Diabetes Detection using Machine Learning Algorithms

Semester Project Report

Machine Learning Project

Submitted by

FARAZ AHMAD KHAN
01-136212-009

MUHAMMAD NAVEED
01-136212-054

Department of Computer Science,
Bahria University, Islamabad. [3rd June 2024]

Table of Contents

Abstract

Developing a machine learning model to classify individuals as diabetic or non- diabetic based on their medical attributes. This project will also include a Flask web application to make the model accessible to end-users.

Introduction

Diabetes is a chronic medical condition that affects millions of people worldwide. Early detection and management are crucial for reducing the risk of severe complications. Machine learning models can assist in identifying patterns in medical data, enabling the early prediction of diabetes. This project aims to leverage machine learning techniques to build an accurate diabetes prediction model and deploy it as a user-friendly web application.

Purpose

The purpose of this project is to leverage machine learning to develop a robust predictive model for diabetes. This model aims to support healthcare providers by offering a supplementary tool for screening and early detection of diabetes in patients.

Scope

The project focuses on using a Random Forest algorithm to classify individuals as diabetic or non-diabetic based on a set of predictor variables. The scope includes data preprocessing, model training, evaluation, and deployment using a Flask-based web application.

DATA AND METHODS

Features

The dataset includes several predictor variables such as:

- **Pregnancies:** Number of times pregnant
- **Glucose:** Plasma glucose concentration 2 hours in an oral glucose tolerance test
- **Blood Pressure:** Diastolic blood pressure (mm Hg)
- **Skin Thickness:** Triceps skin fold thickness (mm)
- **Insulin:** 2-Hour serum insulin (mu U/ml)
- **BMI:** Body mass index (weight in kg/(height in m)^2)
- **Diabetes Pedigree Function:** scores likelihood of diabetes based on family history
- **Age:** Age in years

Dataset

https://www.kaggle.com/johndasilva/diabetes

Dataset of diabetes, taken from the hospital Frankfurt, Germany.

Diabetes Prediction

Predict whether a person has diabetes or not.

Dataset Link: https://www.kaggle.com/johndasilva/diabetes

In [1]:
```
# Importing essential libraries
import numpy as np
import pandas as pd
```

In [2]:
```
# Loading the dataset
df = pd.read_csv('kaggle_diabetes.csv')
```

Data Preprocessing

Data preprocessing involves several steps to clean and prepare the data for modeling.

Data Loading and Exploration

The data will be loaded using pandas function with 'lines=True' to handle the data.

