Diabetics Prediction System based on Life Style

Prepared by

Kakkireni Naga Pavani

Assistant Professor

Department of Electronics and Communication Engineering.

Gokaraju Rangaraju Institute of Engineering & Technology.

Hydeerabad.

Index

1. INTRODUCTION

1.1. Overview

Diabetes is one of the terrible diseases in the world which is not only a disease but a root cause for varieties of diseases like heart strokes, blindness, kidney related diseases, etc. The normal procedure is regularly visiting hospitals or labs for testing whether it is normal or not and eagerly waiting for the reports with tension because the report brings lot of changes in our life style .This process is not only mental stress but also waste of money for paying for diagnostic centers. The solution is modern approach which uses auto AI and machine learning models for prediction. The bigger advantage is early prediction saves the risc from being critical. The main aim of the project is to develop an auto AI IBM model for prediction of diabetes Mellitus accurately.

Artificial Intelligence is a wide domain which contains wide spectrum of methods for modelling variety of solutions to problems. It has applications in healthcare domain, very effectively used in prediction of diseases. Various life style and other parameters have an imp progression of this disease. These parameters and the outcome for number of patients already available in the domain can be used to m from and predict with good amount of accuracy when new/unseen data points are posed to it. This would open new avenues for Mach of healthcare that would assist the healthcare stakeholders to make informed decisions by studying large number of patient data. This them to strategize customized treatment methods to better meet the varying disease symptoms in case of chronic diseases like diabetic.

Diabetes is one of the most common chronic diseases. The number of cases of diabetes in the world is likely to increase m the next 30 years; from 115 million in 2000 to 284 million in 2030. In type I diabetes, the disease is caused by the failure of the pancr sufficient amount of insulin which leads to an uncontrolled increase in blood glucose unless the patient administers insulin, typically b injection. This work is concerned with managing diabetic patients by trying to predict their glucose levels in the near future (30 minut current levels. The goal of this paper is to determine the future blood glucose value of a diabetic patient using an artificial neural network approaches, which involve questioning the patients, feature extraction procedures were implemented (diabetic dynamic model) on dia time series, in order to extract a knowledge (how patient blood glucose level will change a according to the external food intake and a activities). An artificial neural network was then trained using these features in order to predict the future value of blood glucose level accuracy.

1.2. Purpose

The purpose of building this application is to predict whether or not a patient acquires diabetes based on some diagnostic features. Though this application would not substitute a doctor or professional medical advice, it serves as a preliminary investigator the symptoms a person has w.r.t. diabetes.

2. ITERATURE SURVEY

2.1. Existing problem

Diabetes disease prediction is one of the progressive research area in the healthcare fields.Although many data mining techniques have been employed to assess the main causes of diabetes, but only few sets of clinical risk factors are considered. Due to this, some important factors like pre-diabetes health conditions are not considered in their analysis. So the results produced by such techniques may not represent appropriate diabetes pattern and risk factors appropriately.

2.2. Proposed solution

In this, we need 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.

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

Skills Required:

- Flask Integration
- IBM Nodered
- IBM Watson Studio
- IBM Machine Learning
- IBM Cloud Object Storage

Software Requirements:

1. Operating system: Windows 10 9/8/19
2. Coding Language: HTML, PYTHON
3. Services used:
   - IBM Watson Studio-ky
   - Machine Learning-te
   - node-red-hzblo-2020--cloudant
   - Continuous Delivery
4. Cloud Foundry apps: Node RED HZBLO
5. Storage: Cloud Object Storage-ol

Outcome:

Develop an end-to-end web application that predicts the probability of females having diabetes. The application has to be built with Python-Flask or Django framework with the machine learning model trained & deployed on IBM Watson Studio.

3. THEORITICAL ANALYSIS

3.1. Block diagram

Proposed Technical Architecture:

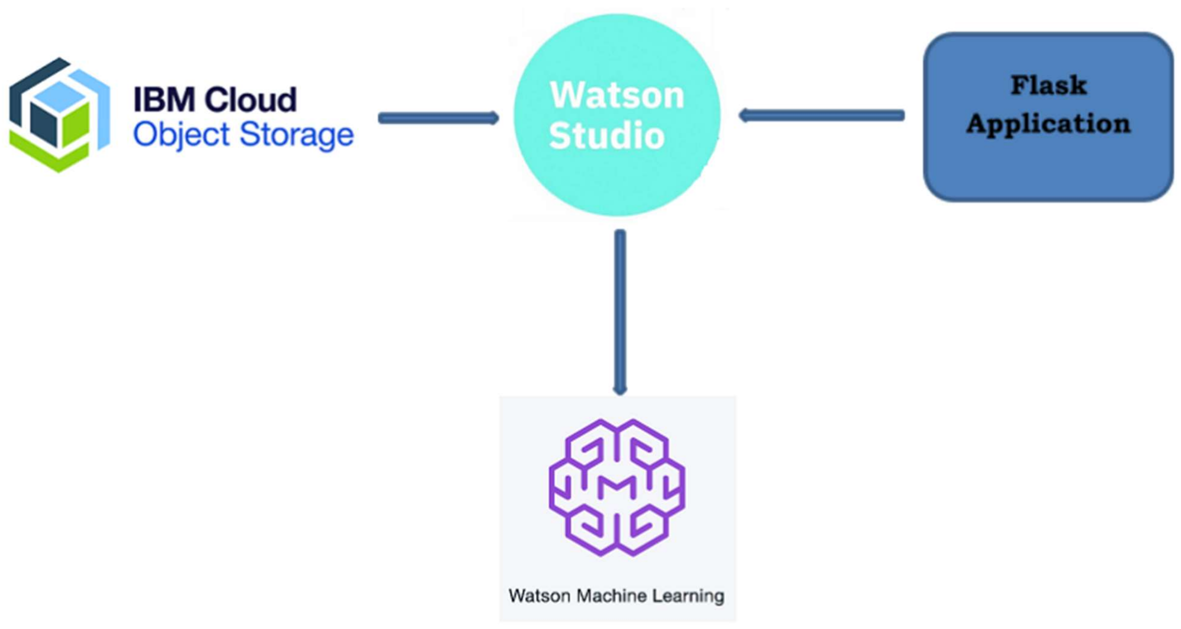


Fig. 3.1 Proposed Technical Architecture

4. EXPERIMENTAL INVESTIGATIONS

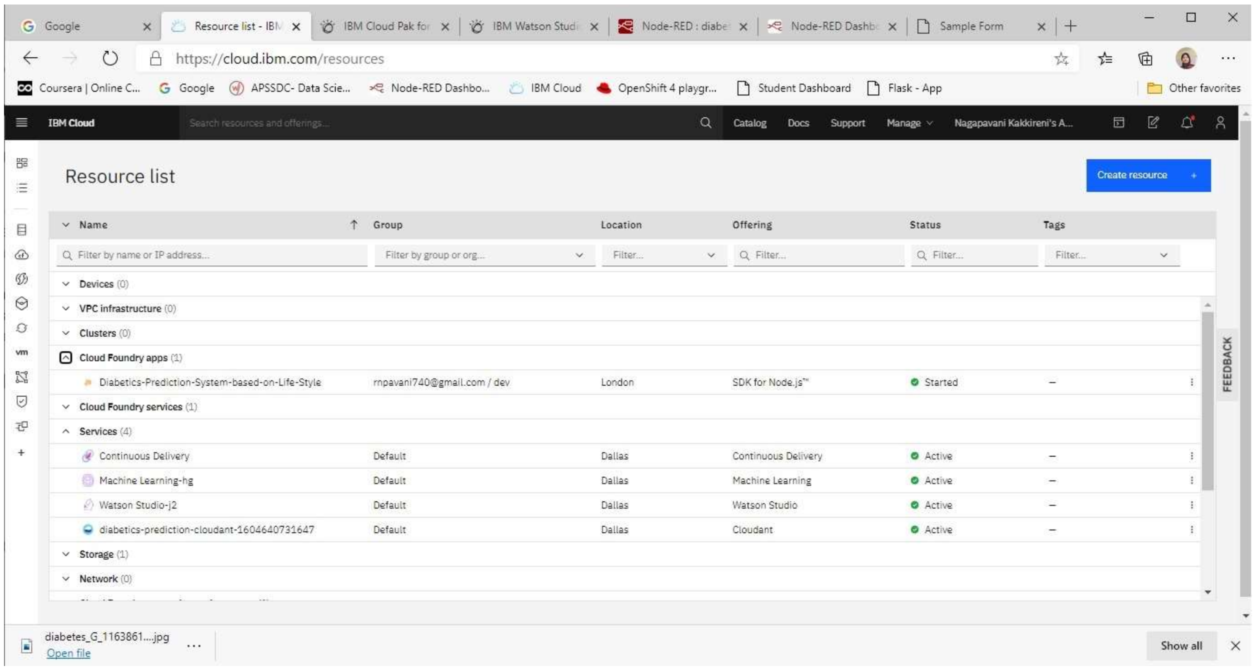
The tools in Machine Learning and Watson Studio available in IBM services catalog were explored to create the project in addition to Node-RED to create the UI.

