Department of Electrical and Computer Engineering
North South University

Senior Design Project

Cerebral Stroke Prediction Using Machine Learning Algorithms

Md. Asif Rahman **1821214042**

Faisal Bin Abdur Rahman **1912038042**

Anharul Islam **1912541042**

Ifrat Jahan **1812274042**

Faculty Advisor:

Dr. K. M. A. Salam

Professor

Department of Electrical and Computer Engineering

Spring, 2023

LETTER OF TRANSMITTAL

Spring, 2023

To

Dr. Rajesh Palit
Chairman,
Department of Electrical and Computer Engineering
North South University, Dhaka

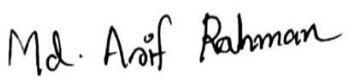
Subject: **Submission of Capstone Project Report on "Cerebral Stroke Prediction Using Machine Learning Algorithms"**

Dear Sir,

With due respect, we would like to submit our **Capstone Project Report** on **"Cerebral Stroke Prediction Using Machine Learning Algorithms"** as a part of our BSc program. The report examines the earlier prediction of cerebral stroke in patients. This project was very valuable to us as it helped us gain practical experience in the data science field that we could apply in real life. We made every effort to fulfil all the requirements of this report.

We will be highly obliged if you kindly receive this report and provide your valuable judgment. It would be our immense pleasure if you find this report useful and informative to have an apparent perspective on the issue.

Sincerely Yours,



.....

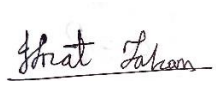
Md. Asif Rahman
ECE Department
North South University, Bangladesh

.....

Faisal Bin Abdur Rahman
ECE Department
North South University, Bangladesh

.....

Anharul Islam
ECE Department
North South University, Bangladesh



.....

Ifrat Jahan
ECE Department
North South University, Bangladesh

APPROVAL

We, Md. Asif Rahman **(ID - 1821214042)**, Faisal Bin Abdur Rahman **(ID – 1912038042),** Anharul Islam **(ID – 1821342642)** and Ifrat Jahan **(ID – 1813050042)** members of CSE: 499 (Senior Design) from the Electrical and Computer Engineering department of **North South University**; have worked on the project titled **"Cerebral Stroke Prediction Using Machine Learning Algorithms"** under the supervision of Dr. K. M. A. Salam as a partial fulfillment of the requirement for the degree of Bachelors of Science in Computer Science & Engineering and has been accepted as satisfactory.

Supervisor's Signature

.....

Dr. K. M. A. Salam
Professor

Department of Electrical & Computer Engineering

North South University

Dhaka, Bangladesh.

Chairman's Signature

.....

Dr. Rajesh Palit
Professor & Chairman

Department of Electrical & Computer Engineering

North South University

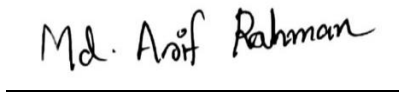
Dhaka, Bangladesh.

DECLARATION

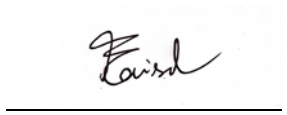
This is to certify that this Project is our original work. No part of this work has been submitted elsewhere partially or fully for the award of any other degree or diploma. Any material reproduced in this project has been properly acknowledged.

Students' names & Signatures

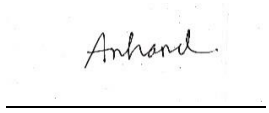
1. **Md. Asif Rahman**



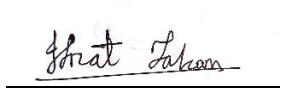
2. **Faisal Bin Abdur Rahman**



3. **Anharul Islam**



4. **Ifrat Jahan**



ACKNOWLEDGEMENT

By mercy of the Almighty Allah, we have completed our senior design capstone project entitled "Cerebral Stroke Prediction Using Machine Learning Algorithms".

Foremost, we would like to express our sincere gratitude to our advisor Dr. K. M. A. Salam for his continuous support in our capstone project progress throughout the whole 499A and 499B, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped us in all the time of research, writing and completing of this project.

Our sincere thanks also go to North South University, Dhaka, Bangladesh for providing an opportunity in our curriculum which enabled us to have an industrial level experience as part of our academics.

We are also very grateful to one of our dearest friends, Safiyatul Hoque, for his help in this project.

Last but not the least, we would like to thank our family as their inspiration and guidance kept us focused and motivated.

ABSTRACT

Cerebral stroke is on the rise, which may kill, disable, and destroy the brain. In this situation, it is important to predict a cerebral stroke early to prevent or lessen the damage caused by a stroke. A cerebral stroke occurs when brain tissue is deprived of oxygen and nutrients due to decreased or blocked blood flow. Currently, machine-learning-based systems are widely used as an effective method for predicting and reducing the potential damage of various diseases. The goal of this research is to find the early signs of a cerebral stroke so that people can take steps to stop more damage. Here, a dataset is used with 5026 points of data, 11 features about stroke, and the five best machine learning models trained for making predictions: decision tree, random forest, KNN, XGBoost, and a neural network model. Compared to the other machine learning models, the random forest and XGBoost models performed better. The accuracy of Random Forest was 97.11%, whereas that of XGBoost was 97%. Thus, the most accurate model, Random Forest, is used to forecast the chance of a stroke. A hosted web application and a mobile app are created to make the system accessible. By facilitating early prediction and intervention, this study can improve the medical system's capacity to prevent the damage of cerebral stroke.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Overview

1.1 Introduction

The brain is an intricate organ that governs life's many facets, such as cognition, memory, emotion, touch, motor control, vision, breathing, internal body temperature, hunger, and much more. A stroke, also known as a brain attack, occurs when an area of the brain loses oxygen and nutrients because the blood supply is cut off or when a cerebral vessel bursts. The brain is either old or injured in both cases. After the age of 25, one in four people will have a stroke [6]. Reducing the risk of stroke is possible by maintaining a healthy lifestyle and catching any problems early. For this, we must make an early prediction of the patient's cerebral stroke in order to receive an early diagnosis. Machine learning is the most effective method for predicting early cerebral strokes, which aids in the process.

1.2 Project Definition

This project is essential for precise and early cerebral stroke prediction based on various medical records. Several models are trained to predict cerebral strokes by analysing a validly gathered dataset. The most accurate model is used for stroke prediction. Based on that model, we have developed a web application that makes our system usable for the user, and a mobile application is also being developed to increase mobility.

1.3 Purpose of our Project / Motivation

Providing improved procedures and dependable evaluations is a significant problem in the modern medical field. Even though stroke has been highlighted as the primary global risk factor in recent days, it is among the diseases that can be treated and managed effectively. The entire efficacy of disease management is dependent on the precise timing of disease detection. The intended strategy aims to monitor these cerebral diseases at an early stage in order to mitigate their dire consequences. Health professionals have generated a vast quantity of clinical data

that is available for analysis and the extraction of useful insights. Data mining techniques are used to extract vital and concealed information from the vast quantity of data currently available. The majority of the clinical record consists of numerous individual materials. As a consequence, making decisions based on data samples is becoming increasingly difficult and complex. Machine learning (ML), a subfield of data mining, can efficiently manage large, well-structured datasets. In the healthcare industry, machine learning may be used to evaluate, identify, and predict severe maladies. The primary objective of this initiative is to equip clinicians with an early detection tool for cerebral stroke. As a result, clients will require more effective treatment while avoiding severe consequences. ML is extraordinarily effective at detecting discrete patterns and evaluating the provided template. As data is processed, ML methods aid in the diagnosis and treatment of cerebral stroke. In this study, we compare the performance of many machine learning techniques for the early detection of cerebral stroke. These techniques include the decision tree model, the random forest model, the KNN, the XGBoost classifier model, and the neural network model.

1.4 Project Goal

The goal of this paper is to use machine learning to identify cerebral stroke at an early stage, which may be useful in healthcare. As machine learning algorithms, this system includes a decision tree model, a random forest model, a KNN, an XGBosst classifier model, and a neural network to identify cerebral stroke based on ten independent features: gender, age, hypertension, heart disease, ever_married, work_type, residence_type, avg_glucose_level, and stroke. A web and mobile application for user usability is designed based on the good accuracy model that we get from the comparison. Based on these features, anyone may use this technique to determine whether or not a patient has suffered a brain stroke.

1.5 Summary

A cerebral stroke is the result of an interruption or reduction in the blood supply to the brain. This project helps people predict cerebral stroke at an early stage by analyzing various stroke-related characteristics to prevent catastrophic outcomes. Comparing various models using a

dataset of 5026 records and 11 attributes reveals that the Random Forest and XGBoost models are more accurate than other implemented models. The most precise random forest model is utilized to develop a web and mobile application.

1.6 Section Description

The report contains total 10 sections. In below there is a short description given of every section.

Section 1 gives an overview of this project and shows the purpose of this project as well.

Section 2 illustrates the existing systems similar to this project and gives an overview of why our project is adopted to fulfil the lacking and why it is better than that of other systems.

Section 3 describes technical features used in this project for implementation.

Section 4 illustrates the methods to implement the full system in an organized way.

Section 5 is the result and analysis part where the result of every model is demonstrated individually. Along with this, the final overview of the software is shown as well. The section also makes a discussion which model is best and why. Finally, there is an comparative analysis given as well to make an overview why our system is better than that of other system implemented.

Section 6 describes the novelty of our project.

Section 7 gives us information about the impact it will make toward different aspect.

Section 8 contains the conclusion and future implementation of this work.

Section 9 hold the bibliography we used though this report.

Section 10 is appendices part. This part contains the codes we used for this project.

2. Existing Systems and Solutions Adopted

2.1 Introduction

Cerebral stroke is a developing condition that usually causes agony and death in modern times. Numerous studies and machine learning algorithms have been widely used to anticipate and characterize this disease

2.2 Existing Solutions

In the research [1], a dataset with 4799 data points and 11 features was used to predict strokes using different machine-learning models, and the improvised random forest was found to have the highest accuracy of 96.7%.

The NIHSS dataset with 13 features was used in the study [2] to find and evaluate strokes in a group of older people over 65. For that, they trained several models. From there, the C4.5 decision tree method had even more incredible accuracy.

Another study [3] presents a prototype developed to categorize strokes using text-mining tools and machine-learning methods. Artificial neural networks trained with 507 observations using a stochastic gradient descent method were found to perform best.

The study [4] aimed to determine the prognosis of ischemic stroke using two ANN models with accuracy ranging from 79 to 95 percent.

According to the research paper [5], the authors developed a stroke prediction system using multiple machine learning algorithms. Among the models evaluated, the AdaBoost, XGBoost, and Random Forest Classifier algorithms achieved the highest accuracy scores, with 0.95, 0.96, and 0.97, respectively.

2.3 Proposed Solutions

This system detects strokes based on 10 independent features using decision tree, random forest, KNN, XGBoost, and neural network models as machine learning algorithms. The most precise model is used to develop a web and mobile application. Based on these features, any user can use this system to determine whether or not a patient is at risk for a cerebral stroke.

2.4 Solutions Adopted and Reasons

While previous studies achieved high accuracy using a dataset of 4,799 patients with 11 variables, there may still be room for improvement due to potential issues with dataset preprocessing or imperfect model tuning. However, for our study, we selected a dataset of 5026 with 11 features, making it more significant than the prior dataset. We also manage the imbalance problem, preprocess the data, and effectively tune the model. As a consequence, our dataset is more valuable than that of the prior study, and as a result, our model produces beneficial results. In addition, we have a web application that makes our system usable for the user, and a mobile application is also being developed to increase mobility. The most accurate model is used for stroke prediction. The medical field will significantly benefit from our established system's ability to identify strokes early and save many lives.

2.5 Summary

The focus of this paper is to detect cerebral stroke prediction based on machine learning that can be helpful in healthcare purposes adopting a better approach from the previously done works.

3. Technical Description

3.1 Machine Learning

Machine learning (ML) is the study of computer programs that apply statistical models and algorithms to learn via reasoning and connections without being explicitly programmed [6]. ML algorithms learn through experience and improve on their own. It discovers approaches, trains models, and automatically determines output using the learnt strategy [7]. Machine learning techniques can also adapt to changing environments.

3.1.1 Evolution of Machine Learning

Although Machine Learning has recently acquired popularity because to the exponential rate of data collection and technology improvements to enable it, its origins may be traced all the way back to the seventeenth century. Since the dawn of time, people have been seeking to make sense of data and analyze it in order to acquire immediate insights. Let's take a fascinating tour through the history of Machine Learning to see how it all started and how it arrived to where it is currently.

Blaise Pascal invented one of the first mechanical adding machines in 1642. It employed a gear and wheel mechanism similar to that seen in odometers and other counting devices. One would wonder what a mechanical adder has to do with Machine Learning, but if one look closely, they will notice that it was the first human effort to automate data processing. The next problem was to store data. The first application of data storage was in a weaving machine created in 1801 by Joseph Marie Jacquard, which employed metal cards pierced with holes to arrange threads. A set of these cards coded a program that controlled the loom. This enabled a procedure to be repeated with reliable results each time. Logic is a way of developing arguments or reasoning that leads to correct or incorrect conclusions. George Boole devised a method of encoding this in 1847 by utilizing Boolean operators (AND, OR, NOR) and having replies expressed by true or false, yes or no, and rendered in binary as 1 or 0. These operators are still used in web searches presently. Herman Hollerith developed the very first combined mechanical computation and punch card system to quickly calculate statistics received from

millions of individuals. The tabulating machine was an electromechanical device used to help summarize information contained on punched cards. The 1880 census in the United States took eight years to complete. Alan Turing, an English mathematician who pioneered artificial intelligence in the 1940s and 1950s, devised the "Turing Test" to assess if a computer had true intelligence. To pass the test, a computer must be capable of convincing a human that it is also human. Arthur Samuel created the first computer learning software in 1952.

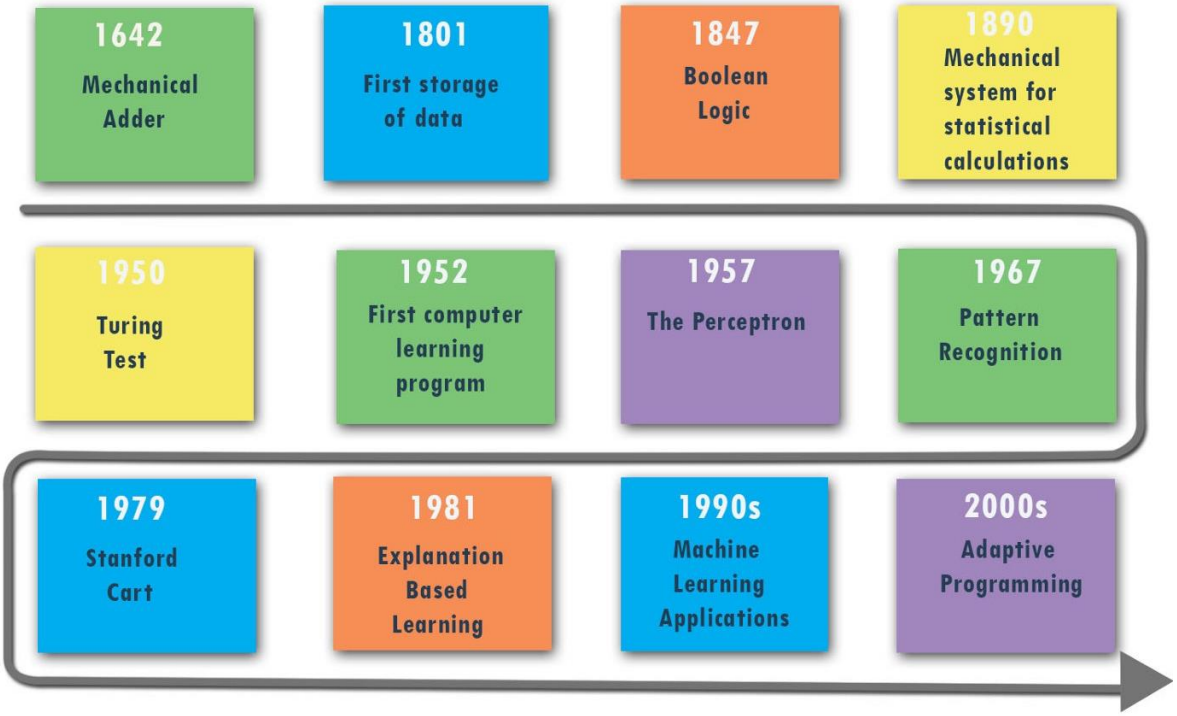


Figure 1. Evolution of machine learning

The software was a checkers game, and the IBM and computer got better at it the more they played it, understanding which movements made up winning tactics in a 'supervised learning mode' and incorporated those moves into their program. Frank Rosenblatt invented the perceptron, a form of neural network, in 1979. A neural network functions similarly to human brain; the brain has billions of neurons that are linked together in a network. The perceptron connects a network of decision points that work together in the broader program to tackle increasingly complicated issues. Gerald Dejong pioneered explanation-based learning (EBL) in a 1981 journal paper. EBL is a sort of supervised learning since past knowledge of the world is supplied through training examples. Given the aim to attain, the computer examines the training data and eliminates unnecessary information to produce a general rule to follow. Machine learning was first used in data mining, adaptive software and online applications, text learning, and language learning in the 1990s. Scientists start developing computer systems to

evaluate enormous volumes of data and derive inferences — or "learn" — from the results. Machine Learning got its name because advances in technology have made it feasible to design programs that, once developed, can continue to learn and improve as new data is given – with no human involvement necessary. The new century saw a surge in adaptive programming. Machine learning is available anywhere adaptive programs are required. These algorithms can recognize patterns, learn from experience, and continually improve themselves depending on feedback from the outside world. Deep learning is one type of adaptive programming, where computers can "see" and recognize things in photos and videos, and this was the primary technology underpinning Amazon GO shops, where shoppers are instantly billed as they walk out without having to wait in checkout lines. This is essentially the history of how Machine Learning evolved into what it is now. [8]

3.2 Models

A model is a machine learning method that has been taught to recognize particular sorts of structures using a machine learning algorithm [9]. That is, it analyses data and discovers hidden structures in a dataset. A dataset's existing answers and feature extraction establish the formula that depends on the input and output functionalities and uses it to fresh data to predict the response [10]. As a result, the model's algorithm takes a collection of data for training, creates a method to forecast the outcome, and saves that technique for future usage.

