Diabetes Milletus Prediction

INTRODUCTION

Diabetes is a chronic disease that occurs when the pancreas is no longer able to make insulin, or when the body cannot make good use of the insulin it produces.
Insulin is a hormone made by the pancreas, that acts like a key to let glucose from the food we eat pass from the blood stream into the cells in the body to produce energy. All carbohydrate foods are broken down into glucose in the blood. Insulin helps glucose get into the cells.
Not being able to produce insulin or use it effectively leads to raised glucose levels in the blood (known as hyperglycaemia). Over the long-term high glucose levels are associated with damage to the body and failure of various organs and tissues.

Types of diabetes:

There are three main types of diabetes – type 1, type 2 and gestational.
Type 1 diabetes can develop at any age, but occurs most frequently in children and adolescents. When you have type 1 diabetes, your body produces very little or no insulin, which means that you need daily insulin injections to maintain blood glucose levels under control. Learn more.
Type 2 diabetes is more common in adults and accounts for around 90% of all diabetes cases. When you have type 2 diabetes, your body does not make good use of the insulin that it produces. The cornerstone of type 2 diabetes treatment is healthy lifestyle, including increased physical activity and healthy diet. However, over time most people with type 2 diabetes will require oral drugs and/or insulin to keep their blood glucose levels under control. Learn more.
Gestational diabetes (GDM) is a type of diabetes that consists of high blood glucose during pregnancy and is associated with complications to both mother and child. GDM usually disappears after pregnancy but women affected and their children are at increased risk of developing type 2 diabetes later in life.

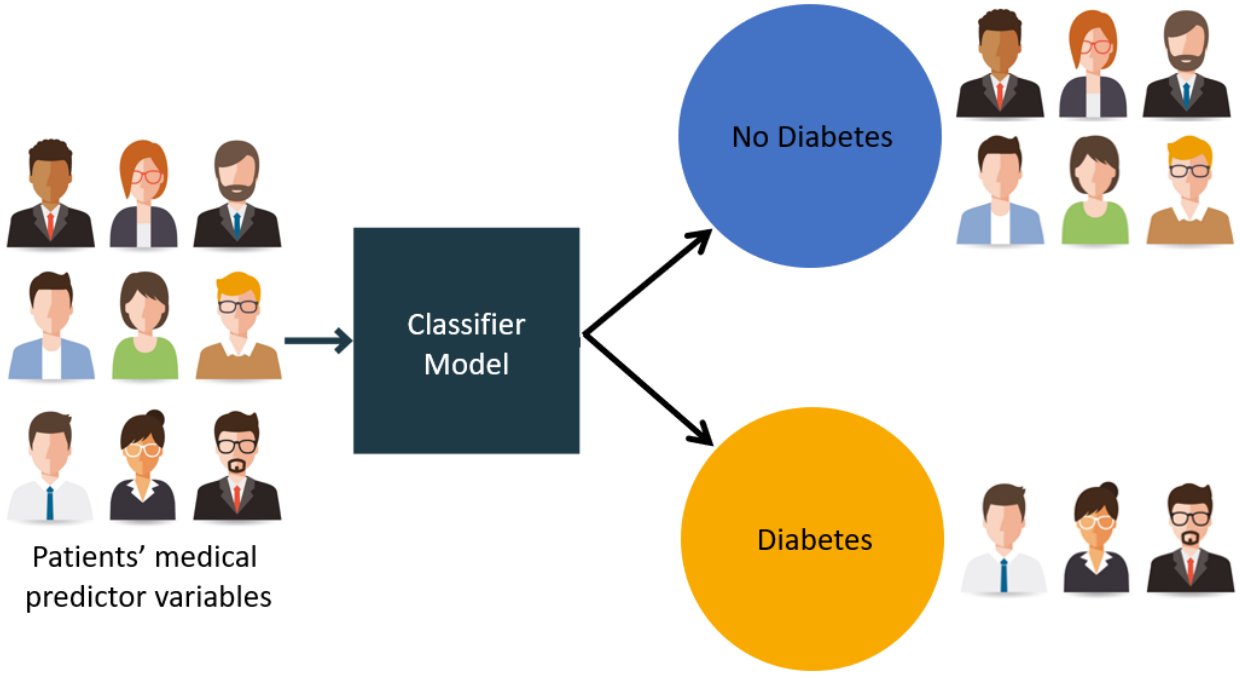
PROBLEM STATEMENT :