Exploring the dataset

In [3]:
```
# Returns number of rows and columns of the dataset
df.shape
```

Out[3]:
```
(2000, 9)
```

In [4]:
```
# Returns an object with all of the column headers
df.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [5]:
```
# Returns different datatypes for each columns (float, int, string, bool, etc.)
df.dtypes
```

Out[5]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

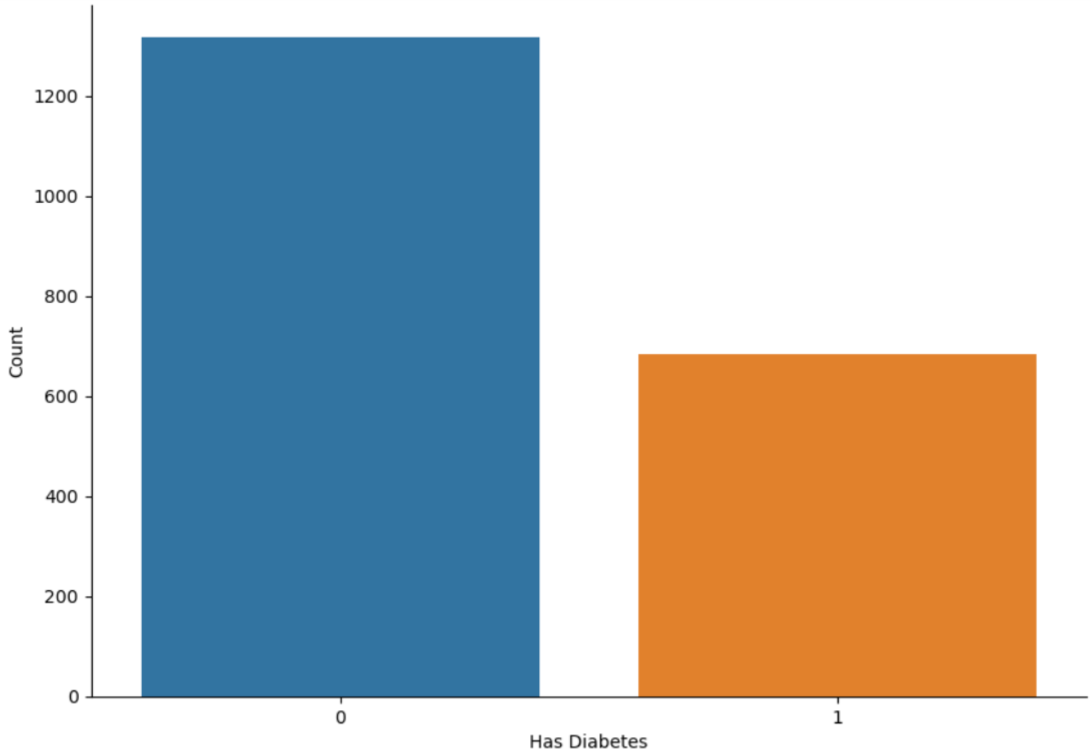
In [7]:
```
# Returns basic information on all columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               2000 non-null   int64  
 1   Glucose                   2000 non-null   int64  
 2   BloodPressure             2000 non-null   int64  
 3   SkinThickness             2000 non-null   int64  
 4   Insulin                   2000 non-null   int64  
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64  
 8   Outcome                   2000 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 140.8 KB
```

In [8]:
```
# Returns basic statistics on numeric columns
df.describe()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	3.703500	121.182500	69.145500	20.935000	80.254000	32.193000	0.470930	33.090500	0.342000
std	3.306063	32.068636	19.188315	16.103243	111.180534	8.149901	0.323553	11.786423	0.474498
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.375000	0.244000	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	40.000000	32.300000	0.376000	29.000000	0.000000
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.800000	0.624000	40.000000	1.000000

