**SUBJECT LINE : ML-MINOR-MAY**

**MACHINE LEARNING WITH PYTHON – VERZEO**

**MINOR PROJECT SUBMISSION**

**PROJECT TITLE : DIABETES PREDICTION**

**STUDENT DETAILS**

|  |  |
| --- | --- |
| **NAME** | SASIREKHA A |
| **MAIL ID** | sasirekha1692000@gmail.com |
| **BATCH NO** | 01 |
| **CONTACT NO** | 9566019039 |
| **SUBJECT LINE** | ML-MINOR-MAY |
| **DATE OF SUBMISSION** | 22 / 06 / 2021 |

**Submitted to : event@verzeo.com**

**DIABETES PREDICTION**

**INTRODUCTION :**

Diabetes is a common chronic disease and poses a great threat to human health. The characteristic of diabetes is that the blood glucose is higher than the normal level, which is caused by defective insulin secretion or its impaired biological effects, or . Diabetes can lead to chronic damage and dysfunction of various tissues, especially eyes, kidneys, heart, blood vessels . Diabetes can be divided into two categories, type 1 diabetes (T1D) and type 2 diabetes (T2D). Patients with type 1 diabetes are normally younger, mostly less than 30 years old. The typical clinical symptoms are increased thirst and frequent urination, high blood glucose levels. This type of diabetes cannot be cured effectively with oral medications alone and the patients are required insulin therapy. Type 2 diabetes occurs more commonly in middle-aged and elderly people, which is often associated with the occurrence of obesity, hypertension, dyslipidemia, arteriosclerosis, and other diseases. The aim of this system is to get the glucose level , insulin level , BMI , age and blood pressure of the patient and to predict whether they have diabetes or not .

**PROBLEM STATEMENT :**

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. 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.

**DATASET :** [**https://drive.google.com/file/d/1msEua9j4LwLhPeO6qSaZ8O4W7iLy6Orl/view?usp=sharing**](https://drive.google.com/file/d/1msEua9j4LwLhPeO6qSaZ8O4W7iLy6Orl/view?usp=sharing)

The data was collected and made available by “National Institute of Diabetes and Digestive and Kidney Diseases” as part of the [Pima Indians Diabetes Database](https://www.kaggle.com/uciml/pima-indians-diabetes-database/data). Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here belong to the Pima Indian heritage (subgroup of Native Americans), and are females of ages 21 and above. The following features have been provided to help us predict whether a person is diabetic or not:

* **Pregnancies:**Number of times pregnant
* **Glucose:** Plasma glucose concentration over 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 (a function which scores likelihood of diabetes based on family history)
* **Age:** Age (years)
* **Outcome:** Class variable (0 if non-diabetic, 1 if diabetic)

Note that the data does have some missing values (see Insulin = 0) in the samples in the previous figure. Ideally we could replace these 0 values with the mean value for that feature

**LIBRARIES USED :**

1. **Pandas** : Data Analysis and Manipulation tool
2. **Numpy** : For working with arrays
3. **Matplotlib** : For data Visualization
4. **Seaborn** : Data visualization library based on matplotlib
5. **Pickle :** Pickle in Python is primarily used in serializing and deserializing a Python object structure
6. **Warning** : Warning messages are displayed by warn() function defined in 'warning' module of Python's standard library
7. **Itertool:** Itertools is a module in python, it is used to iterate over data structures that can be stepped over using a for-loop
8. **Minmaxscaler :** Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.
9. **Train\_test\_split** : The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets.
10. **Logistic Regression** from sklearn.linear\_model : Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).
11. **KNeighboursClassifier from sklearn.neighbor**s : K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. ... KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
12. **SVC from sklearn.svm** : The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.
13. **DecisionTreeClassifier from sklearn.tree** : A decision tree is a flowchart-like structure in which each internal node represents a test on a feature
14. **RandomForestClassifier from sklearn.ensemble** : Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.
15. **Accuracy\_score from sklearn.metrics** : Accuracy classification score. In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y\_true
16. **Confusion matrix from sklearn.metrics** : By definition a confusion matrix is such that C i , j is equal to the number of observations known to be in group and predicted to be in group . Thus in binary classification, the count of true negatives is C 0 , 0 , false negatives is C 1 , 0 , true positives is C 1 , 1 and false positives is C 0 , 1 .
17. **Classification report from sklearn.metrics** : A Classification report is used to measure the quality of predictions from a classification algorithm

**PLATFORM :**

**Visual Studio Code and Google Collaboratory**

**Google Collaboratory** : A Model is developed and tested in collaboratory

**Visual Studio Code** : Used the collaboratory file in VS code and Vs code is used to develop front end for an application . Front end is developed using flask in python