Diabetes mellitus is a chronic disease characterized by hyperglycemia. It may cause many complications. According to the growing morbidity in recent years, in 2040, the world's diabetic patients will reach 642 million, which means that one of the ten adults in the future is suffering from diabetes. There is no doubt that this alarming figure needs great attention. With the rapid development of machine learning, machine learning has been applied to many aspects of medical health for accurate predictions.

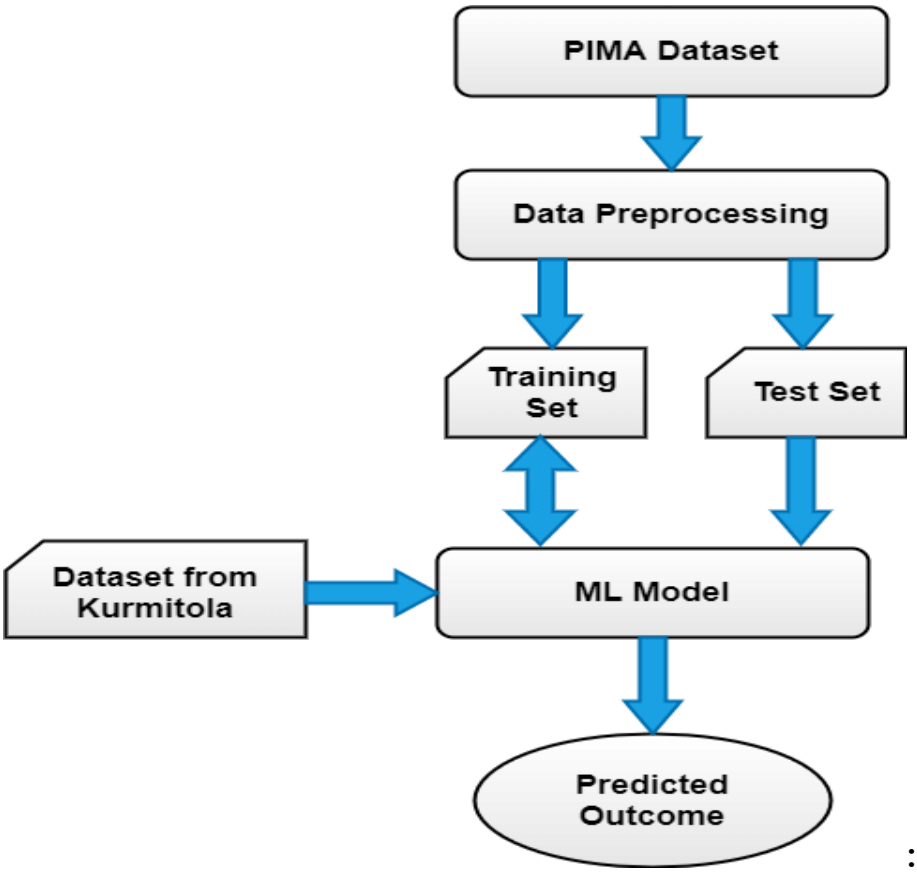
OBJECTIVE

The aim of the project is to build a machine learning model that can efficiently discover the rules to predict the risk level of patients based on the given parameter about their health. Then we evaluate the performance of the model in terms of different parameter like classification accuracy AUC-ROC Curves.

Block Diagram



Machine Learning Workflow



:

Project Flow:

- User interacts with the UI (User Interface) to enter the current mine working conditions.
- Entered readings are analyzed and predictions are made based on interpretation that whether mine will explode or situation will lead to explosion.
- Predictions are popped onto the UI.