3.3 Libraries

Several libraries, including NumPy for array-related problems and mathematical computation, Pandas for data analysis and manipulation of tabular data frames, Matplotlib and Seaborn for data visualization and graphical plotting of the data, and Scikit-Learn for statistical modelling of the algorithms, are utilized here.

3.3.1 NumPy

NumPy is a widely used Python library for processing enormous multidimensional arrays and matrices using a vast collection of high-level mathematical functions. Machine learning uses it extensively for foundational scientific computations. It is especially helpful for linear algebra, the Fourier transform, and random number functions. Internally, advanced libraries such as TensorFlow use NumPy to manipulate tensors.

3.3.2 Scikit-learn

Scikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

3.3.3 Pandas

Pandas is an important Python data analysis library. It has no direct relationship with machine learning. As we know, dataset preparation is required prior to training. Pandas is useful in this situation because it was designed particularly for data extraction and preparation. It provides advanced data structures and a variety of data analysis tools. It provides numerous inherent grouping, combining, and filtering methods.

3.3.4 Matplotlib

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc.

3.3.5 Seaborn

Seaborn is a Matplotlib-based Python data visualization library. It offers a sophisticated interface for creating visually appealing and informative statistical graphics. Seaborn facilitates the creation of scatter plots, bar plots, heatmaps, and distribution plots by providing optimized defaults and user-friendly functions. It is useful for discovering and comprehending patterns and relationships in datasets, making it a popular tool for data analysis and scientific visualization.

3.3.6 Imblearn

Imblearn is a Python library designed to resolve the issue of imbalanced datasets in machine learning. It provides a set of tools and techniques for resampling data to address class imbalances. A class imbalance exists when the distribution of classes in a dataset is highly skewed, with one or more classes substantially underrepresented in comparison to others. Imblearn provides a variety of resampling techniques, such as oversampling, undersampling, and combination techniques, to balance class distribution. In addition, imblearn offers capabilities for data preprocessing, ensemble learning, and model evaluation that are unique to imbalanced datasets. It enables researchers and practitioners working on classification tasks with unbalanced data to construct more robust and accurate models.

3.3.7 Lime

LIME stands for "Local Interpretable Model-Agnostic Explanations." It's a popular Python library used for explaining the predictions of machine learning models. It creates simple, interpretable models that approximate the behavior of complex models for specific instances, helping users understand why a model made a particular prediction.

4. Models and Software Implementation

This part discusses all methodologies and materials, the dataset features summary, block diagrams, and applied models to the system.

4.1 Block Diagram

Fig. 2 depicts our system's process. The system's goal is to anticipate or foresee an outcome in advance, so it must use a dataset of historical data. A data visualization determines whether preprocessing is required and whether the dataset requires class imbalance handling. Before use and display, the dataset must undergo preprocessing, which includes handling class imbalance, removing null and duplicate values, and label encoding string values. When the preprocessing is complete, the data must be divided into two portions: data used to train the model and data used to test and assess the model. 25% of the data will be used for model testing, while the remaining 75% will be used for training. The chosen model will be trained and tested using the training mentioned above and testing data, resulting in an accuracy rating, precision, recall, f1-score, and Brier score (a lower Brier score suggests more accurate forecasts and a higher Brier score indicates less accurate forecasts; it ranges from 0 to 1). This will all be evaluated to select the most effective prediction model. The technology will then be able to forecast the desired outcome accurately.

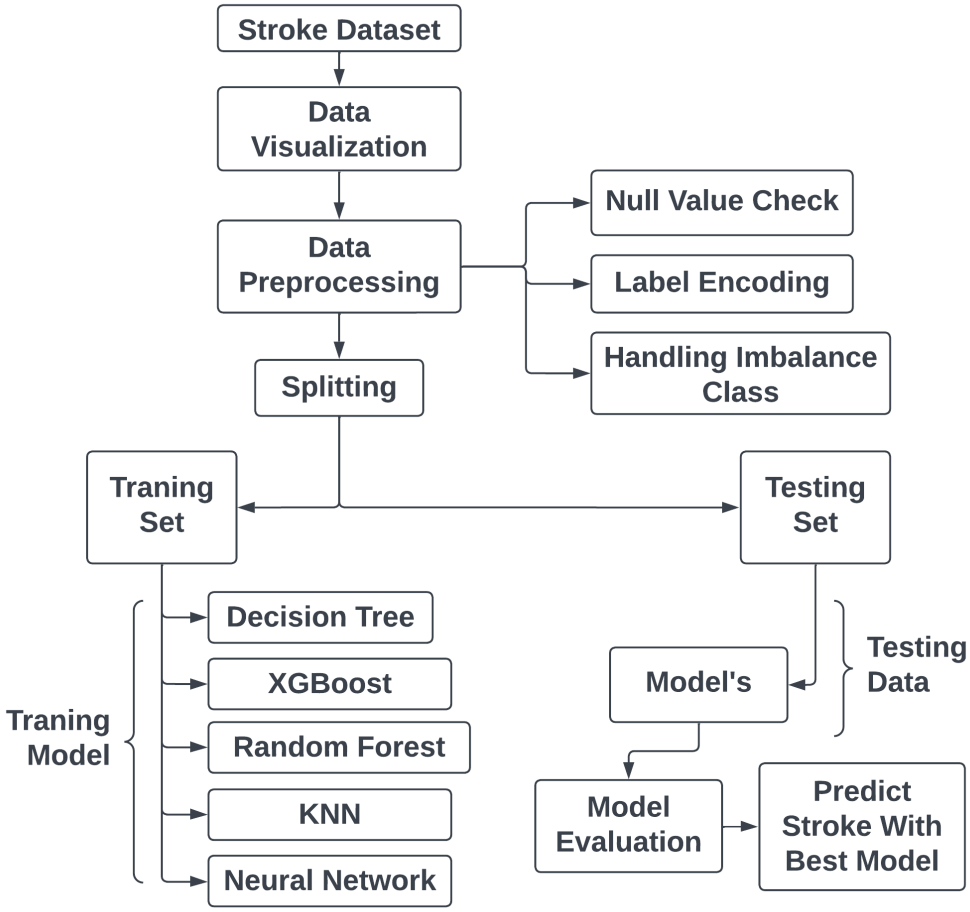


Figure 2. Block diagram of the proposed system

4.2 Dataset

Machine learning is a technique of AI that uses previous datasets to predict or classify something. Our machine-learning system used a dataset of cerebral strokes from Kaggle that consisted of 5026 samples, of which 2935 were female and 2091 were male, and 11 attributes related to cerebral stroke [11]. The attributes were gender, age, hypertension, heart disease, ever-married status, work type, residence type, average glucose level, BMI, smoking status, and stroke (0 or 1). The dataset has no null values; however, this dataset has a class imbalance issue, and some attributes require preprocessing before model train testing. Finding multiple observations in this dataset can help better comprehend the problem and detect whether a patient has a cerebral stroke based on the patient's provided characteristics. Fig. 3 depicts an overview of our original dataset.

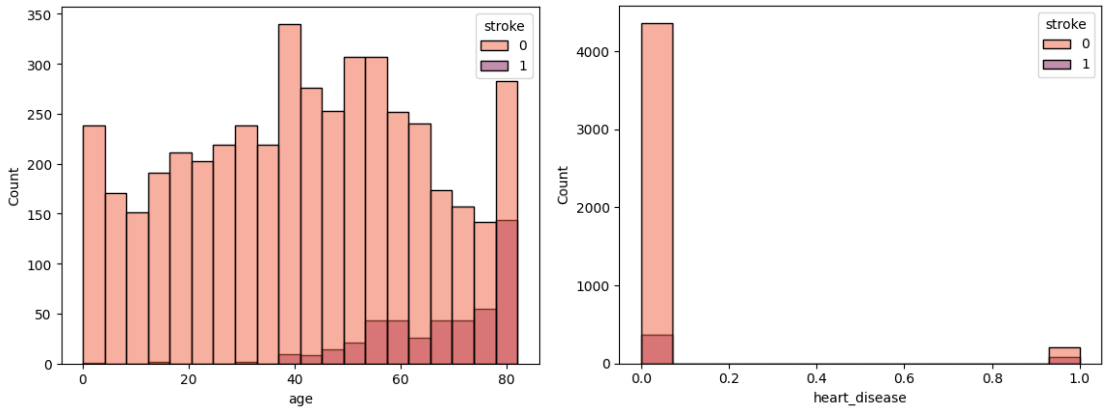
	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
2	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
3	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
4	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1

Figure 3. Dataset overview

4.3 Data Visualization

To comprehend and make sense of massive volumes of data, data visualization is a technique that uses a variety of static and dynamic visualizations within a given context. With the aid of data visualization, we can examine how the data appears, what type of association the qualities of the data have, and what kind of preprocessing it needs. It is the quickest technique to determine whether the characteristics match the output [12].

The histogram plot provides a count of the number of observations within each visualization bin. From the shape of the container, it is simple to determine whether the distribution is Gaussian, skewed, or exponential. Histograms also allow us to identify potential outliers. Fig. 4 shows the histogram plot of feature age, heart_disease, avg_glucose_level and bmi.



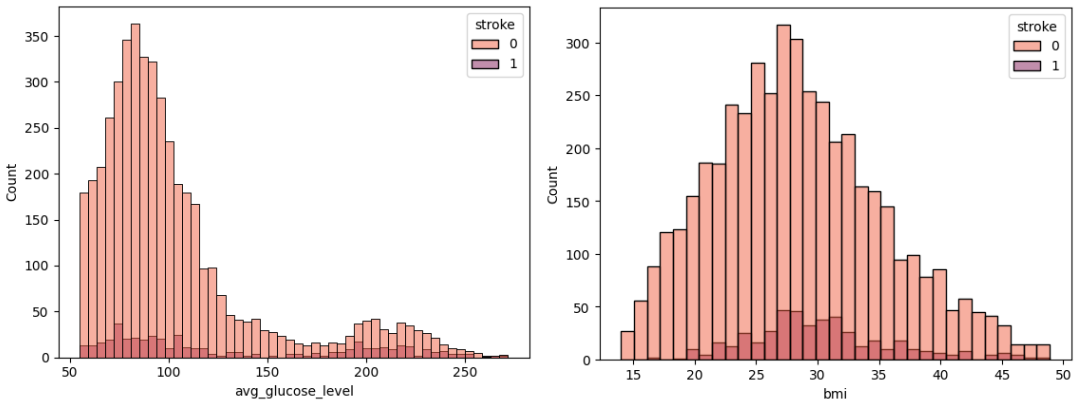


Figure 4. Histogram plot of age, heart_disease, avg_glucose_level, bmi

A KDE (Kernel Density Estimation) plot is a data visualization technique used in machine learning to represent the probability density function of a continuous variable. It provides a smoothed, non-parametric estimate of the underlying distribution of the data, making it easier to understand the shape and characteristics of the data's distribution. KDE plots are particularly useful for visualizing the distribution of features or model prediction residuals, aiding in data exploration, analysis, and model assessment. Fig. 5 shows the KDE plot of feature age, heart_disease, avg_glucose_level and bmi.

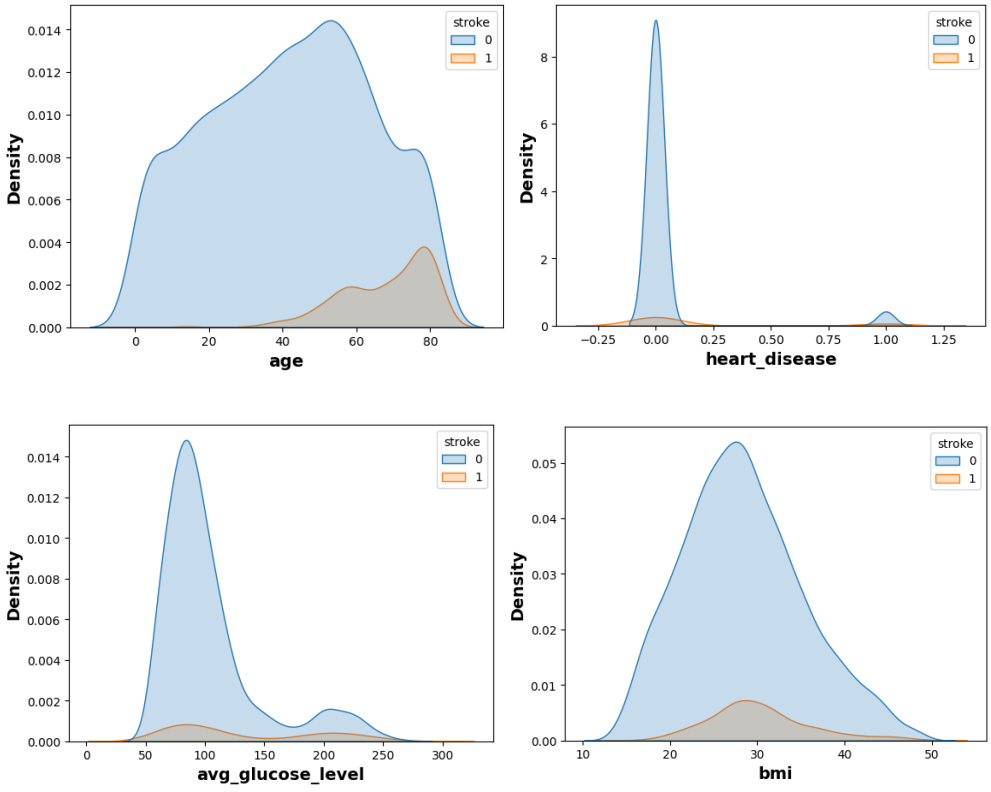


Figure 5. KDE plot of age, heart_disease, avg_glucose_level, bmi

4.4 Data Preprocessing

The workflow for machine learning requires data processing since the accuracy and performance of the model can be significantly impacted by the quality and format of the data. Therefore, data preparation is necessary to make it more suitable for mining and analysis, including redundant value reduction, feature selection, and data discretization [13]. The dataset we used for this study has 11 characteristics, and the "smoking status" column is removed since all research on this has concentrated on factors other than this. The dataset is next examined for null values; thankfully, there aren't any in it. Categorical variables in the dataset are converted to integers using label encoding, and the entire dataset is transformed into a collection of numbers. The dataset used for stroke prediction is imbalanced, with 5026 rows, 454 rows indicating the possibility of a stroke, and 4572 rows indicating the absence of a stroke. Unbalanced data may train a model with excellent accuracy, but precision, confusion matrices, and recall may need to be improved. The SMOTE algorithm is used to solve the class imbalance, which avoids producing duplicate data points in favor of producing artificial data points that are marginally different from the actual data points. The number of strokes before preprocessing is shown in Fig. 6, and the number of strokes after fixing the imbalance with SMOTE is shown in Fig. 7. Compared to undersampling, oversampling, and data augmentation techniques, the SMOTE algorithm performs significantly better.

