

Diabetes Prediction Using Machine Learning

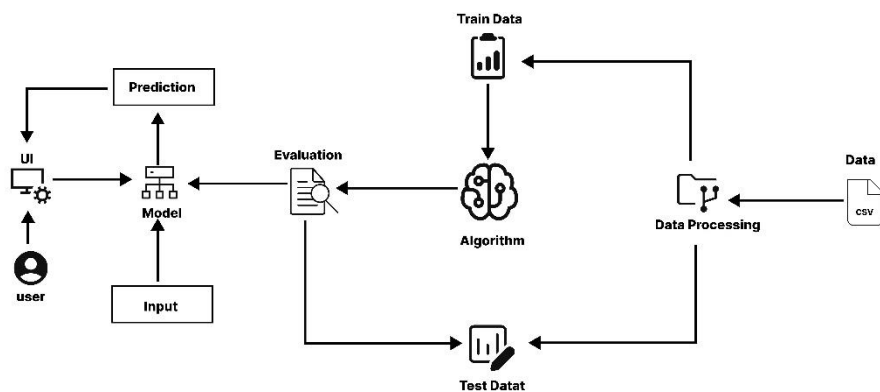
Diabetes Prediction Using Machine Learning

In this project, we aim to use machine learning algorithms to predict the onset of diabetes in individuals based on their health records and other relevant factors such as age, BMI, family history, and lifestyle habits. The dataset used in this project will include information on various clinical parameters such as blood pressure, BMI, Heart diseases and cholesterol levels.

Our goal is to develop a predictive model that can accurately identify individuals who are at high risk of developing diabetes, thereby allowing for early intervention and prevention of the disease. By using machine learning techniques to analyse large amounts of data, we can identify patterns and make accurate predictions that could potentially save lives.

Overall, this project has the potential to contribute to the field of healthcare by improving early detection and prevention of diabetes, ultimately leading to better health outcomes for individuals and communities.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we must complete all the activities listed below,

- Define Problem / Problem Understanding
 - Specify the business problem
 - Business requirements
 - Literature Survey
 - Social or Business Impact.
- Data Collection & Preparation
 - Collect the dataset
 - Data Preparation
- Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
- Model Building
 - Training the model in multiple algorithms
 - Testing the model
- Performance Testing
 - Testing model with multiple evaluation metrics
- Model Deployment
 - Save the best model
 - Integrate with Web Framework

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

The business problem addressed in this project is the early detection and prediction of diabetes using machine learning algorithms. The goal is to develop a predictive model that can accurately identify individuals at high risk of developing diabetes based on their health records and other relevant factors. Early detection and management of diabetes can improve healthcare outcomes, reduce costs, and benefit healthcare providers and insurance companies. Therefore, developing an accurate and reliable predictive model for diabetes detection can have a significant impact on healthcare outcomes and costs.

Activity 2: Business requirements

Business requirements are the specific needs and expectations of the business stakeholders regarding the desired outcome of the project. In the case of the diabetes prediction project, the following are the key business requirements:

- **Accurate prediction:** The predictive model should be accurate in identifying individuals who are at high risk of developing diabetes based on their health records and other relevant factors.
- **Efficiency:** The model should be efficient and fast in analyzing large amounts of data to provide timely predictions.
- **Scalability:** The model should be scalable to handle large datasets and accommodate future growth in data volume.
- **Flexibility:** The model should be flexible and adaptable to accommodate changes in data sources or input parameters.
- **User-friendliness:** The model should be user-friendly, easy to use, and understand by healthcare providers and insurance companies.
- **Integration:** The model should be easily integrated with existing healthcare systems and processes.
- **Security:** The model should be secure and protect patient data privacy.
- **Compliance:** The model should comply with relevant healthcare regulations and standards.