4.1 Tasks Completed:

- Collecting Dataset
- Create IBM Academic Initiative Account
- Login to IBM Cloud
- Create Cloud Object Storage Service
- Download Watson Studio Desktop
- Create Watson Studio Platform
- Create Machine Learning Service
- Create A Project in Watson Studio
- Upload the Dataset
- Train a model in Watson Studio
- Deploying the Model
- Create UI using Node Red service
- Build Flask Application

4.2 Steps to be followed to build the project:

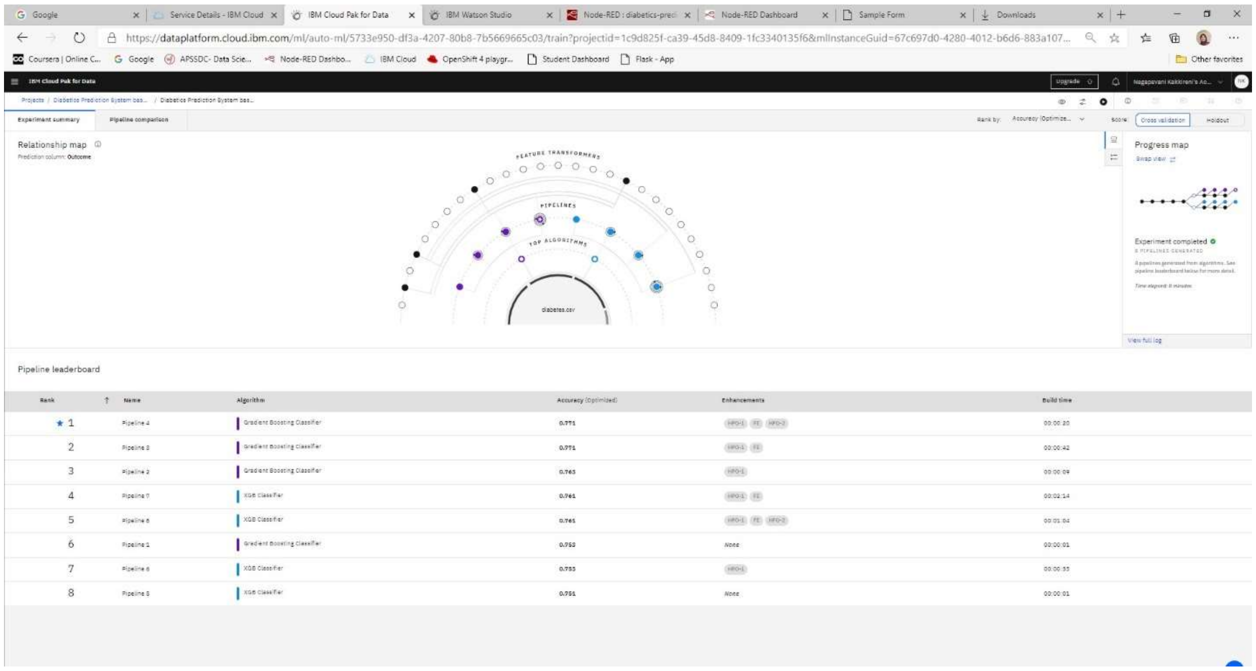
- Create a project in IBM Watson Studio – Predict Heart Failure.
- Add Auto AI experiment.
- Create a Machine Learning instance.
- Associate ML instance to the project.
- Load the dataset to cloud object storage.
- Select the target variable (HEARTFAILURE parameter) in the dataset
- Train the model.
- Deploy.
- Build web application using Node-Red.



Resource list

Name	Group	Location	Offering	Status	Tags
Devices (0)					
VPC infrastructure (0)					
Clusters (0)					
Cloud Foundry apps (1)					
Diabetics-Prediction-System-based-on-Life-Style	rnpavani740@gmail.com / dev	London	SDK for Node.js™	Started	–
Cloud Foundry services (1)					
Services (4)					
Continuous Delivery	Default	Dallas	Continuous Delivery	Active	–
Machine Learning-hg	Default	Dallas	Machine Learning	Active	–
Watson Studio-j2	Default	Dallas	Watson Studio	Active	–
diabetics-prediction-cloudant-1604640731647	Default	Dallas	Cloudant	Active	–
Storage (1)					
Network (0)					

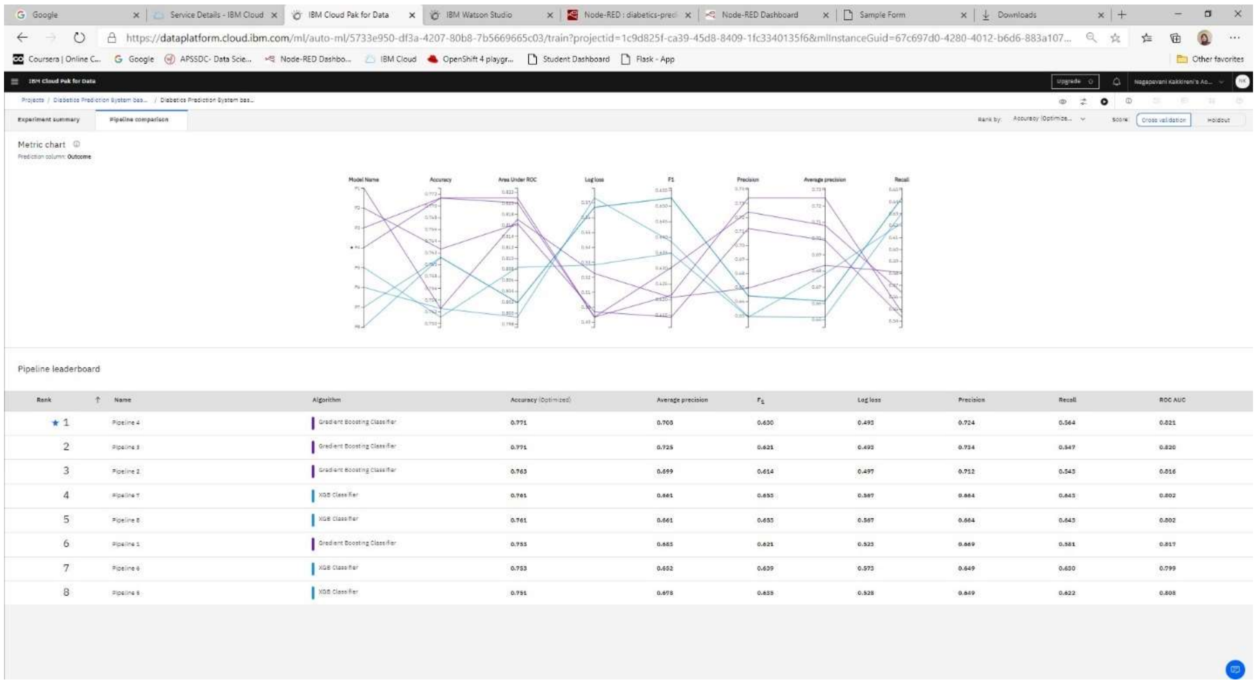
Figure 1:-IBM Cloud services, storage space and cloud foundry app



Pipeline leaderboard

Rank	Name	Algorithm	Accuracy (Optimized)	Enhancements	Build time
★ 1	Pipeline 4	Gradient Boosting Classifier	0.771	HPO-1 FE HPO-2	00:00:20
2	Pipeline 3	Gradient Boosting Classifier	0.771	HPO-1 FE	00:00:42
3	Pipeline 2	Gradient Boosting Classifier	0.763	HPO-1	00:00:09
4	Pipeline 7	XGB Classifier	0.761	HPO-1 FE	00:02:14
5	Pipeline 8	XGB Classifier	0.761	HPO-1 FE HPO-2	00:01:04
6	Pipeline 1	Gradient Boosting Classifier	0.753	None	00:00:01
7	Pipeline 6	XGB Classifier	0.753	HPO-1	00:00:33
8	Pipeline 5	XGB Classifier	0.751	None	00:00:01

Figure 2:- Comparison between various algorithms



Pipeline leaderboard

Rank	Name	Algorithm	Accuracy (Optimized)	Average precision	F_1	Log loss	Precision	Recall	ROC AUC
★ 1	Pipeline 4	Gradient Boosting Classifier	0.771	0.703	0.630	0.493	0.724	0.564	0.821
2	Pipeline 3	Gradient Boosting Classifier	0.771	0.725	0.621	0.493	0.734	0.547	0.820
3	Pipeline 2	Gradient Boosting Classifier	0.763	0.699	0.614	0.497	0.712	0.543	0.816
4	Pipeline 7	XGB Classifier	0.761	0.661	0.653	0.567	0.664	0.643	0.802
5	Pipeline 8	XGB Classifier	0.761	0.661	0.653	0.567	0.664	0.643	0.802
6	Pipeline 1	Gradient Boosting Classifier	0.753	0.683	0.621	0.523	0.669	0.581	0.817
7	Pipeline 6	XGB Classifier	0.753	0.652	0.639	0.573	0.649	0.630	0.799
8	Pipeline 5	XGB Classifier	0.751	0.678	0.635	0.528	0.649	0.622	0.808