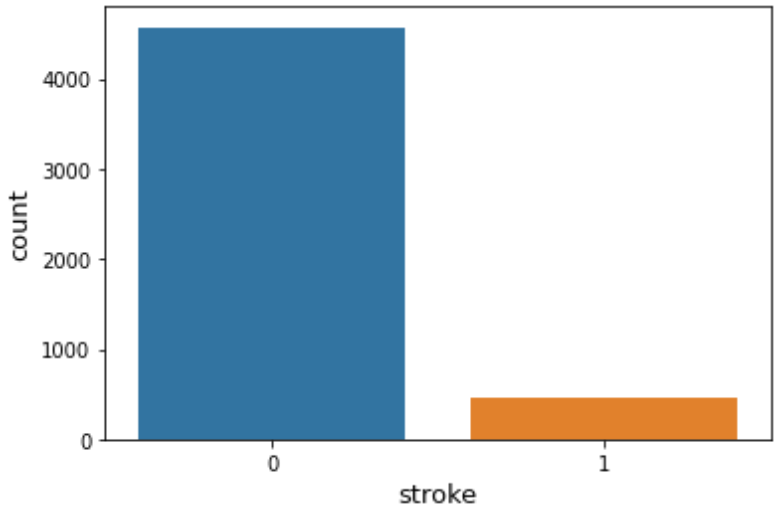


Figure 6. Stroke status before preprocessing.

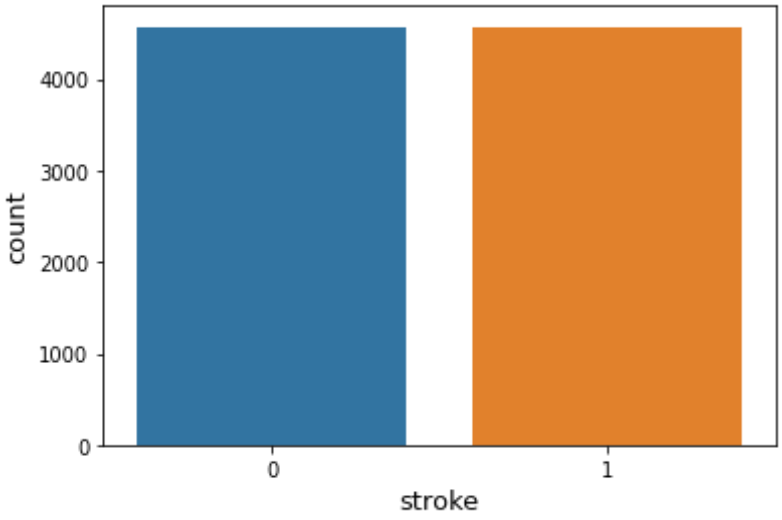


Figure 7. Stroke status after preprocessing.

4.5 Splitting Dataset

4.5.1 Splitting the Independent and Dependent Attribute

There are two components to splitting. First, we separated the dataset's independent columns (input features) and dependent columns (output features). The independent columns are then assigned to the X variable, which has nine input features, while the output column is assigned to the Y variable, which has only one output feature.

4.5.2 Splitting the Training and Testing Sample

Then, we split the total sample of our dataset for the training and testing purpose. With 75% of the data used for training and 25% used for testing that is, we divide our dataset into a training set and a test set.

```
X_train, X_test, Y_train, Y_test = train_test_split(X_balanced, Y_balanced, stratify = Y_balanced, test_size = 0.25, random_state = 49)
```

Figure 8. Train and test split

4.5.3 Feature Selection Technique

The variance threshold approach in Fig. 9 was used to discover any redundant or duplicate features, and we also utilized the Pearson correlation technique in Fig. 10 to determine the correlation in our dataset. These two strategies are termed feature selection techniques." By employing these two strategies, we found that we had no redundant or duplicate features.

```
var_thresh = VarianceThreshold()
var_thresh.fit(X_train)
var_thresh.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True])
```

```
[75] print("All features: ", X_train.columns)
     print("Features Selected: ", X_train.columns[var_thresh.get_support()]) # selects only the True columns
```

```
All features:  Index(['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'Residence_type', 'avg_glucose_level', 'bmi', 'work_type_Govt_job',
       'work_type_Private', 'work_type_Self-employed', 'work_type_children'],
      dtype='object')
Features Selected:  Index(['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'Residence_type', 'avg_glucose_level', 'bmi', 'work_type_Govt_job',
       'work_type_Private', 'work_type_Self-employed', 'work_type_children'],
      dtype='object')
```

Figure 9. Variance threshold approach

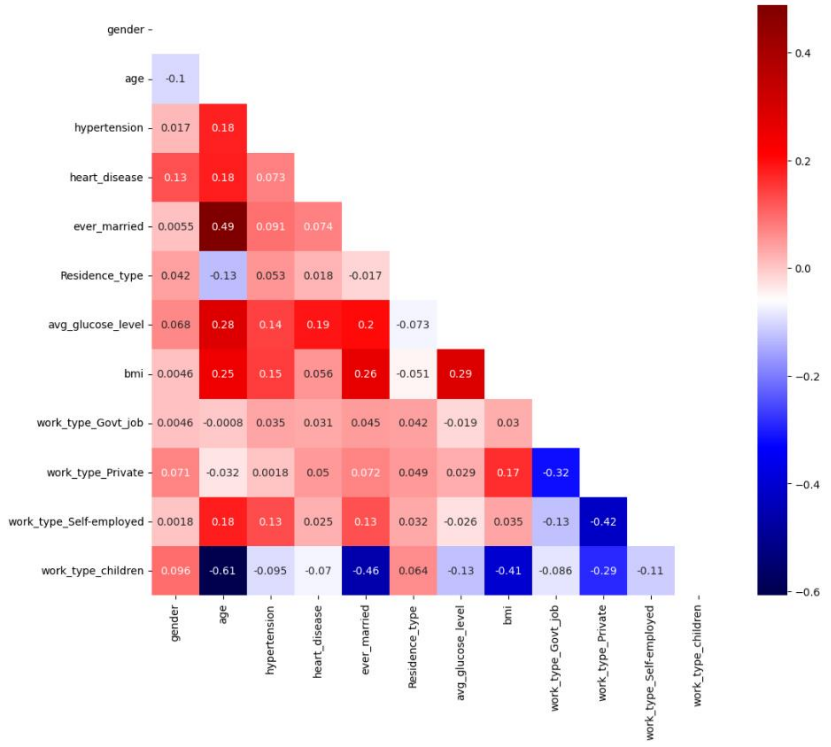


Figure 10. Pearson correlation of features.

4.5.4 Feature Importance

In machine learning, feature importance is a concept that helps us comprehend the relative significance or contribution of various features (input variables) in a dataset to predict the target variable. It is especially useful for obtaining insight into how the model makes predictions and which factors have the greatest impact on those predictions. Numerous machine learning algorithms, such as decision trees and random forests, explicitly output feature importance scores as part of their output. Typically, these scores are calculated based on how frequently a feature is used in the decision segments of the tree and how much its presence or absence impacts the outcome. Fig. 11 shows the importance of each feature of the cerebral stroke dataset for classification problems.

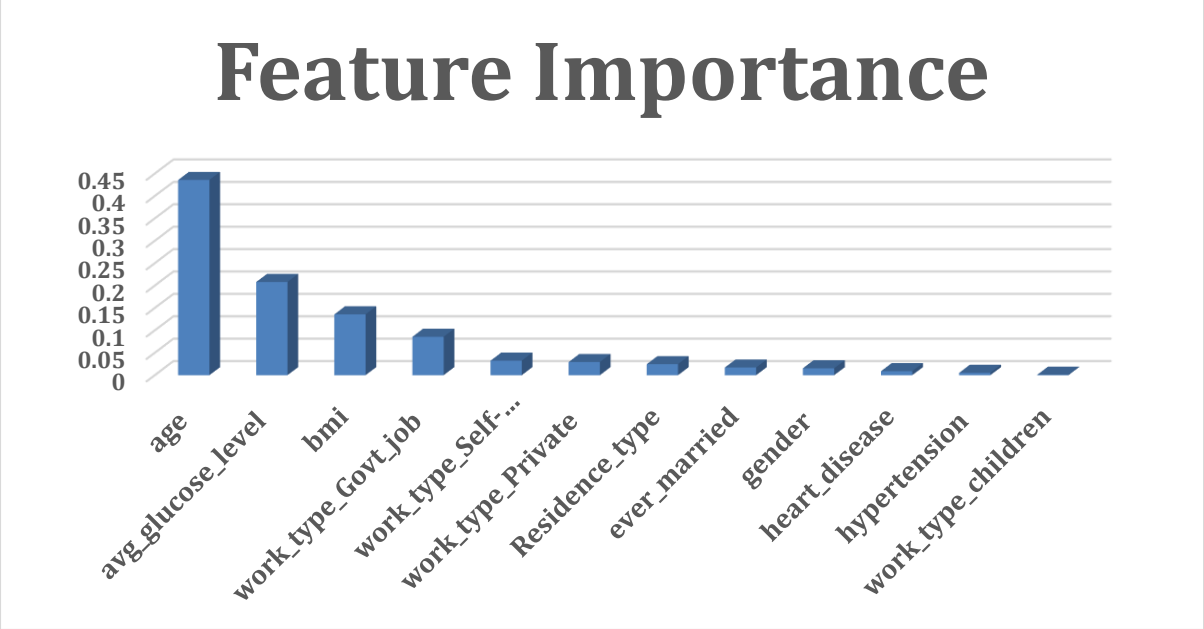


Figure 11. Feature importance of dataset.

4.6 Applied Model

4.6.1 Decision Tree Model

A decision tree is a powerful machine-learning method for classifying and predicting outcomes based on input features. It presents the possible outcomes and decision points in a tree-like

structure that is easy to understand and follow. It uses the training data to learn the connections between input features and output, then uses this knowledge to identify and give decisions on new data. However, it is important to properly adjust and trim the decision tree to avoid overfitting and ensure reliable performance. Fig. 12 shows the general structure of a decision tree, with the root node representing a question and the branches coming up with yes and no as possible answers.

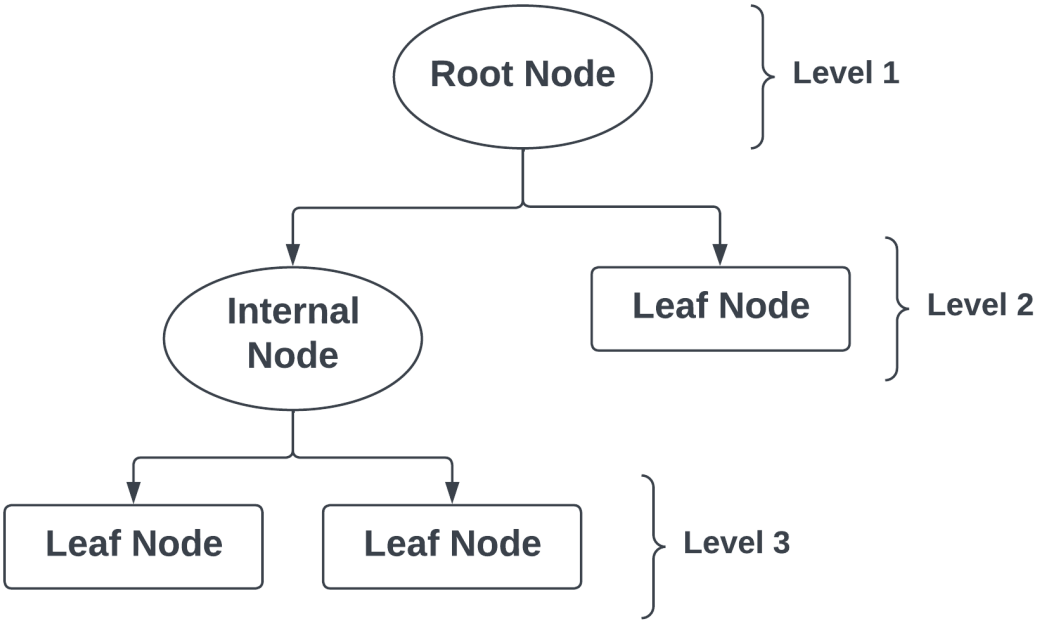


Figure 12. General structure of the decision tree

4.6.2 Random Forest Model

The popular machine learning technique Random Forest uses a forest of decision trees to produce a more robust and accurate model. The Random Forest method involves a forest of decision trees, each of which is trained on a different subset of the data and then votes by the majority's forecast. The random forest method has become commonplace because of its effectiveness in dealing with noisy and high-dimensional data and its simplicity in application and analysis. Random Forest can deal with numerical and categorical data, and it overfits less than separate decision trees. Fig. 13 shows the general structure of a random forest, that has multiple decision trees.

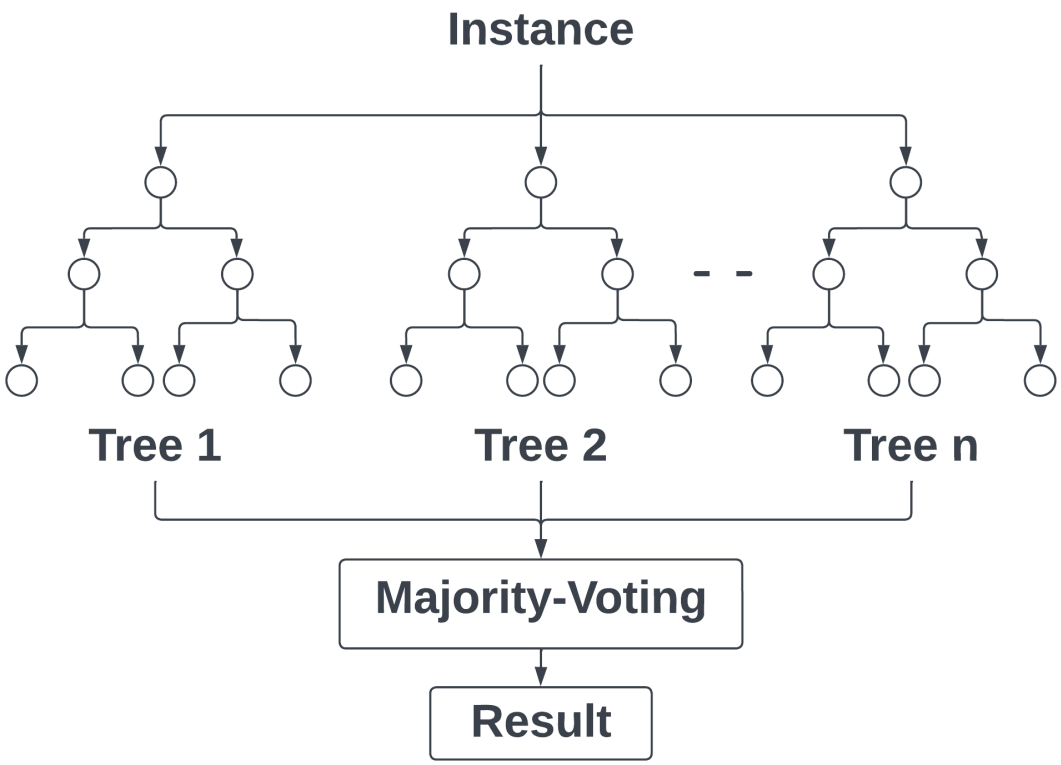


Figure 13. General structure of random forest

4.6.3 XGBoost

XGBoost implements a gradient-boosted tree. Gradient boosting uses weaker models to predict a target variable in supervised learning. Fig. 14 shows that XGBoost minimizes a regularized objective function that combines a convex loss function and a penalty term for model complexity. New trees are added iteratively to anticipate the residuals or mistakes of earlier trees. The final prediction is generated by adding the output from each tree multiplied by a learning rate to the starting prediction. XGBoost's capacity to handle missing data and big datasets effectively is a significant benefit. This approach uses multiple adjustable hyperparameters to improve the model's performance.

