Multiple Disease Prediction Model

Tamanna Sharma

Lovely Professional University

12110221

CONTENTS

Sl. No.	Topic	Page No.
1	Acknowledgement	03
2	Project Objective	**04-05**
3	Project Scope	**06**
4	Data Description	**07-08**
5	Data Pre-Processing	**09-17**
6	Model Building	**18-25**
7	Test Dataset	**26-29**
8	Code	**30-79**
9	Future Scope of Improvements	**80**
10	Certificates	**81**

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my faculty, **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I extend my gratitude to the authors and researchers whose work I have referenced throughout this project. Their groundbreaking research, published papers, and open-source contributions have laid the foundation upon which my project is built. I am indebted to their dedication and commitment to advancing the field of machine learning.

Project Objective

In this project we have made two models, ‘Diabetes Prediction Model’ and ‘Heart Disease Prediction Model’ and we have made a web app with a menu bar having options to predict any of them.

Let us first discuss about the Diabetes Dataset which we have taken from Kaggle. In this dataset, the target attribute is outcome which is binary. So, in this project we need to do binary classification based on the attributes present in our dataset and predict whether a person is suffering from diabetes or not.

Similarly, Heart Disease Dataset is also taken from Kaggle. In this dataset, the target attribute is target which is also binary. Therefore, we need to do the binary classification for heart disease prediction model as well.

Our objective in this project is to make a web app using stream-lit library that will be able to predict the diabetes and heart disease in an individual. For that we first need to create ML models for each one of them. We need to study each dataset and we might need to pre-process the given dataset if we need to. Then, we would train 4 models viz. ‘KNN classifier model’, ‘Naive Bayes classifier model’, ‘Support Vector Machine Model’ and ‘Logistic Regression model’. After training the aforementioned models, we will need to find out the score and classification report. Our next step would be to use the trained models to predict the outcomes using the given test dataset and compare the outcome of each model. We would then choose the best model based on the accuracy score and classification report. After this using pickle5 library we will be dumping the model and when required we will load the model and using stream-lit we will be creating option menu bar and a proper and complete web app to take inputs from users and provide the predicted output in a better way.

Our methodology for solving the problems in the given project is described below:

- Load the required dataset.

- Study the dataset.

- Describe the dataset.

- Visualise the dataset.

- Find out if the dataset needs to be pre-processed. It will be determined on the basis of whether the dataset has null values or outliers or any such discrepancy that might affect the output of the models to be trained.

- If the dataset is required to be pre-processed, take the necessary steps to pre-process the data.

- Find out the principal attributes for training.
- Split the given dataset for training and testing purpose.
- Fit the previously split train data in the aforementioned 4 models.
- Calculate the accuracy of the 4 models and find out the classification reports.
- Plot the necessary graphs.
- Use each trained model to predict the outcomes of the given test dataset.
- Choose the best model among the 4 trained models bases on the accuracy and classification reports and build the web app using stream-lit.

PROJECT SCOPE

The broad scope of 'Multiple Disease Prediction Model' project is given below:

- The given datasets have the attributes based on which it can be predicted that whether a person is suffering from a particular disease or not.
- It is a useful project as the Classifier models can be used to quickly determine whether an individual is healthy or not.
- These models can be used in hospitals by the doctors and they can modify them according to their needs. This will reduce the manual labour and time spent on determining whether a person has a particular disease or not.
- It can also provide personalized predictions to patients.
- The dataset given to us is a shortened form of the original dataset from Kaggle. So, the results might have some mismatch with the real-world applications. But that can be
- avoided if the models are trained accordingly.

DATA DESCRIPTION

Source of the data: Kaggle. The given dataset is a shortened version of the original dataset in Kaggle.

Diabetes Dataset: This dataset has 768 rows and 8 columns.

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

Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Heart Disease Dataset: This dataset has 303 rows and 13 columns.

1. age
2. sex
3. cp: chest pain type (4 values)
4. trestbps: resting blood pressure
5. chol: serum cholestrol in mg/dl
6. fbs: fasting blood sugar > 120 mg/dl
7. restecg: resting electrocardiographic results (values 0,1,2)
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina
10. oldpeak: ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
12. ca: number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect
14. target: Class variable (0 or 1). It is integer valued 0 = no disease and 1 = disease.

The following table shows 5 number statistics of the diabetes dataset:

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
mean	3.845052	120.894531	69.105469	20.666667	79.799479	31.992578	0.471876	32.890625	0.348958
std	3.369578	31.972618	19.355807	15.561237	115.24400	7.884160	0.331329	11.500287	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	127.25000	36.600000	0.626250	40.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Similarly, we can also know about the statistics of heart disease dataset which would be given in code part.

Now we will pre-process the data. The methodology followed is given below:

• Checking for null values.

o If null values are present, we will fill them or drop the row containing the null

value based on the dataset.

• Checking for outliers.

o If outliers are present, they will either be removed or replaced by following a

suitable method depending on the dataset.

