|  |  |
| --- | --- |
| **COURSE** | Artificial Intelligence |
| **PROJECT** | AI Based Diabetes Prediction System |
| **DATE** | 31-10-2023 |

**PROJECT STATEMENT:**

Clearly outline the problem statement, design thinking process, and the phases of development. Describe the dataset used, data preprocessing steps, and feature extraction techniques. Explain the choice of machine learning algorithm, model training, and evaluation metrics. Document any innovative techniques or approaches used during the development.

**SOLUTION ARCHITECTURE:**

Solution architecture is a structured approach to designing complex systems or projects, outlining the components, relationships, and processes to achieve specific goals or solve problems efficiently. It provides a high-level blueprint for project development and implementation.

1. Gather and preprocess data from various healthcare sources, ensuring data quality.

2. Train AI models for diabetes prediction and evaluate their performance.

3. Deploy models in a secure, scalable environment with a user-friendly interface.

4. Continuously monitor and update the system while complying with regulations.

5. Collaborate with healthcare professionals and gather user feedback for

improvements.

**Example**

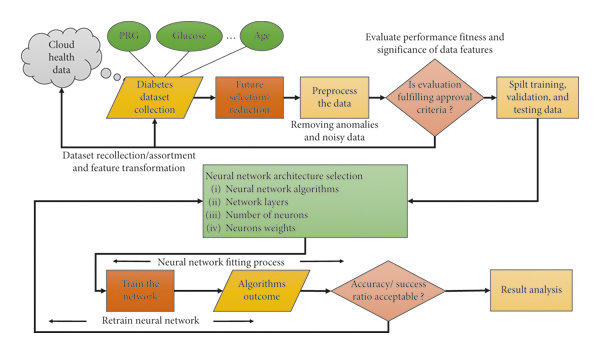


Figure: AI diabetes based prediction system

Reference:https://www.xenonstack.com/blog/artificial-intelligence-diabetes-detection

**EMPATHY MAP**

Empathy Map Canvas:

In creating an empathy map canvas for an AI-based diabetes prediction

system, we focus on understanding the typical user, their expressed needs,

thoughts, emotions, and actions. For instance, users might vocalize their struggles

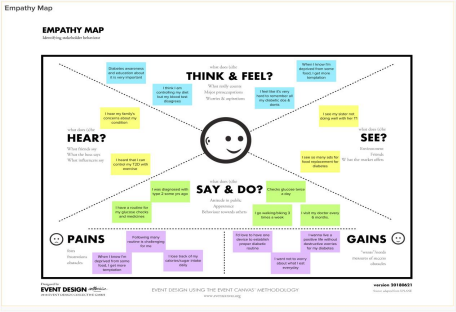
with blood sugar control, harbor concerns about long-term complications,

experience frustration from constant monitoring, and engage in behaviors like

blood sugar tracking and medication adherence. This canvas provides a holistic

view of the user's perspective, helping guide the development of a more user-

centric and effective diabetes prediction system.

****

1. **User Persona:** Start by defining a representative user persona, such as a middle aged individual with type 2 diabetes.

2. **Label "Says":** Create a section labeled "Says" where you note down direct  statements or quotes from the user regarding their diabetes management. For  example, "I struggle with managing my blood sugar levels."

3. **Label "Thinks":** In this section, jot down the thoughts, concerns, or worries that the  user may have about their diabetes. For instance, "I'm concerned about the long term effects of diabetes."

4. **Label "Feels":** Describe the emotions and feelings that the user experiences in  relation to their diabetes. For example, "I feel overwhelmed by the constant  monitoring and lifestyle changes."

5. **Label "Does":** Outline the actions and behaviors the user engages in concerning  diabetes management. This could include activities like checking blood sugar, taking  medications, and making dietary choices.