Activity 3: Literature Survey

A literature survey is an essential step in any diabetes prediction using machine learning:

- "Machine Learning for Diabetes Prediction: A Review" by E. Şahin et al. This paper provides a comprehensive review of the latest research on machine learning for diabetes prediction. The authors discuss the challenges, approaches, and evaluation metrics used in various studies.
- "Predicting Type 2 Diabetes Mellitus Using Machine Learning Techniques" by S. Chakraborty et al. This paper proposes a machine learning approach for predicting the risk of developing type 2 diabetes mellitus based on demographic and clinical data. The

authors compare various algorithms and feature selection techniques and evaluate their performance.

- "Deep Learning for Diabetes Prediction: A Review" by Y. Zhao et al. This paper provides a review of the latest research on deep learning for diabetes prediction, with a focus on the use of convolutional neural networks (CNNs) and recurrent neural networks (RNNs).
- "Diabetes Prediction Using Machine Learning Techniques: A Comparative Study" by S. B. Gawali and R. K. Kamat. This paper compares the performance of various machine learning algorithms, including logistic regression, decision trees, and neural networks, for diabetes prediction using clinical and demographic data.
- "Machine Learning for Early Detection of Diabetic Retinopathy" by A. Gulshan et al. This paper proposes a deep learning approach for the early detection of diabetic retinopathy based on retinal images. The authors use a CNN to classify images into different stages of the disease and achieve high accuracy.

Activity 4: Social or Business Impact

Accurate diabetes prediction using machine learning can have a significant social and business impact. It can help identify individuals at high risk of developing diabetes, leading to earlier intervention and prevention efforts. From a business perspective, accurate prediction can help healthcare providers and insurers manage healthcare costs and resources better. Additionally, it can lead to more personalized healthcare, improving patient outcomes and adherence to treatment plans. In conclusion, accurate diabetes prediction using machine learning can improve patient outcomes, reduce healthcare costs, and lead to more personalized healthcare.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
In [1]: # Importing essential libraries
import numpy as np
import pandas as pd
import pickle
import matplotlib.pyplot as plt
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

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

```
In [3]: df.head()
```

```
Out[3]:
```

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

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

- Let's know the info and describe of our dataset first. To find the shape of our data, the `df.shape` method is used. To find the data type, `df.info()` function is used

```
In [4]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  -
 0   Pregnancies           768 non-null   int64  
 1   Glucose               768 non-null   int64  
 2   BloodPressure         768 non-null   int64  
 3   SkinThickness         768 non-null   int64  
 4   Insulin               768 non-null   int64  
 5   BMI                  768 non-null   float64 
 6   DiabetesPedigreeFunction 768 non-null   float64 
 7   Age                  768 non-null   int64  
 8   Outcome              768 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

- For checking the null values, `df.isnull()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
In [5]: df.isnull()
```

```
Out[5]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False
767	False	False	False	False	False	False	False	False	False

768 rows × 9 columns

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data, we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using Label encoding with the help of list comprehension.

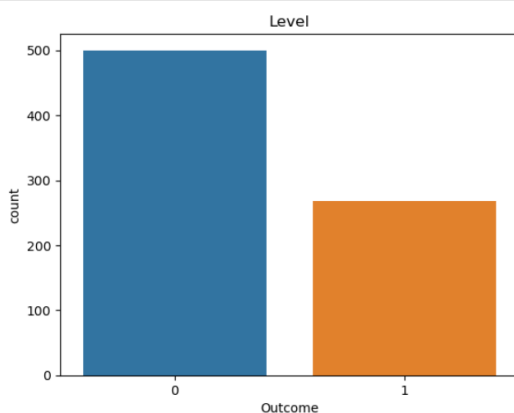
- In our project, categorical features are in many columns Label encoding is done

Activity 2.3: Handling Imbalance Data

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of some columns feature with some mathematical formula

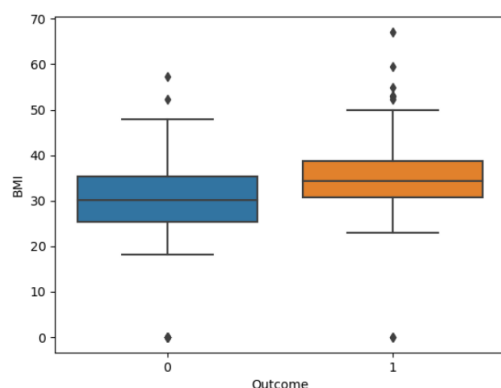
- From the below diagram, we could visualize that some of the feature has outliers Boxplot from seaborn library is used here.