1. Data Collection

Download dataset /create dataset:

2. Data Pre-processing

Importing required Libraries:

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn.preprocessing import LabelEncoder
        from sklearn.preprocessing import OneHotEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import r2_score
        from sklearn.metrics import accuracy_score
        import joblib
        from sklearn.svm import SVC
        from sklearn.metrics import confusion_matrix
```

Pandas: It is a python library mainly used for data manipulation.
NumPy: This python library is used for numerical analysis.
Matplotlib and Seaborn: Both are the data visualization library used for plotting graph which will help us for understanding the data.
R2 Score: Coefficient of Determination or R^2 is another metric used for evaluating the performance of a regression model. The metric helps us to compare our current model with a constant baseline and tells us how much our model is better.
Train_test_split: used for splitting data arrays into training data and for testing data.
Decision trees: are used for handling non-linear data sets effectively.
The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC

```
from sklearn.metrics import r2_score
from sklearn.metrics import roc_curve
from sklearn.metrics import average_precision_score, precision_recall_curve
from sklearn.metrics import auc, plot_precision_recall_curve
from sklearn import metrics
```

ROC curves are used in clinical biochemistry to choose the most appropriate cut-off for a test.

Importing the dataset:

```
In [2]: data=pd.read_csv("diabetes.csv")
```

```
In [3]: data.head(10)
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
0	6	148	72	35	0	33.6	0.627	50	pos
1	1	85	66	29	0	26.6	0.351	31	neg
2	8	183	64	0	0	23.3	0.672	32	pos
3	1	89	66	23	94	28.1	0.167	21	neg
4	0	137	40	35	168	43.1	2.288	33	pos
5	5	116	74	0	0	25.6	0.201	30	neg
6	3	78	50	32	88	31.0	0.248	26	pos
7	10	115	0	0	0	35.3	0.134	29	neg
8	2	197	70	45	543	30.5	0.158	53	pos
9	8	125	96	0	0	0.0	0.232	54	pos

- But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.

- Let's load a .csv data file into pandas. There is a function for it, called **read_csv()**. We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).

- Path names on Windows tend to have backslashes in them. But we want them to mean actual backslashes, not special characters.

Data Visualization:

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods and used for determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions**.**

- To check first five rows of dataset, we have a function call **head( )**.

```
In [2]: data=pd.read_csv("diabetes.csv")
```

```
In [85]: data.head(10)
```

Out[85]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

- This head () function returns the first 5 rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

- To check last five rows of dataset, we have a function call **tail()**.

```
In [4]: data.tail(10)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diabetes
758	1	106	76	0	0	37.5	0.197	26	neg
759	6	190	92	0	0	35.5	0.278	66	pos
760	2	88	58	26	16	28.4	0.766	22	neg
761	9	170	74	31	0	44.0	0.403	43	pos
762	9	89	62	0	0	22.5	0.142	33	neg
763	10	101	76	48	180	32.9	0.171	63	neg
764	2	122	70	27	0	36.8	0.340	27	neg
765	5	121	72	23	112	26.2	0.245	30	neg
766	1	126	60	0	0	30.1	0.349	47	pos
767	1	93	70	31	0	30.4	0.315	23	neg

- For finding the names of the columns present in the dataset we make use of

columns

```
In [5]: data.columns
```

```
Out[5]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Diabetes'],
      dtype='object')
```

● data.columns will return you all the column names which are present in your data.

Taking care of Missing Data:

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously, you could remove the entire line of data but what if you are unknowingly removing crucial information? Of course we would not want to do that. One of the most common ideas to handle the problem is to take a mean of all the values for continuous and for categorical we make use of mode values and replace the missing data.

We will be using **isnull().any()** method to see which column has missing values.

```
In [6]: data.isnull().any()
Out[6]: Pregnancies                 False
        Glucose                     False
        BloodPressure               False
        SkinThickness               False
        Insulin                     False
        BMI                         False
        DiabetesPedigreeFunction    False
        Age                         False
        Diabetes                    False
        dtype: bool
```

Since there are no missing values in the dataset, no need to execute this step.

Feature Scaling

Splitting Data into Train and Test:

When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing. But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you split the data in a random manner. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is, 'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

The train-test split is a technique for evaluating the performance of a machine learning

algorithm.

- **Train Dataset:** Used to fit the machine learning model.
- **Test Dataset:** Used to evaluate the fit machine learning model.

In general you can allocate 80% of the dataset to training set and the remaining 20% to test set.
We will create 4 sets— **X_train** (training part of the matrix of features), X_test (test part of the matrix of features), **Y_train** (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.
There are a few other parameters that we need to understand before we use the class:
1) Test_size — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset
2) **train_size** — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
3) **random_state** — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the
4) **Random_state** -- class, which will become the number generator. If you don't pass anything, the Random_state instance used by np.random will be used instead.
5) Now split our dataset into train set and test using train_test_split class from scikit learn library.

