Machine Learning model to detect weather the person is diabetic or not

M.Tech(Computational Mathematics)

Minor Project Dissertation

Submitted

To

Department of Applied Sciences & Humanities

Faculty of Engineering and Technology

Jamia Millia Islamia

New Delhi-110025

By

Yasar Beg

M.Tech(3rd Semester)2023-2024

Enrollment no. 22-003

Abstract: This machine learning project focuses on developing an effective predictive model for the early detection of diabetes using a comprehensive dataset. The goal is to create a tool that can assist healthcare professionals in identifying individuals at risk of diabetes, facilitating timely intervention and management.

1.Objective: The primary objective of this project is to leverage machine learning techniques to build a predictive model capable of accurately assessing the likelihood of diabetes in individuals. By analysing relevant health parameters, including age, BMI, blood pressure, and glucose levels, the model aims to contribute to proactive healthcare strategies for diabetes prevention.

2.Dataset: The dataset utilized in this project is derived from authoritative health sources and encompasses a diverse set of features related to individuals' health. Features include age, BMI, number of pregnancies, insulin levels, blood pressure, and glucose concentration. Rigorous preprocessing has been conducted to address missing values, normalize numerical features, and encode categorical variables.

Methods:

1.Exploratory Data Analysis (EDA): Conducting a comprehensive exploration of the dataset to uncover patterns, relationships, and potential insights. This step is crucial for understanding the characteristics of individuals with diabetes and those without.

2.Data Preprocessing: Preparing the dataset for machine learning by addressing missing values, normalizing numerical features, and encoding categorical variables. This step ensures the quality of input data for subsequent model training.

3.Model Selection: Experimenting with various machine learning algorithms suitable for binary classification tasks. Algorithms considered include Logistic Regression, Random Forest, Support Vector Machines, and Neural Networks. The performance of each model is evaluated using metrics such as accuracy, precision, recall, and F1 score.

4.Model Training: Training the selected model(s) on the pre-processed dataset. Techniques such as cross-validation are employed to ensure the model's robustness and prevent overfitting to the training data

5.Hyperparameter Tuning: Optimizing the hyperparameters of the chosen model(s) to enhance their performance. This involves systematically adjusting parameters to achieve the best possible predictive accuracy.

6.Evaluation: Assessing the model's performance on a separate test dataset, using a range of metrics to measure its accuracy, sensitivity, specificity, and overall effectiveness. Comparisons with baseline models provide valuable insights into the model's efficacy.

7.Results: Presenting the outcomes of the trained model, including its accuracy on the test dataset and a detailed analysis of key metrics. Visualizations such as confusion matrices enhance the interpretability of the model's performance.

8.Conclusion: Summarizing the key findings of the project and emphasizing the potential impact of the developed model on diabetes prediction. Discussion of limitations and future improvements is included, highlighting the need for further validation in real-world healthcare scenarios.

9. Deployment: If successful, the model can be deployed as part of a decision support system for healthcare professionals. Integration with electronic health records or mobile applications could enhance accessibility and contribute to proactive diabetes management.

10.Keywords: Machine Learning, Diabetes Prediction, Binary Classification, Feature Engineering, Model Evaluation, Healthcare, Data Science.

About Data

1. **Pregnancies:** Pregnancy can have a significant impact on diabetes, and the relationship between the two is complex. There are two main scenarios to consider: gestational diabetes and the impact of pre-existing diabetes on pregnancy.

Pre-existing Diabetes and Pregnancy:

Definition: Gestational diabetes mellitus (GDM) is a type of diabetes that develops during pregnancy, typically around the 24th week. It affects the way the body uses insulin and can lead to increased blood sugar levels.

Risk Factors: Women who are older, overweight, have a family history of diabetes, or belong to certain ethnic groups are at a higher risk of developing gestational diabetes.

Effect on Pregnancy: If not managed properly, gestational diabetes can lead to complications for both the mother and the baby. Complications may include an increased risk of preeclampsia, cesarean delivery, and macrosomia (large birth weight), among others.

Post-Pregnancy Implications: Women who have had gestational diabetes have a higher risk of developing type 2 diabetes later in life.

Pre-existing Diabetes and Pregnancy:

Type 1 Diabetes and Pregnancy: Women with type 1 diabetes need to carefully manage their blood sugar levels before and during pregnancy. Poorly controlled diabetes during pregnancy can lead to an increased risk of miscarriage, birth defects, and other complications.

Type 2 Diabetes and Pregnancy: Women with type 2 diabetes are often advised to manage their blood sugar levels before becoming pregnant. Similar to type 1 diabetes, uncontrolled type 2 diabetes during pregnancy can lead to complications.

Risk Factors: The risks associated with pre-existing diabetes during pregnancy can be mitigated with good blood sugar control, regular monitoring, and medical supervision.

2. Glucose: Without sufficient insulin, glucose cannot enter cells to be used for energy. As a result, glucose accumulates in the bloodstream, leading to hyperglycaemia (high blood sugar).

3. Blood Pressure:

Diabetes and blood pressure are interconnected, and individuals with diabetes are at an increased risk of developing high blood pressure (hypertension). The relationship between diabetes and blood pressure is bidirectional, meaning that each condition can influence and exacerbate the other. Here are some key points regarding the interplay between diabetes and blood pressure:

Increased Risk in Diabetes:

Insulin Resistance: In type 2 diabetes, insulin resistance may contribute to the development of hypertension. Insulin resistance can affect blood vessel function and increase the workload on the heart, leading to elevated blood pressure.

Inflammation: Chronic inflammation associated with diabetes can contribute to blood vessel damage and stiffness, contributing to hypertension.

Effect of High Blood Pressure on Diabetes:

Cardiovascular Risk: Hypertension is a significant risk factor for cardiovascular diseases. Individuals with both diabetes and high blood pressure have a higher risk of heart attacks, strokes, and other cardiovascular complications.

Impact on Kidneys: High blood pressure can also contribute to kidney damage, and individuals with diabetes are already at an increased risk of diabetic nephropathy (kidney disease).

Synergistic Effect:

Vicious Cycle: Diabetes and high blood pressure often create a vicious cycle. Uncontrolled diabetes can contribute to the development or worsening of hypertension, and vice versa.

Complications: The combination of diabetes and high blood pressure increases the risk of complications such as coronary artery disease, peripheral arterial disease, retinopathy (eye problems), and nephropathy (kidney problems).

Management Strategies:

Blood Pressure Control: Individuals with diabetes are advised to monitor and control their blood pressure to reduce the risk of cardiovascular complications. The target blood pressure may vary, but it is often recommended to keep it below 130/80 mmHg.

Lifestyle Modifications: Adopting a healthy lifestyle, including a balanced diet, regular physical activity, weight management, and reducing salt intake, can help control both blood pressure and diabetes.

Medications: In some cases, medications to lower blood pressure, such as angiotensin-converting enzyme (ACE) inhibitors or angiotensin II receptor blockers (ARBs), may be prescribed to individuals with diabetes to help manage both conditions.

4.Skin Thickness:

The relationship between diabetes and skin thickness is not a direct or well-established connection in the context of diabetes management or diagnosis. Diabetes primarily affects blood sugar levels and insulin function, and its impact on the skin is more related to complications and associated conditions rather than the thickness of the skin itself.

However, diabetes can have implications for skin health, and certain skin-related issues may arise in individuals with diabetes. Here are some ways in which diabetes may be associated with skin-related concerns:

Skin Complications in Diabetes:

Dry Skin: People with diabetes are more prone to dry skin, which can lead to itching and discomfort. This may be due to factors such as dehydration and poor circulation.

Infections: Diabetes can affect the immune system, making individuals more susceptible to skin infections. Common skin infections include bacterial and fungal infections.

Delayed Wound Healing: High blood sugar levels can impair the body's ability to heal wounds, increasing the risk of infections and other complications.

Diabetic Dermopathy:

Skin Changes: Some individuals with diabetes may develop a skin condition known as diabetic dermopathy, which is characterized by light brown, scaly patches on the skin. The exact cause is not well understood, but it is believed to be related to changes in blood vessels.

Acanthosis Nigricans:

Skin Darkening and Thickening: Acanthosis nigricans is a skin condition often associated with insulin resistance, a precursor to type 2 diabetes. It can cause dark, thickened patches of skin, typically in the neck, armpits, or groin.

5.BMI: Body Mass Index (BMI) is a measure of body fat based on an individual's weight and height. While there is a correlation between BMI and the risk of developing type 2 diabetes, the relationship is complex, and other factors such as distribution of body fat and genetics also play a role. Here are some key points regarding the relationship between diabetes and BMI

BMI=(Height in meters)/Weight in kilograms

6. Diabetes Pedigree Function: The Diabetes Pedigree Function (DPF) is not a commonly used term or parameter in the context of diabetes. It's possible that there might be a misunderstanding or confusion with another term or concept related to diabetes.

One widely recognized tool in diabetes research and assessment of genetic risk is the Diabetes Pedigree Function (DPF) used in the Diabetes Prediction and Genetic Risk Score (GRS) calculations. However, it's important to note that the Diabetes Pedigree Function is part of a larger formula for predicting the risk of diabetes in relatives of affected individuals, rather than being a standalone metric.

