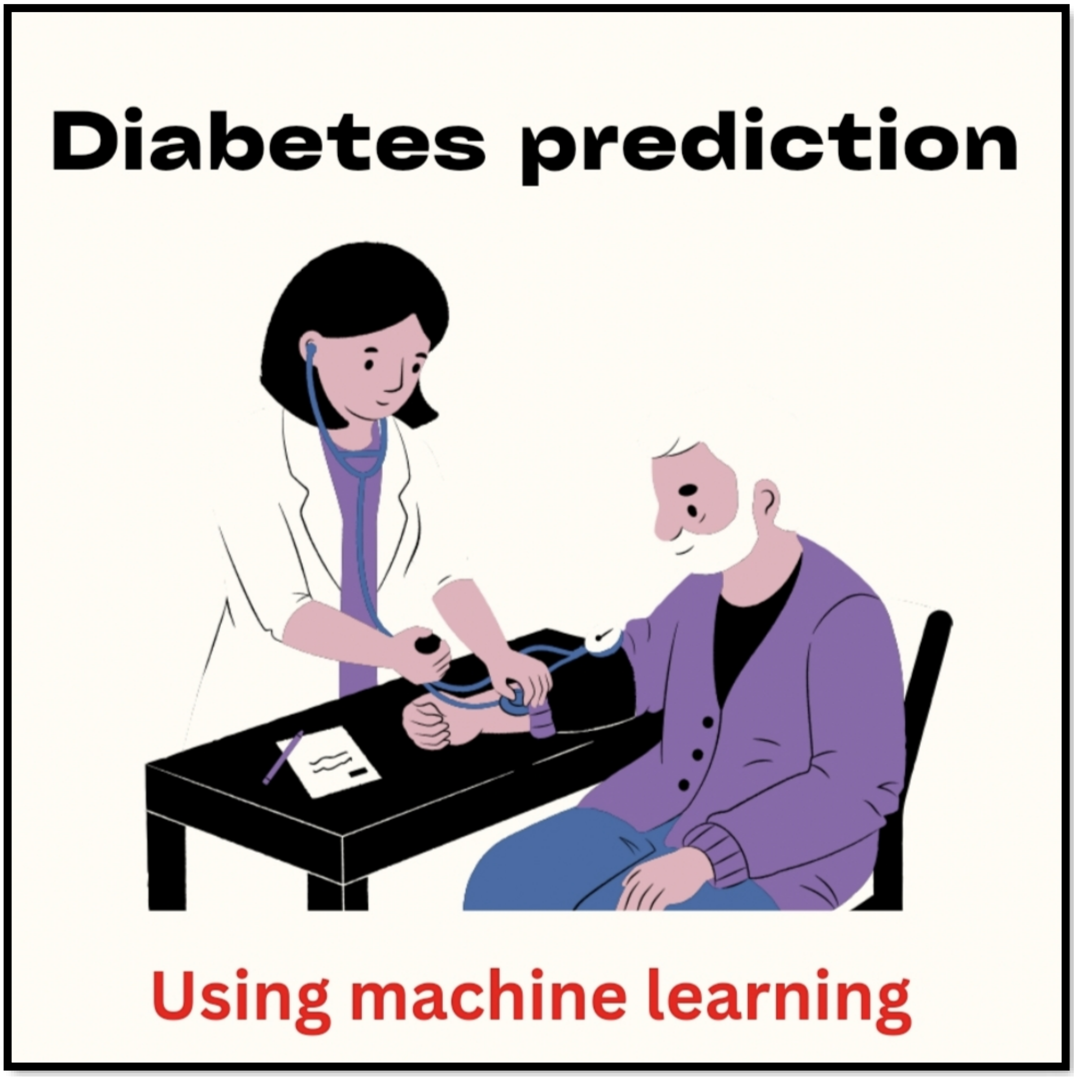
# DIABETES PREDICTION USING MACHINE LEARNING

au622021243028 : KARAN P



## Abstract:

- Globally, diabetes affects 537 million people, making it the deadliest and the most common non-communicable disease. Many factors can cause a person to get affected by diabetes, like excessive body weight, abnormal cholesterol level, family history, physical inactivity, bad food habit etc.
- Increased urination is one of the most common symptoms of this disease. People with diabetes for a long time can get several complications like heart disorder, kidney disease, nerve damage, diabetic retinopathy etc. But its risk can be reduced if it is predicted early.