```
In [69]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

In [70]: x_train

Out[70]: array([[ 0.93691372,  0.91091809,  0.45982725, ...,  0.40708356,
                  0.66479969,  1.76634642],
                [ 0.04601433, -0.74783062, -0.47073225, ..., -0.48135115,
                 -0.08720986, -0.95646168],
                [-1.14185152,  1.38037527,  1.08020025, ...,  2.57740266,
                 -0.13553176, -0.87137393],
                ...,
                [ 0.04601433, -0.84172205, -0.2122435 , ..., -0.92556851,
                 -0.97814487, -1.04154944],
                [ 2.12477957, -1.12339636,  0.25303625, ..., -0.24020459,
                 -0.51908683,  0.14967911],
                [ 0.3429808 ,  0.47275805,  0.66661825, ..., -4.06047387,
                  0.50775352,  3.04266271]])

In [71]: x_train.shape,x_test.shape

Out[71]: ((614, 8), (154, 8))

In [72]: y_train.shape,y_test.shape

Out[72]: ((614, 1), (154, 1))
```

Model Building

Training and testing the model:

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms are Regression algorithms.
Example:

1. Linear Regression.
2. Logistic Regression.
3. Random Forest Regression / Classification.
4. Decision Tree Regression / Classification.

You will need to train the datasets to run smoothly and see an incremental improvement in the prediction rate.

Now we apply Decision TreeClassifier algorithm on our dataset.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

The dependent variable is a binary variable that contains data coded as 1 (yes, positive, etc.) or 0 (no, negitive, etc.).

```
In [73]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import r2_score
         from sklearn.metrics import roc_curve
         from sklearn.metrics import average_precision_score, precision_recall_curve
         from sklearn.metrics import auc, plot_precision_recall_curve
         from sklearn import metrics

In [74]: dtc=DecisionTreeClassifier()

In [75]: dtc=dtc.fit(x_train,y_train)

In [76]: pred1=dtc.predict(x_test)
```

Predict the values

Once the model is trained, it's ready to make predictions. We can use the predict method on the model and pass x_test as a parameter to get the output as **pred1**.

Notice that the prediction output is an array of real numbers corresponding to the input array.

Evaluation:

Finally, we need to check to see how well our model is performing on the test data.

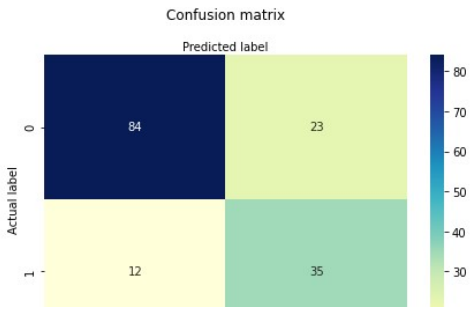
There are many evaluation techniques are there. For this, we evaluate and produce the model.

```
In [75]: dtc=dtc.fit(x_train,y_train)

In [76]: pred1=dtc.predict(x_test)

In [77]: pred1

Out[77]: array([1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,
                0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
                1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
                0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
                0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

In [78]: print("Accuracy:",metrics.accuracy_score(y_test, pred1))
         print("Precision:",metrics.precision_score(y_test, pred1))
         print("Recall:",metrics.recall_score(y_test, pred1))