The formula for the Diabetes Pedigree Function (DPF) is used as part of the diabetes risk assessment in the context of family history. It takes into account the number of affected relatives and their relationship to the individual being assessed. The formula is as follows:

PDF=∑(Number of affected relatives2degree of relationship)/∑(2degree of relationshipNumber of aff ected relatives)

Here, the sum is taken over all affected relatives, and the degree of relationship is determined based on the closeness of the familial relationship (e.g., first-degree relatives like parents or siblings have a degree of relationship of 1, second-degree relatives like grandparents or aunts/uncles have a degree of relationship of 2, and so on).

This information is often used in combination with other factors to estimate the genetic risk of developing diabetes. However, it's crucial to understand that risk prediction for diabetes involves multiple factors, including genetics, lifestyle, and environmental factors. If you are interested in understanding your risk for diabetes or managing the condition, it is recommended to consult with a healthcare professional who can provide personalized advice based on a thorough assessment of your health status and risk factors.

7.AGE: Age is a significant factor in the development and management of diabetes. The relationship between diabetes and age can be observed in different ways, and both type 1 and type 2 diabetes are influenced by age-related factors:

Type 1 Diabetes and Age:

Onset: Type 1 diabetes is often diagnosed in childhood or adolescence, although it can occur at any age. The majority of new diagnoses occur in individuals under the age of 30, with a peak incidence in childhood and early adulthood.

Autoimmune Component: Type 1 diabetes is an autoimmune condition where the immune system attacks and destroys the insulin-producing beta cells in the pancreas. The onset is typically unrelated to lifestyle factors and is more influenced by genetic and environmental factors.

Type 2 Diabetes and Age:

Onset: The risk of developing type 2 diabetes increases with age. It is more commonly diagnosed in adults, particularly in those over the age of 45. However, the incidence of type 2 diabetes in younger populations, including children and adolescents, has been rising due to factors such as obesity and sedentary lifestyles.

Insulin Resistance: Aging is associated with an increased risk of insulin resistance, a condition where the body's cells become less responsive to insulin. This contributes to the development of type 2 diabetes.

Lifestyle Factors: While age itself is a risk factor, lifestyle factors such as poor diet, lack of physical activity, and obesity also play a role in the development of type 2 diabetes.

Gestational Diabetes and Age:

Pregnancy-Related Diabetes: Gestational diabetes is a form of diabetes that occurs during pregnancy. Advanced maternal age is considered a risk factor for gestational diabetes.