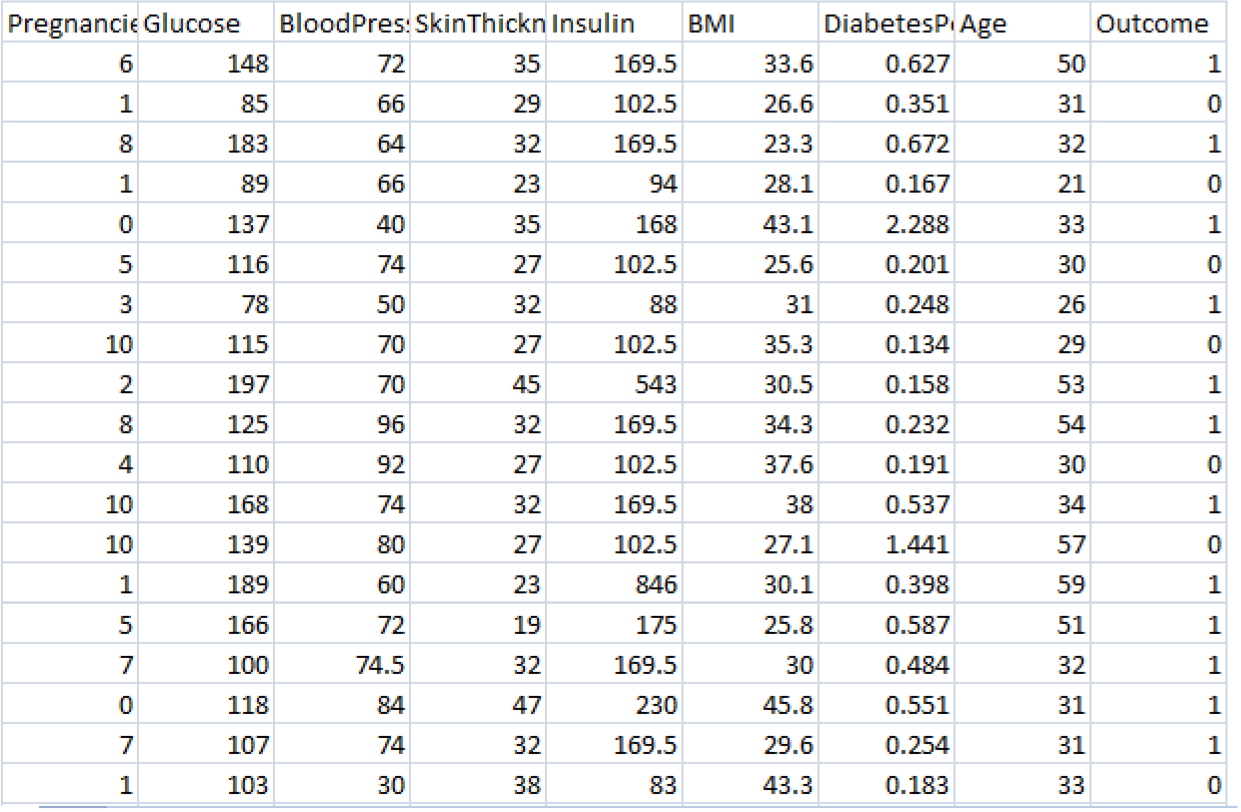
## Introduction

- For predicting blood pressure status, they used conditional decision making and for predicting diabetes, they used SVM, KNN, and decision tree. Among these models, SVM worked better as they got 75% accuracy which is better than other classifier algorithms.
- Random forest is a machine learning system that averages the predictions of several decision trees. As a result, the random forest can be considered an ensemble learning model

## Data source

The datasets consists of several medical predictor variables and one target variable, outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

**Dataset link:** (https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database )



| Pregnancie | Glucose | BloodPres | SkinThickn | Insulin | BMI | DiabetesP | Age | Outcome |
|---|---|---|---|---|---|---|---|---|
| 6 | 148 | 72 | 35 | 169.5 | 33.6 | 0.627 | 50 | 1 |
| 1 | 85 | 66 | 29 | 102.5 | 26.6 | 0.351 | 31 | 0 |
| 8 | 183 | 64 | 32 | 169.5 | 23.3 | 0.672 | 32 | 1 |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 5 | 116 | 74 | 27 | 102.5 | 25.6 | 0.201 | 30 | 0 |
| 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 10 | 115 | 70 | 27 | 102.5 | 35.3 | 0.134 | 29 | 0 |
| 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 8 | 125 | 96 | 32 | 169.5 | 34.3 | 0.232 | 54 | 1 |
| 4 | 110 | 92 | 27 | 102.5 | 37.6 | 0.191 | 30 | 0 |
| 10 | 168 | 74 | 32 | 169.5 | 38 | 0.537 | 34 | 1 |
| 10 | 139 | 80 | 27 | 102.5 | 27.1 | 1.441 | 57 | 0 |
| 1 | 189 | 60 | 23 | 846 | 30.1 | 0.398 | 59 | 1 |
| 5 | 166 | 72 | 19 | 175 | 25.8 | 0.587 | 51 | 1 |
| 7 | 100 | 74.5 | 32 | 169.5 | 30 | 0.484 | 32 | 1 |
| 0 | 118 | 84 | 47 | 230 | 45.8 | 0.551 | 31 | 1 |
| 7 | 107 | 74 | 32 | 169.5 | 29.6 | 0.254 | 31 | 1 |
| 1 | 103 | 30 | 38 | 83 | 43.3 | 0.183 | 33 | 0 |

## Steps involved in Diabetes Prediction Project

- ✓ Collection of data
- ✓ Exploring the data
- ✓ Splitting the data
- ✓ Training the model
- ✓ Evaluating the model
- ✓ Deploying the model

## Model Training:

## Data collection:

The very first step is to choose the dataset for our model. We can get a lot of different datasets from Kaggle. You just need to sign in to Kaggle and search for any dataset you need for the project.This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict whether a patient has diabetes based on diagnostic measurements. 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.

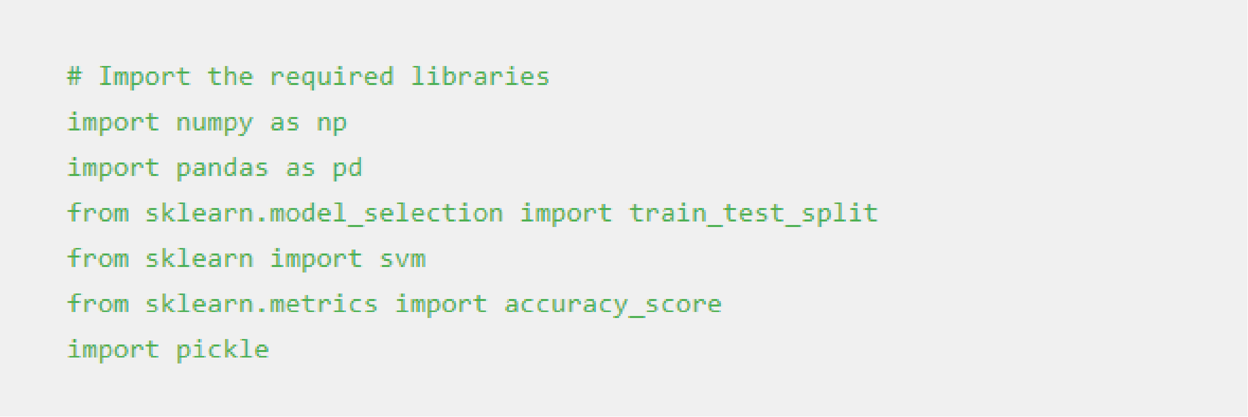
The data contains 9 columns which are as follows

- **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)

## Exploring the Data

Now we have to set the development environment to build our project. For this project, we are going to build this Diabetes prediction using Machine Learning in Google Colab. You can also use Jupyter Notebook.

After downloading the dataset, import the necessary libraries to build the model.

