

# **DIABETES PREDICTION USING** **DATA-SCIENCE**

## **A PROJECT REPORT**

*In partial fulfilment of the requirements for the award of the degree*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**BY**

**Souhardya Gayen**



**NEOTIA INSTITUTE OF TECHNOLOGY,**  
**MANAGEMENT AND SCIENCE**  
**KOLKATA, WEST-BENGAL, INDIA**

**In association with**



**EXPOSYS DATA LABS**

## **CONTENTS:-**

### **1) ABSTRACT**

### **2) LIST OF COMMON MACHINE LEARNING ALGORITHM**

### **3) PROBLEM STATEMENT**

### **4) INTRODUCTION**

### **5) DETAILS OF THE PROJECT**

- a) Data Collection**
- b) About The Data**
- c) Proposed method**
- d) Architecture**

### **6) SYSTEM REQUIREMENTS**

- a) Software Requirements**
- b) Hardware Requirements**

### **7) MODULES USED IN THE PROJECT (with explanation)**

### **8) IMPLEMENTATION ( CODES WITH OUTPUTS )**

### **9) CONCLUSION**

## **ABSTRACT :**

**Diabetes** is an illness caused because of high glucose level in a human body. Diabetes should not be ignored if it is untreated then Diabetes may cause some major issues in a person like: heart related problems, kidney problem, blood pressure, eye damage and it can also affects other organs of human body. Diabetes can be controlled if it is predicted earlier. To achieve this goal this project work we will do early prediction of Diabetes in a human body or a patient for a higher accuracy through applying, Various Machine Learning Techniques. Machine learning techniques Provide better result for prediction by constructing models from datasets collected from patients. In this work we will use Machine Learning Classification and ensemble techniques on a dataset to predict diabetes. Which are K-Nearest Neighbours (KNN), Support Vector Machine (SVM), and Naive Bayes. The accuracy is different for every model when compared to other models. The Project work gives the accurate or higher accuracy model shows that the model is capable of predicting diabetes effectively. Our Result shows that Support Vector Machine and K-Nearest Neighbours achieved higher accuracy compared to other machine learning techniques.

## LIST OF COMMON MACHINE LEARNING ALGORITHMS:

Here is the list of commonly used machine learning algorithms. These algorithms can be applied to almost any data problem:

1. Linear Regression
2. Logistic Regression
3. Decision Tree
4. SVM
5. Naive Bayes
6. KNN (K-Nearest Neighbours)
7. K-Means
8. Random Forest
9. Dimensionality Reduction Algorithms.

There are also Gradient Boosting Algorithms:

1. GBM
2. XGBoost
3. Light GBM
4. CatBoost

### **1. Linear Regression**

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation  $Y = a * X + b$ .

The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these

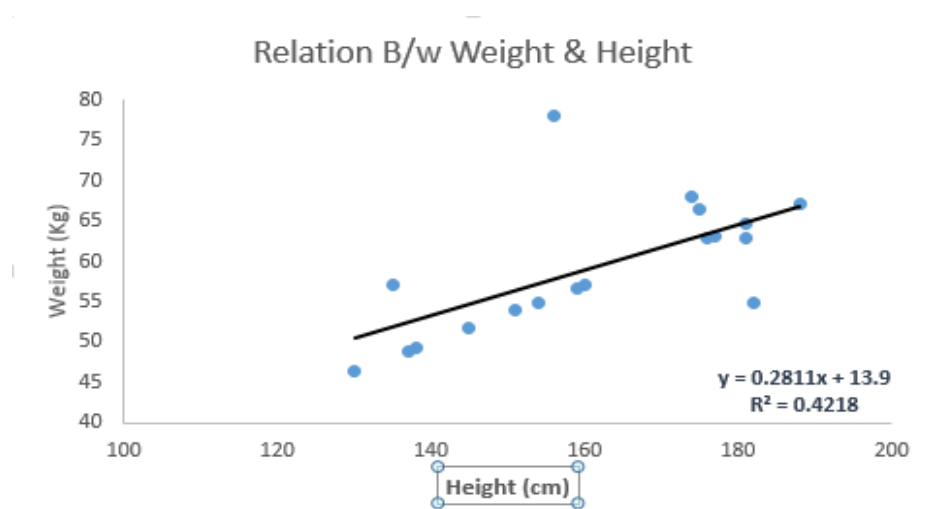
visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

Look at the below example. Here we have identified the best fit line having linear equation  $y=0.2811x+13.9$ . Now using this equation, we can find the weight, knowing the height of a person.



Linear Regression is mainly of two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent variables. While finding the best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression.

## 2. Logistic Regression

It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variables. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as **logit regression**. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

odds=  $p / (1-p)$  = probability of event occurrence / probability of not event occurrences

$$\ln(\text{odds}) = \ln(p/(1-p))$$

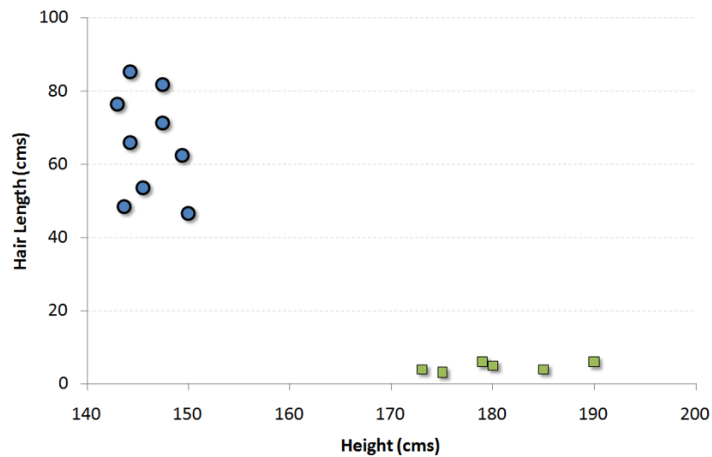
$$\text{logit}(p) = \ln(p/(1-p)) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \dots + b_kX_k$$

Above,  $p$  is the probability of presence of the characteristic of interest. It chooses parameters that maximize the likelihood of observing the sample values rather than that minimize the sum of squared errors (like in ordinary regression).