Figure 3:-Pipeline Comparison

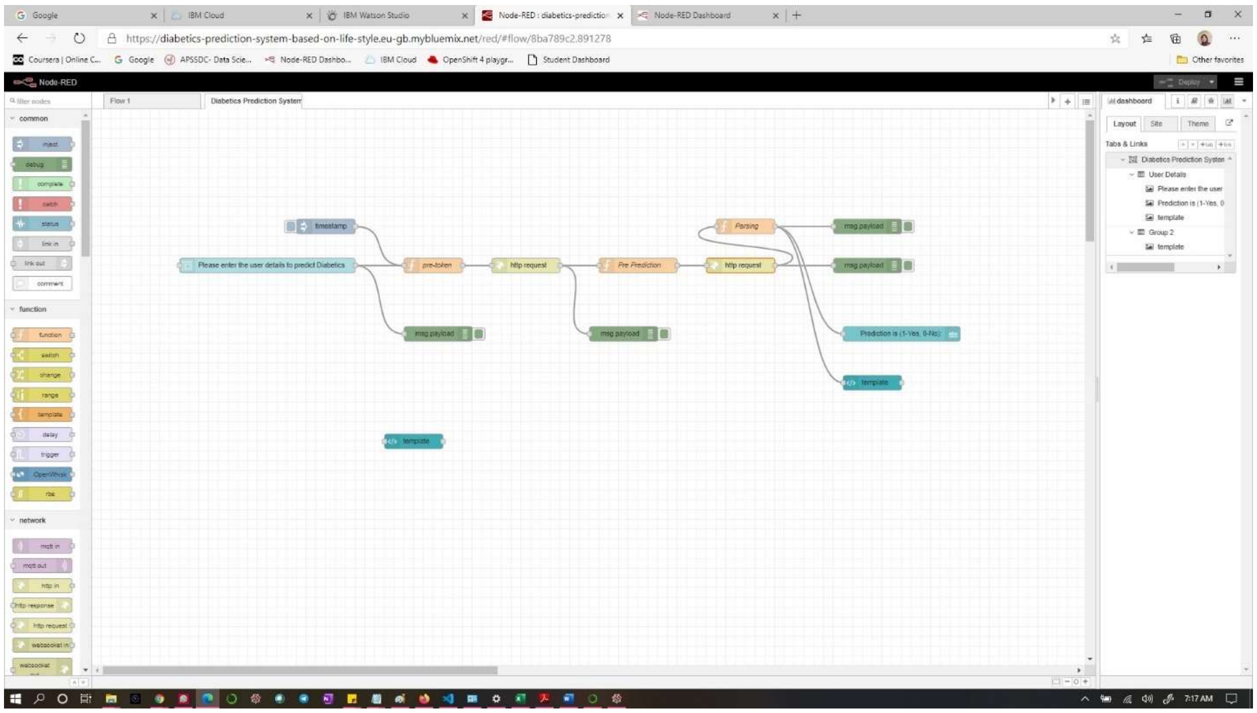


Figure4:-Node-RED-Flow

5. FLOWCHART

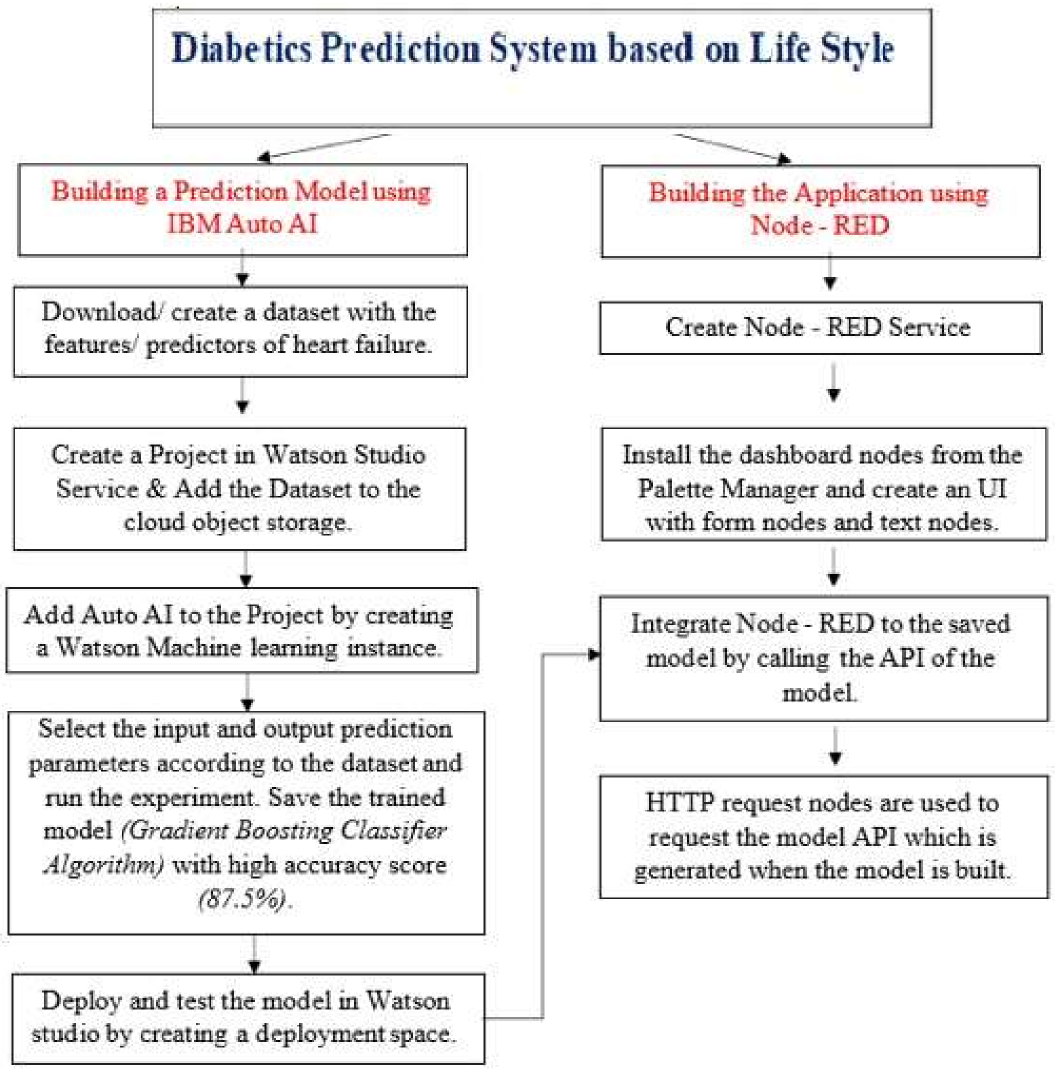
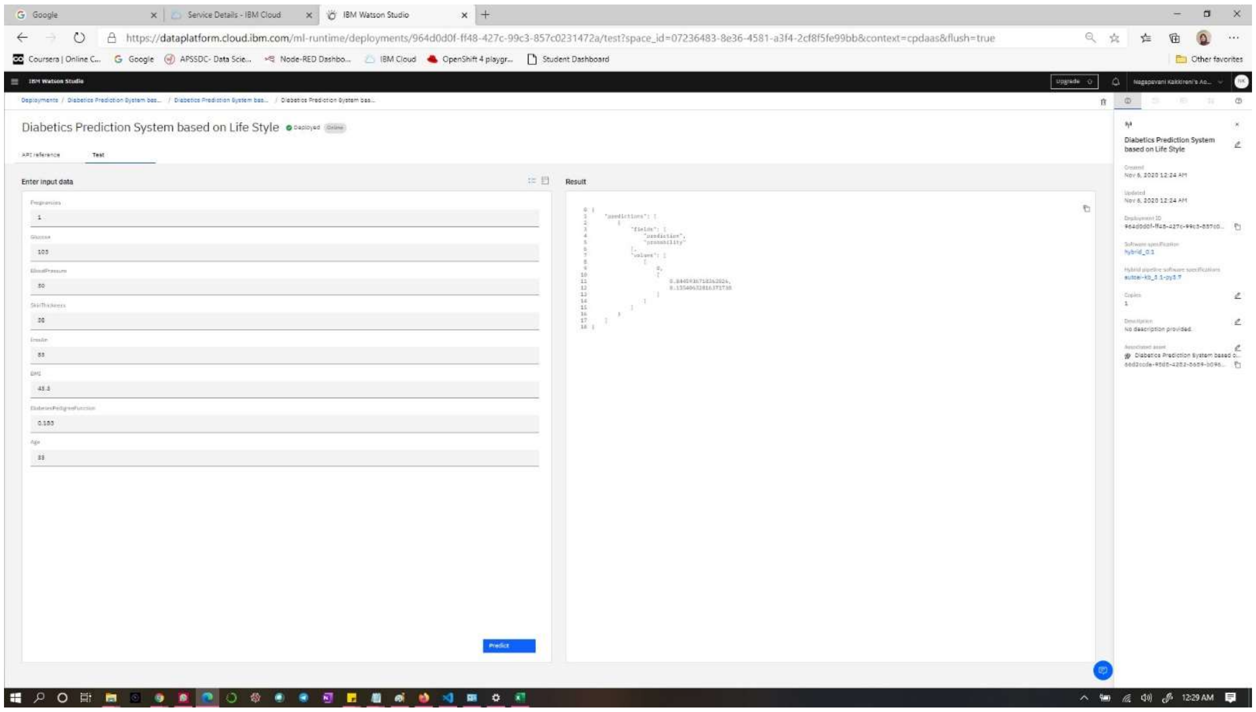