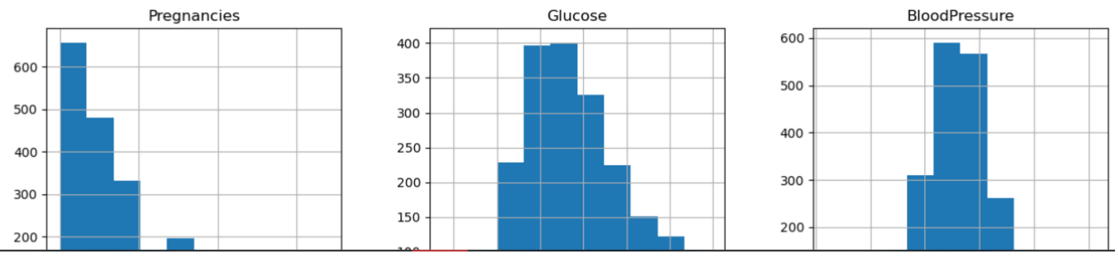


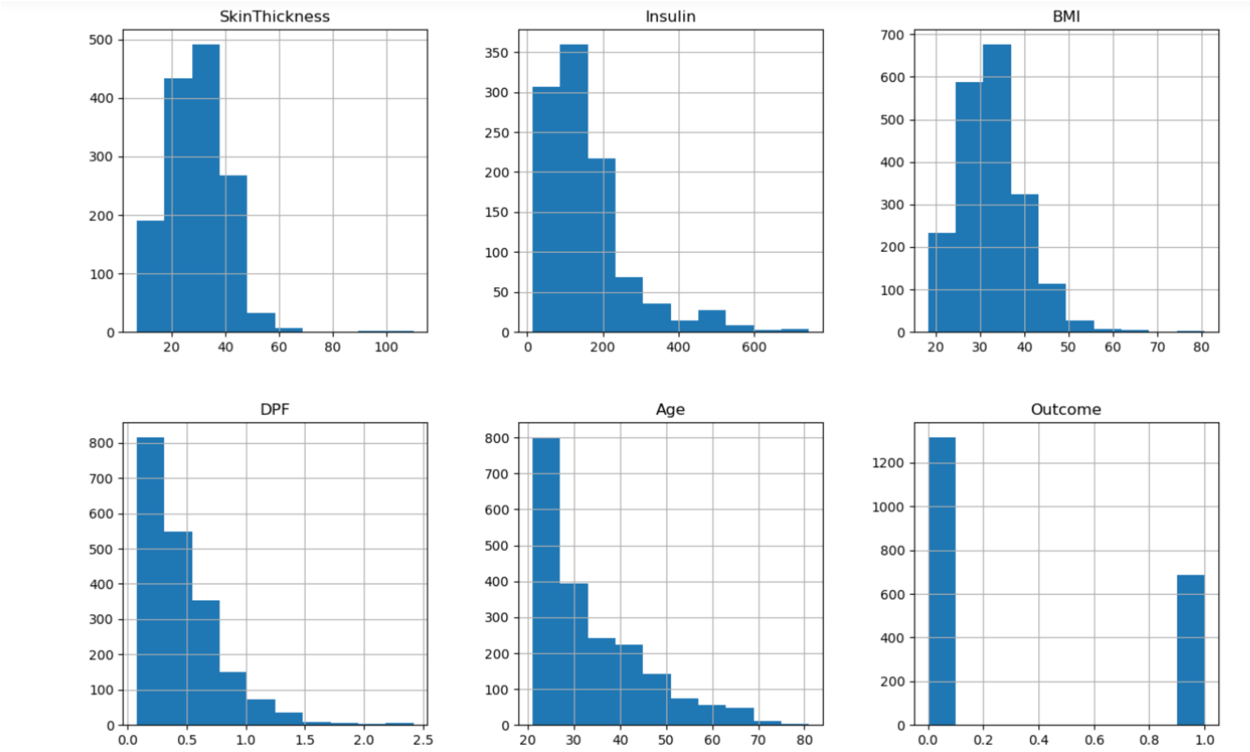
Data Cleaning

In [13]:
```
# Replacing the 0 values from ['Glucose','BloodPressure','SkinThickness','Insulin','BMI'] by NaN
df_copy = df.copy(deep=True)
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose','BloodPressure','SkinThickness','I
df_copy.isnull().sum()
```

Out[13]:
```
Pregnancies        0
Glucose           13
BloodPressure     90
SkinThickness    573
Insulin          956
BMI               28
DPF                0
Age                0
Outcome            0
dtype: int64
```

In [14]:
```
# To fill these Nan values the data distribution needs to be understood
# Plotting histogram of dataset before replacing NaN values
p = df_copy.hist(figsize = (15,15))
```