```
In [8]: import seaborn as sns
sns.countplot(x='Outcome', data=df)
plt.title('Level')
plt.show()
```



```
In [9]: sns.boxplot(data = df, x = "Outcome", y = "BMI")
```

```
Out[9]: <Axes: xlabel='Outcome', ylabel='BMI'>
```



Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features

```
In [7]: df.describe()
```

```
Out[7]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as dist. plot and count plot.

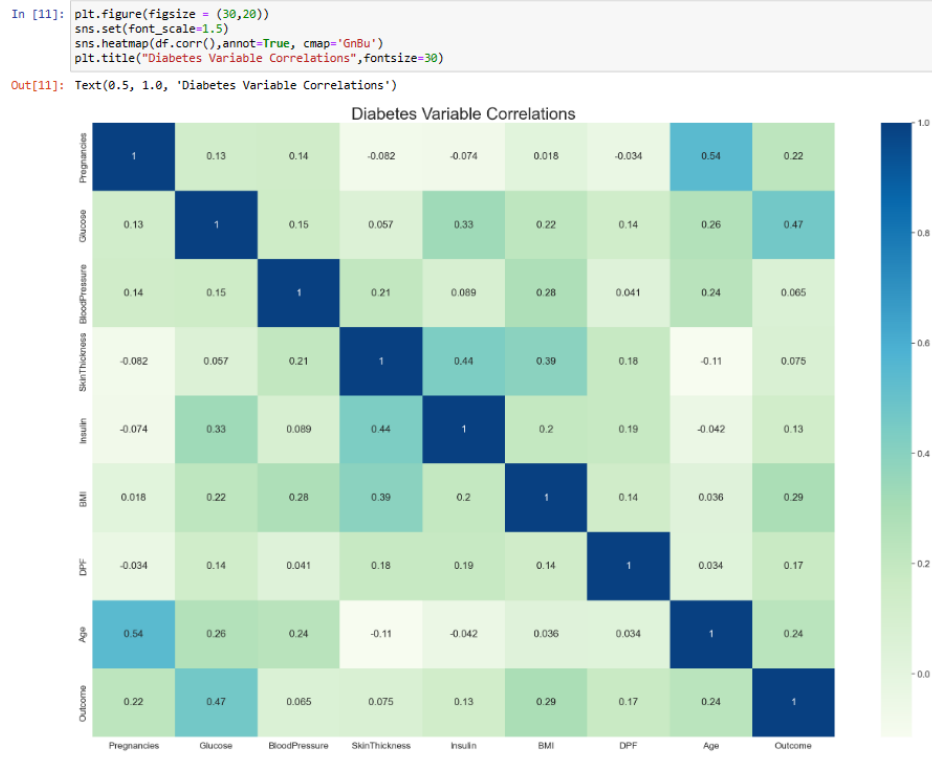
In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features

Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualizing. Count. plot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package



Applying PCA in a Machine Learning Pipeline for Diabetes Prediction:

Applying PCA (Principal Component Analysis) in a pipeline that includes hyperparameter tuning using GridSearchCV, data preprocessing using Standard Scaler, and applying a classifier can improve the performance of a machine learning model for cost prediction by reducing the dimensionality of the data, optimizing the hyperparameters of the classifier, and improving the accuracy of the predictions. StandardScaler scales the data, while PCA reduces dimensionality by identifying the most important features in the data. GridSearchCV helps to optimize the hyperparameters of the classifier, and finally, a suitable classifier is applied to the preprocessed and dimensionality-reduced data to evaluate the performance using appropriate metrics.