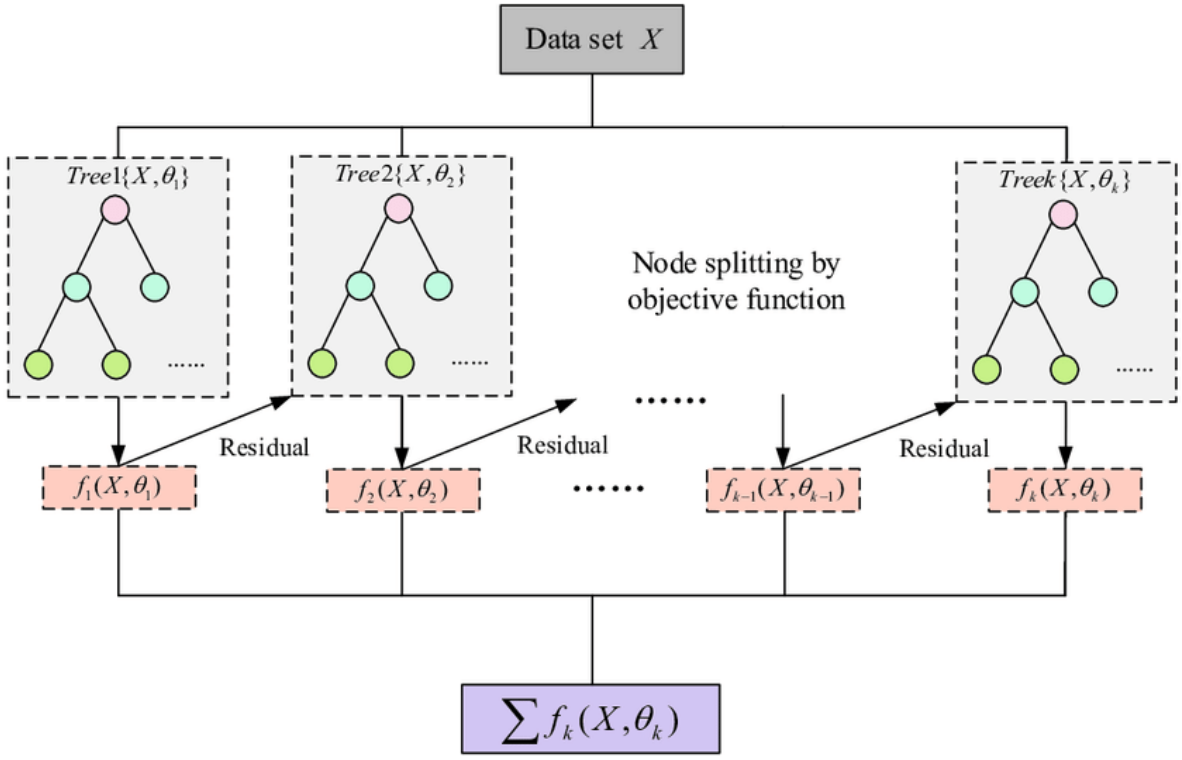


Figure 14. General structure of XGBoost [14]

4.6.4 KNN

K-nearest neighbors are a non-parametric supervised learning approach [15]. KNN is considered a lazy learner since it predicts outcomes without training the model by simply measuring distance. By calculating the distances between a query and each example in the data, choosing the K instances (the needed number) that are the closest to the query, and then either voting for the most frequent label or averaging the labels, Fig. 15 demonstrates how KNN works. KNN has the benefit of being able to intuitively handle multi-class scenarios and perform well given sufficient representative data.

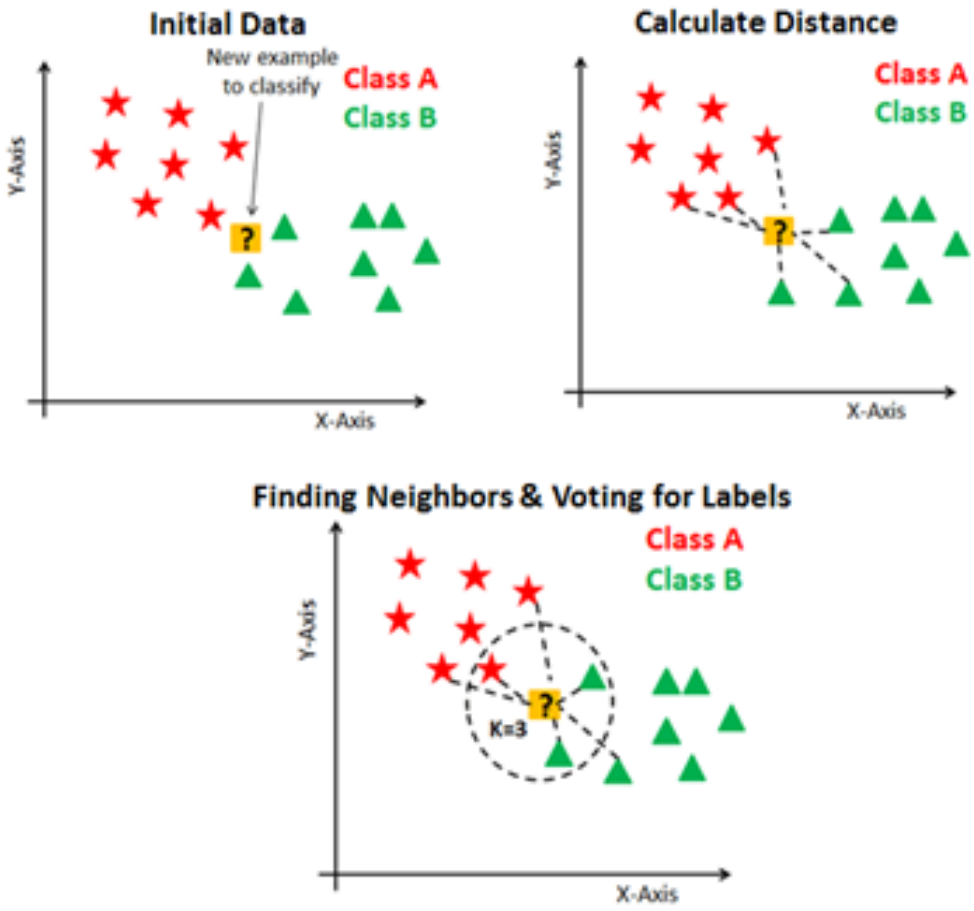


Figure 15. Prediction process of KNN [16]

4.6.5 Neural Network

A neural network is a type of artificial intelligence used to analyze complex data. It is typically a feedforward, in which information is continuously sent from one layer to the next without ever being re-transmitted. It comprises multiple layers of interconnected neurons, as shown in Fig. 16, which process and transmit information. Nodes are organized into layers, with some having connections and labels and others not. Between the input and the output, the system must process layers of data to reach the outcome.

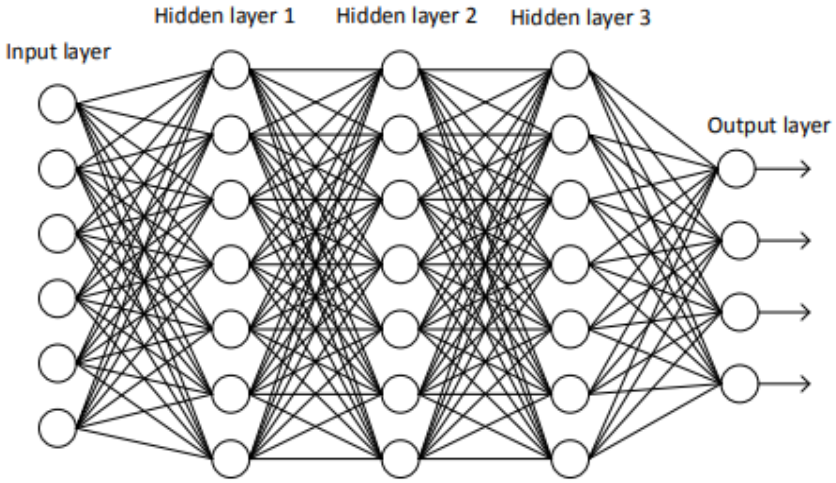


Figure 16. General structure of a neural network [17]

4.7 Libraries

Libraries are essential to the development of our system. From dataset upload to model implementation and evaluation, we used a variety of packages. We utilized the Pandas library to convert the dataset into a Pandas data frame and conduct data analysis. A variety of high-performance data structures and data analysis tools are provided by it. The dataset splitting, confusion matrices, and accuracy score implementations were done using the Scikit library. For visualization, techniques like graphical charting were performed using the Seaborn package. As a result, libraries are an excellent tool for adequately implementing our project and helping us create our system.

4.8 Pickle

Binary protocols can be used with the Pickle module to serialize and deserialize a Python object structure. Pickling converts a Python object hierarchy into a byte stream, and unpickling is the opposite [18]. Pickle is a useful Python utility that lets you share, commit, and re-load pre-trained machine learning models while minimizing time-consuming re-training. It is usually used to store or export the ML model for future use.

4.9 Render

Render is a cloud computing platform that provides hosting and deployment services for websites, web applications, and APIs. It allows developers to quickly deploy and manage their applications on the cloud without worrying about infrastructure management.

4.10 Web Application based on Machine Learning

A cerebral stroke prediction web application is used to tell if a patient has a cerebral stroke. The flowchart of a hosted web application for cerebral stroke prediction is shown in Fig. 17. The machine-learning model is trained using the necessary data set, and the optimal accuracy model is selected. Pickle produces the model's Pickle file, Streamlit is used to construct a web application backend, and the backend and the Pickle file are uploaded to render for the web application's hosting environment. Finally, the hosted web application is built, and the user uses it to predict the result of a patient's stroke.

Figure 17. Block diagram of a web application

4.10.1 Streamlit

The open-source Python library Streamlit makes it easy to make and share beautiful, customized web apps for machine learning and data science [19]. You can create and launch practical data applications in a matter of minutes. Data scripts can be quickly converted into shared web applications using Streamlit. It is purely in Python. Prior front-end experience is optional. Here, we used it to create a web application.

4.11 Mobile Application based on Machine Learning

A smartphone application for stroke prediction has been created to assist users in predicting cerebral strokes earlier, more effectively, and with increased mobility. Fig. 18 depicts the mobile application's flowchart. Here also, it trains the model using the dataset, selects the optimal model for prediction, and exports it by Pickle. Flask is used to develop APIs, and this API gives output in JSON format. Postman is used to testing the functionality of an API. Then, the API and model Pickle file are submitted for hosting on the render cloud. Finally, Android Studio assists in developing an Android app and its user interface.

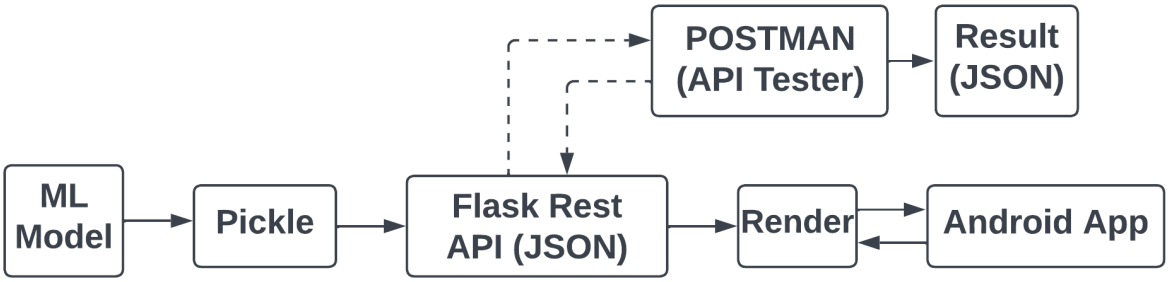


Figure 18. Block diagram of a mobile application

4.11.1 Flask

Flask is an open-source web framework. It is a popular microweb framework for Python API development. A framework with minimal dependence on other libraries is known as a micro-framework. With the capacity to scale up to big applications, Flask is a lightweight but robust web framework that can be set up quickly and simply [20]. Here, we used it to create API for our mobile application.

4.11.2 Postman

Postman is an API development environment for developing, testing, and iterating APIs. We used it to test our Flask REST API. To determine if our API is functioning correctly, we connect Postman to our API and then supply Postman with data to see whether our API can return a

response. If it functions properly, Postman will return the API's response in the necessary format.

4.11.3 Android Studio

Android Studio is the integrated development environment for developing Android applications. It is a Java-integrated software development environment. Here, we used it to develop the Android application and user interface.

4.12 Confusion Matrix

In machine learning, the efficacy of a classification model may be measured using a confusion matrix, a kind of table. It is a method used to determine how efficiently the model predicts the correct class for each sample in the test set. A confusion matrix contains information about the number of true positives, true negatives, false positives, and false negatives produced by a classification model. The matrix's rows represent the projected class, while its columns show the actual class. The matrix's cells represent the number of samples in each category. Fig. 19 shows the structure of the confusion matrix.

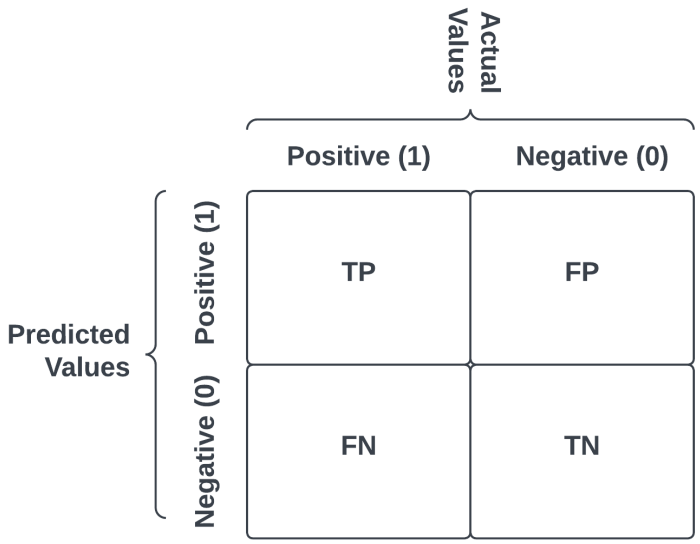


Figure 19. Confusion matrix diagram

5. Result & Analysis

Before accuracy verification, we split the dataset in our study into 75% for training and 25% for testing. The required models are then used for this project because it is a classification task. We can compare the accuracy of these five models in both training and testing.

5.1 Model-based analysis

5.1.1 Decision Tree

The decision tree categorizes an instance, which classifies instances by sorting them along the tree from the root to a leaf node [21]. The entropy impurity function is used to predict the result. This model's training and testing yield 98.86% for training and 94.66% for testing.

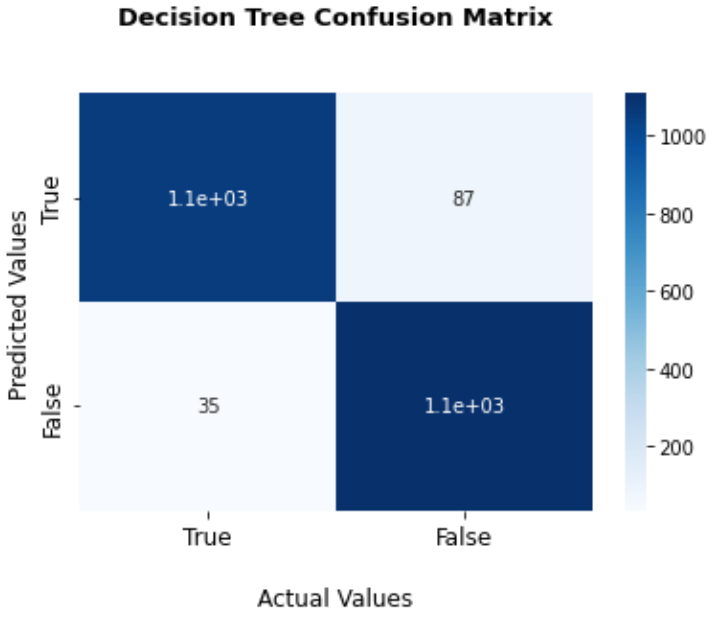


Figure 20. Confusion matrix of the decision tree

In the confusion matrix, 2164 data points are correctly predicted and 122 are incorrectly predicted. The f1-score, recall, and precision are respectively 94.78%, 96.93%, and 92.71%, with a Brier score of 0.05. Regarding the decision tree compared to other algorithms, it involves

less work for data preparation during pre-processing and does not require data scaling. As a result, it is preferable to utilize it in our system.

5.1.2 Random Forest

The random forest model comprises multiple decision trees. This model's training and testing yield 99.89% for training and 97.11% for testing. In the confusion matrix, 2220 data points are correctly predicted, while 66 are mis predicted.