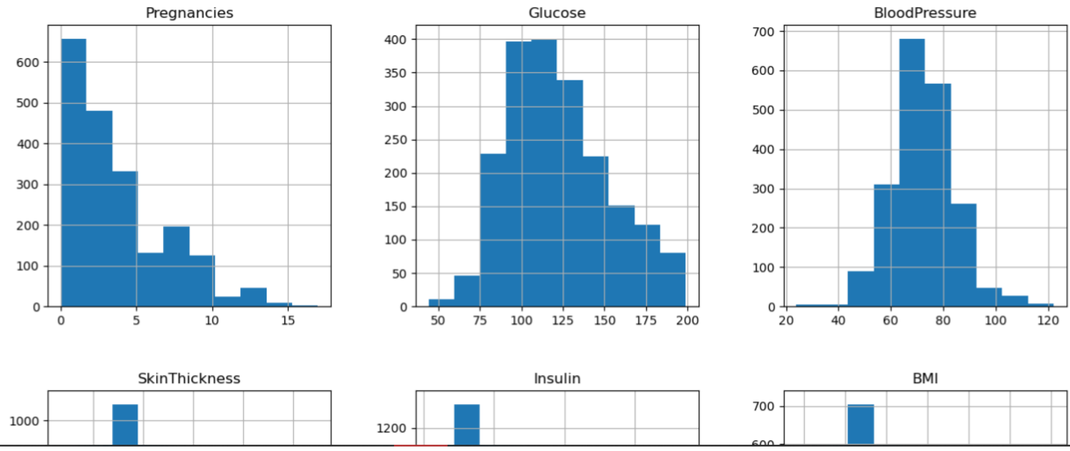


In [15]:

```
# Replacing NaN value by mean, median depending upon distribution
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace=True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace=True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace=True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace=True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace=True)
```

In [16]:

```
# Plotting histogram of dataset after replacing NaN values
p = df_copy.hist(figsize=(15,15))
```



Model Development

- Split the data into training and testing sets.
- Implement various classification algorithms (e.g., Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines).
- Evaluate the performance of these models using appropriate metrics (accuracy, precision, recall, F1 score).

Model Building

In [18]:

```
from sklearn.model_selection import train_test_split

X = df.drop(columns='Outcome')
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
print('X_train size: {}, X_test size: {}'.format(X_train.shape, X_test.shape))
```

```
X_train size: (1600, 8), X_test size: (400, 8)
```

In [19]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [20]:

```
# Using GridSearchCV to find the best algorithm for this problem
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [21]:
```python
# Creating a function to calculate best model for this problem
def find_best_model(X, y):
    models = {
        'logistic_regression': {
            'model': LogisticRegression(solver='lbfgs', multi_class='auto'),
            'parameters': {
                'C': [1,5,10]
              }
        },

        'decision_tree': {
            'model': DecisionTreeClassifier(splitter='best'),
            'parameters': {
                'criterion': ['gini', 'entropy'],
                'max_depth': [5,10]
            }
        },

        'random_forest': {
            'model': RandomForestClassifier(criterion='gini'),
            'parameters': {
                'n_estimators': [10,15,20,50,100,200]
            }
        },

        'svm': {
            'model': SVC(gamma='auto'),
            'parameters': {
                'C': [1,10,20],
                'kernel': ['rbf','linear']
            }
        }

    }

    scores = []
    cv_shuffle = ShuffleSplit(n_splits=5, test_size=0.20, random_state=0)

    for model_name, model_params in models.items():
        gs = GridSearchCV(model_params['model'], model_params['parameters'], cv = cv_shuffle, return_train_score=False)
        gs.fit(X, y)
        scores.append({
            'model': model_name,
            'best_parameters': gs.best_params_,
            'score': gs.best_score_
        })

    return pd.DataFrame(scores, columns=['model','best_parameters','score'])

find_best_model(X_train, y_train)
```

Out[21]:

	model	best_parameters	score
0	logistic_regression	{'C': 10}	0.763125
1	decision_tree	{'criterion': 'gini', 'max_depth': 10}	0.901250
2	random_forest	{'n_estimators': 100}	0.950000
3	svm	{'C': 20, 'kernel': 'rbf'}	0.869375

Note: Since the Random Forest algorithm has the highest accuracy, we futher fine tune the model using hyperparameter optimization.

In [22]:
```python
# Using cross_val_score for gaining average accuracy
from sklearn.model_selection import cross_val_score
scores = cross_val_score(RandomForestClassifier(n_estimators=20, random_state=0), X_train, y_train, cv=5)
print('Average Accuracy : {}%'.format(round(sum(scores)*100/len(scores)), 3))
```