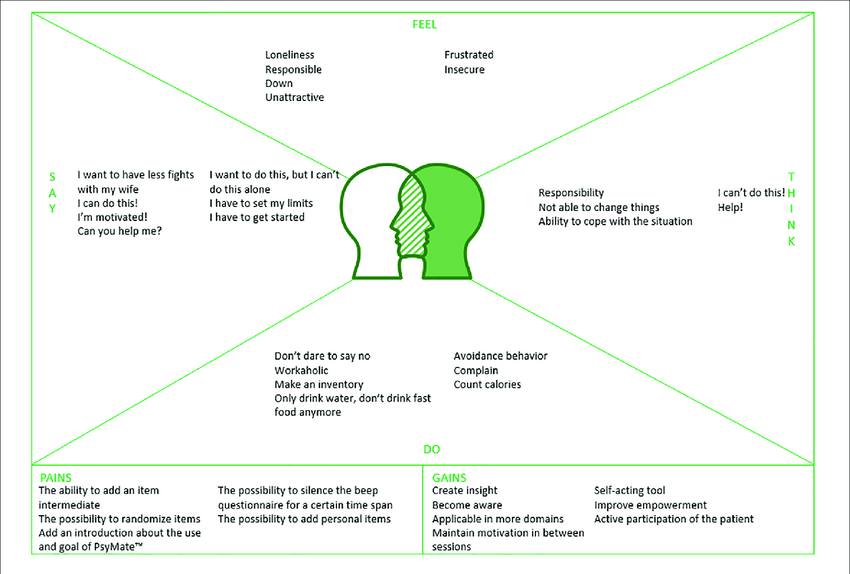
6. **Environment:** Consider the user's physical environment, including where they live  and work, as it can influence their diabetes management.

7. **Influences:** Identify the factors and people that influence the user's decisions and  behaviors, such as healthcare professionals, family, or friends.

8. **Pain Points:** List the challenges and difficulties the user faces in managing their  diabetes, such as the complexity of tracking data or the fear of complications.

9. **Gains:** Note the user's goals, aspirations, and desired outcomes related to their  diabetes, such as better control and improved quality of life.

10. **Takeaways:** Summarize the key insights gained from the empathy map that can  inform the design and development of the AI-based diabetes prediction system,  ensuring it addresses the user's needs, emotions, and actions effectively.



**PROPOSED SOLUTION :**

1. Problem Statement: Diabetes is a prevalent and serious health issue affecting millions of people worldwide. Early detection and management of diabetes are crucial to prevent complications. However, many individuals are unaware of their risk factors or hesitant to get tested. The problem statement should elaborate on the prevalence, impact, and challenges associated with diabetes diagnosis and management.

2. Idea / Solution Description: Our proposed solution is an AI-based diabetes prediction system. Leveraging machine learning algorithms, this system will analyze relevant medical and lifestyle data to predict the likelihood of an individual developing diabetes. The system will offer personalized risk assessments and recommendations for preventive measures.

3. Novelty / Uniqueness: The uniqueness of our solution lies in its ability to harness the power of AI and machine learning to predict diabetes risk accurately. It will continuously adapt and improve its predictions as more data becomes available. Additionally, it will provide personalized insights to users, promoting better health choices.

4. Social Impact / Health Benefits:

- The AI-based diabetes prediction system aims to address several key social and health-related aspects:

- Early Detection: By identifying individuals at risk, it promotes early diagnosis and intervention, reducing the likelihood of complications.

- Improved Quality of Life: People can make informed lifestyle choices to manage their health and reduce the risk of diabetes.

- Reduced Healthcare Costs: By preventing or managing diabetes efficiently, the system can contribute to lower healthcare expenditures.

5. Business Model (Revenue Model):

- Our revenue model could include:

- Subscription-Based Access: Charging users a fee for ongoing access to personalized diabetes risk assessments and recommendations.

- Data Monetization: Collaborating with healthcare organizations, research institutions, or pharmaceutical companies to provide anonymized, aggregated data for research purposes.

- Licensing to Healthcare Providers: Offering healthcare providers access to our AI system to enhance their patient care and risk assessment processes.

6. Scalability of the Solution:

- Our AI-based diabetes prediction system is highly scalable:

- Data Integration: It can integrate with various data sources, such as electronic health records, wearable devices, and lifestyle apps.

- User Base: The system can accommodate a growing user base without compromising prediction accuracy.

- Continuous Improvement: The system will continuously update its algorithms with new data and research, ensuring scalability and adaptability.

This proposed solution aims to address a critical health issue while also providing a sustainable business model. It has the potential to make a significant positive impact on individuals' health and well-being, as well as on the healthcare ecosystem.

