

***Faculty of Science and Technology***

**Assignment Coversheet**

|  |  |
| --- | --- |
| **Student ID number &**  **Student Name** | U3248258  Hongliang Wang |
| **Unit name** | Software Technology |
| **Unit number** | 4483 |
| **Unit Tutor** | Gurpreet Kaur |
| **Assignment name** | ST1 Capstone Project – COLT 1 2023 |
| **Due date** | April 30 2023 |
| **Date submitted** | April 30 2023 |

**You must keep a photocopy or electronic copy of your assignment.**

**Student declaration**

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

**Signature of student:**  **Date: April 30 2023**

Table of Contents

[Introduction 1](#_Toc131590041)

[Methodology 2](#_Toc131590042)

[Stage 1: Algorithm Design Stage 2](#_Toc131590043)

[Dataset Description 3](#_Toc131590044)

[Exploratory Data Analysis 3](#_Toc131590045)

[Predictive Data Analytics Stage 11](#_Toc131590046)

[Model Preparation and Development 14](#_Toc131590047)

[Stage 2: Algorithm Implementation Stage 22](#_Toc131590048)

[Stage 3: Software Deployment Stage 22](#_Toc131590049)

[Conclusions 23](#_Toc131590050)

[References 23](#_Toc131590051)

Github Link: <https://github.com/Liangzainb/Capstone_Project>

# Introduction

This report describes the details of Python Capstone Project for ST1 unit within the scope of the project requirements provided in the assignment handout [1]. I have decided to work on the project using a diabetes dataset available in both UCI [2] and Kaggle data repositories [3].

Diabetes is a chronic disease that occurs when the pancreas does not produce enough insulin or the body cannot effectively use the insulin it produces. Insulin is a hormone that regulates blood sugar. Hyperglycemia, also known as elevated blood sugar or elevated blood sugar, is a common effect of uncontrolled diabetes that over time can severely damage many systems in the body, especially the nerves and blood vessels.There are different types of diabetes, including type 1 diabetes (characterized by insufficient insulin production and requiring daily insulin injections), type 2 diabetes (which affects how the body uses sugar for energy), gestational diabetes (high blood sugar below the diagnostic value of gestational diabetes), and impaired glucose tolerance and impaired fasting glucose (intermediate conditions in the transition between normal and diabetes).Diabetes is a leading cause of blindness, kidney failure, heart attack, stroke and lower limb amputation. Diabetes and diabetes-related kidney disease were responsible for an estimated 2 million deaths in 2019. Eating a healthy diet, getting regular physical activity, maintaining a normal body weight, and avoiding smoking are ways to prevent or delay the onset of type 2 diabetes. The consequences of diabetes can be avoided or delayed through diet, physical activity, medication, and regular screening and treatment for complications.

There are currently four ways to diagnose diabetes: random (or occasional) plasma glucose testing,Fasting plasma glucose testing, oral glucose tolerance test, A1c (glycosylated hemoglobin). Therefore, it would be beneficial to develop a quick alternative to the questionnaire format to replace the current standard. This method can preliminarily judge whether the respondents have diabetes.

This report presents the details of prototype software platform, in terms of several Python software tools developed as part of this capstone project, based on a data driven scientific approach, involving exploratory data analysis, predictive analytics and implementation as a desktop Tkinter application, and online web-based Flask/ Streamlit application. The details of the methodology used is presented in the next Section.This report presents the details of prototype software platform, in terms of several Python software tools developed as part of this capstone project, based on a data driven scientific approach, involving exploratory data analysis, predictive analytics and implementation as a desktop Tkinter application, and online web-based Flask/ Streamlit application. The details of the methodology used is presented in the next Section.

# Methodology

The methodology used for developing the software platform involves 3 stages as outlined below:

1. Design and development of decision support algorithms based on exploratory data analysis and predictive analytics, for identifying the best performing algorithm for solving a real world problem.
2. Implementation of best performing algorithm as a desktop Tkinter software tool.
3. Deployment of the tool as a web or cloud enabled platform tool.

## Stage 1: Algorithm Design Stage

Stage 1 is most important preliminary stage and depending on the complexity of the problem and dataset used, the design of algorithms for exploratory data analysis and predictive analytics algorithms will vary. However, the workflow for algorithm development will be as outlined in the Figure 1 schematic shown below:

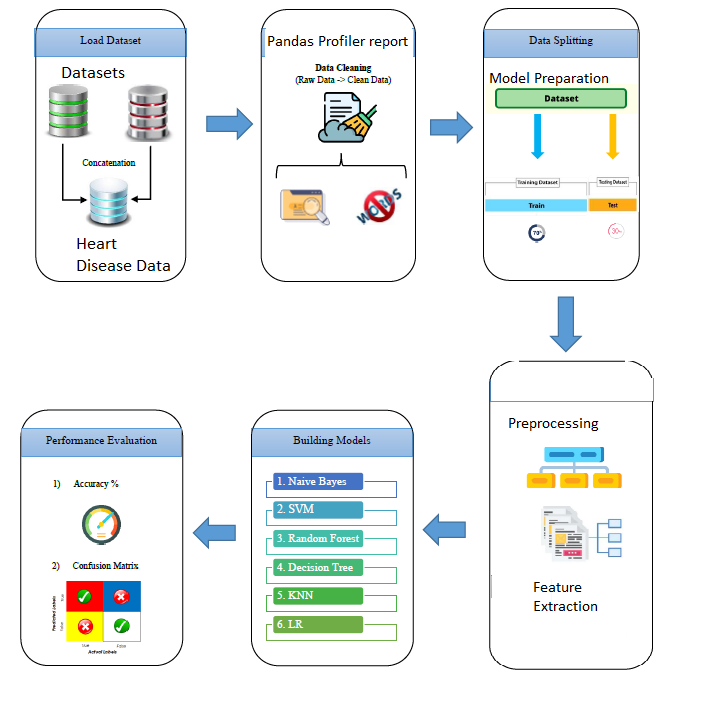


Figure 1: Schematic for Algorithm Design Methodology for Heart Disease Prediction

The details of each building block in Figure 1 schematic for algorithm design is described in the next few sections.

### Dataset Description

This project has only one dataset, which is publicly available from UCI Machine Learning Repository [2] and Kaggle [3]. The dataset contains 520 observations, 16 features and 1 object/class attribute. These 16 features include the diagnostic test results for the patient's symptoms above, as well as other relevant patient information. The target/categorical variables include the outcome Diabetes, which indicates whether the patient has diabetes, where negative indicates the absence of diabetes, and the label positive indicates the presence of diabetes. Therefore, the task at hand was to develop a software tool to predict the outcome of diabetes using the results of the aforementioned symptom query tests and patient information. This dataset has been collected using direct questionnaires from the patients of Sylhet Diabetes Hospital in Sylhet, Bangladesh and approved by a doctor [3].

### Exploratory Data Analysis

The first phase of the software development activity involved understanding the data, basic exploratory data analysis and visualisation. Google Colab was chosen as the experimental environment as it incorporates virtual hardware and resources which does not require additional physical hardware requirement and can be ran directly of a web browser. The python language was used to create the scripts which ran directly on online Jupyter notebook using Google Colab with the help of free google account created, and by saving all the notebook files virtually on google drive without additional configurations. Before the exploratory data analysis can begin, some of the python libraries for EDA need to be imported and dataset acquired, by using the following Python script

from google.colab import drive

drive.mount("/content/drive")

#Import Required Packages for EDA

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import missingno as msno

import plotly.graph\_objects as go

import plotly.express as px

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

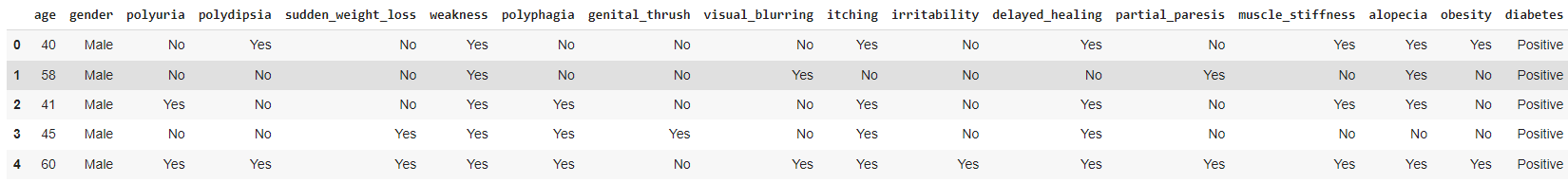
#Read the dataset/s

df = pd.read\_csv('/content/drive/MyDrive/Capstone Project/diabetes.csv')

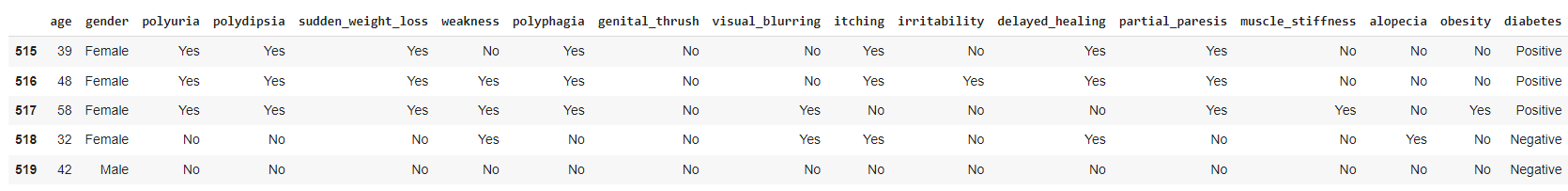
1. The EDA starts with understanding the basic description of data as described next:

#1. Checking description(first 5 and last 5 rows)

df.head()



df.tail() #last 5 rows



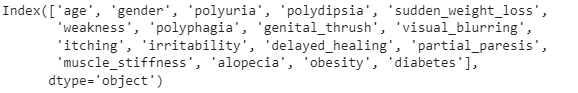
#rows and columns-data shape(attributes & samples)

df.shape

(520, 17)

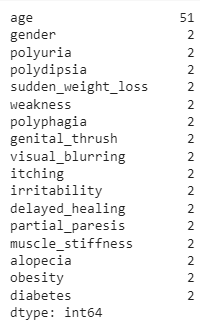
# name of the attributes

df.columns



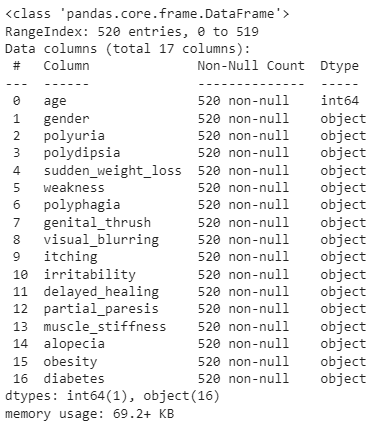
#unique values for each attribute

df.nunique()



#Complete info about data frame

df.info()



