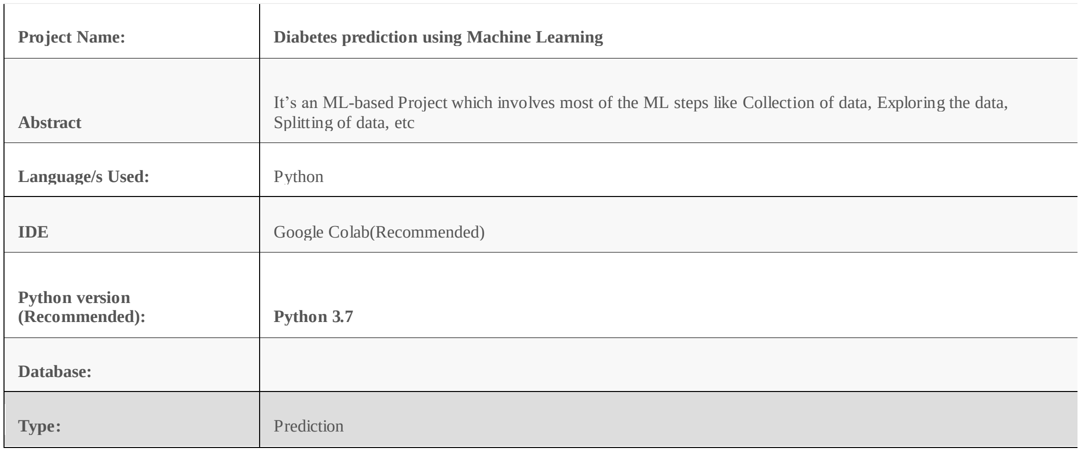
AI based diabetes prediction system

Introduction:

A system is used to predict whether a patient has diabetes based on some of its health-related details such as BMI (Body Mass Index), blood pressure, Insulin, etc.

About Diabetes Prediction using Machine Learning Project:



Project Name:	**Diabetes prediction using Machine Learning**
Abstract	It’s an ML-based Project which involves most of the ML steps like Collection of data, Exploring the data, Splitting of data, etc
Language/s Used:	Python
IDE	Google Colab(Recommended)
Python version (Recommended):	**Python 3.7**
Database:	
Type:	Prediction

Steps involved in Diabetes Prediction Project:

The workflow to build the end-to-end Machine Learning Project to predict Diabetes is as follows-

1. Collection of data
2. Exploring the data
3. Splitting the data
4. Training the model
5. Evaluating the model
6. Deploying the model

Data collection:

The very first step is to choose the dataset for our model. We can get a lot of different datasets from Kaggle. You just need to sign in to Kaggle and search for any dataset you need for the project. The Diabetes dataset required for our model can be downloaded here(https://www.kaggle.com/datasets/mathchi/diabetes-data-set).

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict whether a patient has diabetes based on diagnostic measurements. 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 data contains 9 columns which are as follows

- **Pregnancies**: Number of times pregnant
- **Glucose**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **BloodPressure**: Diastolic blood pressure (mm Hg)
- **SkinThickness**: Triceps skin fold thickness (mm)
- **Insulin**: 2-Hour serum insulin (mu U/ml)
- **BMI**: Body mass index (weight in kg/(height in m)^2)
- **DiabetesPedigreeFunction**: Diabetes pedigree function
- **Age**: Age (years)
- **Outcome**: Class variable (0 or 1)

Exploring the Data:

Now we have to set the development environment to build our project. For this project, we are going to build this Diabetes prediction using Machine Learning in Google Colab. You can also use Jupyter Notebook.