MODEEL BUILDING

In [8]:
```
#Let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

In [9]:
```
#read the data file
data =pd.read_csv("diabetes.csv")
data.head()
```

Out[9]:

	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

In [10]:
```
data.isnull().sum()
```

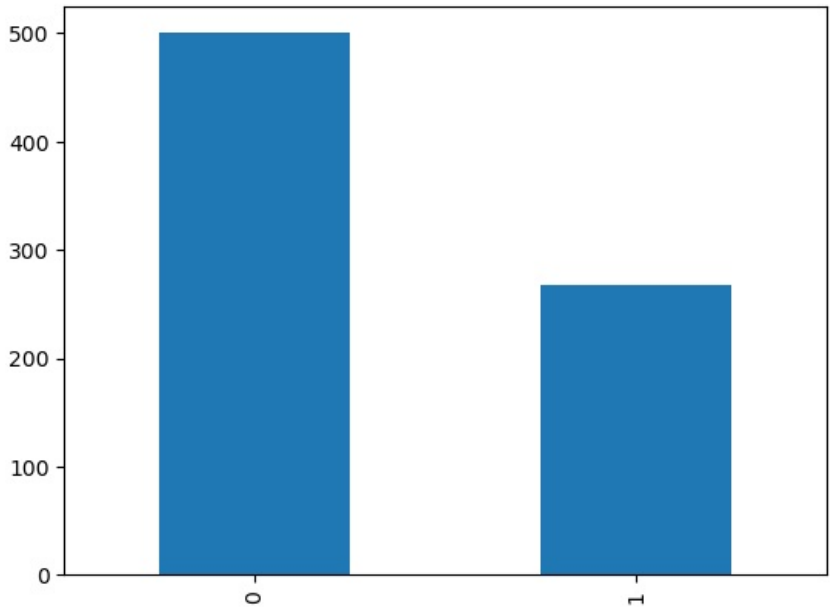
Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

EDA

1.Univariate Analysis on categorical feature

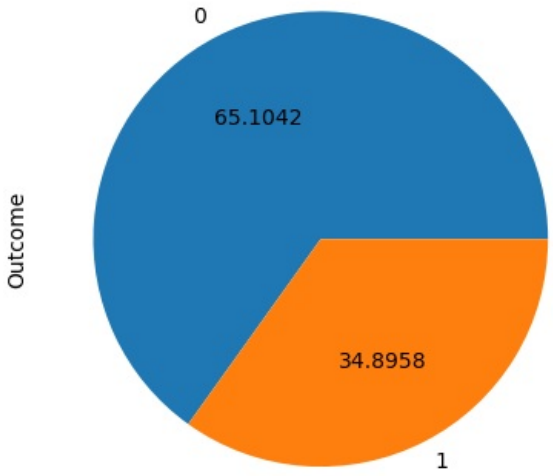
In [11]:
```
#sns.countplot(data['Outcome'])
data['Outcome'].value_counts().plot(kind='bar')
```

Out[11]:
```
<Axes: >
```



In [12]:
```
data['Outcome'].value_counts().plot(kind='pie',autopct='%.4f')
```

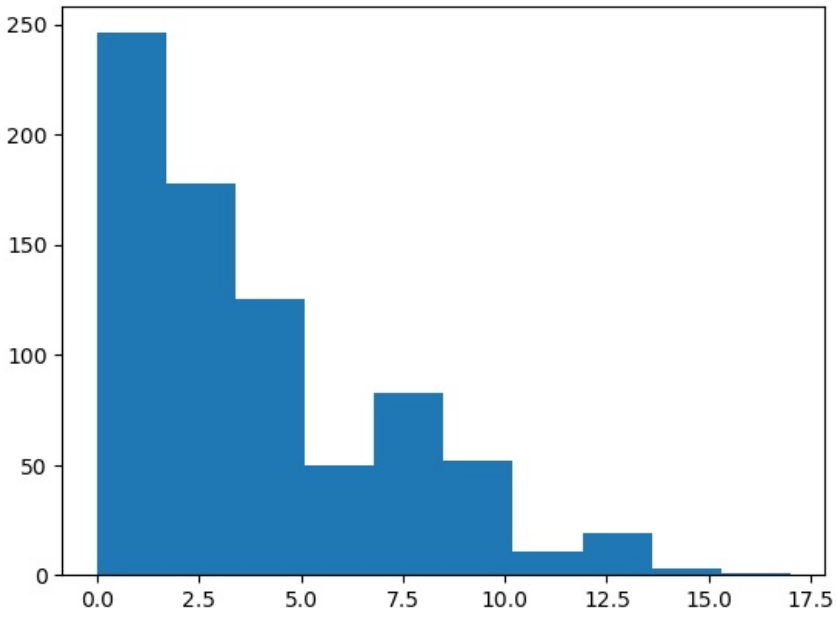
Out[12]:
```
<Axes: ylabel='Outcome'>
```



On numarical feature

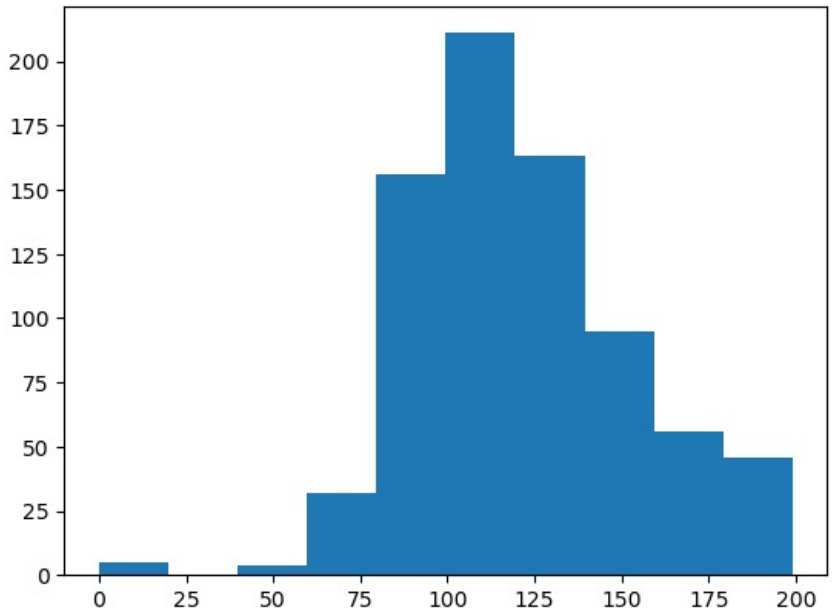
In [13]:
```
plt.hist(data['Pregnancies'])
```

Out[13]:
```
(array([246., 178., 125.,  50.,  83.,  52.,  11.,  19.,   3.,   1.]),
 array([ 0. ,  1.7,  3.4,  5.1,  6.8,  8.5, 10.2, 11.9, 13.6, 15.3, 17. ]),
 <BarContainer object of 10 artists>)
```



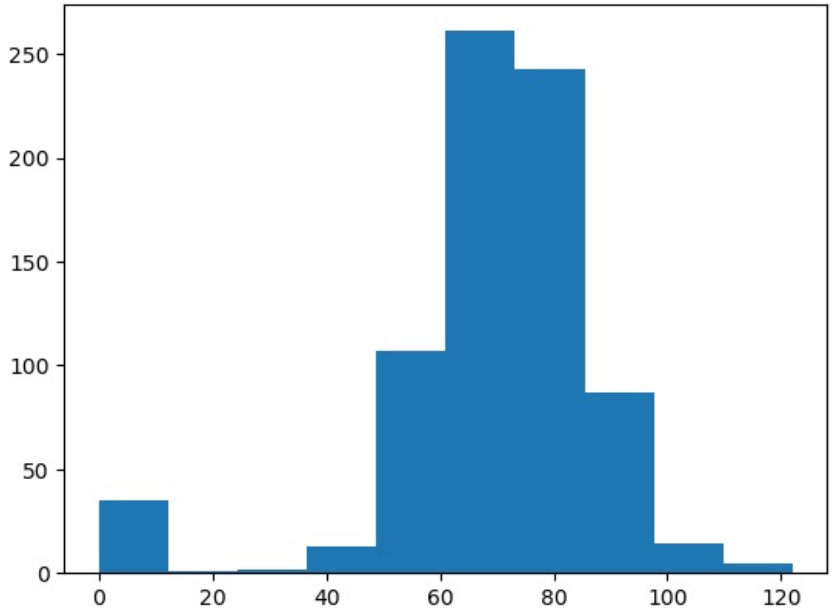
In [14]:
```
plt.hist(data['Glucose'])
```

Out[14]:
```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <BarContainer object of 10 artists>)
```



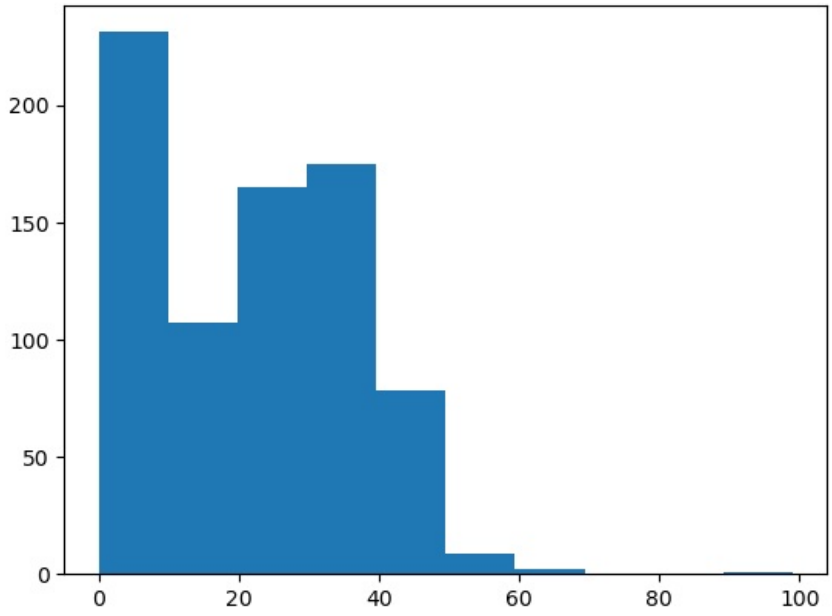
In [15]:
```
plt.hist(data['BloodPressure'])
```

Out[15]:
```
(array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
 array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
        109.8, 122. ]),
 <BarContainer object of 10 artists>)
```



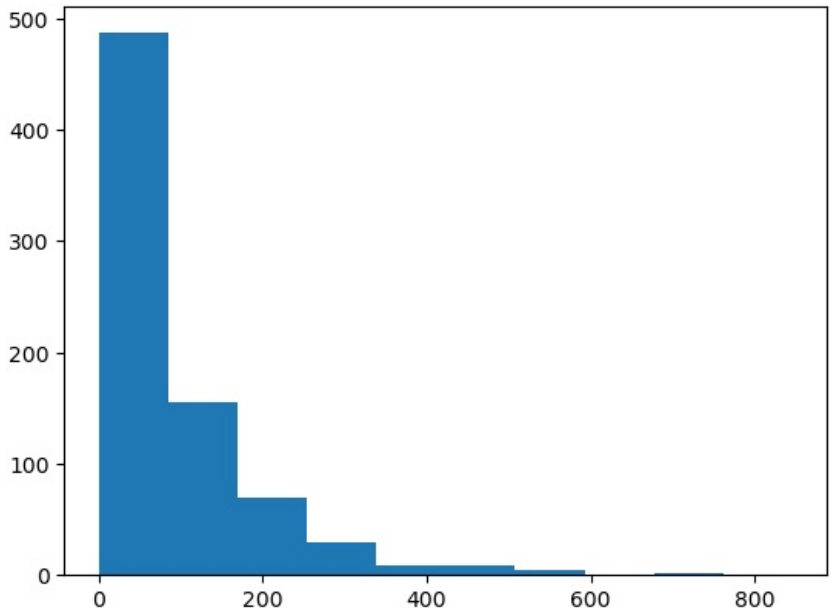
In [16]:
```
plt.hist(data['SkinThickness'])
```

Out[16]:
```
(array([231., 107., 165., 175.,  78.,   9.,   2.,   0.,   0.,   1.]),
 array([ 0. ,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),
 <BarContainer object of 10 artists>)
```



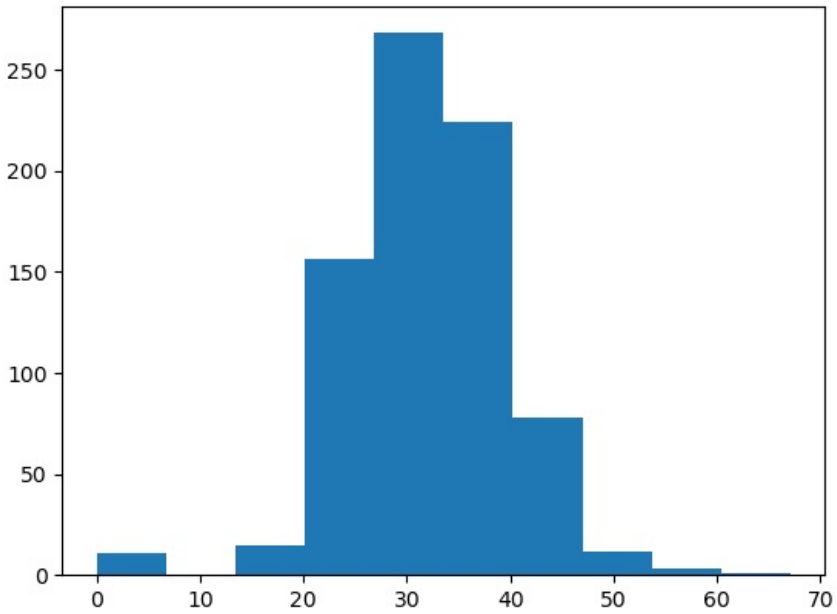
In [17]:
```
plt.hist(data['Insulin'])
```

Out[17]:
```
(array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
 array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
        761.4, 846. ]),
 <BarContainer object of 10 artists>)
```



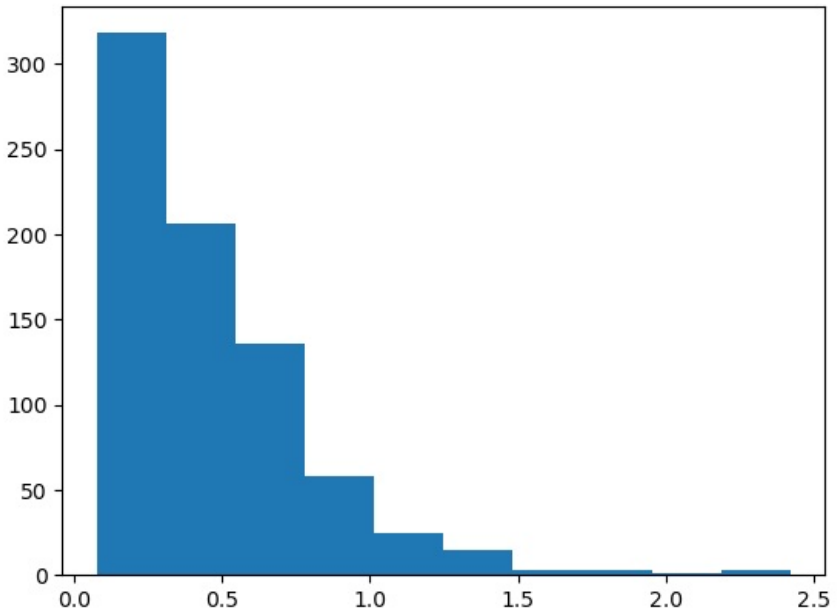
In [18]:
```
plt.hist(data['BMI'])
```

Out[18]:
```
(array([ 11.,   0.,  15., 156., 268., 224.,  78.,  12.,   3.,   1.]),
 array([ 0.  ,  6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,
        60.39, 67.1 ]),