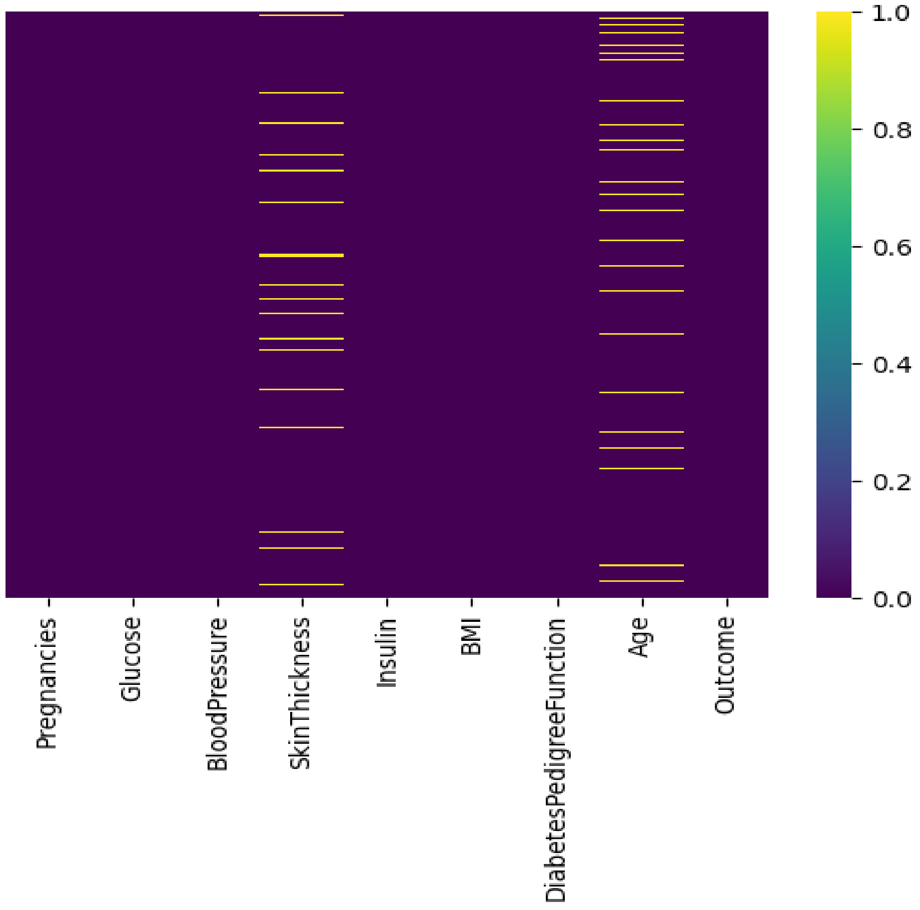
DATA PRE-PROCESSING

We searched for null values in our dataset and formed the following table:

Diabetes:

Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	42
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	45
Outcome	0

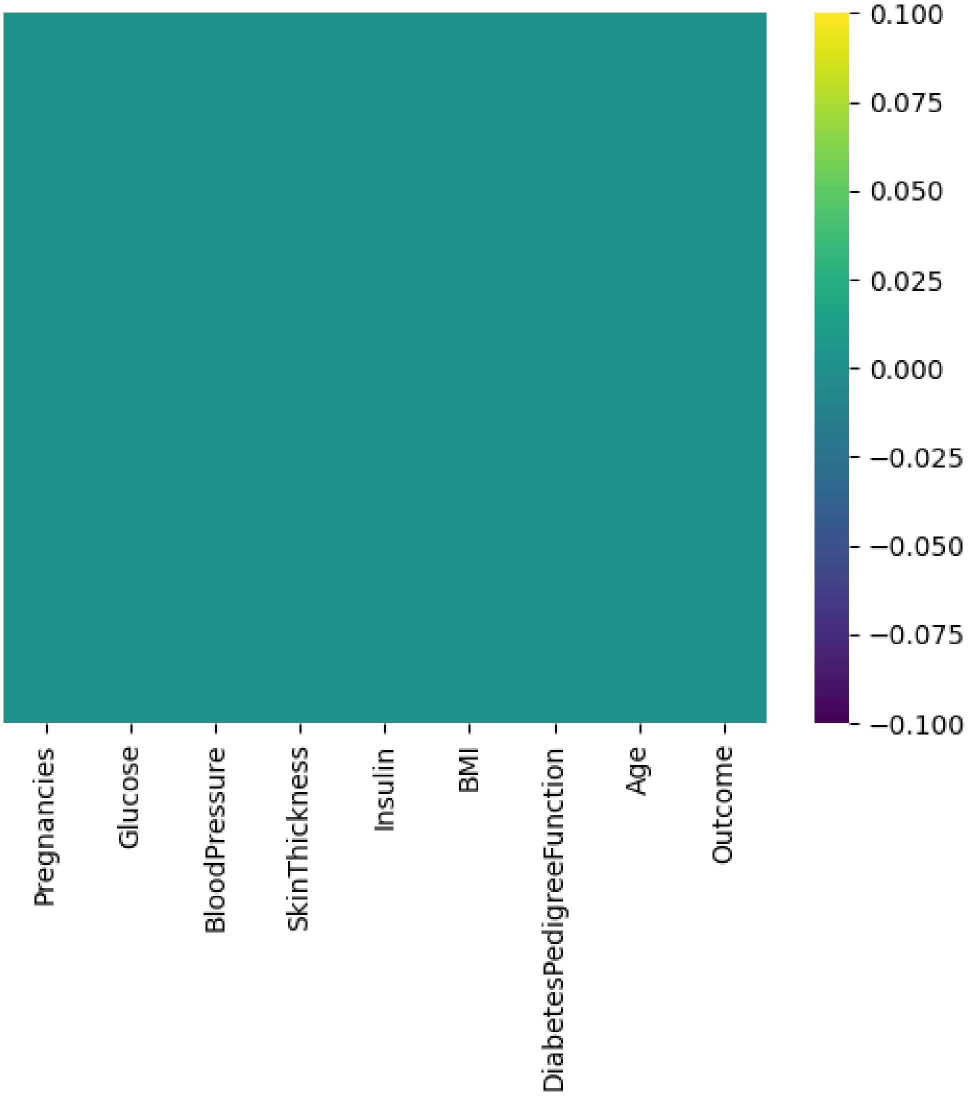
To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:



The heatmap shows that the dataset has null values.

To remove the null values we have filled the missing values with the median in age attribute and SkinThickness attribute.