Diabetics Prediction System based on Life Style
Building a Prediction Model using IBM Auto AI
Download/ create a dataset with the features/ predictors of heart failure.
Create a Project in Watson Studio Service & Add the Dataset to the cloud object storage.
Add Auto AI to the Project by creating a Watson Machine learning instance.
Select the input and output prediction parameters according to the dataset and run the experiment. Save the trained model *(Gradient Boosting Classifier Algorithm)* with high accuracy score *(87.5%)*.
Deploy and test the model in Watson studio by creating a deployment space.
Building the Application using Node - RED
Create Node - RED Service
Install the dashboard nodes from the Palette Manager and create an UI with form nodes and text nodes.
Integrate Node - RED to the saved model by calling the API of the model.
HTTP request nodes are used to request the model API which is generated when the model is built.

Fig 5 : Flowchart

6. RESULT



Diabetics Prediction System based on Life Style

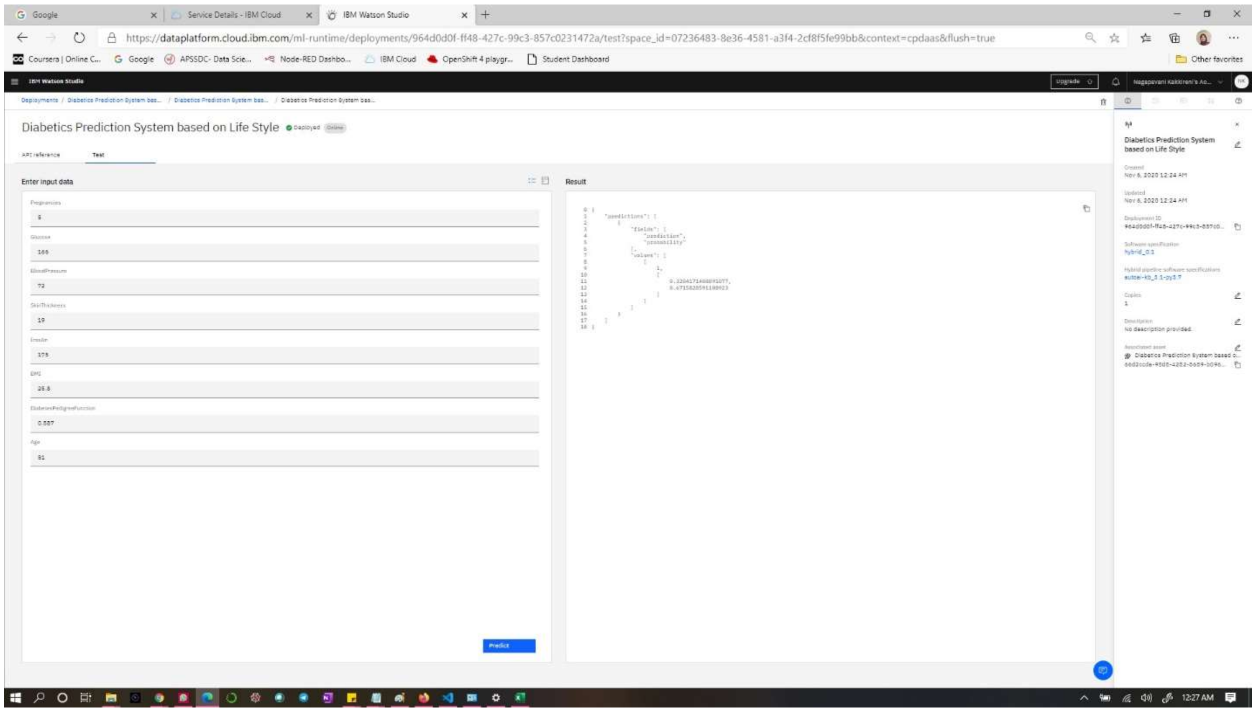
Enter input data

Pregnancies: 1
Glucose: 103
BloodPressure: 80
SkinThickness: 38
Insulin: 83
BMI: 43.3
DiabetesPedigreeFunction: 0.183
Age: 33

Result

```
{
    "predictions": [
        {
            "fields": [
                "prediction",
                "probability"
            ],
            "values": [
                [
                    0,
                    [
                        0.8445936718262026,
                        0.15540632817371738
                    ]
                ]
            ]
        }
    ]
}
```

Figure 6:- Test_Model_Output(0)



Diabetics Prediction System based on Life Style

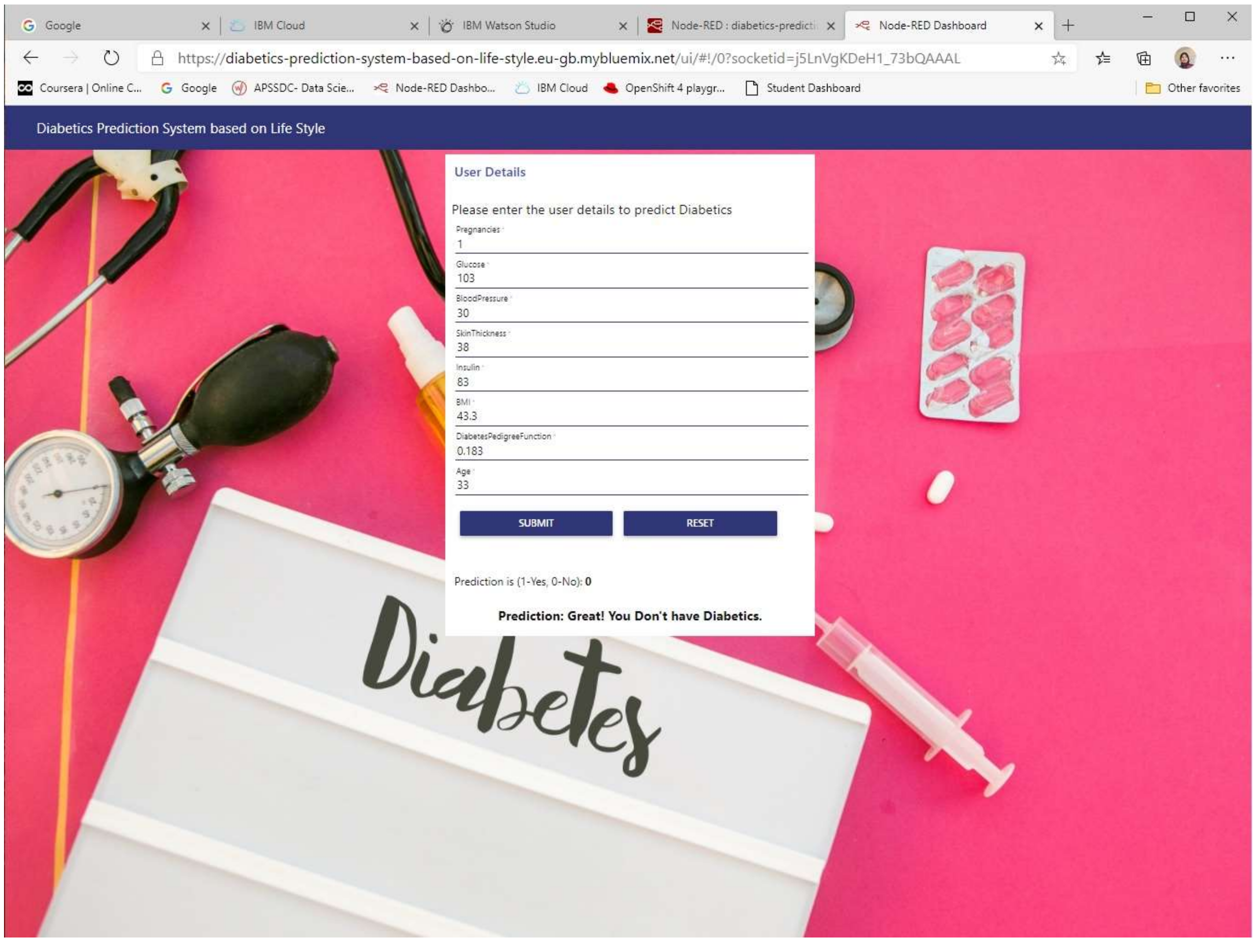
Enter input data

Pregnancies: 5
Glucose: 166
BloodPressure: 72
SkinThickness: 19
Insulin: 175
BMI: 25.8
DiabetesPedigreeFunction: 0.587
Age: 51