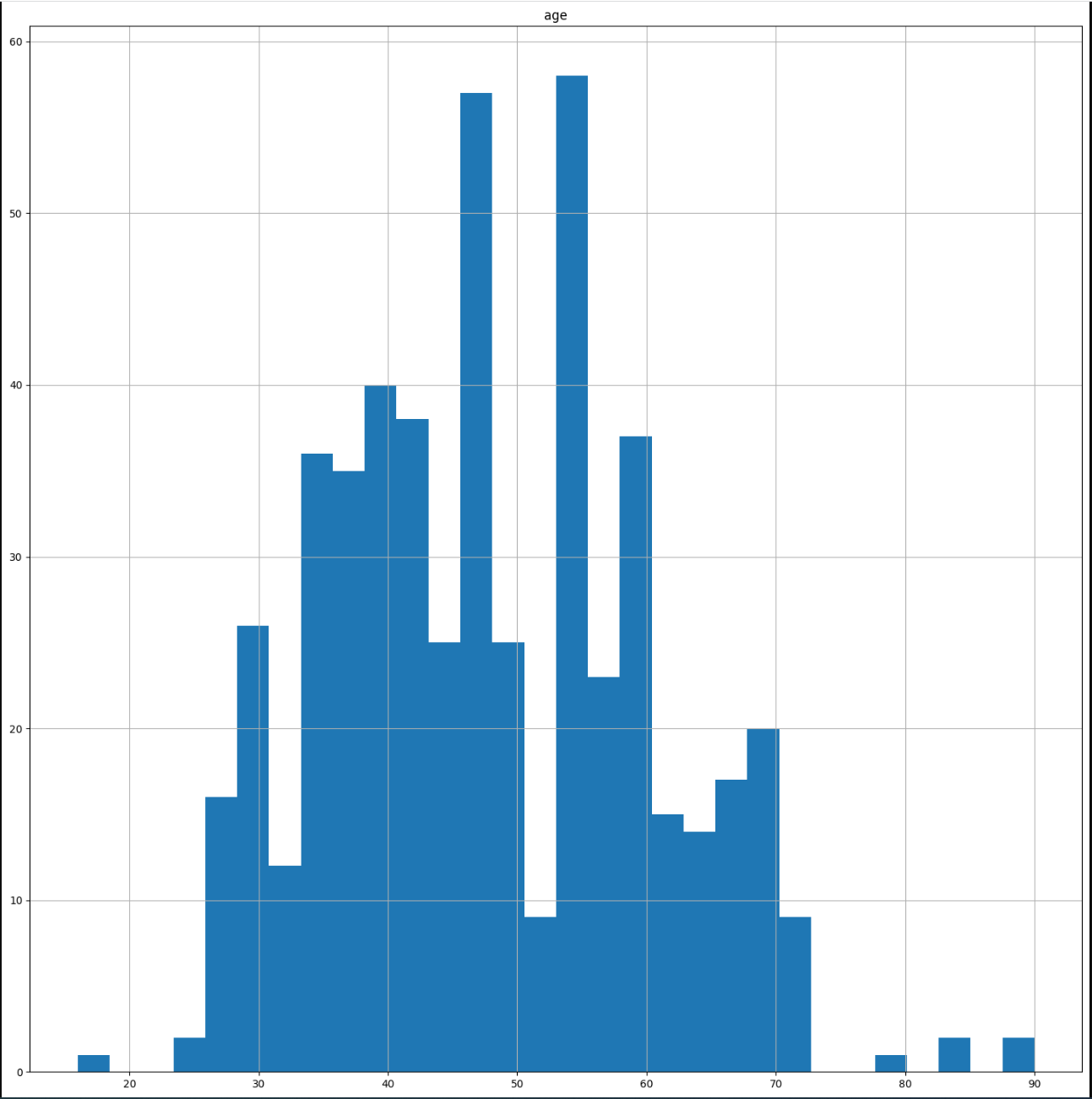
#3. Visualising data  distribution in detail

fig = plt.figure(figsize =(18,18))

ax=fig.gca()

df.hist(ax=ax,bins =30)

plt.show()



#checking target value distribution

print(df.diabetes.value\_counts())

fig, ax = plt.subplots(figsize=(5,4))

name = ["Positive", "Negative"]

ax = df.diabetes.value\_counts().plot(kind='bar')

ax.set\_title("diabetes", fontsize = 13, weight = 'bold')

ax.set\_xticklabels (name, rotation = 0)

# To calculate the percentage

totals = []

for i in ax.patches:

    totals.append(i.get\_height())

total = sum(totals)

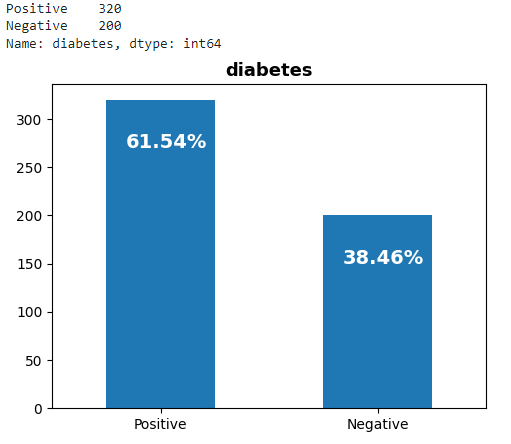
for i in ax.patches:

    ax.text(i.get\_x()+.09, i.get\_height()-50, \

            str(round((i.get\_height()/total)\*100, 2))+'%', fontsize=14,

                color='white', weight = 'bold')

plt.tight\_layout()



!pip install <https://github.com/pandas-profiling/pandas-profiling/archive/master.zip>

#obtain full profiler report

#restart kernel

#re-run import libraries and data

import pandas as pd

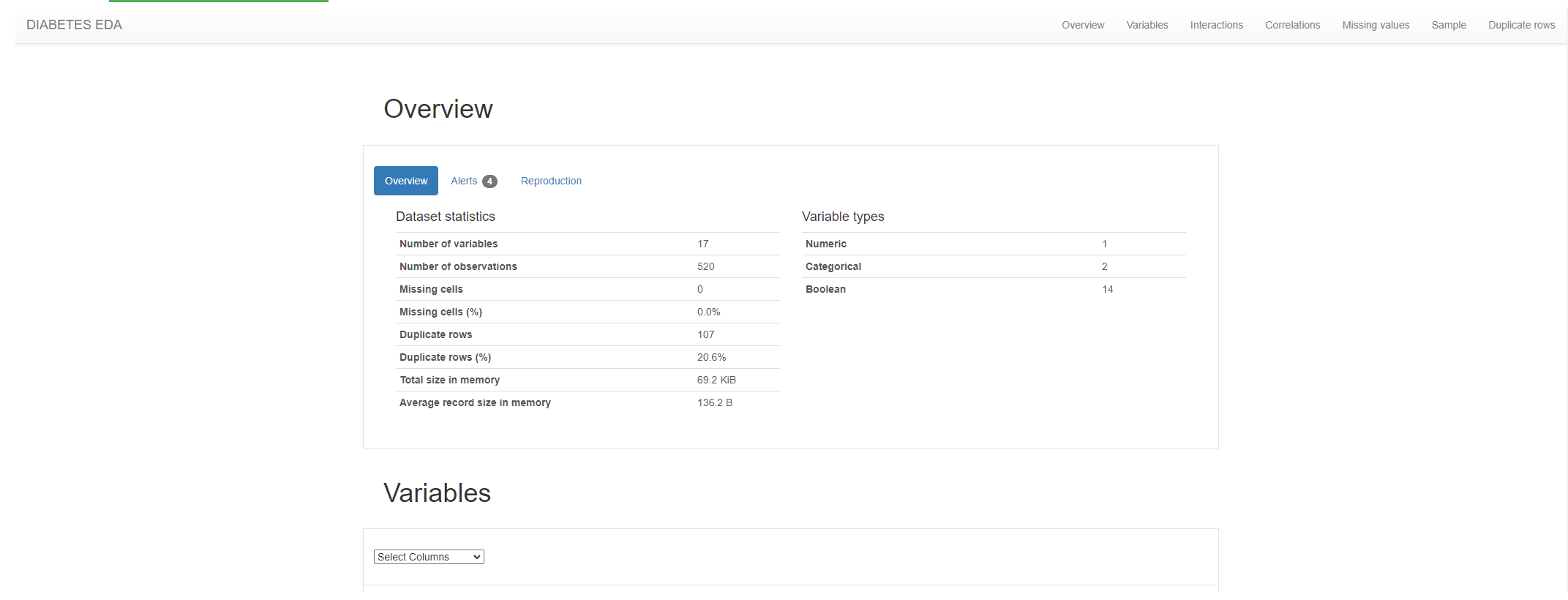
import numpy as np

from pandas\_profiling import ProfileReport

profile = ProfileReport(df,title="DIABETES EDA",

                        html={'style':{'full\_width':True}})

profile.to\_notebook\_iframe()



### Predictive Data Analytics Stage

For predictive analytics, several processing steps are required. These include pre-processing, classifier comparison to identify the best machine learning classifier and performance evaluation with different objective metrics, such as accuracy, classification report, confusion matrix, ROC-AUC curve and prediction report was obtained using the Python scikit-learn package. Each of these steps are described next.

* Pre-processing: Since the dataset consists of a combination of continuous and categorical attributes/variables, there is a need to pre-process the data with attribute transformation, standardization and normalisation. We used scikit-learn’s OrdinalEncoder() function to perform attribute transformation.
* Normalisation of the independent values of the dataframe by was done by dropping the target from the dataframe, normalising it, and then reattaching the target to the dataframe:-

#pre-processing

from sklearn.exceptions import DataDimensionalityWarning

#encode object columns to integers

from sklearn import preprocessing

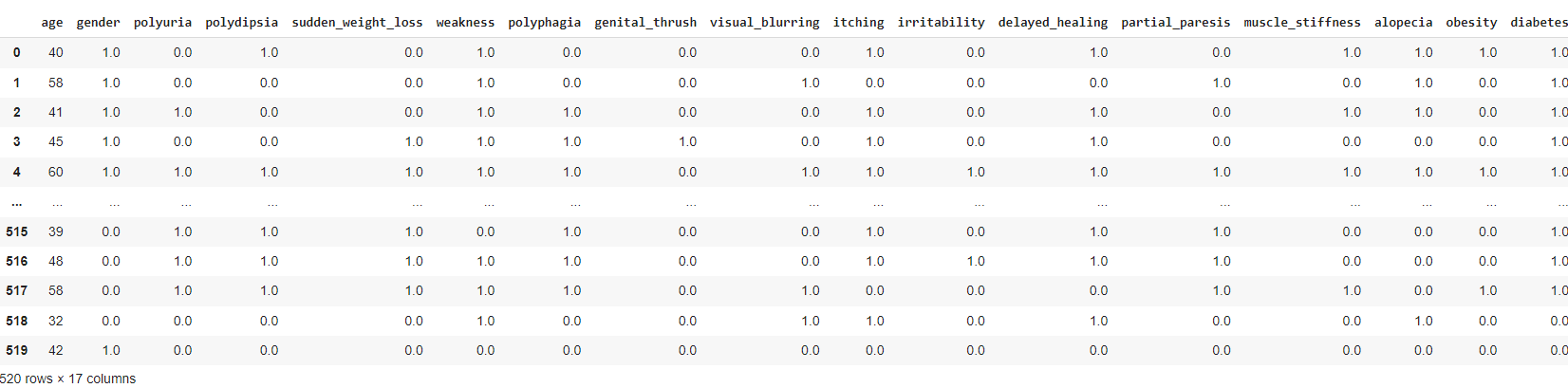
from sklearn.preprocessing import OrdinalEncoder

for col in df:

  if df[col].dtype =='object':

    df[col]=OrdinalEncoder().fit\_transform(df[col].values.reshape(-1,1))

