# A Project Report

**On**

**Diabetes Prediction Model**

Logo, company name

Description automatically generated

### Under the Supervision of

#### Mr. Dhruba Ray

#### Senior Faculty

Ardent Group of Companies

## By

**SIDDHARTH SINGH (202200141)**

*In partial fulfillment of requirements for the award of degree in*

Bachelor of Technology in Computer Science and Engineering

(2024)

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY

## (A constituent college of Sikkim Manipal University)

MAJITAR, RANGPO, EAST SIKKIM – 737136

## **Abstract**

## The Diabetes Prediction Model uses various machine learning algorithms to predict the likelihood of diabetes in individuals based on medical and demographic features. This documentation provides a comprehensive guide to understanding the model's purpose, data Preprocessing steps, model training and evaluation, and the significance of the results.

**Table of Content**

1. Introduction
2. Data Preprocessing
   1. Loading of Dataset
   2. Data Exploration
   3. Handling Missing Values
   4. Feature and Target Variables
   5. Data Scaling
3. Model Training and Prediction
   1. Logistic Regression
   2. Random Forest Classifier
   3. Decision Tree Classifier
   4. Support Vector Machine (SVM)
4. Model Evaluation
   1. Evaluation Metrics
   2. Model Comparisons
5. Conclusion and Significance
6. References
7. **INTRODUCTION**

· **Objective of the Model**: The Diabetes Prediction Model is designed to assess the likelihood of an individual having diabetes. This is done by analyzing various medical and demographic characteristics.

· **Input Features**: The model uses a set of input features to make predictions. These features may include medical data such as blood glucose levels, blood pressure, body mass index (BMI), and age, as well as demographic information such as gender and ethnicity.

· **Importance of Early Detection**: Identifying diabetes early is crucial because it allows for timely intervention. Early detection means that individuals can start managing their condition before it progresses to more severe stages.

· **Management and Treatment**: Once diabetes is detected early, patients can benefit from more effective management strategies and treatments. This can include lifestyle changes, medication, and regular monitoring, all of which contribute to better health outcomes.

· **Improving Patient Outcomes**: Effective early detection and management of diabetes can significantly enhance the quality of life for patients. It reduces the risk of complications associated with diabetes, such as cardiovascular diseases, nerve damage, and kidney problems.

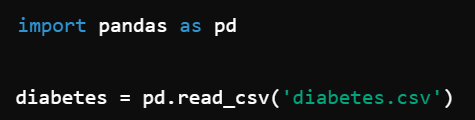
· **Predictive Modeling**: The Diabetes Prediction Model leverages predictive analytic to determine the probability of an individual having diabetes based on the given features. This model helps healthcare professionals in making informed decisions regarding patient care.

· **Data-Driven Approach**: By analyzing historical data and patterns, the model provides valuable insights that aid in understanding the risk factors associated with diabetes. This helps in personalizing treatment plans and preventive measures.

· **Healthcare Benefits**: The use of predictive models in healthcare promotes proactive care rather than reactive treatment, ultimately leading to a healthier population and more efficient use of medical resources.

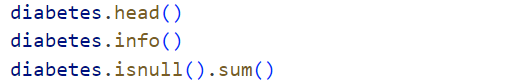
1. **DATA PREPROCESSING**
   1. **Loading the Dataset**

The Dataset used is the Pima Indians Diabetes Dataset, which is loaded with the help of pandas.

****

* 1. **Data Exploration**

It helps us to explore the data to find missing values or to interpret something from that particular Dataset.



· diabetes.head():

* · This command displays the first 5 rows of the diabetes DataFrame. It's useful for getting a quick overview of the data, including the column names and some sample values.

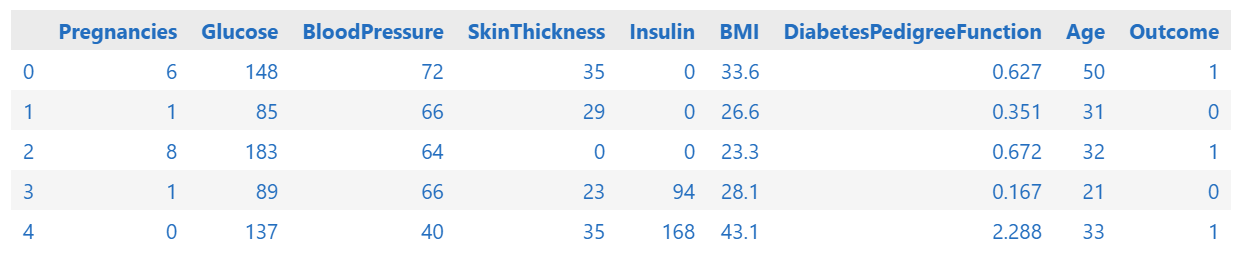
· diabetes.info():

* · This command provides a concise summary of the diabetes DataFrame. It includes information such as the number of non-null entries in each column, the data type of each column, and the memory usage of the DataFrame. This is helpful for understanding the structure of the data and identifying any columns with missing values.