Result

```
{
    "predictions": [
        {
            "fields": [
                "prediction",
                "probability"
            ],
            "values": [
                [
                    1,
                    [
                        0.3204171408891077,
                        0.6795828591108923
                    ]
                ]
            ]
        }
    ]
}
```

Figure 7:- Test_Model_Output(1)



Diabetics Prediction System based on Life Style

User Details

Please enter the user details to predict Diabetics

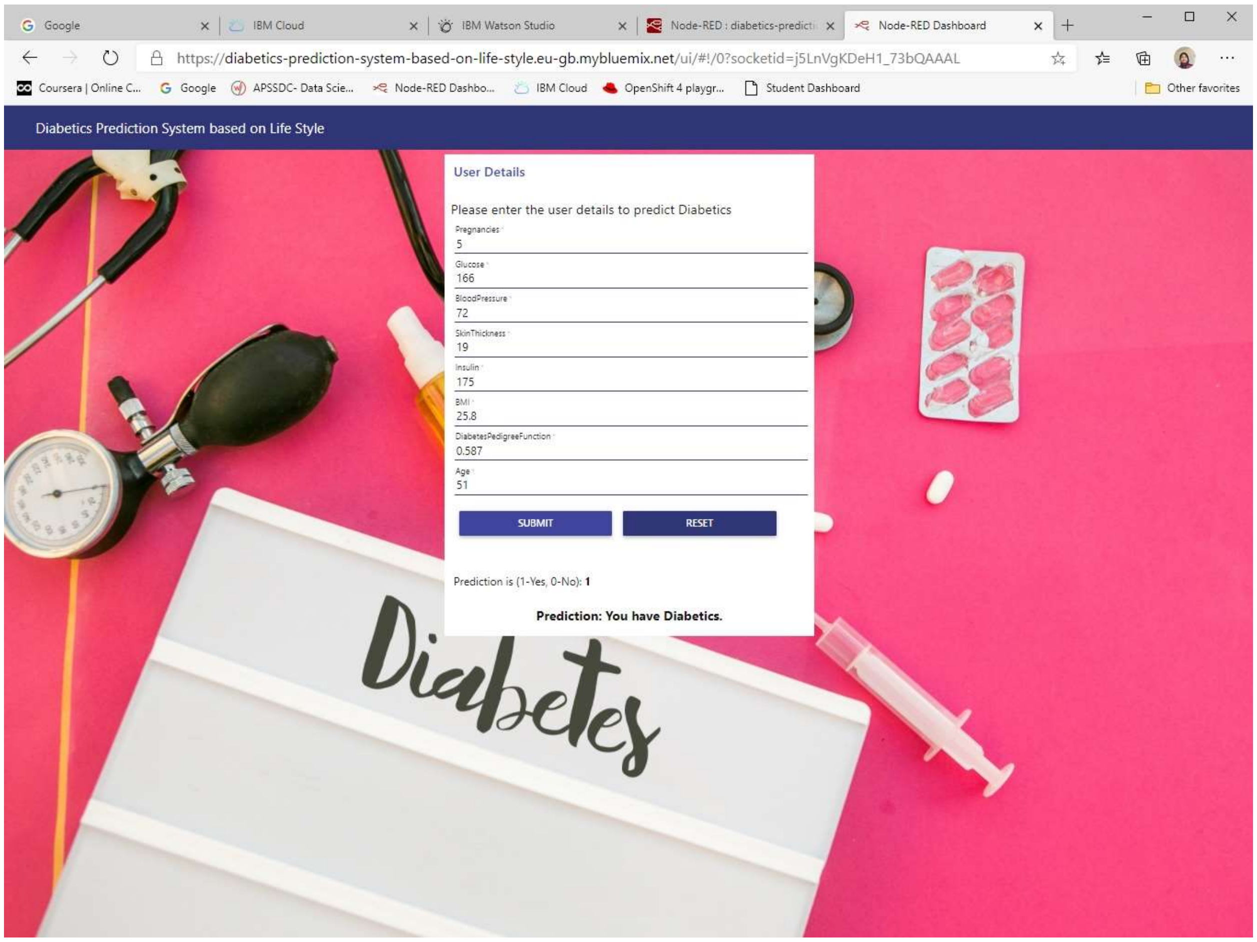
Pregnancies: 1
Glucose: 103
BloodPressure: 30
SkinThickness: 38
Insulin: 83
BMI: 43.3
DiabetesPedigreeFunction: 0.183
Age: 33

SUBMIT RESET

Prediction is (1-Yes, 0-No): **0**

Prediction: Great! You Don't have Diabetics.

Figure 8:- Node-RED-App- Output(0)



Diabetics Prediction System based on Life Style

User Details

Please enter the user details to predict Diabetics

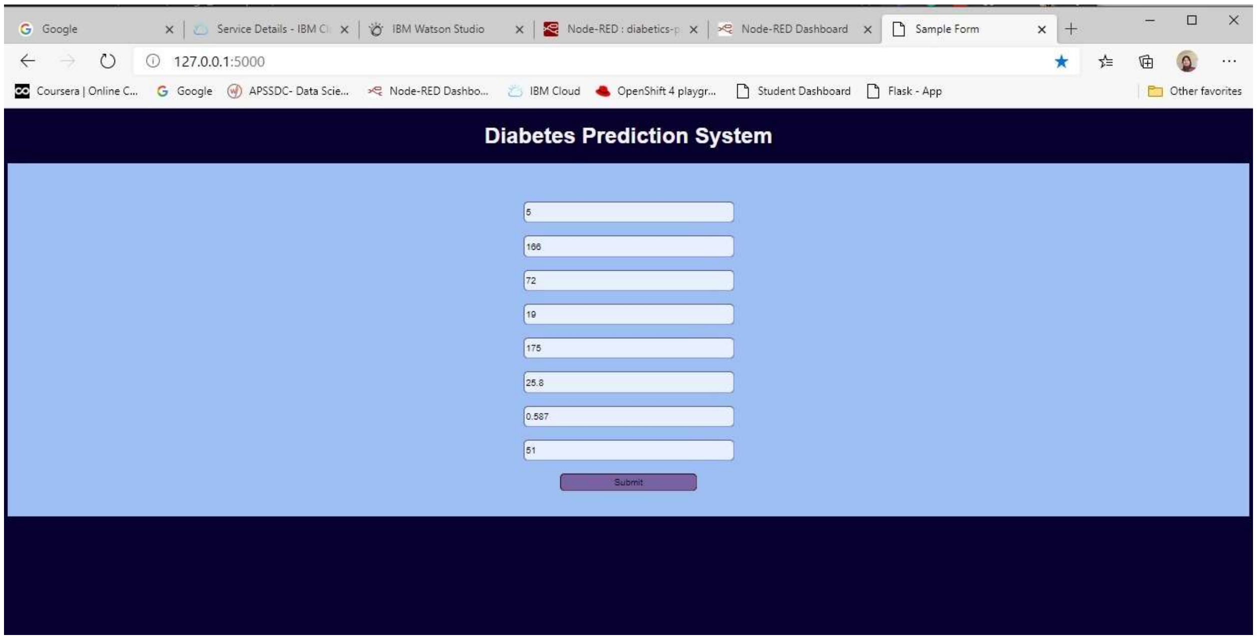
Pregnancies	5
Glucose	166
BloodPressure	72
SkinThickness	19
Insulin	175
BMI	25.8
DiabetesPedigreeFunction	0.587
Age	51

SUBMIT RESET

Prediction is (1-Yes, 0-No): **1**

Prediction: You have Diabetics.