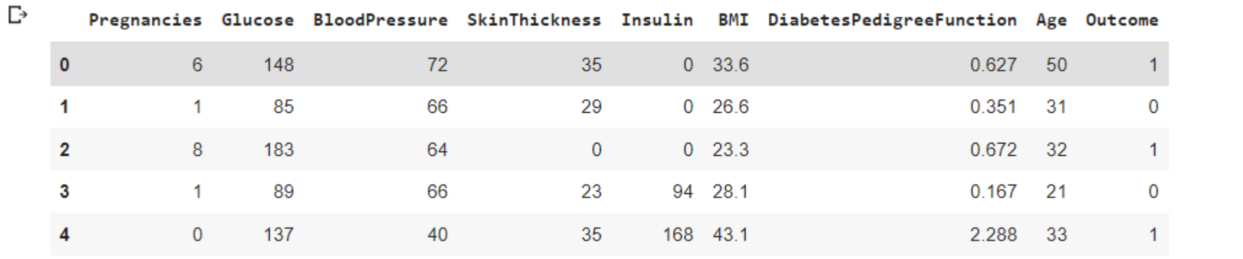


```
# Import the required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
import pickle
```

Load the data using the *read_csv* method in the pandas library. Then the *head()* method in the pandas library is used to print the rows up to the limit we specify. The default number of rows is five.

```
# Load the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('diabetes.csv')

# Print the first 5 rows of the dataset
diabetes_dataset.head()
```

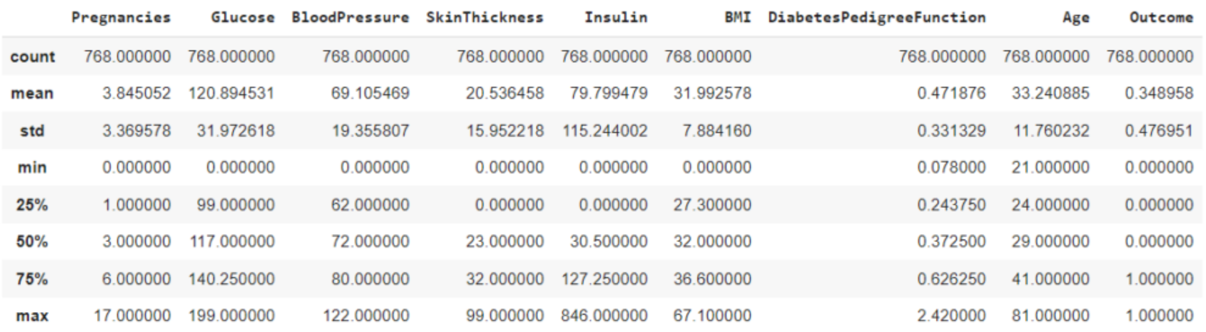
## Output:



| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

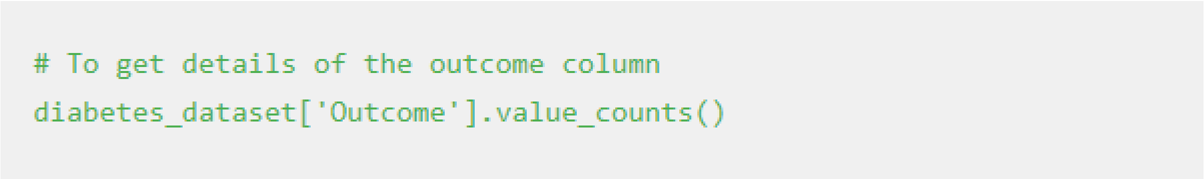
```
# To get the number of rows and columns in the dataset
diabetes_dataset.shape
#prints (768, 9)

# To get the statistical measures of the data
diabetes_dataset.describe()
```

## Output:



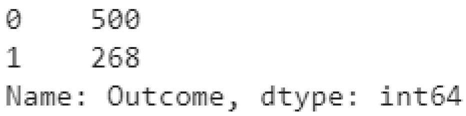
| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

And, it is clear that the Outcome column is the output variable. So let us explore more details about that column.

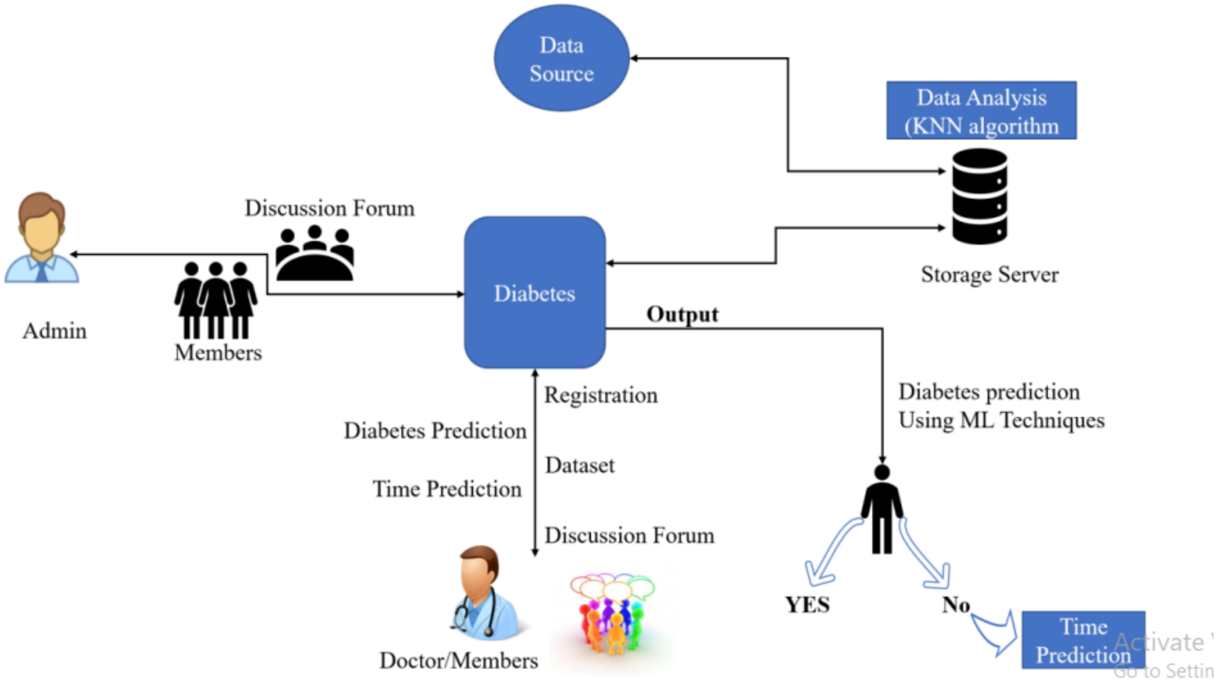


```
# To get details of the outcome column
diabetes_dataset['Outcome'].value_counts()
```

In the output, the value 1 means the person is having Diabetes, and 0 means the person is not having Diabetes. We can see the total count of people with and without Diabetes.