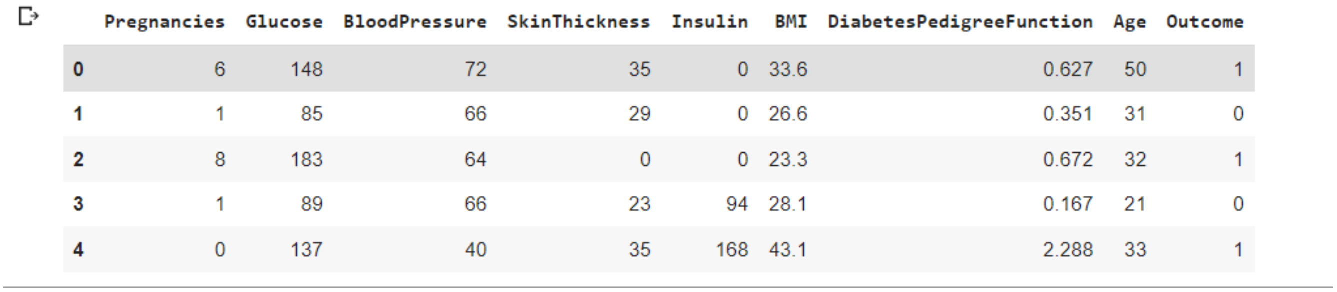
After downloading the dataset, import the necessary libraries to build the model.

```
# Import the required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
import pickle
```

Load the data using the *read_csv* method in the pandas library. Then the *head()* method in the pandas library is used to print the rows up to the limit we specify. The default number of rows is five.

```
# Load the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv('diabetes.csv')

# Print the first 5 rows of the dataset
diabetes_dataset.head()
```

Output:

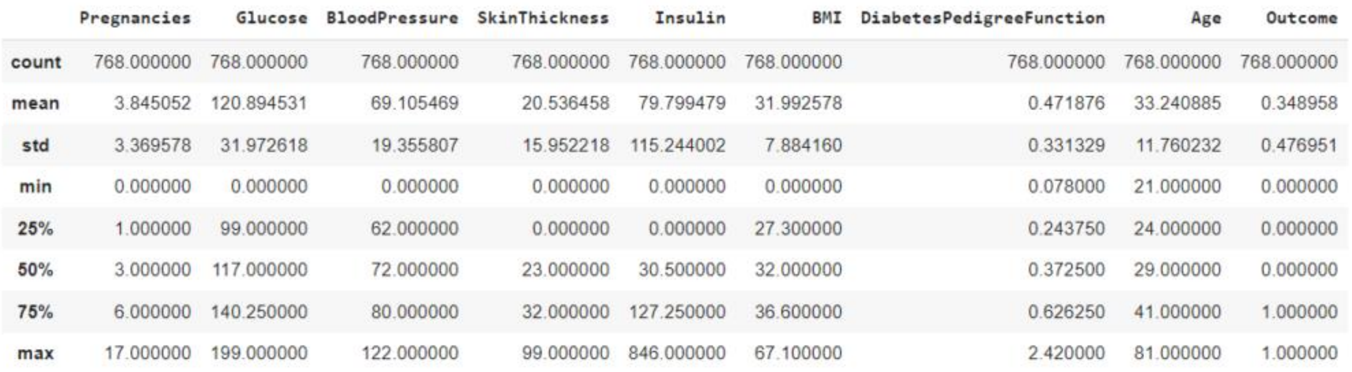


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# To get the number of rows and columns in the dataset
diabetes_dataset.shape
#prints (768, 9)
```

```
# To get the statistical measures of the data
diabetes_dataset.describe()
```

Output**:**

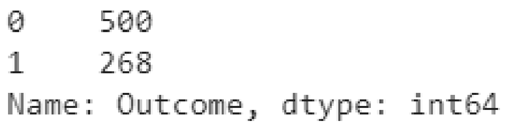


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	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

And, it is clear that the Outcome column is the output variable. So let us explore more details about that column.

