

**DIABETES MELLITUS PREDICTION IN  
PREGNANT WOMEN USING HYBRID DEEP LEARNING**

*by*

**VENKATA KRISHNAMANAIDU MANDADI(vmandad1)**

**LALITH ADITYA CHINTAM REDDY(lchinth1)**

**UMA CHANDRA(uchandra)**

**SHAMEER SHAIK(sshaik20)**

## 1. TITLE

### **Diabetes Mellitus Prediction in Pregnant Women Using Hybrid Deep Learning**

## 2. TEAM MEMBERS

Mandadi.Venakata Krishnamanaidu|vmmandad1|vmmandad1@students.kennesaw.edu Uma Chandra|uchandra|uchandra@students.kennesaw.edu| Shameer shaik|sshaik20|sshaik20@students.kennesaw.edu  
LalithAditya Chintam Reddy|lchinth1|lchinth1@students.kennesaw.edu

## 3. ABSTRACT

Diabetes Mellitus is a major public health concern in our nation. It is a metabolic illness that affects thousands of people and is brought on by the body storing too much glucose. Diabetes can lead to several health problems in humans if it is not identified early on, such as heart disease, kidney disease, nerve damage, eye damage and more. Diabetes mellitus during pregnancy, known as gestational diabetes mellitus (GDM), poses significant risks to both the mother and the unborn child. Early and accurate prediction of GDM is crucial for timely intervention and improved maternal and fetal outcomes.

This study aims to predict the diabetes mellitus in pregnant woman using Deep learning Algorithm such as GRU+LSTM, GRU, LSTM, RNN and Machine Learning algorithms such as XG-Boost and Random Forest classifiers on a real-time diabetes dataset. To yield the best possible remedy for the diabetes mellitus prediction, extensive research and new methods such as SMOTE are considered in this job.

The dataset used for the project is Pima Indian Diabetes Dataset from National Institutes of Diabetes and Digestive and Kidney Diseases. There is total 768 patients and all of them are females of at least 21 years old. The dataset has eight predictor variables and a target variable. This study examines the model accuracies, after applying SMOTE algorithm, GRU+LSTM and RNN stood top with accuracies such as 99% and 96% surpassing accuracies of remaining models Random Forest (85%), XG-BOOST (86%), GRU (90%), LSTM (82%). This paper compares the existing model [3] accuracies, XG Boost(84%), Random Forest(83%), Decision Tree(79%), with the proposed model accuracies, and these proposed models outperformed the accuracy with 99% in GRU+LSTM, 96% in RNN, 86% in XG-Boost and 85% in Random Forest

## 4. INTRODUCTION

Diabetes is a chronic (long-lasting) health problem that alters how the body uses food as fuel. Diabetes will always rank highest among the most common and deadly diseases. Compared to male patients, female patients are more precarious. This is a result of the way women's bodies are made and the hormonal imbalances they experience throughout their lives [1]. Diabetes tends to double a woman's risk of having a heart attack twice compared to a man's two times. Women are more vulnerable to a number of diabetes-related problems, including depression, kidney disease, and vision impairment. Furthermore, it has been discovered that diabetes significantly affects women's unique circumstances, such as pregnancy, menopause, menstrual cycle, urinary tract infection, etc.

There are primarily three forms of diabetes. Diabetes that is dependent on insulin is known as type-1. In this instance, the absence of pancreatic beta cells, which are responsible for producing insulin, causes the body to not make enough of it. It is called Non-Insulin-Dependent Diabetes Type-2. This state, known as impaired glucose tolerance, is brought on by errors in the action or secretion of insulin. In this instance, the body does not make use of the insulin that is created. Because type 3 diabetes is associated with pregnancy, it exclusively affects female individuals. Although it normally ends with the birth of the child, some women may have it longer because of glucose intolerance. This form of the disease is brought on by an elevation in blood sugar in pregnant women where diabetes has not been identified earlier [8].

To identify patterns and project future events, predictive analysis makes use of both historical and current data. This combines several machine learning and deep learning algorithms, data mining techniques, and statistical techniques. Using patient records and predictive modelling, significant forecasts and judgments can be made. Physicians have determined that bad lifestyle choices, obesity, heredity, hormone imbalances, and other factors are the primary causes of diabetes in women. Therefore, a dataset with this data can be utilized to identify the diabetic attack pattern. Diabetes is currently diagnosed by laboratory procedures such as fasting blood glucose measurement and oral glucose tolerance testing. This approach takes a lot of time and resource. Because of this, the application of machine learning and deep learning techniques for early illness prediction has become more and more common. After reviewing the earlier studies, it was discovered that most of them were based on untreated, unbalanced datasets, lacked appropriate data scaling, and only used basic techniques. The goal of this work is to solve each of these concerns and create an early diabetes prediction model by applying ensemble machine learning and deep learning techniques. Furthermore, a tool for explainable artificial intelligence will offer detailed information regarding how various features affect the model.

The combination of Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) in a hybrid deep learning algorithm is often done with the goal of leveraging the strengths of both architectures to improve model performance on certain tasks. STM and GRU are both types of recurrent neural networks (RNNs) designed to capture long-term dependencies in sequential data. LSTM has a more complex structure with separate memory cells and gating mechanisms, while GRU has a simpler structure with a combined memory and gating unit. Combining them

can lead to more memory-efficient models, allowing the network to capture long-term dependencies without being overly complex. GRU has fewer parameters compared to LSTM, which can result in faster training times. The hybrid architecture may benefit from the faster convergence of GRU during training, while still incorporating the memory capabilities of LSTM. This can be particularly useful when dealing with large datasets or resource-constrained environments.

The major workflow involves such as import python packages into IDE and after loading Dataset into the Data frame, Data preprocessing is the next stage where the data gets SMOTED, the impact of an imbalanced dataset has been studied, and as a result, SMOTE has been used to balance it and stored as a dataset, both datasets gets analyzed and validated, and the next stage is Data Visualization followed by Scaling of data, this data is passed for Training and testing split followed by model building, model training, and model testing that gives you the accuracies of all models, finally comparing and declaring the accuracy is the final stage.,j

## 4.1 BACKGROUND

Gestational diabetes mellitus (GDM) is a form of hyperglycaemia that first appears during pregnancy. It is a common complication, affecting up to 15% of pregnant women worldwide. GDM can have serious consequences for both the mother and baby, including increased risk of preeclampsia, caesarean delivery, and macrosomia (large baby) [3]. Early diagnosis and treatment of GDM can help to improve outcomes for both mother and baby. Traditional prediction methods for GDM rely on clinical risk factors, such as age, family history, and pre-pregnancy body mass index (BMI). However, these methods are not always accurate, and they may miss women who are at risk of developing GDM [5].

Machine learning (ML) and deep learning (DL) are powerful tools that can be used to predict GDM with greater accuracy. ML algorithms can learn from large datasets of patient data to identify patterns that are associated with GDM. DL algorithms are a type of ML that can learn from even more complex data, such as medical images. Several studies have shown that ML and DL algorithms can be used to predict GDM with high accuracy. For example, one study [4] found that an ML algorithm could predict GDM with an area under the curve (AUC) of 0.90, which is considered excellent. Another study found that a DL algorithm could predict GDM with an AUC of 0.95.

The use of ML and DL for GDM prediction has several potential benefits. First, these methods can be used to identify women who are at risk of developing GDM before they develop symptoms. This allows for early intervention and treatment, which can help to improve outcomes for both mother and baby. Second, ML and DL algorithms can be used to personalize treatment plans for women with GDM. This can help to ensure that women receive the most effective treatment for their individual needs. Overall, ML and DL are promising new tools for the prediction and treatment of GDM. These methods have the potential to improve outcomes for both mother and baby by providing early diagnosis, personalized treatment, and improved risk stratification.

## 4.2 MOTIVATION

Deploying your Diabetes mellitus prediction in pregnant women using hybrid deep learning model on AWS cloud offers a multitude of benefits that can enhance its scalability, performance, manageability, and accessibility. The cloud deployment can be integrated with electronic health record (EHR) systems to seamlessly incorporate diabetes mellitus prediction into clinical workflows. This ensures high availability and low latency for your model, making it accessible to healthcare providers and researchers across different locations. This can lead to more accurate predictions and improved healthcare outcomes for pregnant women.

## 4.3 PROBLEM STATEMENT

Traditional screening methods for GDM, such as the Oral Glucose Tolerance Test (OGTT), can be time-consuming, inconvenient, and require multiple clinic visits. Existing prediction models often rely on limited sets of clinical features, potentially leading to inaccurate or incomplete risk assessments.

Develop a robust and efficient system for predicting GDM in pregnant women using a hybrid deep learning model. This model will leverage the strengths of different deep learning architectures to Extract complex patterns from large datasets of medical records, including demographics, laboratory tests, and pregnancy history, Learn feature representations that are highly relevant to GDM prediction. Improve prediction accuracy and generalizability compared to traditional.

## 4.4 OBJECTIVES

The overarching objective of this project is to develop a reliable and efficient system for predicting gestational diabetes mellitus (GDM) in pregnant women using a hybrid deep learning model. Design and implement a hybrid deep learning architecture that effectively combines the strengths of different deep learning techniques for GDM prediction. Training the model on a large and diverse dataset of medical records from pregnant women, including demographics, laboratory results, and pregnancy history. Optimize the model's hyper parameters to achieve high accuracy, precision, recall, and F1 score in predicting GDM. Explore strategies for deploying the model on a cloud platform to ensure scalability, accessibility, and data security. Defining data security protocols to comply with healthcare data privacy regulations like HIPAA.

Developing a plan for ongoing model maintenance and improvement based on new data and advancements in deep learning techniques. By achieving these objectives, the project aims to create a valuable tool for early and accurate GDM prediction, ultimately contributing to improved maternal and fetal health during pregnancy.

## 4.5 SCOPE OF THE PROJECT

Overall, the cloud deployment scope should focus on leveraging cloud services to create a scalable, secure, and manageable environment for your GDM prediction model. This ensures efficient operation and lays the foundation for future integration with healthcare systems.

## 5. Methodology

### 5.1 Data Extraction

The dataset employed in this study was the Pima Indian Diabetes Dataset, sourced from the National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK). The dataset comprises medical records for 768 female patients aged 21 years and older, with each record containing eight clinical attributes and a binary outcome variable indicative of diabetes status.