Figure 9:-Node-RED-App-Output(1)



Diabetes Prediction System

5

166

72

19

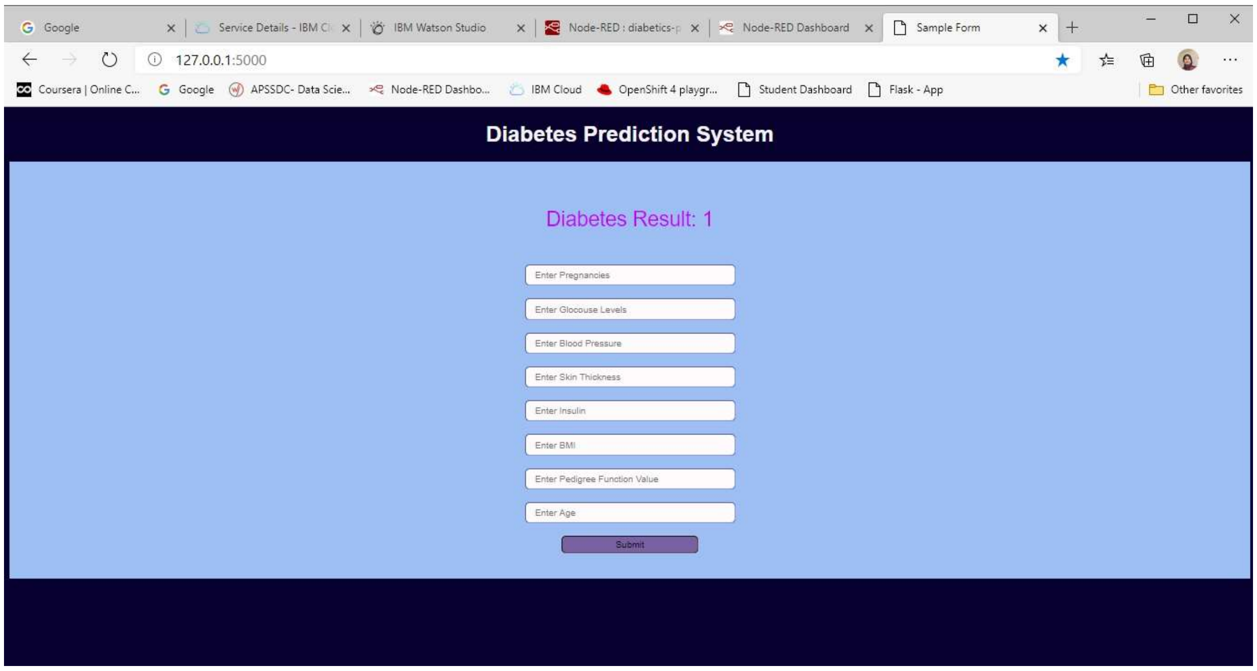
175

25.8

0.587

51

Figure 10:- Flask-Application-Input



Diabetes Prediction System

Diabetes Result: 1

Enter Pregnancies

Enter Glocouse Levels

Enter Blood Pressure

Enter Skin Thickness

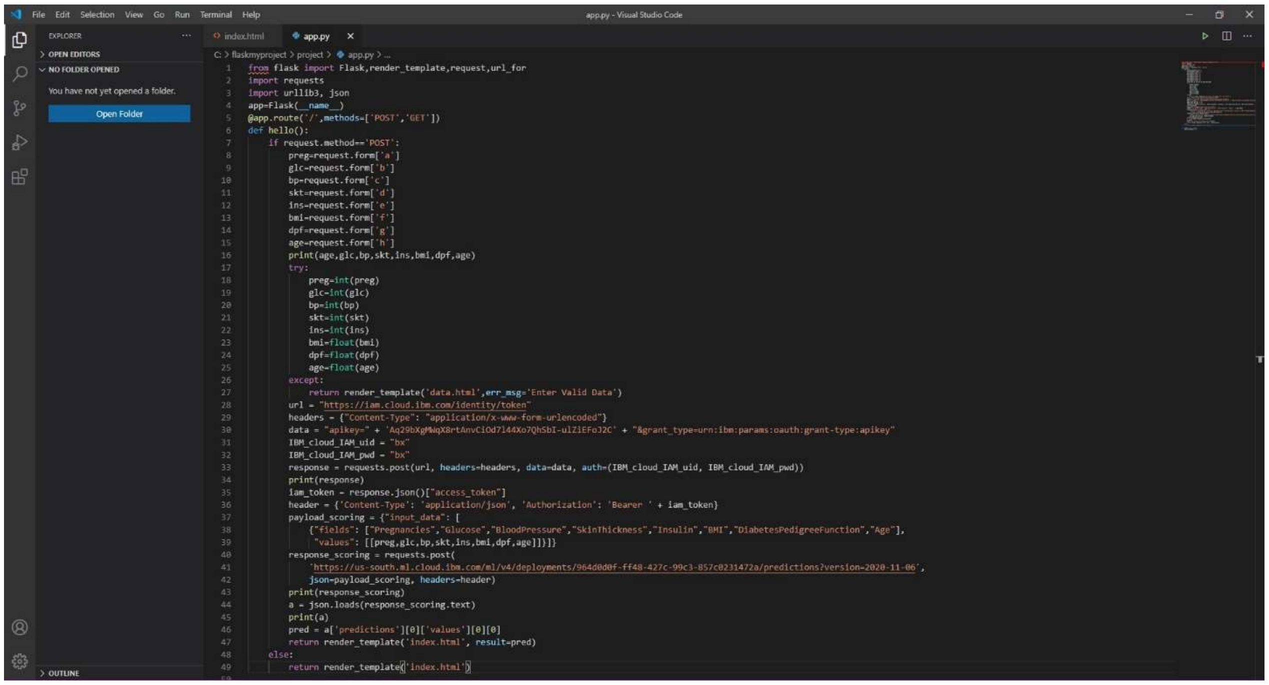
Enter Insulin

Enter BMI

Enter Pedigree Function Value

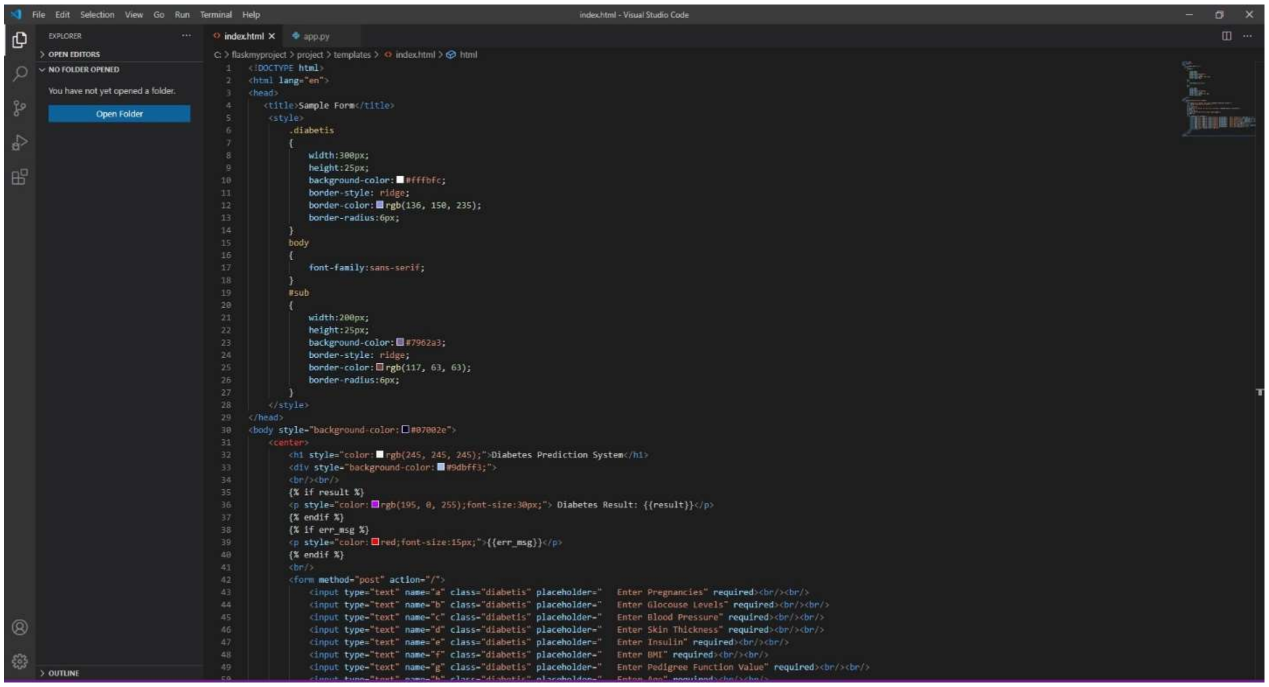
Enter Age

Figure 11:- Flask-Application-Output



```python
from flask import Flask,render_template,request,url_for
import requests
import urllib3, json
app=Flask(__name__)
@app.route('/',methods=['POST','GET'])
def hello():
    if request.method=='POST':
        preg=request.form['a']
        glc=request.form['b']
        bp=request.form['c']
        skt=request.form['d']
        ins=request.form['e']
        bmi=request.form['f']
        dpf=request.form['g']
        age=request.form['h']
        print(age,glc,bp,skt,ins,bmi,dpf,age)
        try:
            preg=int(preg)
            glc=int(glc)
            bp=int(bp)
            skt=int(skt)
            ins=int(ins)
            bmi=float(bmi)
            dpf=float(dpf)
            age=float(age)
        except:
            return render_template('data.html',err_msg='Enter Valid Data')
        url = "https://iam.cloud.ibm.com/identity/token"
        headers = {"Content-Type": "application/x-www-form-urlencoded"}
        data = "apikey=" + 'Aq29bXgMWqX8rtAnvCiOd7144Xo7QhSbI-ulZiEFoJ2C' + "&grant_type=urn:ibm:params:oauth:grant-type:apikey"
        IBM_cloud_IAM_uid = "bx"
        IBM_cloud_IAM_pwd = "bx"
        response = requests.post(url, headers=headers, data=data, auth=(IBM_cloud_IAM_uid, IBM_cloud_IAM_pwd))
        print(response)
        iam_token = response.json()["access_token"]
        header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + iam_token}
        payload_scoring = {"input_data": [
            {"fields": ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age"],
             "values": [[preg,glc,bp,skt,ins,bmi,dpf,age]]}]}
        response_scoring = requests.post(
            'https://us-south.ml.cloud.ibm.com/ml/v4/deployments/964d0d0f-ff48-427c-99c3-857c0231472a/predictions?version=2020-11-06',
            json=payload_scoring, headers=header)
        print(response_scoring)
        a = json.loads(response_scoring.text)
        print(a)
        pred = a['predictions'][0]['values'][0][0]
        return render_template('index.html', result=pred)
    else:
        return render_template('index.html')
```