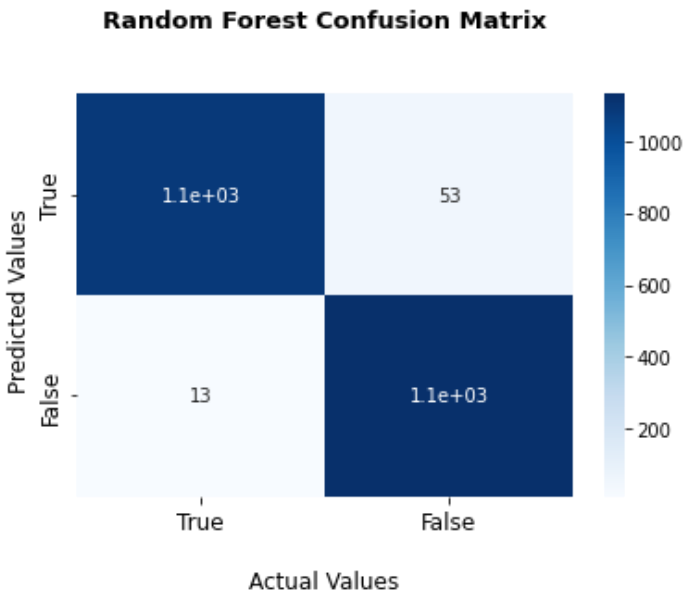


Figure 21. Confusion matrix of the random forest

The f1-score, recall, and precision are respectively 97.16%, 98.86%, and 95.51%, with a Brier score of 0.02. Random forest is a combination of many decision trees, involves less work for data preparation during pre-processing, does not require data scaling, and, if a decision tree has an overfitting problem, it can solve the overfitting problem.

5.1.3 KNN

KNN is a non-parametric, supervised learning classifier that employs closeness to predict or categorize individual groups. Here, we use K = 3 and Euclidian distance for prediction. This model's training and testing results are 95.39% for training and 92.08% for testing. With the

help of the confusion matrix, 2105 data points are predicted correctly and 181 are mispredicted. The f1-score, recall, and precision are respectively 92.52%, 97.98%, and 87.63%, with a Brier score of 0.07. KNN can perform effectively if sufficient representative data is provided.

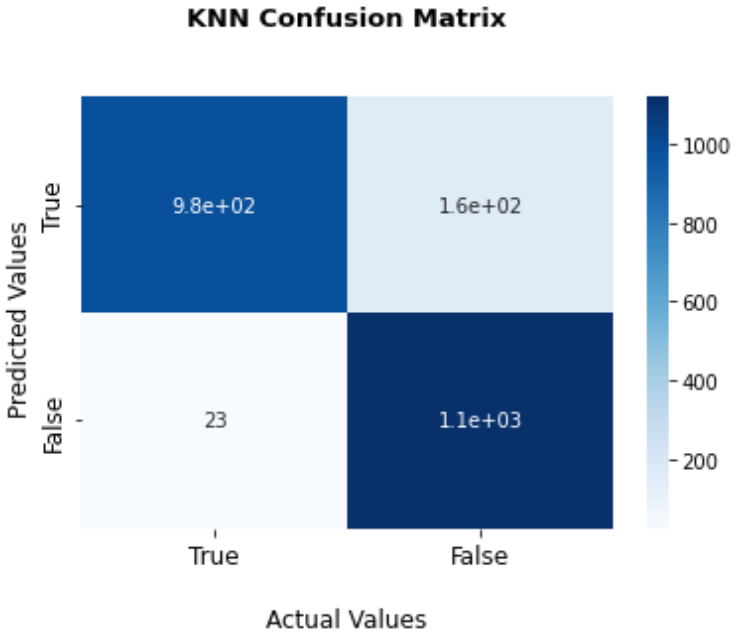


Figure 22. Confusion matrix of the KNN

5.1.4 XGBoost

XGBoost is a gradient-boosted tree algorithm that combines less accurate models to better predict a target variable in supervised learning. This model's training and testing outcomes are 99.84% for training and 97% for testing, respectively. Using the confusion matrix, 2206 data points are accurately predicted and 80 are incorrectly forecasted. The f1-score, recall, and precision are respectively 96.54%, 97.81%, and 95.34%, with a Brier score of 0.03. XGBoost offers several user-friendly capabilities, including parallelization, distributed computing, cache optimization, and more. So, it is prudent to include it in our system.

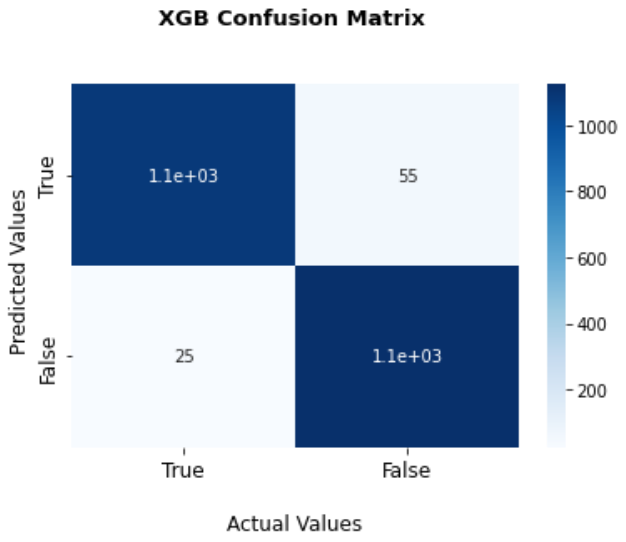


Figure 23. Confusion matrix of the XGBoost

5.1.5 Neural Network

In feedforward networks, data flows from the input to the output layers without looping back, as with neural networks. Our neural network comprises an input, an output, and six hidden layers. To train our data into an NN model, we employ 1105 epochs. Our NN model has an accuracy of 93.21%. Using the confusion matrix, 2131 data points are accurately predicted and 155 are incorrectly forecasted. The f1-score, recall, and precision are respectively 93.44%, 90.49%, and 96.58%, with a Brier score of 0.06.

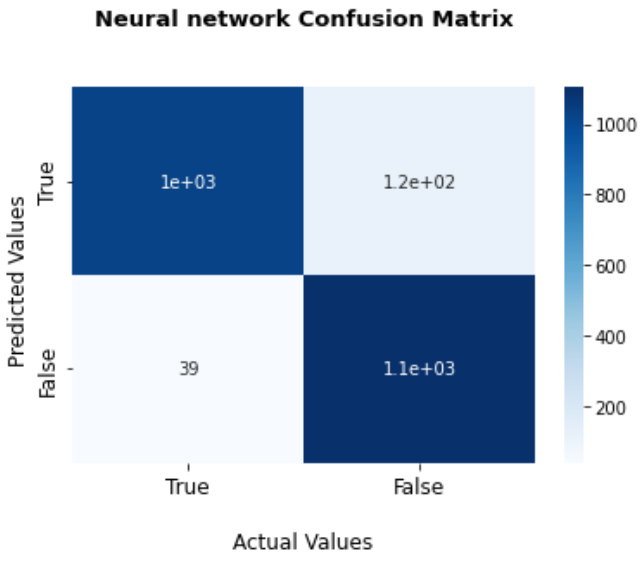


Figure 24. Confusion matrix of the neural network

5.2 Explainable AI

Explainable AI LIME is applied on the highest accuracy model random forest. LIME will clarify the decision or prediction made by the model. Fig. 25 illustrates the explanation of the random forest model decision or prediction of a sample using LIME.

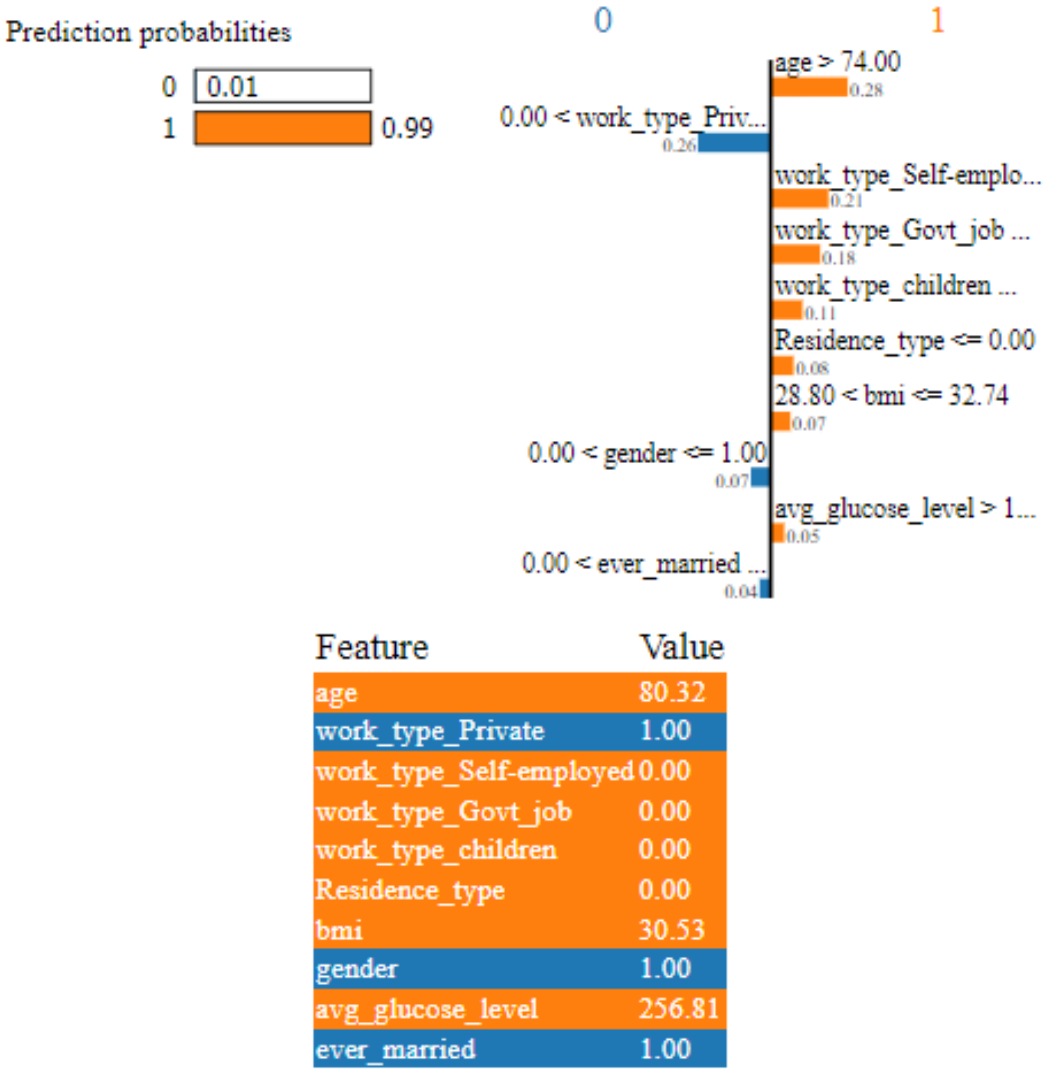


Figure 25. Explanation of model decision using LIME.

5.3 Web Application

A web application is being developed using the most accurate random forest model for our cerebral stroke prediction system. This web application's user interface is fundamental and straightforward. There are just a few radio buttons, select boxes, and input fields that the user

must complete. This web application is hosted on the render cloud. So, it boosts the system's portability. Here, the user just provides the data to our system's web application and clicks the Predict button; the result is then collected from the trained model through the render cloud and shown in the web application output box. It also has good error-handling capabilities. If a user enters incorrect information or leaves any input field blank and hits the Predict button, our system will immediately warn the user of the mistake. Fig. 26 illustrates the UI of web application.

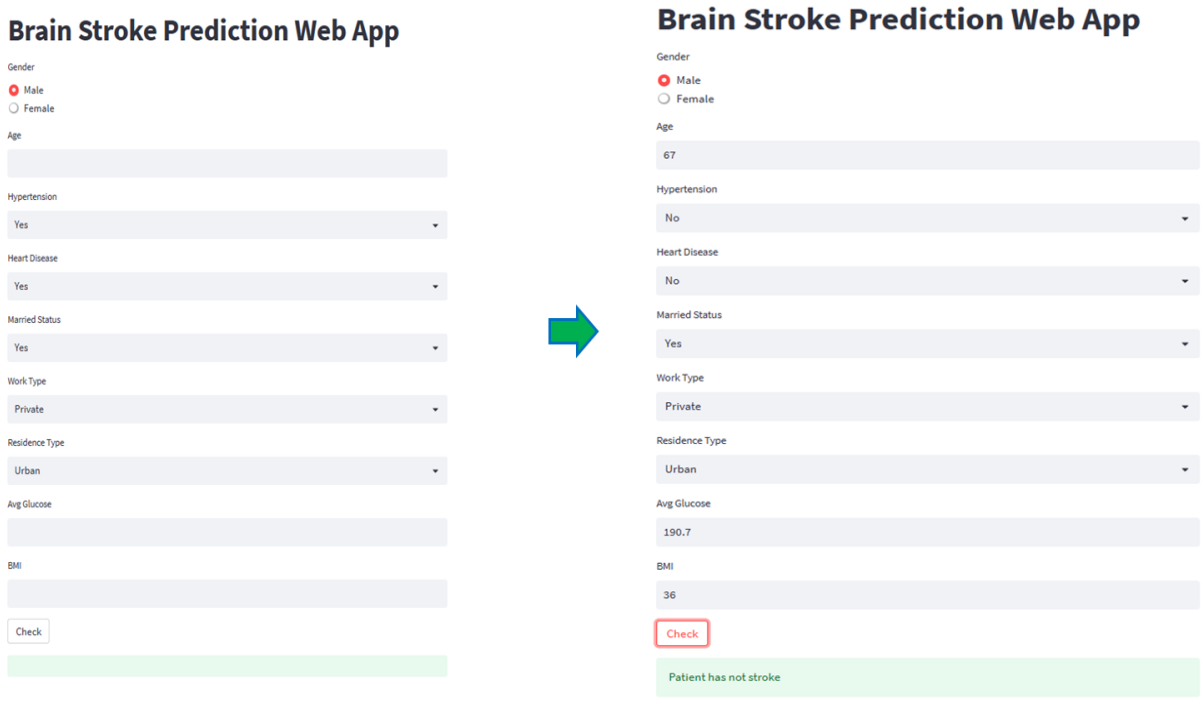


Figure 26. Web application

5.4 Mobile Application

We also developed a mobile application to boost the system's portability. Our Android application is likewise based on a random forest, the most precise model available. It also has a simple and user-friendly interface. The user will enter the relevant information in the input box and click the Predict button, and the result will be retrieved from the trained model through the render cloud and shown in the output box of the program. The interface and backend of the

mobile application are created in Android Studio using XML and Java. Fig. 27 illustrates the UI of mobile application.