Splitting data into train and test:

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
In [14]: # Model Building
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)
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project, we are applying three classification algorithms. The best model is saved based on its performance.

Activity 1.1: Random Forest Regressor

A function named random forest regressor is created and train and test data are passed as the parameters. Inside the function, random forest regressor algorithm is initialized and training data is passed to the model with the fit() function. Test data is predicted with predict () function and saved in a new variable. For evaluating the model with R2_score.

```
In [36]: rf = RandomForestClassifier(max_depth=13, criterion='gini', n_estimators =200, min_samples_leaf=5, random_state=42)
         rf.fit(X_train, Y_train)
```

```
Out[36]:
RandomForestClassifier
RandomForestClassifier(max_depth=13, min_samples_leaf=5, n_estimators=200,
                        random_state=42)
```

```
In [37]: # Make predictions on test set
         y_pred=rf.predict(X_test)

         print('Training set score: {:.4f}'.format(rf.score(X_train, Y_train)))

         print('Test set score: {:.4f}'.format(rf.score(X_test, Y_test)))

Training set score: 0.8693
Test set score: 0.7329
```

```
In [38]: # Check MSE & RMSE
         mse =mean_squared_error(Y_test, y_pred)
         print('Mean Squared Error : '+ str(mse))
         rmse = math.sqrt(mean_squared_error(Y_test, y_pred))
         print('Root Mean Squared Error : '+ str(rmse))

Mean Squared Error : 0.2670807453416149
Root Mean Squared Error : 0.5167985539275578
```

Activity 1.2: Decision Tree Regressor

A function named decision Tree regressor is created and train and test data are passed as the parameters. Inside the function, decision Tree regressor algorithm is initialized and training data is passed to the model with fit() function. Test data is predicted with predict () function and saved in a new variable. For evaluating the model, For evaluating the model with R2_score

Decision Tree

```
In [31]: dt = DecisionTreeClassifier(criterion='gini',max_depth=13,min_samples_leaf=5)
dt.fit(X_train,Y_train)
```

```
Out[31]: ▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=13, min_samples_leaf=5)
```

```
In [32]: # Make predictions on test data
y_pred=dt.predict(X_test)
print('Training set score: {:.4f}'.format(dt.score(X_train,Y_train)))

print('Test set score: {:.4f}'.format(dt.score(X_train,Y_train)))

Training set score: 0.8667
Test set score: 0.8667
```

```
In [33]: # Check MSE and RSME
mse=mean_squared_error(Y_test,y_pred)
print('Mean Squared Error : '+str(mse))

rmse=math.sqrt(mse)
print('Mean Squared Error : '+str(rmse))

Mean Squared Error : 0.34782608695652173
Mean Squared Error :0.5897678246195885
```

Activity 1.3: KNeighborsClassifier

To evaluate the performance of a KNeighborsClassifier model, we need to test it on a separate dataset that it has not seen during training. This separate dataset is called the test data. We typically split the available data into two parts, the training data and the test data. The model is trained on the training data, and then tested on the test data to see how well it generalizes to new, unseen data.

```
In [25]: knn = KNeighborsClassifier(algorithm='auto',n_neighbors=13, weights='uniform')
knn.fit(X_train , Y_train)
```

```
Out[25]: ▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=13)
```

```
In [26]: # make predictions on test set
y_pred=knn.predict(X_test)

print('Training set score: {:.4f}'.format(knn.score(X_train, Y_train)))

print('Test set score: {:.4f}'.format(knn.score(X_test, Y_test)))

Training set score: 0.7600
Test set score: 0.6708
```

```
In [27]: from sklearn.metrics import mean_squared_error
import math # Import math module if not already imported

# Assuming Y_test and y_pred are defined
mse = mean_squared_error(Y_test, y_pred)
print('Mean Squared Error: ' + str(mse))

rmse = math.sqrt(mse)
print('Root Mean Squared Error: ' + str(rmse))

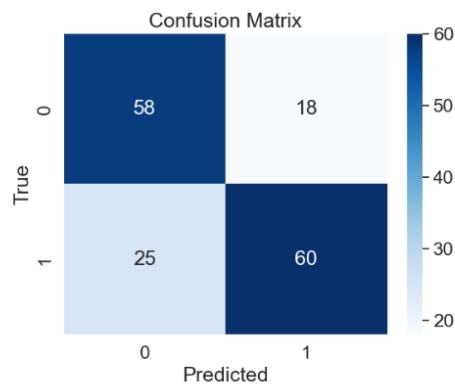
Mean Squared Error: 0.32919254658385094
Root Mean Squared Error: 0.5737530362306164
```