**BRAINSTORMING & IDEA PRIORITIZATION**

Step 1: Brainstorming

1. Data Collection & Integration:

- Collect medical records, lifestyle data, and genetic information for comprehensive risk assessment.

- Integrate with wearable devices and apps to provide real-time data.

2. Machine Learning Algorithms:

- Explore different machine learning models (e.g., logistic regression, neural networks) for predicting diabetes.

- Investigate feature engineering techniques to improve model accuracy.

3. User Interface & Experience:

- Develop a user-friendly app or web platform for data input, risk assessment, and recommendations.

- Consider interactive dashboards, visualizations, and alerts.

4. Personalized Recommendations:

- Provide tailored dietary, exercise, and lifestyle recommendations based on risk assessment.

- Offer medication and healthcare provider suggestions when needed.

5. Data Security & Privacy:

- Implement robust security measures to protect user data and maintain compliance with healthcare regulations.

- Ensure transparent data handling and user consent mechanisms.

6. Research Collaboration:

- Collaborate with healthcare institutions and research organizations for data sharing and validation.

- Explore opportunities for contributing to diabetes research.

7. Continuous Learning and Updates:

- Enable the system to learn from user data and research advancements to improve prediction accuracy.

- Develop a system for providing regular updates and improvements.

8. Customer Support & Education:

- Offer customer support for inquiries and assistance with the system.

- Provide educational resources on diabetes prevention and management.

Step 2: Idea Prioritization

Use a Prioritization Framework:

For each idea generated during brainstorming, evaluate and assign scores based on criteria such as:

- Impact: How much will this feature or enhancement positively impact the users and their health?

- Feasibility: Can this idea be implemented with the available technology and resources?

- Profitability: Does the idea have potential for revenue generation or cost savings?

- Urgency: Is there an immediate need or demand for this feature?

- Alignment with Mission: Does the idea align with the project's mission and goals?

After scoring, calculate a total score for each idea to prioritize them. Ideas with the highest total scores should be considered for implementation first.

Step 3: Final Prioritized Ideas

List the ideas in descending order of their total scores:

1. [Idea 1]: Detailed Risk Assessment with Genetic Data Integration

2. [Idea 3]: User Interface with Interactive Dashboards

3. [Idea 2]: Research Collaboration with Healthcare Institutions

4. [Idea 4]: Personalized Lifestyle Recommendations

5. [Idea 7]: Continuous Learning and Updates

6. [Idea 8]: Customer Support & Education

7. [Idea 6]: Data Security & Privacy Measures

8. [Idea 5]: Machine Learning Model Optimization

9. [Idea 9]: Integration with Wearable Devices and Apps

This prioritized list can serve as a guide for project development, ensuring that high-impact and feasible features are addressed first. Keep in mind that this prioritization may evolve as the project progresses and new insights become available.

**DATASET LINK:**

[**https://www.kaggle.com/datasets/mathchi/diabetes-data-set**](https://www.kaggle.com/datasets/mathchi/diabetes-data-set)

**DESCRIPTION OF THE DATASET:**

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

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

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

**ENSEMBLE APPROACH**

Using an ensemble approach for an AI-based Diabetes Prediction System is a wise choice, as it can enhance prediction accuracy and robustness by combining the strengths of multiple machine learning models. Here's how you can implement an ensemble approach for diabetes prediction:

**Step 1: Data Preprocessing and Splitting**

* Begin by cleaning and preprocessing  diabetes dataset, handling missing values, encoding categorical variables, and scaling or normalizing features as needed.
* Perform feature engineering to create relevant features that may improve prediction accuracy.

**Code:**

1.The below code specifies handling of missing values.

|  |
| --- |
| df.isnull().values.any() |

2.The below code specifies the splitting of data.

|  |
| --- |
| # Data splitting  from sklearn.model\_selection import train\_test\_split  X = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DPF','Age']]  Y = df['Outcome']  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.3, random\_state=0) |

**Step 2: Selection of  Base Models**

      The goal of using an ensemble approach is to leverage the strengths of each individual classifier to improve the overall prediction accuracy of the system. Here's a brief overview of each of these classifiers:

 1.**Random Forest Classifier**:

* RandomForest is an ensemble learning method based on decision trees. It creates multiple decision trees during training and combines their predictions. It is known for its robustness, ability to handle complex data, and resistance to overfitting.