```
# To get details of the outcome column
diabetes_dataset['Outcome'].value_counts()
```

Output:



```
0    500
1    268
Name: Outcome, dtype: int64
```

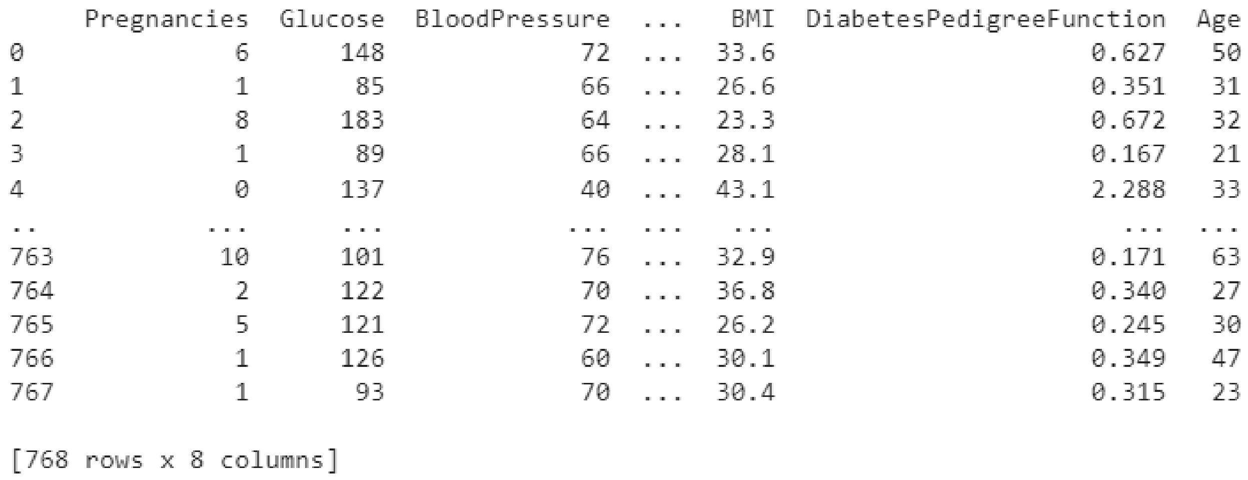
In the output, the value 1 means the person is having Diabetes, and 0 means the person is not having Diabetes. We can see the total count of people with and without Diabetes.

Splitting the data:

The next step in the building of the Machine learning model is splitting the data into training and testing sets. The training and testing data should be split in a ratio of 3:1 for better prediction results.

```
# separating the data and labels
X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
Y = diabetes_dataset['Outcome']

# To print the independent variables
print(X)
```

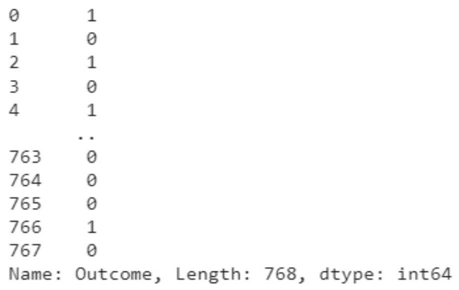
Output:



```
     Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Age
0              6      148             72  ...  33.6                     0.627   50
1              1       85             66  ...  26.6                     0.351   31
2              8      183             64  ...  23.3                     0.672   32
3              1       89             66  ...  28.1                     0.167   21
4              0      137             40  ...  43.1                     2.288   33
..           ...      ...            ...  ...   ...                       ...  ...
763           10      101             76  ...  32.9                     0.171   63
764            2      122             70  ...  36.8                     0.340   27
765            5      121             72  ...  26.2                     0.245   30
766            1      126             60  ...  30.1                     0.349   47
767            1       93             70  ...  30.4                     0.315   23

[768 rows x 8 columns]
```

```
# To print the outcome variable
print(Y)
```

Output:



```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
#Split the data into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2,
stratify=Y, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
```

Output:

```
(768, 8) (614, 8) (154, 8)
```

Training the model:

The next step is to build and train our model. We are going to use a Support vector classifier algorithm to build our model.