After removing the null values, the following heatmap was obtained:

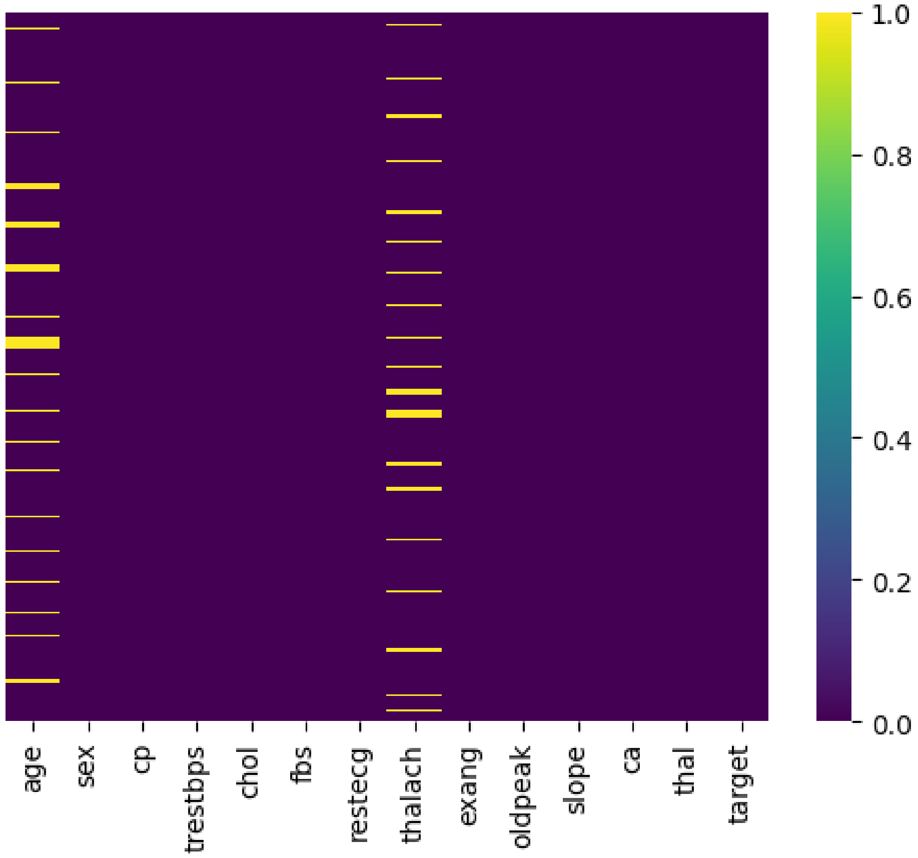


Similarly, the null values are removed from heart disease dataset:

Count of null values:

Age	27
Sex	0
Cp	0
Trestbps	0
Chol	0
Fbs	0
Restecg	0
Thalach	23
Exang	0
Oldpeak	0
Slope	0
Ca	0
Thal	0
Target	0

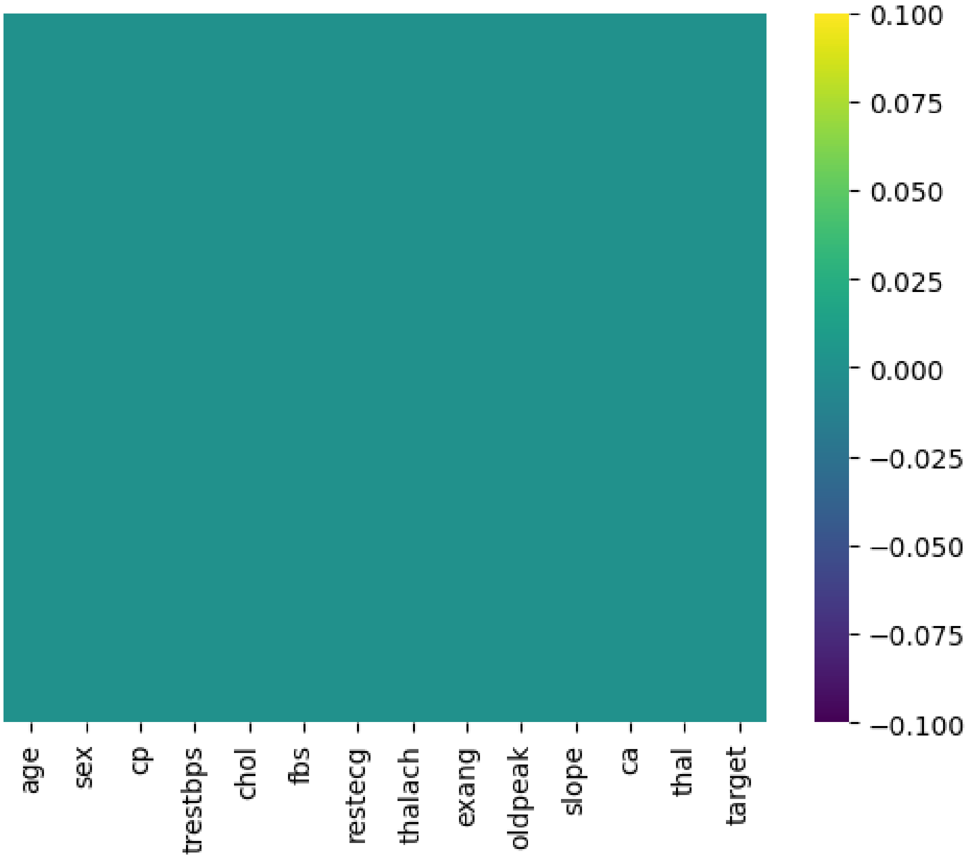
To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:



The heatmap shows that the dataset has null values.