1. **Logistic Regression**:
   * Logistic Regression is a simple yet effective linear classification algorithm. It is widely used in binary classification problems, such as predicting the probability of a binary outcome (e.g., diabetes vs. non-diabetes). It's interpretable and computationally efficient.
2. **Support Vector Machine (SVM)**:
   * SVM is a powerful classification algorithm that finds a hyperplane that best separates data points belonging to different classes. It can handle both linear and non-linear classification tasks through kernel functions. SVMs are known for their ability to handle high-dimensional data and find optimal decision boundaries.

The Voting Classifier combines the predictions of these three classifiers by taking a weighted average (soft voting) of their predicted probabilities. This ensemble approach can often lead to better performance than any single classifier on its own, as it leverages their individual strengths and mitigates their weaknesses.

The choice of using this ensemble of classifiers is a common practice in machine learning, especially when the dataset is complex and diverse. It can provide a more robust and accurate prediction system by aggregating the diverse perspectives of multiple models.

**Code:**

|  |
| --- |
| # Create individual classifiers  rf = RandomForestClassifier(n\_estimators=200)  lr = LogisticRegression()  svm = SVC(probability=True)  # Create a voting classifier  voting\_classifier = VotingClassifier(estimators=[('rf', rf), ('lr', lr), ('svm', svm)], voting='soft')  # Fit the voting classifier on the training data  voting\_classifier.fit(X\_train, Y\_train)  # Evaluate the accuracy of the voting classifier on the test data  accuracy = voting\_classifier.score(X\_test, Y\_test)  print("Accuracy of Voting Classifier:", accuracy) |

* Once we are satisfied with the ensemble's performance, we can use it to make predictions on new, unseen data.
* Next ,we deploy the ensemble model in your Diabetes Prediction System, whether as a web application, mobile app, or integrated into a healthcare system.

**PROJECT OVERVIEW**

* The AI-Based Diabetes Prediction System is a data-driven healthcare solution designed to  predict the likelihood of an individual developing diabetes based on a set of key medical and  demographic features. The project's primary objectives include:

• Data Preprocessing

• Model Selection

• Model Evaluation

• Model Deployment

**DEVELOPMENT STEPS**:

To begin building our project, we'll need to start with data loading, preprocessing and model  selection. Below are the steps you can follow:

**1.DATA COLLECTION AND LOADING**

The very first step is to choose the dataset for our model .This dataset is originally from  the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict  based on diagnostic measurements whether a patient has diabetes.

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

Now we have to set the development environment to build our project. Import the necessary  libraries to built the model. We are ensembleing the random forest ,SVM and logistic regression  model .

Code:

|  |
| --- |
| import numpy as np  import pandas as pd |

Here's a quick overview of the libraries we've imported and what each of them is used for:

➢ numpy (import numpy as np): NumPy is a fundamental library for scientific computing in  Python.

➢ pandas (import pandas as pd): Pandas is a widely used library for data manipulation and  analysis. It provides data structures like DataFrame .

Code:

|  |
| --- |
| # Load the dataset  data = pd.read\_csv("/content/sample\_data/diabetes.csv") |

➢ The primary task is to read the data from the "diabetes.csv" file. This can be done using  the read\_csv function provided by pandas. The loaded data is stored in a pandas  DataFrame, which is a two-dimensional, size-mutable, and potentially heterogeneous  tabular data structure with labeled axes (rows and columns). In this case, the DataFrame  is assigned to the variable "data."

➢ After running this code, the "data" variable will contain the contents of the "diabetes.csv"  file as a DataFrame, and we can use various pandas methods and functions to analyze,  manipulate, and visualize the data as needed for our project or analysis.

**2.DATA EXPLORATION:**

Exploring the data is a crucial step in understanding the dataset, identifying patterns, and  gaining insights that can guide our modeling process. Here are some common data exploration  techniques :

Code:

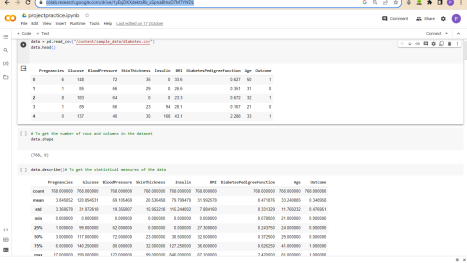
|  |
| --- |
| data.head()  # To get the number of rows and columns in the dataset  data.shape  # To get the statistical measures of the data  data.describe()# To get the statistical measures of the data  data.columns |

➢ **data.head()**: It gives you a quick preview of what your data looks like, including the first  five rows by default.

➢ **data.shape**: This will returns the number of rows and columns in your dataset. The  output will be in the format (number of rows, number of columns).

➢ **data.describe()**: This code provides statistical summary measures for your dataset.  ➢ **data.columns**: This code returns the column names (features) of your dataset.

Output:



**3.DATA PREPROCESSING AND SPLITTING:**

Here are some common data preprocessing steps we can follow ,

➢ **Handling Missing Values:**

Check for missing values in your dataset and decide how to handle them. Checking  for missing values is an essential step in data preprocessing to ensure that the dataset is  clean and ready for analysis or modeling.

Code:

|  |
| --- |
| missing\_values = data.isnull().sum()  print(missing\_values) # Check for missing values   data.isnull().values.any() |

The code **data.isnull().values.any()** is used to check whether there are any missing  (NaN) values in the DataFrame **data**. If the code returns **True**, it means that there are  missing values in the DataFrame **data**. If it returns **False**, it means that there are no  missing values in the DataFrame.

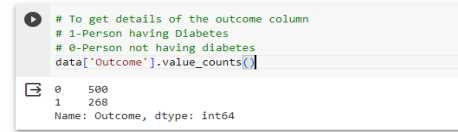
Output:



Code:

|  |
| --- |
| # To get details of the outcome column   # 1-Person having Diabetes   # 0-Person not having diabetes   data['Outcome'].value\_counts() |

Output:

Code:

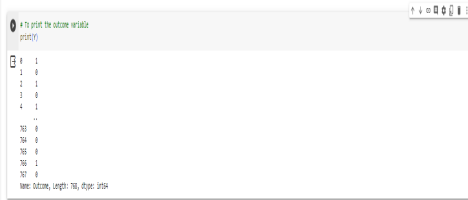
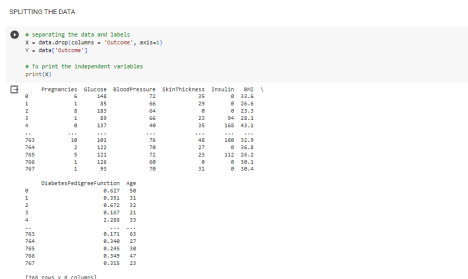
|  |
| --- |
| # separating the data and labels  X = data.drop(columns = 'Outcome', axis=1)  Y = data['Outcome']  # To print the independent variables  print(X)  # To print the outcome variable  print(Y) |

In this code, you are separating your dataset into two essential components for machine learning:

1. X: This contains the independent variables (features) and is used to store all the data  attributes that will be used as input for your machine learning model.

2. Y: This contains the dependent variable (outcome or label) and is used to store the target  you want to predict or classify.

Output:



**CHECK OUT THE CODE :**

**https://colab.research.google.com/drive/1llxazFt1201IWkvuMcy29G0KOArcgw6u#scrollT o=L06tBl2x0jOI**

**DEVELOPMENT STEPS**:

This phase is where you'll leverage our preprocessed data to create, train, and evaluate machine  learning models. Here are the general steps to get started with model building and training :

∙ Model Training

∙ Model Accuracy

**1.MODEL TRAINING:**

Before you can train our machine learning models, you need to import them. Here's how we can  import the necessary machine learning models for our AI-based Diabetes Prediction System.

Code:

|  |
| --- |
| #for numerical operations  import pickle  #for visualization  from sklearn.ensemble import RandomForestClassifier, VotingClassifier from sklearn.linear\_model import LogisticRegression  from sklearn.svm import SVC |

Here is the quick overview of the imported statements;

⮚ **import pickle**: Used for saving and loading Python objects, including machine learning  models, to and from files.

⮚ **from sklearn.ensemble import RandomForestClassifier, VotingClassifier**: Imports  machine learning models, **RandomForestClassifier** for handling complex data and  **VotingClassifier** for combining multiple classifier predictions.

