Title of the Project:

PIMA-INDIANS DIABETES PREDICTION MODEL

Name: Jerin Lalichan
Email: jerinlalichan@gmail.com
Ph: 8921931398,9496396456

Problem Statement:

Build a model to accurately predict whether the patients in the dataset have diabetes or not?

Abstract

The diabetes dataset is a binary classification problem where it needs to be analysed whether a patient is suffering from the disease or not on the basis of many available features in the dataset. Different methods and procedures of cleaning the data, feature extraction, feature engineering and algorithms to predict the onset of diabetes are used based for diagnostic measure on Pima Indians Diabetes Dataset

Dataset Description:

The datasets consist of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

1: *Pregnancies*: Number of times pregnant

2: *Glucose*: Plasma glucose concentration, 2 hours in an oral glucose tolerance test.

3: *Blood Pressure*: Diastolic blood pressure (mm Hg)

4: *Skin Thickness*: Triceps skinfold thickness (mm)

5: *Insulin*: 2-Hour serum insulin (mu U/ml)

6: *BMI*: Body mass index (weight in kg/ (height in m) ^{2})

7: *DiabetesPedigreeFunction*: Diabetes pedigree function

8: *Age*: Age (years)

9: *Outcome*: Class variable (0 or 1) 268 of 768 are 1, the others are 0

Importing Libraries:

```
#importing required libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

Loading Dataset:

```
#Loading dataset
df = pd.read_csv('/content/drive/MyDrive/Luminar_Projects/Diabetes_Prediction/diabetes.csv')
df.head()
```

	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

EDA (Exploratory Data Analysis)

- Analysed the target variable – Outcome, and is found as an imbalanced dataset

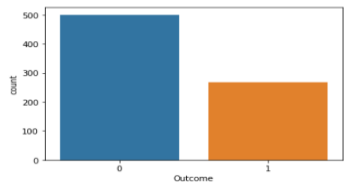
```
# checking the unique values in target variable
df['Outcome'].unique()
```

```
array([1, 0])
```

```
df['Outcome'].value_counts()
```

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

```
# plotting the target variable
sns.countplot(x = df.Outcome,data = df)
plt.show()
```



- Checking the basic info about the dataset such as shape, basic info, describe etc., and Checked for missing values.