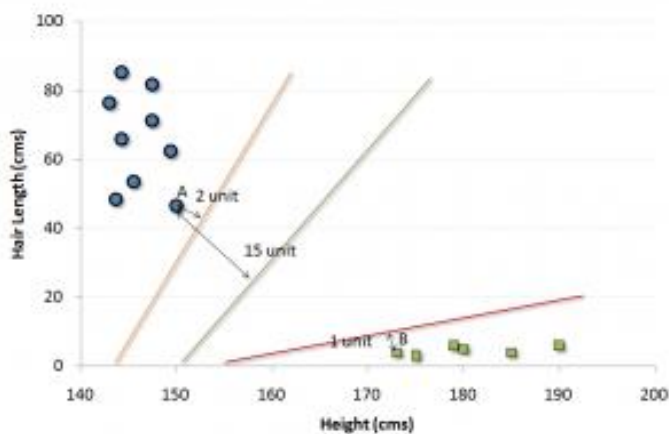
## 3. SVM (Support Vector Machine)

It is a classification method. In this algorithm, we plot each data item as a point in  $n$ -dimensional space (where  $n$  is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two-dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)



Now, we will find some *line* that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.

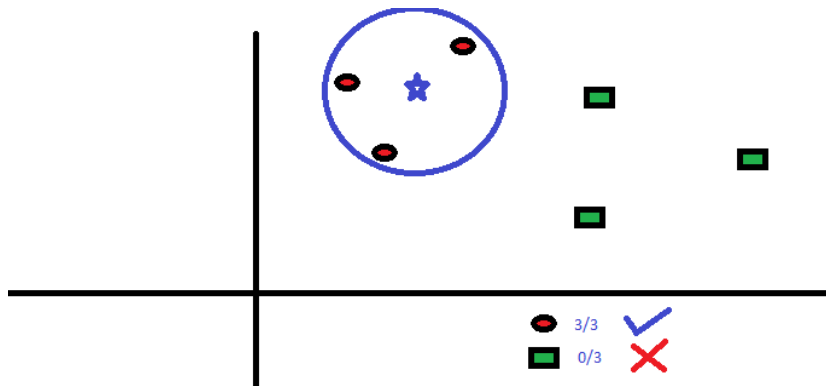


In the example shown above, the line which splits the data into two differently classified groups is the *black* line, since the two closest points are the farthest apart from the line. This line is our classifier. Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

#### 4. kNN (k- Nearest Neighbors)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If  $K = 1$ , then the case is simply assigned to the class of its nearest neighbor. At times, choosing  $K$  turns out to be a challenge while performing kNN modelling.



KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

### Things to consider before selecting kNN:

- KNN is computationally expensive
- Variables should be normalized else higher range variables can bias it
- Works on pre-processing stage more before going for kNN like an outlier, noise removal

## 5. Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is  $N$ , then sample of  $N$  cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are  $M$  input variables, a number  $m \ll M$  is specified such that at each node,  $m$  variables are selected at random out of the  $M$  and the best split on these  $m$  is used to split the node. The value of  $m$  is held constant



during the forest growing.

3. Each tree is grown to the largest extent possible. There is no pruning.

# **PROBLEM STATEMENT**

## **DIABETES PREDICTION USING DATA-SCIENCE**

### **INTRODUCTION**

Diabetes is a common chronic disease and poses a great threat to human health. The characteristic of diabetes is that the blood glucose is higher than the normal level, which is caused by defective insulin secretion or its impaired biological effects, or both (Lonappan et al., 2007). Diabetes can lead to chronic damage and dysfunction of various tissues, especially eyes, kidneys, heart, blood vessels and nerves (Krasteva et al., 2011). Diabetes can be divided into two categories, type 1 diabetes (T1D) and type 2 diabetes (T2D). Patients with type 1 diabetes are normally younger, mostly less than 30 years old. The typical clinical symptoms are increased thirst and frequent urination, high blood glucose levels (Iancu et al., 2008). This type of diabetes cannot be cured effectively with oral medications alone and the patients are required insulin therapy. Type 2 diabetes occurs more commonly in middle-aged and elderly people, which is often associated with the occurrence of obesity, hypertension, dyslipidemia, arteriosclerosis, and other diseases (Robertson et al., 2011).



With the development of living standards, diabetes is increasingly common in people's daily life. Therefore, how to quickly and accurately diagnose and analyze diabetes is a topic worthy studying. In medicine, the diagnosis of

diabetes is according to fasting blood glucose, glucose tolerance, and random blood glucose levels (Iancu et al., 2008; Cox and Edelman, 2009; American Diabetes Association, 2012). The earlier diagnosis is obtained, the much easier we can control it. Machine learning can help people make a preliminary judgment about diabetes mellitus according to their daily physical examination data, and it can serve as a reference for doctors (Lee and Kim, 2016; Alghamdi et al., 2017; Kavakiotis et al., 2017). For machine learning method, how to select the valid features and the correct classifier are the most important problems.

Recently, numerous algorithms are used to predict diabetes, including the traditional machine learning method (Kavakiotis et al., 2017), such as support vector machine (SVM), decision tree (DT), logistic regression and so on. Polat and Günes (2007) distinguished diabetes from normal people by using principal component analysis (PCA) and neuro fuzzy inference. Yue et al. (2008) used quantum particle swarm optimization (QPSO) algorithm and weighted least squares support vector machine (WLS-SVM) to predict type 2 diabetes Duygu and Esin (2011) proposed a system to predict diabetes, called LDA-MWSVM. In this system, the authors used Linear Discriminant Analysis (LDA) to reduce the dimensions and extract the features. In order to deal with the high dimensional datasets, Razavian et al. (2015) built prediction models based on logistic regression for different onsets of type 2 diabetes prediction. Georga et al. (2013) focused on the glucose, and used support vector regression (SVR) to predict diabetes, which is as a multivariate regression problem. Moreover, more and more studies used ensemble methods to improve the accuracy (Kavakiotis et al., 2017). Ozcift and Gulden (2011) proposed a newly ensemble approach, namely rotation forest, which combines 30 machine learning methods. Han et al. (2015) proposed a machine learning method, which changed the SVM prediction rules.