To remove the null values we have filled the missing values with the median in age attribute and thalach attribute.

After removing the null values, the following heatmap was obtained:



Now we have successfully handled Null values and we didn't drop the rows with null values as we have a small dataset.

So, we are moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. ‘outcome’ in diabetes and ‘target’ in heart disease dataset.

Pregnancies	0.221898
Glucose	0.466581
BloodPressure	0.065068
SkinThickness	0.074752
Insulin	0.130548
BMI	0.292695
DiabetesPedigreeFunction	0.173844
Age	0.238356
Outcome	1.000000

Diabetes dataset correlation with Outcome

Age	-0.217072
Sex	-0.280937
Cp	0.433798
Trestbps	-0.144931
Chol	-0.085239
Fbs	-0.028046
Restecg	0.137230
Thalach	0.403064
Exang	-0.436757
Oldpeak	-0.430696
Slope	0.345877
Ca	-0.391724
Thal	-0.344029
Target	1.000000

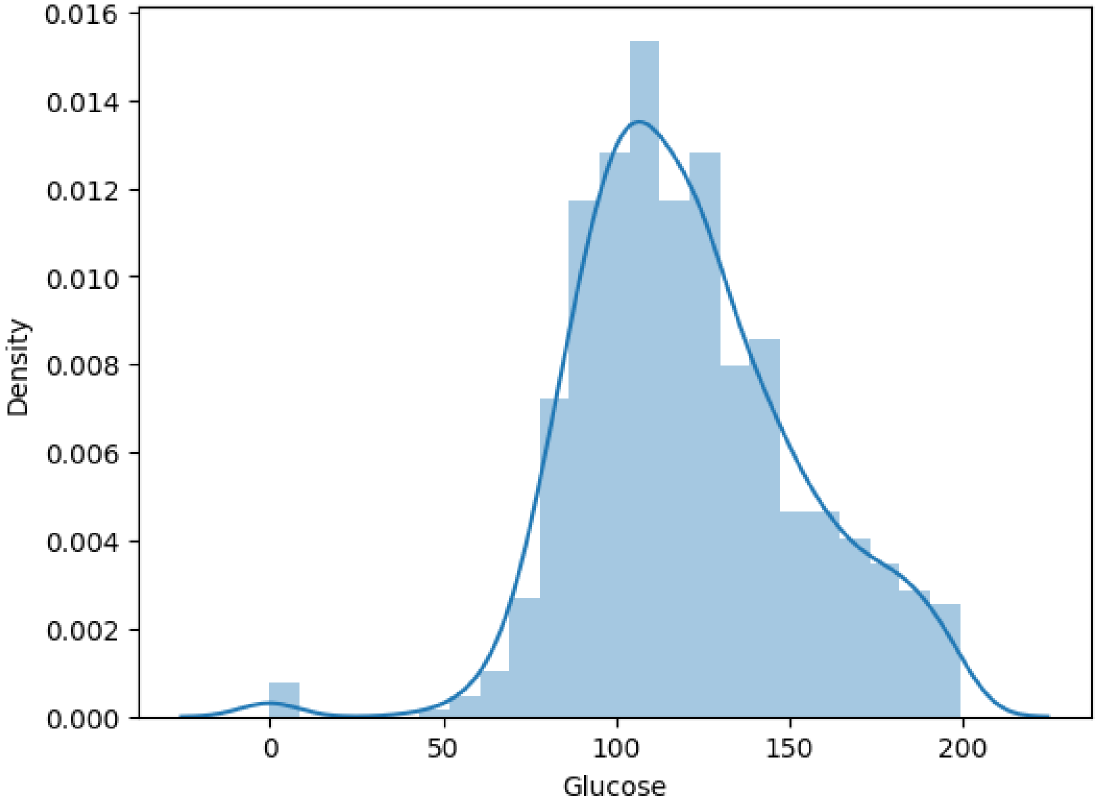
Heart Disease dataset correlation with target

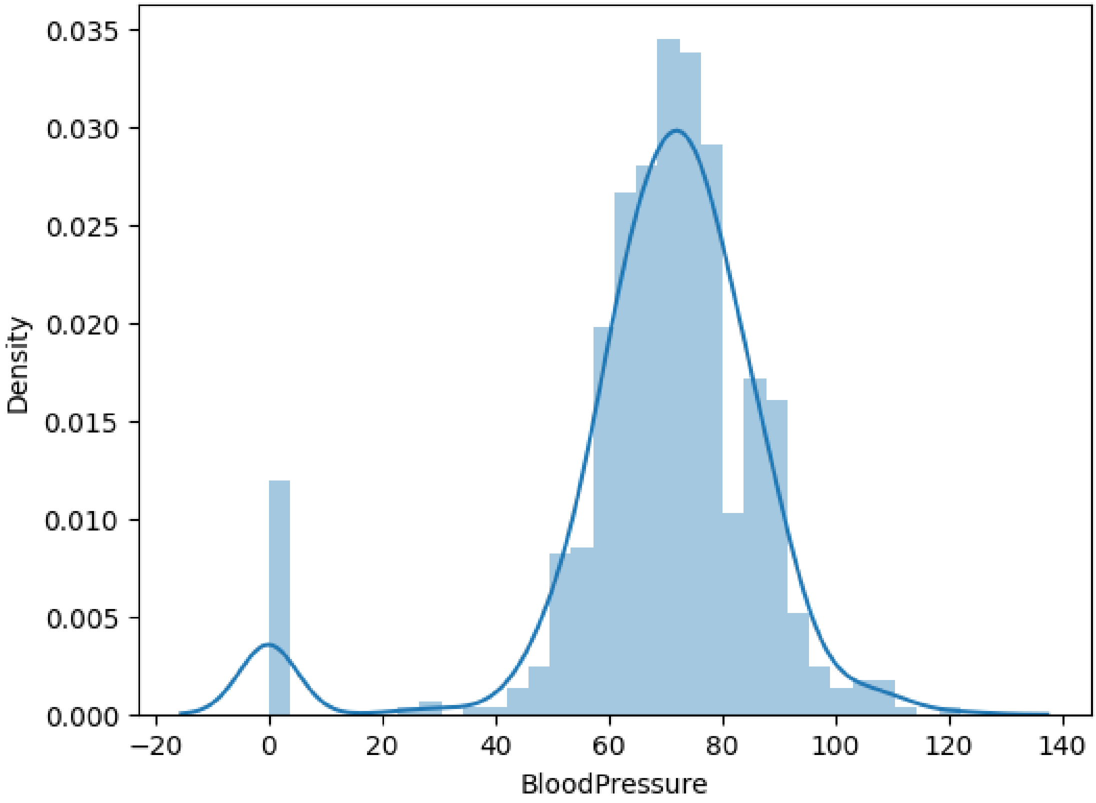
Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