Figure 12:- app.py written in VS code



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Sample Form</title>
     <style>
        .diabetis
        {
            width:300px;
            height:25px;
            background-color:#fffbfc;
            border-style: ridge;
            border-color:rgb(136, 150, 235);
            border-radius:6px;
        }
        body
        {
            font-family:sans-serif;
        }
        #sub
        {
            width:200px;
            height:25px;
            background-color:#7962a3;
            border-style: ridge;
            border-color:rgb(117, 63, 63);
            border-radius:6px;
        }
     </style>
</head>
<body style="background-color:#07002e">
     <center>
        <h1 style="color:rgb(245, 245, 245);">Diabetes Prediction System</h1>
        <div style="background-color:#9dbff3;">
        <br/><br/>
        {% if result %}
        <p style="color:rgb(195, 0, 255);font-size:30px;"> Diabetes Result: {{result}}</p>
        {% endif %}
        {% if err_msg %}
        <p style="color:red;font-size:15px;">{{err_msg}}</p>
        {% endif %}
        <br/>
        <form method="post" action="/">
            <input type="text" name="a" class="diabetis" placeholder="  Enter Pregnancies" required><br/><br/>
            <input type="text" name="b" class="diabetis" placeholder="  Enter Glocouse Levels" required><br/><br/>
            <input type="text" name="c" class="diabetis" placeholder="  Enter Blood Pressure" required><br/><br/>
            <input type="text" name="d" class="diabetis" placeholder="  Enter Skin Thickness" required><br/><br/>
            <input type="text" name="e" class="diabetis" placeholder="  Enter Insulin" required><br/><br/>
            <input type="text" name="f" class="diabetis" placeholder="  Enter BMI" required><br/><br/>
            <input type="text" name="g" class="diabetis" placeholder="  Enter Pedigree Function Value" required><br/><br/>
            <input type="text" name="h" class="diabetis" placeholder="  Enter Age" required><br/><br/>
```

Figure 13:- index.html written in VS code

7. ADVANTAGES & DISADVANTAGES

7.1. Advantages

1. Identification without going to Diagnostic centers.

2. No need to waste money for tests

3. No waiting time and anxiety for reports

4. can Identify at early stage so that care can be taken.

7.2. Disadvantages

The disadvantage of the online prediction tool is its sensitivity and accuracy for clinical use. It completely depends on the dataset used to train the model for prediction.

1. Prediction can be wrong sometimes according to confusion matrix which may create worst consequences.

2. Accuracy can be less sometimes

3. Some more parameters can be needed sometimes for predicting accurately

8. CONCLUSION

Most of the population all around the world has diabetes. Predicting this disease at earlier stages plays vital role for treatment of patient. The Pima Indian diabetes dataset has been used in this model. The algorithm is best among others and got 77% accuracy. The obtained results appear to be very important for predicting diabetes accurately. The main theme is to prevent and cure diabetes and to improve the lives of all people affected by diabetes at faster rate and to help the doctors to start the treatments early.

9. FUTURE SCOPE

The Accuracy obtained is 77%.It can be increased by following different methods either by considering more parameters or huge data set can be used to train the model. The confusion matrix clearly states that there can be false positive, false negative cases which can cause to serious consequences and to be overcome in future researches.

10. BIBILOGRAPHY

10.1. REFERENCES

- https://www.kaggle.com/datasets
- https://cloud.ibm.com/
- https://cloud.ibm.com/catalog/services/watson-studio
- https://cloud.ibm.com/developer/appservice/create-app
- https://smartinternz.com/assets/Steps-to-be-followed-to-download-Watson-Studio-in-your-Local-System.pdf

10.2. APPENDIX

A. Source code:

1) App.py code

```
from flask import Flask,render_template,request,url_for
```

```
import requests
import urllib3, json
app=Flask(__name__)
@app.route('/',methods=['POST','GET'])
def hello():
    if request.method=='POST':
        preg=request.form['a']
        glc=request.form['b']
        bp=request.form['c']
        skt=request.form['d']
        ins=request.form['e']
        bmi=request.form['f']
        dpf=request.form['g']
        age=request.form['h']
        print(age,glc,bp,skt,ins,bmi,dpf,age)
        try:
            preg=int(preg)
            glc=int(glc)
            bp=int(bp)
            skt=int(skt)
            ins=int(ins)
            bmi=float(bmi)
            dpf=float(dpf)
            age=float(age)
        except:
            return render_template('data.html',err_msg='Enter Valid Data')
        url = "https://iam.cloud.ibm.com/identity/token"
        headers = {"Content-Type": "application/x-www-form-urlencoded"}
        data = "apikey=" + 'Aq29bXgMWqX8rtAnvCiOd7l44Xo7QhSbI-ulZiEFoJ2C' +
      "&grant_type=urn:ibm:params:oauth:grant-type:apikey"
        IBM_cloud_IAM_uid = "bx"
        IBM_cloud_IAM_pwd = "bx"
        response = requests.post(url, headers=headers, data=data, auth=(IBM_cloud_IAM_uid,
      IBM_cloud_IAM_pwd))
        print(response)
        iam_token = response.json()["access_token"]
        header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + iam_token}
        payload_scoring = {"input_data": [
            {"fields":
      ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigre
      eFunction","Age"],
             "values": [[preg,glc,bp,skt,ins,bmi,dpf,age]]}]}
        response_scoring = requests.post(
            'https://us-south.ml.cloud.ibm.com/ml/v4/deployments/964d0d0f-ff48-427c-99c3-
      857c0231472a/predictions?version=2020-11-06',
            json=payload_scoring, headers=header)
        print(response_scoring)
        a = json.loads(response_scoring.text)
        print(a)
        pred = a['predictions'][0]['values'][0][0]
        return render_template('index.html', result=pred)
    else:
        return render_template('index.html')

if __name__ == '__main__':
        app.run(debug=True)
```

2) Index.html code

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <title>Sample Form</title>
   <style>
       .diabetis
       {
          width:300px;
          height:25px;
          background-color:#fffbfc;
          border-style: ridge;
          border-color:rgb(136, 150, 235);
          border-radius:6px;
       }
       body
       {
          font-family:sans-serif;
       }
       #sub
       {
          width:200px;
          height:25px;
          background-color:#7962a3;
          border-style: ridge;
          border-color:rgb(117, 63, 63);
          border-radius:6px;
       }
   </style>
</head>
<body style="background-color:#07002e">
   <center>
       <h1 style="color:rgb(245, 245, 245);">Diabetes Prediction System</h1>
       <div style="background-color:#9dbff3;">
       <br/><br/>
```

```
    {% if result %}
    <p style="color:rgb(195, 0, 255);font-size:30px;"> Diabetes Result: {{result}}</p>
    {% endif %}
    {% if err_msg %}
    <p style="color:red;font-size:15px;">{{err_msg}}</p>
    {% endif %}
    <br/>
    <form method="post" action="/">
      <input type="text" name="a" class="diabetis" placeholder="   Enter Pregnancies"
   required><br/><br/>
      <input type="text" name="b" class="diabetis" placeholder="   Enter Glocouse Levels"
   required><br/><br/>
      <input type="text" name="c" class="diabetis" placeholder="   Enter Blood Pressure"
   required><br/><br/>
      <input type="text" name="d" class="diabetis" placeholder="   Enter Skin Thickness"
   required><br/><br/>
      <input type="text" name="e" class="diabetis" placeholder="   Enter Insulin"
   required><br/><br/>
      <input type="text" name="f" class="diabetis" placeholder="   Enter BMI"
   required><br/><br/>
      <input type="text" name="g" class="diabetis" placeholder="   Enter Pedigree Function
   Value" required><br/><br/>
      <input type="text" name="h" class="diabetis" placeholder="   Enter Age"
   required><br/><br/>
      <input type="submit" value="Submit" id="sub">
      <br/><br/><br/>
    </form>
    </div>
  </center>
</body>
   </html>
```