 <BarContainer object of 10 artists>)
```



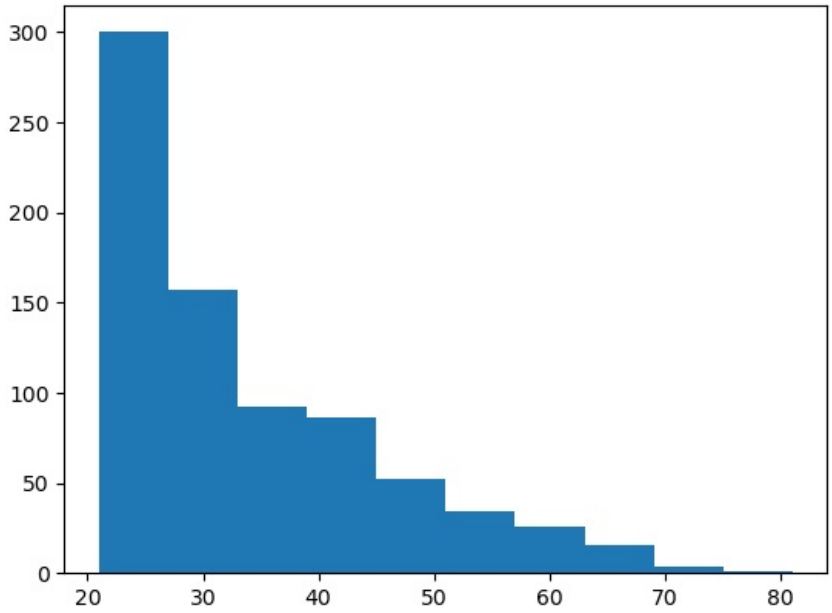
In [19]:
```
plt.hist(data['DiabetesPedigreeFunction'])
```

Out[19]:
```
(array([318., 206., 136.,  58.,  25.,  15.,   3.,   3.,   1.,   3.]),
 array([0.078 , 0.3122, 0.5464, 0.7806, 1.0148, 1.249 , 1.4832, 1.7174,
        1.9516, 2.1858, 2.42  ]),
 <BarContainer object of 10 artists>)
```



In [20]:
```
plt.hist(data['Age'])
```

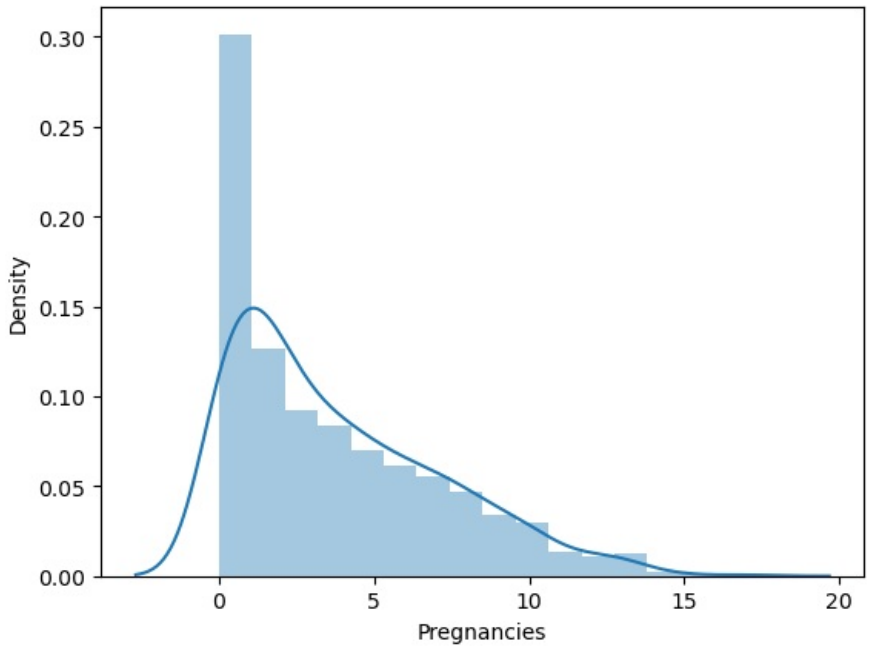
Out[20]:
```
(array([300., 157.,  92.,  86.,  52.,  34.,  26.,  16.,   4.,   1.]),
 array([21., 27., 33., 39., 45., 51., 57., 63., 69., 75., 81.]),
 <BarContainer object of 10 artists>)
```



In [21]:
```
sns.distplot(data['Pregnancies'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\3426279659.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Pregnancies'])
```

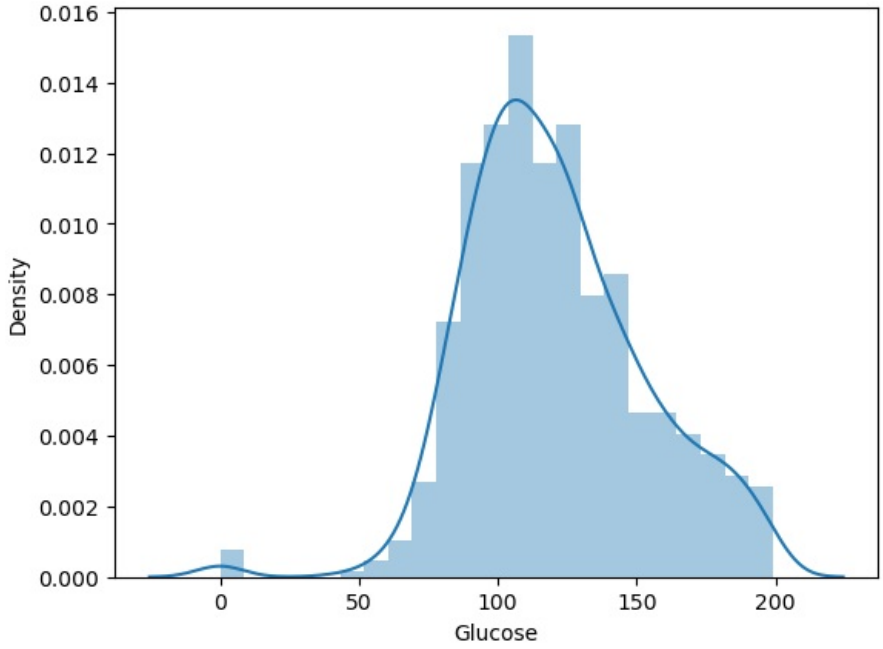
Out[21]:
```
<Axes: xlabel='Pregnancies', ylabel='Density'>
```



In [22]:
```
sns.distplot(data['Glucose'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\1475984872.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Glucose'])
```

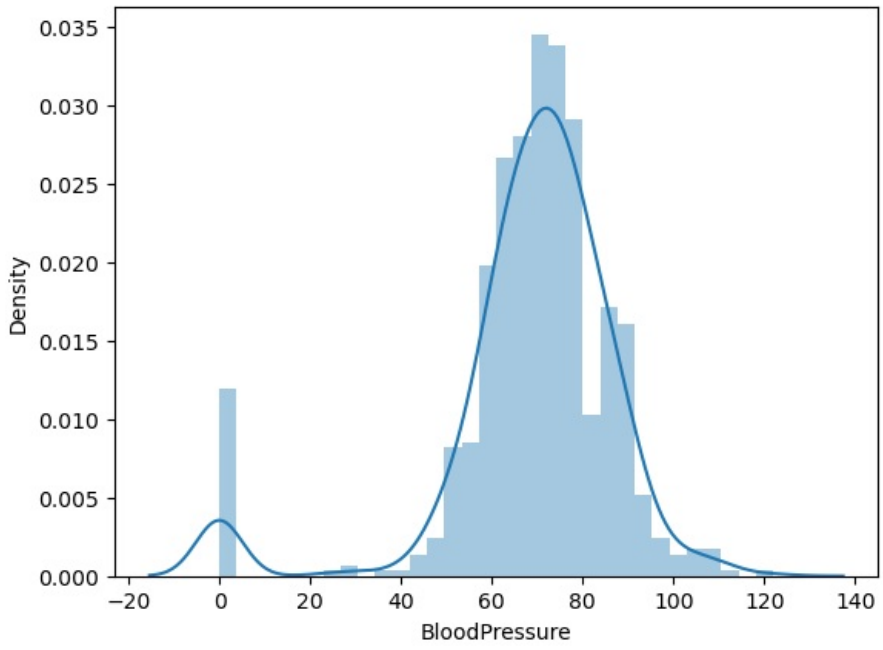
Out[22]:
```
<Axes: xlabel='Glucose', ylabel='Density'>
```



In [23]:
```
sns.distplot(data['BloodPressure'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\3514935741.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['BloodPressure'])
```

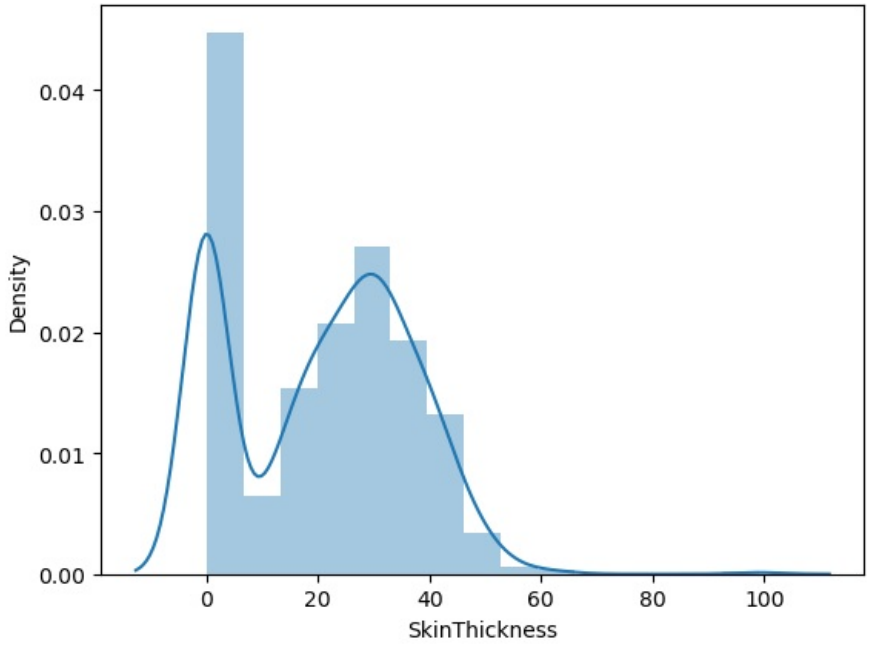
Out[23]:
```
<Axes: xlabel='BloodPressure', ylabel='Density'>
```



In [24]:
```
sns.distplot(data['SkinThickness'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\1087922965.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['SkinThickness'])
```

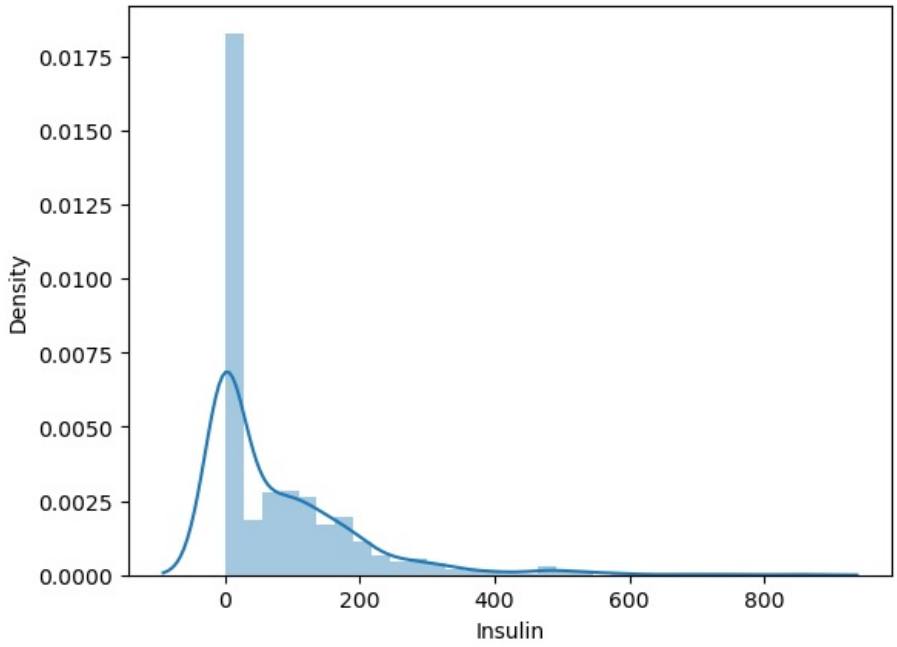
Out[24]:
```
<Axes: xlabel='SkinThickness', ylabel='Density'>
```



In [25]:
```
sns.distplot(data['Insulin'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\4080507389.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Insulin'])
```

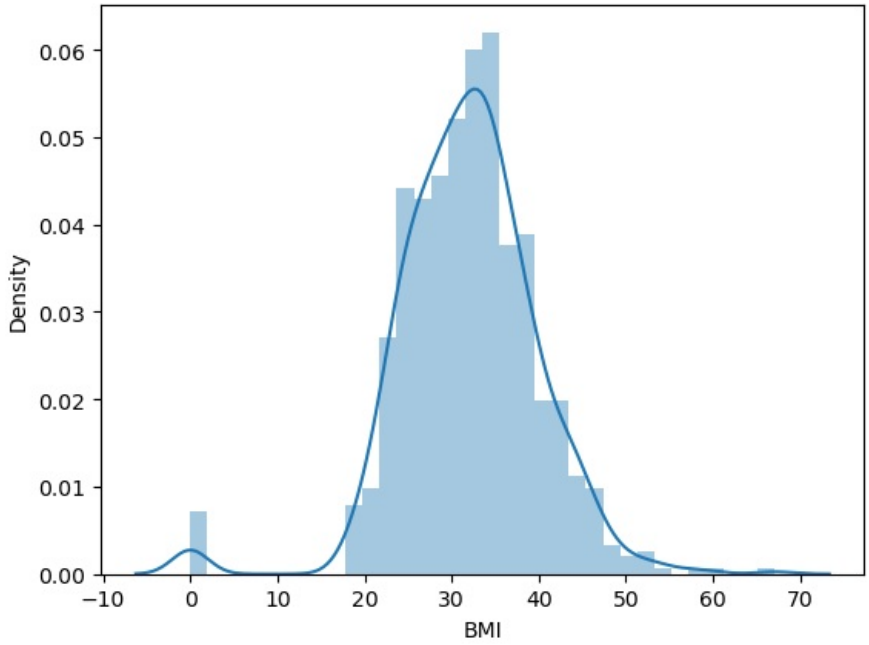
Out[25]:
```
<Axes: xlabel='Insulin', ylabel='Density'>
```



In [26]:
```
sns.distplot(data['BMI'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\613475808.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['BMI'])
```

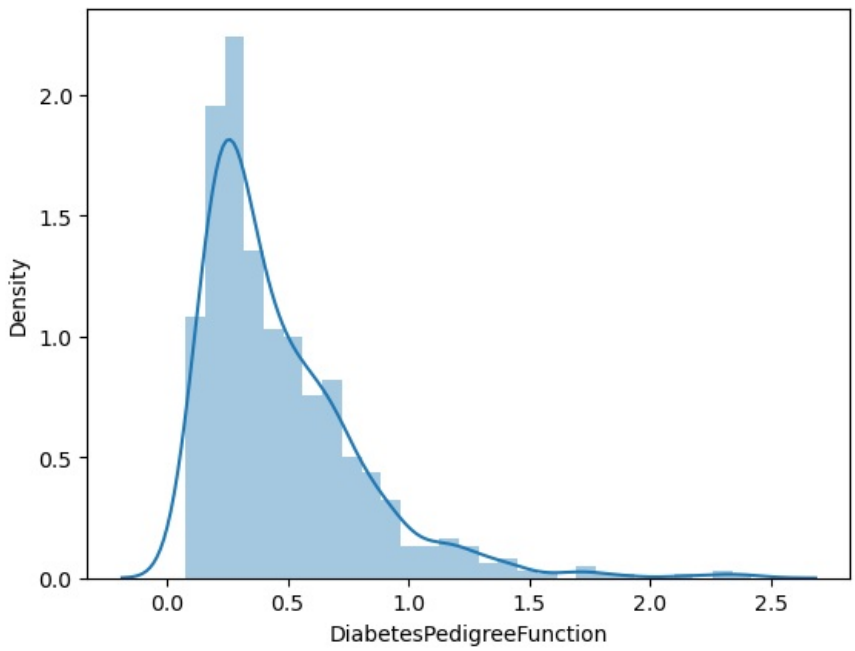
Out[26]:
```
<Axes: xlabel='BMI', ylabel='Density'>
```



In [27]:
```
sns.distplot(data['DiabetesPedigreeFunction'])
```