·

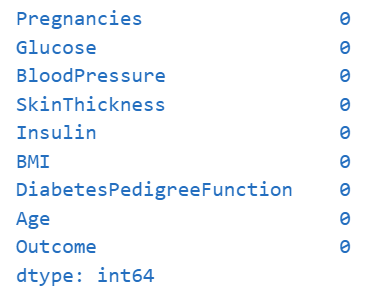
diabetes.isnull().sum():

* · This command checks for missing values in the diabetes DataFrame. diabetes.isnull() returns a DataFrame of the same shape as diabetes, with True for null values and False for non-null values. The .sum() function then sums up these boolean values column-wise, giving the total count of missing values in each column.



* 1. **Handling Missing values**

In our Dataset, there are no missing values. So, we do not need to perform operations to find missing values.

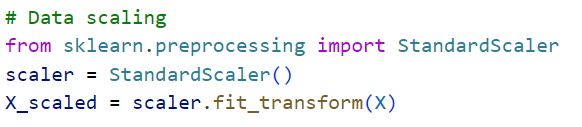


* 1. **Feature and Target Variables**



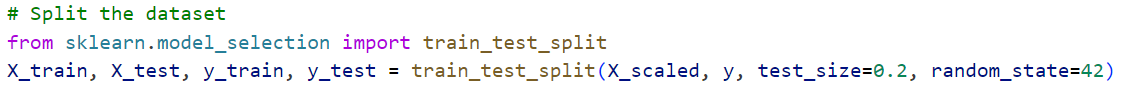
* X = diabetes.drop(columns=['Outcome']):
* · This line creates a new DataFrame X that contains all the columns from the diabetes DataFrame except for the Outcome column. The drop method is used to remove the specified column ('Outcome'), and columns=['Outcome'] indicates which column to drop. This resulting DataFrame X will contain all the features used for training the model, excluding the target variable.
* · y = diabetes['Outcome']:
* · This line creates a Series y that contains only the values of the Outcome column from the diabetes DataFrame. This column typically represents the target variable (e.g., whether or not a patient has diabetes) that you want to predict.
  1. **Data Scaling**

To perform data scaling we use “StandardScaler”.



1. **MODEL TRAINING AND PREDICTION**
   1. **Splitting the Dataset**

Splitting the Dataset into training and testing sets in the ratio 80:20.



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

* · This line imports the train\_test\_split function from the sklearn.model\_selection module. This function is used to split the dataset into training and testing sets.

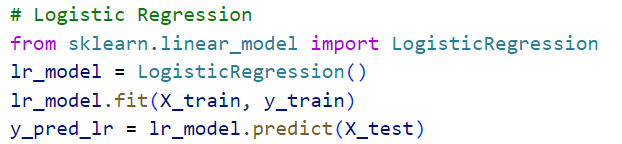
· X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42):

* · train\_test\_split(X\_scaled, y, ...):
  + This function splits the arrays or matrices into random train and test subsets. In this case, X\_scaled (which appears to be the scaled version of the features X) and y (the target variable) are being split.
* test\_size=0.2:
  + This parameter specifies the proportion of the dataset to include in the test split. Here, 0.2 means that 20% of the data will be used for testing, and the remaining 80% will be used for training.
* random\_state=42:
  + This parameter sets the seed for the random number generator, ensuring that the split is reproducible. Using random\_state=42 ensures that the same split will be used every time the code is run.
* X\_train, X\_test, y\_train, y\_test:
  + These are the outputs of the train\_test\_split function. X\_train and y\_train are the training sets for the features and target variable, respectively. X\_test and y\_test are the corresponding test sets.
  1. **Logistic Regression**

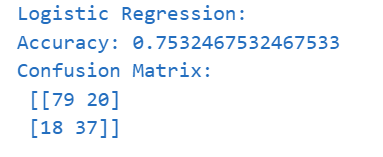
Logistics regression is a type of supervised machine learning algorithm that is used for binary classification problems. It forecasts the likelihood of an event occurring and is binary (1/0 or Yes/No or True/False) depending on one or many independent variables (features). Contrary to what people may think Logistic Regression is used in classification not regression.