3) IBM Auto AI code

Trusted

Jupyter Server: local

Python 3: Not Started

##
#Licensed Materials - Property of IBM #(C) Copyright IBM Corp. 2020 #US Government Users Restricted Rights - Use, duplication disclosure restricted #by GSA ADP Schedule Contract with IBM Corp.
##
##

The auto-generated notebooks are subject to the International License Agreement for Non-Warranted Programs (or equivalent) and License Information document for Watson Studio Auto-generated Notebook (License Terms), such agreements located in the link below. Specifically, the Source Components and Sample Materials clause included in the License Information document for Watson Studio Auto-generated Notebook applies to the auto-generated notebooks. By downloading, copying, accessing, or otherwise using the materials, you agree to the License Terms. http://www14.software.ibm.com/cgi-bin/weblap/lap.pl?li_formnum=L-AMCU-BHU2B7&title=IBM%20Watson%20Studio%20Auto-generated%20Notebook%20V2.1

IBM AutoAI Auto-Generated Notebook v1.14.2

Note: Notebook code generated using AutoAI will execute successfully. If code is modified or reordered,

there is no guarantee it will successfully execute. This pipeline is optimized for the original dataset.

The pipeline may fail or produce sub-optimium results if used with different data. For different data,

please consider returning to AutoAI Experiments to generate a new pipeline. Please read our documentation

for more information:

(Cloud Platform) https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-notebook.html . (Cloud Pak For Data) https://www.ibm.com/support/knowledgecenter/SSQNUZ_3.0.0/wsj/analyze-data/autoai-notebook.html .

Before modifying the pipeline or trying to re-fit the pipeline, consider:
The notebook converts dataframes to numpy arrays before fitting the pipeline
(a current restriction of the preprocessor pipeline). The known_values_list is passed by reference
and populated with categorical values during fit of the preprocessing pipeline. Delete its members before re-fitting.

Representing Pipeline_4
1. Set Up
If lightgbm or xgboost installation fails, please follow:

lightgbm docs
xgboost docs
[-]

```
try:
    import autoai_libs
except Exception as e:
    import subprocess
    out = subprocess.check_output('pip install autoai-libs'.split(' '))
    for line in out.splitlines():
        print(line)
    import autoai_libs
import sklearn
```

[-]

```
# compose a decorator to assist pipeline instantiation via import of modules and installation of
    packages
def decorator_retries(func):
    def install_import_retry(*args, **kwargs):
        retries = 0
        successful = False
        failed_retries = 0
        while retries < 100 and failed_retries < 10 and not successful:
            retries += 1
            failed_retries += 1
            try:
```

2. Compose Pipeline

[-]

```
# metadata necessary to replicate AutoAI scores with the pipeline
_input_metadata = {'separator': ',', 'excel_sheet': 0, 'target_label_name': 'Outcome', 'learning_type':
    'classification', 'subsampling': None, 'pos_label': 1, 'pn': 'P4', 'cv_num_folds': 3,
    'holdout_fraction': 0.1, 'optimization_metric': 'accuracy', 'random_state': 33, 'data_source': ''}

# define a function to compose the pipeline, and invoke it
@decorator_retries
def compose_pipeline():
    import numpy
    from numpy import nan
```

3. Extract needed parameter values from AutoAI run metadata

[-]

```
# Metadata used in retrieving data and computing metrics.  Customize as necessary for your
     environment.
#data_source='replace_with_path_and_csv_filename'
target_label_name = _input_metadata['target_label_name']
learning_type = _input_metadata['learning_type']
optimization_metric = _input_metadata['optimization_metric']
random_state = _input_metadata['random_state']
cv_num_folds = _input_metadata['cv_num_folds']
holdout_fraction = _input_metadata['holdout_fraction']
if 'data_provenance' in _inpu
```

4. Create dataframe from dataset in Cloud Object Storage

[-]

```
# @hidden_cell
# The following code contains the credentials for a file in your IBM Cloud Object Storage.
# You might want to remove those credentials before you share your notebook.
credentials_0 = {
    'ENDPOINT': 'https://s3-api.us-geo.objectstorage.softlayer.net',
    'IBM_AUTH_ENDPOINT': 'https://iam.bluemix.net/oidc/token/',
    'APIKEY': 'UEESapZQQ3jWKIOk_eUF6Oz9t-ARMCPLvx4AKCYUUbDt',
    'BUCKET': 'diabeticspredictionsystembasedonl-donotdelete-pr-mgqupj7gse7154',
    'FILE': 'diabetes.
[-]




# Read the data as a dataframe
import pandas as pd

csv_encodings=['UTF-8','Latin-1'] # supplement list of encodings as necessary for your data
df = None
readable = None  # if automatic detection fails, you can supply a filename here

# First, obtain a readable object
# Cloud Object Storage data access
# Assumes COS credentials are in a dictionary named 'credentials_0'
```

5. Preprocess Data

```
[-]
# Drop rows whose target is not defined
target = target_label_name # your target name here
if learning_type == 'regression':
    df[target] = pd.to_numeric(df[target], errors='coerce')
df.dropna('rows', how='any', subset=[target], inplace=True)

[-]
# extract X and y
df_X = df.drop(columns=[target])
df_y = df[target]


[-]
# Detach preprocessing pipeline (which needs to see all training data)
```

```
preprocessor_index = -1
preprocessing_steps = []
for i, step in enumerate(pipeline.steps):
    preprocessing_steps.append(step)
    if step[0]=='preprocessor':
        preprocessor_index = i
        break
#if len(pipeline.steps) > preprocessor_index+1 and pipeline.steps[preprocessor_index + 1][0] ==
    'cognito':
    #preprocessor_index += 1
```

[-]

```
# Preprocess X
# preprocessor should see all data for cross_validate on the remaining steps to match autoai scores
known_values_list.clear()  #  known_values_list is filled in by the preprocessing_pipeline if needed
preprocessing_pipeline.fit(df_X.values, df_y.values)
X_prep = preprocessing_pipeline.transform(df_X.values)
```

6. Split data into Training and Holdout sets

[-]

```
# determine learning_type and perform holdout split (stratify conditionally)
if learning_type is None:
    # When the problem type is not available in the metadata, use the sklearn type_of_target to
     determine whether to stratify the holdout split
    # Caution:  This can mis-classify regression targets that can be expressed as integers as
     multiclass, in which case manually override the learning_type
    from sklearn.utils.multiclass import type_of_target
    if type_of_target(df_y.values) in ['m
```

7. Generate features via Feature Engineering pipeline

[-]

```
#Detach Feature Engineering pipeline if next, fit it, and transform the training data
fe_pipeline = None
if pipeline.steps[0][0] == 'cognito':
    try:
        fe_pipeline = Pipeline(steps=[pipeline.steps[0]])
        X = fe_pipeline.fit_transform(X, y)
        X_holdout = fe_pipeline.transform(X_holdout)
        pipeline.steps = pipeline.steps[1:]
    except IndexError:
        try:
```

8. Additional setup: Define a function that returns a scorer for the target's positive label

[-]

```
# create a function to produce a scorer for a given positive label
def make_pos_label_scorer(scorer, pos_label):
    kwargs = {'pos_label':pos_label}
    for prop in ['needs_proba', 'needs_threshold']:
        if prop+'=True' in scorer._factory_args():
            kwargs[prop] = True
    if scorer._sign == -1:
        kwargs['greater_is_better'] = False
    from sklearn.metrics import make_scorer
    scorer=make_scorer(scorer._score_func, **kwargs)
```

9. Fit pipeline, predict on Holdout set, calculate score, perform cross-validation

[-]

```
# fit the remainder of the pipeline on the training data
pipeline.fit(X,y)
```

[-]

```
# predict on the holdout data
y_pred = pipeline.predict(X_holdout)
```

[-]

```
# compute score for the optimization metric
# scorer may need pos_label, but not all scorers take pos_label parameter
```

```
from sklearn.metrics import get_scorer
scorer = get_scorer(optimization_metric)
score = None
#score = scorer(pipeline, X_holdout, y_holdout)  # this would suffice for simple cases
pos_label = None  # if you want to supply the pos_label, specify it here
if pos_label is None and 'pos_label' in _input_metadata:
    pos_label=_input_metadata['pos_label']
try:


[-]
# cross_validate pipeline using training data
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold, KFold
if learning_type == 'classification':
    fold_generator = StratifiedKFold(n_splits=cv_num_folds, random_state=random_state)
else:
    fold_generator = KFold(n_splits=cv_num_folds, random_state=random_state)
cv_results = cross_validate(pipeline, X, y, cv=fold_generator,
     scoring={optimization_metric:scorer}, return_train_score=True)
import num
[-]
cv_results
```