**IMPLEMENTATION AND ANALYSIS**

**STEP 1 : IMPORT LIBRARIES AND DATASET**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import pickle

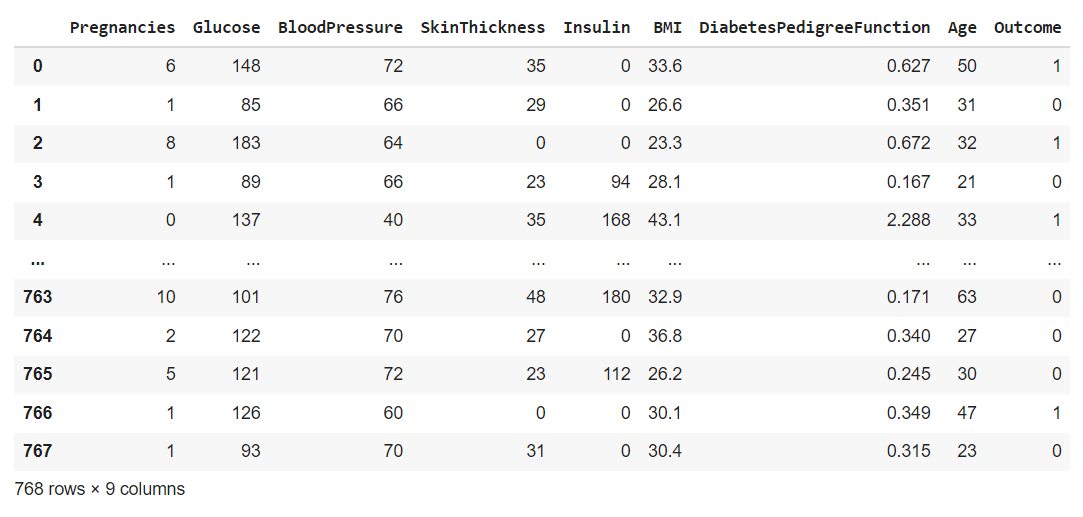
import warnings

warnings.filterwarnings('ignore')

#Import Dataset

dataset = pd.read\_csv('diabetes.csv')

dataset

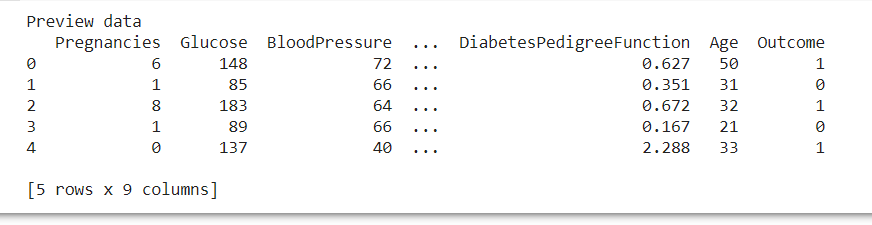


**DESCRIPTIVE STATISTICS :**

**Preview data :**

print("Preview data")

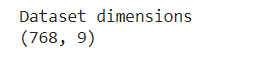
print(dataset.head())



**DATASET DIMENSION :**

print("Dataset dimensions")

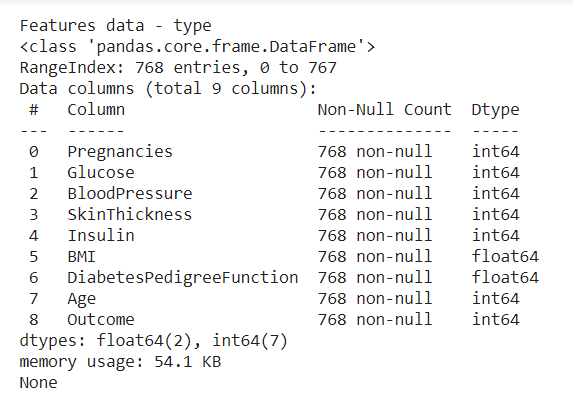
print(dataset.shape)



**FEATURE – DATA TYPE**

print("Features data - type")

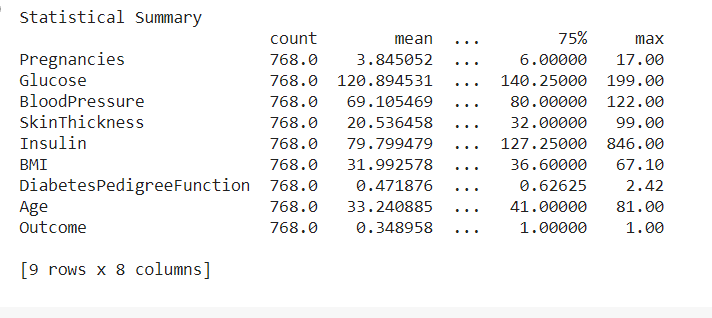
print(dataset.info())

****

**STATISTICAL SUMMARY :**

print("Statistical Summary ")

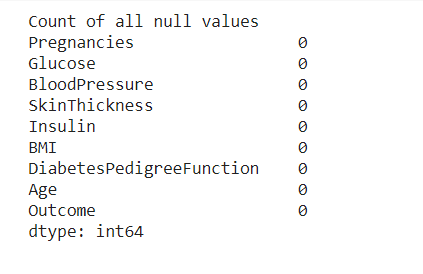
print(dataset.describe().T)



**COUNT ALL NULL VALUES**

print("Count of all null values ")

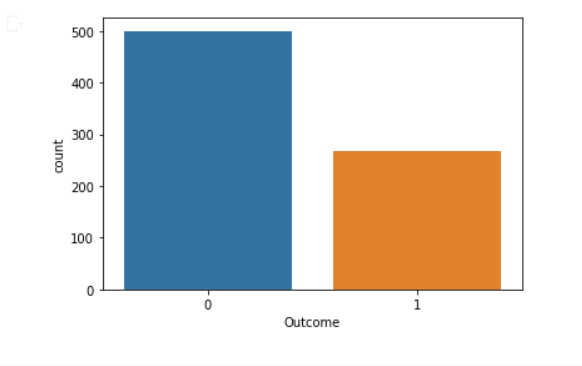
dataset.isnull().sum()



**DATA VISUALIZATION :**

sns.countplot( x = 'Outcome' , data = dataset )

plt.show()

****

**HISTOGRAM OF EACH FEATURE :**

import itertools

col = dataset.columns[:8]

plt.subplots(figsize = (20, 15))

length = len(col)

for i, j in itertools.zip\_longest(col, range(length)):