         Accuracy: 0.7727272727272727
         Precision: 0.603448275862069
         Recall: 0.7446808510638298
```

We also plot **Confusion Matrix** for the same to evaluate.

A **confusion matrix** is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall.

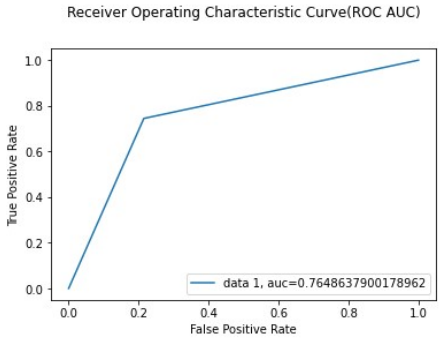
```
In [79]: cnf_matrix = metrics.confusion_matrix(y_test, pred1)
         cnf_matrix
         class_names=[0,1] # name  of classes
         fig, ax = plt.subplots()
         tick_marks = np.arange(len(class_names))
         plt.xticks(tick_marks, class_names)
         plt.yticks(tick_marks, class_names)
         # create heatmap
         sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
         ax.xaxis.set_label_position("top")
         plt.tight_layout()
         plt.title('Confusion matrix', y=1.1)
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')

Out[79]: Text(0.5, 257.44, 'Predicted label')
```



ROC AUC Curves

ROC stands for Receiver Operating Characteristic curve. This is a graph that shows the performance of a machine learning model on a classification problem by plotting the true positive rate and the false positive rate. AUC stands for Area Under the Curve. It is used to measure the entire area under the ROC curve.

```
In [80]: y_pred_proba = dtc.predict_proba(x_test)[::,1]
         fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
         auc = metrics.roc_auc_score(y_test, y_pred_proba)
         plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
         plt.title('Receiver Operating Characteristic Curve(ROC AUC)', y=1.1)
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend(loc=4)
         plt.show()
```



Now we will apply **Random Forest** algorithm

A **Random Forest** is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

```
In [82]: from sklearn.ensemble import RandomForestClassifier
         ran=RandomForestClassifier(n_estimators=100)
         ran.fit(x_train,y_train)

         <ipython-input-82-2af034180312>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
         nge the shape of y to (n_samples,), for example using ravel().
           ran.fit(x_train,y_train)

Out[82]: RandomForestClassifier()

In [83]: pred2=ran.predict(x_test)
```

Predict the values

Once the model is trained, it's ready to make predictions. We can use the predict method on the model and pass x_test as a parameter to get the output as **pred2**.

Notice that the prediction output is an array of real numbers corresponding to the input array.

Evaluation:

Finally, we need to check to see how well our model is performing on the test data. There are many evaluation techniques are there.

```
In [84]: pred2
```

```
Out[84]: array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
                0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
In [85]: print("Accuracy:",metrics.accuracy_score(y_test, pred2))
         print("Precision:",metrics.precision_score(y_test, pred2))
         print("Recall:",metrics.recall_score(y_test, pred2))
```

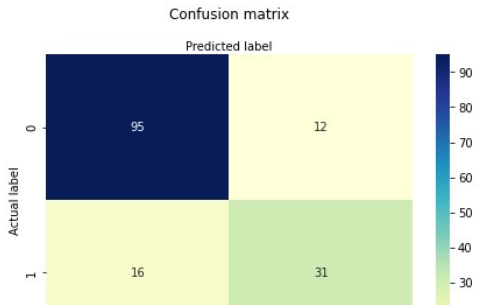
```
Accuracy: 0.8181818181818182
Precision: 0.7209302325581395
Recall: 0.6595744680851063
```

We also plot **Confusion Matrix** for the same to evaluate.

A **confusion matrix** is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall.

```
In [86]: cnf_matrix = metrics.confusion_matrix(y_test, pred2)
         cnf_matrix
         class_names=[0,1] # name  of classes
         fig, ax = plt.subplots()
         tick_marks = np.arange(len(class_names))
         plt.xticks(tick_marks, class_names)
         plt.yticks(tick_marks, class_names)
         # create heatmap
         sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
         ax.xaxis.set_label_position("top")
         plt.tight_layout()
         plt.title('Confusion matrix', y=1.1)
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')
```

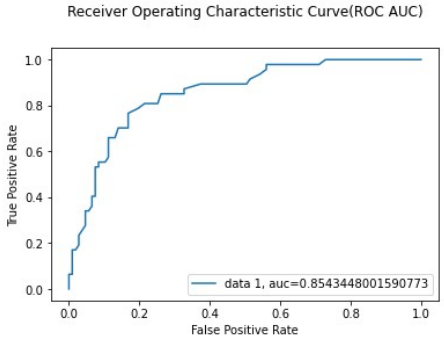
```
Out[86]: Text(0.5, 257.44, 'Predicted label')
```



ROC AUC Curves

ROC stands for Receiver Operating Characteristic curve. This is a graph that shows the performance of a machine learning model on a classification problem by plotting the true positive rate and the false positive rate. AUC stands for Area Under the Curve. It is used to measure the entire area under the ROC curve.