```
#shape of the dataset
df.shape
```

```
(768, 9)
```

```
# Dataset basic informations
df.info()
```

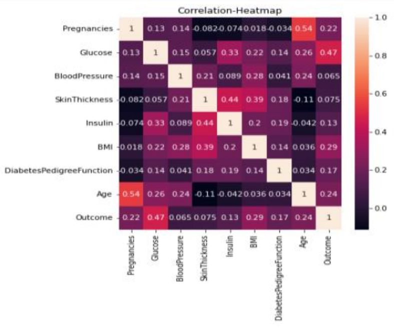
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   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
```

```
# Describing the data
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

- Correlation between target variable is found and is plotted as heatmap

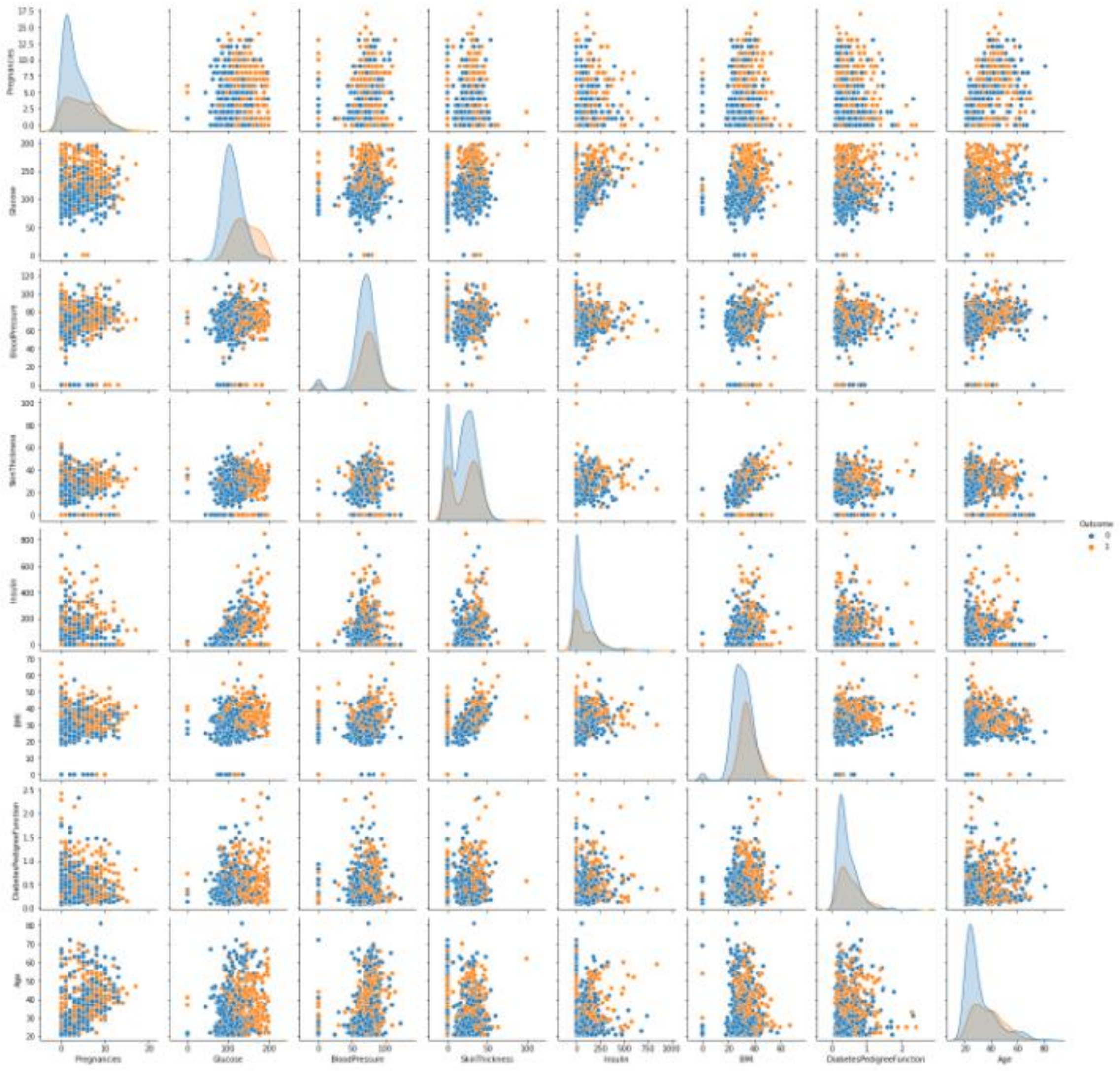
```
# Correlation using heatmap n dataset
plt.figure(figsize=(6,6))
sns.heatmap(df.corr(),annot=True)
plt.title('Correlation-Heatmap')
plt.show()
```



- Pairplot is plotted

```
# Pairplot -sns
sns.pairplot(df,hue='Outcome')
```

```
<seaborn.axisgrid.PairGrid at 0x7ff72b1040d0>
```



- Missing Values: Since certain variables in the dataset cannot be zero, The zeros in the dataset are treated as nothing but missing values. And are printed using for-loop.

```
# finding number of zero values
print('Total zero Glucose values: ' + str(df[df["Glucose"]==0].shape[0]))
print('Total zero BloodPressure values: ' + str(df[df["BloodPressure"]==0].shape[0]))
print('Total zero SkinThickness values: ' + str(df[df["SkinThickness"]==0].shape[0]))
print('Total zero Insulin values: ' + str(df[df["Insulin"]==0].shape[0]))
print('Total zero BMI values: ' + str(df[df["BMI"]==0].shape[0]))
print('Total zero DiabetesPedigreeFunction values: ' + str(df[df["DiabetesPedigreeFunction"]==0].shape[0]))
print('Total zero Age values: ' + str(df[df["Age"]==0].shape[0]))
```

```
Total zero Glucose values: 5
Total zero BloodPressure values: 35
Total zero SkinThickness values: 227
Total zero Insulin values: 374
Total zero BMI values: 11
Total zero DiabetesPedigreeFunction values: 0
Total zero Age values: 0
```

Missing Value Treatment:

- Method adopted – Replacing with median

```
# Replacing with median
df['SkinThickness'].loc[(df['SkinThickness']==0)] =df['SkinThickness'].median()
df['Glucose'].loc[(df['Glucose']==0)] =df['Glucose'].median()
df['BloodPressure'].loc[(df['BloodPressure']==0)] =df['BloodPressure'].median()
df['Insulin'].loc[(df['Insulin']==0)] =df['Insulin'].median()
df['BMI'].loc[(df['BMI']==0)] =df['BMI'].median()
```

Outlier Treatment:

- Method adopted – Outlier Detection with Standard Deviation