df



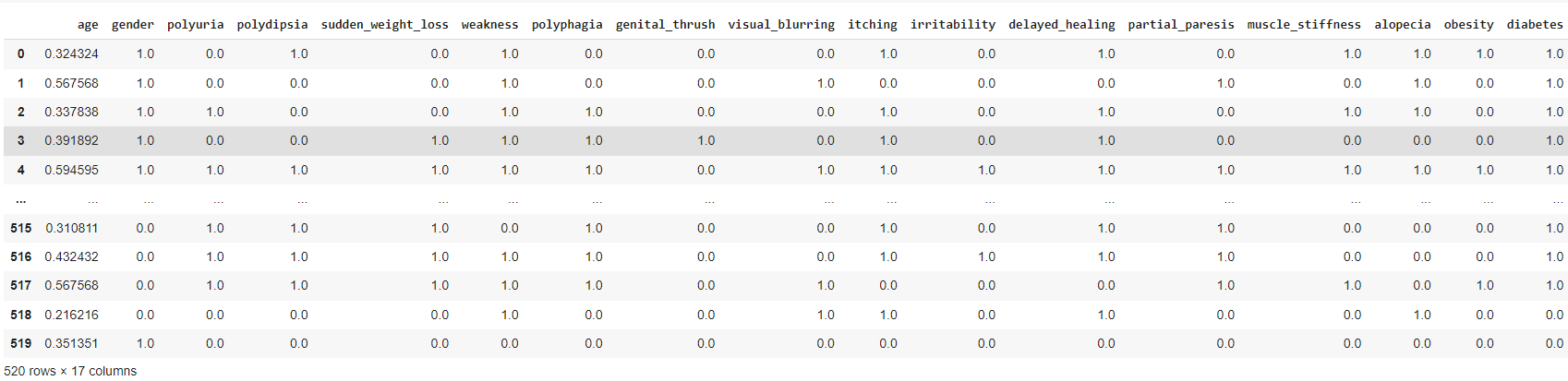
class\_label =df['diabetes']

df = df.drop(['diabetes'], axis =1)

df = (df-df.min())/(df.max()-df.min())

df['diabetes']=class\_label

df



#pre-processing

diabetes\_data = df.copy()

le = preprocessing.LabelEncoder()

age = le.fit\_transform(list(diabetes\_data["age"]))

gender = le.fit\_transform(list(diabetes\_data["gender"]))

polyuria = le.fit\_transform(list(diabetes\_data["polyuria"]))

polydipsia = le.fit\_transform(list(diabetes\_data["polydipsia"]))

sudden\_weight\_loss = le.fit\_transform(list(diabetes\_data["sudden\_weight\_loss"]))

weakness = le.fit\_transform(list(diabetes\_data["weakness"]))

polyphagia = le.fit\_transform(list(diabetes\_data["polyphagia"]))

genital\_thrush = le.fit\_transform(list(diabetes\_data["genital\_thrush"]))

visual\_blurring = le.fit\_transform(list(diabetes\_data["visual\_blurring"]))

itching = le.fit\_transform(list(diabetes\_data["itching"]))

irritability = le.fit\_transform(list(diabetes\_data["irritability"]))

delayed\_healing = le.fit\_transform(list(diabetes\_data["delayed\_healing"]))

partial\_paresis = le.fit\_transform(list(diabetes\_data["partial\_paresis"]))

muscle\_stiffness = le.fit\_transform(list(diabetes\_data["muscle\_stiffness"]))

alopecia = le.fit\_transform(list(diabetes\_data["alopecia"]))

obesity = le.fit\_transform(list(diabetes\_data["obesity"]))

diabetes = le.fit\_transform(list(diabetes\_data["diabetes"]))

### Model Preparation and Development

Steps used for machine learning model preparation are described below:

* + Convert the dataframe to training and validation/test subsets by taking a random sample of 80% of the data and defining it as train subset. This leaves 20% of the data for validation/testing
  + Create the validation/test set by dropping all of the rows that comprise the training set from the dataframe.
  + Create y\_train by using using the last column of train (target class).
  + Create x\_train by using all of the columns in train except the last one.
  + The validation set of y\_val and x\_val or (y\_test and x\_test), can be created using the same methodology that used to create y\_train and x\_train

x = list(zip(age, gender, polyuria, polydipsia, sudden\_weight\_loss, weakness, polyphagia, genital\_thrush, visual\_blurring, itching, irritability, delayed\_healing, partial\_paresis, muscle\_stiffness, alopecia, obesity))

y = list(diabetes)

# Test options and evaluation metric

num\_folds = 5

seed = 7

scoring = 'accuracy'

# Model Test/Train

# Splitting what we are trying to predict into 4 different arrays -

# X train is a section of the x array(attributes) and vise versa for Y(features)

# The test data will test the accuracy of the model created

x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(x, y, test\_size = 0.20, random\_state=seed)

#splitting 20% of our data into test samples. If we train the model with higher data it already has seen that information and knows

#size of train and test subsets after splitting

np.shape(x\_train), np.shape(x\_test)

#size of train and test subsets after splitting

np.shape(x\_train), np.shape(x\_test)

((416, 16), (104, 16))

# Predictive analytics model development by comparing different Scikit-learn classification algorithms

from sklearn.preprocessing import StandardScaler

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

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

from sklearn.metrics import accuracy\_score

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import ExtraTreesClassifier

models = []

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC()))

models.append(('GBM', GradientBoostingClassifier()))

models.append(('RF', RandomForestClassifier()))

# evaluate each model in turn

results = []

names = []

print("Performance on Training set")

for name, model in models:

  kfold = KFold(n\_splits=num\_folds,shuffle=True,random\_state=seed)

  cv\_results = cross\_val\_score(model, x\_train, y\_train, cv=kfold, scoring='accuracy')

  results.append(cv\_results)

  names.append(name)

  msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

  msg += '\n'

  print(msg)

Performance on Training set

NB: 0.889329 (0.033761)

SVM: 0.615462 (0.033923)

GBM: 0.966294 (0.017742)

RF: 0.978342 (0.009047)

# Compare Algorithms' Performance

fig = plt.figure()

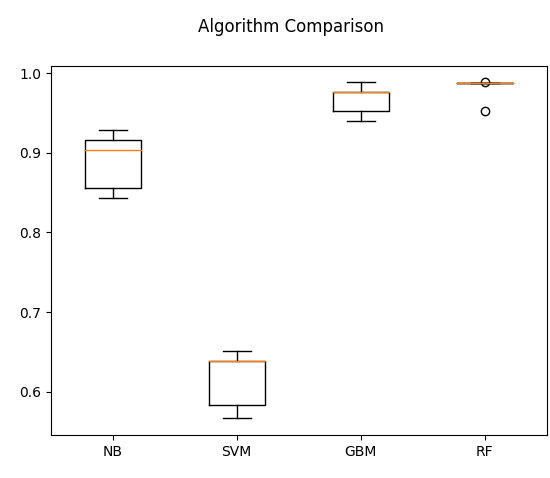
fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()



#Model Evaluation by testing with independent/external test data set.

# Make predictions on validation/test dataset

#Model Evaluation by testing with independent/external test data set.

# Make predictions on validation/test dataset

models.append(('DT', DecisionTreeClassifier()))

models.append(('NB', GaussianNB()))

models.append(('SVM', SVC()))

models.append(('GBM', GradientBoostingClassifier()))

models.append(('RF', RandomForestClassifier()))

dt = DecisionTreeClassifier()

nb = GaussianNB()

gb = GradientBoostingClassifier()

rf = RandomForestClassifier()

best\_model = rf

best\_model.fit(x\_train, y\_train)

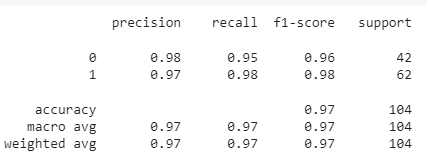
y\_pred = best\_model.predict(x\_test)

print("Best Model Accuracy Score on Test Set:", accuracy\_score(y\_test, y\_pred))

Best Model Accuracy Score on Test Set: 0.9711538461538461

#Model Performance Evaluation Metric 1 - Classification Report

print(classification\_report(y\_test, y\_pred))



#Model Performance Evaluation Metric 2

#Confusion matrix

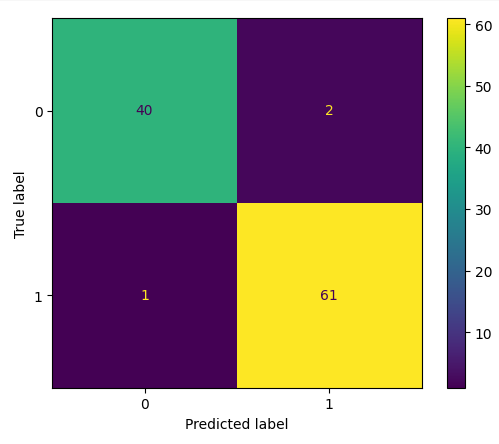
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

plt.show()



#Model Evaluation Metric 3- ROC-AUC curve

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve

best\_model = rf

best\_model.fit(x\_train, y\_train)

rf\_roc\_auc = roc\_auc\_score(y\_test,best\_model.predict(x\_test))

fpr,tpr,thresholds = roc\_curve(y\_test, best\_model.predict\_proba(x\_test)[:,1])

plt.figure()

plt.plot(fpr,tpr,label = 'Random Forest(area = %0.2f)'% rf\_roc\_auc)

plt.plot([0,1],[0,1],'r--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

plt.xlabel('False positive rate')

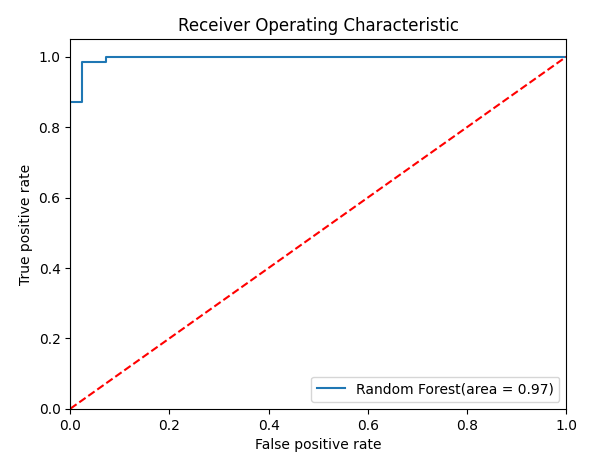
plt.ylabel('True positive rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc='lower right')

plt.savefig('LOC\_ROC')

plt.show()



#Model Evaluation Metric 4-prediction report

for x in range(len(y\_pred)):

  print("Predicted: ", y\_pred[x], "Actual: ", y\_test[x], "Data: ", x\_test[x],)

Predicted: 1 Actual: 1 Data: (26, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (40, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (19, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1)

Predicted: 1 Actual: 1 Data: (26, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (37, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (29, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (12, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0)

Predicted: 0 Actual: 1 Data: (21, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (14, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1)

Predicted: 1 Actual: 1 Data: (4, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (31, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (21, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (15, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (7, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (17, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1)

Predicted: 0 Actual: 0 Data: (6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (15, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1)

Predicted: 1 Actual: 1 Data: (44, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (11, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (21, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (24, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (19, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (47, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (16, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (29, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0)

Predicted: 1 Actual: 0 Data: (25, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1)

Predicted: 0 Actual: 0 Data: (23, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (31, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1)

Predicted: 1 Actual: 1 Data: (24, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (14, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (21, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (37, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1)

Predicted: 0 Actual: 0 Data: (34, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1)

Predicted: 0 Actual: 0 Data: (22, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (13, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (29, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (44, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (23, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (24, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (6, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (24, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (41, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0)

Predicted: 0 Actual: 0 Data: (23, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (12, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (8, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)

Predicted: 0 Actual: 0 Data: (19, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (25, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (31, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1)

Predicted: 0 Actual: 0 Data: (6, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (38, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1)

Predicted: 0 Actual: 0 Data: (47, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (48, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (24, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0)

Predicted: 0 Actual: 0 Data: (47, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0)

Predicted: 0 Actual: 0 Data: (8, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (29, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (14, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (40, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (19, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1)

Predicted: 1 Actual: 1 Data: (14, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (16, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (38, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1)