### Key Concepts:

1. **Logistic Function (Sigmoid Function)**: Transforms the linear combination of input features into a probability (a value between 0 and 1).
2. **Odds and Log-Odds**: Logistic regression models the log-odds of the probability of an event occurring.
3. **Decision Boundary**: The threshold probability (usually 0.5) used to classify the input into one of the two classes.



* Import Logistic Regression: It imports the Logistic Regression class from the scikit-learn library.
* Create a Model: It initializes a Logistic Regression model and assigns it to the variable lr\_model.
* Train the Model: The code trains the model using the training data (X\_train, y\_train) using the fit method.
* Make Predictions: After training, the model predicts the class labels for the test data (X\_test) using the predict method, storing the predictions in the variable y\_pred\_lr.



* 1. **Random Forest Classifier**

Random Forest Classifier is a sub type of ensemble learning where a large number of decision trees is built in the training phase and the classification mode is the arithmetic mean of the trees’ classes or the forecast mode for regression problems. Bagging is also used together with random feature selection in which the conception of bootstrapping are integrated into the model to minimize over fitting.

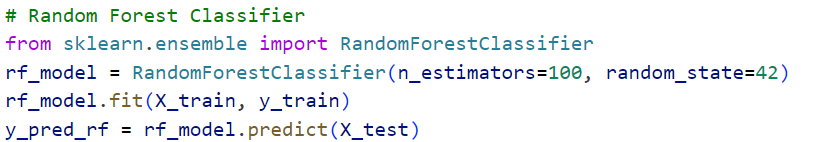
### Key Concepts:

1. **Ensemble Method**: Combines multiple models to improve performance.
2. **Bagging (Bootstrap Aggregating)**: Each tree is trained on a random subset of the data with replacement.
3. **Feature Randomness**: When splitting nodes, a random subset of features is considered, adding another layer of randomness that helps prevent overfitting.
4. **Majority Voting**: For classification, the class that gets the most votes from all the trees in the forest is chosen.

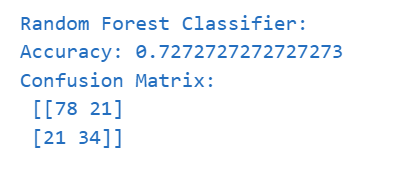
### Hyper-parameters of Random Forest:

Some important hyper-parameters you can tune in a Random Forest model:

* **n\_estimators**: The number of trees in the forest.
* **max\_depth**: The maximum depth of the tree.
* **min\_samples\_split**: The minimum number of samples required to split an internal node.
* **min\_samples\_leaf**: The minimum number of samples required to be at a leaf node.
* **max\_features**: The number of features to consider when looking for the best split.



* “from sklearn.ensemble import RandonForestClassifier”: By importing this tool, you can use it to build a Random Forest model to make predictions.
* “RandomForestClassifier(n\_estimators=100, random\_state=42)”: This line of code is creating a new instance of the RandomForestClassifier class, with two specific settings:
  + n\_estimators=100: This means the Random Forest model will use 100 individual decision trees to make predictions. Think of it like having 100 experts voting on the best answer. The more trees (experts), the more accurate the predictions will be.
  + random\_state=42: This is like setting a secret code to ensure that the model behaves consistently. When you run the model multiple times, you'll get the same results because the random number generator is seeded with the same value (42). It's like setting a specific starting point for a random process.
* “rf\_model.fit(X\_train, y\_train)”:
  + X\_train: This is the feature data, which is like the pictures of the animals. It's the input data that the model uses to learn.
  + y\_train: This is the target data, which is like the labels on the pictures. It's the output data that the model is trying to predict.
  + fit(): This is the method that trains the model using the training data.
* “y\_pred\_rf = rf\_model.predict(X\_test)”:
  + X\_test: This is the feature data for the testing set, which is like the new set of pictures.
  + rf\_model: This is the trained Random Forest model, which has already learned to recognize patterns in the training data.
  + predict(): This is the method that uses the trained model to make predictions on new data.
  + y\_pred\_rf: This is the variable that stores the predicted values

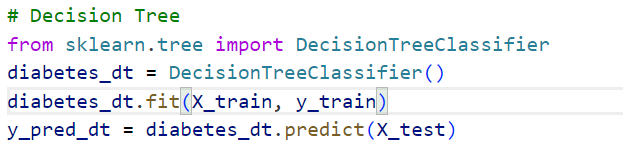


* 1. **Decision Tree**

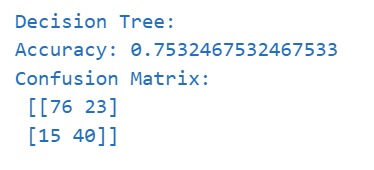
A Decision Tree is another method of learning that falls under the divide of supervised learning methods that can be used in classification and regression problems. It does this by dividing the data into subsets based on the value of the input features; this results in a tree structure where a node is a feature, a branch is a decision rule, and a leaf node is a consequence (class label or numeric value).