The selection of this dataset was justified based on its focus on physiological and demographic factors associated with diabetes mellitus, particularly relevant for the early detection of Gestational Diabetes Mellitus (GDM) among pregnant women. The availability of multiple medically significant features, such as plasma glucose concentration, BMI, and number of pregnancies, aligns well with the project's objective of developing a robust predictive system for early diabetes detection.

### 5.2 Data Preparation

#### 5.2.1 Data Cleaning and Preprocessing

An extensive data cleaning phase was conducted to ensure the quality and consistency of the dataset. Initial assessments revealed missing or ambiguous values, particularly in critical attributes such as insulin levels and blood pressure.

To address class imbalance and enhance minority class representation, the Synthetic Minority Oversampling Technique (SMOTE) was employed. SMOTE synthetically generated new instances of the minority class based on feature-space similarities, thereby balancing the dataset and mitigating model bias. Duplicates and inconsistent entries were identified and eliminated following standard validation protocols. Numerical features, including glucose concentration and BMI, were subjected to min-max normalization to ensure uniformity in scale, facilitating improved model convergence and stability during training. Given that the dataset was predominantly numerical, encoding of categorical variables was minimal. However, the binary target variable indicating diabetes presence was explicitly encoded as either 0 (non-diabetic) or 1 (diabetic).

#### 5.2.2 Feature Engineering

Feature engineering was primarily undertaken through the balancing of class distributions via SMOTE. No manual creation of new features was performed; rather, deep learning architectures were leveraged to automatically capture complex, non-linear interactions among features during model training.

### 5.3 Data Exploration

#### 5.3.1 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was conducted to gain preliminary insights into the dataset and to inform subsequent modeling decisions. Descriptive statistical measures, such as mean, median, standard deviation, and interquartile ranges, were computed for all features.

### 5.3.2 Visualization Tools and Techniques

Data visualization was performed utilizing the Matplotlib and Seaborn libraries within Python. Correlation matrices, heatmaps, boxplots, and histograms were generated to identify feature distributions, detect outliers, and ascertain inter-feature relationships. In particular, a strong positive correlation was observed between plasma glucose concentration and the presence of diabetes, guiding the emphasis placed on glucose-related variables during model development.

### 5.3.3 Trends, Patterns, and Insights

The EDA phase revealed that features such as glucose concentration, BMI, and number of pregnancies exhibited the strongest associations with the target variable. These findings reinforced the decision to prioritize these variables during model training and optimization. Furthermore, EDA confirmed the necessity of balancing the dataset to prevent model performance degradation due to class imbalance.

## 5.4 PREDICTIVE MODELING

### 5.4.1 PROPOSED ALGORITHM

This project is focused on the development and deployment of diabetes prediction models, which will use a combination of machine learning and deep learning algorithms to generate accurate findings. Pregnancies, Glucose levels, Blood Pressure, Skin Thickness, Insulin levels, BMI (Body Mass Index), DiabetesPedigreeFunction, and Age are all used in the prediction job. These characteristics serve as the base for training and testing diabetes prediction algorithms.

This project's methodology involves the use of one Hybrid Deep Learning approach and two known Machine Learning algorithms. Random Forest, XG Boost, GRU (Gated Recurrent Unit), LSTM (Long Short-Term Memory), and RNN (Recurrent Neural Network) were chosen for comparison.

The dataset split, in which the complete dataset is partitioned in a 75:25 ratio, is a significant feature of the project. This partitioning strategy uses 75% of the data to train the models, allowing them to learn patterns and relationships from the features provided. The remaining 25% of the dataset acts as the testing set, containing previously unknown data against which the models will be assessed. This careful data separation provides a thorough evaluation of the models' prediction accuracy and generalization capabilities.

a) **XG-Boost Algorithm:** XB is an enhanced type of GB that is very powerful and can be employed for regression and classification. This is accomplished by the iterative construction of consecutive decision trees, each of which corrects the errors of the others to create a robust model. Depending on the kind of work, an objective function that minimizes mean squared error or another loss function drives XGBoost's training. MSE is used for regression, and the following log loss equation is used for classification:

For Regression,

$$\text{Mean Squared Error (MSE)} = \frac{1}{num} \sum (x_j - x^{\wedge}_y)^2 \quad (1)$$

$num = \text{total no. of data points}$   
 $x_j = \text{Actual Target Value of } j\text{th data point}$   
 $\hat{x}_y = \text{Predicted value of } j\text{th data point}$

For Classification,

$$\mathbf{Log Loss} = -(x_j \log a_j + (1 - x_j) \log(1 - a_j)) \quad (2)$$



$x_j = \text{Actual class label of } j\text{th data point}$

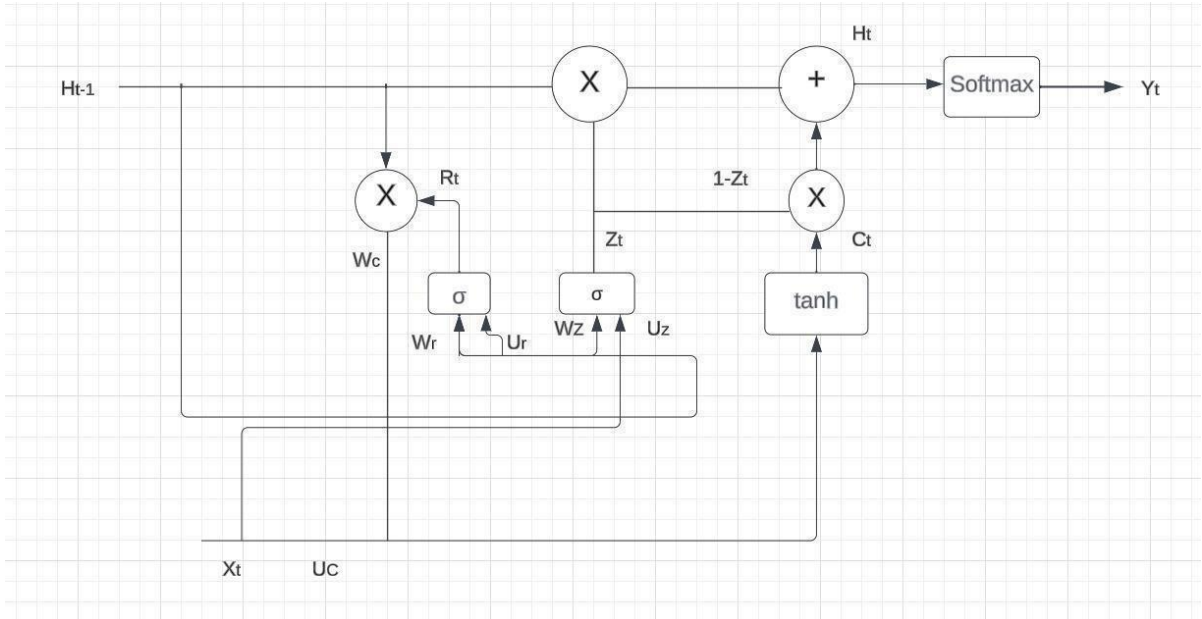
$a_j = \text{Predicted probability that } j\text{th data point belongs to class } - 1$

b) **Random Forest Algorithm:** It is one of strong ML methods utilizing MDT to estimate reliable and stable predictions possessing high prediction capacity and outperforming in classification, regression, HDM, as well as feature selection. One of the scores that are applied for measuring possible impurities of child node, is called Gini Impurity that is part of random forest algorithm. For the best class separation, it selects split with the lowest Gini impurity and the equation is written as follows:

$$\text{Gini Impurity } (I) = \sum (a_j * (1 - a_j)) \quad (3)$$

$a_j = \text{proportion of instances belonging to class } j$

c) **GRU Model:** Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture, it introduces gating mechanisms to control the flow of information within the network. Here are the mathematical equations for a simplified version of the GRU:



**Figure2:** GRU Model Architecture

**Equations:**

Given  $x_t$  as the input at time step  $t$ ,  $h_{t-1}$  as the previous hidden state, and  $r_t$  (reset gate),  $z_t$  (update gate) as the reset and update gates respectively, the GRU equations are as follows:

**Update Gate ( $z_t$ ):**

- This gate determines how much of the past information should be passed along to the future.
- $z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$

**Reset Gate ( $r_t$ ):**

- This gate determines how much of the past information to forget.
- $r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$

**Candidate Hidden State:**

- A candidate hidden state  $\tilde{h}_t$  is calculated, which is a new hidden state based on the past hidden state  $h_{t-1}$  and the current input  $x_t$ , which contributions controlled by the reset gate.
- $\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b)$

**Final Hidden State Update:**

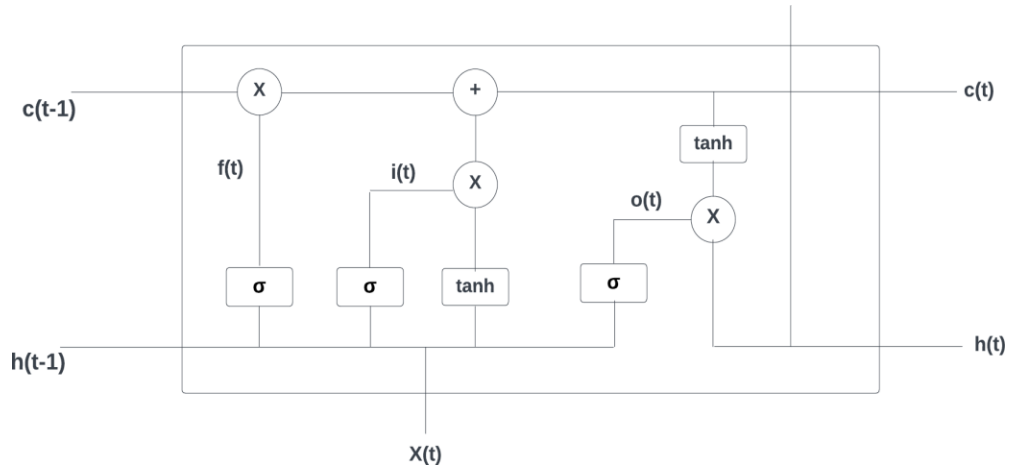
- The final hidden state is a combination of the past hidden state  $h_{t-1}$  and the new candidate hidden state  $\tilde{h}_t$ , which contributions controlled by the update gate.
- $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

**Where:**

- $W_z, W_r, W$  are weight matrices for the update gate, reset gate, and new memory content respectively.
- $b_z, b_r, b$  are bias terms
- $\sigma$  is the sigmoid activation function
- $\odot$  denotes element-wise multiplication
- $[h_{t-1}, x_t]$  represents the concatenation of  $h_{t-1}$  and  $x_t$ .