```
# Build the model
classifier = svm.SVC(kernel='linear')

# Train the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

After building the model, the model has to predict output with test data. After the prediction of the outcome with test data, we can calculate the accuracy score of the prediction results by the model.

```
# Accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy score of the training data : ', training_data_accuracy)

# Accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score of the test data : ', test_data_accuracy)
```

Output:

```
Accuracy score of the training data: 0.7833876221498371
Accuracy score of the test data: 0.7727272727272727
```

Evaluating the model:

```
input_data = (5,166,72,19,175,25.8,0.587,51)

# Change the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the array for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = classifier.predict(input_data_reshaped)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

Output:

The person is diabetic

Saving the file

```
# Save the trained model
filename = 'trained_model.sav'
pickle.dump(classifier, open(filename, 'wb'))

# Load the saved model
loaded_model = pickle.load(open('trained_model.sav', 'rb'))
```

Once you run this code a new file named trained_model.sav will be saved in the project folder.

Deploying the model:

One of the most important and final steps in building a Machine Learning project is Model deployment. There are many frameworks available for deploying the Machine learning model on the web. Some of the most used Python frameworks are Django and Flask. But these frameworks require a little knowledge of languages such as HTML, CSS, and JavaScript.

So, a new framework known as Streamlit was introduced to deploy the Machine Learning model without the need to have the knowledge of Front End Languages. It is quite easy to deploy using Streamlit. So, we will use the Streamlit framework to deploy our model. Although Streamlit has many advantages over the other frameworks, lot more features are under development. If you are getting started in Machine Learning then this framework will be a perfect start to deploy your machine learning model on the web.

Python Code to Deploy ML model using Streamlit

To install Streamlit run the following command in the command prompt or terminal.

```
pip install streamlit
```

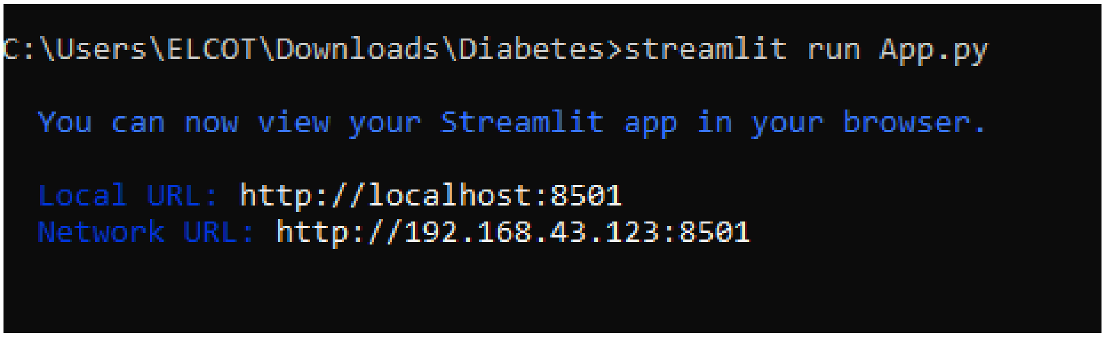
Open a new Python file and put the following code.

App.py

```
import numpy as np
import pickle
import streamlit as st

# Load the saved model
loaded_model = pickle.load(open('C:/Users/ELCOT/Downloads/trained_model.sav', 'rb'))

# Create a function for Prediction
def diabetes_prediction(input_data):

    # Change the input_data to numpy array
    input_data_as_numpy_array = np.asarray(input_data)

    # Reshape the array as we are predicting for one instance
    input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

    prediction = loaded_model.predict(input_data_reshaped)
    print(prediction)

    if (prediction[0] == 0):
      return 'The person is not diabetic'
    else:
      return 'The person is diabetic'

def main():

    # Give a title
    st.title('Diabetes Prediction Web App')

    # To get the input data from the user
    Pregnancies = st.text_input('Number of Pregnancies')
    Glucose = st.text_input('Glucose Level')
    BloodPressure = st.text_input('Blood Pressure value')
    SkinThickness = st.text_input('Skin Thickness value')
    Insulin = st.text_input('Insulin Level')
    BMI = st.text_input('BMI value')
    DiabetesPedigreeFunction = st.text_input('Diabetes Pedigree Function value')
    Age = st.text_input('Age of the Person')