```
# Outlayer treatment- Capping
for column in continuecols:
    upperlimit=df[column].mean()+3*df[column].std()
    lowerlimit=df[column].mean()-3*df[column].std()
    df.loc[(df[column]>upperlimit),column]=upperlimit
    df.loc[(df[column]<lowerlimit),column]=lowerlimit
```

Splitting the dataset into train and test sets:

```
X=df.drop(['Outcome'],axis=1)
y=df['Outcome']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=10)
```

Scaling Down:

- Scaled down the data set using StandardScalar()
- Train using fit-transform and Test using transform.
- Created a pickle file “scale.pkl” for scaling process using jobib.

```
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.transform(X_test)


import joblib
joblib.dump(scale,'/content/drive/MyDrive/Luminar_Projects/Diabetes_Prediction/Scale.pkl')
```

Data modelling

Algorithms:

Applying these classification Algorithms: Logistic Regression, Decision Tree Classifier, AdaBoost Classifier, Gradient Boosting, Random Forest, SVC and Voting classifier.

```
# Apply all classification Algorithms -- Include Train Acc, Test acc, Precision, Recall, F1 score  -- In For Loop
lr_clf = LogisticRegression()
df_clf= DecisionTreeClassifier()
rf_clf= RandomForestClassifier()
adboost_clf= AdaBoostClassifier()
grad_clf=GradientBoostingClassifier()
svc_clf=SVC()

voting  = VotingClassifier(estimators=[('Logistic_Regression',lr_clf),('Decision_Tree_Classifier',df_clf),('Random_Forest_Classifier',rf_clf),
                              ('AdaBoost_Classifier',adboost_clf),('Gradient_Boosting_Classifier',grad_clf),('SVC',svc_clf)], voting='hard')


voting.fit(X_train,y_train)
```

- checking train vs test accuracy scores:

```python
#checking train vs test accuracy scores
for clf in (lr_clf,df_clf,rf_clf,adboost_clf,grad_clf,svc_clf):
  clf.fit(X_train,y_train)
  y_pred = clf.predict(X_test)
  print(clf.__class__.__name__,clf.score(X_train,y_train))
  print(clf.__class__.__name__, accuracy_score(y_test,y_pred))

  print("=========================")
```

```
LogisticRegression 0.7877094972067039
LogisticRegression 0.7402597402597403
=========================
DecisionTreeClassifier 1.0
DecisionTreeClassifier 0.7142857142857143
=========================
RandomForestClassifier 1.0
RandomForestClassifier 0.7489177489177489
=========================
AdaBoostClassifier 0.8528864059590316
AdaBoostClassifier 0.7316017316017316
=========================
GradientBoostingClassifier 0.925512104283054
GradientBoostingClassifier 0.7662337662337663
=========================
SVC 0.7858472998137802
SVC 0.7056277056277056
```

Hyperparameter Tuning:

- It is done in all algorithms to find the best Model, and the accuracy & best parameters are printed out.

```python
# from sklearn.model_selection import GridSearchCV

lr_clf = LogisticRegression()
df_clf= DecisionTreeClassifier()
rf_clf= RandomForestClassifier()
adboost_clf= AdaBoostClassifier()
grad_clf=GradientBoostingClassifier()
svc_clf=SVC()

clf_list=[lr_clf,df_clf,rf_clf,adboost_clf,grad_clf,svc_clf]
```

```python
from sklearn.model_selection import GridSearchCV
lr_clf = LogisticRegression()
df_clf= DecisionTreeClassifier()
rf_clf= RandomForestClassifier()
adboost_clf= AdaBoostClassifier()
grad_clf=GradientBoostingClassifier()
svc_clf=SVC()

clf_list=[lr_clf,df_clf,rf_clf,adboost_clf,grad_clf,svc_clf]

grid_params_lr= [{'penalty':['l1','l2'],'solver':['saga']}]

grid_params_df =[{'criterion':["gini","entropy"], 'splitter':['best','random'],'max_depth':[3,4,5],'min_samples_split':[2,3,4],'max_features':["auto",

grid_params_rf=[{'n_estimators': [4, 6, 9], 'max_features': ['log2', 'sqrt','auto'], 'criterion': ['entropy', 'gini'],'max_depth': [2, 3, 5, 10]}]

grid_params_adboost=[{'n_estimators':[10,50,250,1000],'learning_rate':[0.01,0.1],}]

grid_params_grad=[{'loss':['deviance', 'exponential'],'learning_rate':[1,7,9],'criterion':['friedman_mse','squared_error']}]

grid_params_svc=[{'kernel':['linear','poly','rbf'],'degree':[3,4,5]}]



clf_params=[grid_params_lr,grid_params_df,grid_params_rf,grid_params_adboost,grid_params_grad,grid_params_svc]
```