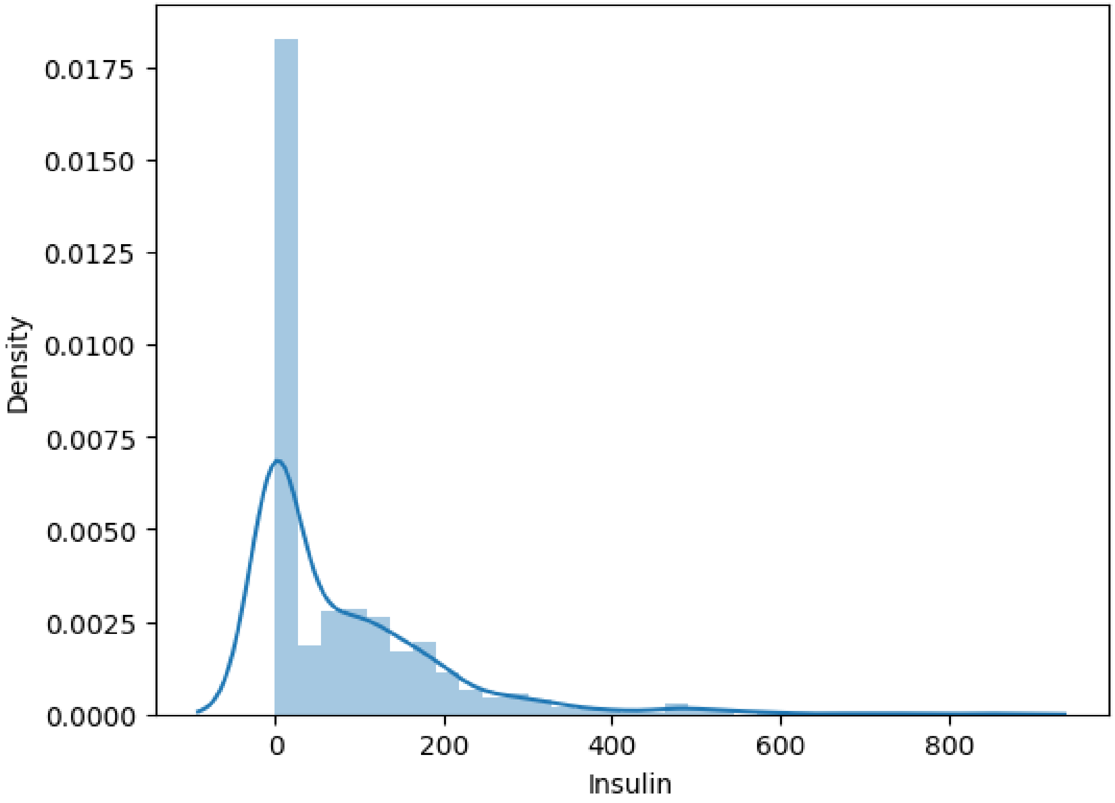
Most common causes of outliers on a data set:

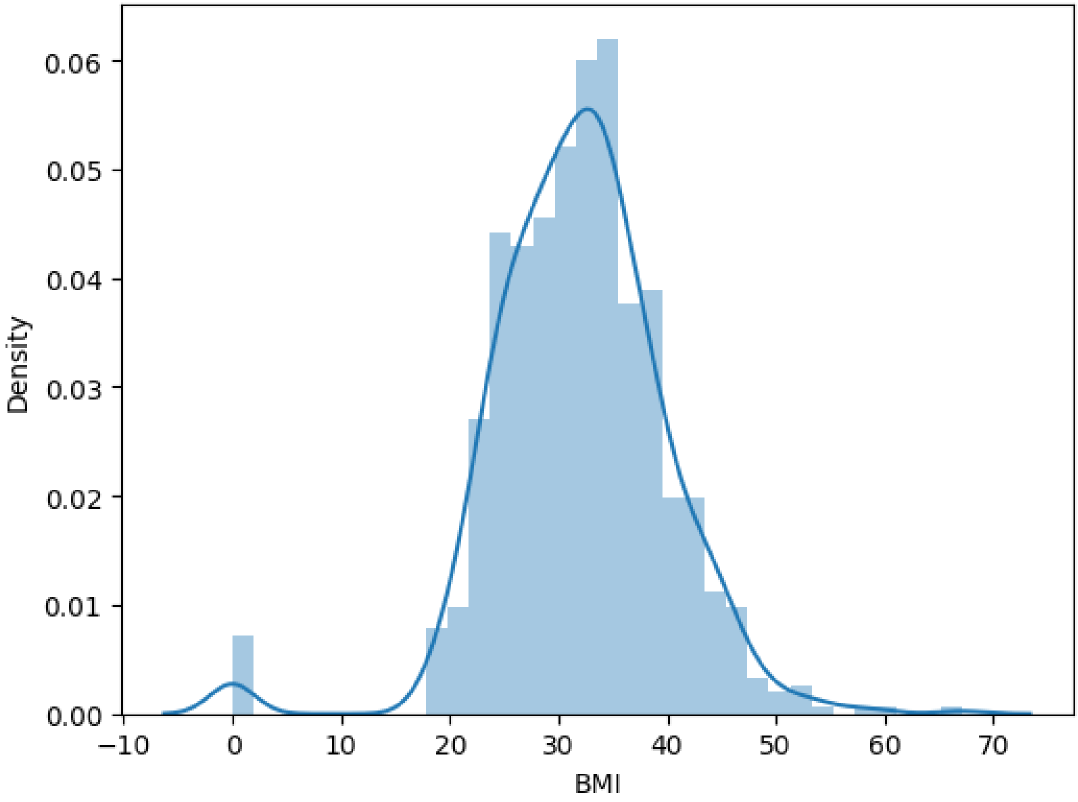
- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (data extraction or experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation or data set unintended mutations)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