```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\1100508857.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['DiabetesPedigreeFunction'])
```

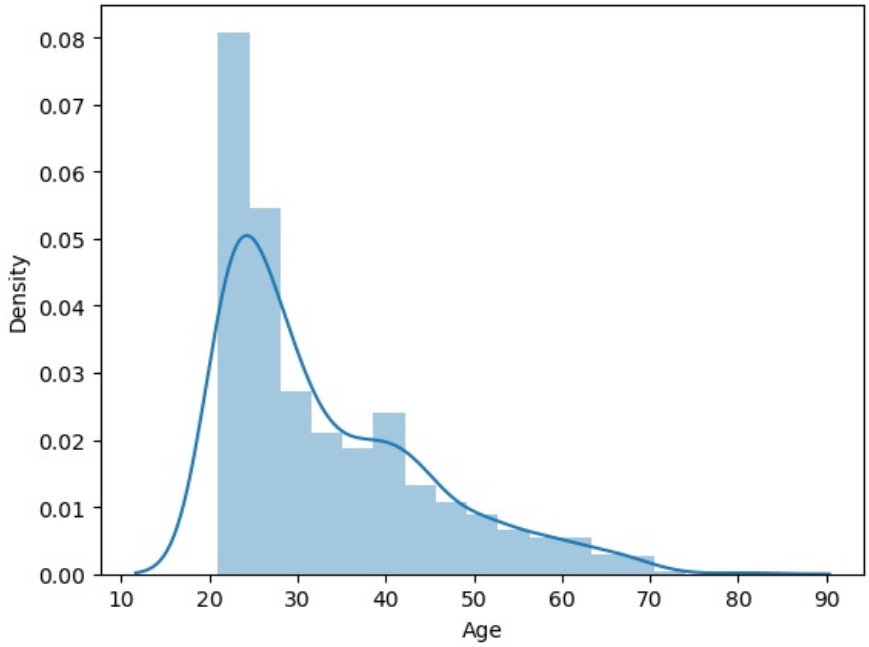
Out[27]:
```
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Density'>
```



In [28]:
```
sns.distplot(data['Age'])
```

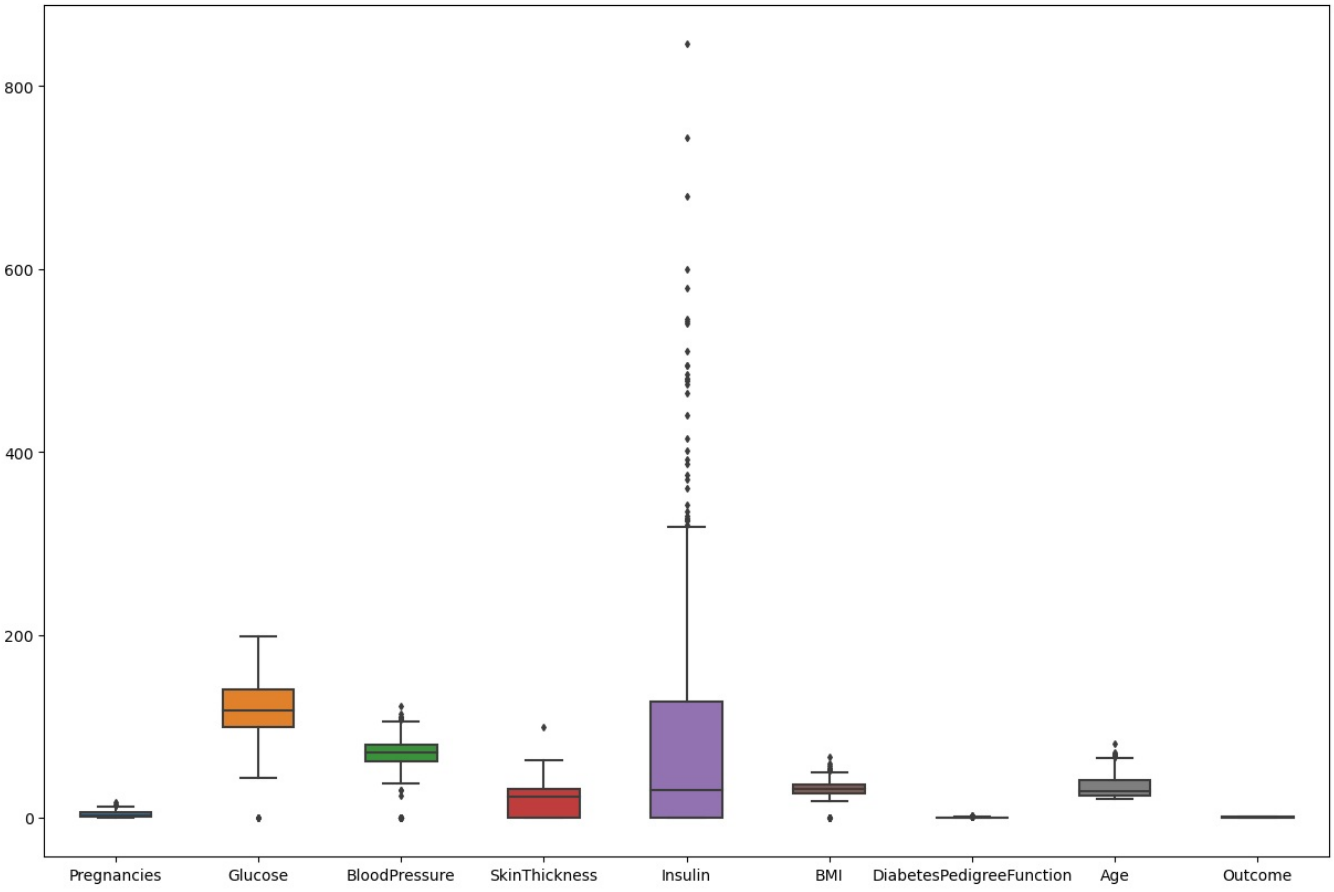
```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\2317092479.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Age'])
```

Out[28]:
```
<Axes: xlabel='Age', ylabel='Density'>
```



In [29]:
```
#now we have dealt with the 0 values and data looks better. But, there still are
#outliers present in some columns.lets visualize it
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