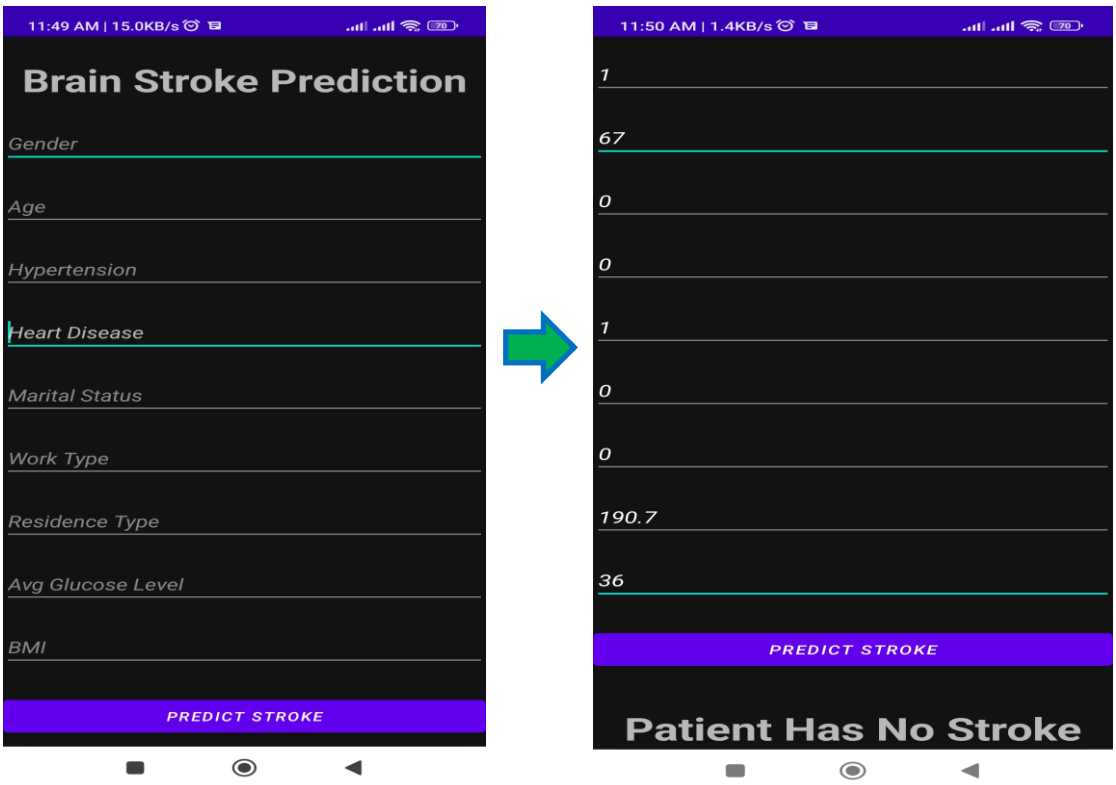


Figure 27. Mobile application

5.5 Discussion

Five models have so far been put into practice, and they all produce results that are pretty decent in terms of accuracy, precision, recall, and F1-score during training and testing. These techniques are rapid and effective when compared to other classification algorithms. Fig. 28 shows that the random forest and XGBoost outperformed other models with the highest score. In terms of the Brier score, Fig. 29 shows that all have a very decent Brier score, but random forest and XGBoost have the lowest Brier score. The Brier score is a cost function (loss function), ranging from 0 to 1. So, a lower Brier score means the model is more accurate to predict. However, random forest and XGBoost are the two most accurate models in our system, but random forest performed better than XGBoost in all performance criteria. So, the best and most accurate random forest model is chosen for the prediction and used for the web application and mobile app. The evaluation scores of all machine learning models are shown in Table 1.

Table 1. Evaluation scores of models

Model	Accuracy	Precision	Recall	F1-Score	Brier-Score
Neural Network	93.21	96.58	90.49	93.44	0.06
Random Forest	**97.11**	**95.51**	**98.86**	**97.16**	**0.02**
XGBoost	97	95.34	97.81	96.54	0.03
Decision Tree	94.66	92.71	96.93	94.78	0.05
KNN	92.08	87.63	97.98	92.52	0.07

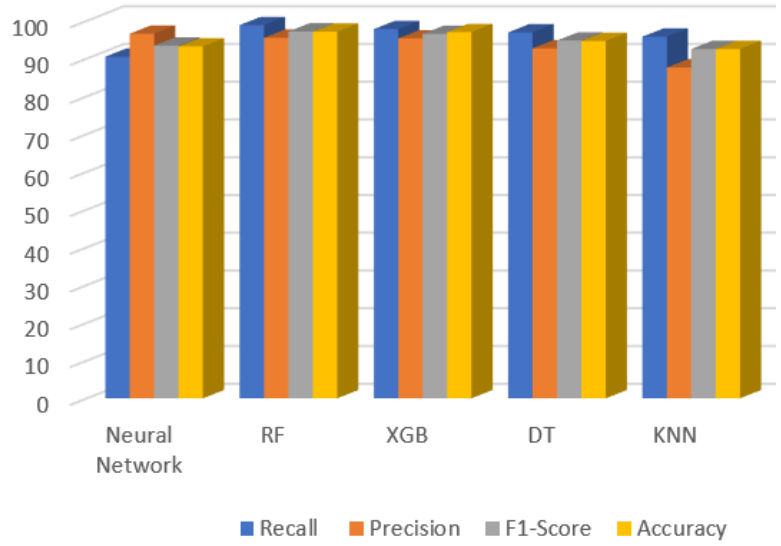


Figure 28. Evaluation graph of models

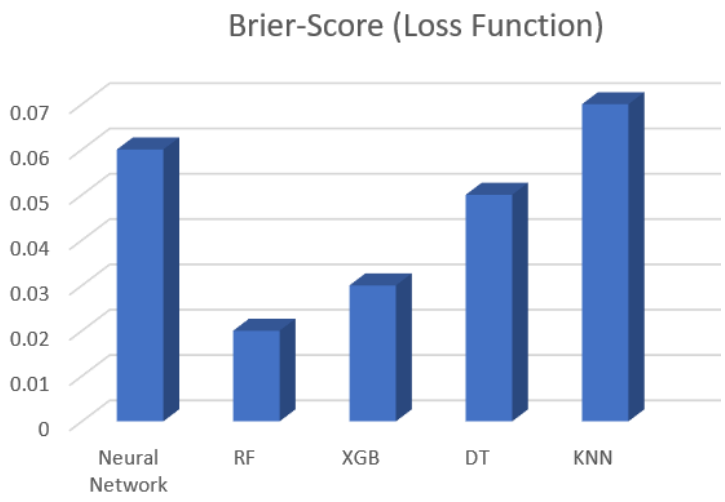


Figure 29. Brier score of each model

5.6 Comparative Analysis

A comparison of the preceding research's parameters and structures is shown in Table 2 for comparison. This will improve the system in comparison to past experiments.

Table 2. Comparison Analysis

Reference	Main parameters	Result
[1]	Improvised Random Forest	Accuracy: 96.7%
[2]	C4.5 Decision Tree	Accuracy: 91.11%
[3]	Artificial Neural Network	Accuracy:95%
[22]	K- Nearest Neighbor	Accuracy: 95%
Our System	Random Forest, XGBoost, and Decision Tree.	Random Forest: Accuracy:97.11%, XGBoost: Accuracy: 97% Decision Tree: Accuracy: 94.66%

6. Novelty of Project

Our system is trained with a dataset of 5026 with 11 features, making it more significant than the prior dataset. We also manage the imbalance problem, preprocess the data, and effectively tune the model. As a consequence, our dataset is more valuable than that of the prior study, and as a result, our model produces beneficial results. The two best accuracy models are Random Forest and XGBoost. On this most accurate model, we build a web application and a mobile application for our system.

7. Impact

With better trained models, the medical field will significantly benefit from our established system's ability to identify strokes early and save many lives. In addition, we have a web application that makes our system usable for the user, and a mobile application is also being developed to increase mobility.

8. Complex Engineering Problems and Activities

Complex engineering problems and activities are written in the Washington Accord (WA) format. The Washington Accord format is a standardized way of writing complex engineering problems and activities. This format defines three key factors: knowledge factor, problem outcome, and activities. Knowledge is defined as K and It has eight parameters (K1-K8), Problem outcome is defined P It has seven parameters(P1-P7), Activities are defined as A. They have five parameters, A1-A5 [*].

8.1 Complex Engineering Problems (CEP)

Table 3. Complex Engineering Problems of Our Project

Attributes		Addressing the complex engineering problems (P) in the project
P1	Depth of knowledge required (K3-K8)	The Project requires knowledge and mechanism of Machine Learning (K3), Dataset, Python, Machine Learning Models, API, Framework and Cloud Server (K4, K5), Engineering & IT Tools (K6), Involve Environmental Effects (K7), and Scientific Research Papers (K8).
P2	Range of conflicting requirements	There are no problems of conflicting issues in our project
P3	Depth of analysis required	No unique way to design. Depth of analysis needed to select a specific solution from many alternatives.
P4	Familiarity of issues	Cerebral stroke prediction using machine learning entails understanding the complexities of integrating diverse patient data sources, managing class imbalance, addressing temporal dynamics, quantifying uncertainty, and ensuring clinical interpretability, all while striving to develop robust and accurate predictive models to aid in timely stroke risk assessment.
P5	Extent of applicable codes	There is no existing code or standard for this project.
P6	Extent of stakeholder involvement	There are no several stakeholders' involvement.
P7	Interdependence	Project involves a number of interdependent sub-systems such as training platform system, data preprocessing, feature engineering, model optimization, application UI for users.

8.2 Complex Engineering Activities (CEA)

Table 4. Complex Engineering Activities of Our Project

Attributes		Addressing the complex engineering activities (A) in the project
A1	Range of resources	This project involves human resource, dataset resource, information, modern tool, framework, cloud server etc.
A2	Level of interactions	Involves interactions between group members to develop our software, analysis between models to obtain a better predictive model, collecting information and a dataset, etc.
A3	Innovation	Employs innovative engineering skills by introducing technology in a different manner in the medical sector and AI sector.
A4	Consequences to society / Environment	Early stroke prediction and project publicity should emphasize this innovation and its significance.
A5	Familiarity	Needs to be familiar with the various machine learning models, dataset preprocessing technique, EDA, feature engineering, API, cloud server and framework.

9. Conclusions

9.1 Summary

Early cerebral stroke prediction is the study's objective. To conduct our investigation, we used five different models: decision trees, random forests, XGBoost, KNN, and neural networks. Here we have collected a stroke dataset of 5026 data points with 11 features. The organization needed to be improved in this dataset. So, after the modest preprocessing of the dataset, we ultimately prepared it for our models. Then, we trained our models on that preprocessed dataset, and they all performed well. The relative accuracy of Random Forest, XGBoost, Decision Tree, KNN, and Neural Network was 97.11%, 97%, 94.66%, 92.08%, and 93.21%, respectively. The best models are Random Forest and XGBoost. In past studies, our model outperformed theirs. Thus, the random forest is used to predict cerebral strokes and to create a hosted web application and a mobile app. Our study's technique will enhance cerebral stroke treatment and safeguard people against the increasing risk of strokes due to unhealthy habits and diets. Early detection using our cerebral stroke prediction technology can help prevent additional damage and improve people's safety.

9.2 Limitations

In our system, the dataset we used had a class imbalance issue. This issue was fixed by generating synthetic data points using the SMOTE algorithm. Here, we could enhance accuracy even further if we had additional data and no concerns about imbalances or null values.

9.3 Future Improvement

In the future, we may add an IoT device to gather stroke parameters automatically from patients and submit them to our system for predictions. We may also include some deep learning techniques or a more advanced machine model to improve accuracy, as well as a chatbot to provide stroke prevention information and suggestions to the user.

10. Bibliography

[1] V. Bandi, D. Bhattacharyya, and D. Midhunchakkravarthy, “Prediction of Brain Stroke Severity Using Machine Learning,” Revue d’Intelligence Artificielle, vol. 34 , pp.753–761, Dec. 2020.

[2] J. Yu, S. Park, H. Lee, C.-S. Pyo, and Y. S. Lee, “An Elderly Health Monitoring System Using Machine Learning and In-Depth Analysis Techniques on the NIH Stroke Scale,” Mathematics, vol. 8, pp. 1-16, Jul. 2020.

[3] P. Govindarajan, R. K. Soundarapandian, A. H. Gandomi, R. Patan, P. Jayaraman, and R. Manikandan, “Classification of stroke disease using machine learning algorithms,” Neural Computing and Applications, vol. 32, pp. 817–828, Jan. 2019.

[4] C. A. Cheng, Y. C. Lin, and H. W. Chiu, “Prediction of the prognosis of ischemic stroke patients after intravenous thrombolysis using artificial neural networks,” Studies in Health Technology and Informatics, vol. 202, pp. 115–118, Jul. 2014.

[5] S. Gupta and S. Raheja, "Stroke Prediction using Machine Learning Methods," International Conference on Cloud Computing, Data Science & Engineering, pp. 553-558, 2022.

[6] Carbonell, Jaime G., Ryszard S. Michalski, and Tom M. Mitchell. "An overview of machine learning." Machine learning, pp: 3-23.

[7] J. Alzubi, A. Nayyar, and A. Kumar, “Machine learning from theory to algorithms: an overview,” Journal of Physics: Conference Series, vol. 1142.

[8] A. Sheth. “History of Machine Learning.” Medium, Bloombench, 25 Aug. 2017.

[9] Ö. Çelik, “A research on machine learning methods and its applications,” Journal of Educational Technology and Online Learning, vol. 1, 2018.

[10] J. Schmidt, M. Marques, S. Botti, and M. Marques, “Recent advances and applications of machine learning in solid-state materials science,” Npj Computational Materials, vol. 5, no. 83, 2019.

[11] “Cerebral Stroke Dataset,” Kaggle. [Online]. Available: https://www.kaggle.com/datasets/zzettrkalpakbal/full-filled-brain-stroke-dataset

[12] “ML - Understanding Data with Visualization,” Tutorialspoint. [Online]. Available:https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_understanding_data_with_visualization

[13] C. Fan, M. Chen, X. Wang, J. Wang, and B. Huang, “A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data,” Frontiers, Feb. 15, 2021.

[14] R. Guo, Z. Zhao, T. Wang, G. Liu, J. Zhao, and D. Gao, “Degradation State Recognition of Piston Pump Based on ICEEMDAN and XGBoost,” Applied Sciences, vol. 10, no. 18, pp. 1-17, Sep. 2020.

[15] “k-nearest neighbors algorithm,” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[16] “KNN figure” [online] Available: https://vatsalparsaniya.github.io/ML_Knowledge/Nearest_neighbours/Readme.html

[17] A. Kyrykovych, "Deep Neural Networks," KDnuggets. [online]. Available: https://www.kdnuggets.com/2020/02/deep-neural-networks.html

[18] “Pickle-python object serialization,” Python documentation. [Online]. Available: https://docs.python.org/3/library/pickle.html

[19] G. Nath, A. Coursey, Y. Li, S. Prabhu, H. Garg, S. C. Halder and S. Sengupta, “An interactive web-based tool for predicting and exploring brain cancer survivability,” Healthcare Analytics, vol. 3, pp. 1-11, 2023.

[20] S. Narayanan, N. M. Balamurugan, M. K and P. B. Palas, "Leveraging Machine Learning Methods for Multiple Disease Prediction using Python ML Libraries and Flask API," International Conference on Applied Artificial Intelligence and Computing, pp. 694-701, 2022.

[21] D. V. Kumar and V. V. J. R. Krishniah, "An automated framework for stroke and hemorrhage detection using decision tree classifier," 2016 International Conference on Communication and Electronics Systems, pp. 1-6, 2016.

[22] S. F. Sung, C. Y. Hsieh, Y. H. K. Yang et al., “Developing a stroke severity index based on administrative data was feasible using data mining techniques,” Journal of Clinical Epidemiology, vol. 68, pp. 1292–1300, 2015.