SoftMax block represents the SoftMax function. This function is used to convert the output of the network to a probability distribution over the possible outputs.

d) **LSTM Model:** LSTM networks are recurrent neural networks designed to handle long-range dependencies in sequential data, using a complex structure and mechanisms for information retention and forgetting.



**Figure3:** LSTM Model Architecture

**Equations:**

**Input Gate ( $i_t$ ):**

- The input gate updates the cell state with new information.
- $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

**Forget Gate ( $f_t$ ):**

- The forget gate decides what information from the cell state should be thrown away or kept.
- $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

**Cell State Update ( $\tilde{C}_t$ ):**

- A candidate cell state  $\tilde{C}_t$  is calculated based on the current input and the previous hidden state.
- $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

**Update Cell State ( $C_t$ ):**

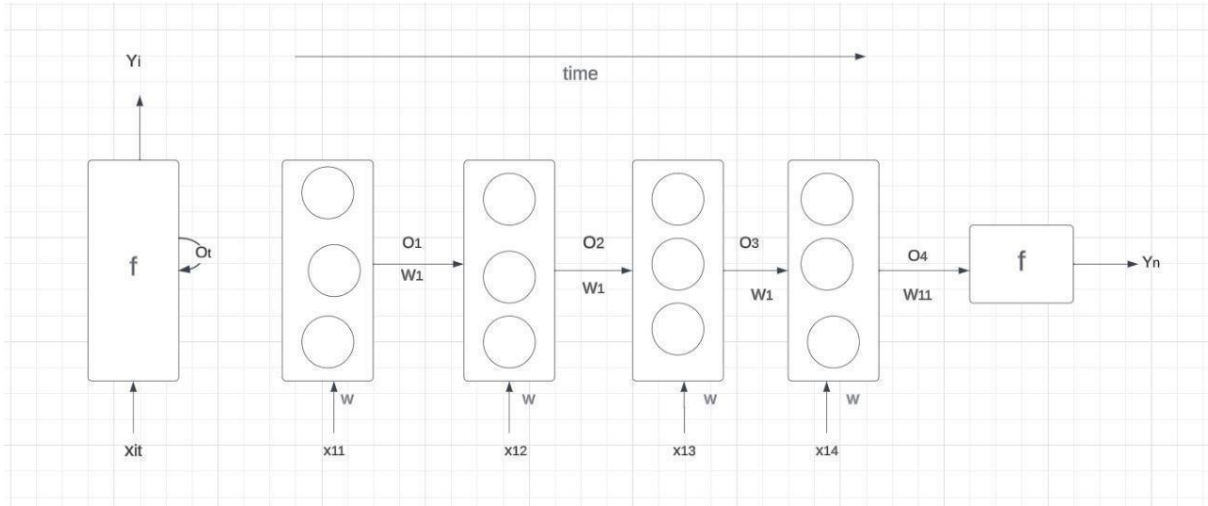
- The cell state is updated based on the forget gate, input gate, and the candidate cell state.

- $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$

### Output Gate ( $o_t$ ):

- The Output gate decides what the next hidden state  $h_t$  should be.
- $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

e) **RNN Model:** A Recurrent Neural Network (RNN) is a form of artificial neural network developed for sequential data processing. RNNs, unlike standard feedforward neural networks, feature connections that create internal cycles, allowing them to maintain a hidden state that collects information about earlier inputs in the sequence.



**Figure4:** RNN Model Architecture

### Equations:

#### 1. Hidden State Update:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$$

Here,

$h_t$  is the hidden state at time  $t$ ,

$x_t$  is the input at time  $t$ ,

$W_{hh}$  is the weight matrix for the hidden state,

$W_{xh}$  is the weight matrix for the input,

$b_h$  is the bias term,

$\tanh$  is the hyperbolic tangent activation function

#### 2. Output Calculation:

$$y_t = W_{hy} \cdot h_t + b_y$$

Here,

$y_t$  is the output at time t,

$W_{hy}$  is the weight matrix for the output,

$b_y$  is the bias term.

### Components used in the above diagram are:

X(t): This is the input to the network at time stamp t.

h(t): This is the hidden state of the network at time stamp t.

W(ih): This is the weight matrix that connects the input layer to the hidden layer. W(hh):

This is the weight matrix that connects the hidden layer to itself.

b(h): This is the bias vector for the hidden layer.

O(t): This is the output of the network at time stamp t.

**f) Hybrid GRU-LSTM Model:** A hybrid GRU (Gradient Recurrent Unit) and LSTM (Long Short-Term Memory) model combines elements from both GRU and LSTM architectures. This hybrid approach seeks to capitalize on the advantages of both types of recurrent neural networks. GRU and LSTM are both intended to overcome the vanishing gradient problem and capture long-term dependencies in sequential data, but their internal methods differ.

The mathematical equations would include computations for both the GRU and LSTM components, followed by a classification output layer. Here is an abbreviated version:

#### I. GRU Equations:

$$h_t^{GRU} = GRU(h_{t-1}^{GRU}, x_t)$$

For Computations the same equations of the GRU Model can be used.

#### II. LSTM Equations:

$$c_t^{LSTM}, h_t^{LSTM} = LSTM(c_{t-1}^{LSTM}, h_{t-1}^{LSTM}, x_t)$$

For Computations the same equations of the LSTM Model can be used.

### III. Hybrid Output for Classification:

Output = Output Layer ( $h^{GRU}_t, h^{LSTM}_t$ )

The hidden states from the GRU and LSTM components are combined in the Output Layer function.

Sigmoid Activation Function:

$$Output = \sigma(W_{out} \cdot [h^{GRU}_t, h^{LSTM}_t] + b_{out})$$

Here,  $[h^{GRU}_t, h^{LSTM}_t]$  represents the concatenation of the GRU and LSTM hidden states,  $W_{out}$  is the weight matrix,  $b_{out}$  is the bias term.

## 5.5 Evaluation Metrics

Evaluation of model performance was based on the following metrics:

Accuracy:

Measures the proportion of correct predictions over the total predictions made. This served as the primary evaluation metric to benchmark overall model performance.

Precision:

Quantifies the proportion of true positive predictions relative to all positive predictions made. It is particularly important in medical diagnoses to minimize false positives.

Recall (Sensitivity):

Measures the proportion of actual positive cases correctly identified by the model. Given the clinical significance of undetected diabetes, recall was considered a critical metric.

F1-Score:

Represents the harmonic mean of precision and recall, providing a balanced measure that is robust to class imbalance.

Area Under the Curve - Receiver Operating Characteristic (AUC-ROC) (Referenced from comparative studies):

Indicates the model's ability to distinguish between the positive and negative classes across various threshold levels.

### 5.5.1 Justification for Metric Selection

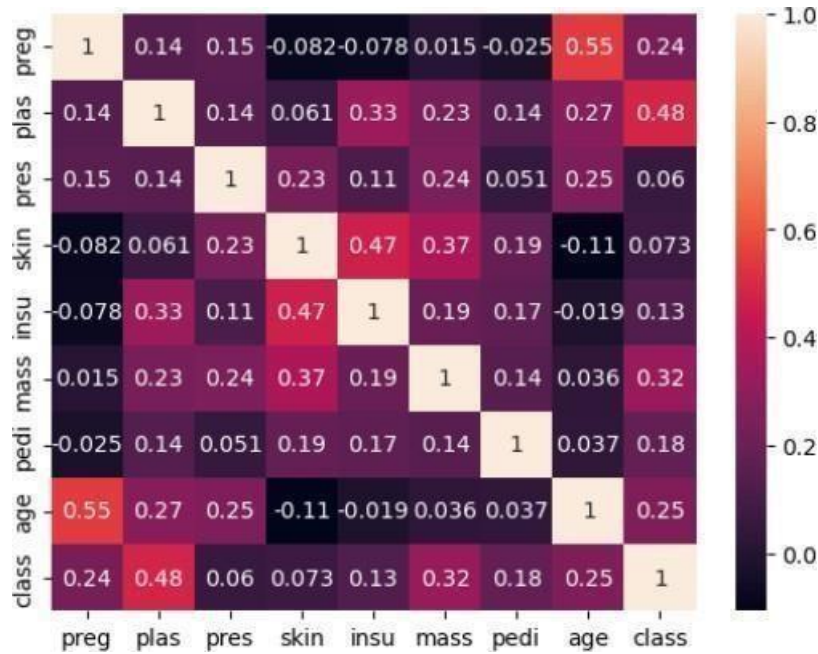
Given the healthcare context, where misclassification could have serious repercussions, emphasis was placed not solely on accuracy but also on recall and F1-score to ensure that models maintained a high level of sensitivity and balanced predictive capability.

## 5.6 PROPOSED WORK

**Step 1 (Import Modules):** Importing essential libraries that are needed for the suggested job is the initial step. Graphs are created using the seaborn and matplotlib libraries. Training and testing splitter is implemented using scikit\_learn library. Python's NumPy is used for mathematical and numerical operations, whereas pandas is used for data analysis and manipulation, usually with tabular data.

**Step 2 (Loading Dataset):** Start Loading the UCI dataset in the Training folder into the RAM during the runtime.

**Step 3 (Feature Selection):** The selection of features has a significant impact on machine learning models. Training the model with redundant data and meaningless characteristics can result in models that are faulty and provide incorrect predictions. The dataset was examined for missing or empty values. In any column, there were no missing or null values. The correlation matrix was then used to determine that no one feature was significantly correlated with the outcome. As a result, no feature was eliminated for correlation. Finally, outliers were discovered. Box plot approach was used in some columns. Outliers were left in the dataset because removing them from such a small-scale dataset resulted in worse performance. The dataset was therefore constructed with the relevant features.



**Figure7:** Correlation in the Dataset

**Step 4 (Over Sampling Using SMOTE):** After the loading of the dataset, the dataset was thoroughly analysed. The dataset was discovered to be unbalanced. In the result class, the non-diabetic to diabetes ratio was 1.86 to 1. Oversampling was used to balance out the dataset to alleviate this problem. Oversampling was accomplished using the Synthetic Minority Oversampling Technique (SMOTE). It evened out the distribution of classes by more

frequently reproducing minority class samples, as random instances of minorities were mixed with smote to produce new instances. Following SMOTE, the diabetic to nondiabetic case ratio was 1:1, achieving our oversampling goal. The total number of occurrences became from 768 to 1000.

**Step 5 (Data Visualization):** This step is used to know the hidden relationship or pattern between the attributes. Bar plots display the distribution of categorical data; pair plots reveal relationships between pairs of variables; heatmaps visualize correlations or patterns in a dataset, and pie charts represent the composition of a whole as parts.

**Step 6 (Standardization):** As distinct attributes were measured on different scales, scaling them would level the playing field for all data points. The goal of scaling in this case is to reduce all features to a single scale. Because the majority of the algorithms utilized were gradient descent-based, feature scaling would minimize the step size, resulting in speedier outputs. Standardization was implemented in the dataset for scaling.

**Step 7 (Train–Test Split):** Using the scikit-learn library to access the 'train\_test\_split' function, which enables us to neatly divide the dataset into two parts: one for training and the other for testing. This Project gives the results of 75:25 ration split.

**Step 8 (Model Building):** Bring in the required model libraries, including Random Forest, XG-Boost, GRU, LSTM, RNN. Assign each of these models to its own variable for easy access.

**Step 9 (Model Training):** In this stage, the data is passed in the Dependent variable and independent variable separately into the model to train our model with our dataset, where it will find hidden patterns that help it in its prediction of classes for new/unfamiliar inputs.

**Step 10 (Testing the Model):** Provide input with the independent testing dataset and allow the model to predict target values. In this case, the model utilizes its learning from the training process to predict likely outcomes using the new and previously unseen data.

**Step 11 (Accuracy calculation):** Matching the actual world values of the dependent characteristic with those our model anticipated. With a tool such as a confusion matrix, can be useful to judge how good or not each model's predictions perform. This enables to determine the level of success of our model within its practical relevance setting.

**Step 12 (Accuracy Comparison and best model declaration):** After obtaining the accuracy scores of all the models, then move to the final practical decision. Choose the best one or two exemplary models showing highest accuracy, and take this as the high rated models by this dataset. The best performing models are the ones that will be used in predicting and drawing meaningful conclusions in future.



## 6. Experiments

### 6.1 summary of experimental datasets

<https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

The dataset used for the project is Pima Indian Diabetes Dataset from National Institutes of Diabetes and Digestive and Kidney Diseases. There is total 768 patients and all of them are females of at least 21 years old. The dataset has eight predictor variables and a target variable. There were 500 cases of non-diabetic patients compared to 278 diabetic patients. All the data of predictor variables were numerical and the target variable were categorical. The predictor variables are:

- Pregnancy: Number of times the patient had been pregnant
- Glucose in blood: Glucose concentration in 2 hours of oral glucose consumption test
- Blood Pressure: Diastolic blood pressure in (mm Hg)
- Fold thickness of skin (Triceps)
- Insulin: Serum insulin in body after 2 hours of food intake. ( $\mu\text{h/ml}$ )
- BMI: Body mass index of patient.
- Diabetes Pedigree Function: Indicates the function which scores likelihood of diabetes based on family history or genetics.
- Age of patient

The original dataset has an uneven composition. The Synthetic Minority Over-Sampling Technique (SMOTE) technique is applied to the imbalanced dataset to produce a balanced dataset. This technique employs oversampling. Any classes with insufficient rows are given extra rows such that the dataset's total number of rows for each class label is verified to be equal, or more or fewer. Asymmetry occurs in an uneven dataset. Multiple factors can impact the accuracy of the models resulting from an imbalanced dataset with a skewed class distribution.

It is crucial to weigh the data accordingly. It is possible to improve the findings' accuracy by oversampling the positive class label. This study uses SMOTE to do oversampling. The minority class occurs more frequently when using the SMOTE technique.

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6.000000	148.000000	72.000000	35.000000	0.00000	33.600000	0.627000	50.000000	tested_positive
1	1.000000	85.000000	66.000000	29.000000	0.00000	26.600000	0.351000	31.000000	tested_negative
2	8.000000	183.000000	64.000000	0.000000	0.00000	23.300000	0.672000	32.000000	tested_positive
3	1.000000	89.000000	66.000000	23.000000	94.00000	28.100000	0.167000	21.000000	tested_negative
4	0.000000	137.000000	40.000000	35.000000	168.00000	43.100000	2.288000	33.000000	tested_positive

After SMOTE, the ratio was 1:1 for diabetic to nondiabetic cases, which fulfilled our target of oversampling. Total number of instances increased from 768 to 1000. 500 cases of non-diabetic patients and 268 diabetic patients has become 500 diabetic patients.

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6.000000	148.000000	72.000000	35.000000	0.00000	33.600000	0.627000	50.000000	tested_positive
1	1.000000	85.000000	66.000000	29.000000	0.00000	26.600000	0.351000	31.000000	tested_negative
2	8.000000	183.000000	64.000000	0.000000	0.00000	23.300000	0.672000	32.000000	tested_positive
3	1.000000	89.000000	66.000000	23.000000	94.00000	28.100000	0.167000	21.000000	tested_negative
4	0.000000	137.000000	40.000000	35.000000	168.00000	43.100000	2.288000	33.000000	tested_positive
...	...	...	...	...	...	...	...	...	...
994	4.112680	128.273325	86.816247	40.561308	99.59885	35.405738	0.326197	36.960990	tested_positive
995	0.887431	128.536006	70.703019	0.000000	0.00000	35.695255	0.614867	48.963615	tested_positive
996	6.736276	189.134854	84.803245	0.000000	0.00000	35.755130	0.201246	61.978155	tested_positive
997	1.532892	164.001414	74.050966	33.355244	0.00000	44.879901	0.396190	32.674601	tested_positive
998	8.178019	128.425088	63.531894	0.000000	0.00000	33.405517	0.520211	46.134645	tested_positive

999 rows × 9 columns

## 6.2 Parameter Tuning

### 6.2.1 Hyperparameter Tuning Strategy

Hyperparameter optimization was critical to maximizing model performance. The following tuning approaches were adopted:

**Grid Search:**

An exhaustive search over specified parameter grids was performed, particularly for Random Forest and XGBoost models. Grid Search systematically evaluated combinations of hyperparameter values to identify optimal settings.

**Manual Adjustment:**

For deep learning models (GRU, LSTM, GRU+LSTM, and RNN), hyperparameters such as number of epochs, batch size, dropout rates, and learning rates were manually fine-tuned based on validation performance during training iterations

**Cross-Validation:**

A 5-fold cross-validation technique was employed for traditional machine learning models to assess the generalizability of hyperparameter settings.

### 6.2.2 Optimal Parameters Selection

The optimal hyperparameters were determined by selecting the configurations that yielded the highest validation accuracy and F1-score without evidence of overfitting.

In deep learning experiments, early stopping techniques were additionally utilized to terminate training if the validation loss ceased improving over a set number of epochs.

Selected Optimal Parameters Included:

Random Forest:

Number of trees = 100

Maximum depth = 10

XGBoost:

Learning rate = 0.01

Max depth = 8

Subsample ratio = 0.8

GRU+LSTM Hybrid Model:

Learning rate = 0.001

Batch size = 32

Epochs = 100

Dropout = 0.2

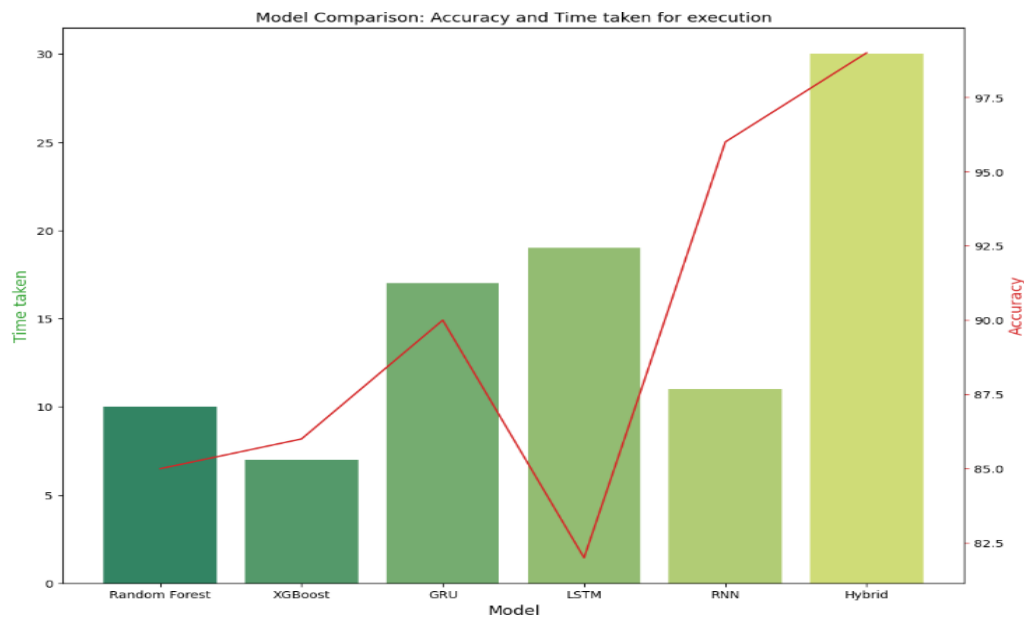
## 6.3 RESULTS AND DISCUSSION

XG-BOOST achieved an accuracy of 86%, with precision, recall, and F1-score of around 0.86. This implies a balanced performance across several evaluation parameters, demonstrating XG- BOOST's robustness in predicting diabetes. RANDOM FOREST achieved an accuracy of 85%, with constant precision, recall, and F1-score values of 0.85. The model's consistent performance across these metrics suggests its dependability in predicting diabetes cases.

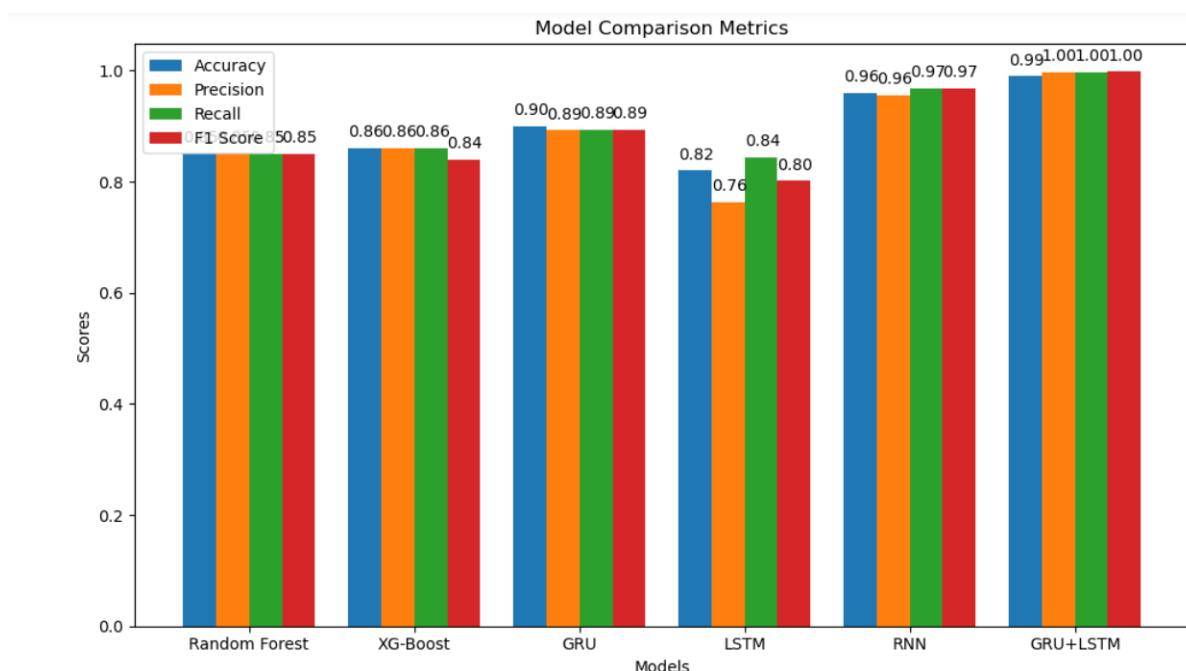
GRU, a recurrent neural network, outperformed other models with a 90% accuracy rate. It had a constant high precision, recall, and F1-score of 0.8930, 0.8933, and 0.8933, respectively. This shows that GRU excelled at both detecting positive situations properly and minimizing false positives. Another type of recurrent neural network, LSTM, reached 82% accuracy. Its precision (0.7630) was, however, slightly lower than that of other models, indicating a larger probability of false positives. Despite this, LSTM demonstrated a strong balance of recall (0.8443) and F1-score (0.8016), highlighting its capacity to successfully catch true positive cases.

RNN, a fundamental type of recurrent neural network, performed admirably, with an accuracy of

96%. It had an F1-score of 0.9678 because to its remarkable precision (0.9554) and recall (0.9678). These findings show that RNN performed admirably in accurately predicting positive situations while avoiding false positives and false negatives. The combination of GRU and LSTM, denoted as GRU+LSTM, outperformed all other models with 99% accuracy. Its precision, recall, and F1-score were all exceptionally high, at 0.997 and 0.998, respectively, demonstrating its remarkable predicting powers. The combined cumulative effect of GRU and LSTM appears to have considerably enhanced the model's overall performance. The Below Diagram Shows the Comparison between Accuracy and time taken for execution for each model.



The visualizations compare the performance of various models in terms of accuracy and time for execution. The accuracy percentages for models such as Random Forest, XG-Boost, GRU, LSTM, RNN, and GRU+LSTM are depicted in the line plot, highlighting their relative strengths. Meanwhile, the bar chart shows how much time each model takes to process. Notably, GRU+LSTM has the best accuracy at 99%, whereas Random Forest and XG-Boost have comparable accuracy. The bar chart provides insights into each model's computational efficiency, assisting in the selection of the best appropriate model based on both accuracy and time for execution.



The bar chart compares the performance of different models using key metrics. Each bar represents a model, and the height of the bars illustrates their corresponding accuracy, precision, recall, and F1 score. The models, including Random Forest, XG-Boost, GRU, LSTM, RNN, and GRU+LSTM, exhibit varying levels of success across these metrics. For instance, GRU+LSTM stands out with the highest accuracy of 99%. The visual representation provides a quick and clear comparison, aiding in the selection of the most effective model based on the desired performance criteria.

### 6.3.1 Practical Contributions

The practical contributions of the experimental results are significant:

#### Clinical Implications:

The development of a near-accurate predictive system for GDM can enable healthcare providers to identify high-risk pregnant women early in the gestational period, facilitating timely medical intervention and improved maternal-fetal outcomes.

#### Operational Efficiency:

By automating the early prediction process through machine learning and deep learning systems, healthcare facilities can potentially reduce the need for costly, time-intensive laboratory-based diagnostic procedures.

#### Research Advancement:

This study provides empirical validation of hybrid deep learning models in structured clinical datasets, contributing to the growing body of literature advocating for deep learning applications in medical diagnostics.

## 7. Conclusions

### 7.1 Summary of Key Project Outcomes

This study developed an advanced predictive system for the early detection of Gestational Diabetes Mellitus (GDM) in pregnant women by employing a combination of hybrid deep learning models and traditional machine learning classifiers.

The key contributions and findings of the project are summarized as follows:

A comprehensive preprocessing pipeline was implemented, including data balancing using SMOTE, normalization, and feature analysis, ensuring data quality and model readiness.

Multiple models were developed and compared, including Random Forest, XGBoost, LSTM, GRU, RNN, and a hybrid GRU+LSTM architecture.

Among the models tested, the Hybrid GRU+LSTM model achieved the highest predictive performance, with an accuracy of 99%, demonstrating its superior capability in capturing complex, non-linear interactions among features.

Traditional ensemble models such as Random Forest and XGBoost also showed strong, albeit lower, performance with accuracies around 85-86%, validating the effectiveness of classical machine learning for structured healthcare data.

The outcomes confirm the hypothesis that hybrid deep learning techniques offer significant advantages for predictive analytics in the medical domain, even when applied to moderately sized clinical datasets.

### 7.2 Broader Implications for Stakeholders and Practical Utility

The findings of this research have important implications for various stakeholders:

#### Healthcare Providers:

Early and accurate prediction of GDM allows clinicians to implement timely interventions, reducing adverse outcomes for both mothers and infants. The deployment of such predictive models in clinical settings can streamline risk stratification, reduce unnecessary diagnostic testing, and personalize patient management strategies.

#### Public Health Policymakers:

Predictive analytics models can assist in resource allocation, particularly in prenatal care programs, enabling focused attention on high-risk groups and thus improving overall maternal and child health outcomes.

#### Medical Researchers:

The study reinforces the utility of deep learning architectures, particularly hybrid models, for handling healthcare datasets that traditionally exhibit challenges such as limited size, class imbalance, and multi-collinearity.

#### Technology Providers and Startups:

The deployment of such predictive systems on cloud platforms (e.g., AWS) opens avenues for scalable, secure, and accessible healthcare decision support tools. This can democratize early disease detection technologies, making them available even in resource-constrained settings.

### 7.3 Future Research Directions and Possible Improvements

While the study achieved excellent predictive performance, several avenues for future research and enhancements are identified:

#### Expansion of Dataset Size:

Larger and more diverse datasets incorporating demographic and genetic information could be utilized to further validate and generalize the model across different populations.

#### Incorporation of Time-Series Data:

Future models could integrate longitudinal patient data (e.g., glucose readings over time during pregnancy) to enhance the predictive capability of recurrent architectures.

#### Explainable AI (XAI) Approaches:

Integrating interpretability tools such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) can help provide healthcare practitioners with transparent reasoning behind predictions, promoting trust and adoption.

#### Multi-Modal Data Integration:

Future studies could explore the integration of clinical data with imaging data (e.g., ultrasound scans) or lifestyle factors to create more holistic predictive models.

#### Real-World Clinical Trials:

Validating the model performance in live clinical environments through pilot studies would be critical for demonstrating practical utility and assessing real-world effectiveness.

#### Optimization for Deployment:

Building lightweight, resource-efficient models suitable for deployment on edge devices (e.g., hospital workstations, mobile health apps) could maximize accessibility.

## 8. References

- [1] Kirti Kangra, Jaswinder Singh, “Comparative analysis of predictive machine learning algorithms for diabetes mellitus – 2023”.
- [2] Akkur E., Türk F. Optimized machine learning based predictive diagnosis approach for diabetes mellitus. *J Med Palliat Care / JOMPAC / jompac*. 2023; 4(4): 270-276.
- [3] F. Muntasir, M. S. Anower and M. Nahiduzzaman, "Majority Voting Ensemble Approach for Predicting Diabetes Mellitus in Female Patients from Unbalanced Dataset," 2023 International Conference on Electrical, Computer and Communication Engineering (ECCE), Chittagong, Bangladesh, 2023, pp. 1-6, doi: 10.1109/ECCE57851.2023.10101629.
- [4] d. Rifatul Islam, Semonti Banik, Kazi Naimur Rahman, Mohammad Mizanur Rahman, A comparative approach to alleviating the prevalence of diabetes mellitus using machine learning, *Computer Methods and Programs in Biomedicine Update*, Volume 4, 2023, 100113, ISSN 2666-9900, <https://doi.org/10.1016/j.cmpbup.2023.100113>.
- [5] S. Mahajan, P. K. Sarangi, A. K. Sahoo and M. Rohra, "Diabetes Mellitus Prediction using Supervised Machine Learning Techniques," 2023 International Conference on Advancement in Computation & Computer Technologies (InCACCT), Gharuan, India, 2023, pp. 587-592, doi: 10.1109/InCACCT57535.2023.10141734.
- [6] Laila UE, Mahboob K, Khan AW, Khan F, Taekeun W. An Ensemble Approach to Predict Early-Stage Diabetes Risk Using Machine Learning: An Empirical Study. *Sensors (Basel)*. 2022 Jul 13;22(14):5247. doi: 10.3390/s22145247. PMID: 35890927; PMCID: PMC9324493.
- [7] Siddhartha Suprasad Mohanty , Babita Majh, “Diabetic Mellitus Prediction Using Deep Learning – 2022”.
- [8] Houri, O., Gil, Y., Chen, R. et al. Prediction of Type 2 Diabetes Mellitus According to Glucose Metabolism Patterns in Pregnancy Using a Novel Machine Learning Algorithm. *J. Med. Biol. Eng.* 42, 138–144 (2022). <https://doi.org/10.1007/s40846-022-00685-9>
- [9] Victor Chang, Meghana Ashok Ganatra, Karl Hall, Lewis Golightly, Qianwen Ariel Xu, An assessment of machine learning models and algorithms for early prediction and diagnosis of diabetes using health indicators, *Healthcare Analytics*, Volume 2, 2022, 100118, ISSN 2772-4425, <https://doi.org/10.1016/j.health.2022.100118>.



## CODE AND ITS OUTPUT SCREENSHOTS

### PYTHON CODE FOR FLASK FRAMEWORK

Code:

```
from flask import Flask, render_template,  
  
request from diabetes import calc  
  
app = Flask(__name__)# interface between my server and my application wsgi  
  
#import pickle  
import json  
import requests  
  
#model = pickle.load(open(r'C:\Users\nagap\Downloads\Updated files\model.pkl','rb'))  
  
@app.route('/')#binds to an url
```

```

def helloworld():
    return render_template("index.html")

@app.route("/prediction",
methods=["GET"]) def redirect_internal():
    return render_template("/prediction.html")

@app.route('/predicted', methods =['POST'])#binds to
an url def login():
    p
    =request.form["i1"]
    q=
    request.form["i2"]
    r= request.form["i3"]
    s= request.form["i4"]
    t= request.form["i5"]
    u=
    request.form["i6"]
    v=
    request.form["i7"]
    w=
    request.form["i8"]

    output =
    calc(float(p),float(q),float(r),float(s),float(t),float(u),float(v),float(w))
    print(output)

    data = {"data":
f"{output[0]},{output[1]},{output[2]},{output[3]},{output[4]},{output[5]},{output[6]},{output[7]}"
}
    r = requests.post("https://5f7fhumgbg.execute-api.ap-south-1.amazonaws.com/diabetes/api-
diabetes",json = data)
    response_content = r.content.decode('utf-8')
    # output =
    model.predict([calc(float(p),float(q),float(r),float(s),float(t),float(u),float(v),float(w))]) #
    print(output)
    try:
        out =
        float(response_content)
    except ValueError:
        print("Error: The API response is not a valid
        number.") out = None

    if out is not
    None: if
    out > 2:
        op =
        "Diabetic" else:
        op = "Non-Diabetic"

    print(response_content)

    return render_template("prediction.html",submit_to_check_result = "The prediction says person is "

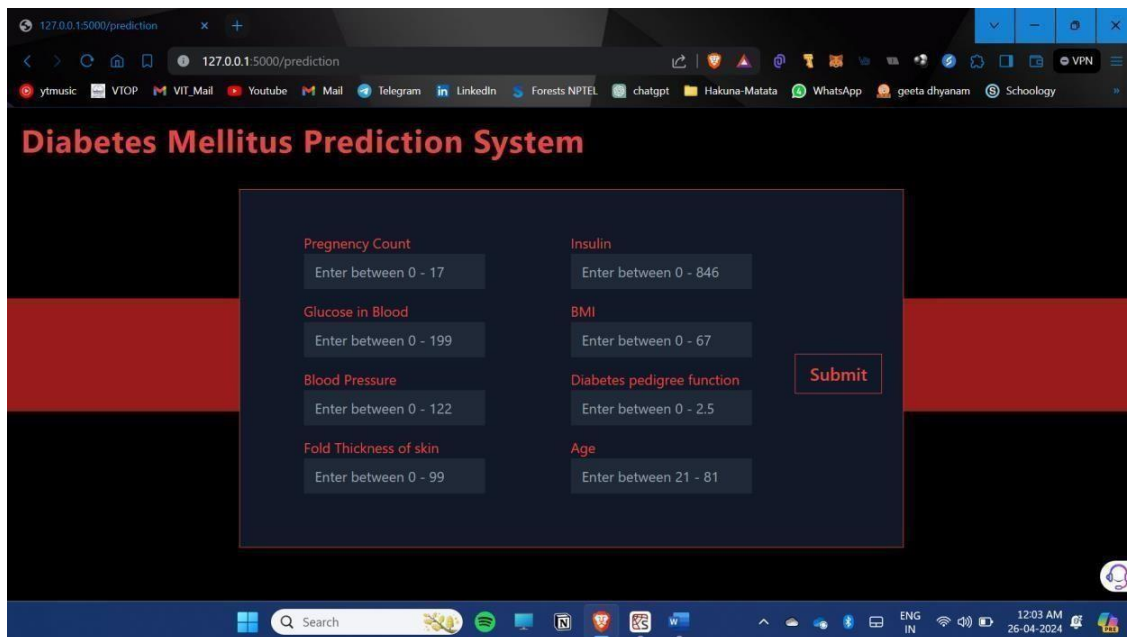
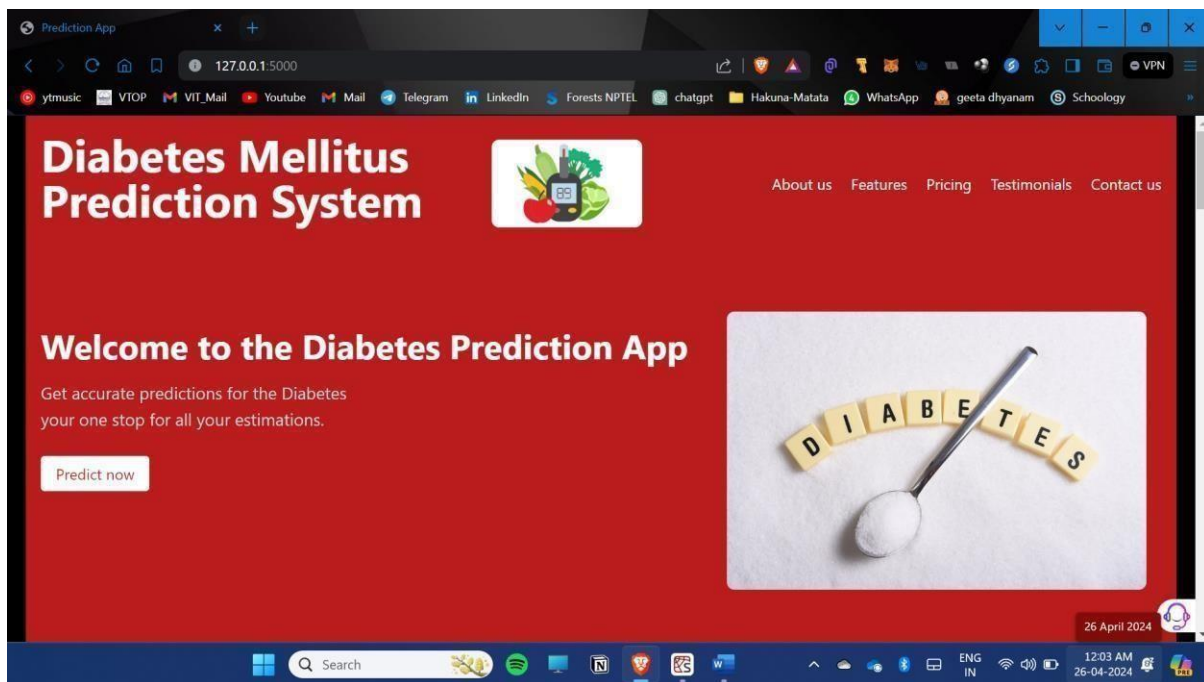
```

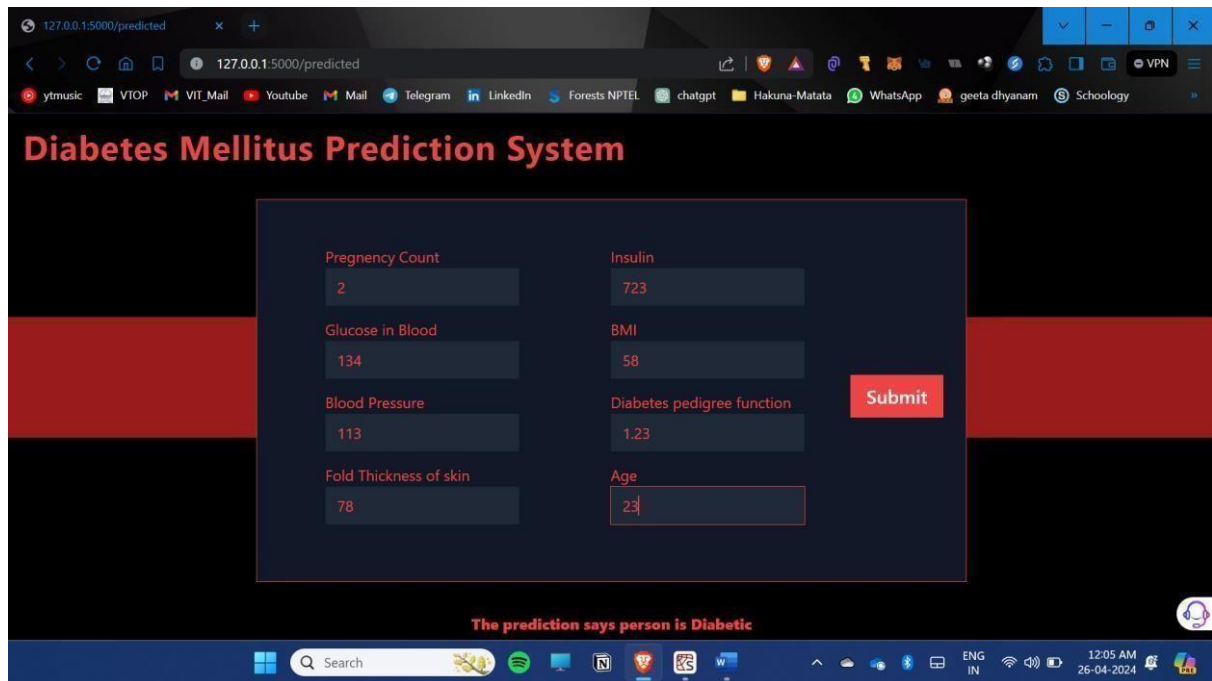
```
+ str(op))

#@app.route('/admin')#binds to an
url #def admin():
    # return "Hey Admin How are you?"

if __name__ == '__main__':
    app.run(debug=False)
```

## 5.1 OUTPUT WITH SCREENSHOTS:





## DEPLOYMENT IN AWS CLOUD WITH SCREENSHOTS

### CODE:

```
# In[1]:
import numexpr
import sagemaker
import boto3
from sagemaker.amazon.amazon_estimator import get_image_uri
import sagemaker
from sagemaker import RandomCutForest
from sagemaker.serializers import CSVSerializer
import sagemaker.amazon.common as smac
# from sagemaker.amazon.record_pb2 import CounterExample
# from sagemaker.amazon.amazon_estimator import RecordEncoder

# In[2]:

from sagemaker.session import s3_input, Session

# In[3]:
bucket_name = 'diabetesNaniPredictions' # <--- CHANGE THIS VARIABLE TO A UNIQUE NAME
FOR YOUR BUCKET
my_region = boto3.session.Session().region_name # set the region of the instance
print(my_region)
```

```
# In[4]:
```

```
AWS_REGION = "ap-south-1"
client = boto3.client("s3", region_name=AWS_REGION)
bucket_name = "diabetesnanipredictionsnani"
location = {'LocationConstraint': AWS_REGION}
response = client.create_bucket(Bucket=bucket_name, CreateBucketConfiguration=location)
print("Amazon S3 bucket has been created")
```

```
# In[ ]:
```

```
s3 = boto3.resource('s3')
# try:
#     if my_region == 'ap-south-1':
#         s3.create_bucket(Bucket=bucket_name)
#     print('S3 bucket created successfully')
# except Exception as e:
#     print('S3 error: ',e)
```

```
# In[ ]:
```

```
# set an output path where the trained model will be saved
prefix = 'random-forest-as-a-built-in-algo'
output_path = 's3://{}/{}/output'.format(bucket_name, prefix)
print(output_path)
# ##### Downloading The Dataset And Storing in S3
```

```
# In[ ]:
```

```
import pandas as pd
import urllib
# try:
#     urllib.request.urlretrieve ("https://d1.awsstatic.com/tmt/build-train-deploy-machine-learning-
model-sagemaker/bank_clean.27f01fbbdf43271788427f3682996ae29ceca05d.csv", "bank_clean.csv")
# print('Success: downloaded bank_clean.csv.')
# except Exception as e:
#     print('Data load error: ',e)
try:
    data = pd.read_csv('./diabetesdata.csv')
```

```
print('Success: Data loaded into dataframe.')
except Exception as e:
    print('Data load error: ',e)
```

```
# In[8]:
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
# In[9]:
```

```
data
```

```
# In[10]:
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data["class"] = le.fit_transform(data["class"])
data["class"]
```

```
# In[11]:
```

```
data
```

```
# In[12]:
```

```
#extracting numerical columns values
x_independent = data.iloc[:, :-1]
y_dependent=data.iloc[:, 8:9]
```

```
# In[13]:
```

```
y_dependent
```

```
# In[14]:
```

```
name=x_independent.columns
name
```

```
# In[15]:
```

```
names2=y_dependent.columns
names2
```

```

# In[16]:
#Normalisation
from sklearn.preprocessing import MinMaxScaler

# In[17]:
scale=MinMaxScaler()

# In[18]:
X_scaled=scale.fit_transform(x_independent)

# In[19]:
X=pd.DataFrame(X_scaled,columns=name)

# In[20]:

y_dependent=pd.DataFrame(y_dependent,columns=names2)

# In[21]:

y_dependent

# In[22]:

X

# In[23]:

x=X #independent values
y=y_dependent

# In[24]:
y

# In[25]:
d_new = pd.concat([x,y],axis=1)
data = d_new

# In[26]:

data

```



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

```
### Train Test split
```

```
import numpy as np
train_data, test_data = np.split(data.sample(frac=1, random_state=38), [int(0.75 * len(data))])
print(train_data.shape, test_data.shape)
```

```
# In[28]:
```

```
test_data
```

```
# In[29]:
```

```
### Saving Train And Test Into Buckets
```

```
## We start with Train Data
```

```
import os
```

```
pd.concat([train_data['class'], train_data.drop(['class'],
                                                axis=1)],
```

```
        axis=1).to_csv('train.csv', index=False, header=False)
```

```
boto3.Session().resource('s3').Bucket(bucket_name).Object(os.path.join(prefix,
'train/train.csv')).upload_file('train.csv')
```

```
s3_input_train = sagemaker.TrainingInput(s3_data='s3://{}/{}/train'.format(bucket_name, prefix),
content_type='text/csv', distribution='ShardedByS3Key')
```

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

```
# Test Data Into Buckets
```

```
pd.concat([test_data['class'], test_data.drop(['class'], axis=1)], axis=1).to_csv('test.csv', index=False,
header=False)
```

```
boto3.Session().resource('s3').Bucket(bucket_name).Object(os.path.join(prefix,
'test/test.csv')).upload_file('test.csv')
```

```
s3_input_test = sagemaker.TrainingInput(s3_data='s3://{}/{}/test'.format(bucket_name, prefix),
content_type='text/csv', distribution='ShardedByS3Key')
```

```
# ### Building Models RandomcutForest- Inbuilt Algorithm
```

```
# In[31]:
```

```
# this line automatically looks for the XGBoost image URI and builds an XGBoost container.
```

```
# specify the repo_version depending on your preference.
```

```

# container = get_image_uri(boto3.Session().region_name,
#                             'xgboost',
#                             repo_version='1.0-1')

# In[32]:

from sagemaker.amazon.amazon_estimator import image_uris
container = image_uris.retrieve(region=boto3.Session().region_name, framework='randomcutforest',
                                version='1.0-1')

# In[33]:

## Initialize hyperparameters for multi-class classification
# hyperparameters = {
#     "max_depth": "5",
#     "eta": "0.2",
#     "gamma": "4",
#     "min_child_weight": "6",
#     "subsample": "0.7",
#     "objective": "multi:softmax", # Set objective to multi-class softmax
#     "num_class": 8, # Number of classes in your output column
#     "num_round": 50
# }

# In[34]:

# define the hyperparameters for
hyperparameters = {
    'num_trees': 100, # Number of trees in the forest
    'eval_metrics': 'accuracy', # Evaluation metric
    'feature_dim': 8 # number of features in your training data
}

# In[35]:

## construct a SageMaker estimator that calls the xgboost-container
# estimator = sagemaker.estimator.Estimator(image_uri=container, #
#                                             hyperparameters=hyperparameters,
#                                             role=sagemaker.get_execution_role(),
#                                             instance_count=1,
#                                             instance_type='ml.m5.2xlarge',

```

```
#             volume_size=5, # 5 GB
#             output_path=output_path,
#             use_spot_instances=True,
#             max_run=300,
#             max_wait=600)
```

```
# In[36]:
```

```
# construct a SageMaker estimator
estimator = sagemaker.estimator.Estimator(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.m5.2xlarge',
    volume_size=5,
    output_path=output_path,
    use_spot_instances=True,
    max_run=300,
    max_wait=600,
    hyperparameters=hyperparameters,
    image_uri=container)
```

```
# In[37]:
```

```
estimator.fit({'train': s3_input_train})
```

```
# ### Deploy Machine Learning Model As Endpoints
```

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

```
xgb_predictor = estimator.deploy(initial_instance_count=1,instance_type='ml.m4.xlarge')
```

```
# ##### Prediction of the Test Data
```

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

```
pip install --upgrade sagemaker
```

```
# In[40]:
```

```
from sagemaker.serializers import CSVSerializer
test_data_array = test_data.drop(['class'], axis=1).values #load the data into an array
```

```

xgb_predictor.content_type = 'text/csv' # set the data type for an inference
xgb_predictor.serializer = CSVSerializer() # set the serializer type
predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!
predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
print(predictions_array.shape)

```

# In[43]:

```

result=xgb_predictor.predict([[0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]])
result

```

# In [ ]:

```

result.decode('utf-8')

```

# In [ ]:

```

result[]

```

```

predictions_array

```

# In [ ]:

```

cm = pd.crosstab(index=test_data['NObeyesdad'], columns=np.round(predictions_array),
rownames=['Observed'], colnames=['Predicted'])
tn = cm.iloc[0,0]; fn = cm.iloc[1,0]; tp = cm.iloc[1,1]; fp = cm.iloc[0,1]; p =
(tp+tn)/(tp+tn+fp+fn)*100
print("\n{0:<20}{1:<4.1f}%\n".format("Overall Classification Rate: ", p))
print("{0:<15}{1:<15}{2:>8}".format("Predicted", "No Purchase", "Purchase"))
print("Observed")
print("{0:<15}{1:<2.0f}% ({2:<}){3:>6.0f}% ({4:<})".format("No Purchase", tn/(tn+fn)*100,tn,
fp/(tp+fp)*100, fp))
print("{0:<16}{1:<1.0f}% ({2:<}){3:>7.0f}% ({4:<}) \n".format("Purchase", fn/(tn+fn)*100,fn,
tp/(tp+fp)*100, tp))

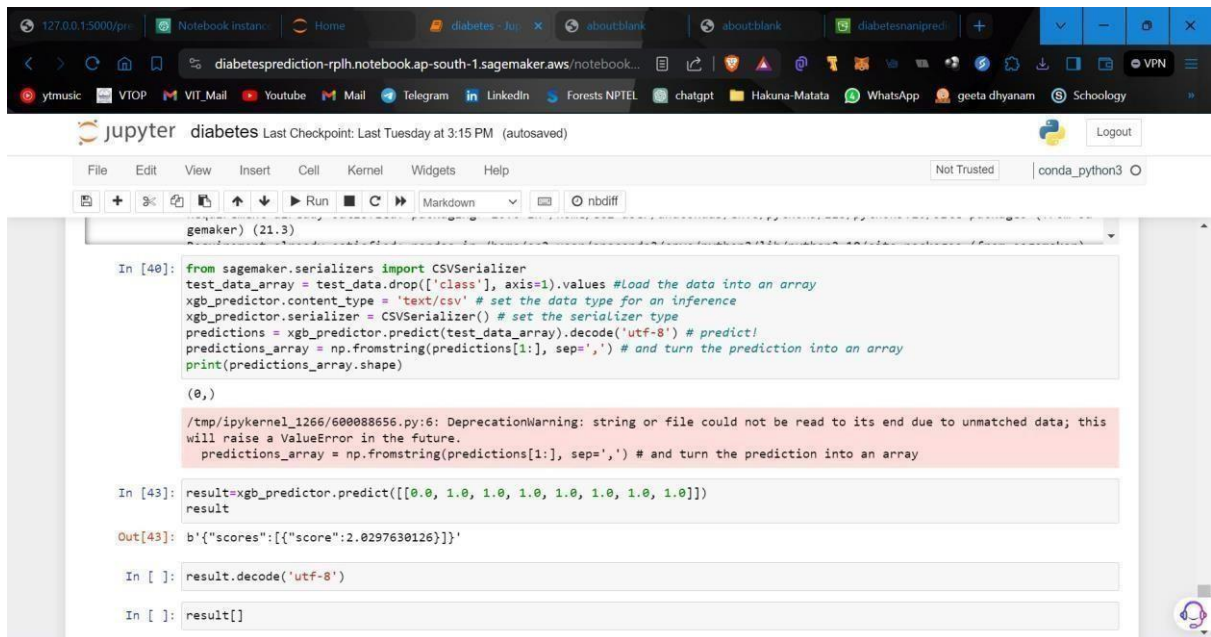
```

```
# ##### Deleting The Endpoints
```

```
# In[ ]:
```

```
sagemaker.Session().delete_endpoint(xgb_predictor.endpoint)
bucket_to_delete = boto3.resource('s3').Bucket(bucket_name)
bucket_to_delete.objects.all().delete()
```

## OUTPUT SCREENSHOTS:



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [40]: from sagemaker.serializers import CSVSerializer
test_data_array = test_data.drop(['class'], axis=1).values #Load the data into an array
xgb_predictor.content_type = 'text/csv' # set the data type for an inference
xgb_predictor.serializer = CSVSerializer() # set the serializer type
predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!
predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
print(predictions_array.shape)

(0,)
```

Below the code, there is a red warning box:

```
/tmp/ipykernel_1266/690088656.py:6: DeprecationWarning: string or file could not be read to its end due to unmatched data; this
will raise a ValueError in the future.
  predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
```

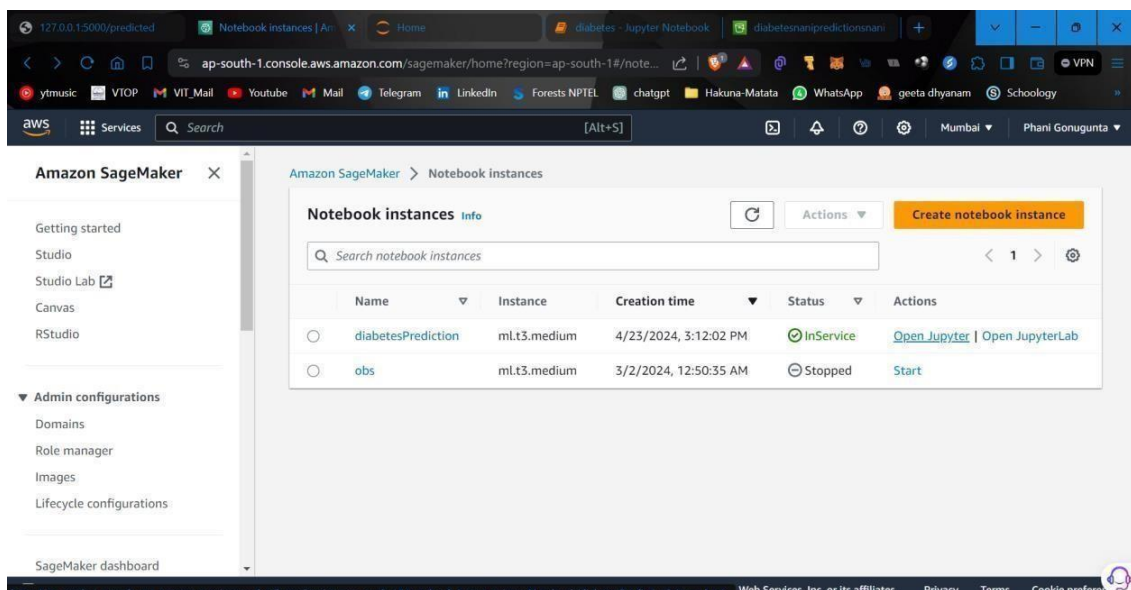
The output of the code is:

```
In [43]: result=xgb_predictor.predict([[0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]])
result
Out[43]: b'{"scores":[{"score":2.0297630126}]}'

In [ ]: result.decode('utf-8')

In [ ]: result[]
```

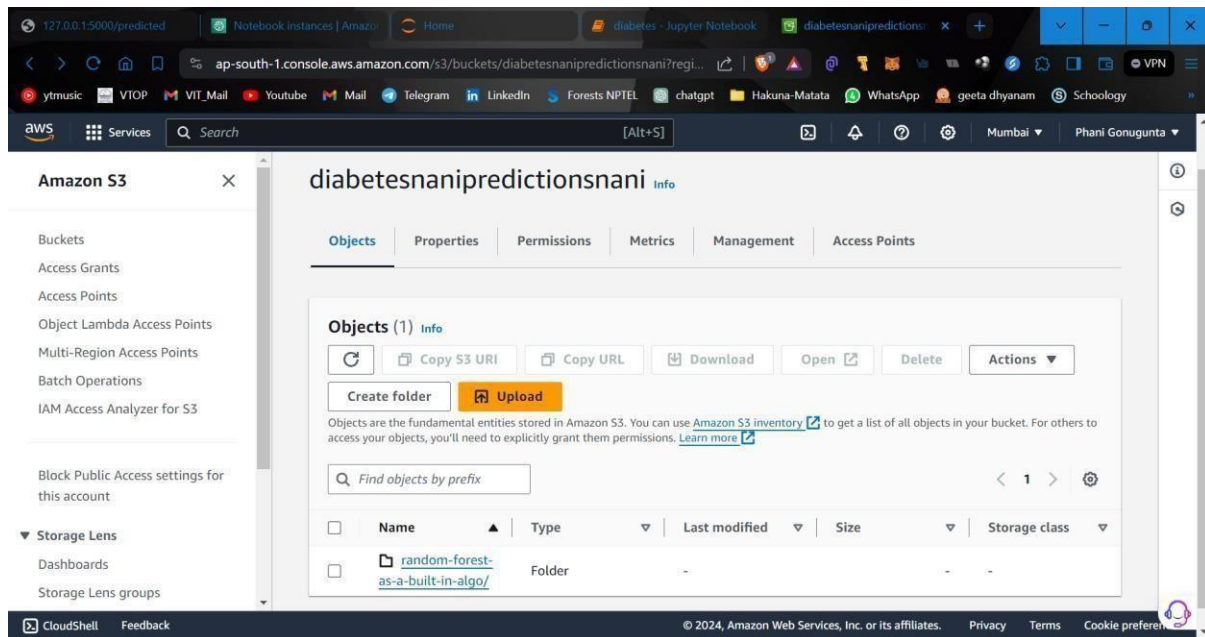
## SAGEMAKER:



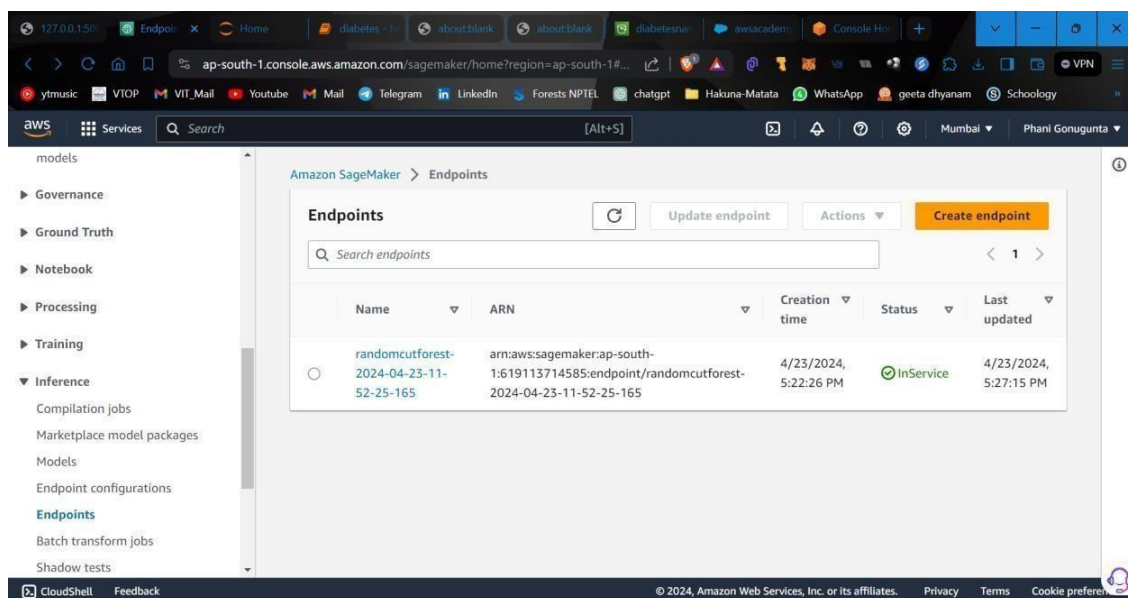
The screenshot shows the Amazon SageMaker console interface. The left sidebar contains navigation links: Getting started, Studio, Studio Lab, Canvas, RStudio, Admin configurations (Domains, Role manager, Images, Lifecycle configurations), and SageMaker dashboard. The main content area displays the 'Notebook instances' page with a table of instances.

Name	Instance	Creation time	Status	Actions
diabetesPrediction	ml.t3.medium	4/23/2024, 3:12:02 PM	InService	<a href="#">Open Jupyter</a>   <a href="#">Open JupyterLab</a>
obs	ml.t3.medium	3/2/2024, 12:50:35 AM	Stopped	<a href="#">Start</a>

## AMAZON S3:



## ENDPOINTS:



## AWS LAMBDA:

The screenshot shows the AWS Lambda console in the 'Functions' section. The left sidebar contains navigation links: Dashboard, Applications, Functions (selected), Additional resources (Code signing configurations, Event source mappings, Layers, Replicas), and Related AWS resources (Step Functions state machines). The main content area displays a table of functions with the following columns: Function name, Description, Package type, Runtime, and Last modified. A single function is listed: 'nanilambdafunc' with a description of '-', package type of 'Zip', runtime of 'Python 3.12', and last modified '3 days ago'. Above the table, there is a search bar, a 'Filter by tags and attributes or search by keyword' input, and a 'Create function' button. The top of the console shows the AWS logo, a search bar, and the user's name 'Phani Gonugunta'.

Function name	Description	Package type	Runtime	Last modified
nanilambdafunc	-	Zip	Python 3.12	3 days ago

## API GATEWAY:

The screenshot shows the AWS API Gateway console. The left sidebar contains navigation links: APIs, Custom domain names, VPC links, API: diabetes-model-api (selected), Resources, Stages, Authorizers, Gateway responses, Models, Resource policy, Documentation, Dashboard, API settings, Usage plans, API keys, Client certificates, and Settings. The main content area displays the 'POST' method for the resource '/api-diabetes'. Below the method, there is a 'Test' button. The 'Test' results section shows the following data:

Request	Latency ms	Status
/api-diabetes	1186	200

The response body is displayed as: 0.8804416623.