⮚ **from sklearn.linear\_model import LogisticRegression**: Imports **LogisticRegression**, a  simple yet effective linear classification algorithm commonly used in binary  classification.

⮚ **from sklearn.svm import SVC**: Imports **SVC** (Support Vector Classifier), a powerful  classification algorithm for finding optimal decision boundaries in binary and multi-class  classification tasks.

As we started importing the models, the next phase is creating the instance on models,

Code:

|  |
| --- |
| # Create instances of the models  rf = RandomForestClassifier(n\_estimators=200)  lr = LogisticRegression()  svm = SVC(probability=True) |

 In the code snippet you provided, you're creating instances of machine learning models. Here's  what each line does:

⮚ **rf = RandomForestClassifier(n\_estimators=200)**: This line creates an instance of the  **RandomForestClassifier** model. It specifies that the random forest should consist of 200  decision trees (you can adjust this number as needed). This model is suitable for handling  complex data and is robust against overfitting.

⮚ **lr = LogisticRegression()**: Here, you're creating an instance of the **LogisticRegression** model. This model is a straightforward yet effective choice for linear classification tasks,  particularly in binary classification. It's known for its interpretability and computational  efficiency.

⮚ **svm = SVC(probability=True)**: This line creates an instance of the **SVC** (Support  Vector Classifier) model with the **probability** parameter set to **True**. The **probability**

parameter allows the model to predict probabilities, which is often useful in classification  tasks. SVMs are known for their ability to find optimal decision boundaries and handle  high-dimensional data.

Code:

|  |
| --- |
| X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.3,  random\_state=0) |

After running this code, **X\_train** and **Y\_train** will contain the features and target variable for the  training set, while **X\_test** and **Y\_test** will contain the features and target variable for the testing  set. This allows us to train our machine learning models on the training set and evaluate their  performance on the testing set.

As our model is splitted and trained ,the next thing we can do is creating a voting classifier and  fit our model into them .

Code:

|  |
| --- |
| # Create a voting classifier  voting\_classifier = VotingClassifier(estimators=[('rf', rf), ('lr', lr), ('svm', svm)],  voting='soft')  # Fit the voting classifier on the training data  print(voting\_classifier.fit(X\_train, Y\_train)) |

Output:



**2.MODEL ACCURACY:**

Moving on to model accuracy is an important step to evaluate how well your machine learning  models are performing. We can assess the model accuracy using various metrics. The accuracy  score is a common metric to measure the overall correctness of your model's predictions.

Code:

|  |
| --- |
| # Evaluate the accuracy of the voting classifier on the test data  accuracy = voting\_classifier.score(X\_test, Y\_test)  print("Accuracy of Voting Classifier:", accuracy) |

Here's a brief explanation:

⮚ **voting\_classifier.score(X\_test, Y\_test)**: The **score** method of the **voting\_classifier** object is used to evaluate the model's accuracy on the provided test data. **X\_test** contains  the features, and **Y\_test** contains the true target labels.

⮚ **accuracy = ...**: The result of the accuracy calculation is assigned to the variable  **accuracy**.

⮚ **print("Accuracy of Voting Classifier:", accuracy)**: This line prints the accuracy of the  Voting Classifier on the test data to the console.

The accuracy score represents the proportion of correct predictions made by our model on the  test data. It's a common metric to assess the model's performance in classification tasks. The  accuracy value will be between 0 (no correct predictions) and 1 (all predictions are correct).

In our case, the accuracy value will tell you how well the Voting Classifier is performing in  predicting diabetes based on the test data.

Output:



Code:

|  |
| --- |
| # Save the voting classifier to a file  filename = 'model/voting\_diabetes.pkl'  pickle.dump(voting\_classifier, open('/content/sample\_data/code.py', 'wb')) print("SUCCESS") |

Here's a summary of what the code does:

⮚ It specifies the file path where the model will be saved using the **filename** variable. ⮚ It saves the **voting\_classifier** model to the specified file path using **pickle.dump()**. ⮚ It prints a success message indicating that the model has been saved.

Output:



We can now proceed to the next steps in your project, such as model deployment or further  analysis, knowing that the model has been successfully saved.

**CHECK OUT THE CODE:**

**https://colab.research.google.com/drive/1llxazFt1201IWkvuMcy29G0KOArcgw6u#scrollT o=13VKf1\_reJSu**