```
Average Accuracy : 95%
```

Model Evaluation and Selection

- Perform cross-validation to ensure model robustness.
- Select the best-performing model based on evaluation metrics.

Note: Since the Random Forest algorithm has the highest accuracy, we futher fine tune the model using hyperparameter optimization.

[22]:
```python
# Using cross_val_score for gaining average accuracy
from sklearn.model_selection import cross_val_score
scores = cross_val_score(RandomForestClassifier(n_estimators=20, random_state=0), X_train, y_train, cv=5)
print('Average Accuracy : {}%'.format(round(sum(scores)*100/len(scores)), 3))
```

```
Average Accuracy : 95%
```

[23]:
```python
# Creating Random Forest Model
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train, y_train)
```

Out[23]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=20, random_state=0)
```

Model Evaluation

In [24]:
```python
# Creating a confusion matrix
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[24]:
```
array([[272,   0],
       [  5, 123]], dtype=int64)
```

In [25]:
```python
# Plotting the confusion matrix
plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Random Forest Classifier Model - Test Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

In [26]:
```
# Accuracy Score
score = round(accuracy_score(y_test, y_pred),4)*100
print("Accuracy on test set: {}%".format(score))
```

```
Accuracy on test set: 98.75%
```

In [27]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       272
           1       1.00      0.96      0.98       128

    accuracy                           0.99       400
   macro avg       0.99      0.98      0.99       400
weighted avg       0.99      0.99      0.99       400
```

In [28]:
```
# Creating a confusion matrix for training set
y_train_pred = classifier.predict(X_train)
cm = confusion_matrix(y_train, y_train_pred)
cm
```

Out[28]:
```
array([[1044,    0],
       [   1,  555]], dtype=int64)
```

In [30]:
```
# Accuracy Score
score = round(accuracy_score(y_train, y_train_pred),4)*100
print("Accuracy on trainning set: {}%".format(score))
```

```
Accuracy on trainning set: 99.94%
```

In [31]:
```
# Classification Report
print(classification_report(y_train, y_train_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1044
           1       1.00      1.00      1.00       556

    accuracy                           1.00      1600
   macro avg       1.00      1.00      1.00      1600
weighted avg       1.00      1.00      1.00      1600
```

PREDICTION

- Developing a function for prediction
- Manual Prediction test to ensure the successful prediction of the model

Predictions

In [32]:
```
# Creating a function for prediction
def predict_diabetes(Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age):
    preg = int(Pregnancies)
    glucose = float(Glucose)
    bp = float(BloodPressure)
    st = float(SkinThickness)
    insulin = float(Insulin)
    bmi = float(BMI)
    dpf = float(DPF)
    age = int(Age)

    x = [[preg, glucose, bp, st, insulin, bmi, dpf, age]]
    x = sc.transform(x)

    return classifier.predict(x)
```

In [33]:
```
# Prediction 1
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age
prediction = predict_diabetes(0, 50, 140, 19, 58, 26, 0.444, 40)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Great! You don't have diabetes.
  warnings.warn(
```

In [34]:
```
# Prediction 2
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age
prediction = predict_diabetes(1, 117, 88, 24, 145, 34.5, 0.403, 40)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Oops! You have diabetes.
```

```
C:\Users\ADT\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have val
Scaler was fitted with feature names
  warnings.warn(
```

In [35]:
```
# Prediction 2
# Input sequence: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DPF, Age
prediction = predict_diabetes(5, 120, 92, 10, 81, 26.1, 0.551, 67)[0]
if prediction:
  print('Oops! You have diabetes.')
else:
  print("Great! You don't have diabetes.")
```

```
Great! You don't have diabetes.
```

FRONT END

- Saving the Model to be used for Model Deployment.
- Next step will be the front end of the project