Machine learning methods are widely used in predicting diabetes, and they get preferable results. Decision tree is one of popular machine learning methods in medical field, which has grateful classification power. Random forest generates many decision trees. Neural network is a recently popular machine learning method, which has a better performance in many aspects. So in this study, we used k nearest neighbour, Support vector machine and Naive Bayes to predict the diabetes.

# **DETAILS OF THE PROJECT**

## **1. Data Collection.**

- The dataset required for this project was taken from “*kaggle.com*”.
- Kaggle allows users to find and publish datasets to solve challenges or web-based data science projects.

## **2. About the Data.**

- The dataset consists of 768 Rows and 9 Columns.
- The description of each Columns is as follows:-

### **Content**

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on. Here Outcome columns which is a target variable, is consist of 0's and 1's, Where 1 indicates the patient is Diabetic and 0 indicates the patient is not Diabetic.

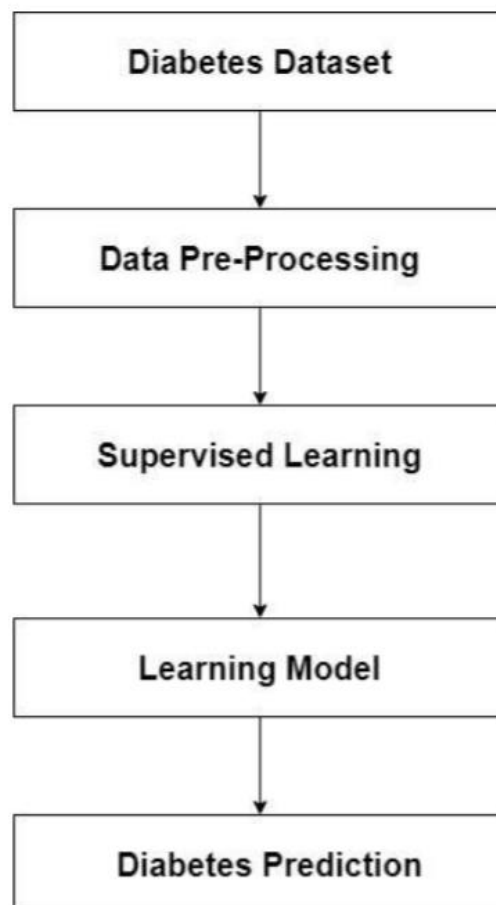
## **3. Proposed Methodology :**

In the conducted research the purpose is to classify the data available into diabetic or non-diabetic using the supervised learning algorithms. The dataset will be divided into training and testing sets. In order to achieve more accuracy we must train more data. Than we will to a comparative analysis on the results achieved from the algorithms for early detection of diabetes. The models like Support vector machine, Naive Bayes prove to be most useful in detection of diabetes in a patient. The centre objective of

our model is to achieve a better accuracy and overall improvement in early diagnosis of diabetes.

#### **4. Architecture :**

The system will detect whether the person has diabetes or not using the dataset. If diabetes is detected the classification value will be 1 and if not the value will be 0. We will be using 3 machine learning model in order to detect the disease. The models used are KNN, Support vector machine and Naive Bayes. The system architecture will help in the future too diagnosis diabetes. It will also predict whether someone has diabetes or not on the basis of using a trained dataset.



**Figure: Architecture Diagram**

# **SYSTEM REQUIREMENTS**

## **SOFTWARE REQUIREMENTS**

- **os: windows or linux**
- **python IDE:python 2.7.x and above**
- **jupyter notebook or Google Colab**
- **setup tools and pin to be installed for 3.6 and above**
- **language python**

## **HARDWARE REQUIREMENTS**

- **RAM:4GB and higher**
- **processor :intel i3 and above**
- **hard disk:500GB: minimum**

## **Modules used in the Project.**

- **NumPy :-** It is a general purpose array processing package.it provides a high performance multidimensional array object, and tools for working with these arrays .its also an efficient multidimensional container of generic data.
- **Pandas :-** It is the most popular python library that is used for data analysis.it provides highly optimized performance with back-end source code is purely written in **C** or **python** .subdivided into two parts it is :- **series and dataframe**.
- **Matplotlib :-** It is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environment across platforms .it tries to make easy things easy and hard things possible. you can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code.
- **Seaborn :-** It is a python data visualization library based on matplotlib.it provides a high level interface for drawing attractive and informative statistical graphics.it is closely integrated with pandas datastructures.
- **Scikit-learn :-** it is a free machine-learning library for python programming language.it features various classification ,regression and clustering algorithms including support vector machines ,random forests, gradient boosting ,k-means, DBSCAN and is designed to incorporate with the python numerical and scientific libraries numpy and scipy.