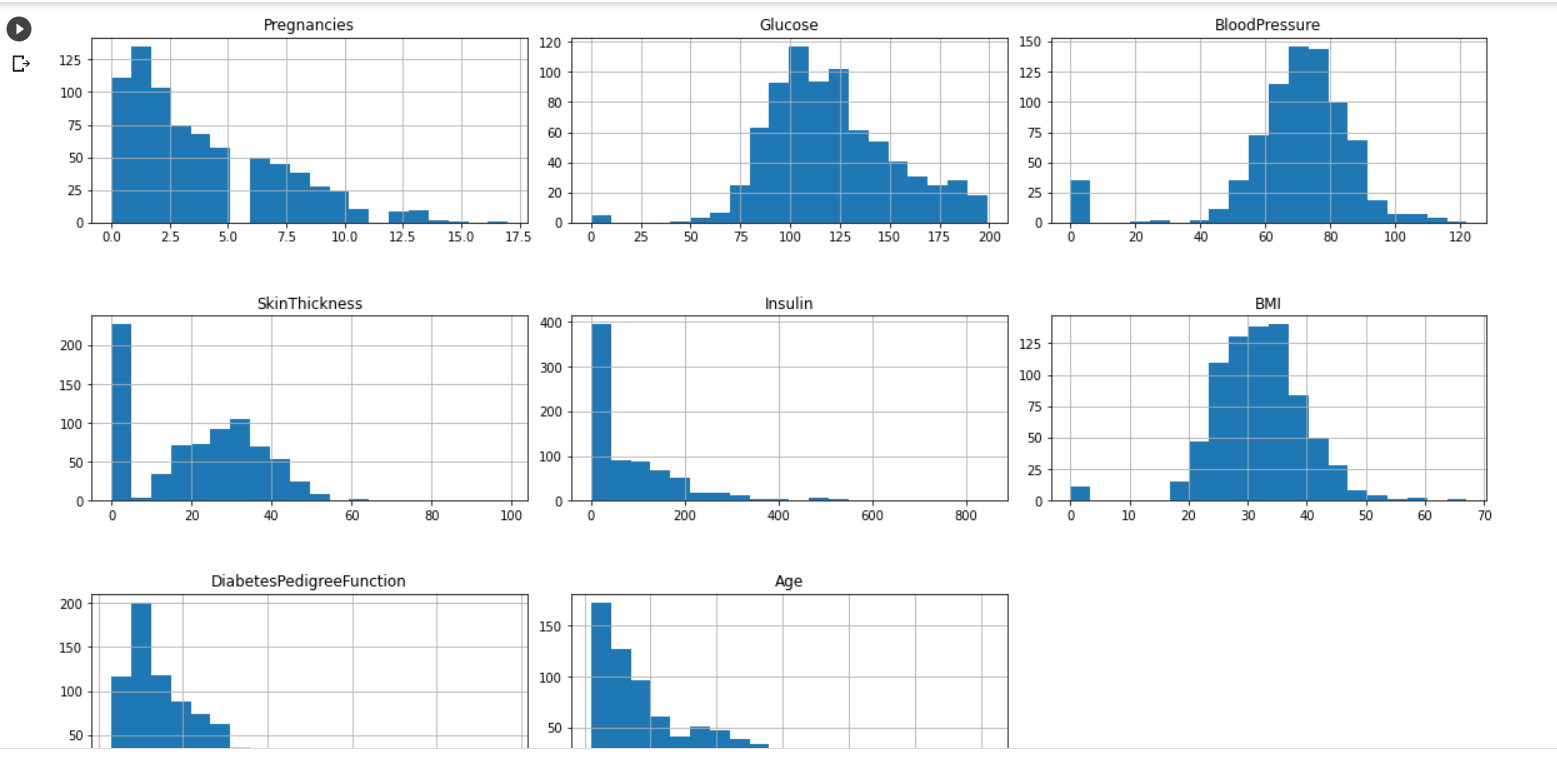
    plt.subplot((length/2), 3, j + 1)

    plt.subplots\_adjust(wspace = 0.1,hspace = 0.5)

    dataset[i].hist(bins = 20)

    plt.title(i)

plt.show()



**PAIRPLOT :**

sns.pairplot(data = dataset, hue = 'Outcome')

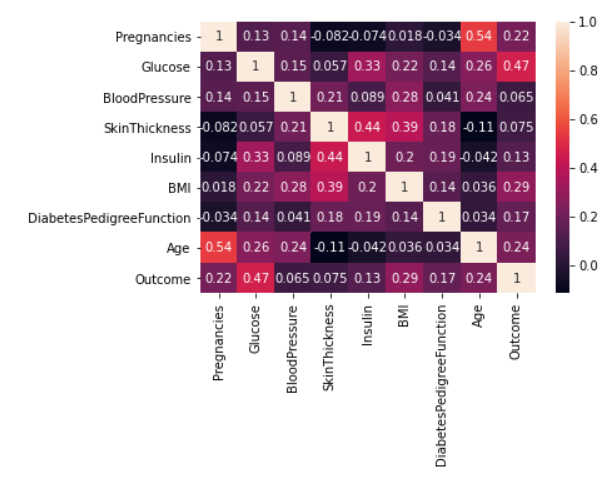
plt.show()



**HEATMAP :**

sns.heatmap(dataset.corr(), annot = True)

plt.show()



**DATA PREPROCESSING :**

dataset\_new = dataset

# Replace zero with NaN

dataset\_new[["Pregnancies" , "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI" , "DiabetesPedigreeFunction", "Age"]] = dataset\_new[["Pregnancies","Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age"]].replace(0, np.NaN)

# Replacing NaN with mean values

dataset\_new["Pregnancies"].fillna(dataset\_new["Pregnancies"].mean(), inplace = True)

dataset\_new["Glucose"].fillna(dataset\_new["Glucose"].mean(), inplace = True)

dataset\_new["BloodPressure"].fillna(dataset\_new["BloodPressure"].mean(), inplace = True)

dataset\_new["SkinThickness"].fillna(dataset\_new["SkinThickness"].mean(), inplace = True)

dataset\_new["Insulin"].fillna(dataset\_new["Insulin"].mean(), inplace = True)

dataset\_new["BMI"].fillna(dataset\_new["BMI"].mean(), inplace = True)