```
In [87]: y_pred_proba = ran.predict_proba(x_test)[::,1]
         fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
         auc = metrics.roc_auc_score(y_test, y_pred_proba)
         plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
         plt.title('Receiver Operating Characteristic Curve(ROC AUC)', y=1.1)
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend(loc=4)
         plt.show()
```



Saving a model:

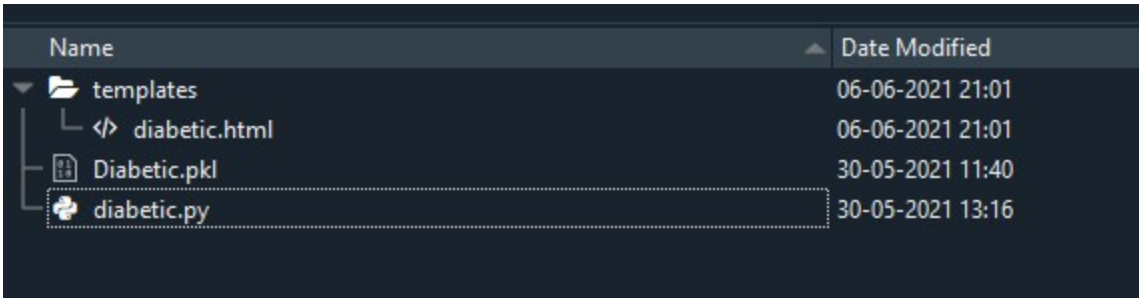
Model is saved so it can be used in future and no need to train it again.

```
In [89]: import pickle
         pickle.dump(ran,open('Diabetic.pkl','wb'))
```

1) Build python code
2) Importing Libraries
3) Routing to the html Page
4) Showcasing prediction on UI
5) Run The app in local browser

Project Structure:

Create a Project folder that contains files as shown below



- We are building a Flask Application that needs HTML pages stored in the templates folder
- Templates folder contains index.html
- Static folder contains CSS and image files.

Task 1: Importing Libraries

```
from flask import Flask,render_template,request
import pickle
import numpy as np

app=Flask(__name__) #your application
ran=pickle.load(open('diabetic.pkl','rb'))
```

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 Pickle library to load the model file.

Task 2: Routing to the html Page

Here, declared constructor is used to route to the HTML page created earlier.

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

```python
@app.route('/') # default route
def home():
    return render_template("diabetic.html")


@app.route('/predict',methods=['post'])
def predict():
    Pregnancies=float(request.form['Pregnancies'])
    Glucose=float(request.form['Glucose'])
    BloodPressure=float(request.form['BloodPressure'])
    SkinThickness=float(request.form['SkinThickness'])
    Insulin=float(request.form['Insulin'])
    BMI=float(request.form['BMI'])
    DiabetesPedigreeFunction=float(request.form['DiabetesPedigreeFunction'])
    Age=float(request.form['Age'])

    print(Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age)
    a=np.array([[Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age]])

    result=ran.predict(a)

    return render_template('diabetic.html',x=result)
```

Task 3: Main Function

This is used to run the application in a local host.

```python
if __name__ == '__main__':
    app.run(port=8000) # yo
```

Activity 3: Run the application :

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type “python app.py” command.
- It will show the local host where your app is running on http://127.0.0.1:8000/
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

```
In [2]: runfile('C:/Users/S AJAY/Desktop/Internship/Flask/diabetic.py', wdir='C:/Users/S AJAY/Desktop/
Internship/Flask')
 * Serving Flask app "diabetic" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

Output Screen :