# Description

The dataset contains several medical predictor (Independent) variables and one target variable, (Outcome). Predictor variables include:

1. Pregnancies
2. Glucose
3. BloodPressure
4. SkinThickness
5. Insulin
6. BMI
7. DiabetesPedigreeFunction
8. Age

Dataset url: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

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.

1) Pregnancies: Number of times pregnant

2) Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

3) BloodPressure: Diastolic blood pressure (mm Hg)

4) SkinThickness: Triceps skin fold 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)

## Step 1: Importing Libraries

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns

## Step 2: Load the Dataset

diabetes\_dataset = pd.read\_csv('diabetes.csv')

## Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

**3.1) Understanding Your Variables**

3.1.1) Head of the dataset  
 3.1.2) The shape of the dataset  
 3.1.3) List types of columns  
 3.1.4) Info of the dataset  
 3.1.5) Summary of the dataset

**3.1.1) Head of the Dataset**

# Display first five records  
diabetes\_dataset.head()

# Display last five records  
diabetes\_dataset.tail()

# Disply random records  
diabetes\_dataset.sample(5)

**3.1.2) The Shape of Dataset**

# Numbers of rows and columns   
diabetes\_dataset.shape

**3.1.3) List types of columns**

# List types of all columns  
diabetes\_dataset.dtypes

**3.1.4) Info of the Dataset**

# Checking for null values  
diabetes\_dataset.info()

# Statistical Summary  
diabetes\_dataset.describe()

# Step 4: Split the data frame in X & Y

target\_name = 'Outcome'  
  
# Separate object for target feature  
y = diabetes\_dataset[target\_name]  
  
# Separate obhect for input feature  
X = diabetes\_dataset.drop(target\_name,axis=1)

X.head()

y.head()

# Step 5: Apply Feature Scaling

Various Data Scaling Techniques:

1. Normalizer
2. MinMax Scaler
3. Binarizer
4. Standard Scaler

# Apply Standard Scaler  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
SSX = scaler.fit\_transform(X)

# Step 6: Train Test Split

from sklearn.model\_selection import train\_test\_split  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(SSX,y,test\_size=0.2,random\_state=2)

X\_train.shape,y\_train.shape

# Step 7: Build CLASSIFICATION Algorithms

**8.1) Logistic Regression**

from sklearn.linear\_model import LogisticRegression  
lr = LogisticRegression(solver='liblinear',multi\_class='ovr')  
lr.fit(X\_train,y\_train)

**8.2) K-Nearest Neighbors Classifier(KNN)**

from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier()  
knn.fit(X\_train,y\_train)

**8.3) Naive-Bayes Classifier**

from sklearn.naive\_bayes import GaussianNB  
nb = GaussianNB()  
nb.fit(X\_train, y\_train)

**8.4) Support Vector Machine (SVM)**

from sklearn.svm import SVC  
sv = SVC(kernel='linear')  
sv.fit(X\_train,y\_train)

**8.5) Decision Tree**

from sklearn.tree import DecisionTreeClassifier  
dt = DecisionTreeClassifier()  
dt.fit(X\_train,y\_train)

**8.6) Random Forest**

from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier(criterion='entropy')  
rf.fit(X\_train,y\_train)

# Step 8: Making Prediction

**9.1) Making Prediction using Logistic Regression**

lr\_pred = lr.predict(X\_test)

**9.2) Making Prediction using KNN**

knn\_pred = knn.predict(X\_test)

**9.3) Making Prediction using Naive Bayes**

nb\_pred = nb.predict(X\_test)

**9.4) Making Prediction using SVM**

sv\_pred = sv.predict(X\_test)

**9.5) Making Prediction using Decision Tree**

dt\_pred = dt.predict(X\_test)

**9.6) Making Prediciton using Random Forest**

rf\_pred = rf.predict(X\_test)

# Step 9: Model Evaluation

**10.1) Train & Test Scores**

from sklearn.metrics import accuracy\_score

# Train & Test Scores of Logistic Regression  
print("Accuracy (Train) score of Logistic Regression ",lr.score(X\_train,y\_train)\*100)  
print("Accuracy (Test) score of Logistic Regression ", lr.score(X\_test,y\_test)\*100)  
print("Accuracy score of Logistic Regression ", accuracy\_score(y\_test,lr\_pred)\*100)

# Train & Test Scores of KNN  
print("Accuracy (Train) score of KNN ",knn.score(X\_train,y\_train)\*100)  
print("Accuracy (Test) score of KNN ", knn.score(X\_test,y\_test)\*100)  
print("Accuracy score of KNN ", accuracy\_score(y\_test,knn\_pred)\*100)

# Train & Test Scores of Naive-Bayes  
print("Accuracy (Train) score of Naive Bayes ",nb.score(X\_train,y\_train)\*100)  
print("Accuracy (Test) score of Naive Bayes ", nb.score(X\_test,y\_test)\*100)  
print("Accuracy score of Naive Bayes ", accuracy\_score(y\_test,nb\_pred)\*100)

# Train & Test Scores of SVM  
print("Accuracy (Train) score of SVM ",sv.score(X\_train,y\_train)\*100)  
print("Accuracy (Test) score of SVM ", sv.score(X\_test,y\_test)\*100)  
print("Accuracy score of SVM ", accuracy\_score(y\_test,sv\_pred)\*100)

# Train & Test Scores of Decision Tree  
print("Accuracy (Train) score of Decision Tree ",dt.score(X\_train,y\_train)\*100)  
print("Accuracy (Test) score of Decision Tree ", dt.score(X\_test,y\_test)\*100)  
print("Accuracy score of Decision Tree ", accuracy\_score(y\_test,dt\_pred)\*100)

# Train & Test Scores of Random Forest  
print("Accuracy (Train) score of Random Forest ",rf.score(X\_train,y\_train)\*100)  
print("Accuracy (Test) score of Random Forest ", rf.score(X\_test,y\_test)\*100)  
print("Accuracy score of Random Forest ", accuracy\_score(y\_test,rf\_pred)\*100)

# Step 10: Making a prediction System

input\_data = (5,166,72,19,175,25.8,0.587,51)  
  
# changing 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)  
  
# standardize the input data  
std\_data = scaler.transform(input\_data\_reshaped)  
print(std\_data)  
  
prediction = lr.predict(std\_data)  
print(prediction)  
  
if (prediction[0] == 0):  
 print('The person is not diabetic')  
else:  
 print('The person is diabetic')

# Step 11: Saving the trained Model

import pickle  
filename = 'diabetes\_model.sav'  
pickle.dump(lr,open(filename,'wb'))