## **IMPLEMENTATION**

### **ACTUAL CODES WITH OUTPUT**

- **Importing necessary libraries :**

```
✓ 0s #Importing necessary libraries

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

### **Explanation:-**

We import the numpy, pandas, matplotlib and seaborn(provides high-level interface for attractive and informative statistical graphics).

- **Importing dataset :**

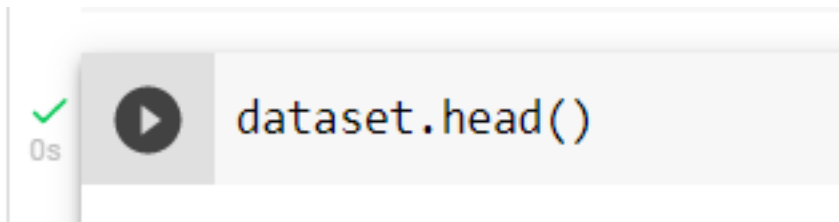
```
✓ 0s [4] #loading the Datasheet from the csv file using pandas
      dataset=pd.read_csv('diabetes.csv')
```

### **Explanation:-**

The csv file(dataset) is read using the “pd.read\_csv(‘filename)’” command.



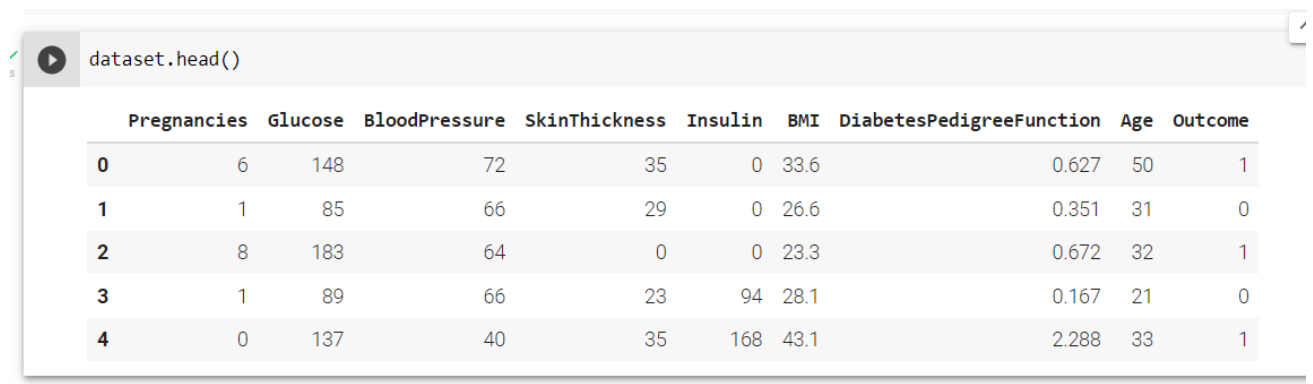
## • Understanding the Data :



### Explanation:-

head(), gives us a quick look at our dataset.

### Output :-



A screenshot of a code editor interface. On the left, there is a green checkmark and the text '0s'. In the center, there is a grey play button icon. To the right of the play button, the text 'dataset.head()' is displayed in a monospaced font.

|   | 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       |

## • Describing the Data :

```
[ ] #printing shape of the data  
dataset.shape
```

(768, 9)

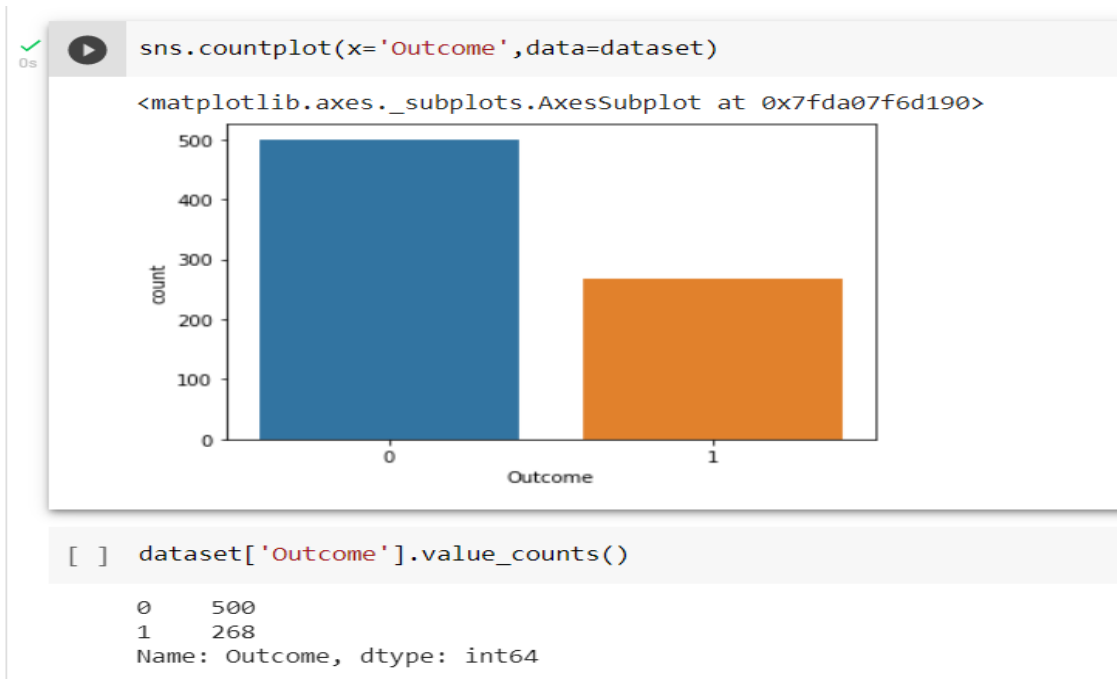
```
#describing the data  
dataset.describe()
```

|       | 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   |

## Explanation:-

Checking the shape (rows x columns) of the dataset.

- **Counting Values of Outcome:**



The table displays the mean values of various features grouped by the 'Outcome' variable. The code used to generate this table is:

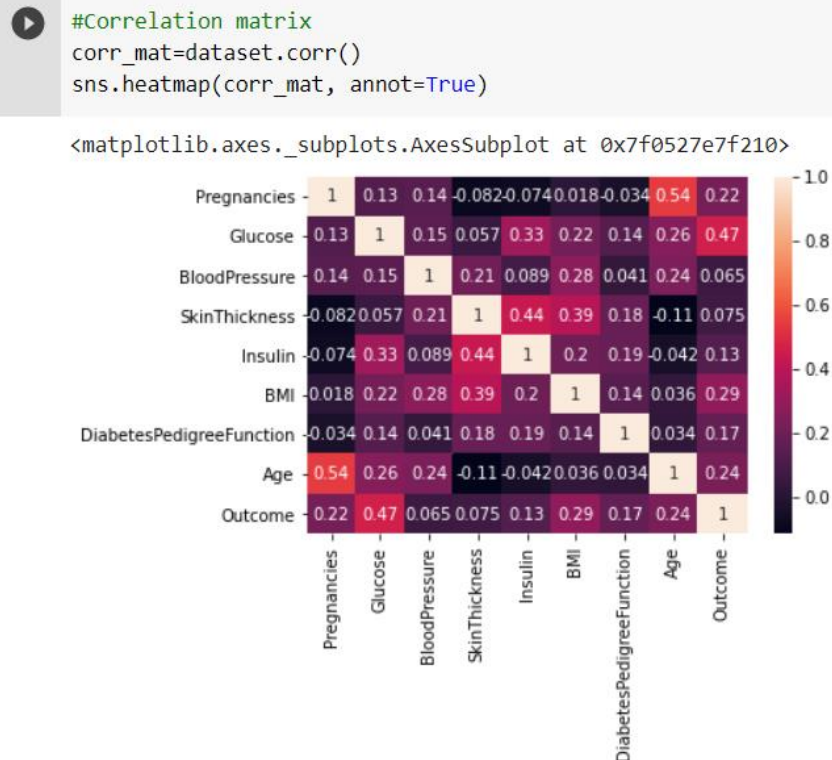
```
dataset.groupby('Outcome').mean()
```

|         | Pregnancies | Glucose    | BloodPressure | SkinThickness | Insulin    | BMI       | DiabetesPedigreeFunction | Age       |
|---------|-------------|------------|---------------|---------------|------------|-----------|--------------------------|-----------|
| Outcome |             |            |               |               |            |           |                          |           |
| 0       | 3.298000    | 109.980000 | 68.184000     | 19.664000     | 68.792000  | 30.304200 | 0.429734                 | 31.190000 |
| 1       | 4.865672    | 141.257463 | 70.824627     | 22.164179     | 100.335821 | 35.142537 | 0.550500                 | 37.067164 |

## Explanation:-

Counting values of outcomes having 0 or 1, 0 means non diabetic and 1 means diabetic.

- **Finding Correlation matrix :**



## Explanation:-

Correlation matrix to show correlation between two variables, 0.x means x% similar. For Example, correlation between Glucose and Outcome is 47% that means output depends majorly on Glucose.

- **Finding Null Values :**