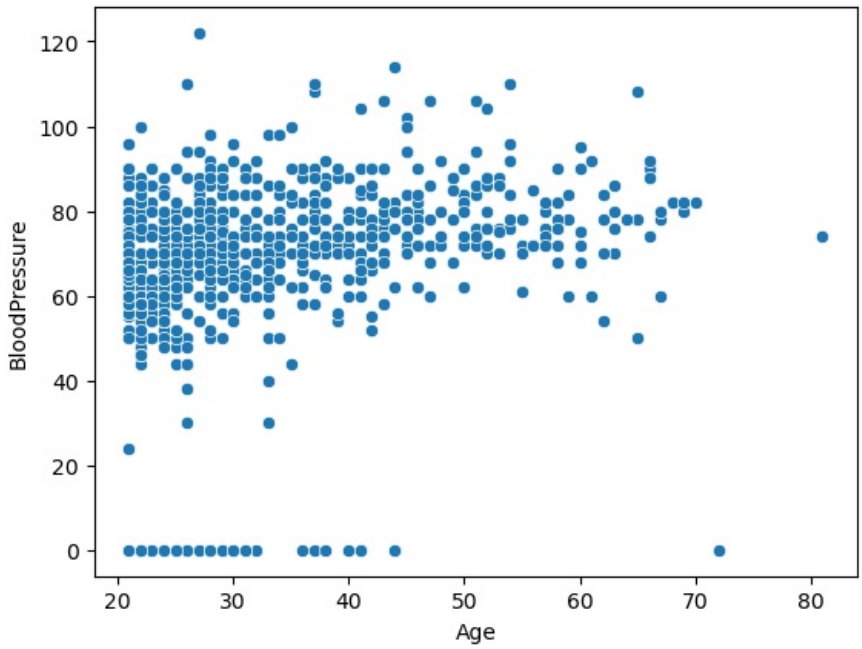
Out[29]:
```
<Axes: >
```



Bivariate Analysis

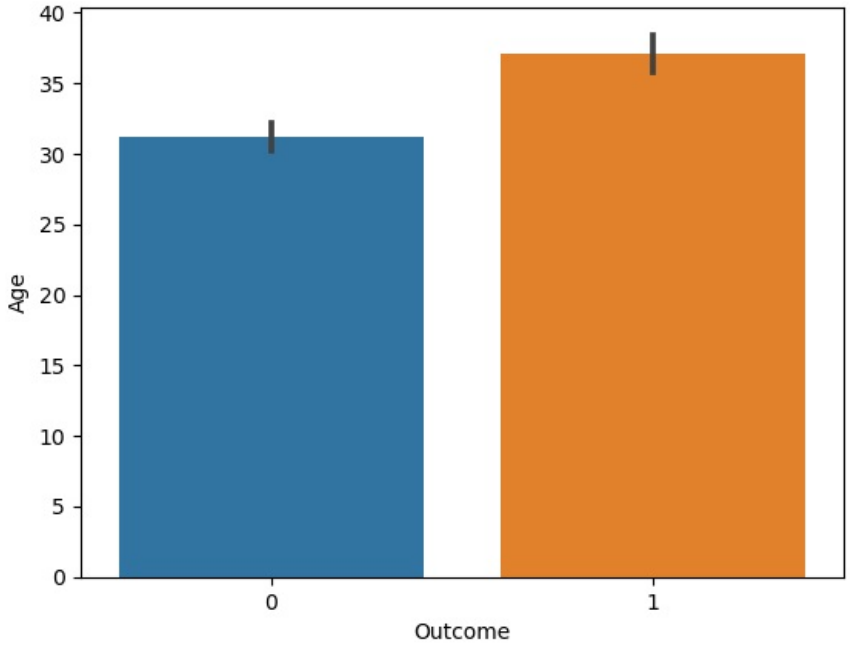
In [30]:
```
sns.scatterplot(x=data['Age'],y=data['BloodPressure'])
```

Out[30]:
```
<Axes: xlabel='Age', ylabel='BloodPressure'>
```



In [31]:
```
sns.barplot(x=data['Outcome'],y=data['Age'])
```

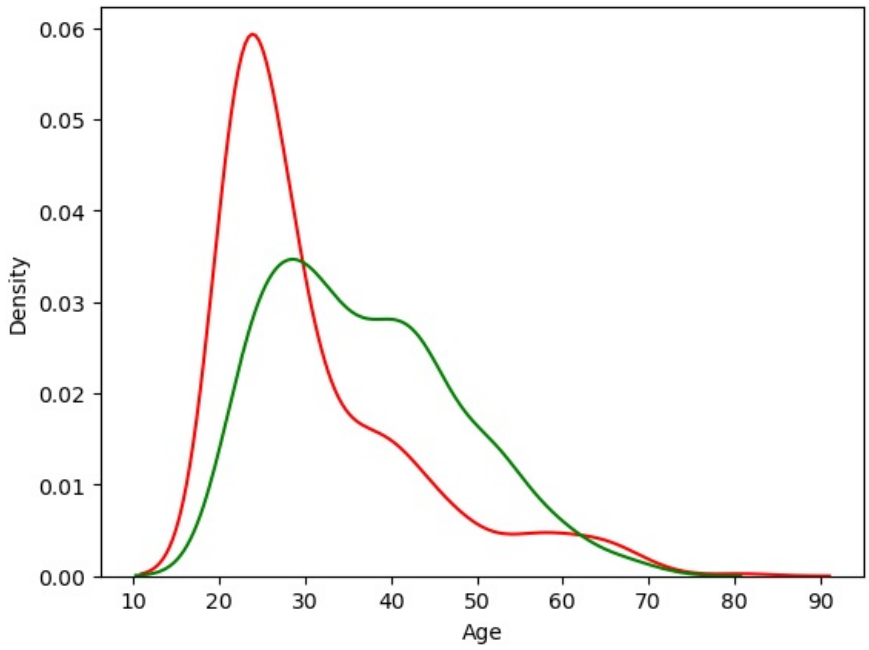
Out[31]:
```
<Axes: xlabel='Outcome', ylabel='Age'>
```