11. Appendices

11.1 Model Training

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as splt
import seaborn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,brier_score_loss
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score,classification_report
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier


"""#**Data Upload**"""

#loading the csv data to a Pandas DataFrame
brain_stroke = pd.read_csv('/content/full_data_brain.csv')

# print first five rows of the dataframe
brain_stroke.head()

#number of rows and columns in the dataset
brain_stroke.shape

brain_stroke['stroke'].value_counts()

brain_stroke.info()

brain_stroke.isnull().sum()

brain_stroke.duplicated().sum()

brain_stroke.isnull().sum()

"""# ***Data Visualization***"""

for i in brain_stroke.columns:
```

```
    if i == 'bmi' or i == 'age' or i == 'avg_glucose_level':
        continue
    else:
        seaborn.countplot(data=brain_stroke, x = i,palette = 'rocket')
        plt.show()
        print("\n")

for i in brain_stroke.columns:
    seaborn.histplot(data=brain_stroke, x = i,hue = 'stroke',palette =
'rocket_r')
    plt.show()
    print("\n")

"""# **Data Preprocessing**"""

brain_stroke['gender'].unique()

dataChanginggender = {
    "Male" : 1,
    "Female" : 0
}

brain_stroke['gender'] = brain_stroke['gender'].map(dataChanginggender)
brain_stroke

brain_stroke['ever_married'].unique()

dataChangingmarrried = {
    "Yes" : 1,
    "No" : 0
}

brain_stroke['ever_married'] =
brain_stroke['ever_married'].map(dataChangingmarrried)
brain_stroke

brain_stroke['work_type'].unique()

brain_stroke['Residence_type'].unique()

dataChangingResidence_type= {
    "Urban" : 0,
    "Rural" : 1
}

brain_stroke['Residence_type'] =
brain_stroke['Residence_type'].map(dataChangingResidence_type)
brain_stroke
```

```
brain_stroke=pd.get_dummies(brain_stroke,columns=['work_type'])

#final formation of table
brain_stroke

brain_stroke.isnull().sum()

"""# **Data Spliting**"""

X = brain_stroke.drop(columns = ['stroke','smoking_status'], axis=1)
Y = brain_stroke['stroke']

X.shape

print(X)

Y.shape

print(Y)

from imblearn.over_sampling import RandomOverSampler, SMOTE
s=SMOTE()
X_balanced, Y_balanced = s.fit_resample(X, Y)

Y_balanced.value_counts()

sns.barplot(x=brain_stroke["stroke"].value_counts().index, y =
brain_stroke["stroke"].value_counts().values)
plt.xlabel("stroke", fontsize=13,fontweight="regular")
plt.ylabel("count",fontsize=13,fontweight="regular")

sns.barplot(x=Y_balanced.value_counts().index, y =
Y_balanced.value_counts().values)
plt.xlabel("stroke", fontsize=13,fontweight="regular")
plt.ylabel("count",fontsize=13,fontweight="regular")

X_train, X_test, Y_train, Y_test = train_test_split(X_balanced,
Y_balanced, stratify = Y_balanced, test_size = 0.25, random_state = 49)

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

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

X_balanced.info()

"""# **Decision Tree Model**"""

dtm = DecisionTreeClassifier(criterion = 'entropy', max_depth = 15)
```

```
dtm.fit(X_train, Y_train)

"""**Accuracy score of DT**


"""

X_train_prediction_dtm = dtm.predict(X_train)
accuracy_score( Y_train,X_train_prediction_dtm,)
print('Accuracy on training data : ',
round(accuracy_score(X_train_prediction_dtm, Y_train,)*100,2), '%')

X_test_prediction_dtm = dtm.predict(X_test)
accuracy_score(X_test_prediction_dtm, Y_test,)
print('Accuracy on testing data : ',
round(accuracy_score(X_test_prediction_dtm, Y_test,)*100,2), '%')

"""**Loss score of DT**"""

#loss score of train
print('loss on training data : ',
brier_score_loss(Y_train,X_train_prediction_dtm))

#loss score of test
print('loss on test data : ',
brier_score_loss(Y_test,X_test_prediction_dtm))

"""**Precision score of DT**"""

#precision score of train
precision_train= precision_score(Y_train,X_train_prediction_dtm)
print("Train precision score: ",precision_train*100,'%')

#precision score of test
precision_test= precision_score(Y_test,X_test_prediction_dtm)
print("Test precision score: ",precision_test*100,'%')

"""**Recall score of DT**"""

#recall score of train
recall_score_train= recall_score(Y_train,X_train_prediction_dtm)
print("Train recall score: ",recall_score_train*100,'%')

#recall score of test
recall_score_test= recall_score(Y_test,X_test_prediction_dtm)
print("Test recall score: ",recall_score_test*100,'%')
```

```
"""**F1 score of DT**"""

#f1 score of train
f1_score_train= f1_score(Y_train,X_train_prediction_dtm)
print("Train f1 score: ",f1_score_train*100,'%')

#f1 score of test
f1_score_test= f1_score(Y_test,X_test_prediction_dtm)
print("Test f1 score: ",f1_score_test*100,'%')

"""**Feature importances**"""

#importance of feature
dtm.feature_importances_

#importance table
df_dtm=pd.DataFrame({"Features_Names":X.columns,"Importances":dtm.featu
re_importances_})
df_dtm_srt=df_dtm.sort_values(by="Importances",ascending=False)

df_dtm_srt

plt.figure(figsize=[10,10])
plt.bar(df_dtm_srt["Features_Names"],df_dtm_srt["Importances"])

"""**DT Train Confusion Matrix**"""

confusion_matrix(Y_train,X_train_prediction_dtm)

axdt = sns.heatmap(confusion_matrix(Y_train,X_train_prediction_dtm),
annot=True, cmap='Blues')

axdt.set_title('Decision Tree Confusion Matrix\n\n');
axdt.set_xlabel('\nActual Values')
axdt.set_ylabel('Predicted Values');

axdt.xaxis.set_ticklabels(['True','False'])
axdt.yaxis.set_ticklabels(['True','False'])

splt.show()

"""**DT Test Confusion Matrix**"""

confusion_matrix(Y_test,X_test_prediction_dtm)

axdt = sns.heatmap(confusion_matrix(Y_test,X_test_prediction_dtm),
annot=True, cmap='Blues')
```

```
axdt.set_title('Decision Tree Confusion Matrix\n\n');
axdt.set_xlabel('\nActual Values')
axdt.set_ylabel('Predicted Values');

axdt.xaxis.set_ticklabels(['True','False'])
axdt.yaxis.set_ticklabels(['True','False'])

splt.show()

"""**Testing Decision Tree**"""

#input=(1,67,0,1,1,0,228.69,36.6,0,1,0,0) #stroke
input=(0,22,0,0,0,1,79.81,27.7,0,1,0,0)  #no stroke
input_np=np.asarray(input)
input_rs=input_np.reshape(1,-1)
prd=dtm.predict(input_rs)
if(prd==0):
  print("Patient has not stroke\n")
else:
  print("Patient has stroke\n")

"""**DT classification report**"""

print(classification_report(Y_test,X_test_prediction_dtm))

"""**Overfit checking for DT**"""

#overfit checking for DT
max_depth_param=range(1,16,1)
train_acc=[] #list of train accuracy
test_acc=[]  #list of test accuracy
for i in max_depth_param:
    dtm = DecisionTreeClassifier(criterion = 'entropy', max_depth = i)
    dtm.fit(X_train, Y_train)
    X_train_prediction_dtm = dtm.predict(X_train) #predict with trained
value
    train_acc.append(accuracy_score(X_train_prediction_dtm,
Y_train,)*100) #accuracy of train in each depth
    X_test_prediction_dtm = dtm.predict(X_test) #predict with test
value
    test_acc.append(accuracy_score(X_test_prediction_dtm, Y_test,)*100)
#accuracy of train in each depth

fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(max_depth_param,train_acc,label="train")
ax.plot(max_depth_param,test_acc,label="test")
ax.tick_params(axis='x', colors='black')
```

```
ax.tick_params(axis='y', colors='black')
plt.title('Overfit Checking Graph')
plt.ylabel('accuracy')
plt.xlabel('depth')
plt.legend()
plt.show()

"""# ***Decision tree Optimise***"""

parameters = {'max_depth' : (3,5,7,9,10,15,20,25)
              , 'criterion' : ('gini', 'entropy')
              , 'max_features' : ('auto', 'sqrt', 'log2')
              , 'min_samples_split' : (2,4,6)
             }

DT_grid  = RandomizedSearchCV(DecisionTreeClassifier(),
param_distributions = parameters, cv = 5, verbose = True)

DT_grid.fit(X_train, Y_train)
DT_grid.best_estimator_

DT_Model = DecisionTreeClassifier(max_depth=25, max_features='sqrt',
min_samples_split=6)

DT_Model.fit(X_train, Y_train)

print (f'Train Accuracy - : {DT_Model.score(X_train,
Y_train)*100:.3f}')
print (f'Test Accuracy - : {DT_Model.score(X_test, Y_test)*100:.3f}')

"""# ***XGB Classifier***"""

#colsample_bytree=0.7, gamma=0.0, max_depth=10
#colsample_bytree=0.5, gamma=0.0, max_depth=15, min_child_weight=3
xgb = XGBClassifier(colsample_bytree=0.5, gamma=0.0, max_depth=17,
min_child_weight=3)
xgb.fit(X_train.values, Y_train.values)

"""**Accuracy Score Of XGB**"""

#Training accuracy
X_train_prediction_xgb = xgb.predict(X_train.values)
accuracy_score(X_train_prediction_xgb, Y_train.values,)
print('Accuracy on training data : ',
round(accuracy_score(X_train_prediction_xgb, Y_train.values,)*100,2),
'%')
```

```
#Testing accuracy
X_test_prediction_xgb = xgb.predict(X_test.values)
accuracy_score(X_test_prediction_xgb, Y_test.values,)
print('Accuracy on testing data : ',
round(accuracy_score(X_test_prediction_xgb, Y_test.values,)*100,2),
'%')

"""**Loss score of xgb**"""

#loss score of train
print('loss on training data : ',
brier_score_loss(Y_train,X_train_prediction_xgb))

#loss score of test
print('loss on test data : ',
brier_score_loss(Y_test,X_test_prediction_xgb))

"""**Precision Score Of XGB**"""

#precision score of train
precision_train= precision_score(Y_train.values,X_train_prediction_xgb)
print("Train precision score: ",precision_train*100,'%')

#precision score of test
precision_test= precision_score(Y_test.values,X_test_prediction_xgb)
print("Test precision score: ",precision_test*100,'%')

"""**Recall Score Of XGB**"""

#recall score of train
recall_score_train= recall_score(Y_train.values,X_train_prediction_xgb)
print("Train recall score: ",recall_score_train*100,'%')

#recall score of test
recall_score_test= recall_score(Y_test.values,X_test_prediction_xgb)
print("Test recall score: ",recall_score_test*100,'%')

"""**F1 Score Of XGB**"""

#f1 score of train
f1_score_train= f1_score(Y_train.values,X_train_prediction_xgb)
print("Train f1 score: ",f1_score_train*100,'%')

#f1 score of test
f1_score_test= f1_score(Y_test.values,X_test_prediction_xgb)
print("Test f1 score: ",f1_score_test*100,'%')
```

```
"""**XGB train confusion matrix**"""

#train confusin matrix
confusion_matrix(Y_train.values,X_train_prediction_xgb)

#train confusin matrix
axxgb = sns.heatmap(confusion_matrix(Y_train,X_train_prediction_xgb),
annot=True, cmap='Blues')

axxgb.set_title('XGB Confusion Matrix with labels\n\n');
axxgb.set_xlabel('\nActual Values')
axxgb.set_ylabel('Predicted Values ');

axxgb.xaxis.set_ticklabels(['True','False'])
axxgb.yaxis.set_ticklabels(['True','False'])

## Display the visualization of the Confusion Matrix.
splt.show()

"""**XGB test confusion matrix**"""

#test confusin matrix
confusion_matrix(Y_test.values,X_test_prediction_xgb)

#test confusin matrix
axxgb = sns.heatmap(confusion_matrix(Y_test,X_test_prediction_xgb),
annot=True, cmap='Blues')

axxgb.set_title('XGB Confusion Matrix with labels\n\n');
axxgb.set_xlabel('\nActual Values')
axxgb.set_ylabel('Predicted Values ');

axxgb.xaxis.set_ticklabels(['True','False'])
axxgb.yaxis.set_ticklabels(['True','False'])

## Display the visualization of the Confusion Matrix.
splt.show()

"""**XGB classification report**"""

print(classification_report(Y_test,X_test_prediction_xgb))

"""**Testing XGB**"""

#input=(1,67,0,1,1,0,228.69,36.6,0,1,0,0) #stroke
input=(0,22,0,0,0,1,79.81,27.7,0,1,0,0)  #no stroke
input_np=np.asarray(input)
input_rs=input_np.reshape(1,-1)
```

```
prd=xgb.predict(input_rs)
if(prd==0):
  print("Patient has not stroke\n")
else:
  print("Patient has stroke\n")

"""**Overfit Checking of XGB**"""

#overfit checking for XGB
max_depth_param=range(1,10,1)
train_acc=[] #list of train accuracy
test_acc=[]  #list of test accuracy
for i in max_depth_param:
    xgb =  XGBClassifier(colsample_bytree=0.7, gamma=0.0, max_depth=i)
    xgb.fit(X_train.values, Y_train.values)
    X_train_prediction_xgb = xgb.predict(X_train.values) #predict with
trained value
    train_acc.append(accuracy_score(X_train_prediction_xgb,
Y_train.values,)*100) #accuracy of train in each depth
    X_test_prediction_xgb = xgb.predict(X_test.values) #predict with
test value
    test_acc.append(accuracy_score(X_test_prediction_xgb,
Y_test.values,)*100) #accuracy of train in each depth

fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(max_depth_param,train_acc,label="train")
ax.plot(max_depth_param,test_acc,label="test")
ax.tick_params(axis='x', colors='black')
ax.tick_params(axis='y', colors='black')
plt.title('Overfit Checking Graph')
plt.ylabel('accuracy')
plt.xlabel('depth')
plt.legend()
plt.show()

"""# ***Random Forest Model***"""

from sklearn.ensemble import RandomForestClassifier

Rclf = RandomForestClassifier(max_depth=18)

Rclf.fit(X_train, Y_train)

"""**Accuracy score of RF**"""

X_train_prediction_rclf = Rclf.predict(X_train)
accuracy_score(X_train_prediction_rclf, Y_train,)
```

```
print('Accuracy on training data : ',
round(accuracy_score(X_train_prediction_rclf, Y_train,)*100,2), '%')

X_test_prediction_rclf = Rclf.predict(X_test)
accuracy_score(X_test_prediction_rclf, Y_test,)
print('Accuracy on testing data : ',
round(accuracy_score(X_test_prediction_rclf, Y_test,)*100,2), '%')

"""**Loss score of RF**"""

#loss score of train
print('loss on training data : ',
brier_score_loss(Y_train,X_train_prediction_rclf))

#loss score of test
print('loss on test data : ',
brier_score_loss(Y_test,X_test_prediction_rclf))

"""**Precision Score of RF**


"""

#precision score of train
precision_train= precision_score(Y_train,X_train_prediction_rclf)
print("Train precision score: ",precision_train*100,'%')

#precision score of test
precision_test= precision_score(Y_test,X_test_prediction_rclf)
print("Test precision score: ",precision_test*100,'%')

"""**Recall Score of RF**"""

#recall score of train
recall_score_train= recall_score(Y_train,X_train_prediction_rclf)
print("Train recall score: ",recall_score_train*100,'%')

#recall score of test
recall_score_test= recall_score(Y_test,X_test_prediction_rclf)
print("Test recall score: ",recall_score_test*100,'%')

"""**f1 Score of RF**"""

#f1 score of train
f1_score_train= f1_score(Y_train,X_train_prediction_rclf)
print("Train f1 score: ",f1_score_train*100,'%')

#f1 score of test
```

```
f1_score_test= f1_score(Y_test,X_test_prediction_rclf)
print("Test f1 score: ",f1_score_test*100,'%')

"""**RF Train Confusion Matrix**"""

#train confusin matrix
predRF = Rclf.predict(X_train)
confusion_matrix(Y_train,predRF)

#train confusin matrix
axrclf = sns.heatmap(confusion_matrix(Y_train,predRF), annot=True,
cmap='Blues')

axrclf.set_title('Random Forest Confusion Matrix with labels\n\n');
axrclf.set_xlabel('\nActual Values')
axrclf.set_ylabel('Predicted Values ');

axrclf.xaxis.set_ticklabels(['True','False'])
axrclf.yaxis.set_ticklabels(['True','False'])

## Display the visualization of the Confusion Matrix.
splt.show()

"""**RF Test Confusion Matrix**"""

#test confusin matrix
predRF = Rclf.predict(X_test)
confusion_matrix(Y_test,predRF)

#test confusin matrix
axrclf = sns.heatmap(confusion_matrix(Y_test,predRF), annot=True,
cmap='Blues')

axrclf.set_title('Random Forest Confusion Matrix with labels\n\n');
axrclf.set_xlabel('\nActual Values')
axrclf.set_ylabel('Predicted Values ');

axrclf.xaxis.set_ticklabels(['True','False'])
axrclf.yaxis.set_ticklabels(['True','False'])

## Display the visualization of the Confusion Matrix.
splt.show()

"""**Testing Random Forest**"""

#input=(1,67,0,1,1,0,228.69,36.6,0,1,0,0) #stroke
#input=(1,67,0,0,1,0,190.7,36,0,1,0,0) # No stroke
input=(0,22,0,0,0,1,79.81,27.7,0,1,0,0)  #no stroke
```

```
input_np=np.asarray(input)
input_rs=input_np.reshape(1,-1)
prd=Rclf.predict(input_rs)
if(prd==0):
  print("Patient has not stroke\n")
else:
  print("Patient has stroke\n")

"""**RF classification report**"""

print(classification_report(Y_test,X_test_prediction_rclf))

"""**Random Forest Optimization With Random Search CV**"""

# Number of trees in random forest
# linspace: Return evenly spaced numbers over a specified interval
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)

Rclf1 = RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=Rclf1,param_distributions=rand
om_grid,n_iter=100,cv=3,verbose=2,
                               random_state=100,n_jobs=-1)
# Controls the verbosity: the higher, the more messages
# Number of jobs to run in parallel
### fit the randomized model
rf_randomcv.fit(X_train,Y_train)

rf_randomcv.best_params_

best_random_grid=rf_randomcv.best_estimator_
# storing the best hyperparameters
```