```
dataset.isna().sum()
```

|                          |       |
|--------------------------|-------|
| Pregnancies              | 0     |
| Glucose                  | 0     |
| BloodPressure            | 0     |
| SkinThickness            | 0     |
| Insulin                  | 0     |
| BMI                      | 0     |
| DiabetesPedigreeFunction | 0     |
| Age                      | 0     |
| Outcome                  | 0     |
| dtype:                   | int64 |

## Explanation:-

For finding the number of missing values in a dataset we use `dataset.isnull()`. We call `dataset.isnull().sum()` to give the output of series containing data about count of NaN in each column.

We get no null values...thus the data set has no missing values.

### • Creating Feature Matrix :

✓  
0s



```
x=dataset.iloc[:, :-1].values #Independent matrix  
y=dataset.iloc[:, -1].values #Dependent
```

### Shape of x :

✓  
0s

```
[14] x.shape
```

```
(768, 8)
```

✓  
0s

```
x[0]
```

```
array([ 6.    , 148.   , 72.    , 35.    , 0.    , 33.6   , 0.627,  
       50.    ])
```

## Explanation:-

Taking all our independent columns into single array and dependent values into another array.

## • Exploratory Data Analysis :

CHECKING WHICH COLUMNS ARE USEFUL OR NOT -

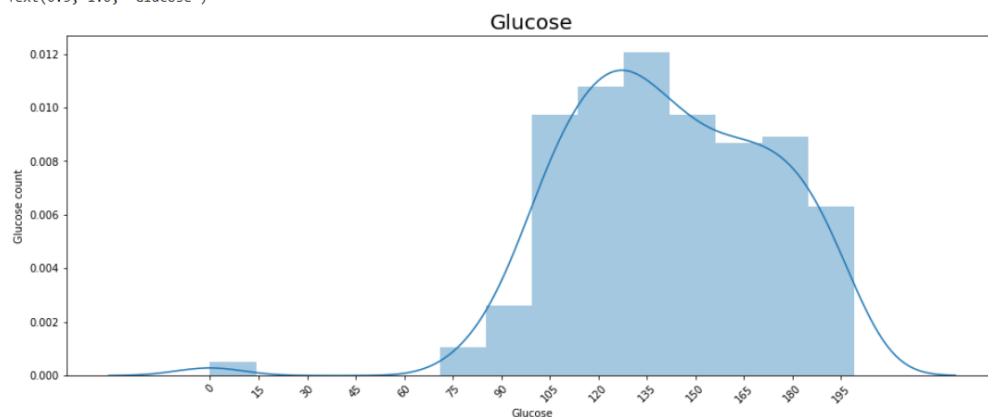
### Glucose for diabetic :

```
✓ 0s ▶ fig = plt.figure(figsize=(16,6))

sns.distplot(dataset["Glucose"][dataset["Outcome"] == 1])
plt.xticks([i for i in range(0,201,15)],rotation = 45)
plt.ylabel("Glucose count")
plt.title("Glucose",fontsize = 20)
```

### Output :

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version.  
warnings.warn(msg, FutureWarning)  
Text(0.5, 1.0, 'Glucose')



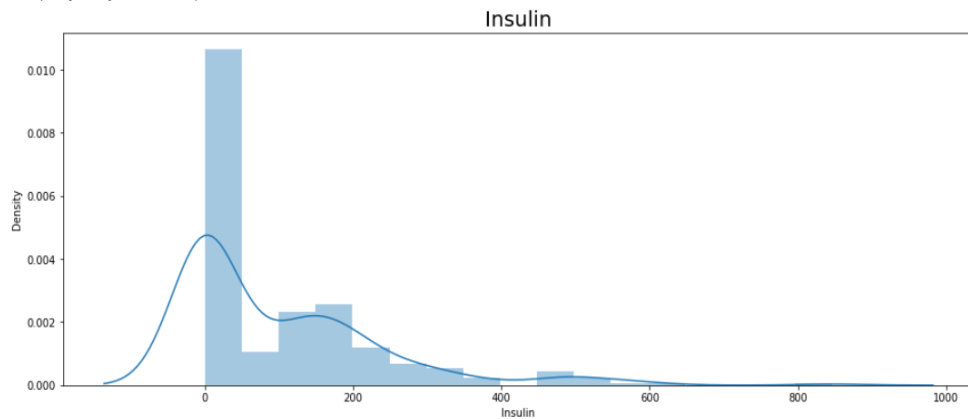
### Insulin for diabetic :