Milestone 5: Performance Testing

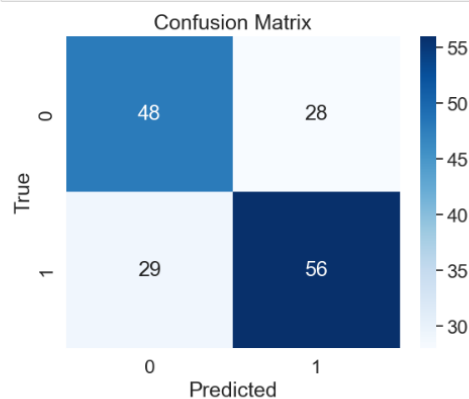
Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

```
In [39]: def plot_confusion_matrix(y_true, y_pred):  
cm = confusion_matrix(y_true, y_pred)  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True)  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion Matrix')  
plt.show()  
  
# Assuming Y_test and y_pred are defined  
plot_confusion_matrix(Y_test, y_pred)
```



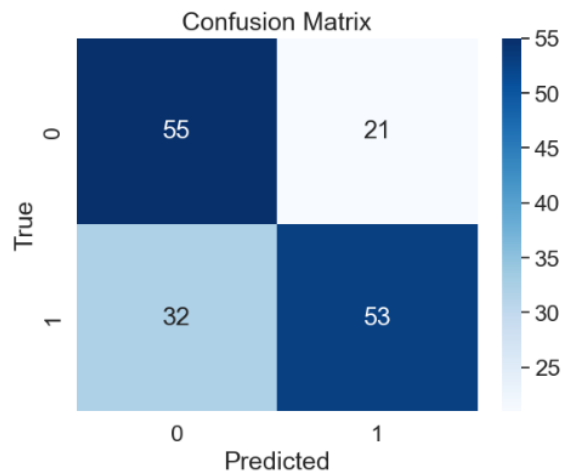
```
In [34]: import seaborn as sns  
from sklearn.metrics import confusion_matrix  
import matplotlib.pyplot as plt  
  
def plot_confusion_matrix(y_true, y_pred):  
cm = confusion_matrix(y_true, y_pred)  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True)  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion Matrix')  
plt.show()  
  
# Assuming Y_test and y_pred are defined  
plot_confusion_matrix(Y_test, y_pred)
```



```
In [29]: import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()

# Assuming Y_test and y_pred are defined
plot_confusion_matrix(Y_test, y_pred)
```



Activity 1.1: Compare the model

For comparing the below two models, with their R_2 score on training and testing data. the results of models are displayed as output. From the below three models random forest regressor is performing well

```
In [47]: # Compare accuracies
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(X_test,Y_test)))

Logistic Regression Test Accuracy: 0.6832298136645962
Decision Tree Test Accuracy: 0.6086956521739131
Random Forest Test Accuracy: 0.6956521739130435
```

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
In [48]: # Creating Random Forest Model
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20)
classifier.fit(X_train, Y_train)
```

```
Out[48]:
RandomForestClassifier
RandomForestClassifier(n_estimators=20)
```

```
In [49]: # Creating a pickle file for the classifier
filename = 'diabetes-prediction-rfc-model.pkl'
pickle.dump(classifier, open(filename, 'wb'))
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the users where they have to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project create two HTML files namely

index.html
prediction.html
result.html

and save them in the templates folder. Refer this [ENTER THE LINK](#) for templates, static and python file

Activity 2.2: Build Python code:

Import the libraries in python file

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (`__name__`) as argument.