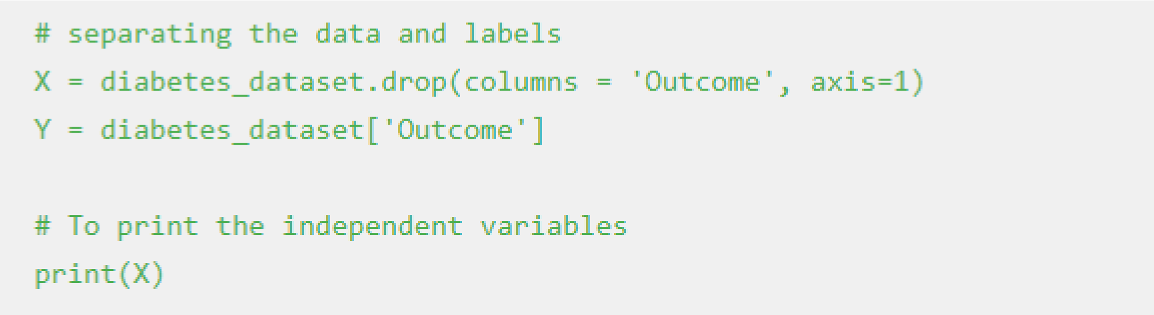


```
0    500
1    268
Name: Outcome, dtype: int64
```



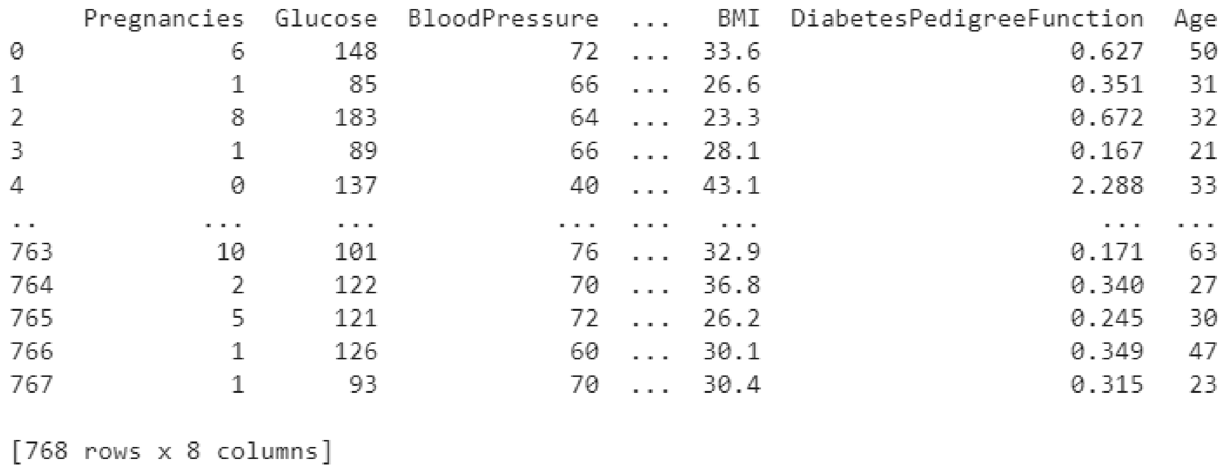
## Splitting the data

The next step in the building of the Machine learning model is splitting the data into training and testing sets. The training and testing data should be split in a ratio of 3:1 for better prediction results.



```
# separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']

# To print the independent variables
print(X)
```

## Output:



```
     Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Age
0              6      148             72  ...  33.6                     0.627   50
1              1       85             66  ...  26.6                     0.351   31
2              8      183             64  ...  23.3                     0.672   32
3              1       89             66  ...  28.1                     0.167   21
4              0      137             40  ...  43.1                     2.288   33
..           ...      ...            ...  ...   ...                       ...  ...
763           10      101             76  ...  32.9                     0.171   63
764            2      122             70  ...  36.8                     0.340   27
765            5      121             72  ...  26.2                     0.245   30
766            1      126             60  ...  30.1                     0.349   47
767            1       93             70  ...  30.4                     0.315   23

[768 rows x 8 columns]
```

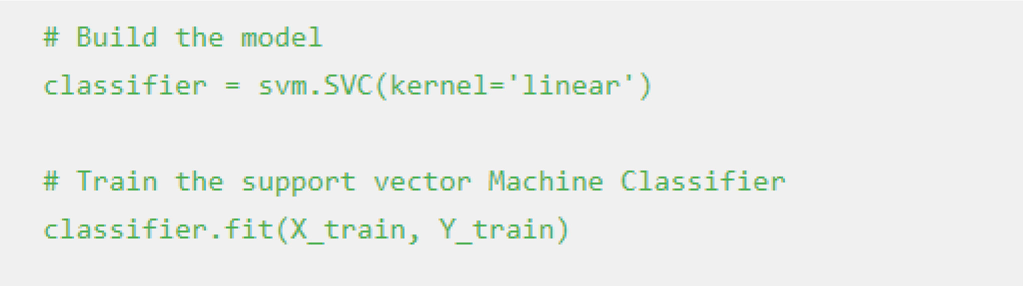
```
#Split the data into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y,
random_state=2)
print(X.shape, X_train.shape, X_test.shape)
```