```
✓ 0s ▶ fig = plt.figure(figsize = (16,6))

sns.distplot(dataset["Insulin"][dataset["Outcome"]==1])
plt.xticks()
plt.title("Insulin",fontsize = 20)
```

## Output :

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version.  
warnings.warn(msg, FutureWarning)  
Text(0.5, 1.0, 'Insulin')
```



## BMI for diabetic:

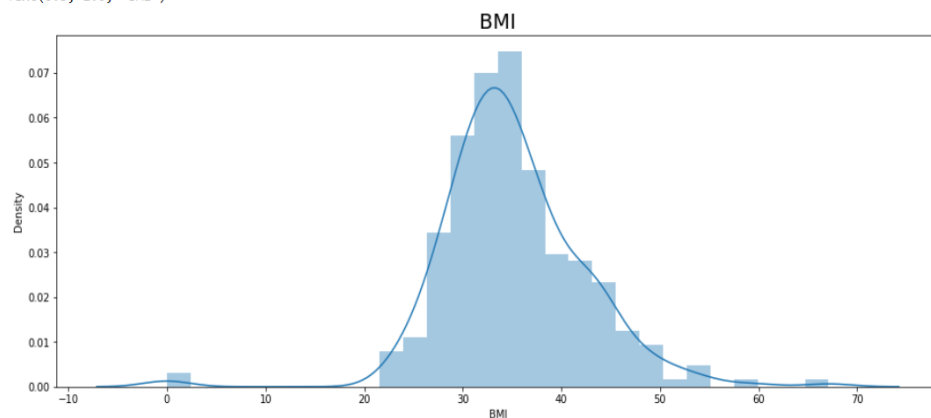
✓  
2s



```
fig = plt.figure(figsize =(16,6))  
  
sns.distplot(dataset["BMI"][dataset["Outcome"]==1])  
plt.xticks()  
plt.title("BMI",fontsize = 20)
```

## Output :

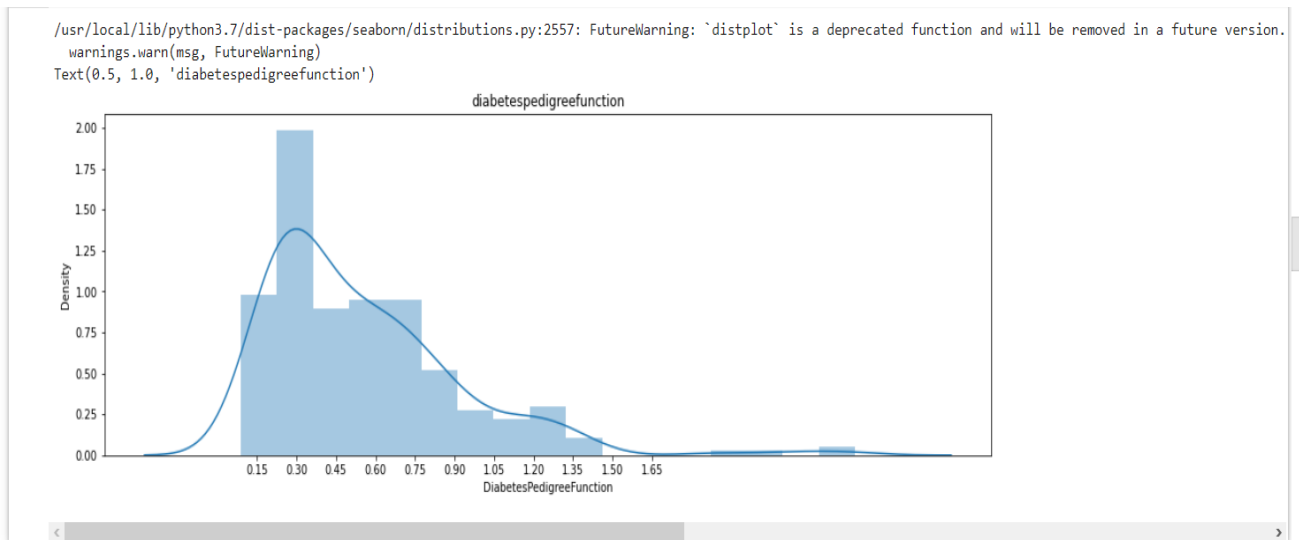
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version.  
warnings.warn(msg, FutureWarning)  
Text(0.5, 1.0, 'BMI')
```



## Diabeticpedigreefunction for diabetic :

```
✓ 0s ▶ fig = plt.figure(figsize = (16,5))
sns.distplot(dataset["DiabetesPedigreeFunction"][dataset["Outcome"] == 1])
plt.xticks([i*0.15 for i in range(1,12)])
plt.title("diabetespedigreefunction")
```

## Output:

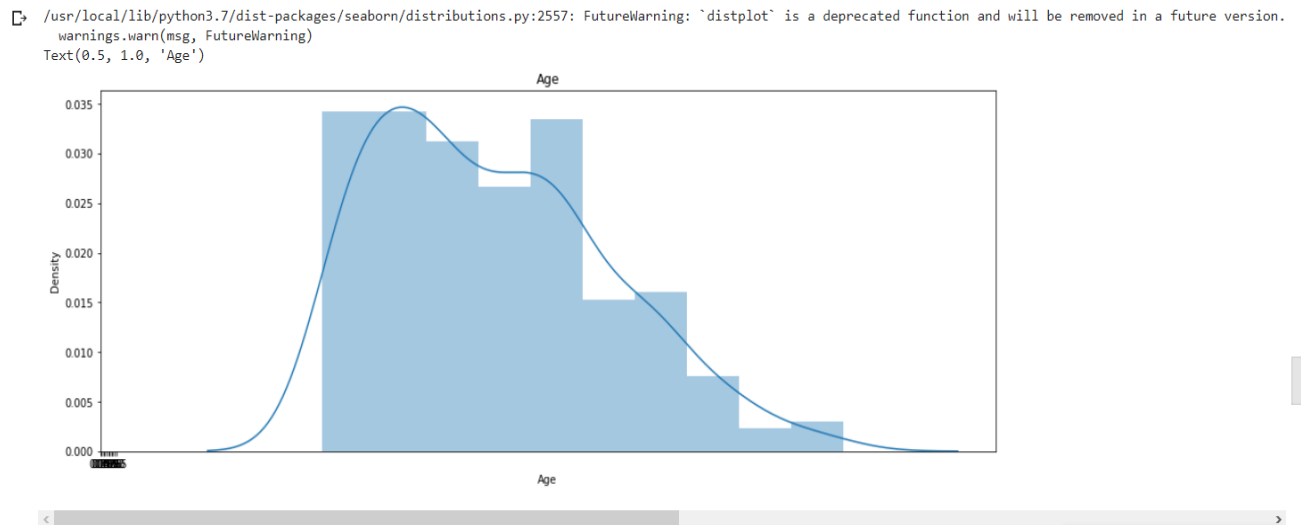


## Age for diabetic :