```
classifier.fit(X_train, y_train)

# Creating a pickle file for the classifier
filename = 'diabetes-prediction-rfc-model.pkl'
pickle.dump(classifier, open(filename, 'wb'))
```

Model Deployment:

- Develop a Flask web application to provide an interface for users to input their medical data and receive predictions.
- Ensure the web application is user-friendly and includes necessary instructions for users.

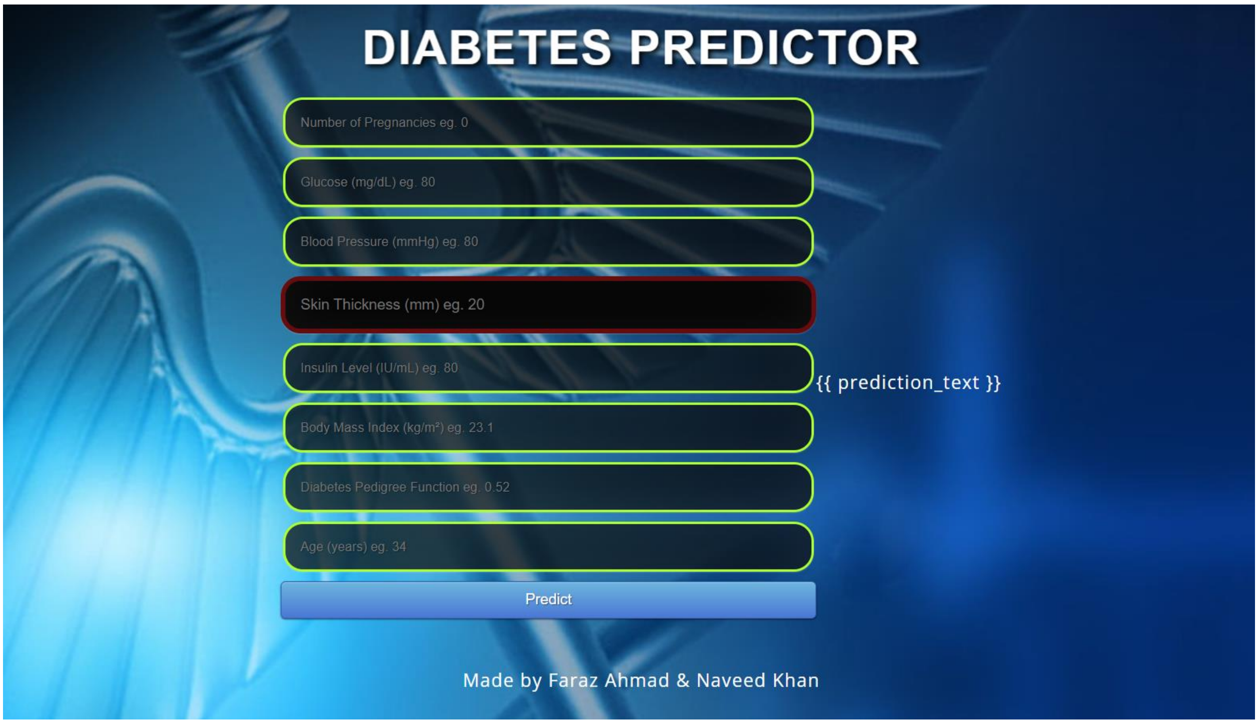
App.py

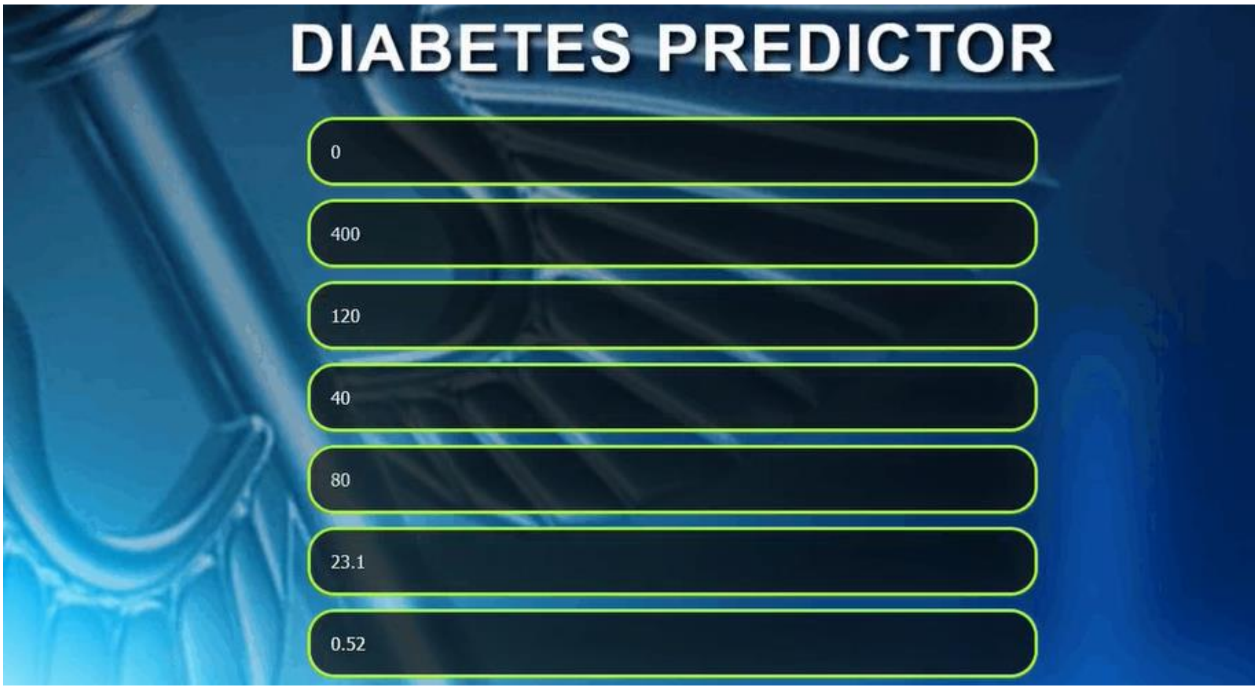
```python

from flask import Flask, render_template, request
import pickle
import numpy as np
filename = 'diabetes-prediction-rfc-model.pkl'
classifier = pickle.load(open(filename, 'rb'))

app = Flask(__name__)

@app.route('/')
def home():
	return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        preg = int(request.form['pregnancies'])
        glucose = int(request.form['glucose'])
        bp = int(request.form['bloodpressure'])
        st = int(request.form['skinthickness'])
        insulin = int(request.form['insulin'])
        bmi = float(request.form['bmi'])
        dpf = float(request.form['dpf'])
        age = int(request.form['age'])

        data = np.array([[preg, glucose, bp, st, insulin, bmi, dpf, age]])
        my_prediction = classifier.predict(data)

        return render_template('result.html', prediction=my_prediction)

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

User Interface

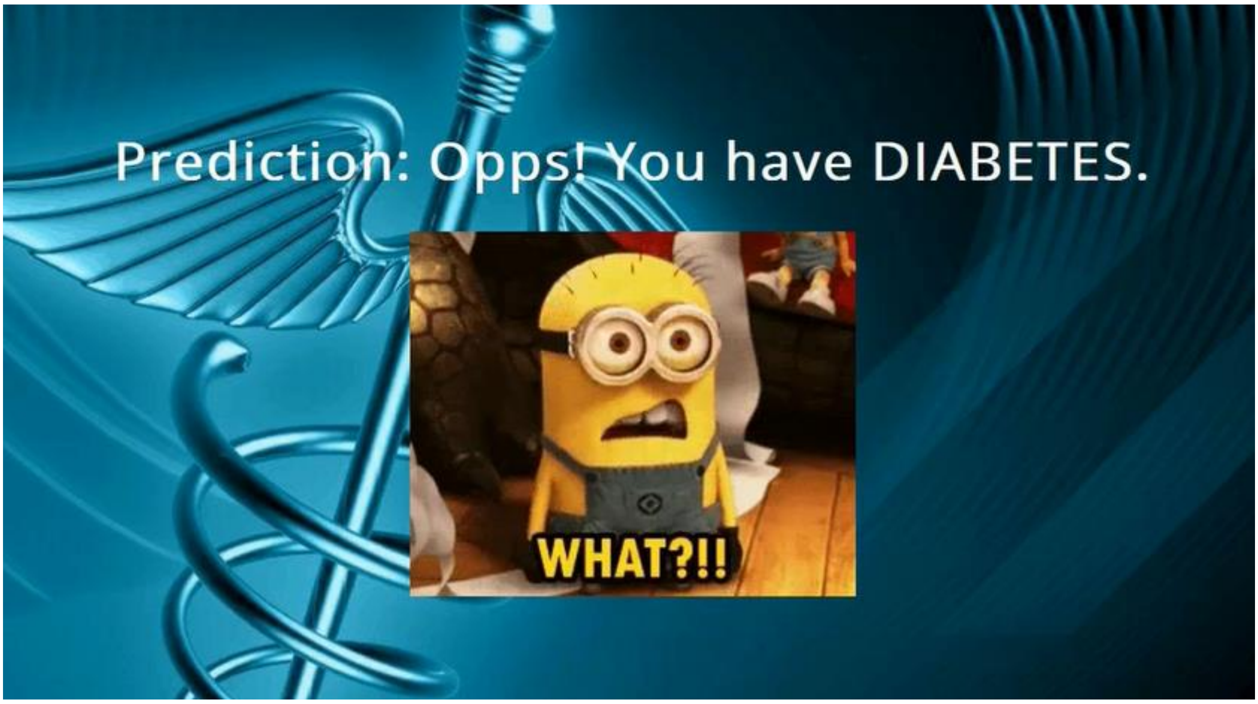
- **Input Form:** Form to enter patient data.
- **Prediction Output:** Display the prediction result (diabetic or non-diabetic) along with confidence scores.

Index.html


DIABETES PREDICTOR
Number of Pregnancies eg. 0
Glucose (mg/dL) eg. 80
Blood Pressure (mmHg) eg. 80
Skin Thickness (mm) eg. 20
Insulin Level (IU/mL) eg. 80
{{ prediction_text }}
Body Mass Index (kg/m²) eg. 23.1
Diabetes Pedigree Function eg. 0.52
Age (years) eg. 34
Predict
Made by Faraz Ahmad & Naveed Khan


DIABETES PREDICTOR
0
400
120
40
80
23.1
0.52


Prediction: Hurrah !!! You DON'T have diabetes.
IT'S TIME TO PARTYYYYYYY !


Prediction: Opps! You have DIABETES.
WHAT?!!

Tools and Technologies

- **Programming Language:** Python
- **Libraries:** Pandas, NumPy, Scikit-learn, GridsearchCV, Sufflesplit ,Matplotlib, Seaborn, Flask
- **ML Libraries:** Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Support Vector Machine
- **IDE:** Jupyter Notebook, Visual Studio Code
- **Version Control:** Git

Outcomes

- An accurate and robust machine learning model for diabetes classification.
- A fully functional Flask web application for diabetes prediction.

Resources

- **Hardware:** Standard development laptop or desktop.
- **Software:** Open-source libraries and tools.

Conclusion

This project aims to leverage the power of machine learning to aid in the early detection of diabetes. By deploying a web application, the project ensures accessibility and usability for a broader audience, thereby contributing to better health outcomes.