```python
for clf,clf_param in zip(clf_list,clf_params):
  print(f"The Classifier is {clf} and its hyper params are {clf_param}")

  grid_clf = GridSearchCV(estimator=clf,param_grid=clf_param,scoring="accuracy",cv=10)
  grid_clf.fit(X_train_scaled,y_train)

  print(f"The Train accuracy for the {clf} is {grid_clf.score(X_train_scaled,y_train)}")

  print(f"The Test accuracy for the {clf} is {grid_clf.score(X_test_scaled,y_test)}")

  print(f"The Best param for the {clf} is {grid_clf.best_params_}")
  print("-------------------\n")
```

- From the output, AdaBoost classifier found to be performing the best on the basis of Accuracy value.

```
The Classifier is AdaBoostClassifier() and its hyper params are [{'n_estimators': [10, 50, 250, 1000], 'learning_rate': [0.01, 0.1]}]
The Train accuracy for the AdaBoostClassifier() is 0.8081936685288641
The Test accuracy for the AdaBoostClassifier() is 0.7359307359307359
The Best param for the AdaBoostClassifier() is {'learning_rate': 0.01, 'n_estimators': 1000}
```

Train accuracy was found to be = 80.81%
Test accuracy was found to be = 73.59%

- Created a Pickle file of the model using joblib.

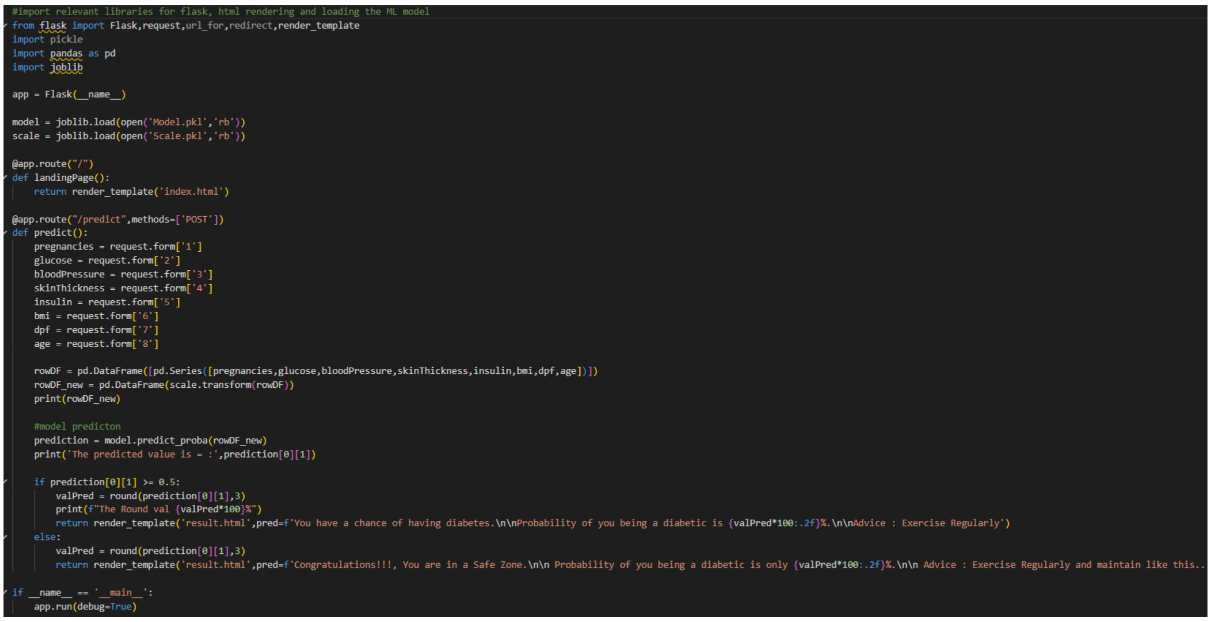
```
# Making the best file

adboost_clf = AdaBoostClassifier( learning_rate =0.01, n_estimators=1000)
adboost_clf.fit(X_train_scaled,y_train)
```