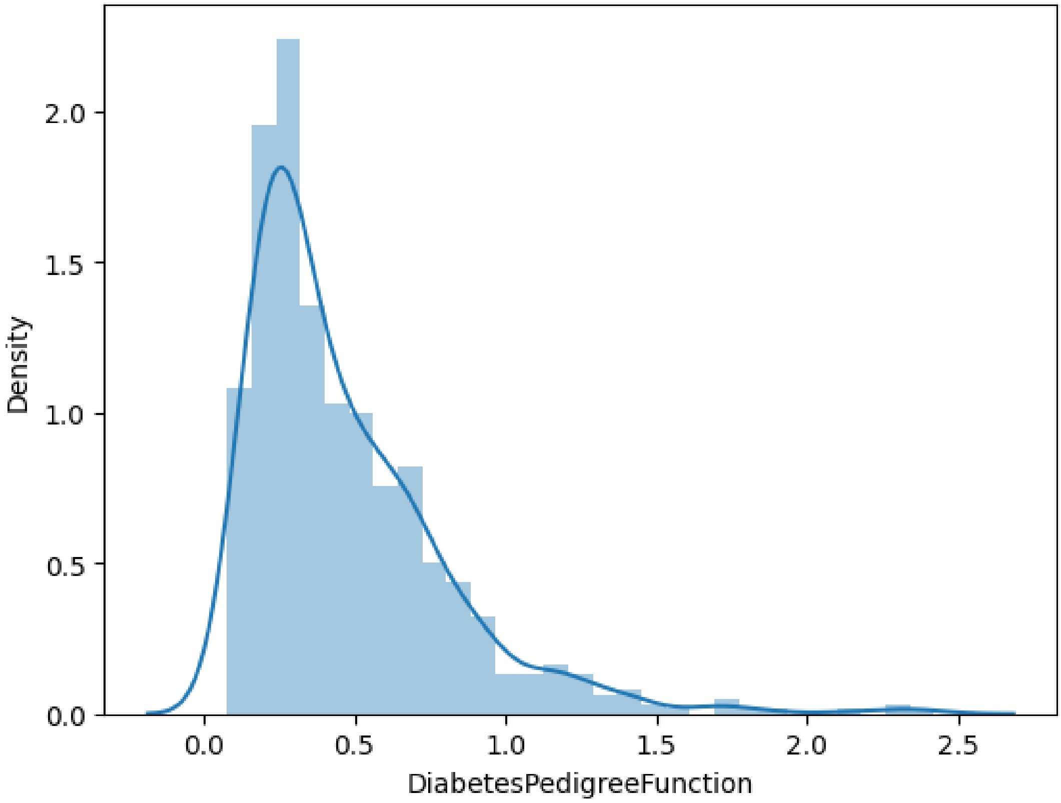
We plot distribution graph to visualise the outliers in diabetes dataset. The plots are given below:






0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Density
0
200
400
600
800
Insulin