Predicted: 1 Actual: 1 Data: (15, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (14, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (16, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)

Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (30, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (44, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (23, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1)

Predicted: 0 Actual: 0 Data: (19, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (37, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (33, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0)

Predicted: 0 Actual: 0 Data: (36, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)

Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (39, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1)

Predicted: 1 Actual: 1 Data: (41, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (21, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (37, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (30, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (47, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0)

Predicted: 0 Actual: 0 Data: (18, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (49, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0)

Predicted: 1 Actual: 0 Data: (32, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0)

Predicted: 1 Actual: 1 Data: (33, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0)

Predicted: 0 Actual: 0 Data: (23, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (10, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (27, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (43, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1)

Predicted: 0 Actual: 0 Data: (30, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0)

Predicted: 0 Actual: 0 Data: (30, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0)

Predicted: 1 Actual: 1 Data: (33, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (15, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0)

Predicted: 0 Actual: 0 Data: (21, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (17, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1)

Predicted: 1 Actual: 1 Data: (37, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (11, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)

Predicted: 1 Actual: 1 Data: (36, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0)

Predicted: 1 Actual: 1 Data: (42, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0)

Google colab Link: <https://colab.research.google.com/drive/1NT4MxCedqkOvmz3D7OcI-NpYtU_NUS5G?usp=share_link>

## Stage 2: Algorithm Implementation Stage

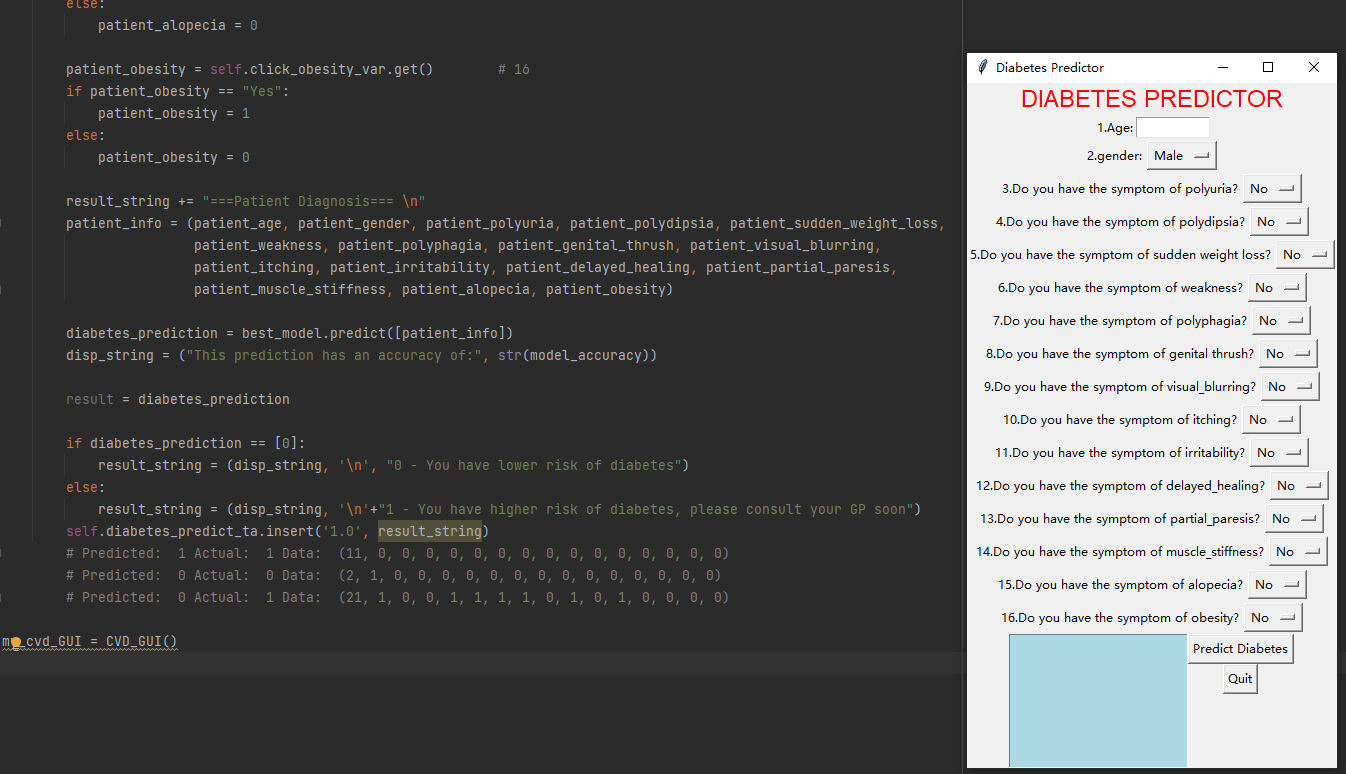
1. Implementation of best performing algorithm as a desktop Tkinter software tool.
2. Deployment of the tool as a web or cloud enabled platform tool.

Once the best performing algorithm and machine learning model for diabetes prediction has been identified from stage 1, the implementation of the algorithm as a desktop software tool using python Tkinter package.

The Pycharm project for the implementation is available at this google drive link:

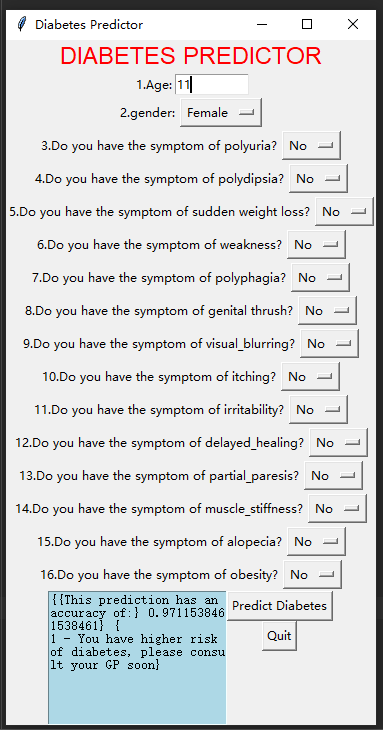
<https://drive.google.com/drive/folders/1JoeVFcuy7KX5Aes3S-vglo3uLiacf0g-?usp=share_link>

Gui:



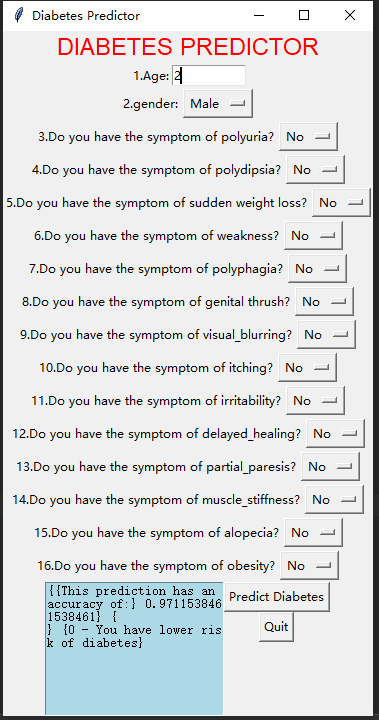
First Condition: True vs True

# Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)



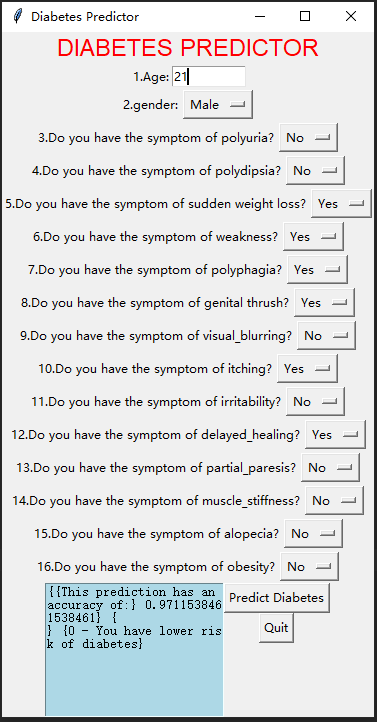
Second Condition: False vs False

# Predicted: 0 Actual: 0 Data: (2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)



Third Condition: False vs True

# Predicted: 0 Actual: 1 Data: (21, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0)



Codes:

import sklearn  
from sklearn.utils import shuffle  
from sklearn import datasets  
import matplotlib.pyplot as pyplot  
import pickle  
from matplotlib import style  
from sklearn import svm  
import pandas as pd  
import numpy as np  
from sklearn import linear\_model, preprocessing  
  
# Load libraries  
import numpy  
from matplotlib import pyplot as plt  
from pandas import read\_csv  
from pandas import set\_option  
from pandas.plotting import scatter\_matrix  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import GridSearchCV  
from sklearn.metrics import classification\_report  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  
from sklearn.metrics import accuracy\_score  
from sklearn.pipeline import Pipeline  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import ExtraTreesClassifier  
  
# Data file import  
diabetes\_data = pd.read\_csv("diabetes.csv")  
  
# Attribute to be predicted  
predict = "diabetes"  
  
# Dataset/Column to be Predicted, X is all attributes and y is the features  
# x = np.array(heart\_data.drop([predict], 1)) # Will return a new data frame that doesn't have hd in it  
# y = np.array(heart\_data[predict])  
le = preprocessing.LabelEncoder()  
age = le.fit\_transform(list(diabetes\_data["age"])) # 1  
gender = le.fit\_transform(list(diabetes\_data["gender"])) # 2  
polyuria = le.fit\_transform(list(diabetes\_data["polyuria"])) # 3  
polydipsia = le.fit\_transform(list(diabetes\_data["polydipsia"])) # 4  
sudden\_weight\_loss = le.fit\_transform(list(diabetes\_data["sudden\_weight\_loss"])) # 5  
weakness = le.fit\_transform(list(diabetes\_data["weakness"])) # 6  
polyphagia = le.fit\_transform(list(diabetes\_data["polyphagia"])) # 7  
genital\_thrush = le.fit\_transform(list(diabetes\_data["genital\_thrush"])) # 8  
visual\_blurring = le.fit\_transform(list(diabetes\_data["visual\_blurring"])) # 9  
itching = le.fit\_transform(list(diabetes\_data["itching"])) # 10  
irritability = le.fit\_transform(list(diabetes\_data["irritability"])) # 11  
delayed\_healing = le.fit\_transform(list(diabetes\_data["delayed\_healing"])) # 12  
partial\_paresis = le.fit\_transform(list(diabetes\_data["partial\_paresis"])) # 13  
muscle\_stiffness = le.fit\_transform(list(diabetes\_data["muscle\_stiffness"])) # 14  
alopecia = le.fit\_transform(list(diabetes\_data["alopecia"])) # 15  
obesity = le.fit\_transform(list(diabetes\_data["obesity"])) # 16  
diabetes = le.fit\_transform(list(diabetes\_data["diabetes"])) # 17  
  
  
x = list(zip(age, gender, polyuria, polydipsia, sudden\_weight\_loss, weakness, polyphagia, genital\_thrush, visual\_blurring, itching, irritability, delayed\_healing, partial\_paresis, muscle\_stiffness, alopecia, obesity))  
y = list(diabetes)  
# Test options and evaluation metric  
num\_folds = 5  
seed = 7  
scoring = 'accuracy'  
  
# Model Test/Train  
# Splitting what we are trying to predict into 4 different arrays -  
# X train is a section of the x array(attributes) and vise versa for Y(features)  
# The test data will test the accuracy of the model created  
x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(x, y, test\_size=0.20, random\_state=seed)  
# splitting 20% of our data into test samples. If we train the model with higher data it already  
# has seen that information and knows.  
  