    # Code for Prediction
    diagnosis = ''

    # Create a button for Prediction

    if st.button('Diabetes Test Result'):
        diagnosis = diabetes_prediction([Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age])

    st.success(diagnosis)

if __name__ == '__main__':
    main()
```

Save the file after pasting the code. And then to deploy using streamlit go to command prompt and run the following command.

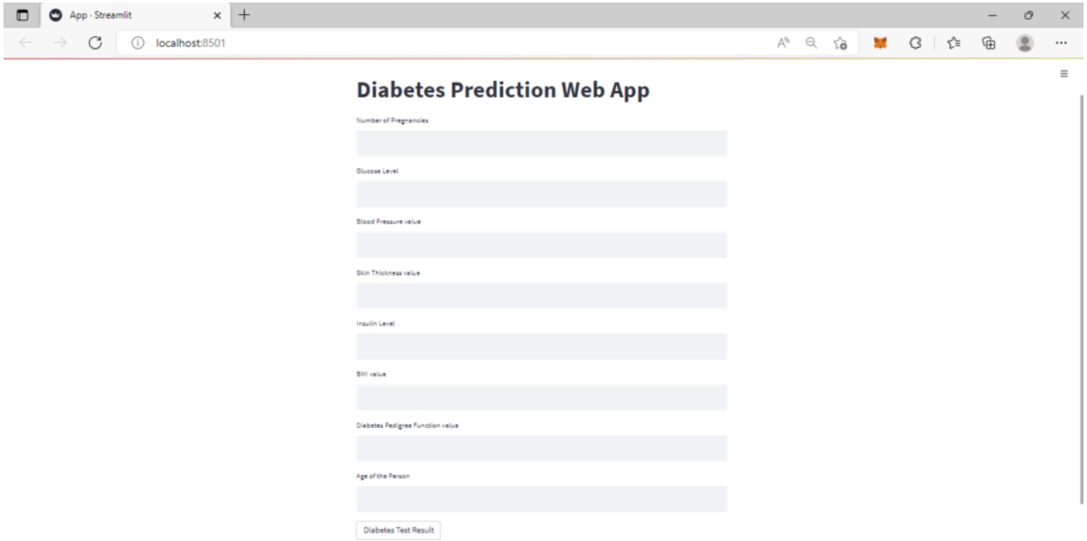
```
streamlit run App.py
(or)
streamlit run filename.py
```



```
C:\Users\ELCOT\Downloads\Diabetes>streamlit run App.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.43.123:8501
```

After running the command the web app will open in the localhost webserver. Otherwise, go to your browser and type *localhost:8501*. The following output will be shown.

Output:



Diabetes Prediction Web App

Number of Pregnancies

Glucose Level

Blood Pressure value

Skin Thickness value

Insulin Level

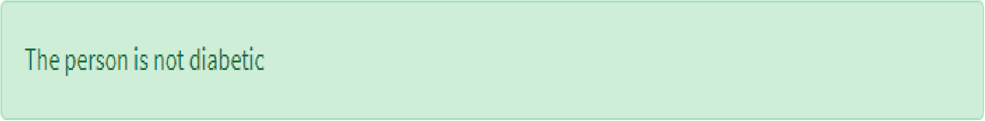
BMI value

Diabetes Pedigree Function value

Age of the Person

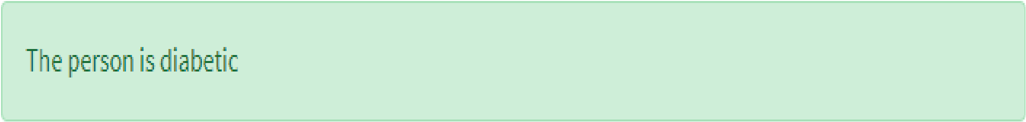
Diabetes Test Result

Sample Input data for a person does not have diabetes is {1, 85, 66, 29, 0, 26.6, 0.351, 31}. These data as input will generate the following output in the web app.



The person is not diabetic

Sample input data for a person who have diabetes is {6, 148, 72, 35, 0, 33.6, 0.627, 50}. These data as input will generate the following output in the web app.



The person is diabetic

Conclusion

After using all these patient records, we are able to build a machine learning model (random forest – best one) to accurately predict whether or not the patients in the dataset have diabetes or not along with that we were able to draw some insights from the data via data analysis and visualization.