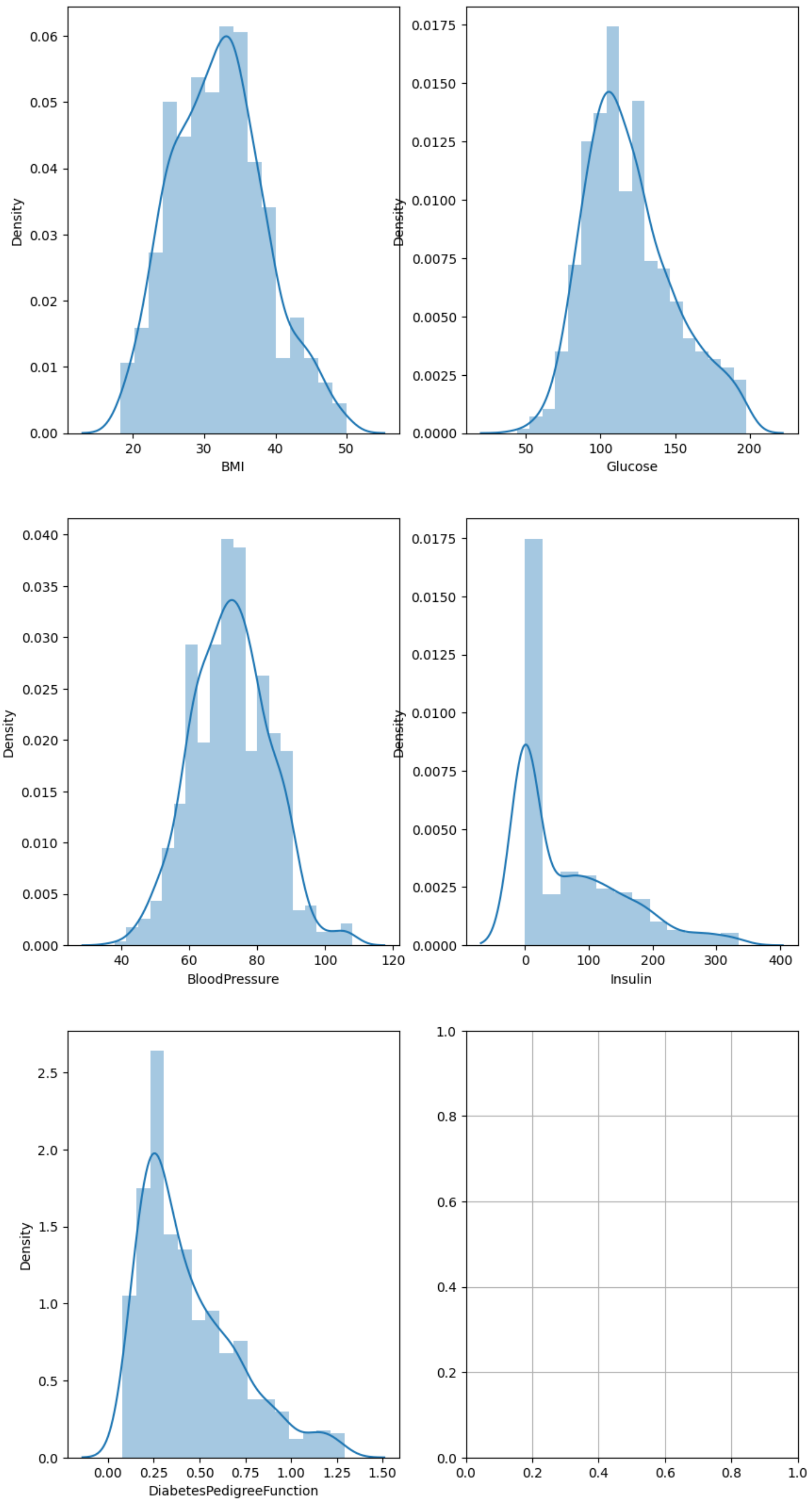

0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
−10
0
10
20
30
40
50
60
70
BMI



We have removed outliers using Z scores. We have created a function which will return a series where True represents a row that contains an outlier (absolute z-score greater than 3) and False does not.

We have also created a function that will iterate over all the columns and will provide us a dataframe denoting whether there is an outlier or not and lastly we will be filtering out the outliers.

After removing outliers:


Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
20
30
40
50
BMI
Density
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
50
100
150
200
Glucose
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
40
60
80
100
120
BloodPressure
Density
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0
100
200
300
400
Insulin
Density
0.0
0.5
1.0
1.5
2.0
2.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
DiabetesPedigreeFunction
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

MODEL BUILDING

Splitting data for training and testing purpose

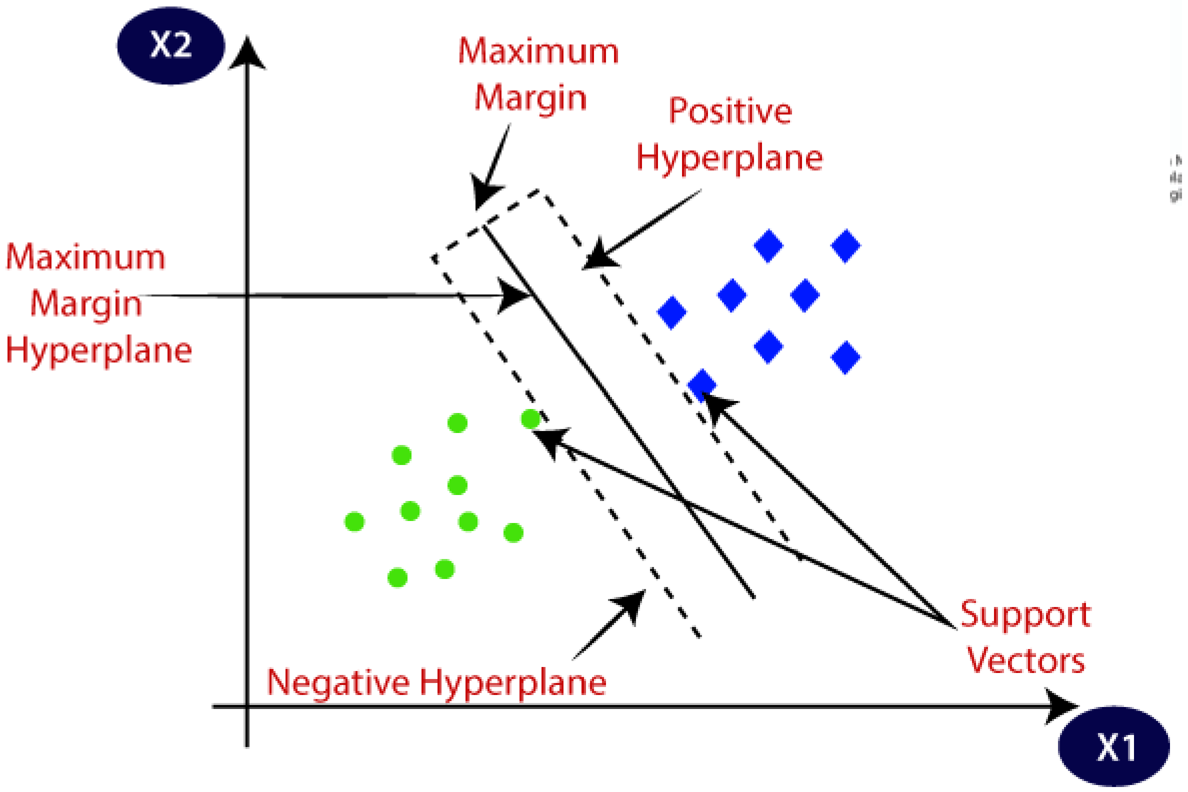
We split the given train dataset into two parts for training and testing purpose. The split ratio we used is 0.80 which indicates we used 80% data for training purpose and 20% data for testing purpose. We will be using the same split ratio for all the models trained. The same models are used for both the prediction systems. We will be looking at the diabetes system here.