# Check with different Scikit-learn classification algorithms  
models = []  
models.append(('DT', DecisionTreeClassifier()))  
models.append(('NB', GaussianNB()))  
models.append(('SVM', SVC()))  
models.append(('GBM', GradientBoostingClassifier()))  
models.append(('RF', RandomForestClassifier()))  
# evaluate each model in turn  
results = []  
names = []  
  
for name, model in models:  
 kfold = KFold(n\_splits=num\_folds, shuffle=True, random\_state=seed)  
 cv\_results = cross\_val\_score(model, x\_train, y\_train, cv=kfold, scoring='accuracy')  
 results.append(cv\_results)  
 names.append(name)  
 msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())  
 msg += '\n'  
 print(msg)  
  
# Compare Algorithms' Performance  
fig = pyplot.figure()  
fig.suptitle('Algorithm Comparison')  
ax = fig.add\_subplot(111)  
pyplot.boxplot(results)  
ax.set\_xticklabels(names)  
pyplot.show()  
  
  
# Make predictions on validation/test dataset  
dt = DecisionTreeClassifier()  
nb = GaussianNB()  
gb = GradientBoostingClassifier()  
rf = RandomForestClassifier()  
  
best\_model = rf  
best\_model.fit(x\_train, y\_train)  
y\_pred = best\_model.predict(x\_test)  
model\_accuracy = accuracy\_score(y\_test, y\_pred)  
print("Best Model Accuracy Score on Test Set:", model\_accuracy)  
  
# Model Evaluation Metric 1  
print(classification\_report(y\_test, y\_pred))  
  
# Model Evaluation Metric 2  
# Confusion matrix  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  
cm = confusion\_matrix(y\_test, y\_pred)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
disp.plot()  
plt.show()  
  
# Model Evaluation Metric 3  
from sklearn.metrics import roc\_auc\_score  
from sklearn.metrics import roc\_curve  
  
best\_model = rf  
best\_model.fit(x\_train, y\_train)  
rf\_roc\_auc = roc\_auc\_score(y\_test, best\_model.predict(x\_test))  
fpr, tpr, thresholds = roc\_curve(y\_test, best\_model.predict\_proba(x\_test)[:, 1])  
  
plt.figure()  
plt.plot(fpr, tpr, label='Random Forest(area = %0.2f)' % rf\_roc\_auc)  
plt.plot([0, 1], [0, 1], 'r--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False positive rate')  
plt.ylabel('True positive rate')  
plt.title('Receiver Operating Characteristic')  
plt.legend(loc='lower right')  
plt.savefig('LOC\_ROC')  
plt.show()  
  
# Check actual/ground truth vs predicted diagnosis  
for x in range(len(y\_pred)):  
 print("Predicted: ", y\_pred[x], "Actual: ", y\_test[x], "Data: ", x\_test[x],)

GUI:

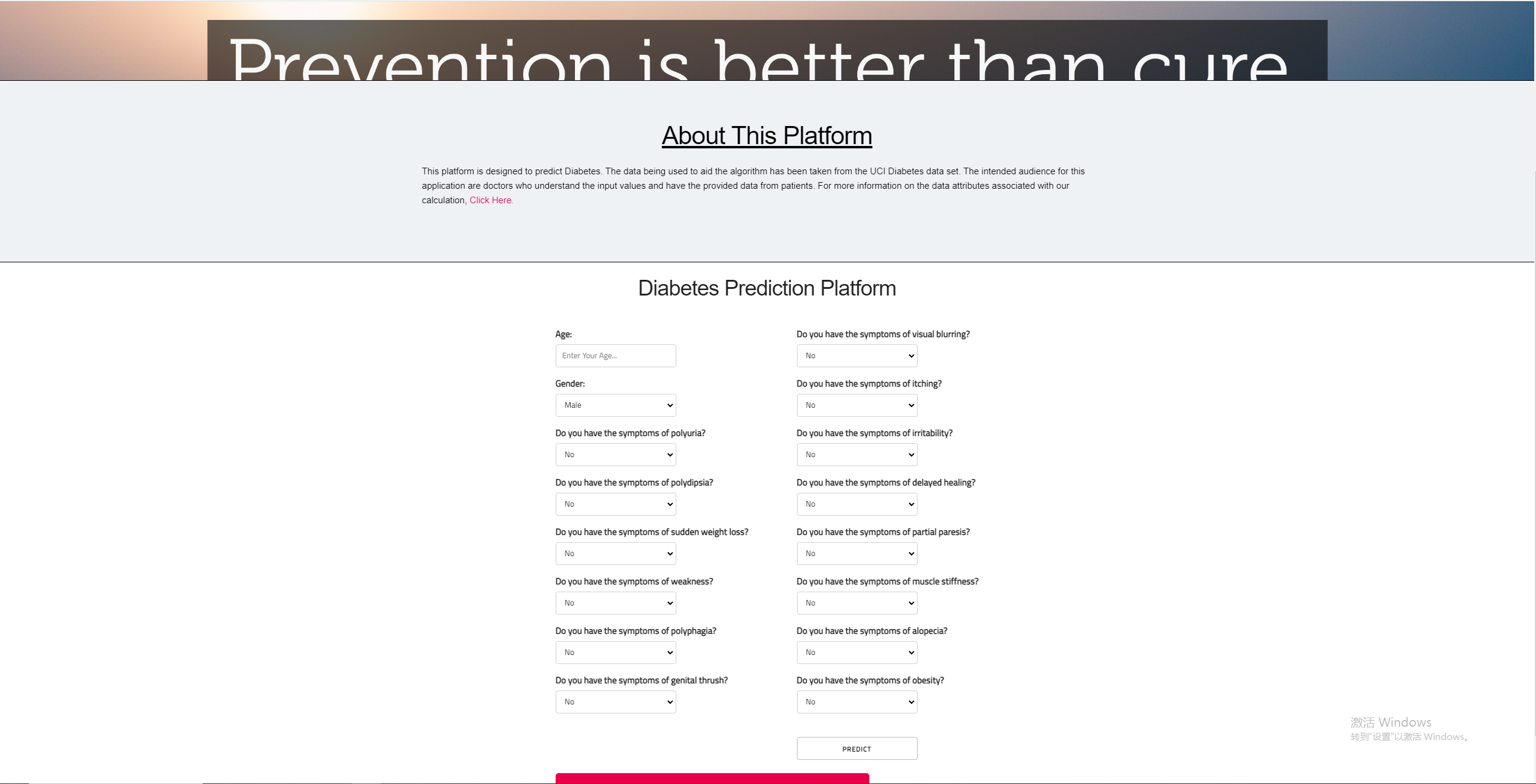
import tkinter as tk  
from cvd\_model import \*  
  
  
class CVD\_GUI:  
 def \_\_init\_\_(self):  
  
 # Create the main window.  
 self.main\_window = tk.Tk()  
 self.main\_window.title("Diabetes"  
 " Predictor")  
  
 # Create two frames to group widgets.  
 self.zero\_frame = tk.Frame()  
 self.one\_frame = tk.Frame()  
 self.two\_frame = tk.Frame()  
 self.three\_frame = tk.Frame()  
 self.four\_frame = tk.Frame()  
 self.five\_frame = tk.Frame()  
 self.six\_frame = tk.Frame()  
 self.seven\_frame = tk.Frame()  
 self.eight\_frame = tk.Frame()  
 self.nine\_frame = tk.Frame()  
 self.ten\_frame = tk.Frame()  
 self.eleven\_frame = tk.Frame()  
 self.twelve\_frame = tk.Frame()  
 self.thirteen\_frame = tk.Frame()  
 self.fourteen\_frame = tk.Frame()  
 self.fifteen\_frame = tk.Frame()  
 self.sixteen\_frame = tk.Frame()  
 self.seventeen\_frame = tk.Frame()  
  
 # Create the widgets for zero frame. (0 title display)  
 self.title\_label = tk.Label(self.zero\_frame, text='DIABETES PREDICTOR', fg="Red", font=("Helvetica", 18))  
 self.title\_label.pack()  
  
 # Create the widgets for one frame. (1 age input)  
 self.age\_label = tk.Label(self.one\_frame, text='1.Age:')  
 self.age\_entry = tk.Entry(self.one\_frame, bg="white", fg="black", width=10)  
 # self.age\_entry.insert(0,'50')  
 self.age\_label.pack(side='left')  
 self.age\_entry.pack(side='left')  
  
 # Create the widgets for two frame. (2 gender input)  
 self.gender\_label = tk.Label(self.two\_frame, text='2.gender:')  
 self.click\_gender\_var = tk.StringVar()  
 self.click\_gender\_var.set("Male")  
 self.gender\_inp = tk.OptionMenu(self.two\_frame, self.click\_gender\_var, "Male", "Female")  
 self.gender\_label.pack(side='left')  
 self.gender\_inp.pack(side='left')  
  
 # Create the widgets for three frame. (3 polyuria symptom input)  
 self.polyuria\_label = tk.Label(self.three\_frame, text='3.Do you have the symptom of polyuria?')  
 self.click\_polyuria\_var = tk.StringVar()  
 self.click\_polyuria\_var.set("No")  
 self.polyuria\_inp = tk.OptionMenu(self.three\_frame, self.click\_polyuria\_var, "No", "Yes")  
 self.polyuria\_label.pack(side='left')  
 self.polyuria\_inp.pack(side='left')  
  
 # Create the widgets for four frame. (4 polydipsia symptom input)  
 self.polydipsia\_label = tk.Label(self.four\_frame, text='4.Do you have the symptom of polydipsia?')  
 self.click\_polydipsia\_var = tk.StringVar()  
 self.click\_polydipsia\_var.set("No")  
 self.polydipsia\_inp = tk.OptionMenu(self.four\_frame, self.click\_polydipsia\_var, "No", "Yes")  
 self.polydipsia\_label.pack(side='left')  
 self.polydipsia\_inp.pack(side='left')  
  
 # Create the widgets for five frame. (5 sudden\_weight\_loss symptom input)  
 self.sudden\_weight\_loss\_label = tk.Label(self.five\_frame, text='5.Do you have the symptom of '  
 'sudden weight loss?')  
 self.click\_sudden\_weight\_loss\_var = tk.StringVar()  
 self.click\_sudden\_weight\_loss\_var.set("No")  
 self.sudden\_weight\_loss\_inp = tk.OptionMenu(self.five\_frame, self.click\_sudden\_weight\_loss\_var, "No", "Yes")  
 self.sudden\_weight\_loss\_label.pack(side='left')  
 self.sudden\_weight\_loss\_inp.pack(side='left')  
  
 # Create the widgets for six frame. (6 weakness symptom input)  
 self.weakness\_label = tk.Label(self.six\_frame, text='6.Do you have the symptom of weakness?')  
 self.click\_weakness\_var = tk.StringVar()  
 self.click\_weakness\_var.set("No")  
 self.weakness\_inp = tk.OptionMenu(self.six\_frame, self.click\_weakness\_var, "No", "Yes")  
 self.weakness\_label.pack(side='left')  
 self.weakness\_inp.pack(side='left')  
  
 # Create the widgets for seven frame. (7 polyphagia symptom input)  
 self.polyphagia\_label = tk.Label(self.seven\_frame, text='7.Do you have the symptom of polyphagia?')  
 self.click\_polyphagia\_var = tk.StringVar()  
 self.click\_polyphagia\_var.set("No")  
 self.polyphagia\_inp = tk.OptionMenu(self.seven\_frame, self.click\_polyphagia\_var, "No", "Yes")  
 self.polyphagia\_label.pack(side='left')  
 self.polyphagia\_inp.pack(side='left')  
  