```
✓ 0s ▶ fig = plt.figure(figsize = (16,6))

sns.distplot(dataset["Age"][dataset["Outcome"] == 1])
plt.xticks([i*0.15 for i in range(1,12)])
plt.title("Age")
```

## Output:



## Explanation:-

This Process is very important for data cleaning, as we can see, it shows which variables are participating how much in the outcome, so from that we can drop those variables which are not participating that much on the outcome.

- **Removing unnecessary columns:**

```
✓ 0s ▶ x = dataset.drop(["Pregnancies", "BloodPressure", "SkinThickness", "Outcome"], axis = 1)
      y = dataset.iloc[:, -1]
```



## Explanation:-

Dropping the "Pregnancies", "BloodPressure", "SkinThickness", "Outcome" columns as they have less participation on the outcome than other, so that our dataset will be clear and our output will be more accurate.

- **Splitting dataset :**

```
✓ [24] from sklearn.model_selection import train_test_split
0s      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
      #test_size 0.2 means for testing data 20% and training data 80%

✓ [25] x_train.shape
0s      (614, 5)

✓ [26] x_test.shape
0s      (154, 5)
```

## Explanation:-

Splitting dataset into training set and test set. test\_size 0.2 means for testing data 20% and training data 80%. Training set size is 80% of original dataset (769,9) after removing unnecessary data and Test set size is 20% of original dataset (769,9) after removing unnecessary data.

- **Feature Scaling :**

```
✓ [29] from sklearn.preprocessing import StandardScaler  
0s sc = StandardScaler()  
x_train = sc.fit_transform(x_train)  
x_test = sc.transform(x_test)
```

```
✓ x_train  
0s array([[ 0.91569367,  0.3736349 ,  0.37852648,  0.67740401,  1.69955804],  
        [-0.75182191, -0.69965674, -0.50667229, -0.07049698, -0.96569189],  
        [ 1.38763205,  5.09271083,  2.54094063, -0.11855487, -0.88240283],  
        ...,  
        [-0.84620959, -0.69965674, -0.94927168, -0.95656442, -1.04898095],  
        [-1.12937261, -0.69965674, -0.26640405, -0.50001442,  0.11706589],  
        [ 0.47521786, -0.69965674, -4.07275877,  0.52121586,  2.94889395]])
```

## Explanation:-

It's needed to standardize the independent features present in the data in a fixed range. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

- **Model Building -**



## K Nearest Neighbour :

### Building a KNN Model using skicit learn

```
✓ [29] from sklearn.neighbors import KNeighborsClassifier  
0s knn = KNeighborsClassifier(n_neighbors=25, metric='minkowski')  
#n_neighbors is 25 bcoz for x_train we got 614 which is near to 25^2  
#metric means on what factor choosing so as its KNN so our metric is minkowski i.e., distance  
knn.fit(x_train, y_train)
```

```
➤ KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                       metric_params=None, n_jobs=None, n_neighbors=25, p=2,  
                       weights='uniform')
```

## Explanation:-

n\_neighbors is 25 bcoz for x\_train we got 614 which is near to  $25^2$

Metric means on what factor choosing so as its KNN so our metric is minkowski i.e., distance

### ➤ Predicting the data

```
✓ [34] knn_y_pred = knn.predict(x_test)
```

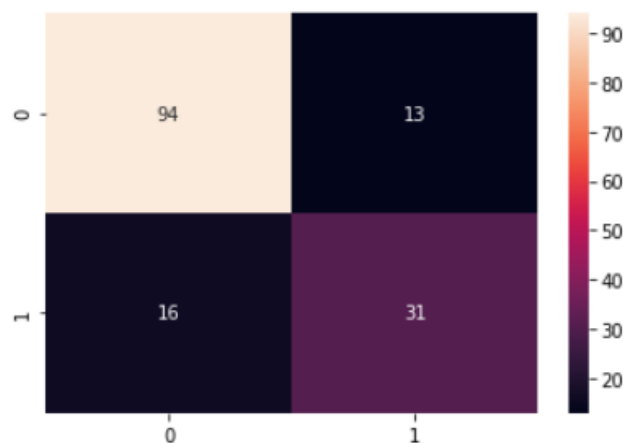
```
✓ knn_y_pred
```

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

### ➤ Visualizing Confusion matrix

```
✓ # Confusion matrix
from sklearn.metrics import confusion_matrix
knn_cm = confusion_matrix(y_test, knn_y_pred)
sns.heatmap(knn_cm, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f78c40dd0>
```



## Explanation:-

Its needed To check how many are correct or wrong .

The above heatmap says 0,0 means true negative and 1,1 means true positive and 0,1 means even person is negative but showing result positive and 1,0 means person is positive but shows negative so its danger so we need to accurate our model.

## ➤ Evaluating “correct” , “Incorrect” and “Accuracy”

```
✓ [38] print("Correct:",sum(knn_y_pred==y_test))  
0s      print("Incorrect : ",sum(knn_y_pred != y_test))  
      print("Accuracy:",sum(knn_y_pred ==y_test)/len(knn_y_pred))
```