## Output:

*(768, 8) (614, 8) (154, 8)*

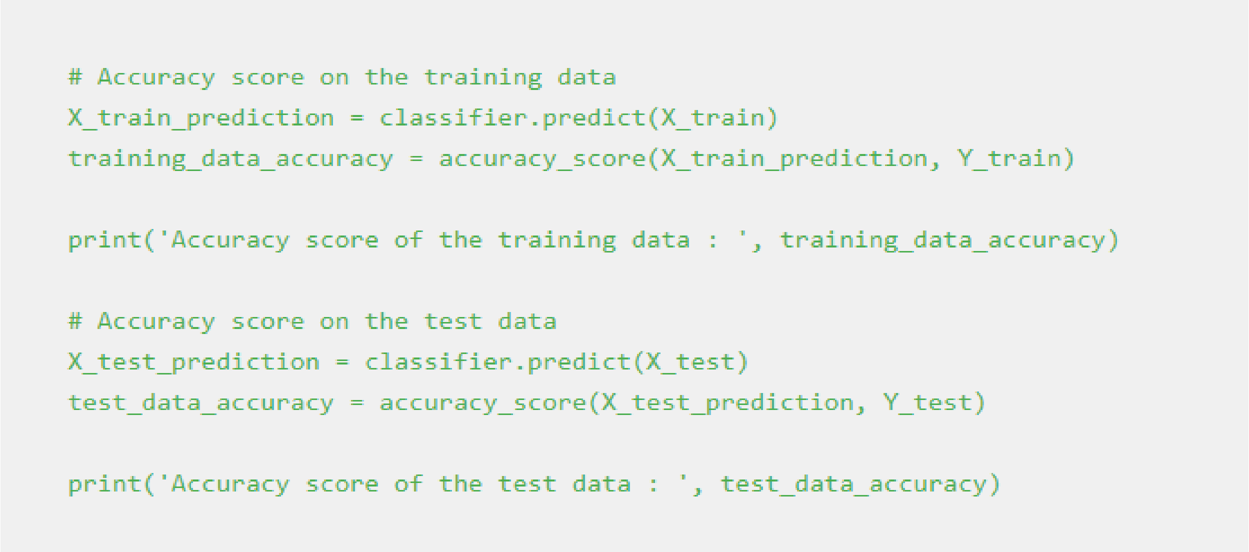
## Training the model

The next step is to build and train our model. We are going to use a Support vector classifier algorithm to build our model.



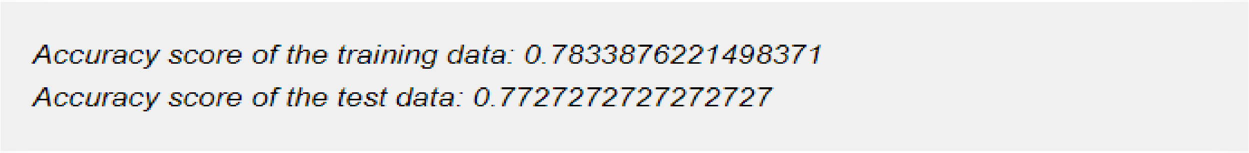
```
# Build the model
classifier = svm.SVC(kernel='linear')

# Train the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

After building the model, the model has to predict output with test data. After the prediction of the outcome with test data, we can calculate the accuracy score of the prediction results by the model.



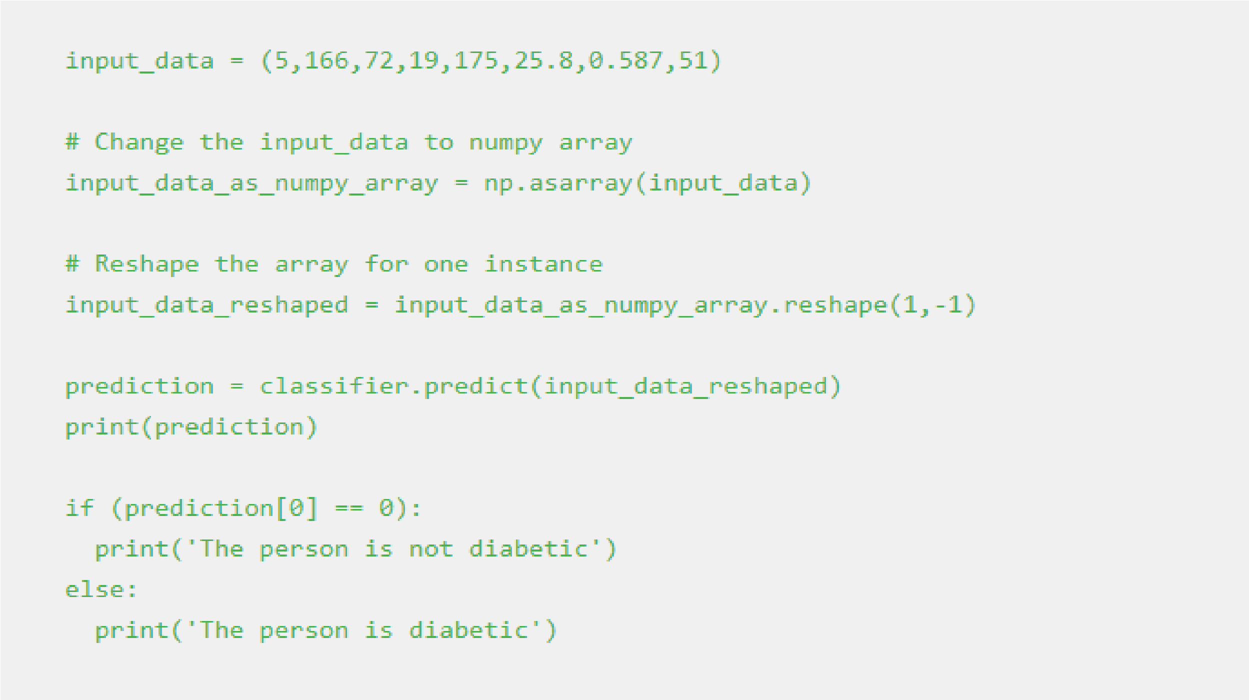
```
# Accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy score of the training data : ', training_data_accuracy)

# Accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score of the test data : ', test_data_accuracy)
```

## Output:



```
Accuracy score of the training data: 0.7833876221498371
Accuracy score of the test data: 0.7727272727272727
```

## 5. Evaluating the model

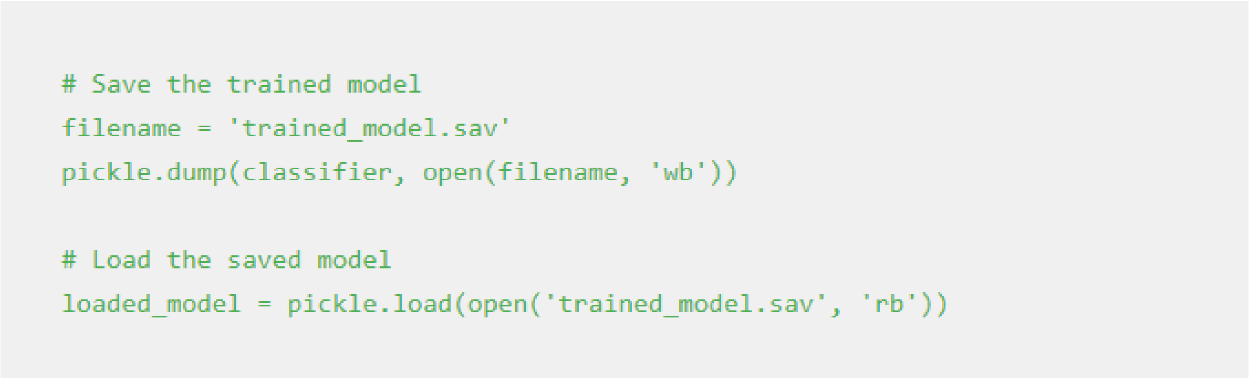