 # Create the widgets for eight frame. (8 genital\_thrush symptom input)  
 self.genital\_thrush\_label = tk.Label(self.eight\_frame, text='8.Do you have the symptom of genital thrush?')  
 self.click\_genital\_thrush\_var = tk.StringVar()  
 self.click\_genital\_thrush\_var.set("No")  
 self.genital\_thrush\_inp = tk.OptionMenu(self.eight\_frame, self.click\_genital\_thrush\_var, "No", "Yes")  
 self.genital\_thrush\_label.pack(side='left')  
 self.genital\_thrush\_inp.pack(side='left')  
  
 # Create the widgets for nine frame. (9 symptom visual\_blurring input)  
 self.visual\_blurring\_label = tk.Label(self.nine\_frame, text='9.Do you have the symptom of visual\_blurring?')  
 self.click\_visual\_blurring\_var = tk.StringVar()  
 self.click\_visual\_blurring\_var.set("No")  
 self.visual\_blurring\_inp = tk.OptionMenu(self.nine\_frame, self.click\_visual\_blurring\_var, "No", "Yes")  
 self.visual\_blurring\_label.pack(side='left')  
 self.visual\_blurring\_inp.pack(side='left')  
  
 # Create the widgets for ten frame. (10 itching symptom input)  
 self.itching\_label = tk.Label(self.ten\_frame, text='10.Do you have the symptom of itching?')  
 self.click\_itching\_var = tk.StringVar()  
 self.click\_itching\_var.set("No")  
 self.itching\_inp = tk.OptionMenu(self.ten\_frame, self.click\_itching\_var, "No", "Yes")  
 self.itching\_label.pack(side='left')  
 self.itching\_inp.pack(side='left')  
  
 # Create the widgets for eleven frame. (11 irritability symptom input)  
 self.irritability\_label = tk.Label(self.eleven\_frame, text='11.Do you have the symptom of irritability?')  
 self.click\_irritability\_var = tk.StringVar()  
 self.click\_irritability\_var.set("No")  
 self.irritability\_inp = tk.OptionMenu(self.eleven\_frame, self.click\_irritability\_var, "No", "Yes")  
 self.irritability\_label.pack(side='left')  
 self.irritability\_inp.pack(side='left')  
  
 # Create the widgets for twelve frame. (12 delayed\_healing symptom input)  
 self.delayed\_healing\_label = tk.Label(self.twelve\_frame, text='12.Do you have the symptom of delayed\_healing?')  
 self.click\_delayed\_healing\_var = tk.StringVar()  
 self.click\_delayed\_healing\_var.set("No")  
 self.delayed\_healing\_inp = tk.OptionMenu(self.twelve\_frame, self.click\_delayed\_healing\_var, "No", "Yes")  
 self.delayed\_healing\_label.pack(side='left')  
 self.delayed\_healing\_inp.pack(side='left')  
  
 # Create the widgets for thirteen frame. (13 partial\_paresis symptom input)  
 self.partial\_paresis\_label = tk.Label(self.thirteen\_frame, text='13.Do you have the symptom of partial\_paresis?')  
 self.click\_partial\_paresis\_var = tk.StringVar()  
 self.click\_partial\_paresis\_var.set("No")  
 self.partial\_paresis\_inp = tk.OptionMenu(self.thirteen\_frame, self.click\_partial\_paresis\_var, "No", "Yes")  
 self.partial\_paresis\_label.pack(side='left')  
 self.partial\_paresis\_inp.pack(side='left')  
  
 # Create the widgets for fourteen frame. (14 symptom input)  
 self.muscle\_stiffness\_label = tk.Label(self.fourteen\_frame, text='14.Do you have the symptom of '  
 'muscle\_stiffness?')  
 self.click\_muscle\_stiffness\_var = tk.StringVar()  
 self.click\_muscle\_stiffness\_var.set("No")  
 self.muscle\_stiffness\_inp = tk.OptionMenu(self.fourteen\_frame, self.click\_muscle\_stiffness\_var, "No", "Yes")  
 self.muscle\_stiffness\_label.pack(side='left')  
 self.muscle\_stiffness\_inp.pack(side='left')  
  
 # Create the widgets for fifteen frame. (15 symptom input)  
 self.alopecia\_label = tk.Label(self.fifteen\_frame, text='15.Do you have the symptom of alopecia?')  
 self.click\_alopecia\_var = tk.StringVar()  
 self.click\_alopecia\_var.set("No")  
 self.alopecia\_inp = tk.OptionMenu(self.fifteen\_frame, self.click\_alopecia\_var, "No", "Yes")  
 self.alopecia\_label.pack(side='left')  
 self.alopecia\_inp.pack(side='left')  
  
 # Create the widgets for sixteen frame. (16 symptom input)  
 self.obesity\_label = tk.Label(self.sixteen\_frame, text='16.Do you have the symptom of obesity?')  
 self.click\_obesity\_var = tk.StringVar()  
 self.click\_obesity\_var.set("No")  
 self.obesity\_inp = tk.OptionMenu(self.sixteen\_frame, self.click\_obesity\_var, "No", "Yes")  
 self.obesity\_label.pack(side='left')  
 self.obesity\_inp.pack(side='left')  
  
 # Create the widgets for fifteen frame = diabetes (17 prediction of heart disease)  
 self.diabetes\_predict\_ta = tk.Text(self.seventeen\_frame, height=10, width=25, bg='light blue')  
  
 # Create predict button and quit button  
 self.btn\_predict = tk.Button(self.seventeen\_frame, text='Predict Diabetes', command=self.predict\_diabetes)  
 self.btn\_quit = tk.Button(self.seventeen\_frame, text='Quit', command=self.main\_window.destroy)  
  
 self.diabetes\_predict\_ta.pack(side='left')  
 self.btn\_predict.pack()  
 self.btn\_quit.pack()  
  
 # Pack the frames.  
 self.zero\_frame.pack()  
 self.one\_frame.pack()  
 self.two\_frame.pack()  
 self.three\_frame.pack()  
 self.four\_frame.pack()  
 self.five\_frame.pack()  
 self.six\_frame.pack()  
 self.seven\_frame.pack()  
 self.eight\_frame.pack()  
 self.nine\_frame.pack()  
 self.ten\_frame.pack()  
 self.eleven\_frame.pack()  
 self.twelve\_frame.pack()  
 self.thirteen\_frame.pack()  
 self.fourteen\_frame.pack()  
 self.fifteen\_frame.pack()  
 self.sixteen\_frame.pack()  
 self.seventeen\_frame.pack()  
  
 # Enter the tkinter main loop.  
 tk.mainloop()  
  
 def predict\_diabetes(self):  
 result\_string = ""  
  
 self.diabetes\_predict\_ta.delete(0.0, tk.END)  
 patient\_age = self.age\_entry.get() # 1  
  
 patient\_gender = self.click\_gender\_var.get() # 2  
 if patient\_gender == "Male":  
 patient\_gender = 1  
 else:  
 patient\_gender = 0  
  
 patient\_polyuria = self.click\_polyuria\_var.get() # 3  
 if patient\_polyuria == "Yes":  
 patient\_polyuria = 1  
 else:  
 patient\_polyuria = 0  
  
 patient\_polydipsia = self.click\_polydipsia\_var.get() # 4  
 if patient\_polydipsia == "Yes":  
 patient\_polydipsia = 1  
 else:  
 patient\_polydipsia = 0  
  
 patient\_sudden\_weight\_loss = self.click\_sudden\_weight\_loss\_var.get() # 5  
 if patient\_sudden\_weight\_loss == "Yes":  
 patient\_sudden\_weight\_loss = 1  
 else:  
 patient\_sudden\_weight\_loss = 0  
  
 patient\_weakness = self.click\_weakness\_var.get() # 6  
 if patient\_weakness == "Yes":  
 patient\_weakness = 1  
 else:  
 patient\_weakness = 0  
  
 patient\_polyphagia = self.click\_polyphagia\_var.get() # 7  
 if patient\_polyphagia == "Yes":  
 patient\_polyphagia = 1  
 else:  
 patient\_polyphagia = 0  
  
 patient\_genital\_thrush = self.click\_genital\_thrush\_var.get() # 8  
 if patient\_genital\_thrush == "Yes":  
 patient\_genital\_thrush = 1  
 else:  
 patient\_genital\_thrush = 0  
  
 patient\_visual\_blurring = self.click\_visual\_blurring\_var.get() # 9  
 if patient\_visual\_blurring == "Yes":  
 patient\_visual\_blurring = 1  
 else:  
 patient\_visual\_blurring = 0  
  
 patient\_itching = self.click\_itching\_var.get() # 10  
 if patient\_itching == "Yes":  
 patient\_itching = 1  
 else:  
 patient\_itching = 0  
  
 patient\_irritability = self.click\_irritability\_var.get() # 11  
 if patient\_irritability == "Yes":  
 patient\_irritability = 1  
 else:  
 patient\_irritability = 0  
  
 patient\_delayed\_healing = self.click\_delayed\_healing\_var.get() # 12  
 if patient\_delayed\_healing == "Yes":  
 patient\_delayed\_healing = 1  
 else:  
 patient\_delayed\_healing = 0  
  
 patient\_partial\_paresis = self.click\_partial\_paresis\_var.get() # 13  
 if patient\_partial\_paresis == "Yes":  
 patient\_partial\_paresis = 1  
 else:  
 patient\_partial\_paresis = 0  
  
 patient\_muscle\_stiffness = self.click\_muscle\_stiffness\_var.get() # 14  
 if patient\_muscle\_stiffness == "Yes":  
 patient\_muscle\_stiffness = 1  
 else:  
 patient\_muscle\_stiffness = 0  
  
 patient\_alopecia = self.click\_alopecia\_var.get() # 15  
 if patient\_alopecia == "Yes":  
 patient\_alopecia = 1  
 else:  
 patient\_alopecia = 0  
  
 patient\_obesity = self.click\_obesity\_var.get() # 16  
 if patient\_obesity == "Yes":  
 patient\_obesity = 1  
 else:  
 patient\_obesity = 0  
  
 result\_string += "===Patient Diagnosis=== \n"  
 patient\_info = (patient\_age, patient\_gender, patient\_polyuria, patient\_polydipsia, patient\_sudden\_weight\_loss,  
 patient\_weakness, patient\_polyphagia, patient\_genital\_thrush, patient\_visual\_blurring,  
 patient\_itching, patient\_irritability, patient\_delayed\_healing, patient\_partial\_paresis,  
 patient\_muscle\_stiffness, patient\_alopecia, patient\_obesity)  
  
 diabetes\_prediction = best\_model.predict([patient\_info])  
 disp\_string = ("This prediction has an accuracy of:", str(model\_accuracy))  
  
 result = diabetes\_prediction  
  
 if diabetes\_prediction == [0]:  
 result\_string = (disp\_string, '\n', "0 - You have lower risk of diabetes")  
 else:  
 result\_string = (disp\_string, '\n'+"1 - You have higher risk of diabetes, please consult your GP soon")  
 self.diabetes\_predict\_ta.insert('1.0', result\_string)  
 # Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)  
 # Predicted: 0 Actual: 0 Data: (2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)  
 # Predicted: 0 Actual: 1 Data: (21, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0)  
  
my\_cvd\_GUI = CVD\_GUI()