In [32]:
```
sns.distplot(data[data['Outcome']==0]['Age'],color='r',hist=False)
sns.distplot(data[data['Outcome']==1]['Age'],color='g',hist=False)
```

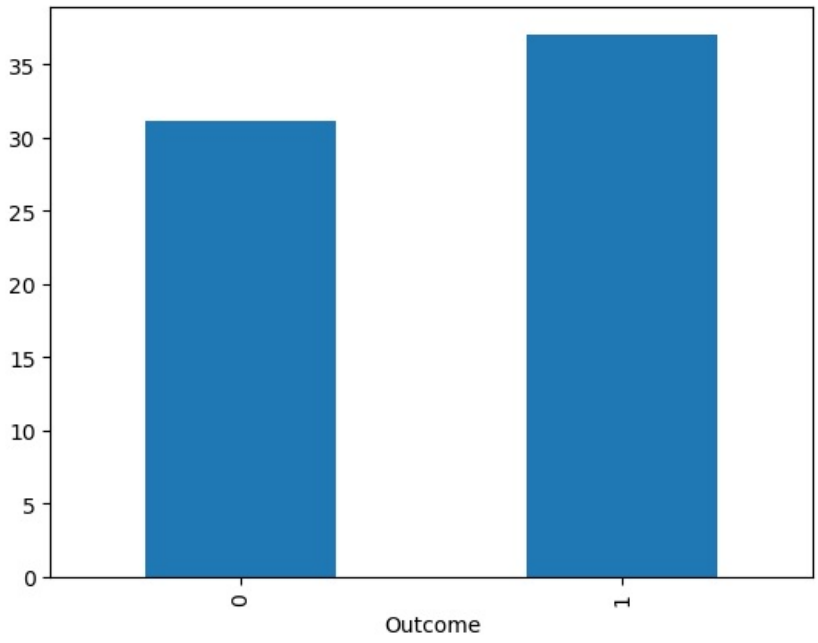
```
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\282527456.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[data['Outcome']==0]['Age'],color='r',hist=False)
C:\Users\yasar beg\AppData\Local\Temp\ipykernel_25128\282527456.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[data['Outcome']==1]['Age'],color='g',hist=False)
```

Out[32]:
```
<Axes: xlabel='Age', ylabel='Density'>
```



In [33]:
```
(data.groupby('Outcome').mean()['Age']).plot(kind='bar')
```

Out[33]:
```
<Axes: xlabel='Outcome'>
```



In [34]:
```
sns.pairplot(data)
```

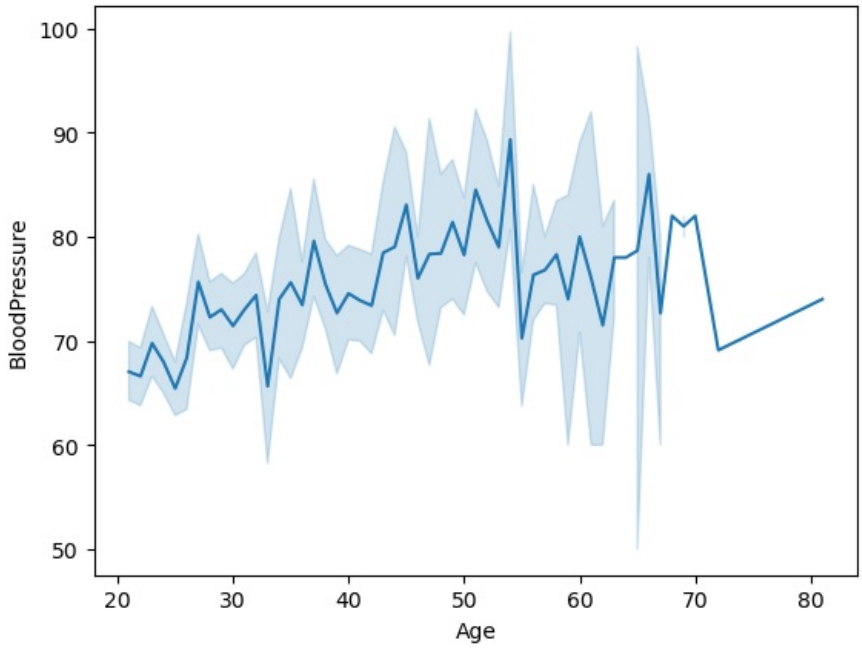
Out[34]:
```
<seaborn.axisgrid.PairGrid at 0x13fb529f3a0>
```

In [35]:
```
sns.pairplot(data,hue='Outcome')
```

Out[35]:
```
<seaborn.axisgrid.PairGrid at 0x13fba8385b0>
```

In [51]:
```
sns.lineplot(x=data['Age'],y=data['BloodPressure'])
```

Out[51]:
```
<Axes: xlabel='Age', ylabel='BloodPressure'>
```



In [52]:
```
data.describe()
```

Out[52]:

	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	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	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

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [53]:
```
#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose, insuline can't be zero so
# now replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].mean())
```

In [54]:
```
data.shape
```

Out[54]:
```
(768, 9)
```

In [55]:
```
data.head()
```

Out[55]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	1

In [56]:
```
#segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

In [57]:
```
# separate dataset into train and test
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
X_train.shape, X_test.shape
```

Out[57]:
```
((576, 8), (192, 8))
```

In [58]:
```
import pickle
##standard Scaling- Standardization
def scaler_standard(X_train, X_test):
    #scaling the data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    #saving the model
    file = open('diabetes.csv','wb')
    pickle.dump(scaler,file)
    file.close()

    return X_train_scaled, X_test_scaled
```

In [59]:
```
X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

In [60]:
```
X_train_scaled
```

Out[60]:
```
array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
         0.51627505,  2.953134  ]])
```

In [61]:
```
log_reg = LogisticRegression()

log_reg.fit(X_train_scaled,y_train)
```

Out[61]:
```
LogisticRegression()
```

In [65]:
```
## Hyperparameter Tuning
## GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
```

In [66]:
```
logreg = LogisticRegression()
clf = GridSearchCV(logreg,                    # model
                   param_grid = parameters,   # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=10)                     # number of folds

clf.fit(X_train_scaled,y_train)
```

Out[66]:
```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [67]:
```
clf.best_params_
```

Out[67]:
```
{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

In [68]:
```
clf.best_score_
```

Out[68]:
```
0.763793103448276
```

let's see how well our model performs on the test data set.

In [69]:
```
y_pred = clf.predict(X_test_scaled)
```

In [69]:
```
y_pred = clf.predict(X_test_scaled)
```

accuracy = accuracy_score(y_test,y_pred) accuracy

In [70]:
```
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

Out[70]:
```
array([[117,  13],
       [ 26,  36]], dtype=int64)
```

In [71]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

In [72]:
```
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_negative)
Accuracy
```

Out[72]:
```
0.796875
```

In [73]:
```
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[73]:
```
0.9
```

In [74]:
```
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[74]:
```
0.8181818181818182
```

In [75]:
```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[75]:
```
0.8571428571428572
```

Decision Tree Model

In [76]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [77]:
```
X_train_scaled
```

Out[77]:
```
array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
         0.51627505,  2.953134  ]])
```

In [78]:
```
dt=DecisionTreeClassifier()
```

In [79]:
```
dt.fit(X_train_scaled,y_train)
```

Out[79]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [80]:
```
y1_pred=dt.predict(X_test_scaled)
```

In [81]:
```
y1_pred
```

Out[81]:
```
array([1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0], dtype=int64)
```

In [82]:
```
conf_mat = confusion_matrix(y_test,y1_pred)
conf_mat
```

Out[82]:
```
array([[104,  26],
       [ 21,  41]], dtype=int64)
```

In [83]:
```
true_positive = conf_mat[0][0]
```

```
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

In [84]:
```
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_negative)
Accuracy
```

Out[84]:
```
0.7552083333333334
```

In [85]:
```
## Hyperparameter Tuning
## GridSearch CV
from sklearn.model_selection import GridSearchCV
import numpy as np
import warnings
warnings.filterwarnings('ignore')
# parameter grid
parameters = {
    'criterion': ["gini", "entropy", "log_loss"],
    'max_depth':[2,3,5],
    'max_features':["auto", "sqrt", "log2"],
}
```

dth = DecisionTreeClassifier() clf = GridSearchCV(dth, # model param_grid = parameters, # hyperparameters scoring='accuracy', # metric for scoring cv=10) # number of folds clf.fit(X_train_scaled,y_train)

In [86]:
```
clf.best_params_
```

Out[86]:
```
{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

In [87]:
```
y_pred2=clf.predict(X_test_scaled)
```

In [88]:
```
conf_mat = confusion_matrix(y_test,y_pred2)
conf_mat
```

Out[88]:
```
array([[117,  13],
       [ 26,  36]], dtype=int64)
```

In [89]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

In [90]:
```
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_negative)
Accuracy
```

Out[90]:
```
0.796875
```

Random Forest classifier

In [91]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [92]:
```
rf=RandomForestClassifier()
```

In [93]:
```
rf.fit(X_train,y_train)
```

Out[93]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [94]:
```
y_pred3=rf.predict(X_test)
```

In [95]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [96]:
```
accuracy_score(y_test,y_pred3)
```

Out[96]:
```
0.7760416666666666
```

In [97]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import ReLU,LeakyReLU
from keras.layers import Dropout
```

```
WARNING:tensorflow:From C:\Users\yasar beg\AppData\Roaming\Python\Python310\site-packages\keras\src\losses.py:2
976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softm
ax_cross_entropy instead.
```

In [98]:
```
classifier=Sequential()
```

```
WARNING:tensorflow:From C:\Users\yasar beg\AppData\Roaming\Python\Python310\site-packages\keras\src\backend.py:
873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

In [99]:
```
classifier.add(Dense(units=256,kernel_initializer='he_uniform',activation='relu',input_dim=8))
```

```
classifier.add(Dense(units=128,kernel_initializer='he_uniform',activation='relu'))
classifier.add(Dense(units=1,kernel_initializer='glorot_uniform',activation='sigmoid'))
classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\yasar beg\AppData\Roaming\Python\Python310\site-packages\keras\src\optimizers\
__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [100…

```
classifier.fit(X_train,y_train,validation_split=.30,batch_size=10,epochs=1000)
```

```
Epoch 1/1000
WARNING:tensorflow:From C:\Users\yasar beg\AppData\Roaming\Python\Python310\site-packages\keras\src\utils\tf_ut
ils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValu
e instead.