### Key Concepts:

1. **Root Node**: The top node of the tree where the first decision is made.
2. **Internal Nodes**: Nodes where decisions are made based on features.
3. **Leaf Nodes**: Terminal nodes that provide the output (class label or value).
4. **Splitting**: The process of dividing a node into two or more sub-nodes.
5. **Impurity Measures**: Metrics like Gini impurity, entropy (for classification), or mean squared error (for regression) to decide the best split.



* “from sklearn.tree import DecisionTreeClassifier”:
  + “from” is a keyword that tells Python to import something.
  + sklearn.tree is the module within the scikit-learn library that contains decision tree algorithms.
  + import is the keyword used to import a specific item from a module.
  + DecisionTreeClassifier is the specific class being imported, which allows you to create decision tree models.
* “diabetes\_dt.fit(X\_train, y\_train)”:
  + diabetes\_dt: This is the name of the decision tree classifier object you've created.
  + fit(): This is the method used to train machine learning models in scikit-learn.
  + X\_train: This represents the training data features.
  + y\_train: This represents the corresponding target values (labels) for the training data.
* “y\_pred\_dt = dt\_model.predict(X\_test)”:
  + X\_test: This is the feature data for the testing set, which is like the new set of pictures.
  + dt\_model: This is the trained Decision Tree, which has already learned to recognize patterns in the training data.
  + predict(): This is the method that uses the trained model to make predictions on new data.
  + y\_pred\_dt: This is the variable that stores the predicted values.

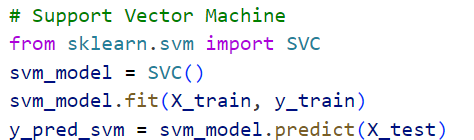


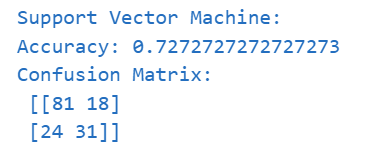
* 1. **Support Vector Machine**

SVM stands for Support Vector Machine is a supervised learning method used for creating models for classification as well as regression analysis. One of its important characteristics is that it performs especially well on the data of high dimensions; another fact about this algorithm is that it solves binary classification tasks. SVM on the other hand seeks to find the best hyperplane that will bound the data well according to its classes.

### Key Concepts:

1. **Hyperplane**: A decision boundary that separates different classes in the feature space.
2. **Support Vectors**: Data points that are closest to the hyperplane and influence its position and orientation.
3. **Margin**: The distance between the hyperplane and the nearest data points (support vectors). SVM aims to maximize this margin.





1. **MODEL EVALUATION**

Evaluating the model based on following parameters

Accuracy Score: The proportion of the number of true positive cases and false negative cases to the total number of cases; the efficiency of the model.

Classification Report: A confusion matrix that gives more specific information about the model’s attributes such as accuracy, recall, f1-score, and support of each class to reveal further information about its per-class performance.

Confusion Matrix: A table that demonstrates the true positive, true negative, false positive and false negative predictions, giving detailed description or overall results of the model.

Mean Squared Error (MSE): The squared sum of the differences between the predicted and actual values divided by the number of values, used to determine the accuracy of a regression line.



1. **CONCLUSION AND SIGNIFICANCE**

The Diabetes Prediction Model employs modern artificial neural network means of estimating an individual’s probabilities of acquiring diabetes. The given model uses Support Vector Machines, Decision Trees, Random Forests, Logistic Regression to predict data by considering extensive medical data accurately. Such forecasts help in timely diagnosis and proper intervention if perhaps diabetic issues exist, and hence can spare complicated difficulties which are likely to accompany un-diagnosed diabetes. Ongoing assessments and comparisons with other methods help to sustain the model’s reliability and relevance to assisting healthcare professionals to implement individualized care processes. Finally, this increases patients’ welfare because early starts a proper biomedical treatment approach that aims at using health care resources effectively for better diabetes control.

1. **REFERENCES**
2. Pima Indians Diabetes Dataset: https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
3. Scikit-learn documentation: Scikit-learn