```
X_train_prediction_rclf = best_random_grid.predict(X_train)
print('Accuracy on training data : ', round(accuracy_score(Y_train,
X_train_prediction_rclf)*100,2), '%')

X_test_prediction_rclf = best_random_grid.predict(X_test)
print('Accuracy on testing data : ', round(accuracy_score(Y_test,
X_test_prediction_rclf)*100,2), '%')

axdt = sns.heatmap(confusion_matrix(Y_test, X_test_prediction_rclf),
annot=True, cmap='Blues')

axdt.set_title('Random Forest Confusion Matrix with optimization\n\n');
axdt.set_xlabel('\nActual Values')
axdt.set_ylabel('Predicted Values');

axdt.xaxis.set_ticklabels(['True','False'])
axdt.yaxis.set_ticklabels(['True','False'])

splt.show()

print(classification_report(Y_test, X_test_prediction_rclf))

"""**Explainable AI**"""

#%pip install lime
import lime
from lime import lime_tabular

explainer = lime_tabular.LimeTabularExplainer(
    training_data=np.array(X_train),
    feature_names=X_train.columns,
    class_names=['0', '1'], # Check your class names in the CSV file.
It should be 1, 2, 3, 4, 5 I think
    mode='classification'
)

exp = explainer.explain_instance(
    data_row=X_test.iloc[1],
    predict_fn=Rclf.predict_proba
)

exp.show_in_notebook(show_table=True)

"""# **KNN Model**"""

#model implimentation
knn = KNeighborsClassifier(n_neighbors = 3, p=3)
```

```
knn.fit(X_train, Y_train)

"""**Accuracy Score of KNN**"""

X_train_prediction_knn = knn.predict(X_train)
accuracy_score(X_train_prediction_knn, Y_train,)
print('Accuracy on training data : ',
round(accuracy_score(X_train_prediction_knn, Y_train,)*100,2), '%')

X_test_prediction_knn = knn.predict(X_test)
accuracy_score(X_test_prediction_knn, Y_test,)
print('Accuracy on testing data : ',
round(accuracy_score(X_test_prediction_knn, Y_test,)*100,2), '%')

"""**Loss score of KNN**"""

#loss score of train
print('loss on training data : ',
brier_score_loss(Y_train,X_train_prediction_knn))

#loss score of test
print('loss on test data : ',
brier_score_loss(Y_test,X_test_prediction_knn))

"""**Precision Score of KNN**"""

#precision score of train
precision_train= precision_score(Y_train,X_train_prediction_knn)
print("Train precision score: ",precision_train*100,'%')

#precision score of test
precision_test= precision_score(Y_test,X_test_prediction_knn)
print("Test precision score: ",precision_test*100,'%')

"""**Recall Score of KNN**"""

#recall score of train
recall_score_train= recall_score(Y_train,X_train_prediction_knn)
print("Train recall score: ",recall_score_train*100,'%')

#recall score of test
recall_score_test= recall_score(Y_test,X_test_prediction_knn)
print("Test recall score: ",recall_score_test*100,'%')

"""**F1 score of KNN**"""

#f1 score of train
```

```
f1_score_train= f1_score(Y_train,X_train_prediction_knn)
print("Train f1 score: ",f1_score_train*100,'%')

#f1 score of test
f1_score_test= f1_score(Y_test,X_test_prediction_knn)
print("Test f1 score: ",f1_score_test*100,'%')

"""**Confusion matrix of train**"""

confusion_matrix(Y_train,X_train_prediction_knn)

axknn = sns.heatmap(confusion_matrix(Y_train,X_train_prediction_knn),
annot=True, cmap='Blues')

axknn.set_title('KNN Confusion Matrix\n\n');
axknn.set_xlabel('\nActual Values')
axknn.set_ylabel('Predicted Values ');

axknn.xaxis.set_ticklabels(['True','False'])
axknn.yaxis.set_ticklabels(['True','False'])

splt.show()

"""**Confusion matrix of test**"""

confusion_matrix(Y_test,X_test_prediction_knn)

axknn = sns.heatmap(confusion_matrix(Y_test,X_test_prediction_knn),
annot=True, cmap='Blues')

axknn.set_title('KNN Confusion Matrix\n\n');
axknn.set_xlabel('\nActual Values')
axknn.set_ylabel('Predicted Values ');

axknn.xaxis.set_ticklabels(['True','False'])
axknn.yaxis.set_ticklabels(['True','False'])

splt.show()

print(classification_report(Y_test,X_test_prediction_knn))

"""**Testing Of KNN Model**"""

input=(1,67,0,1,1,0,228.69,36.6,0,1,0,0) #stroke
#input=(0,22,0,0,0,1,79.81,27.7,0,1,0,0)  #no stroke
input_np=np.asarray(input)
input_rs=input_np.reshape(1,-1)
prd=knn.predict(input_rs)
```

```
if(prd==0):
  print("Patient has not stroke\n")
else:
  print("Patient has stroke\n")

"""# ***Neural Network Model***"""

import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
from sklearn.preprocessing import StandardScaler

#for DNN seperate spliting
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X_balanced,
Y_balanced, stratify = Y_balanced, test_size = 0.25, random_state = 49)
sclr=StandardScaler()
X_train_std=sclr.fit_transform(X1_train)
X_test_std=sclr.transform(X1_test)

model=keras.Sequential([
    keras.layers.Flatten(input_shape=(12,)),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dense(30, activation='sigmoid'),
    keras.layers.Dense(30, activation='sigmoid'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(20, activation='sigmoid'),
    keras.layers.Dense(2, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_std, Y1_train, validation_split=0.20,
epochs=1105)

#visualization of accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'lower right')

#visualization of loss
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'upper right')

#Accuracy score of NN
loss, accuracy = model.evaluate(X_test_std, Y1_test)
print(accuracy*100)

Y1_pred=model.predict(X_test_std)

#converstion of lebel
Y1_pred_lebel=[np.argmax(i) for i in Y1_pred]

"""**Neural Network Test Confusion Matrix**"""

#neural network test confusion matrix
confusion_matrix(Y1_pred_lebel,Y1_test)

#neural network test confusion matrix
confusion_matrix(Y1_test,Y1_pred_lebel)

#test confusin matrix
axxgb = sns.heatmap(confusion_matrix(Y1_test,Y1_pred_lebel),
annot=True, cmap='Blues')

axxgb.set_title('Neural Confusion Matrix with labels\n\n');
axxgb.set_xlabel('\nActual Values')
axxgb.set_ylabel('Predicted Values ');

axxgb.xaxis.set_ticklabels(['True','False'])
axxgb.yaxis.set_ticklabels(['True','False'])

## Display the visualization of the Confusion Matrix.
splt.show()

"""**Precision score of NN**

"""

#precision score of test
precision_test= precision_score(Y1_pred_lebel,Y1_test)
print("Test precision score: ",precision_test*100,'%')
```

```
"""**Recall score of NN**

"""

#recall score of test
recall_score_test= recall_score(Y1_pred_lebel,Y1_test)
print("Test recall score: ",recall_score_test*100,'%')

"""**F1 score score of NN**


"""

#f1 score of test
f1_score_test= f1_score(Y1_pred_lebel,Y1_test)
print("Test f1 score: ",f1_score_test*100,'%')

"""**Loss score of NN**"""

#loss score of test
print('loss on test data : ', brier_score_loss(Y1_pred_lebel,Y1_test))

"""**Classification report of NN**


"""

#classification report of test
print(classification_report(Y1_pred_lebel,Y1_test))
```

11.2 Web Application Implementation

Brain_web.py

```
import numpy as np
import pickle
import streamlit as st

#load model
loaded_model= pickle.load(open('model.sav','rb'))

#stroke prediction function
def brainStroke(input_data):
    input_np=np.asarray(input_data,dtype=object)
    input_rs=input_np.reshape(1,-1)
    try:
      prd=loaded_model.predict(input_rs)
    except:
```

```
      return 'Fill the required space or use correct string'
    #giving result according to condition
    if(prd==0):
      return 'Patient has not stroke'
    else:
      return 'Patient has stroke'

def main():
    st.title('Brain Stroke Prediction Web App')
    gender=st.radio("Gender",["Male","Female"])
    age=st.text_input("Age")
    hypertension=st.selectbox("Hypertension",["Yes","No"])
    heart_disease=st.selectbox("Heart Disease",["Yes","No"])
    ever_married=st.selectbox("Married Status",["Yes","No"])
    work_type=st.selectbox("Work Type",["Private","Self-
employed","Children","Govt_job"])
    Residence_type=st.selectbox("Residence Type",["Urban","Rural"])
    avg_glucose_level=st.text_input("Avg Glucose")
    bmi=st.text_input("BMI")
    diagnosis= ''
    if st.button("Check"):
        #for gender conversion
        if gender == 'Male':
            gender = 1
        else:
            gender = 0

        # for hypertension conversion
        if hypertension == 'Yes':
            hypertension = 1
        else:
            hypertension = 0

        # for heart_disease conversion
        if heart_disease == 'Yes':
            heart_disease = 1
        else:
            heart_disease = 0

        # for ever_married conversion
        if ever_married == 'Yes':
            ever_married = 1
        else:
            ever_married = 0

        # for Residence_type conversion
        if Residence_type == 'Urban':
            Residence_type = 0
        else:
            Residence_type = 1

        #for work_type conversion
        if work_type == 'Private':
            work_type = 0
        elif work_type == 'Self-employed':
            work_type = 1
        elif work_type == 'Children':
            work_type = 2
        else:
            work_type = 3
```

```
diagnosis=brainStroke([gender,age,hypertension,heart_disease,ever_married,w
ork_type,Residence_type,avg_glucose_level,bmi])
    st.success(diagnosis)


if __name__=='__main__':
    main()
```

11.3 Mobile App API

app.py

```
from flask import Flask,request, jsonify
import pickle
import numpy as np

model = pickle.load(open('model.pkl','rb'))
app = Flask(__name__)
@app.route('/')
def hello_world():
    return "Hello Patient"

@app.route('/predict', methods=['POST'])
def predict():
    gender=request.form.get('gender')
    age = request.form.get('age')
    hypertension = request.form.get('hypertension')
    heart_disease = request.form.get('heart_disease')
    ever_married = request.form.get('ever_married')
    work_type= request.form.get('work_type')
    Residence_type = request.form.get('Residence_type')
    avg_glucose_level = request.form.get('avg_glucose_level')
    bmi = request.form.get('bmi')
```

```
    input_np                                                                =
np.array([[gender,age,hypertension,heart_disease,ever_married,work_type
,Residence_type,avg_glucose_level,bmi]])

    prd = model.predict(input_np)[0]

    return jsonify({'result':str(prd)})


if __name__ == '__main__':

    app.run(debug=True)
```

11.4 Software Implementation

BuildConfig.java

```
package com.example.my_application_brain;

public final class BuildConfig {
  public static final boolean DEBUG = Boolean.parseBoolean("true");
  public static final String APPLICATION_ID =
"com.example.my_application_brain";
  public static final String BUILD_TYPE = "debug";
  public static final int VERSION_CODE = 1;
  public static final String VERSION_NAME = "1.0";
}
```

MainActivity.java

```
package com.example.my_application_brain;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
```

```
import org.json.JSONException;
import org.json.JSONObject;

import java.util.HashMap;
import java.util.Map;

public class MainActivity extends AppCompatActivity {
    EditText
gender,age,hypertension,heart_disease,ever_married,work_type,Residence_
type,avg_glucose_level,bmi;
    Button predict;
    TextView result;
    String url ="https://brain-mbl-app.onrender.com/predict";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        gender = findViewById(R.id.gender);
        age = findViewById(R.id.age);
        hypertension = findViewById(R.id.hypertension);
        heart_disease = findViewById(R.id.heart_disease);
        ever_married = findViewById(R.id.ever_married);
        work_type = findViewById(R.id.work_type);
        Residence_type = findViewById(R.id.Residence_type);
        avg_glucose_level = findViewById(R.id.avg_glucose_level);
        bmi = findViewById(R.id.bmi);
        predict=findViewById(R.id.predict);
        result=findViewById(R.id.result);

        predict.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                StringRequest stringRequest= new
StringRequest(Request.Method.POST, url,
                        new Response.Listener<String>() {
                            @Override
                            public void onResponse(String response) {
                                try {
                                    JSONObject jsonObject= new
JSONObject(response);
                                    String data=
jsonObject.getString("result");
                                    if(data.equals("1")){
                                        result.setText("Patient Has
Stroke");
                                    }else{
```

```
                                        result.setText("Patient Has No
Stroke");
                                    }

                                } catch (JSONException e) {
                                    throw new RuntimeException(e);
                                }


                            }
                        },
                        new Response.ErrorListener() {
                            @Override
                            public void onErrorResponse(VolleyError
error) {
                                Toast.makeText(MainActivity.this,
error.getMessage(), Toast.LENGTH_SHORT).show();

                            }
                        }){
                    @Override
                    protected Map<String,String> getParams(){
                       Map<String,String> params= new
HashMap<String,String>();

params.put("gender",gender.getText().toString());
                        params.put("age",age.getText().toString());

params.put("hypertension",hypertension.getText().toString());

params.put("heart_disease",heart_disease.getText().toString());

params.put("ever_married",ever_married.getText().toString());

params.put("work_type",work_type.getText().toString());

params.put("Residence_type",Residence_type.getText().toString());

params.put("avg_glucose_level",avg_glucose_level.getText().toString());
                        params.put("bmi",bmi.getText().toString());
                        return params;
                    }
                };
                RequestQueue queue=
Volley.newRequestQueue(MainActivity.this);
                queue.add(stringRequest);
```

```
            }
        });


    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    >
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
        >
            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Brain Stroke Prediction"
                android:textSize="35dp"
                android:textStyle="bold"
                android:textAlignment="center"
                android:layout_marginTop="25dp"></TextView>
            <EditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Gender"
                android:textStyle="italic"
                android:id="@+id/gender"
                android:layout_marginTop="25dp"
                ></EditText>
            <EditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Age"
                android:textStyle="italic"
                android:id="@+id/age"
                android:layout_marginTop="25dp"
                ></EditText>
            <EditText
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Hypertension"
                android:textStyle="italic"
                android:id="@+id/hypertension"
                android:layout_marginTop="25dp"
                ></EditText>
```

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Heart Disease"
    android:textStyle="italic"
    android:id="@+id/heart_disease"
    android:layout_marginTop="25dp"
    ></EditText>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Marital Status"
    android:textStyle="italic"
    android:id="@+id/ever_married"
    android:layout_marginTop="25dp"
    ></EditText>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Work Type"
    android:textStyle="italic"
    android:id="@+id/work_type"
    android:layout_marginTop="25dp"
    ></EditText>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Residence Type"
    android:textStyle="italic"
    android:id="@+id/Residence_type"
    android:layout_marginTop="25dp"
    ></EditText>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Avg Glucose Level"
    android:textStyle="italic"
    android:id="@+id/avg_glucose_level"
    android:layout_marginTop="25dp"
    ></EditText>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="BMI"
    android:textStyle="italic"
    android:id="@+id/bmi"
    android:layout_marginTop="25dp"
    ></EditText>
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Predict Stroke"
    android:textStyle="italic"
    android:layout_marginTop="30dp"
    android:id="@+id/predict"></Button>
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=""
    android:textStyle="bold"
    android:textSize="35dp"
```

```
                android:textAlignment="center"
                android:layout_marginTop="40dp"
                android:id="@+id/result"></TextView>




        </LinearLayout>

    </ScrollView>


</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.My_Application_Brain"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

ExampleInstrumentedTest.java

```
package com.example.my_application_brain;

import android.content.Context;

import androidx.test.platform.app.InstrumentationRegistry;
import androidx.test.ext.junit.runners.AndroidJUnit4;

import org.junit.Test;
import org.junit.runner.RunWith;
```

```
import static org.junit.Assert.*;

@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
    @Test
    public void useAppContext() {
        // Context of the app under test.
        Context appContext =
InstrumentationRegistry.getInstrumentation().getTargetContext();
        assertEquals("com.example.my_application_brain",
appContext.getPackageName());
    }
}
```

ExampleUnitTest.java

```
package com.example.my_application_brain;

import org.junit.Test;

import static org.junit.Assert.*;

public class ExampleUnitTest {
    @Test
    public void addition_isCorrect() {
        assertEquals(4, 2 + 2);
    }
}
```

----THE END----