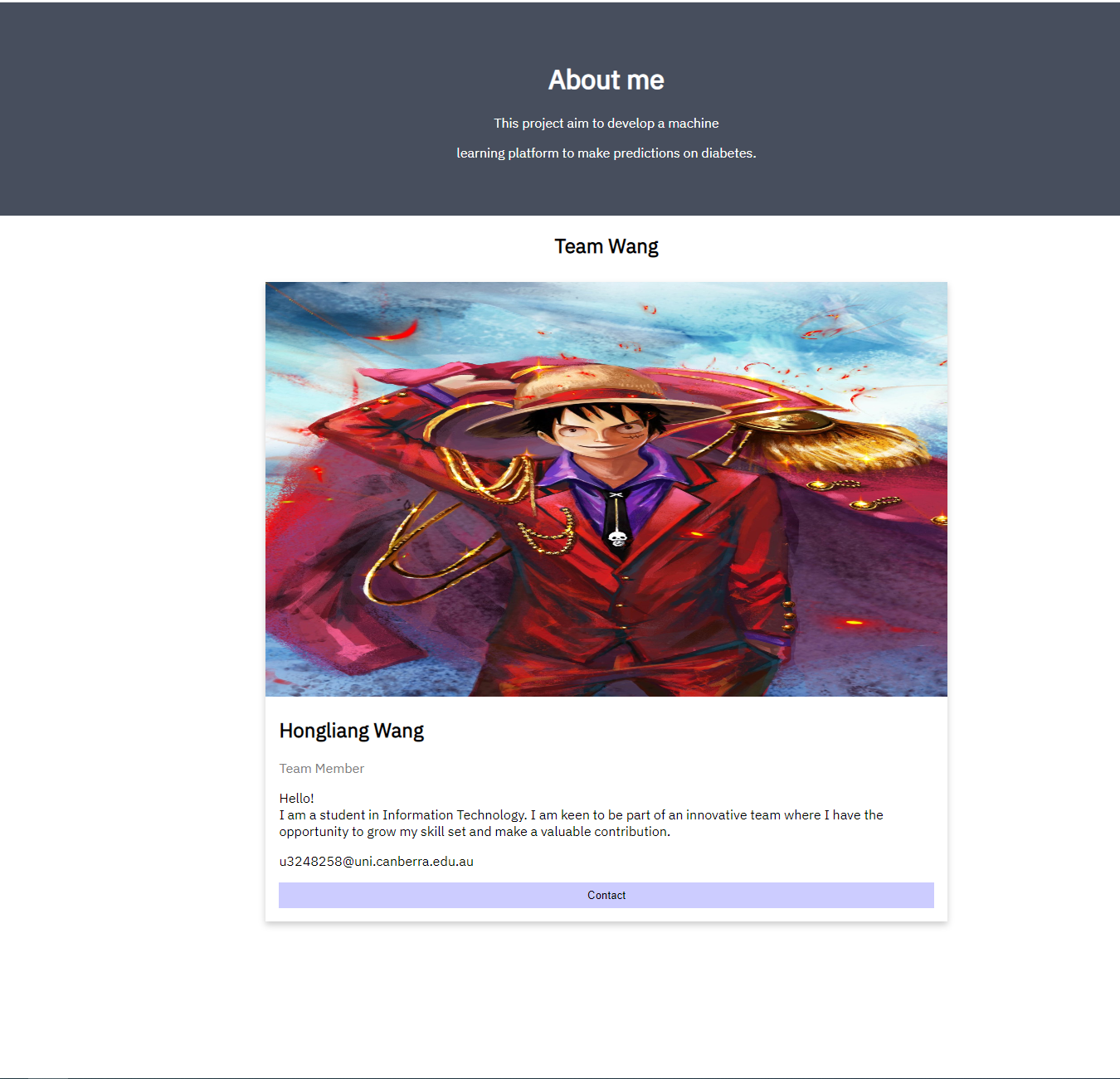
## Stage 3: Software Deployment Stage

The deployment of software as a desktop tool as in stage 2, limits its applicability and does not allow wider usage by all care team involved in the managing the chronic disease for the patient. Hence there is a need to deploy this software as a web based tool or cloud based tool. The deployment of diabetes prediction as a web based platform was done using Flask API, a widely used micro web framework for creating APIs in Python. Flask is a simple yet powerful web framework in Python, with an the ability to scale up to complex applications. The Flask project deployment for the diabetes prediction is available at this google drive link:

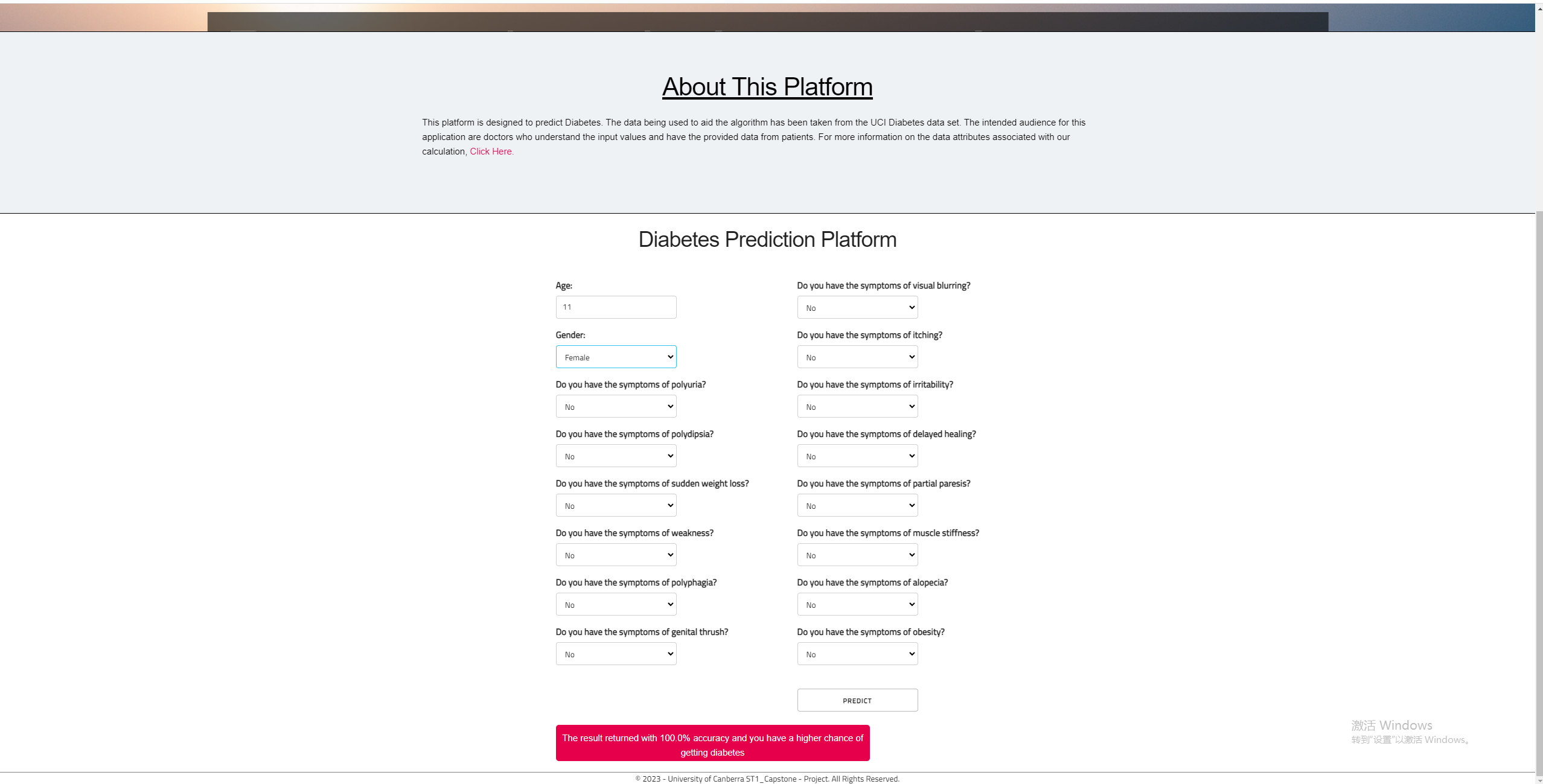
<https://drive.google.com/drive/folders/1JoeVFcuy7KX5Aes3S-vglo3uLiacf0g-?usp=share_link>

web app:

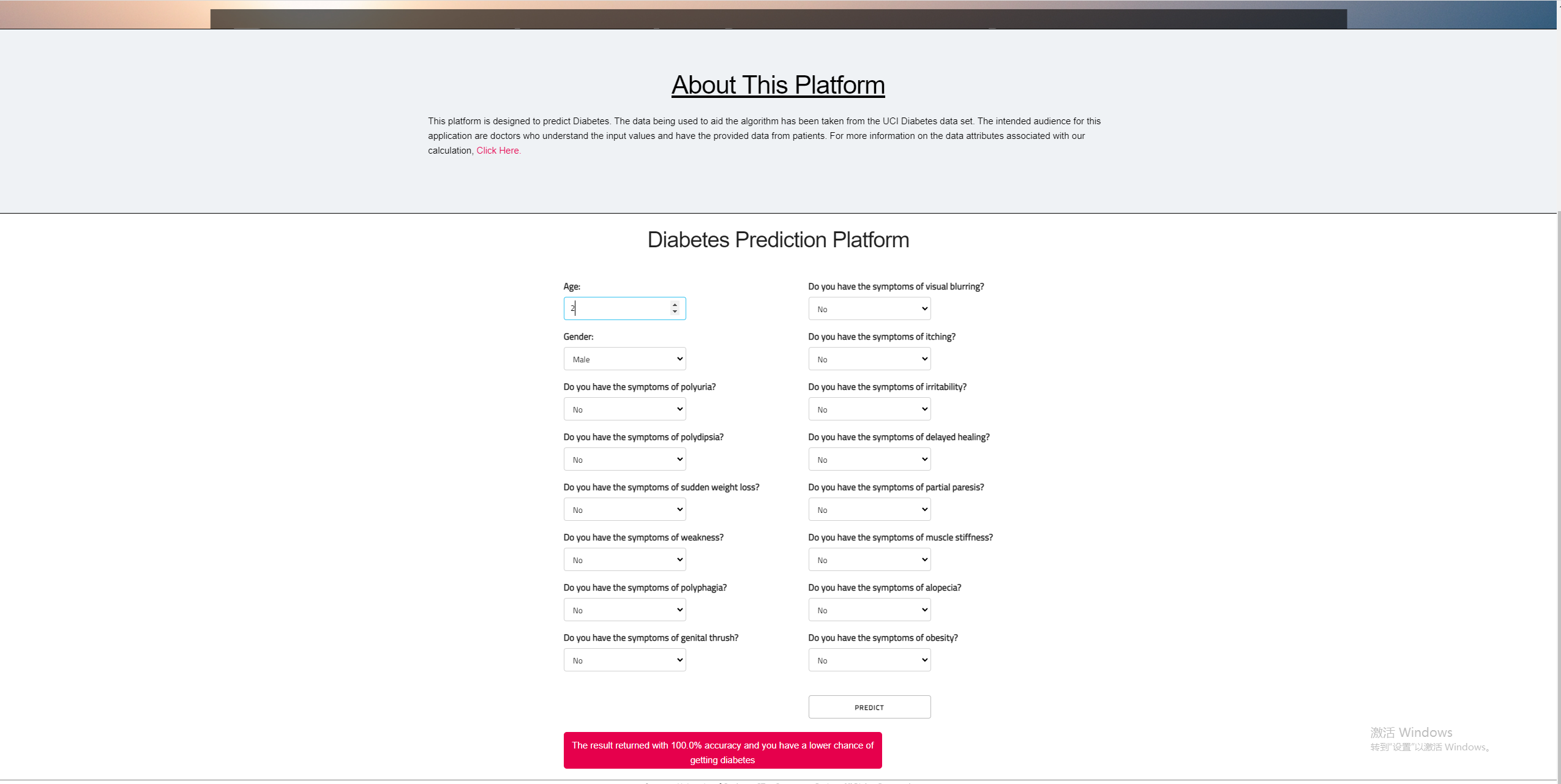




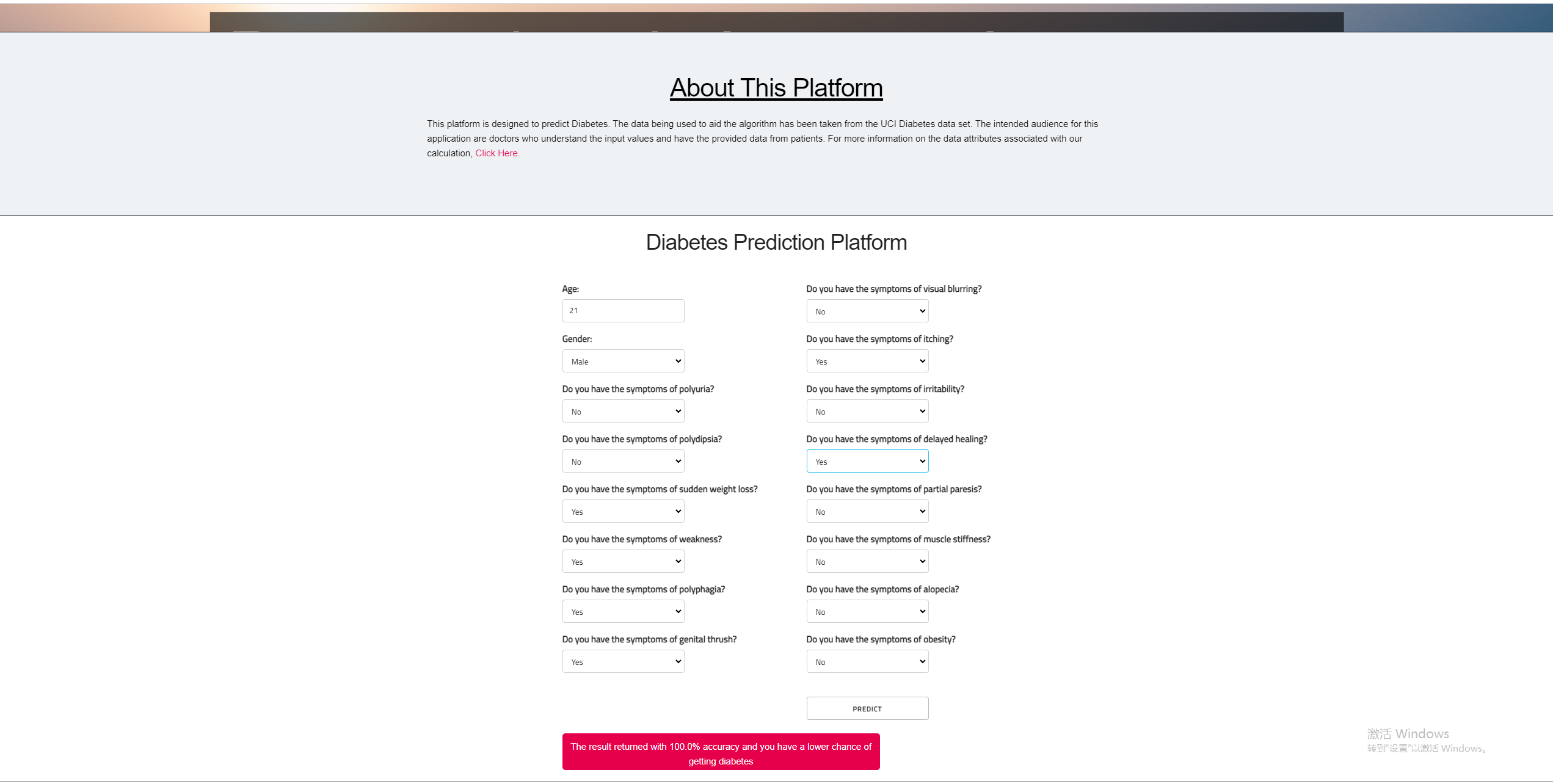
# Predicted: 1 Actual: 1 Data: (11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)



# Predicted: 0 Actual: 0 Data: (2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)



# Predicted: 0 Actual: 1 Data: (21, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0)



Same results as GUI

Codes:

from flask import Flask, Blueprint, render\_template, redirect, url\_for, request  
from werkzeug.middleware.proxy\_fix import ProxyFix  
import sys  
from argparse import ArgumentParser  
from cvd\_model import \*  
  
appweb = Blueprint('hello', \_\_name\_\_)  
  
  
@appweb.route('/')  
def home():  
 return render\_template("index.html")  
  
  
@appweb.route('/send', methods=['POST'])  
def send(predict=predict):  
 if request.method == 'POST':  
 patient\_age = request.form['age'] # 1  
 patient\_gender = request.form['gender'] # 2  
 patient\_polyuria = request.form['polyuria'] # 3  
 patient\_polydipsia = request.form['polydipsia'] # 4  
 patient\_sudden\_weight\_loss = request.form['suddenweightloss'] # 5  
 patient\_weakness = request.form['weakness'] # 6  
 patient\_polyphagia = request.form['polyphagia'] # 7  
 patient\_genital\_thrush = request.form['genitalthrush'] # 8  
 patient\_visual\_blurring = request.form['visualblurring'] # 9  
 patient\_itching = request.form['itching'] # 10  
 patient\_irritability = request.form['irritability'] # 11  
 patient\_delayed\_healing = request.form['delayedhealing'] # 12  
 patient\_partial\_paresis = request.form['partialparesis'] # 13  
 patient\_muscle\_stiffness = request.form['musclestiffness'] # 14  
 patient\_alopecia = request.form['alopecia'] # 15  
 patient\_obesity = request.form['obesity'] # 16  
  
 if patient\_gender == "male":  
 patient\_gender = 1  
 else:  
 patient\_gender = 0  
  
 if patient\_polyuria == "Yes":  
 patient\_polyuria = 1  
 else:  
 patient\_polyuria = 0  
  
 if patient\_polydipsia == "Yes":  
 patient\_polydipsia = 1  
 else:  
 patient\_polydipsia = 0  
  
 if patient\_sudden\_weight\_loss == "Yes":  
 patient\_sudden\_weight\_loss = 1  
 else:  
 patient\_sudden\_weight\_loss = 0  
  
 if patient\_weakness == "Yes":  
 patient\_weakness = 1  
 else:  
 patient\_weakness = 0  
  
 if patient\_polyphagia == "Yes":  
 patient\_polyphagia = 1  
 else:  
 patient\_polyphagia = 0  
  
 if patient\_genital\_thrush == "Yes":  
 patient\_genital\_thrush = 1  
 else:  
 patient\_genital\_thrush = 0  
  
 if patient\_visual\_blurring == "Yes":  
 patient\_visual\_blurring = 1  
 else:  
 patient\_visual\_blurring = 0  
  
 if patient\_itching == "Yes":  
 patient\_itching = 1  
 else:  
 patient\_itching = 0  
  
 if patient\_irritability == "Yes":  
 patient\_irritability = 1  
 else:  
 patient\_irritability = 0  
  
 if patient\_delayed\_healing == "Yes":  
 patient\_delayed\_healing = 1  
 else:  
 patient\_delayed\_healing = 0  
  
 if patient\_partial\_paresis == "Yes":  
 patient\_partial\_paresis = 1  
 else:  
 patient\_partial\_paresis = 0  
  
 if patient\_muscle\_stiffness == "Yes":  
 patient\_muscle\_stiffness = 1  
 else:  
 patient\_muscle\_stiffness = 0  
  
 if patient\_alopecia == "Yes":  
 patient\_alopecia = 1  
 else:  
 patient\_alopecia = 0  
  
 if patient\_obesity == "Yes":  
 patient\_obesity = 1  
 else:  
 patient\_obesity = 0  
  
 # Accuracy of Model  
 model.fit(x\_train, y\_train) # <-- this line  
 acc = model.score(x\_train, y\_train)  
  
 predict\_real = model.predict([[patient\_age, patient\_gender, patient\_polyuria, patient\_polydipsia,  
 patient\_sudden\_weight\_loss, patient\_weakness, patient\_polyphagia,  
 patient\_genital\_thrush, patient\_visual\_blurring, patient\_itching,  
 patient\_irritability, patient\_delayed\_healing, patient\_partial\_paresis,  
 patient\_muscle\_stiffness, patient\_alopecia, patient\_obesity]])  
  
 if predict\_real == [0]:  
 predict = "The result returned with " + str(round(acc, 2)\*100) + "% accuracy and you have a lower " \  
 "chance of getting diabetes"  
 else:  
 predict = "The result returned with " + str(round(acc, 2)\*100) + "% accuracy and you have a higher " \  
 "chance of getting diabetes"  
  
 return render\_template('index.html', predict=predict)  
  
 else:  
 return render\_template('index.html', predict=predict)  
  
  
@appweb.route('/about')  
def about():  
 return render\_template("about.html")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 # arg parser for the standard anaconda-project options  
 parser = ArgumentParser(prog="home",  
 description="Simple Flask Application")  
 parser.add\_argument('--anaconda-project-host', action='append', default=[],  
 help='Hostname to allow in requests')  
 parser.add\_argument('--anaconda-project-port', action='store', default=8086, type=int,  
 help='Port to listen on')  
 parser.add\_argument('--anaconda-project-iframe-hosts',  
 action='append',  
 help='Space-separated hosts which can embed us in an iframe per our Content-Security-Policy')  
 parser.add\_argument('--anaconda-project-no-browser', action='store\_true',  
 default=False,  
 help='Disable opening in a browser')  
 parser.add\_argument('--anaconda-project-use-xheaders',  
 action='store\_true',  
 default=False,  
 help='Trust X-headers from reverse proxy')  
 parser.add\_argument('--anaconda-project-url-prefix', action='store', default='',  
 help='Prefix in front of urls')  
 parser.add\_argument('--anaconda-project-address',  
 action='store',  
 # default='0.0.0.0',  
 help='IP address the application should listen on.')  
  
 args = parser.parse\_args()  
  
 app = Flask(\_\_name\_\_)  
 app.register\_blueprint(appweb, url\_prefix=args.anaconda\_project\_url\_prefix)  
  
 app.config['PREFERRED\_URL\_SCHEME'] = 'https'  
  
 app.wsgi\_app = ProxyFix(app.wsgi\_app)  
 app.run(host=args.anaconda\_project\_address, port=args.anaconda\_project\_port)

# Conclusions

This report presents the work done towards the ST1 capstone project for design, development, implementation and deployment of data driven heart disease prediction software platform using Python. This platform allows leveraging the studies of non-invasive clinical tests run on patients, and building a data driven disease prediction platform, where the historical data and insight about a problem or an event can be used to predict the future risk or probability of the event. As can be seen from the outcomes of this project, we can train models that can predict diabetes with substantial agreement with the results of invasive coronary angiography. The availability of predictive analytics tools as both desktop software tool and a web based tool, allows the wider application of this project to clinical, non-clinical and allied health care team for managing the diabetes for the patients.

## References

1. <https://uclearn.canberra.edu.au/courses/13733/assignments/107529>
2. D Dutta, I. (2021). Early Stage Diabetes Risk Prediction Dataset. Kaggle. Retrieved from <https://www.kaggle.com/datasets/ishandutta/early-stage-diabetes-risk-prediction-dataset>
3. UCI Machine Learning Repository. (n.d.). Early stage diabetes risk prediction dataset. Retrieved [insert date of access], from <https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset>
4. World Health Organization. (2021, January 27). Diabetes. <https://www.who.int/news-room/factsheets/detail/diabetes#:~:text=Diabetes%20is%20a%20chronic%20disease,hormone%20that%20regulates%20blood%20glucose>.