```
import joblib
joblib.dump(adboost_clf,'/content/drive/MyDrive/Luminar_Projects/Diabetes_Prediction/Model.pkl')
```

Web Framework:

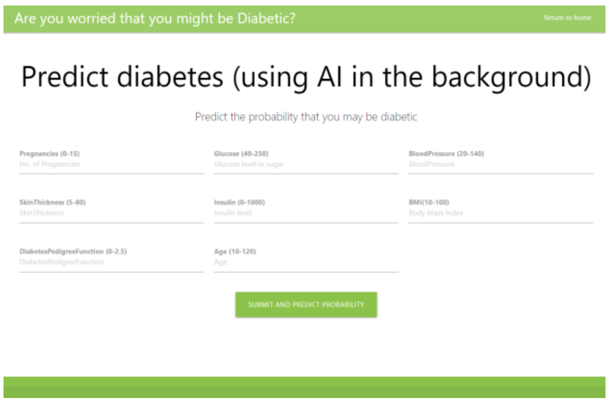
- I have used Flask for setting the back-end API.
- The UI is set using html code.

Flask code:



```
#import relevant libraries for flask, html rendering and loading the ML model
from flask import Flask,request,url_for,redirect,render_template
import pickle
import pandas as pd
import joblib

app = Flask(__name__)

model = joblib.load(open('Model.pkl','rb'))
scale = joblib.load(open('Scale.pkl','rb'))

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

@app.route("/predict",methods=['POST'])
def predict():
    pregnancies = request.form['1']
    glucose = request.form['2']
    bloodPressure = request.form['3']
    skinThickness = request.form['4']
    insulin = request.form['5']
    bmi = request.form['6']
    dpf = request.form['7']
    age = request.form['8']

    rowDF = pd.DataFrame([pd.Series([pregnancies,glucose,bloodPressure,skinThickness,insulin,bmi,dpf,age])])
    rowDF_new = pd.DataFrame(scale.transform(rowDF))
    print(rowDF_new)

    #model predicton
    prediction = model.predict_proba(rowDF_new)
    print('The predicted value is = :',prediction[0][1])

    if prediction[0][1] >= 0.5:
        valPred = round(prediction[0][1],3)
        print(f"The Round val {valPred*100}%")
        return render_template('result.html',pred=f'You have a chance of having diabetes.\n\nProbability of you being a diabetic is {valPred*100:.2f}%.\n\nAdvice : Exercise Regularly')
    else:
        valPred = round(prediction[0][1],3)
        return render_template('result.html',pred=f'Congratulations!!!, You are in a Safe Zone.\n\n Probability of you being a diabetic is only {valPred*100:.2f}%.\n\n Advice : Exercise Regularly and maintain like this..

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

User Interface:

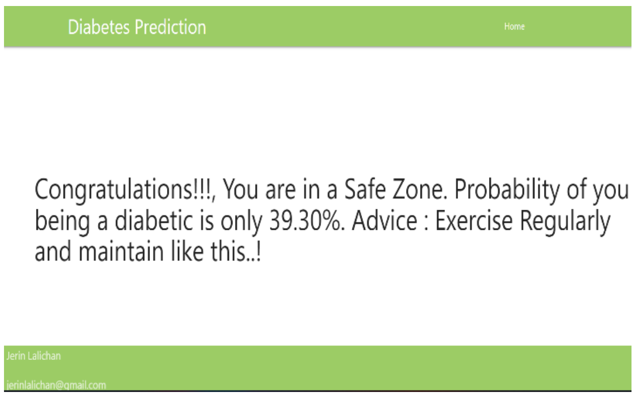


Are you worried that you might be Diabetic?

Predict diabetes (using AI in the background)

Predict the probability that you may be diabetic

Pregnancies (0-15) No. of Pregnancies	Glucose (40-250) Glucose level in sugar	BloodPressure (20-140) BloodPressure
SkinThickness (5-80) SkinThickness	Insulin (0-1000) Insulin level	BMI(10-100) Body Mass Index
DiabetesPedigreeFunction (0-2.5) DiabetesPedigreeFunction	Age (10-120) Age	



Diabetes Prediction

Congratulations!!!, You are in a Safe Zone. Probability of you being a diabetic is only 39.30%. Advice : Exercise Regularly and maintain like this..!

Jerin Lalichan
jerinlalichan@gmail.com

Model Deployment:

- I have done the project in google colab and later pushed the files to my GitHub repository
  https://github.com/Jeri-n/Pima_Indians_Diabetes_Prediction_Model
- I have deployed the project through Heroku
  https://ai-diabetes-predictor.herokuapp.com/