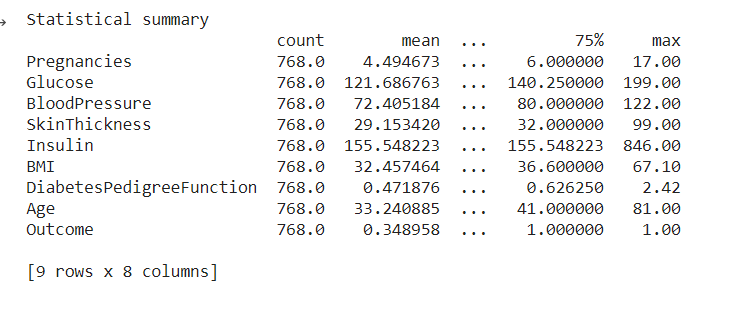
dataset\_new["DiabetesPedigreeFunction"].fillna(dataset\_new["DiabetesPedigreeFunction"].mean(), inplace = True)

dataset\_new["Age"].fillna(dataset\_new["Age"].mean(), inplace = True)

# Statistical summary

print("Statistical summary")

print(dataset\_new.describe().T)



**FEATURE SCALING USING MINMAXSCALER :**

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0,1))

dataset\_scaled = sc.fit\_transform(dataset\_new)

dataset\_scaled = pd.DataFrame(dataset\_scaled)

# Selecting features - [Glucose, Insulin, BMI, Age]

X = dataset\_scaled.iloc[:, [1, 2 , 4 , 5 , 7]].values

Y = dataset\_scaled.iloc[:, 8].values

# Splitting X and Y

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.20, random\_state = 42, stratify = dataset\_new['Outcome'] )

# Checking dimensions

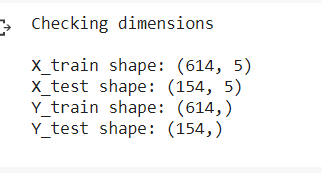
print("Checking dimensions\n")

print("X\_train shape:", X\_train.shape)

print("X\_test shape:", X\_test.shape)

print("Y\_train shape:", Y\_train.shape)

print("Y\_test shape:", Y\_test.shape)



**DATA MODELLING :**

**LOGISTICS REGRESSION :**

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression(random\_state = 42)

logreg.fit(X\_train, Y\_train)

**K NEAREST NEIGHBOR ALGORITHM :**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors = 24, metric = 'minkowski', p = 2)

knn.fit(X\_train, Y\_train)

**SUPPORT VECTOR CLASSIFIER ALGORITHM :**

from sklearn.svm import SVC

svc = SVC(kernel = 'linear', random\_state = 42)

svc.fit(X\_train, Y\_train)

**NAÏVE BAYES ALGORITHM :**

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X\_train, Y\_train)

**DECISION TREE ALGORITHM :**

from sklearn.tree import DecisionTreeClassifier

dectree = DecisionTreeClassifier(criterion = 'entropy', random\_state = 42)

dectree.fit(X\_train, Y\_train)

**RANDOM FOREST ALGORITHM :**

from sklearn.ensemble import RandomForestClassifier

ranfor = RandomForestClassifier(n\_estimators = 11, criterion = 'entropy', random\_state = 42)

ranfor.fit(X\_train, Y\_train)

**MAKING PREDICTION ON TEST DATASET :**

Y\_pred\_logreg = logreg.predict(X\_test)

Y\_pred\_knn = knn.predict(X\_test)

Y\_pred\_svc = svc.predict(X\_test)

Y\_pred\_nb = nb.predict(X\_test)

Y\_pred\_dectree = dectree.predict(X\_test)

Y\_pred\_ranfor = ranfor.predict(X\_test)

**MODEL EVALUATION :**

from sklearn.metrics import accuracy\_score

accuracy\_logreg = accuracy\_score(Y\_test, Y\_pred\_logreg)

accuracy\_knn = accuracy\_score(Y\_test, Y\_pred\_knn)

accuracy\_svc = accuracy\_score(Y\_test, Y\_pred\_svc)

accuracy\_nb = accuracy\_score(Y\_test, Y\_pred\_nb)

accuracy\_dectree = accuracy\_score(Y\_test, Y\_pred\_dectree)

accuracy\_ranfor = accuracy\_score(Y\_test, Y\_pred\_ranfor)

**ACCURACY ON TEST SET**

print("Logistic Regression: " + str(accuracy\_logreg \* 100))

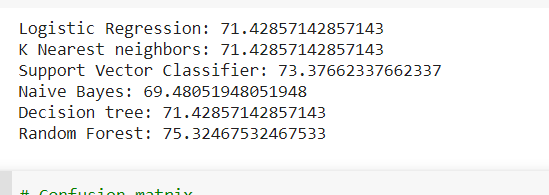
print("K Nearest neighbors: " + str(accuracy\_knn \* 100))

print("Support Vector Classifier: " + str(accuracy\_svc \* 100))

print("Naive Bayes: " + str(accuracy\_nb \* 100))

print("Decision tree: " + str(accuracy\_dectree \* 100))

print("Random Forest: " + str(accuracy\_ranfor \* 100))



**CONFUSION MATRIX :**

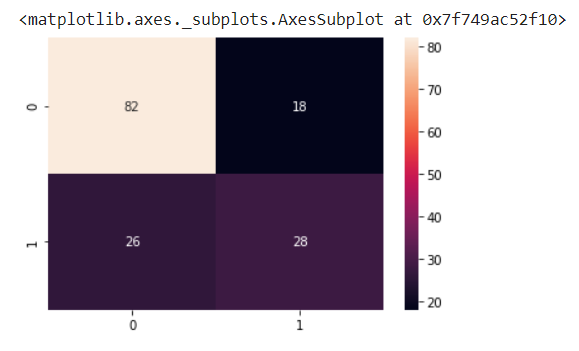
from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(Y\_test, Y\_pred\_knn)

cm

# Heatmap of Confusion matrix

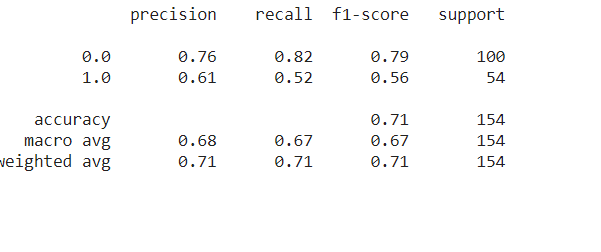
sns.heatmap(pd.DataFrame(cm), annot=True)



**CLASSIFICATION REPORT**

from sklearn.metrics import classification\_report

print(classification\_report(Y\_test, Y\_pred\_knn))



**SAVING THE MODEL :**

pickle.dump(ranfor, open('model.pkl','wb'))

model = pickle.load(open('model.pkl','rb'))

**LINK :**

**https://colab.research.google.com/drive/1gA6Oz1Hw\_9Tm6dW70KzsQM\_PkbSJqp01?usp=sharing**

**app.py**

import numpy as np

import pandas as pd

from flask import Flask, request, jsonify, render\_template

import pickle

app = Flask(\_\_name\_\_)

model = pickle.load(open('model.pkl', 'rb'))

dataset = pd.read\_csv('diabetes.csv')

dataset\_X = dataset.iloc[:,[1, 2 , 4 , 5 , 7]].values

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature\_range = (0,1))

dataset\_scaled = sc.fit\_transform(dataset\_X)

@app.route('/')

def home():

    return render\_template('index.html')

@app.route('/predict',methods=['POST'])

def predict():

    '''

    For rendering results on HTML GUI

    '''

    float\_features = [float(x) for x in request.form.values()]

    final\_features = [np.array(float\_features)]

    prediction = model.predict( sc.transform(final\_features) )

    if prediction == 1:

        pred = "You have Diabetes, please consult a Doctor."

    elif prediction == 0:

        pred = "You don't have Diabetes."

    output = pred

    return render\_template('index.html', prediction\_text='{}'.format(output))

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

**index .html**

<!DOCTYPE html>

<html >

<!--From https://codepen.io/frytyler/pen/EGdtg-->

<head>

  <meta charset="UTF-8">

  <title>Diabetes Predictor</title>

  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>

<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>

<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>

<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

</head>

<body>

 <div class="login">

  <h1>Diabetes Predictor</h1>

    <form action="{{ url\_for('predict')}}"method="post">

      <input type="text" name="Glucose Level" placeholder="Glucose Level" required="required" />

      <input type="text" name="Blood Pressure" placeholder="Blood Pressure Level" required="required" />

      <input type="text" name="Insulin" placeholder="Insulin" required="required" />

      <input type="text" name="BMI" placeholder="BMI" required="required" />

      <input type="text" name="Age" placeholder="Age" required="required" />

    <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>

    </form>

   <br>

   <br>

   {{ prediction\_text }}

 </div>

</body>

</html>

**Style.css**

@import url(https://fonts.googleapis.com/css?family=Open+Sans);

.btn { display: inline-block; \*display: inline; \*zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; line-height: 18px; color: #333333; text-align: center;text-shadow: 0 1px 1px rgba(255, 255, 255, 0.75); vertical-align: middle; background-color: #f5f5f5; background-image: -moz-linear-gradient(top, #ffffff, #e6e6e6); background-image: -ms-linear-gradient(top, #ffffff, #e6e6e6); background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#ffffff), to(#e6e6e6)); background-image: -webkit-linear-gradient(top, #ffffff, #e6e6e6); background-image: -o-linear-gradient(top, #ffffff, #e6e6e6); background-image: linear-gradient(top, #ffffff, #e6e6e6); background-repeat: repeat-x; filter: progid:dximagetransform.microsoft.gradient(startColorstr=#ffffff, endColorstr=#e6e6e6, GradientType=0); border-color: #e6e6e6 #e6e6e6 #e6e6e6; border-color: rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.25); border: 1px solid #e6e6e6; -webkit-border-radius: 4px; -moz-border-radius: 4px; border-radius: 4px; -webkit-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); -moz-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); cursor: pointer; \*margin-left: .3em; }

.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }

.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }

.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transition: background-position 0.1s linear; -moz-transition: background-position 0.1s linear; -ms-transition: background-position 0.1s linear; -o-transition: background-position 0.1s linear; transition: background-position 0.1s linear; }

.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: #ffffff; }

.btn-primary.active { color: rgba(255, 255, 255, 0.75); }

.btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-gradient(top, #6eb6de, #4a77d4); background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#6eb6de), to(#4a77d4)); background-image: -webkit-linear-gradient(top, #6eb6de, #4a77d4); background-image: -o-linear-gradient(top, #6eb6de, #4a77d4); background-image: linear-gradient(top, #6eb6de, #4a77d4); background-repeat: repeat-x; filter: progid:dximagetransform.microsoft.gradient(startColorstr=#6eb6de, endColorstr=#4a77d4, GradientType=0);  border: 1px solid #3762bc; text-shadow: 1px 1px 1px rgba(0,0,0,0.4); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.5); }

.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; background-color: #4a77d4; }

.btn-block { width: 100%; display:block; }

\* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {

    width: 100%;

    height:100%;

    font-family: 'Open Sans', sans-serif;

    background: #092756;

    color: #fff;

    font-size: 18px;

    text-align:center;

    letter-spacing:1.2px;

    background: -moz-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%),-moz-linear-gradient(top,  rgba(57,173,219,.25) 0%, rgba(42,60,87,.4) 100%), -moz-linear-gradient(-45deg,  #670d10 0%, #092756 100%);

    background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -webkit-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -webkit-linear-gradient(-45deg,  #670d10 0%,#092756 100%);

    background: -o-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -o-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -o-linear-gradient(-45deg,  #670d10 0%,#092756 100%);

    background: -ms-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -ms-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -ms-linear-gradient(-45deg,  #670d10 0%,#092756 100%);

    background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), linear-gradient(to bottom,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), linear-gradient(135deg,  #670d10 0%,#092756 100%);

    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );

}

.login {

    position: absolute;

    top: 40%;

    left: 50%;

    margin: -150px 0 0 -150px;

    width:400px;

    height:400px;

}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }

input {

    width: 100%;

    margin-bottom: 10px;

    background: rgba(0,0,0,0.3);

    border: none;

    outline: none;

    padding: 10px;

    font-size: 13px;

    color: #fff;

    text-shadow: 1px 1px 1px rgba(0,0,0,0.3);

    border: 1px solid rgba(0,0,0,0.3);

    border-radius: 4px;

    box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px rgba(255,255,255,0.2);

    -webkit-transition: box-shadow .5s ease;

    -moz-transition: box-shadow .5s ease;

    -o-transition: box-shadow .5s ease;

    -ms-transition: box-shadow .5s ease;

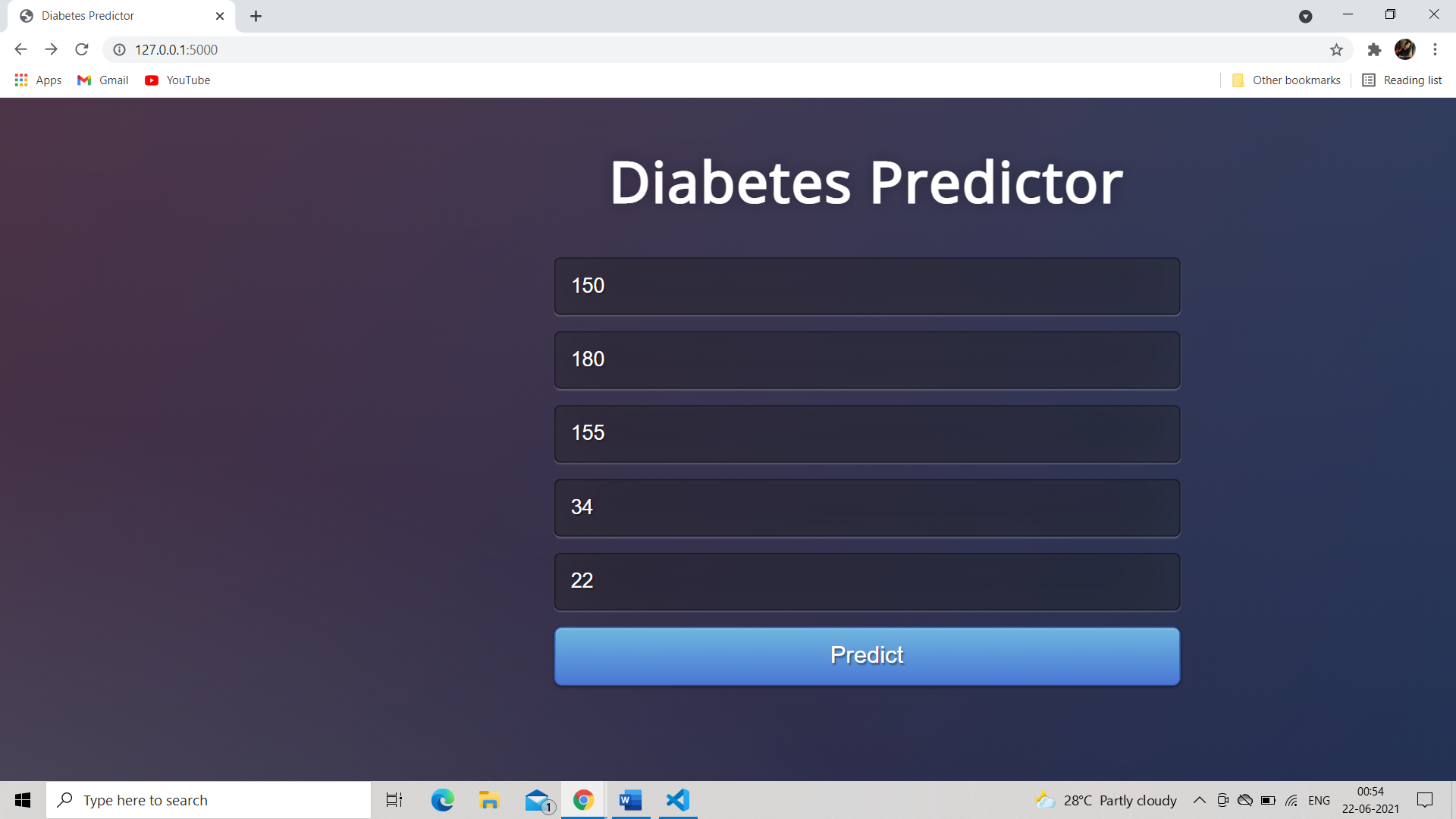
    transition: box-shadow .5s ease;

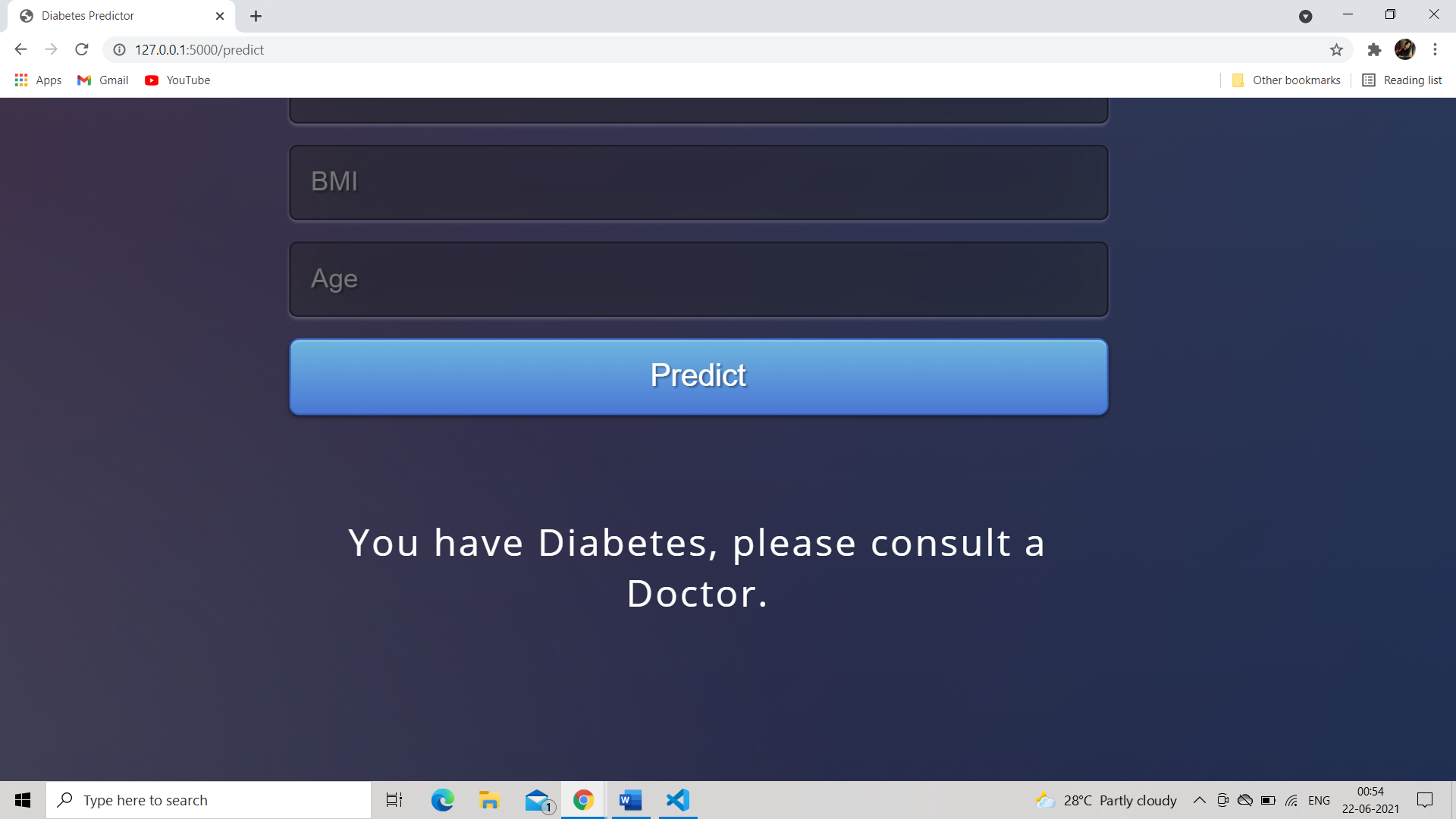
}

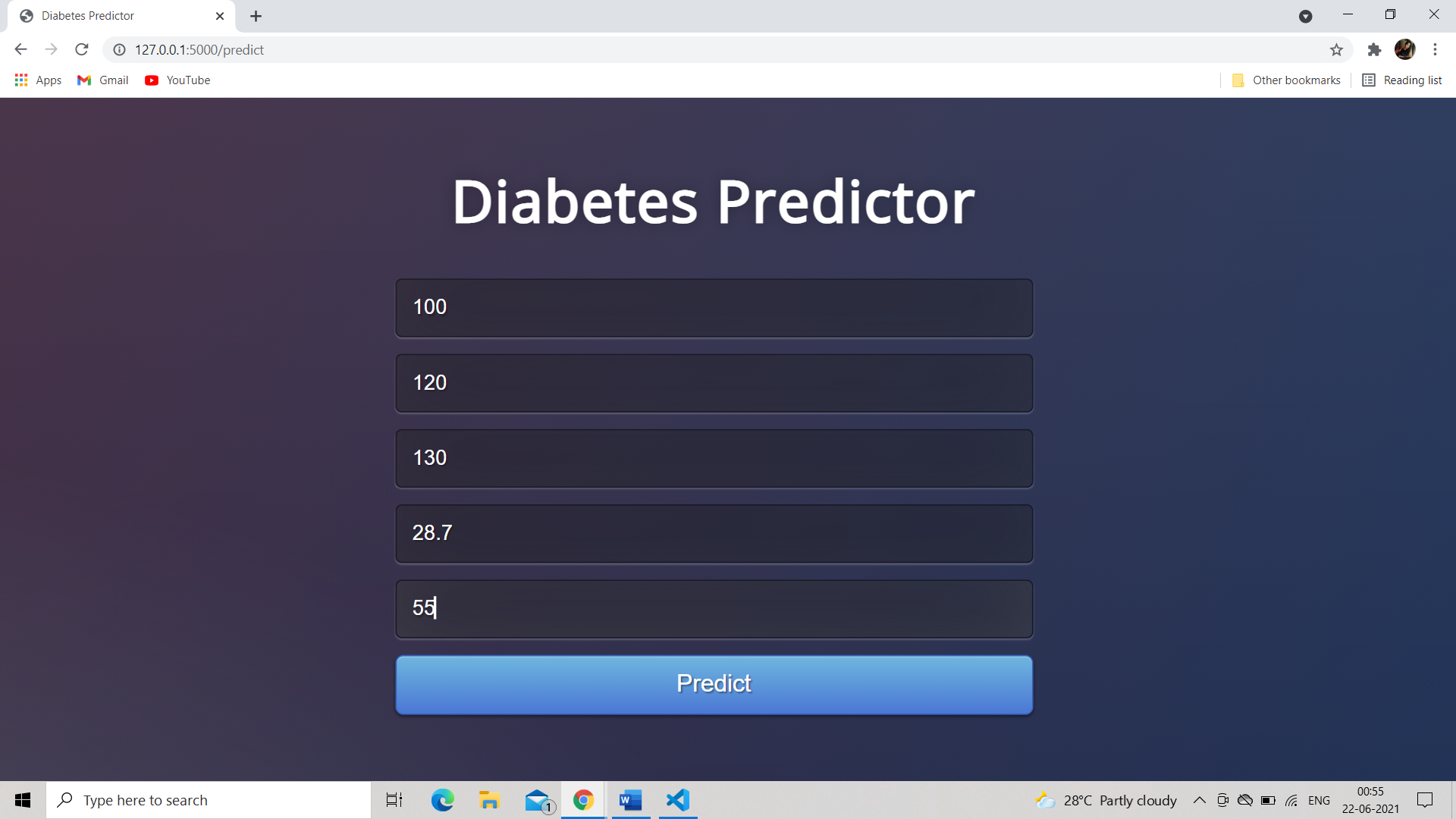
input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0 1px 1px rgba(255,255,255,0.2); }

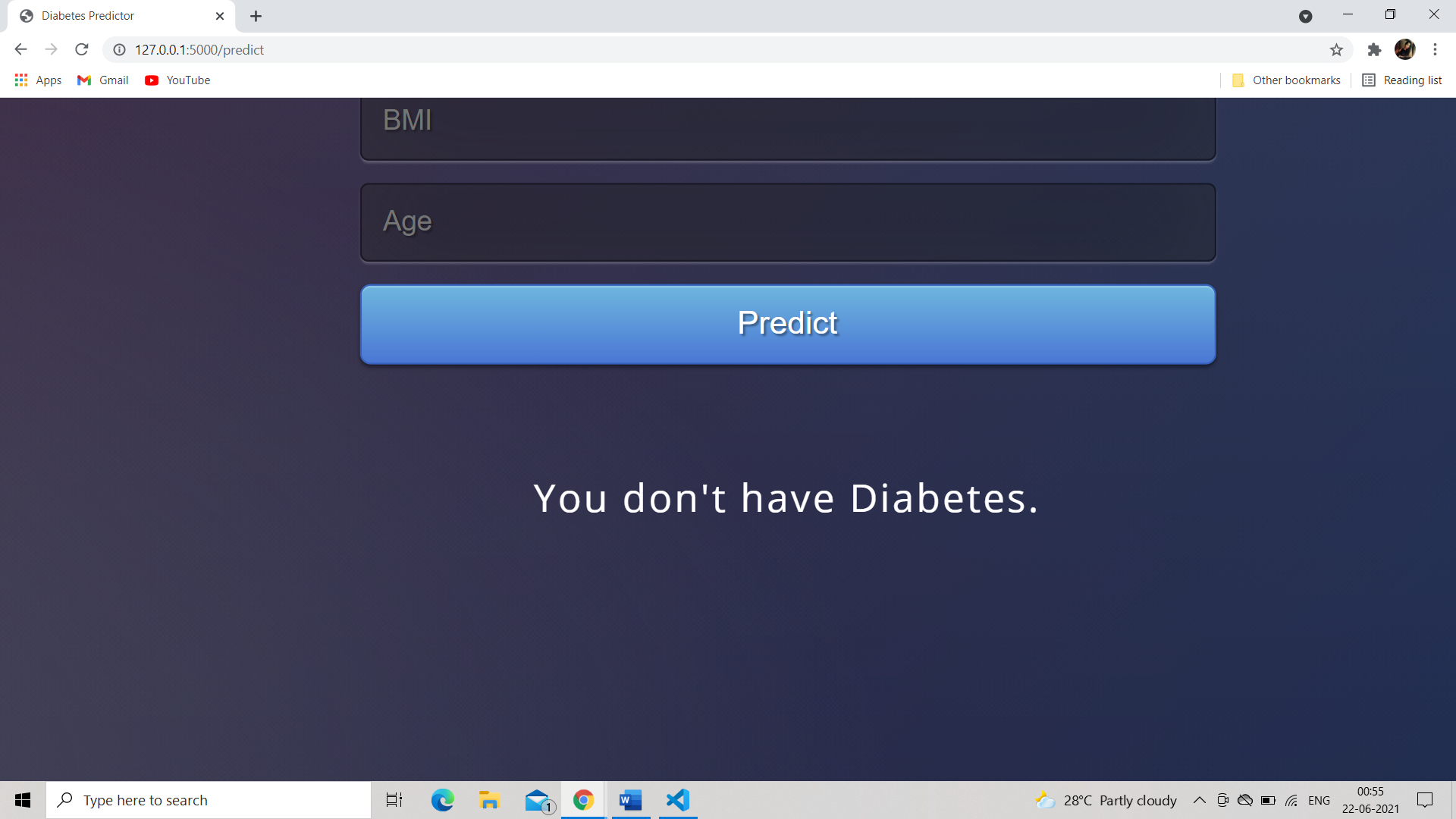
**OUTPUT SCREENSHOT :**











**CONCLUSION :**

Thus an application is developed successfully to predict the diabetes using Random Forest Classifier with an accuracy of 75.32%