Render HTML page:

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

Here we are routing our app to prediction () function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model Predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

```
# Importing essential libraries
from flask import Flask, render_template, request
import pickle
import numpy as np

# Load the Random Forest Classifier model
filename = 'Prediction.pkl'
classifier = pickle.load(open(filename, 'rb'))

app = Flask(__name__)
```

Main Function:

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

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

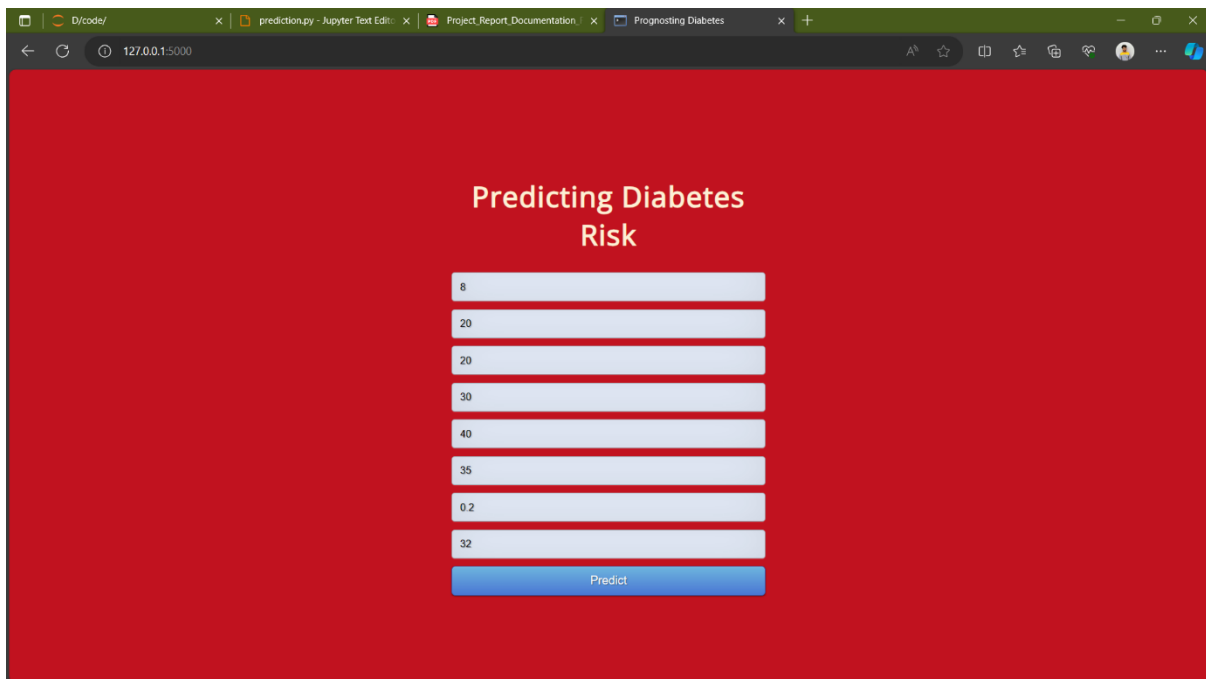
if __name__ == '__main__':
    app.run(debug=True)
```


Activity 2.3: Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
In [2]: runfile('C:/Users/Manu/OneDrive/Documents/D/code/deployment.py', wdir='C:/Users/Manu/OneDrive/Documents/D/code')
* Serving Flask app 'deployment'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Now, Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result



The screenshot shows a web browser window with the address bar set to 127.0.0.1:5000. The page has a solid red background. In the center, the text "Predicting Diabetes Risk" is displayed in white. Below this text is a vertical stack of eight light blue input fields, each containing a white number. The numbers are 8, 20, 20, 30, 40, 35, 0.2, and 32. At the bottom of this stack is a blue button with the word "Predict" in white text.