```
input_data = (5,166,72,19,175,25.8,0.587,51)

# Change the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = classifier.predict(input_data_reshaped)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

## Output:

*The person is diabetic*

## Saving the file



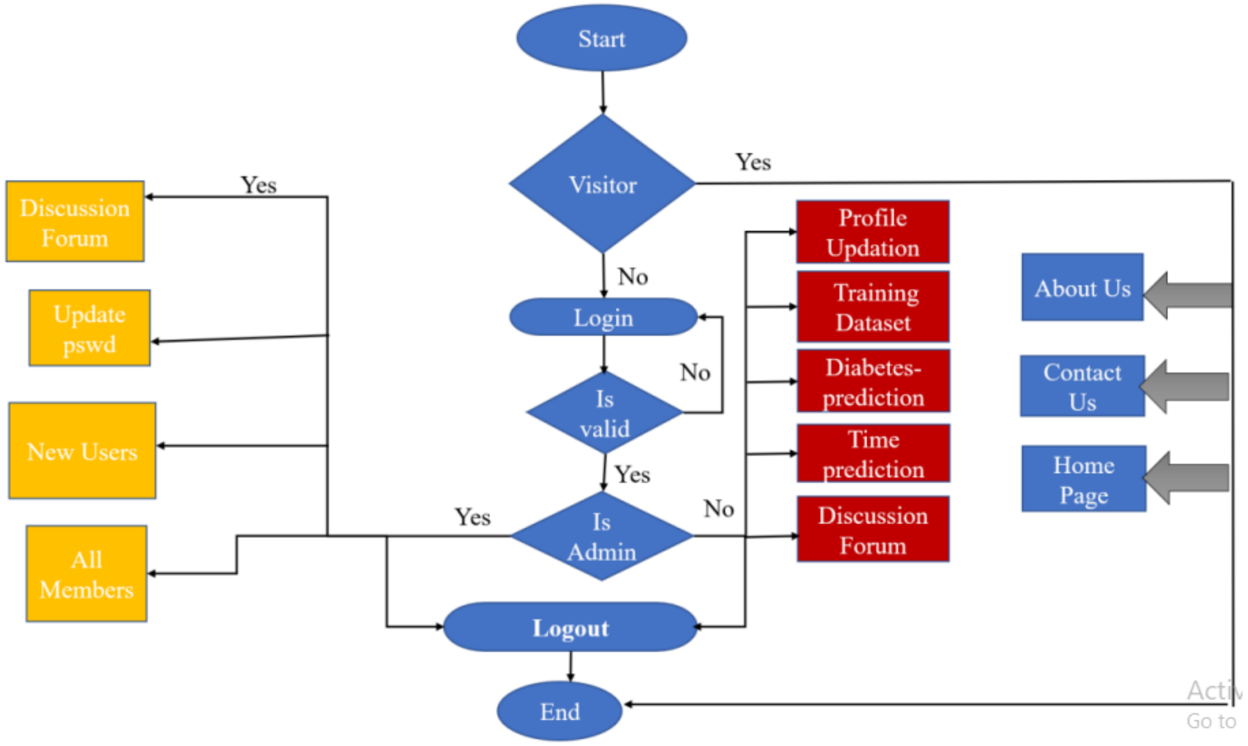
```
# Save the trained model
filename = 'trained_model.sav'
pickle.dump(classifier, open(filename, 'wb'))

# Load the saved model
loaded_model = pickle.load(open('trained_model.sav', 'rb'))
```

Once you run this code a new file named trained_model.sav will be saved in the project folder.

# Deploying the model

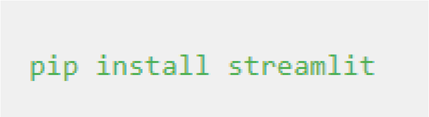
One of the most important and final steps in building a Machine Learning project is Model deployment. There are many frameworks available for deploying the Machine learning model on the web. Some of the most used Python frameworks are Django and Flask. But these frameworks require a little knowledge of languages such as HTML, CSS, and JavaScript.

So, a new framework known as Streamlit was introduced to deploy the Machine Learning model without the need to have the knowledge of Front End Languages. It is quite easy to deploy using Streamlit. So, we will use the Streamlit framework to deploy our model. Although Streamlit has many advantages over the other frameworks, lot more features are under development. If you are getting started in Machine Learning then this framework will be a perfect start to deploy your machine learning model on the web.



# Python Code to Deploy ML model using Streamlit

To install Streamlit run the following command in the command prompt or terminal.



```
pip install streamlit
```

Open a new Python file and put the following code.

## *App.py*

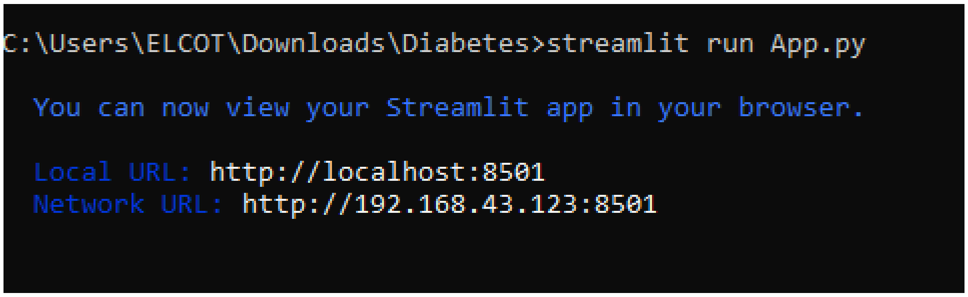
```
import numpy as np

import pickle

import streamlit as st

# Load the saved model

loaded_model = pickle.load(open('C:/Users/ELCOT/Downloads/trained_model.sav', 'rb'))



# Create a function for Prediction

def diabetes_prediction(input_data):

#Change the input_data to numpy array

input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array as we are predicting for one instance

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = loaded_model.predict(input_data_reshaped)
```

```
        print(prediction)
if (prediction[0] == 0):
    return 'The person is not diabetic'
 else:
        return 'The person is diabetic'
def main():
# Give a title
st.title('Diabetes Prediction Web App')
#To get the input data from the user
Pregnancies = st.text_input('Number of Pregnancies')
Glucose = st.text_input('Glucose Level')
BloodPressure = st.text_input('Blood Pressure value')
SkinThickness = st.text_input('Skin Thickness value')
Insulin = st.text_input('Insulin Level')
BMI = st.text_input('BMI value')
DiabetesPedigreeFunction = st.text_input('Diabetes Pedigree Function value')
Age = st.text_input('Age of the Person')

# Code for Prediction
diagnosis = ''
# Create a button for Prediction
```

```
if st.button('Diabetes Test Result'):

diagnosis = diabetes_prediction([Pregnancies, Glucose, BloodPressure,
SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age])

st.success(diagnosis)

if __name__ == '__main__':

main()
```

Save the file after pasting the code. And then to deploy using streamlit go to command prompt and run the following command.

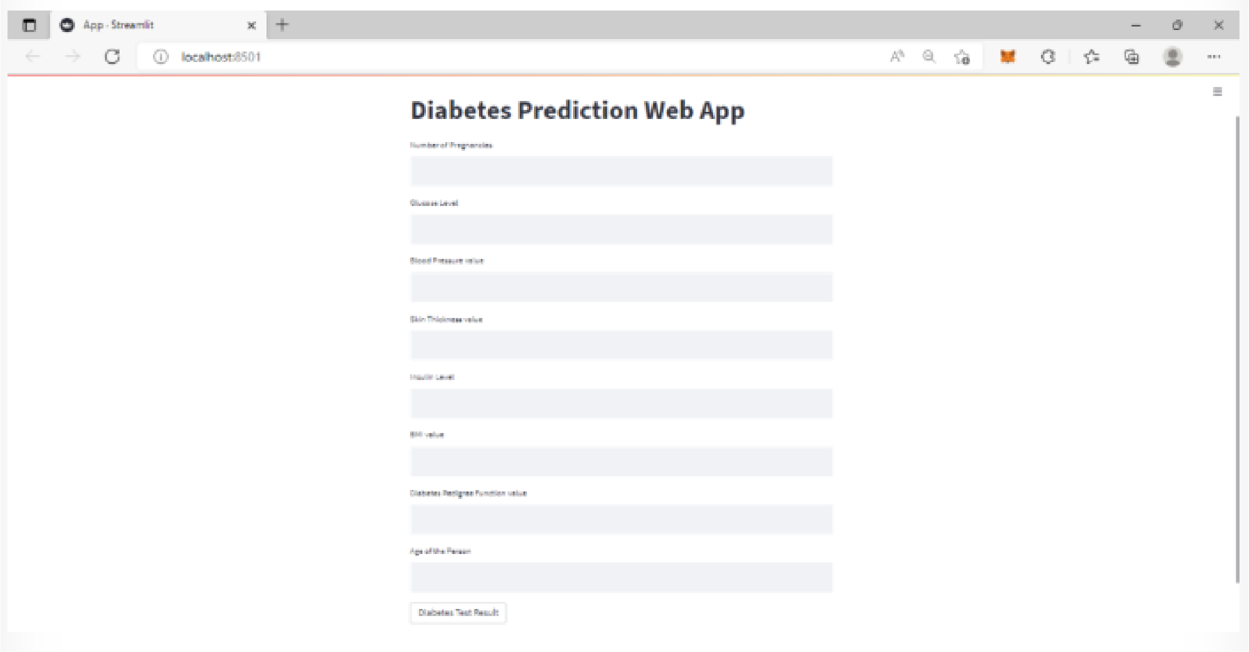
```
streamlit run App.py
(or)
streamlit run filename.py
```



```
C:\Users\ELCOT\Downloads\Diabetes>streamlit run App.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.43.123:8501
```

After running the command the web app will open in the localhost webserver. Otherwise, go to your browser and type *localhost:8501*. The following output will be shown.

## Output:



Diabetes Prediction Web App

Number of Pregnancies

Glucose Level

Blood Pressure value

Skin Thickness value

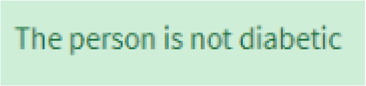
Insulin Level

BMI value

Diabetes Pedigree Function value

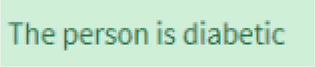
Age of the Person

Sample Input data for a person does not have diabetes is {1, 85, 66, 29, 0, 26.6, 0.351, 31}. These data as input will generate the following output in the web app.



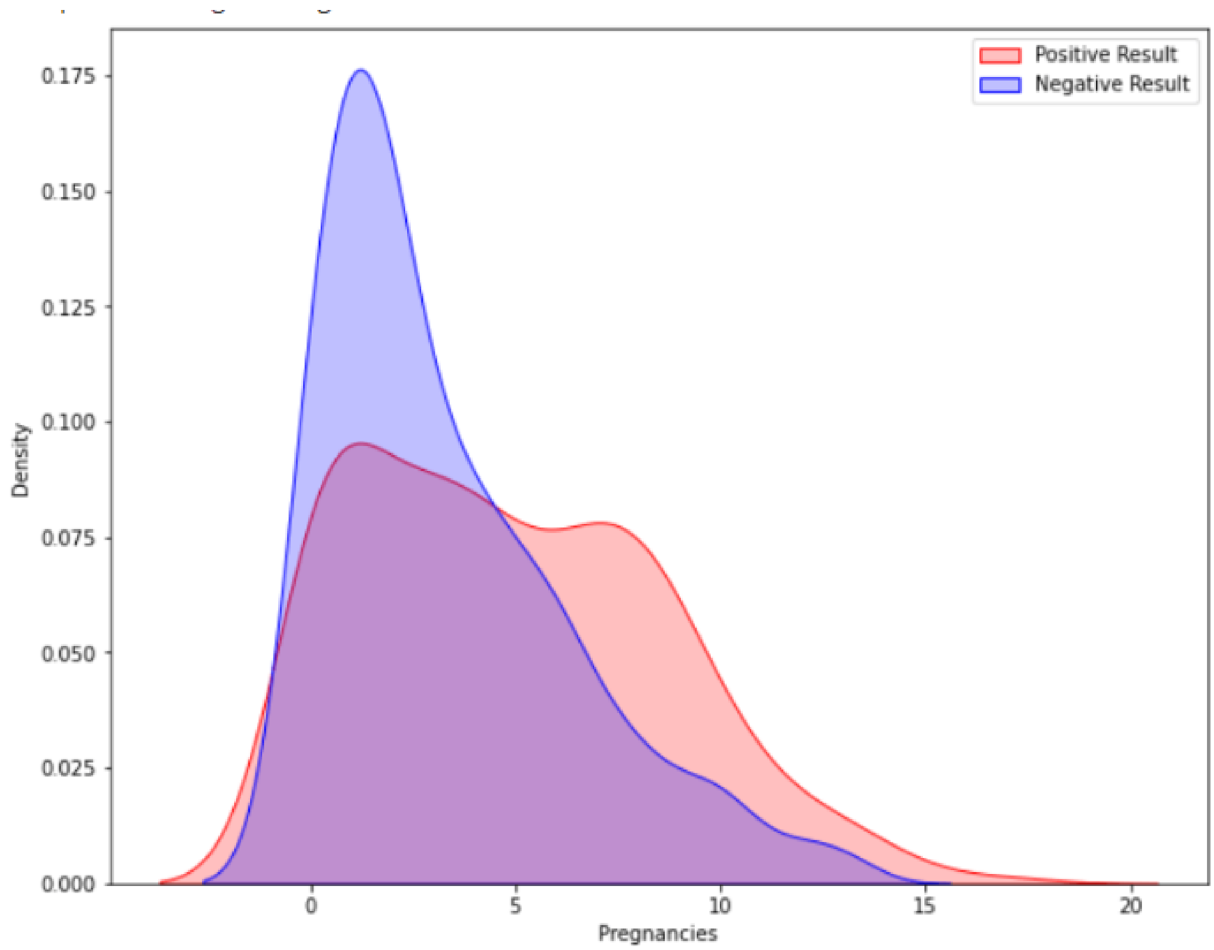
The person is not diabetic

Sample input data for a person who have diabetes is {6, 148, 72, 35, 0, 33.6, 0.627, 50}. These data as input will generate the following output in the web app.



The person is diabetic

## Sample output



### BMI (Body Mass Index):

Calculate BMI from height and weight if not already provided. BMI is a relevant feature as it's associated with diabetes risk.

### Age Categories:

Create age groups or categories to capture the potential nonlinear relationship between age and diabetes risk. For example, you can create groups like "young," "middle-aged," and "senior."

### Family History:

Represent family history of diabetes as a binary feature (1 for yes, 0 for no). This can be a significant predictor.

**Blood Pressure Categories:**

Categorize blood pressure measurements into different groups such as "Normal," "Prehypertension," "Hypertension Stage 1," and "Hypertension Stage 2."

**Glucose Level Transformation:**

Consider transforming glucose levels if needed. For example, you can convert continuous glucose values into categorical features, such as "Low," "Normal," and "High."

**Interaction Features:**

Create interaction features between pairs of variables. For example, you can create a feature that multiplies age by BMI to capture the combined effect of age and body mass.

**Polynomial Features:**

Introduce polynomial features of certain variables. For instance, add squared or cubed terms of variables that have a nonlinear relationship with the target.

**Medical Ratios:**

Create new features by calculating ratios, such as the ratio of glucose levels to insulin levels. Some ratios might provide better predictive power.

**Time-Based Features:**

If you have longitudinal data, consider creating features related to time, like the number of years since diagnosis or the number of medical check-ups in a year.

**One-Hot Encoding:**

Convert categorical variables (e.g., gender, ethnicity) into one-hot encoded binary features.

**Missing Value Indicators:**

If missing values are prevalent in the dataset, create binary indicators for missing values in certain variables. This helps the model learn how to handle missing data.

**Feature Scaling:**

Apply feature scaling techniques (e.g., standardization or normalization) to ensure that all features have the same scale, which can be important for algorithms like logistic regression or support vector machines.

**Feature Selection:**

Use feature selection techniques (e.g., Recursive Feature Elimination or feature importance from tree-based models) to identify the most important features for the model.

**Domain-Specific Features:**

Consult with healthcare experts to identify additional domain-specific features that might be relevant for diabetes prediction.

**Feature Crosses:**

Create new features by taking the product or division of two or more existing features. This can help capture complex relationships.

Remember to validate your feature engineering choices by evaluating the model's performance using appropriate metrics and techniques, such as cross-validation and hyperparameter tuning. Feature engineering is often an iterative process, and you may need to refine your feature set based on the model's performance and domain knowledge.

# Conclusion

In conclusion, machine learning offers a promising approach for diabetes prediction and risk assessment. By leveraging the power of data and advanced algorithms, it is possible to build models that can assist in identifying individuals at risk of diabetes. Here are some key takeaways:

**Data is Key:** High-quality, well-curated healthcare data is essential for building accurate diabetes prediction models. The quality of predictions is closely tied to the quality of the data.

**Feature Engineering:** Thoughtful feature engineering is crucial. The choice of features and their transformations can significantly impact the predictive performance of the model.

**Model Selection:** The choice of the machine learning algorithm should be based on the nature of the data and the problem at hand. Common choices include logistic regression, decision trees, random forests, support vector machines, and neural networks.

**Evaluation Metrics:** Model performance should be evaluated using appropriate metrics for classification tasks, such as accuracy, precision, recall, F1 score, and area under the ROC curve. It's important to consider the specific context and the trade-offs between false positives and false negatives.

**Validation and Cross-Validation:** To assess a model's generalization performance and minimize overfitting, use techniques like k-fold cross-validation.

**Interpretability:** In healthcare, model interpretability is critical. It's important to understand why a model makes certain predictions, especially for regulatory and ethical reasons.

**Data Privacy and Ethics:** Handling healthcare data requires strict adherence to privacy regulations (e.g., HIPAA in the United States) and ethical guidelines to protect patient confidentiality and rights.

**Model Deployment and Monitoring:** Deploying a model into a real-world healthcare setting requires careful consideration of integration, monitoring, and ongoing maintenance to ensure its continued effectiveness.

**Collaboration with Healthcare Professionals:** Machine learning models for diabetes prediction should be developed in collaboration with healthcare experts who can provide domain knowledge and validate the model's outputs.

**Ethical and Social Implications:** Be aware of the ethical and social implications of diabetes prediction models. Avoid perpetuating bias and disparities in healthcare outcomes.

In summary, machine learning for diabetes prediction has the potential to make a significant positive impact on healthcare by enabling early identification of at-risk individuals, personalized treatment plans, and improved patient outcomes. However, it must be approached with care, responsibility, and a deep understanding of both the data and the healthcare domain. Collaboration between data scientists, healthcare professionals, and data privacy experts is key to the successful development and deployment of such models.