WARNING:tensorflow:From C:\Users\yasar beg\AppData\Roaming\Python\Python310\site-packages\keras\src\engine\base
_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.exe
cuting_eagerly_outside_functions instead.

41/41 [==============================] - 1s 7ms/step - loss: 6.4166 - accuracy: 0.5782 - val_loss: 7.1092 - val
_accuracy: 0.6763
Epoch 2/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.9842 - accuracy: 0.6278 - val_loss: 2.9164 - val
_accuracy: 0.5723
Epoch 3/1000
41/41 [==============================] - 0s 3ms/step - loss: 4.4580 - accuracy: 0.5980 - val_loss: 6.5423 - val
_accuracy: 0.6069
Epoch 4/1000
41/41 [==============================] - 0s 3ms/step - loss: 4.3506 - accuracy: 0.6452 - val_loss: 14.1354 - va
l_accuracy: 0.4451
Epoch 5/1000
41/41 [==============================] - 0s 3ms/step - loss: 4.5971 - accuracy: 0.6278 - val_loss: 3.2493 - val
_accuracy: 0.6185
Epoch 6/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.1192 - accuracy: 0.6427 - val_loss: 4.8884 - val
_accuracy: 0.6532
Epoch 7/1000
41/41 [==============================] - 0s 3ms/step - loss: 2.8002 - accuracy: 0.6303 - val_loss: 2.5701 - val
_accuracy: 0.6763
Epoch 8/1000
41/41 [==============================] - 0s 3ms/step - loss: 4.3877 - accuracy: 0.6154 - val_loss: 6.9639 - val
_accuracy: 0.7168
Epoch 9/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.7558 - accuracy: 0.6551 - val_loss: 2.3625 - val
_accuracy: 0.5954
Epoch 10/1000
41/41 [==============================] - 0s 3ms/step - loss: 1.8798 - accuracy: 0.6501 - val_loss: 6.7342 - val
_accuracy: 0.3468
Epoch 11/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.3200 - accuracy: 0.6303 - val_loss: 7.4367 - val
_accuracy: 0.4104
Epoch 12/1000
41/41 [==============================] - 0s 4ms/step - loss: 7.7595 - accuracy: 0.5906 - val_loss: 10.8674 - va
l_accuracy: 0.6994
Epoch 13/1000
41/41 [==============================] - 0s 3ms/step - loss: 4.6997 - accuracy: 0.6377 - val_loss: 3.6709 - val
_accuracy: 0.6416
Epoch 14/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.5256 - accuracy: 0.6625 - val_loss: 4.0147 - val
_accuracy: 0.6936
Epoch 15/1000
41/41 [==============================] - 0s 3ms/step - loss: 2.3945 - accuracy: 0.6576 - val_loss: 4.1559 - val
_accuracy: 0.6069
Epoch 16/1000
41/41 [==============================] - 0s 3ms/step - loss: 2.3974 - accuracy: 0.6898 - val_loss: 4.5048 - val
_accuracy: 0.5376
Epoch 17/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.7479 - accuracy: 0.6203 - val_loss: 4.1075 - val
_accuracy: 0.5665
Epoch 18/1000
41/41 [==============================] - 0s 3ms/step - loss: 2.3449 - accuracy: 0.6799 - val_loss: 8.0383 - val
_accuracy: 0.6647
Epoch 19/1000
41/41 [==============================] - 0s 3ms/step - loss: 3.4332 - accuracy: 0.6328 - val_loss: 5.5141 - val
_accuracy: 0.6590
Epoch 20/1000
41/41 [==============================] - 0s 3ms/step - loss: 2.3402 - accuracy: 0.6303 - val_loss: 3.1011 - val
_accuracy: 0.6647
Epoch 21/1000
41/41 [==============================] - 0s 3ms/step - loss: 1.6816 - accuracy: 0.6774 - val_loss: 2.8666 - val
_accuracy: 0.5202
Epoch 22/1000
41/41 [==============================] - 0s 3ms/step - loss: 1.7759 - accuracy: 0.6700 - val_loss: 3.3286 - val
_accuracy: 0.6590
Epoch 23/1000
41/41 [==============================] - 0s 3ms/step - loss: 2.1634 - accuracy: 0.6799 - val_loss: 2.7228 - val
_accuracy: 0.6532
Epoch 24/1000
41/41 [==============================] - 0s 3ms/step - loss: 1.3649 - accuracy: 0.7171 - val_loss: 2.8004 - val
```

```
Epoch 974/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0178 - accuracy: 0.9975 - val_loss: 5.8535 - val
_accuracy: 0.6301
Epoch 975/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0213 - accuracy: 0.9950 - val_loss: 5.9004 - val
_accuracy: 0.6474
Epoch 976/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0178 - accuracy: 0.9950 - val_loss: 5.9228 - val
_accuracy: 0.6416
Epoch 977/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0164 - accuracy: 0.9975 - val_loss: 5.9760 - val
_accuracy: 0.6474
Epoch 978/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0166 - accuracy: 0.9975 - val_loss: 5.9857 - val
_accuracy: 0.6358
Epoch 979/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0158 - accuracy: 0.9975 - val_loss: 5.9742 - val
_accuracy: 0.6416
Epoch 980/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0158 - accuracy: 0.9975 - val_loss: 6.0998 - val
_accuracy: 0.6301
Epoch 981/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0151 - accuracy: 0.9975 - val_loss: 6.0688 - val
_accuracy: 0.6301
Epoch 982/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0154 - accuracy: 1.0000 - val_loss: 6.0862 - val
_accuracy: 0.6127
Epoch 983/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0159 - accuracy: 0.9975 - val_loss: 6.1476 - val
_accuracy: 0.6416
Epoch 984/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0130 - accuracy: 0.9975 - val_loss: 6.1720 - val
_accuracy: 0.6301
Epoch 985/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0131 - accuracy: 0.9975 - val_loss: 6.2474 - val
_accuracy: 0.6358
Epoch 986/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0125 - accuracy: 1.0000 - val_loss: 6.2298 - val
_accuracy: 0.6243
Epoch 987/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0162 - accuracy: 0.9975 - val_loss: 6.2501 - val
_accuracy: 0.6185
Epoch 988/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0154 - accuracy: 0.9975 - val_loss: 6.2654 - val
_accuracy: 0.6416
Epoch 989/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0131 - accuracy: 1.0000 - val_loss: 6.3384 - val
_accuracy: 0.6127
Epoch 990/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0164 - accuracy: 0.9950 - val_loss: 6.4028 - val
_accuracy: 0.6185
Epoch 991/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0203 - accuracy: 0.9950 - val_loss: 6.3391 - val
_accuracy: 0.6358
Epoch 992/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0179 - accuracy: 0.9950 - val_loss: 6.3458 - val
_accuracy: 0.6069
Epoch 993/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0178 - accuracy: 0.9975 - val_loss: 6.3165 - val
_accuracy: 0.6185
Epoch 994/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0137 - accuracy: 0.9950 - val_loss: 6.4066 - val
_accuracy: 0.6532
Epoch 995/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0125 - accuracy: 1.0000 - val_loss: 6.4155 - val
_accuracy: 0.6358
Epoch 996/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0143 - accuracy: 0.9975 - val_loss: 6.4738 - val
_accuracy: 0.6301
Epoch 997/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0142 - accuracy: 1.0000 - val_loss: 6.5320 - val
_accuracy: 0.6416
Epoch 998/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0198 - accuracy: 0.9975 - val_loss: 6.5121 - val
_accuracy: 0.6012
Epoch 999/1000
41/41 [==============================] - 0s 2ms/step - loss: 0.0122 - accuracy: 1.0000 - val_loss: 6.4995 - val
_accuracy: 0.6069
Epoch 1000/1000
41/41 [==============================] - 0s 3ms/step - loss: 0.0149 - accuracy: 0.9975 - val_loss: 6.6109 - val
_accuracy: 0.6127
```

Out[100]: <keras.src.callbacks.History at 0x13fcb52f280>

In [43]:
```
#import pickle
#pickle.dump(log_reg,file)
#file.close()
```

In [ ]: