Superior University – Gold Campus

Diabetes Prediction System

Developed By:

Name: Sadia Eman Abdul Ghafoor

Roll Number: SU92_F23-BSDSM_019

Section: BSDS_3A

Submitted To:

Teacher: Sir Rashik Ali

Institution:

Superior University – Gold Campus

Date of Submission: 30-Nov -2024

Table of Contents:

1.Introduction:

This project is a web-based **Diabetes Prediction System** designed to classify individuals as diabetic or non-diabetic based on their medical data. It uses a machine learning model integrated with a user-friendly web interface built using **Flask**.

The primary purpose of this project is to assist users with an initial diagnosis by predicting diabetes risk using common medical metrics.

---

2. Objectives:

- To build a lightweight, accurate system for diabetes prediction.
- To integrate machine learning with web technology for a seamless user experience.
- To develop a responsive and visually appealing user interface.
- To deploy a functional model for real-time predictions.

---

3. Setup and Tools

Tools Used:

Tool	Purpose
Python 3.8.8	Programming language for development.
Flask Framework	For building the web application and handling HTTP requests.
PyCharm IDE	Integrated Development Environment for writing and testing the application.
Jupyter Notebook	Used for model training and experimentation.
Bootstrap	For modern UI design and responsiveness.

Setup Instructions:

1. Install Python v3.8.8:

   Download from [Python.org](https://www.python.org).
2. Install Required Libraries:
3. Save the Machine Learning Model:

   the model file ***(model_joblib_diabetes)*** is in the project directory.
4. Run the Flask Application
5. Use **PyCharm** to debug and execute Flask applications efficiently.

---

4. Libraries and Technologies

Library	Purpose
Flask	Web framework for building dynamic applications.
pandas	Data manipulation for managing input and prediction data.
joblib	For saving and loading trained machine learning models.
scikit-learn	Machine learning library for model training and evaluation.
Bootstrap	Provides modern, responsive, and attractive UI components.

---

5. Dataset Overview

- **Dataset Name:** *PIMA Indian Diabetes Dataset*
- **Description:**

  The dataset includes medical diagnostic data for predicting diabetes. It contains 8 features (e.g., glucose, BMI) and a target column indicating whether the person is diabetic (1) or not (0).

Feature	Description
Pregnancies	Number of pregnancies.
Glucose	Plasma glucose concentration.
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.
Diabetes Pedigree Function	Genetic history of diabetes.
Age	Patient age in years.

6. Model Development

- **Model Used:** Logistic Regression.
- **Training Steps:**

  1. Load and clean the dataset.
  2. Handle missing values and normalize data.
  3. Split the data into 80% training and 20% testing subsets.
  4. Train a logistic regression model and evaluate performance using metrics like accuracy and F1 score.

- **Accuracy Achieved:** 78%.

7. Application Workflow

How it Works:

1. **User Inputs:**

   Users input medical data via the web form (e.g., glucose levels, BMI, age).

2. **Model Processing:**

   The input data is passed to a trained machine learning model for predictions.

3. **Result Display:**

   The model predicts the outcome (`Diabetic` or `Non-Diabetic`), which is displayed on a result page.

8.Key Code Sections

Libraries

```
import pandas as pd
```

```
data = pd.read_csv('diabetes.csv')
```

1. Display Top 5 Rows of The Dataset

```
data.head()
```

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

2. Check Last 5 Rows of The Dataset

```
data.tail()
```

	Unnamed: 0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	763	10	101	76	48	180	32.9	0.171	63	0
764	764	2	122	70	27	0	36.8	0.340	27	0
765	765	5	121	72	23	112	26.2	0.245	30	0
766	766	1	126	60	0	0	30.1	0.349	47	1
767	767	1	93	70	31	0	30.4	0.315	23	0

3. Find Shape of Our Dataset (Number of Rows And Number of Columns)

]:
```
data.shape
```

]:
```
(768, 10)
```

]:
```
print("Number of Rows",data.shape[0])
print("Number of Columns",data.shape[1])
```

```
Number of Rows 768
Number of Columns 10
```

4. Get Information About Our Dataset Like Total Number Rows, Total Number of Columns, Datatypes of Each Column And Memory Requirement

]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Unnamed: 0                768 non-null    int64  
 1   Pregnancies               768 non-null    int64  
 2   Glucose                   768 non-null    int64  
 3   BloodPressure             768 non-null    int64  
 4   SkinThickness             768 non-null    int64  
 5   Insulin                   768 non-null    int64  
 6   BMI                       768 non-null    float64
 7   DiabetesPedigreeFunction  768 non-null    float64
 8   Age                       768 non-null    int64  
 9   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(8)
memory usage: 60.1 KB
```

5. Check Null Values In The Dataset

]:
```
data.isnull().sum()
```

]:
```
Unnamed: 0                  0
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

6. Get Overall Statistics About The Dataset

```
data.describe()
```

	Unnamed: 0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	383.500000	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876
std	221.846794	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000
25%	191.750000	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750
50%	383.500000	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500
75%	575.250000	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250
max	767.000000	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000

```
import numpy as np
```

```
data_copy = data.copy(deep=True)
```

```
data.columns
```

```
Index(['Unnamed: 0', 'Pregnancies', 'Glucose', 'BloodPressure',
       'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',
       'Outcome'],
      dtype='object')
```

```
data_copy[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI']] = data_copy[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI']].replace(0,np.nan)
```

```
data_copy.isnull().sum()
```

```
Unnamed: 0                    0
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

```
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
```

7. Store Feature Matrix In X and Response(Target) In Vector y

```
X = data.drop('Outcome',axis=1)
y = data['Outcome']
```

8. Splitting The Dataset Into The Training Set And Test Set

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,
                                               random_state=42)
```

9. Scikit-Learn Pipeline

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.pipeline import Pipeline
```

```
pipeline_lr  = Pipeline([('scalar1',StandardScaler()),
                         ('lr_classifier',LogisticRegression())])

pipeline_knn = Pipeline([('scalar2',StandardScaler()),
                          ('knn_classifier',KNeighborsClassifier())])

pipeline_svc = Pipeline([('scalar3',StandardScaler()),
                         ('svc_classifier',SVC())])

pipeline_dt = Pipeline([('dt_classifier',DecisionTreeClassifier())])
pipeline_rf = Pipeline([('rf_classifier',RandomForestClassifier(max_depth=3))])
pipeline_gbc = Pipeline([('gbc_classifier',GradientBoostingClassifier())])
```

```
pipelines = [pipeline_lr,
             pipeline_knn,
             pipeline_svc,
             pipeline_dt,
             pipeline_rf,
             pipeline_gbc]
```

```
pipelines
```

```
[Pipeline(steps=[('scalar1', StandardScaler()),
                 ('lr_classifier', LogisticRegression())]),
 Pipeline(steps=[('scalar2', StandardScaler()),
                 ('knn_classifier', KNeighborsClassifier())]),
 Pipeline(steps=[('scalar3', StandardScaler()), ('svc_classifier', SVC())]),
 Pipeline(steps=[('dt_classifier', DecisionTreeClassifier())]),
 Pipeline(steps=[('rf_classifier', RandomForestClassifier(max_depth=3))]),
 Pipeline(steps=[('gbc_classifier', GradientBoostingClassifier())])]
```

```
for pipe in pipelines:
    pipe.fit(X_train,y_train)
```

```
pipe_dict = {0:'LR',
             1:'KNN',
             2:'SVC',
             3:'DT',
             4: 'RF',
             5: 'GBC'}
```

```
pipe_dict
```

```
{0: 'LR', 1: 'KNN', 2: 'SVC', 3: 'DT', 4: 'RF', 5: 'GBC'}
```

```
for i,model in enumerate(pipelines):
    print("{} Test Accuracy:{}".format(pipe_dict[i],model.score(X_test,y_test)*100))
```

```
LR Test Accuracy:76.62337662337663
KNN Test Accuracy:68.83116883116884
SVC Test Accuracy:73.37662337662337
DT Test Accuracy:72.07792207792207
RF Test Accuracy:76.62337662337663
GBC Test Accuracy:75.97402597402598
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
X = data.drop('Outcome',axis=1)
y = data['Outcome']
```

```
rf =RandomForestClassifier(max_depth=3)
```

```
rf.fit(X,y)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=3)
```

Prediction on New Data

```
data = pd.read_csv('diabetes.csv').drop(columns=['Unnamed: 0'], errors='ignore')
data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = data[['Glucose', 'BloodPressure
data.fillna(data.mean(), inplace=True)
X = data.drop('Outcome', axis=1)
y = data['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
rf = RandomForestClassifier(max_depth=3, random_state=42)
rf.fit(X_train, y_train)
new_data = pd.DataFrame({
    'Pregnancies': [6],
    'Glucose': [148.0],
    'BloodPressure': [72.0],
    'SkinThickness': [35.0],
    'Insulin': [79.799479],
    'BMI': [33.6],
    'DiabetesPedigreeFunction': [0.627],
    'Age': [50]
})
new_data = new_data[X.columns]
p = rf.predict(new_data)
print('Prediction for new data:', 'Diabetic' if p[0] == 1 else 'Non-Diabetic')
```

```
Prediction for new data: Diabetic
```

18. Save Model Using Joblib

```
]: import joblib
```

```
]: joblib.dump(rf,'model_joblib_diabetes')
```

```
]: ['model_joblib_diabetes']
```

```
]: model = joblib.load('model_joblib_diabetes')
```

```
]: model.predict(new_data)
```

```
]: array([1], dtype=int64)
```

Flask App Code:

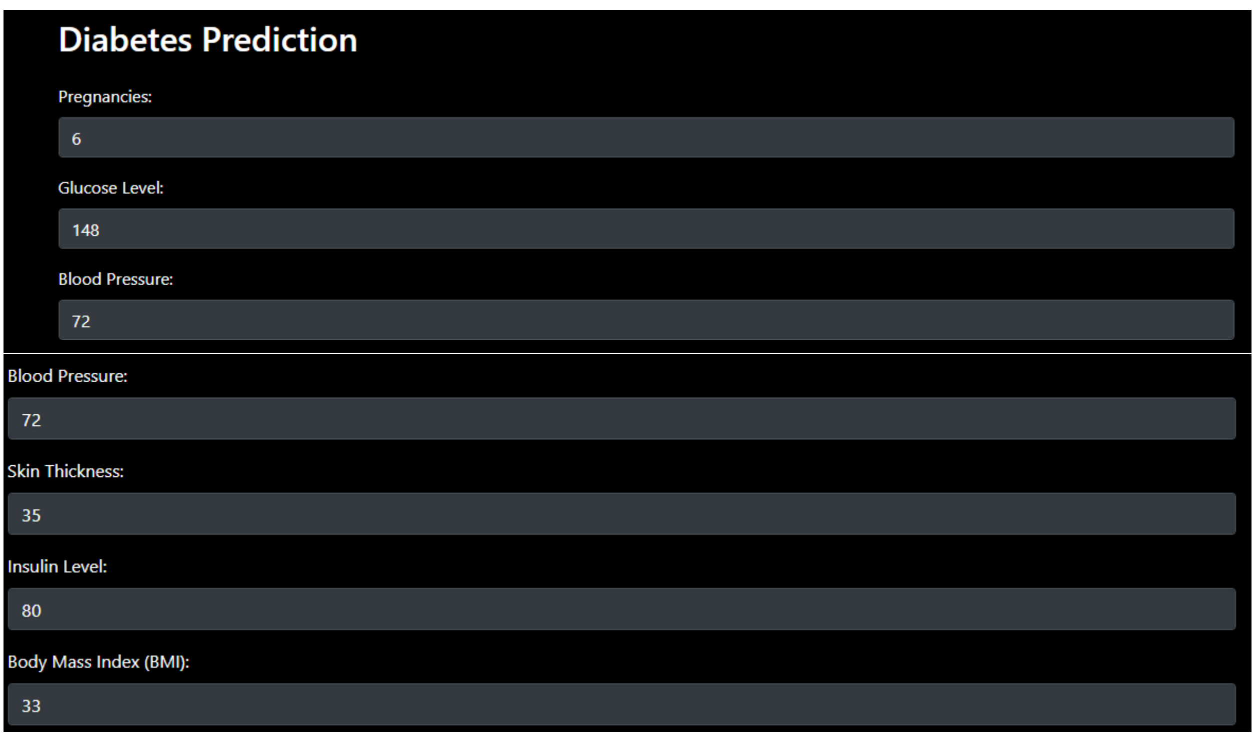
```
from flask import Flask, render_template, request
import pandas as pd
import joblib

app = Flask(__name__)

# Load the model
model = joblib.load('model_joblib_diabetes')

@app.route('/')


def home():
    return render_template('index.html')

1 usage (1 dynamic)
@app.route( rule: '/predict', methods=['POST'])
def predict():
# Get data from form
pregnancies = float(request.form['pregnancies'])
glucose = float(request.form['glucose'])
blood_pressure = float(request.form['bloodpressure'])  # Corrected
skin_thickness = float(request.form['skinthickness'])  # Corrected
insulin = float(request.form['insulin'])
bmi = float(request.form['bmi'])
pedigree_function = float(request.form['diabetespedigree'])  # Corrected
age = float(request.form['age'])
```

```python
    # Prepare the input data for prediction
    input_data = pd.DataFrame([[pregnancies, glucose, blood_pressure, skin_thickness,
                                insulin, bmi, pedigree_function, age]],
                              columns=['Pregnancies', 'Glucose', 'BloodPressure',
                                       'SkinThickness', 'Insulin', 'BMI',
                                       'DiabetesPedigreeFunction', 'Age'])

    # Make the prediction
    prediction = model.predict(input_data)

    result = 'Diabetic' if prediction[0] == 1 else 'Non-Diabetic'
    return render_template('result.html', prediction=result)

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

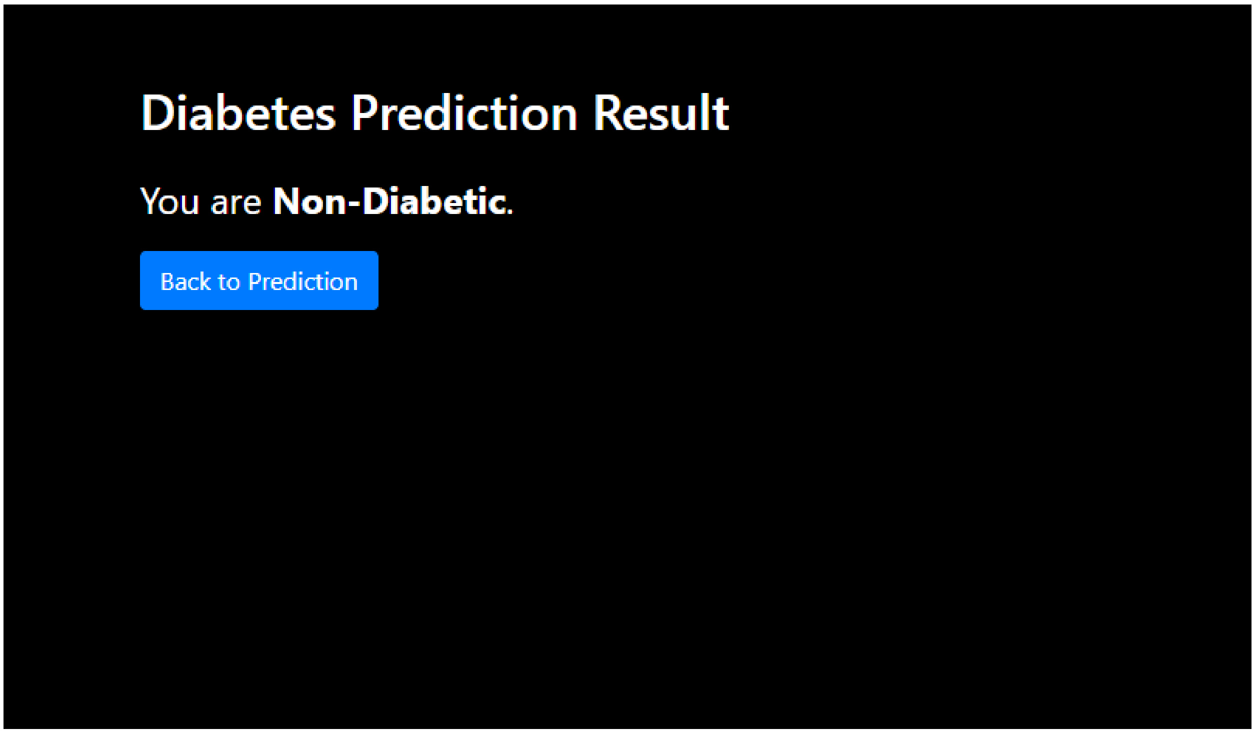
---

9. Screenshots

Home Page:



Result Page:



---

10. Challenges and Solutions

Challenge	Solution
Flask not running in Jupyter Notebook.	Used PyCharm for Flask development.
Model integration issues.	Ensured proper data preprocessing and column alignment during prediction.

User input validation.	Added required attributes and handled input in Python.

---

11. Conclusion

This project demonstrates how machine learning models can be effectively integrated into web applications. The **Diabetes Prediction System** offers a practical, easy-to-use tool for preliminary diabetes risk assessment.