```
Correct: 125  
Incorrect : 29  
Accuracy: 0.8116883116883117
```

```
✓ #Verfying accuracy using inbuilt methods  
0s from sklearn.metrics import accuracy_score  
    accuracy_score(y_test,knn_y_pred)
```


```
0.8116883116883117
```

## Explanation:-

We evaluate the parameters we needed the accuracy we got from the KNN is – 81.68%.

## Support Vector Machine :


### Building a SVM Model using skicit learn

```
✓ 0s  from sklearn.svm import SVC
svc=SVC(kernel="linear",random_state=0)
svc.fit(x_train,y_train)

SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```


### ➤ Predicting the data

```
✓ 0s [45] svc_y_pred = svc.predict(x_test)
```

```
✓ 0s  svc_y_pred

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, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 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, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

### ➤ Confusion matrix :

```
✓ 0s  svc_cm = confusion_matrix(y_test,svc_y_pred)
print(svc_cm)
```

```
[[96 11]
 [18 29]]
```

## ➤ Evaluating “correct” , “Incorrect” and “Accuracy”

```
✓ [49] print("Correct:",sum(svc_y_pred == y_test))  
0s      print("Incorrect : ",sum(svc_y_pred != y_test))  
      print("Accuracy:",sum(svc_y_pred ==y_test)/len(svc_y_pred))
```

```
Correct: 125  
Incorrect : 29  
Accuracy: 0.8116883116883117
```

```
✓ #Verfying accuracy using inbuilt methods  
0s from sklearn.metrics import accuracy_score  
    accuracy_score(y_test,svc_y_pred)
```

```
📄 0.8116883116883117
```

## Explanation:-

We evaluate the parameters we needed the accuracy we got from the SVM is same as KNN that is – 81.68%.



## Naive Bias :

### Building a Naïve Bias Model using skicit learn

```
✓ [44] from sklearn.naive_bayes import GaussianNB  
0s      nb_classifier = GaussianNB()  
      nb_classifier.fit(x_train,y_train)  
  
      GaussianNB(priors=None, var_smoothing=1e-09)
```

### ➤ Predicting the data

```
✓ [51] nb_y_pred =nb_classifier.predict(x_test)  
0s
```

```
✓ nb_y_pred  
0s  
array([[1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,  
        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,  
        1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,  
        1, 0, 0, 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, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,  
        0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

### ➤ Confusion matrix :

```
nb_cm = confusion_matrix(nb_y_pred,y_test)  
print(nb_cm)
```

```
[[94 21]  
 [13 26]]
```

## ➤ Evaluating “correct” , “Incorrect” and “Accuracy”

```
✓ [47] print("Correct:",sum(nb_y_pred == y_test))  
0s      print("Incorrect : ",sum(nb_y_pred != y_test))  
      print("Accuracy:",sum(nb_y_pred ==y_test)/len(nb_y_pred))
```

```
Correct: 120  
Incorrect : 34  
Accuracy: 0.7792207792207793
```

### Explanation:-

We evaluate the parameters we needed the accuracy we got from the Naïve bias is lesser than KNN and SVM that is – 77.92%.

So as we can see Naïve Bias giving the accuracy lesser than SVM and KNN, and SVM and KNN both are giving same Accuracy.

| Algorithm  | Accuracy |
|------------|----------|
| KNN        | 81.68%   |
| SVM        | 81.68%   |
| Naïve Bias | 77.92%.  |

So we will not take the Naïve bias Classifier as its giving the less accuracy. So we can use either SVM classifier or KNN classifier to predict Diabetes using some random Values, as



they are giving same accuracy, I am using SVM classifier to predict Diabetes with some random Values.

## Predicting Diabetes with some random values :

### ➤ Building SVM Classifier :

```
✓ 0s from sklearn.svm import SVC
svc=SVC(kernel="linear",random_state=0)
svc.fit(x_train,y_train)

➤ SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```

### ➤ Taking the random data and Predicting Diabetes :

#### ❖ Input 1:

```
✓ 0s #Predicting Diabetes with some random values
# input_data fromat = (Glucose level , Insulin, BMI, Daibetes PF, Age)
input_data = ( 85 , 0 , 26.6 , 0.351 , 31 )

# 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 = sc.transform(input_data_reshaped)
print(std_data)

prediction = svc.predict(std_data)
print(prediction)

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

➤ [[-1.12937261 -0.69965674 -0.70900344 -0.34682988 -0.21609035]]
[0]
The person is not diabetic
```

## ❖ Input 2:

```
#Predicting Diabetes with some random values
# input_data format = (Glucose level , Insulin, BMI, Diabetes PF, Age)
#input_data = ( 85 , 0 , 26.6 , 0.351 , 31 )

input_data = ( 145 , 0 , 44.2 , 0.63 , 64 )
|
# 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 = sc.transform(input_data_reshaped)
print(std_data)

prediction = svc.predict(std_data)
print(prediction)

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

[[ 0.75838088 -0.69965674 1.51663919 0.49117967 2.53244865]]  
[1]  
The person is diabetic

***This is our desired Result.***

## **Explanation:-**

- Predicting Diabetes with some random values, The input\_data format = (Glucose level , Insulin, BMI, Diabetes PF, Age).
- Next we will change the input\_data to numpy array, then reshape the array as we are predicting for one instance.
- After that we standardize the input data.

- Then we used SVM classifier to predict the Diabetes using the input\_data.
- So when the prediction value is '1', it will show "The person is Diabetic" and if its '0' then it will show "The person is not diabetic".

## **CONCLUSION:**

In this Project Study, we have used different types of machine learning algorithm for detection of diabetes. We implemented machine learning algorithms on the dataset and performed classification to signify the best machine learning algorithm for diabetes prediction on the bases of old data available. The higher accuracy the better prediction rate we will achieve. The SVM and KNN algorithm obtained the best accuracy. While on the other hand Naïve Bias had the lowest score.

The overall experimentation displayed that KNN and SVM is better than other algorithms (Naive Bias) in diabetes prediction. In our future work we will work on the diagnosis of diabetes in young adults.

---

THANK YOU