing, Export, etc.
* External Constraints: Logistics, Environment, Laws, and Regulations, etc.

**LITERATURE SURVEY**

**3. LITERATURE SURVEY**

**3.1 Diabetes Disease Prediction Using Machine Learning on Big Data of Healthcare:**

<https://ieeexplore.ieee.org/document/8697439>

**ABSTRACT:** Healthcare domain is a very prominent research field with rapid technological advancement and increasing data day by day. In order to deal with large volume of healthcare data we need Big Data Analytics which is an emerging approach in Healthcare domain. Millions of patients seek treatments around the globe with various procedure. Analyzing the trends in treatment of patients for diagnosis of a particular disease will help in making informed and efficient decisions to improve the overall quality of healthcare. Machine Learning is a very promising approach which helps in early diagnosis of disease and might help the practitioners in decision making for diagnosis. This paper aims at building a classifier model using WEKA tool to predict diabetes disease by employing Naive Bayes, Support Vector Machine, Random Forest and Simple CART algorithm. The research hopes to recommend the best algorithm based on efficient performance result for the prediction of diabetes disease. Experimental results of each algorithm used on the dataset was evaluated. It is observed that Support Vector Machine performed best in prediction of the disease having maximum accuracy.

**3.2 Prediction of Diabetes using Classification Algorithms:**

<https://www.sciencedirect.com/science/article/pii/S1877050918308548>

**ABSTRACT:** Diabetes is considered as one of the deadliest and chronic diseases which causes an increase in blood sugar. Many complications occur if diabetes remains untreated and unidentified. The tedious identifying process results in visiting of a patient to a diagnostic center and consulting doctor. But the rise in machine learning approaches solves this critical problem. The motive of this study is to design a model which can prognosticate the likelihood of diabetes in patients with maximum accuracy. Therefore three machine learning classification algorithms namely Decision Tree, SVM and Naive Bayes are used in this experiment to detect diabetes at an early stage. Experiments are performed on Pima Indians Diabetes Database (PIDD) which is sourced from UCI machine learning repository. The performances of all the three algorithms are evaluated on various measures like Precision, Accuracy, F-Measure, and Recall. Accuracy is measured over correctly and incorrectly classified instances. Results obtained show Naive Bayes outperforms with the highest accuracy of 76.30% comparatively other algorithms. These results are verified using Receiver Operating Characteristic (ROC) curves in a proper and systematic manner.

**3.3 Global and regional diabetes prevalence estimates for 2019 and projections for 2030 and 2045: Results from the International Diabetes Federation Diabetes Atlas, 9th edition:**

<https://www.sciencedirect.com/science/article/pii/S0168822719312306>

**ABSTRACT:** Aims To provide global estimates of diabetes prevalence for 2019 and projections for 2030 and 2045. Methods A total of 255 high-quality data sources, published between 1990 and 2018 and representing 138 countries were identified. For countries without high quality in-country data, estimates were extrapolated from similar countries matched by economy, ethnicity, geography and language. Logistic regression was used to generate smoothed age-specific diabetes prevalence estimates (including previously undiagnosed diabetes) in adults aged 20–79 years. Results The global diabetes prevalence in 2019 is estimated to be 9.3% (463 million people), rising to 10.2% (578 million) by 2030 and 10.9% (700 million) by 2045. The prevalence is higher in urban (10.8%) than rural (7.2%) areas, and in high-income (10.4%) than low-income countries (4.0%). One in two (50.1%) people living with diabetes do not know that they have diabetes. The global prevalence of impaired glucose tolerance is estimated to be 7.5% (374 million) in 2019 and projected to reach 8.0% (454 million) by 2030 and 8.6% (548 million) by 2045. Conclusions Just under half a billion people are living with diabetes worldwide and the number is projected to increase by 25% in 2030 and 51% in 2045.

**3.4 Using the ADAP Learning Algorithm to Forcast the Onset of Diabetes Mellitus:**

<https://www.researchgate.net/publication/248284447_Using_the_ADAP_Learning_Algorithm_to_Forcast_the_Onset_of_Diabetes_Mellitus>

**ABSTRACT:** Neural networks or connectionist models for parallel processing are not new. However, a resurgence of interest in the past half decade has occurred. In part, this is related to a better understanding of what are now referred to as hidden nodes. These algorithms are considered to be of marked value in pattern recognition problems. Because of that, we tested the ability of an early neural network model, ADAP, to forecast the onset of diabetes mellitus in a high risk population of Pima Indians. The algorithm's performance was analyzed using standard measures for clinical tests: sensitivity, specificity, and a receiver operating characteristic curve. The crossover point for sensitivity and specificity is 0.76. We are currently further examining these methods by comparing the ADAP results with those obtained from logistic regression and linear perceptron models using precisely the same training and forecasting sets. A description of the algorithm is included.

**3.5 Application of Machine Learning in Disease Prediction:**

<https://ieeexplore.ieee.org/document/8777449>

**ABSTRACT:** The application of machine learning in the field of medical diagnosis is increasing gradually. This can be contributed primarily to the improvement in the classification and recognition systems used in disease diagnosis which is able to provide data that aids medical experts in early detection of fatal diseases and therefore, increase the survival rate of patients significantly. In this paper, we apply different classification algorithms, each with its own advantage on three separate databases of disease (Heart, Breast cancer, Diabetes) available in UCI repository for disease prediction. The feature selection for each dataset was accomplished by backward modeling using the p-value test. The results of the study strengthen the idea of the application of machine learning in early detection of diseases.

**SYSTEM ANALYSIS**

**4. SYSTEM ANALYSIS**

**4.1 EXISTING SYSTEM:**

In the existing system, researchers utilized the WEKA tool to analyze healthcare data for predicting diabetes. They employed publicly available datasets, such as the one from UCI and the Pima Indians Diabetes Database. After accessing the datasets, they preprocessed the data within the WEKA tool. For model training and testing, they split the data into 70% for training and 30% for testing, without using cross-validation. The machine learning classifiers used include Naive Bayes, Support Vector Machine, Random Forest, and Simple CART. Evaluation involved comparing the performance of different classifiers, focusing on metrics like accuracy. The process involved dataset selection, preprocessing, algorithm application, and performance analysis, aiming to predict diabetes accurately using machine learning techniques.

**4.1.1 DISADVANTAGES OF EXISTING SYSTEM:**

1. Omitting cross-validation can lead to overfitting or underfitting of models, potentially reducing the generalization ability of the predictive models.
2. While the existing system employs popular classifiers, it may overlook newer or more advanced algorithms that could potentially improve prediction accuracy.
3. Reliance on only one or two datasets limits the diversity of data sources and may not capture the full spectrum of factors influencing diabetes prediction.
4. Sole reliance on accuracy as the performance metric may overlook other important aspects such as sensitivity, specificity, and area under the curve (AUC) of the receiver operating characteristic (ROC).

# 4.2 Proposed System:

The proposed system entails the development and implementation of a predictive model for diabetes diagnosis using machine learning algorithms. It involves collecting comprehensive datasets containing relevant risk factors associated with diabetes, including genetic predisposition, lifestyle habits, and medical history. These datasets are preprocessed to ensure data quality and compatibility for analysis. Subsequently, various supervised machine learning algorithms such as K-Nearest Neighbors, Naïve Bayes, Decision Tree Classifier, Random Forest, and Support Vector Machine are employed for model training and evaluation. The system focuses on achieving high accuracy and computational efficiency in diabetes prediction, leveraging the strengths of each algorithm and employing techniques to mitigate overfitting and underfitting. Cross-validation techniques are utilized to validate model performance and ensure robustness. The proposed system aims to provide healthcare professionals with a reliable tool for early detection and proactive management of diabetes, ultimately improving patient outcomes and reducing the burden on healthcare systems.

# 4.2.1 Advantages of proposed system:

1. The proposed system emphasizes gathering extensive datasets covering various risk factors associated with diabetes, enabling more robust and accurate predictive models.
2. Incorporating a wider range of machine learning algorithms allows for a more thorough exploration of model performance and potentially higher prediction accuracy.
3. Integration of cross-validation techniques ensures better model generalization and reliability by validating performance across multiple folds of data.
4. Leveraging advanced techniques to mitigate overfitting and underfitting enhances model robustness and ensures better performance in real-world scenarios.

Top of Form

Top of Form

### **4.3 FUNCTIONAL REQUIREMENTS**

1. Data Collection

2. Image Processing

3. Training and Testing

4. Modelling

5. Predicting

### **4.4 NON FUNCTIONAL REQUIREMENTS**

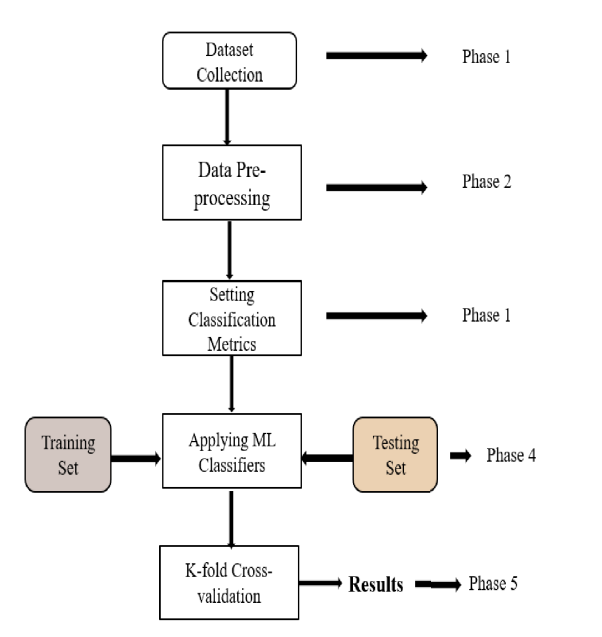
NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *“how fast does the website load?”* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non- functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

* Usability requirement
* Serviceability requirement
* Manageability requirement
* Recoverability requirement
* Security requirement
* Data Integrity requirement
* Capacity requirement
* Availability requirement
* Scalability requirement
* Interoperability requirement
* Reliability requirement
* Maintainability requirement
* Regulatory requirement
* Environmental requirement

**SYSTEM DESIGN**

**5. SYSTEM DESIGN**

**5.1 SYSTEM ARCHITECTURE:**

****

**Fig.5.1.1 System architecture**

**DATA FLOW DIAGRAM:**

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

**Import libraries**

**VERIFY**

**NO PROCESS**

**Yes NO**

**Importing the dataset**

**Data processing**

**Feature selection**

**Splitting the data into train & test**

**Building the model in colab – Naïve Bayes – SVM – Decision Tree – Random Forest - KNN**

**Training the model**

**Signup & sign in**

**User input**

**Final outcome**

**End process**

**5.2 UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

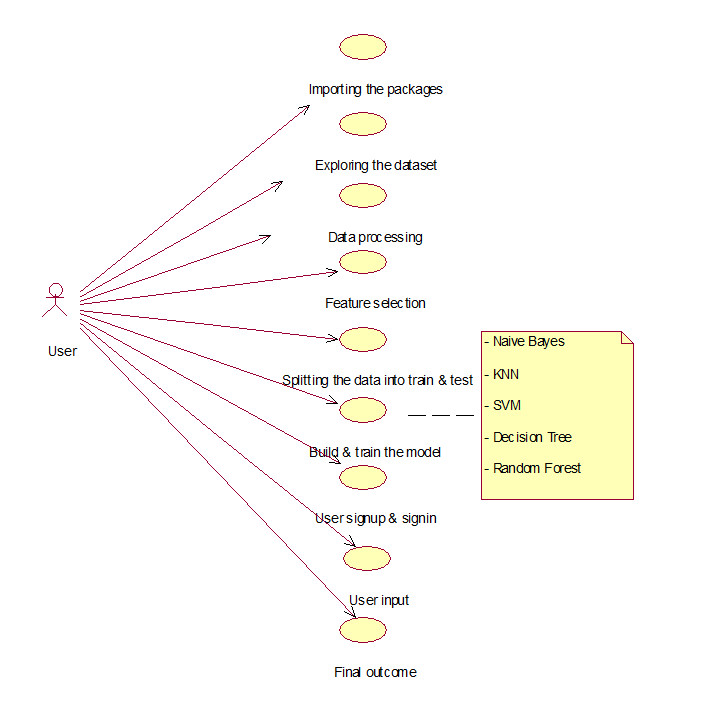
The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:**

The Primary goals in the design of the UML are as follows:

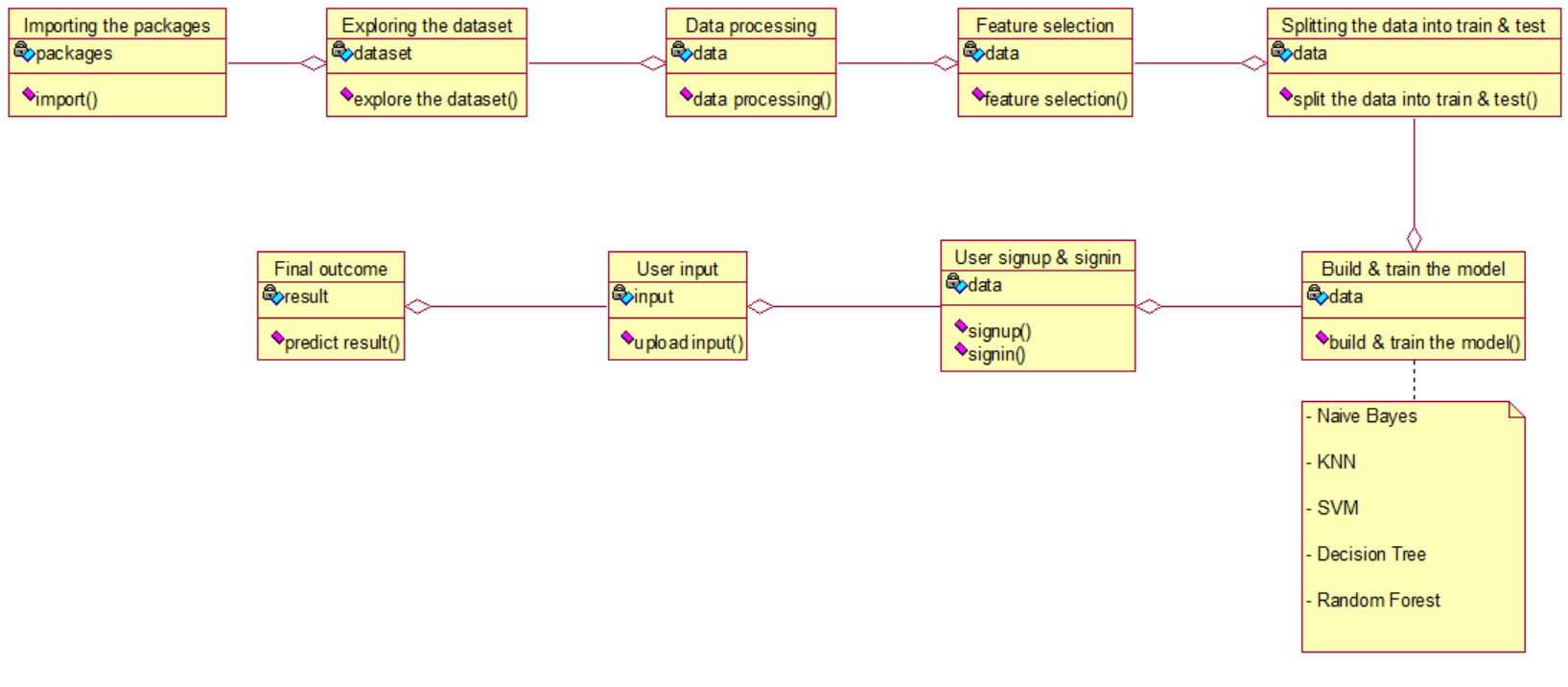
1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

**Use case diagram:**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

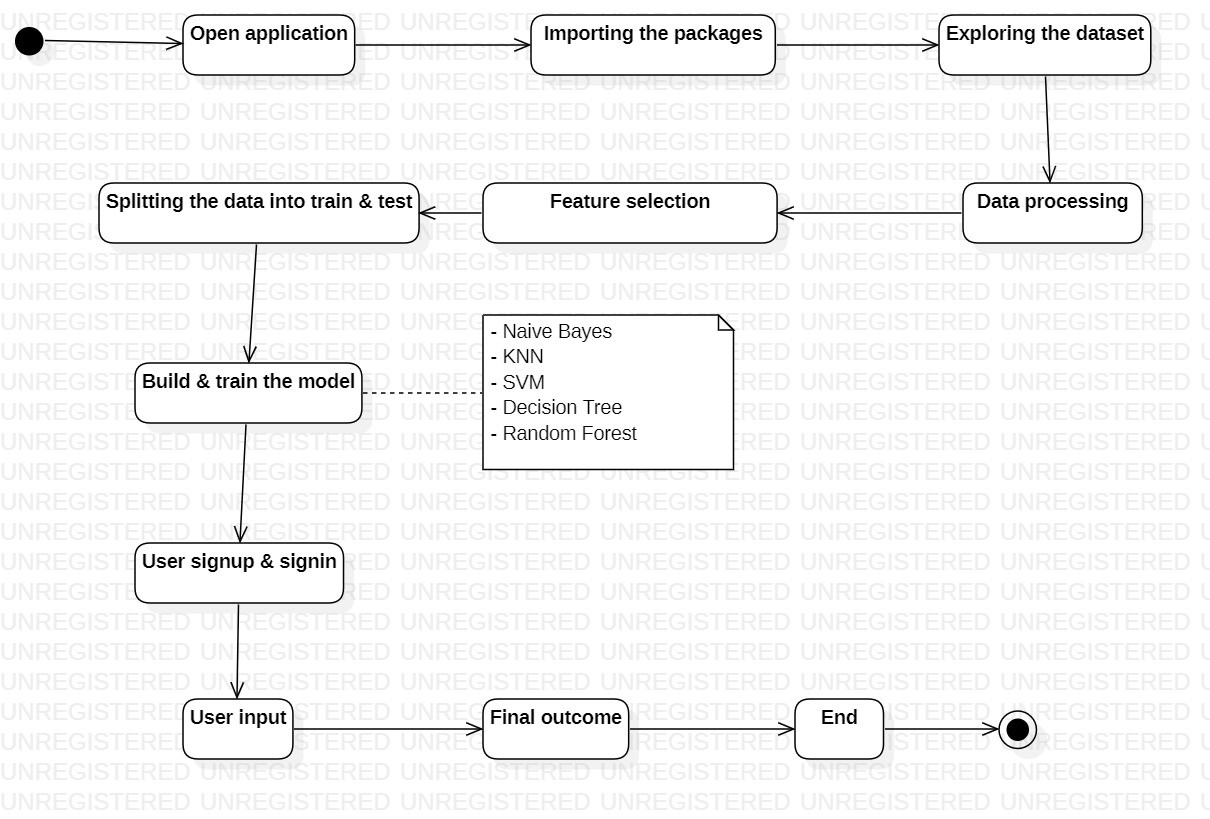
**Class diagram:**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.



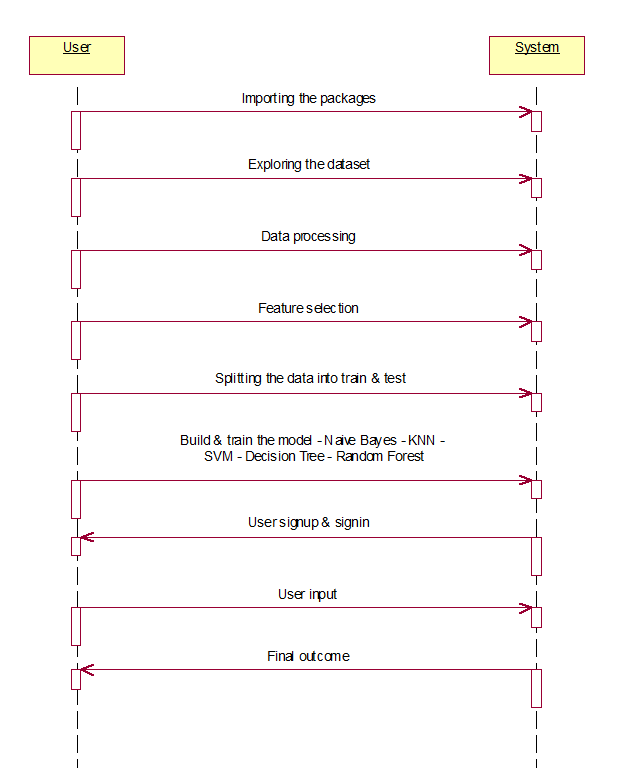
**Activity diagram:**

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.



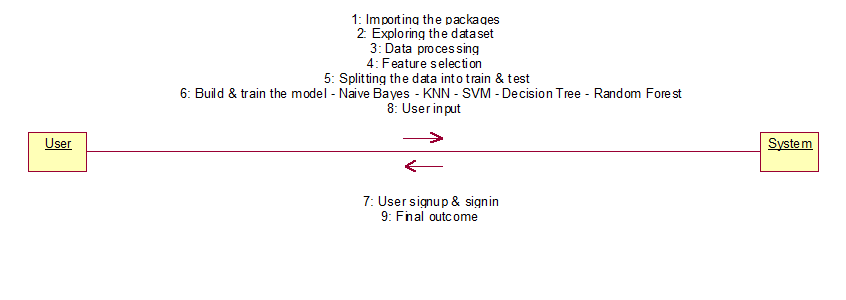
**Sequence diagram:**

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".



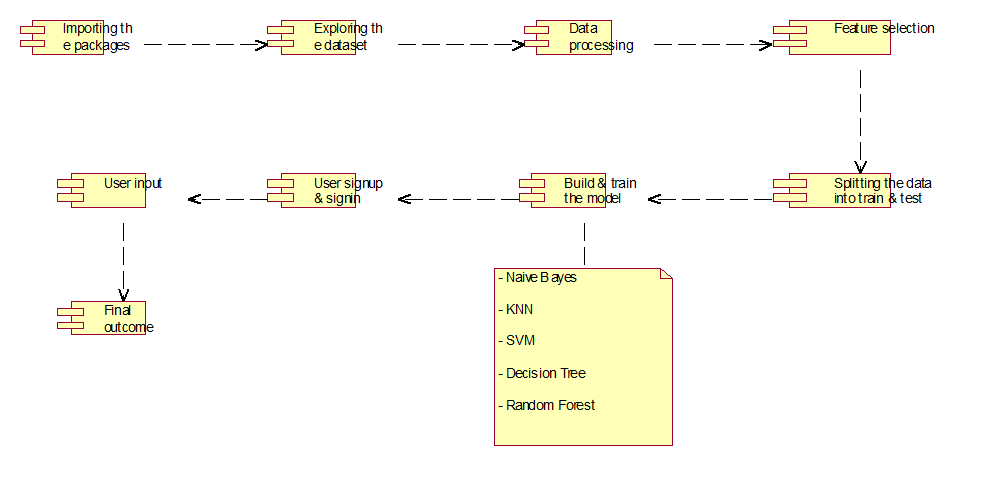
**Collaboration diagram:**

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

****

**Component diagram:**

The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.



**Deployment diagram:**

The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

****

**IMPLEMENTATION**

1. **IMPLEMENTATION**

**MODULES:**

* **Data loading:** using this module we are going to import the dataset.
* **Data Preprocessing:** using this module we will explore the data.
* **Splitting data into train & test:** using this module data will be divided into train & test
* **Model generation:** Model building - Naïve Bayes – SVM – Decision Tree – Random Forest - KNN . Algorithms accuracy calculated
* **User signup & login:** Using this module will get registration and login
* **User input:** Using this module will give input for prediction
* **Prediction:** final predicted displayed

**Algorithms:**

**Naïve Bayes:**

Naïve Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features. It calculates the probability of each class given a set of features and selects the class with the highest probability. Naïve Bayes is utilized in the project for diabetes disease prediction by calculating the probability of a patient having diabetes given their set of risk factors. It handles categorical and continuous data efficiently and is computationally inexpensive, making it suitable for large datasets.

**Support Vector Machine (SVM):**

SVM is a supervised learning algorithm used for classification tasks. It constructs a hyperplane or set of hyperplanes in a high-dimensional space to separate data points into different classes with the maximum margin. SVM is applied in the project for diabetes prediction by finding the optimal hyperplane that separates patients with diabetes from those without. It is effective in handling high-dimensional data and is particularly useful when dealing with nonlinear relationships between features.

**Decision Tree:**

Decision Tree is a tree-like model where an internal node represents a feature, the branch represents a decision rule, and each leaf node represents the outcome. It partitions the data into subsets based on the value of the features. Decision Tree is employed in the project for diabetes prediction by recursively partitioning the dataset based on the most significant features. It is interpretable and easy to visualize, providing insights into the decision-making process.

**Random Forest:**

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes for classification or the mean prediction for regression. Random Forest is utilized in the project for diabetes prediction by building a forest of decision trees and aggregating their predictions. It reduces overfitting and improves prediction accuracy by introducing randomness in the tree-building process.

**K-Nearest Neighbors (KNN):**

KNN is a non-parametric algorithm used for classification and regression tasks. It assigns the class membership of a data point based on the majority class among its k nearest neighbors. KNN is applied in the project for diabetes prediction by calculating the distance between data points in the feature space and classifying a patient based on the classes of its nearest neighbors. It is simple yet effective, especially when the decision boundary is not well-defined.

**6.2 SAMPLE CODE:**

*import numpy as* *np # linear algebra*

*import pandas as* *pd # data processing, CSV file I/O (e.g. pd.read\_csv)*

*import matplotlib.pyplot as* *plt #to plot charts*

*import seaborn as* *sns #used for data visualization*

*import* *warnings #avoid warning flash*

*warnings.filterwarnings('ignore')*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

*import* *os*

*for* *dirname, \_, filenames* ***in*** *os.walk('/kaggle/input'):*

*for* *filename* ***in*** *filenames:*

*print(os.path.join(dirname, filename))*

df=pd.read\_csv("../input/pima-indians-diabetes-database/diabetes.csv")

df.head()

df.shape

df.columns

df.dtypes

df.info()

df.describe()

df=df.drop\_duplicates()

df.isnull().sum()

print(df[df['BloodPressure']==0].shape[0])

print(df[df['Glucose']==0].shape[0])

print(df[df['SkinThickness']==0].shape[0])

print(df[df['Insulin']==0].shape[0])

print(df[df['BMI']==0].shape[0])

df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())*#normal distribution*

df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())*#normal distribution*

df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].median())*#skewed distribution*

df['Insulin']=df['Insulin'].replace(0,df['Insulin'].median())*#skewed distribution*

df['BMI']=df['BMI'].replace(0,df['BMI'].median())*#skewed distribution*

sns.countplot('Outcome',data=df)

*#histogram for each feature*

df.hist(bins=10,figsize=(10,10))

plt.show()

plt.figure(figsize=(16,12))

sns.set\_style(style='whitegrid')

plt.subplot(3,3,1)

sns.boxplot(x='Glucose',data=df)

plt.subplot(3,3,2)

sns.boxplot(x='BloodPressure',data=df)

plt.subplot(3,3,3)

sns.boxplot(x='Insulin',data=df)

plt.subplot(3,3,4)

sns.boxplot(x='BMI',data=df)

plt.subplot(3,3,5)

sns.boxplot(x='Age',data=df)

plt.subplot(3,3,6)

sns.boxplot(x='SkinThickness',data=df)

plt.subplot(3,3,7)

sns.boxplot(x='Pregnancies',data=df)

plt.subplot(3,3,8)

sns.boxplot(x='DiabetesPedigreeFunction',data=df)

from pandas.plotting import scatter\_matrix

scatter\_matrix(df,figsize=(20,20));

corrmat=df.corr()

sns.heatmap(corrmat, annot=True)

df\_selected=df.drop(['BloodPressure','Insulin','DiabetesPedigreeFunction'],axis='columns')

from sklearn.preprocessing import QuantileTransformer

x=df\_selected

quantile = QuantileTransformer()

X = quantile.fit\_transform(x)

df\_new=quantile.transform(X)

df\_new=pd.DataFrame(X)

df\_new.columns =['Pregnancies', 'Glucose','SkinThickness','BMI','Age','Outcome']

df\_new.head()

plt.figure(figsize=(16,12))

sns.set\_style(style='whitegrid')

plt.subplot(3,3,1)

sns.boxplot(x=df\_new['Glucose'],data=df\_new)

plt.subplot(3,3,2)

sns.boxplot(x=df\_new['BMI'],data=df\_new)

plt.subplot(3,3,3)

sns.boxplot(x=df\_new['Pregnancies'],data=df\_new)

plt.subplot(3,3,4)

sns.boxplot(x=df\_new['Age'],data=df\_new)

plt.subplot(3,3,5)

sns.boxplot(x=df\_new['SkinThickness'],data=df\_new)

target\_name='Outcome'

y= df\_new[target\_name]*#given predictions - training data*

X=df\_new.drop(target\_name,axis=1)*#dropping the Outcome column and keeping all other columns as X*

X.head() *# contains only independent features*

y.head() *#contains dependent feature*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test= train\_test\_split(X,y,test\_size=0.2,random\_state=0)*#splitting data in 80% train, 20%test*

X\_train.shape,y\_train.shape

X\_test.shape,y\_test.shape

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import f1\_score, precision\_score, recall\_score

from sklearn.model\_selection import GridSearchCV

*#List Hyperparameters to tune*

knn= KNeighborsClassifier()

n\_neighbors = list(range(15,25))

p=[1,2]

weights = ['uniform', 'distance']

metric = ['euclidean', 'manhattan', 'minkowski']

*#convert to dictionary*

hyperparameters = dict(n\_neighbors=n\_neighbors, p=p,weights=weights,metric=metric)

*#Making model*

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

grid\_search = GridSearchCV(estimator=knn, param\_grid=hyperparameters, n\_jobs=-1, cv=cv, scoring='f1',error\_score=0)

best\_model = grid\_search.fit(X\_train,y\_train)

*#Best Hyperparameters Value*

print('Best leaf\_size:', best\_model.best\_estimator\_.get\_params()['leaf\_size'])

print('Best p:', best\_model.best\_estimator\_.get\_params()['p'])

print('Best n\_neighbors:', best\_model.best\_estimator\_.get\_params()['n\_neighbors'])

*#Predict testing set*

knn\_pred = best\_model.predict(X\_test)

print("Classification Report is:**\n**",classification\_report(y\_test,knn\_pred))

print("**\n** F1:**\n**",f1\_score(y\_test,knn\_pred))

print("**\n** Precision score is:**\n**",precision\_score(y\_test,knn\_pred))

print("**\n** Recall score is:**\n**",recall\_score(y\_test,knn\_pred))

print("**\n** Confusion Matrix:**\n**")

sns.heatmap(confusion\_matrix(y\_test,knn\_pred))

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import GridSearchCV

param\_grid\_nb = {

'var\_smoothing': np.logspace(0,-2, num=100)

}

nbModel\_grid = GridSearchCV(estimator=GaussianNB(), param\_grid=param\_grid\_nb, verbose=1, cv=10, n\_jobs=-1)

best\_model= nbModel\_grid.fit(X\_train, y\_train)

nb\_pred=best\_model.predict(X\_test)

print("Classification Report is:**\n**",classification\_report(y\_test,nb\_pred))

print("**\n** F1:**\n**",f1\_score(y\_test,nb\_pred))

print("**\n** Precision score is:**\n**",precision\_score(y\_test,nb\_pred))

print("**\n** Recall score is:**\n**",recall\_score(y\_test,nb\_pred))

print("**\n** Confusion Matrix:**\n**")

sns.heatmap(confusion\_matrix(y\_test,nb\_pred))

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import f1\_score, precision\_score, recall\_score

model = SVC()

kernel = ['poly', 'rbf', 'sigmoid']

C = [50, 10, 1.0, 0.1, 0.01]

gamma = ['scale']

*# define grid search*

grid = dict(kernel=kernel,C=C,gamma=gamma)

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

grid\_search = GridSearchCV(estimator=model, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='f1',error\_score=0)

grid\_result = grid\_search.fit(X, y)

svm\_pred=grid\_result.predict(X\_test)

print("Classification Report is:**\n**",classification\_report(y\_test,svm\_pred))

print("**\n** F1:**\n**",f1\_score(y\_test,knn\_pred))

print("**\n** Precision score is:**\n**",precision\_score(y\_test,knn\_pred))

print("**\n** Recall score is:**\n**",recall\_score(y\_test,knn\_pred))

print("**\n** Confusion Matrix:**\n**")

sns.heatmap(confusion\_matrix(y\_test,svm\_pred))

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import f1\_score, precision\_score, recall\_score

from sklearn.model\_selection import GridSearchCV

dt = DecisionTreeClassifier(random\_state=42)

*# Create the parameter grid based on the results of random search*

params = {

'max\_depth': [5, 10, 20,25],

'min\_samples\_leaf': [10, 20, 50, 100,120],

'criterion': ["gini", "entropy"]

}

grid\_search = GridSearchCV(estimator=dt,

param\_grid=params,

cv=4, n\_jobs=-1, verbose=1, scoring = "accuracy")

best\_model=grid\_search.fit(X\_train, y\_train)

dt\_pred=best\_model.predict(X\_test)

print("Classification Report is:**\n**",classification\_report(y\_test,dt\_pred))

print("**\n** F1:**\n**",f1\_score(y\_test,dt\_pred))

print("**\n** Precision score is:**\n**",precision\_score(y\_test,dt\_pred))

print("**\n** Recall score is:**\n**",recall\_score(y\_test,dt\_pred))

print("**\n** Confusion Matrix:**\n**")

sns.heatmap(confusion\_matrix(y\_test,dt\_pred))

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import f1\_score, precision\_score, recall\_score

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.model\_selection import GridSearchCV

*# define models and parameters*

model = RandomForestClassifier()

n\_estimators = [1800]

max\_features = ['sqrt', 'log2']

*# define grid search*

grid = dict(n\_estimators=n\_estimators,max\_features=max\_features)

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

grid\_search = GridSearchCV(estimator=model, param\_grid=grid, n\_jobs=-1, cv=cv, scoring='accuracy',error\_score=0)

best\_model = grid\_search.fit(X\_train, y\_train)

rf\_pred=best\_model.predict(X\_test)

print("Classification Report is:**\n**",classification\_report(y\_test,rf\_pred))

print("**\n** F1:**\n**",f1\_score(y\_test,knn\_pred))

print("**\n** Precision score is:**\n**",precision\_score(y\_test,knn\_pred))

print("**\n** Recall score is:**\n**",recall\_score(y\_test,knn\_pred))

print("**\n** Confusion Matrix:**\n**")

sns.heatmap(confusion\_matrix(y\_test,rf\_pred))

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report,confusion\_matrix

from sklearn.metrics import f1\_score, precision\_score, recall\_score,accuracy\_score

reg = LogisticRegression()

reg.fit(X\_train,y\_train)

lr\_pred=reg.predict(X\_test)

print("Classification Report is:**\n**",classification\_report(y\_test,lr\_pred))

print("**\n** F1:**\n**",f1\_score(y\_test,lr\_pred))

print("**\n** Precision score is:**\n**",precision\_score(y\_test,lr\_pred))

print("**\n** Recall score is:**\n**",recall\_score(y\_test,lr\_pred))

print("**\n** Confusion Matrix:**\n**")

sns.heatmap(confusion\_matrix(y\_test,lr\_pred))

**SOFTWARE ENVIRONMENT**

**7. SOFTWARE ENVIRONMENT**

**What is Anaconda for Python?**

Anaconda software helps you create an environment for many different versions of Python and package versions. Anaconda is also used to install, remove, and upgrade packages in your project environments. Furthermore, you may use Anaconda to deploy any required project with a few mouse clicks. This is why it is perfect for beginners who want to learn Python.

Now that you know what Anaconda Python is, let's look at how to install it.

**How to install Anaconda for Python?**



To install Anaconda, just head to the Anaconda Documentation website and follow the instructions to download the installer for your operating system. Once the installer successfully downloads, double-click on it to start the installation process.

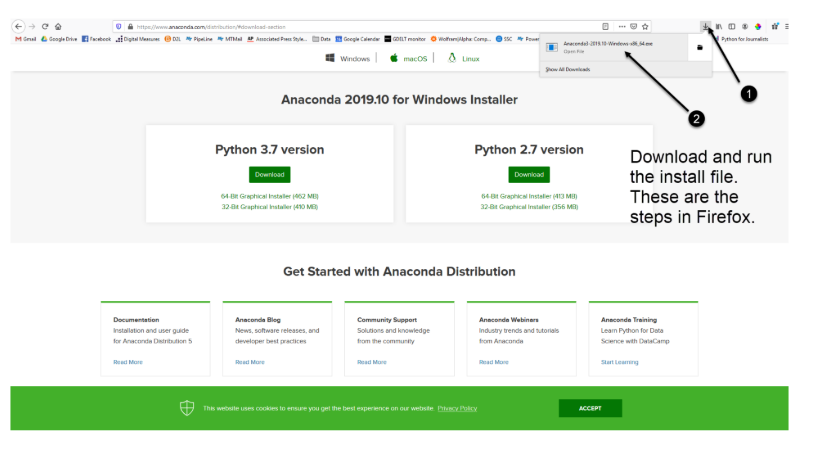
Follow the prompts and agree to the terms and conditions. When you are asked if you want to "add Anaconda to my PATH environment variable," make sure that you select "yes." This will ensure that Anaconda is added to your system's PATH, which is a list of directories that your operating system uses to find the files it needs.

Once the installation is complete, you will be asked if you want to "enable Anaconda as my default Python." We recommend selecting "yes" to use Anaconda as your default Python interpreter.

### **Python Anaconda Installation**

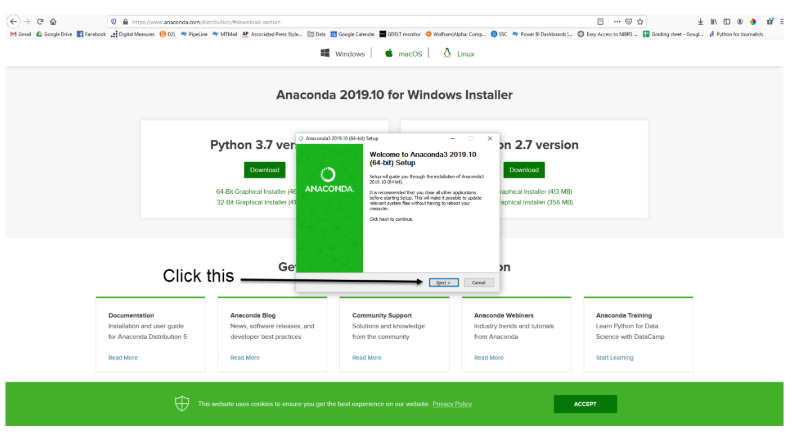
Next in the Python anaconda tutorial is its installation. The latest version of Anaconda at the time of writing is 2019.10. Follow these steps to download and install Anaconda on your machine:

1. Go to this link and download Anaconda for Windows, Mac, or Linux: – [Download anaconda](https://www.anaconda.com/distribution/)

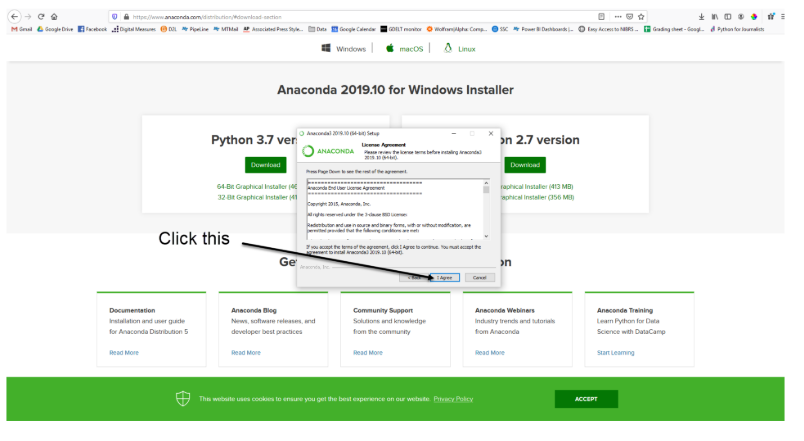


You can download the installer for Python 3.7 or for Python 2.7 (at the time of writing). And you can download it for a 32-bit or 64-bit machine.

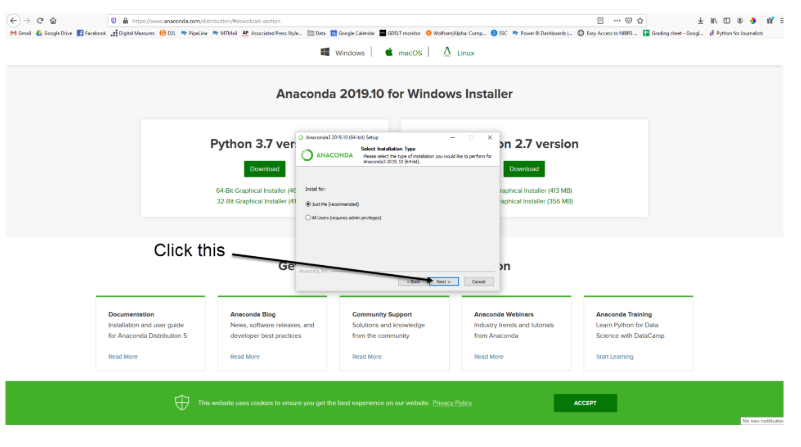
2. Click on the downloaded .exe to open it. This is the Anaconda setup. Click next.



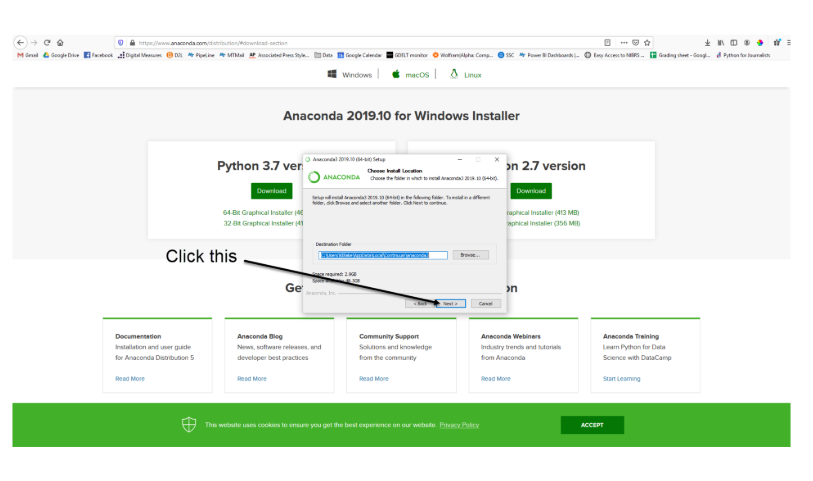
3. Now, you’ll see the license agreement. Click on ‘I Agree’.



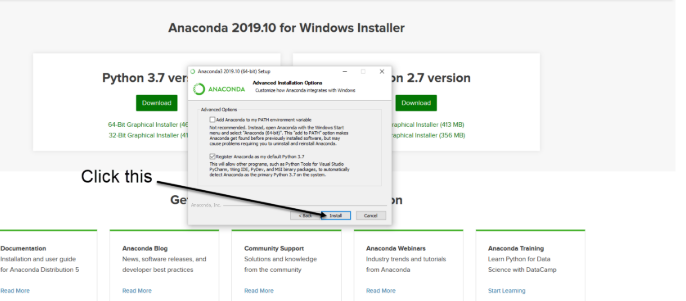
4. You can install it for all users or just for yourself. If you want to install it for all users, you need administrator privileges.



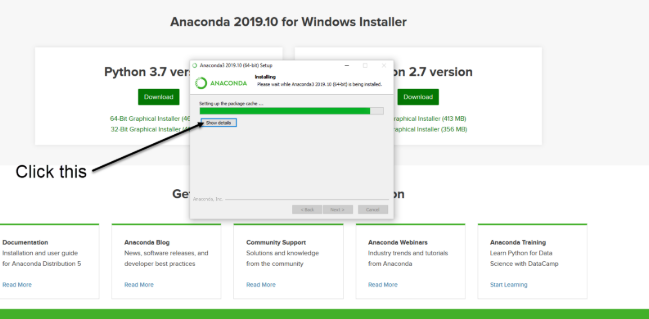
5. Choose where you want to install it. Here, you can see the available space and how much you need.



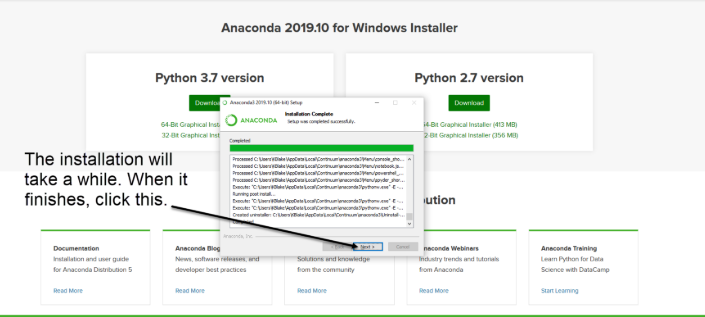
6. Now, you’ll get some advanced options. You can add Anaconda to your system’s PATH environment variable, and register it as the primary system Python 3.7. If you add it to PATH, it will be found before any other installation. Click on ‘Install’.



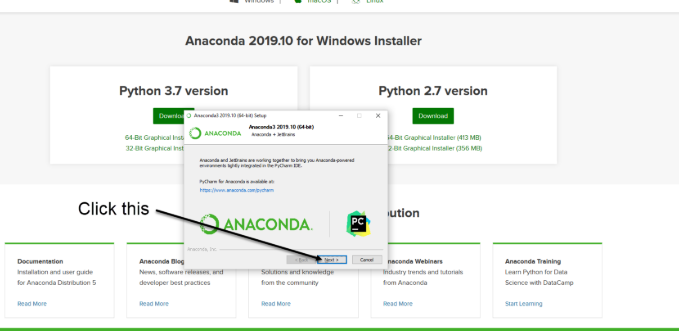
7. It will unpack some packages and extract some files on your machine. This will take a few minutes.



8. The installation is complete. Click Next.



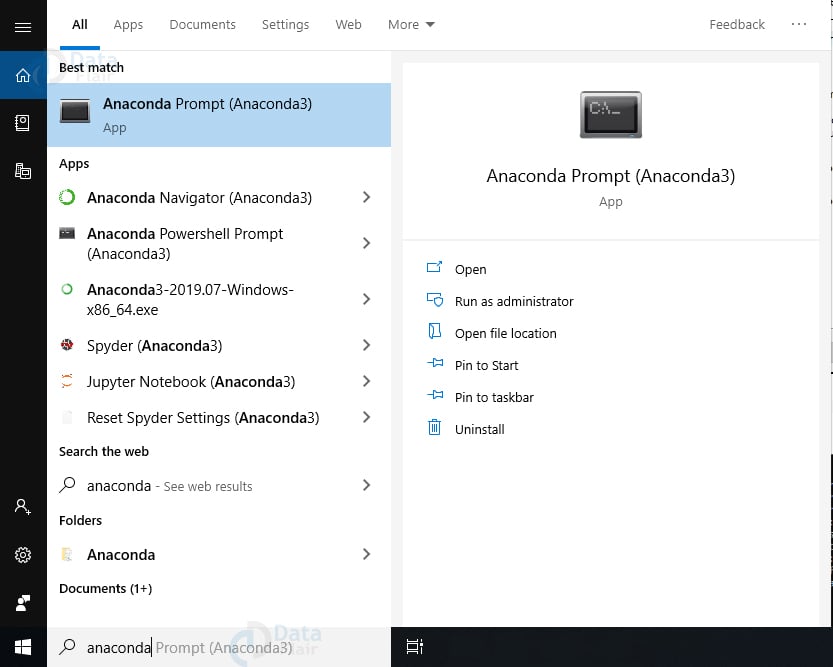
9. This screen will inform you about PyCharm. Click Next.



10. The installation is complete. You can choose to get more information about Anaconda cloud and how to get started with Anaconda. Click Finish.



11. If you search for Anaconda now, you will see the following options:



**PYTHON LANGUAGE:**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

## Features in Python:

There are many features in Python, some of which are discussed below as follows:

### **1. Free and Open Source**

[Python](https://www.geeksforgeeks.org/python-programming-language/)language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword. [Download Python](https://www.python.org/downloads/) Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

### **2. Easy to code**

Python is a [high-level programming language](https://www.geeksforgeeks.org/difference-between-high-level-and-low-level-languages/). Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

### **3. Easy to Read**

As you will see, learning Python is quite simple. As was already established, Python’s syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

### **4. Object-Oriented Language**

One of the key features of [Python is Object-Oriented programming](https://www.geeksforgeeks.org/python-oops-concepts/). Python supports object-oriented language and concepts of classes, object encapsulation, etc.

### **5. GUI Programming Support**

Graphical User interfaces can be made using a module such as [PyQt5](https://www.geeksforgeeks.org/pyqt5-qaction/), PyQt4, wxPython, or [Tk in python](https://www.geeksforgeeks.org/python-gui-tkinter/). PyQt5 is the most popular option for creating graphical apps with Python.

### **6. High-Level Language**

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

### **7. Extensible feature**

Python is an **Extensible** language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

### **8. Easy to Debug**

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program’s issues once you understand how to [interpret](https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/)Python’s error traces. Simply by glancing at the code, you can determine what it is designed to perform.

### **9. Python is a Portable language**

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as [Linux](https://www.geeksforgeeks.org/introduction-to-linux-operating-system/), Unix, and Mac then we do not need to change it, we can run this code on any platform.

### **10. Python is an Integrated language**

Python is also an Integrated language because we can easily integrate Python with other languages like C, [C++](http://www.geeksforgeeks.org/c-plus-plus/), etc.

### **11. Interpreted Language:**

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, [Java](https://www.geeksforgeeks.org/java/), etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called **bytecode**.

### **12. Large Standard Library**

Python has a large [standard library](https://www.geeksforgeeks.org/libraries-in-python/) that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as [regular expression](https://www.geeksforgeeks.org/regular-expression-python-examples-set-1/)s, [unit-testing](https://www.geeksforgeeks.org/unit-testing-software-testing/), web browsers, etc.

### **13. Dynamically Typed Language**

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

### **14. Frontend and backend development**

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like javascript. Backend is the strong forte of Python it’s extensively used for this work cause of its frameworks like [Django](https://www.geeksforgeeks.org/django-tutorial/)and [Flask](https://www.geeksforgeeks.org/flask-creating-first-simple-application/).

### **15. Allocating Memory Dynamically**

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

**What is Machine Learning : -**

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain.Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

**Categories Of Machine Leaning :-**

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.

Supervised learning involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. We will see examples of both types of supervised learning in the following section.

Unsupervised learning involves modeling the features of a dataset without reference to any label, and is often described as "letting the dataset speak for itself." These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. We will see examples of both types of unsupervised learning in the following section.

## Need for Machine Learning

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven’t surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, “to make decisions, based on data, with efficiency and scale”.

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can’t do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

## Challenges in Machine Learning :-

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

**Quality of data** − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

**Time-Consuming task** − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

**Lack of specialist persons** − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

**No clear objective for formulating business problems** − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

**Issue of overfitting & underfitting** − If the model is overfitting or underfitting, it cannot be represented well for the problem.

**Curse of dimensionality** − Another challenge ML model faces is too many features of data points. This can be a real hindrance.

**Difficulty in deployment** − Complexity of the ML model makes it quite difficult to be deployed in real life.

**LIBRARIES/PACKGES :-**

**Tensorflow**

TensorFlow is a [free](https://en.wikipedia.org/wiki/Free_software) and [open-source](https://en.wikipedia.org/wiki/Open-source_software) [software library for dataflow and differentiable programming](https://en.wikipedia.org/wiki/Library_(computing)) across a range of tasks. It is a symbolic math library, and is also used for [machine learning](https://en.wikipedia.org/wiki/Machine_learning) applications such as [neural networks](https://en.wikipedia.org/wiki/Neural_networks). It is used for both research and production at [Google](https://en.wikipedia.org/wiki/Google).‍

TensorFlow was developed by the [Google Brain](https://en.wikipedia.org/wiki/Google_Brain) team for internal Google use. It was released under the [Apache 2.0](https://en.wikipedia.org/wiki/Apache_License) [open-source license](https://en.wikipedia.org/wiki/Open-source_license) on November 9, 2015.

**Numpy**

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

**Pandas**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**Matplotlib**

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and [IPython](http://ipython.org/) shells, the [Jupyter](http://jupyter.org/) Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the [sample plots](https://matplotlib.org/tutorials/introductory/sample_plots.html) and [thumbnail gallery](https://matplotlib.org/gallery/index.html).

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

**Scikit – learn**

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

**SYSTEM TESTING**

**8. SYSTEM TESTING**

System testing, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application. System testing verifies that an application performs tasks as designed. This step, a kind of black box testing, focuses on the functionality of an application. System testing, for example, might check that every kind of user input produces the intended output across the application.

**Phases of system testing:**

A video tutorial about this test level. System testing examines every component of an application to make sure that they work as a complete and unified whole. A QA team typically conducts system testing after it checks individual modules with functional or user-story testing and then each component through integration testing.

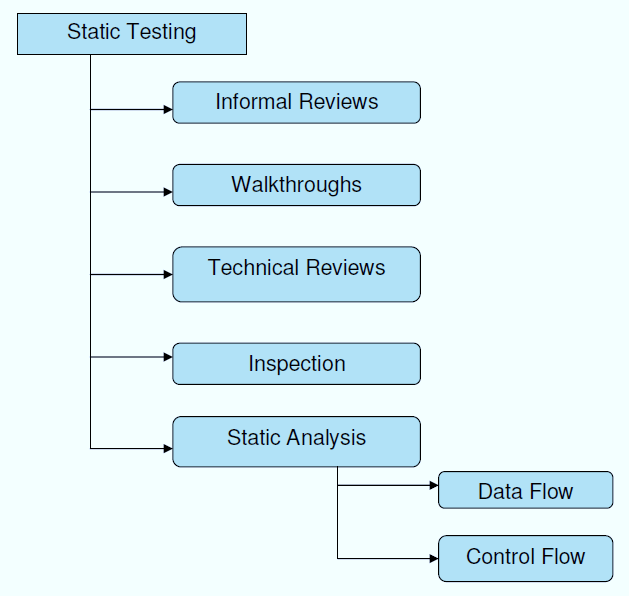
If a software build achieves the desired results in system testing, it gets a final check via acceptance testing before it goes to production, where users consume the software. An app-dev team logs all defects, and establishes what kinds and amount of defects are tolerable.

**8.1Software Testing Strategies:**

Optimization of the approach to testing in software engineering is the best way to make it effective. A software testing strategy defines what, when, and how to do whatever is necessary to make an end-product of high quality. Usually, the following software testing strategies and their combinations are used to achieve this major objective:

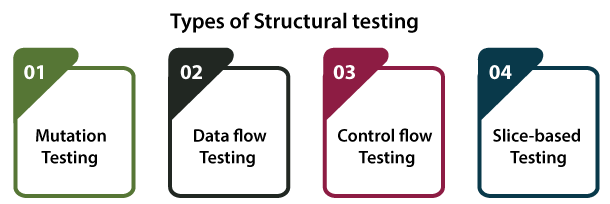
Static Testing:

The early-stage testing strategy is static testing: it is performed without actually running the developing product. Basically, such desk-checking is required to detect bugs and issues that are present in the code itself. Such a check-up is important at the pre-deployment stage as it helps avoid problems caused by errors in the code and software structure deficits.



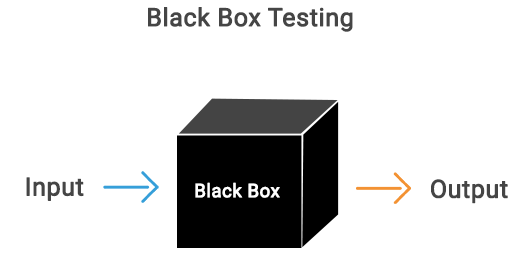
**Structural Testing:**

It is not possible to effectively test software without running it. Structural testing, also known as white-box testing, is required to detect and fix bugs and errors emerging during the pre-production stage of the software development process. At this stage, unit testing based on the software structure is performed using regression testing. In most cases, it is an automated process working within the test automation framework to speed up the development process at this stage. Developers and QA engineers have full access to the software’s structure and data flows (data flows testing), so they could track any changes (mutation testing) in the system’s behavior by comparing the tests’ outcomes with the results of previous iterations (control flow testing).



**Behavioral Testing:**

The final stage of testing focuses on the software’s reactions to various activities rather than on the mechanisms behind these reactions. In other words, behavioral testing, also known as black-box testing, presupposes running numerous tests, mostly manual, to see the product from the user’s point of view. QA engineers usually have some specific information about a business or other purposes of the software (‘the black box’) to run usability tests, for example, and react to bugs as regular users of the product will do. Behavioral testing also may include automation (regression tests) to eliminate human error if repetitive activities are required. For example, you may need to fill 100 registration forms on the website to see how the product copes with such an activity, so the automation of this test is preferable.



**8.2 TEST CASES:**

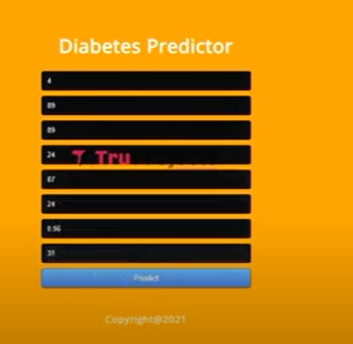
|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **INPUT** | **If available** | **If not available** |
| 1 | User signup | User get registered into the application | There is no process |
| 2 | User signin | User get login into the application | There is no process |
| 3 | Enter input for prediction | Prediction result displayed | There is no process |

**SCREENS**

1. **SCREENSHOTS**

SCREENS:







**CONCLUSION**

**10. CONCLUSION**

In conclusion, this study explored the efficacy of five supervised machine learning algorithms—Naïve Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest, and K-Nearest Neighbors (KNN)—for the prediction of diabetes disease. Through comprehensive analysis and evaluation, we observed varying levels of accuracy and computational efficiency among these algorithms. Notably, KNN exhibited the highest stability and accuracy of 76%, followed closely by other classifiers surpassing 70%. By incorporating all relevant risk factors and employing cross-validation techniques, we ensured robust model performance and minimized overfitting or underfitting issues.

Our findings suggest that machine learning algorithms offer promising avenues for early detection and proactive management of diabetes, thereby potentially reducing healthcare burdens and improving patient outcomes. Each algorithm showcased unique strengths and limitations, highlighting the importance of selecting the most appropriate algorithm based on the specific characteristics of the dataset and the desired computational resources.

Future research directions may include refining feature selection techniques, exploring ensemble methods combining multiple algorithms, and investigating the generalizability of the models across diverse demographic and clinical settings. Overall, this study contributes valuable insights to the field of healthcare informatics, paving the way for enhanced diagnostic capabilities and personalized treatment strategies for diabetes and other complex diseases.

**BIBILOGRAPHY**

**11. REFERENCES**

[1] P. Saeedi, I. Petersohn, P. Salpea, B. Malanda, S. Karuranga, N. Unwin, S. Colagiuri, L. Guariguata, A. A. Motala, K. Ogurtsova, J. E. Shaw, D. Bright, and R. Williams, “Global and regional diabetes prevalence estimates for 2019 and projections for 2030 and 2045: Results from the international diabetes federation diabetes atlas, 9th edition,” Diabetes Research and Clinical Practice, vol. 157, p. 107843, 2019.

[2] A. Mir and S. N. Dhage, “Diabetes disease prediction using machine learning on big data of healthcare,” in 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1–6.

[3] D. Sisodia and D. S. Sisodia, “Prediction of diabetes using classification algorithms,” Procedia Computer Science, vol. 132, pp. 1578 – 1585, 2018, international Conference on Computational Intelligence and Data Science. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1877050918308548>

[4] J. Smith, J. Everhart, W. Dickson, W. Knowler, and R. Johannes, “Using the adap learning algorithm to forcast the onset of diabetes mellitus,” Proceedings - Annual Symposium on Computer Applications in Medical Care, vol. 10, 11 1988.

[5] P. S. Kohli and S. Arora, “Application of machine learning in disease prediction,” in 2018 4th International Conference on Computing Communication and Automation (ICCCA), 2018, pp. 1–4.

[6] Wes McKinney, “Data Structures for Statistical Computing in Python,” in Proceedings of the 9th Python in Science Conference, Stefan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61. ´

[7] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R’ıo, M. Wiebe, P. Peterson, P. G’erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” Nature, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: <https://doi.org/10.1038/s41586-020-2649-2>

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Edouard Duchesnay, “Scikit-learn: Machine Learning in Python,” ´ Journal of Machine Learning Research, vol. 12, no. 85, p. 28252830, 2011.

[9] S. Yadav and S. Shukla, “Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification,” in 2016 IEEE 6th International Conference on Advanced Computing (IACC), 2016, pp. 78–83.