SVM classifier

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Confusion Matrix:

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted	0	1
Actual		
0	82	8
1	23	20

Classification Report:

	precision	recall	f1-score	support
0	0.78	0.91	0.84	90
1	0.71	0.47	0.56	43
accuracy			0.77	133
macro avg	0.75	0.69	0.70	133
weighted avg	0.76	0.77	0.75	133

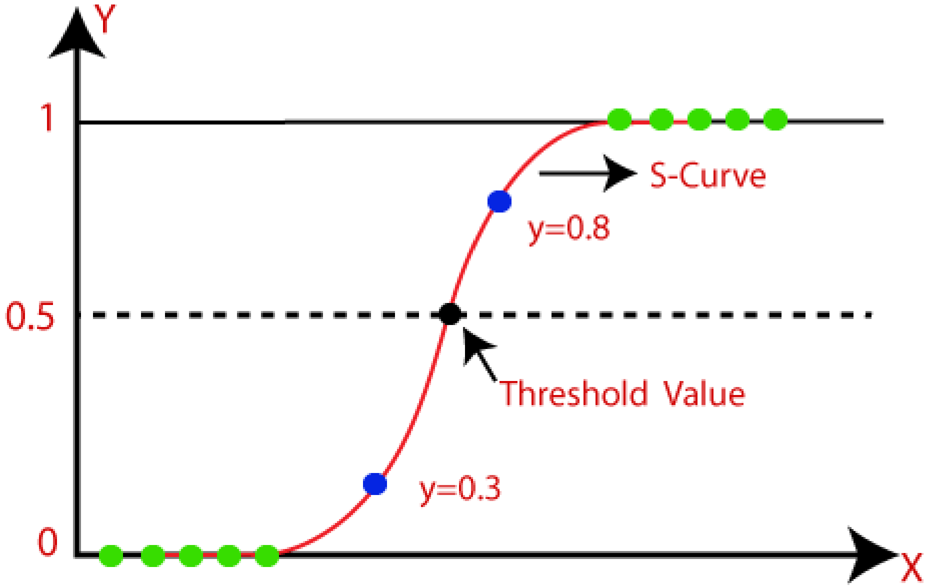
Logistic Regression model:

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.



Confusion matrix:

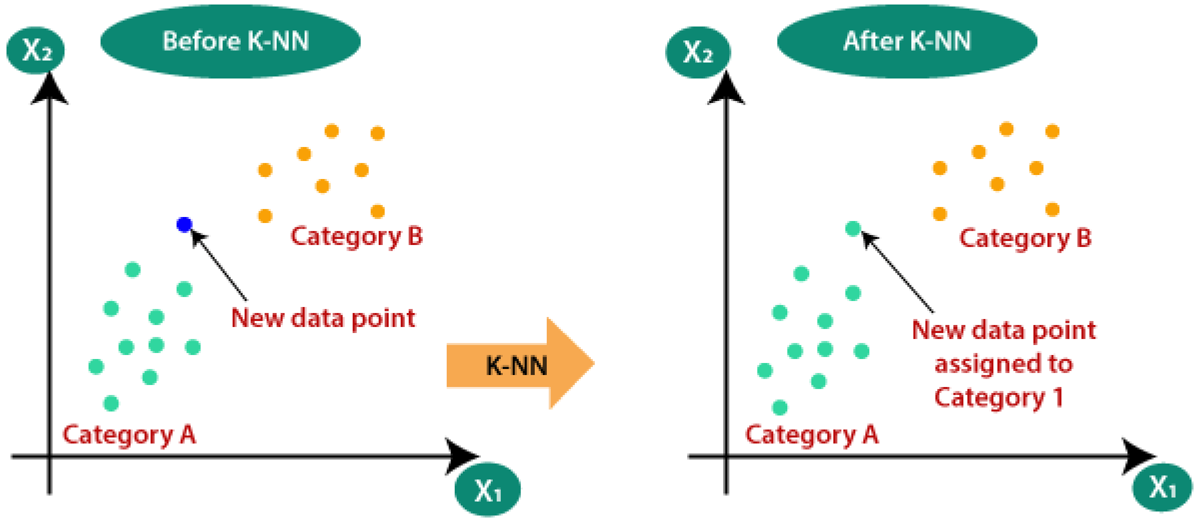
Predicted	0	1
Actual		
0	81	9
1	22	21

Classification Report:

	precision	recall	f1-score	support
0	0.79	0.90	0.84	90
1	0.70	0.49	0.58	43
accuracy			0.77	133
macro avg	0.74	0.69	0.71	133
weighted avg	0.76	0.77	0.75	133

KNN Classifier:

k-NN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression. In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). For e.g. if k = 1, then the object is simply assigned to the class of that single nearest neighbor. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.



The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Confusion Matrix:

Predicted	0	1
Actual		

0	83	7
1	25	18

Classification report:

	precision	recall	f1-score	support
0	0.77	0.92	0.84	90
1	0.72	0.42	0.53	43
accuracy			0.76	133
macro avg	0.74	0.67	0.68	133
weighted avg	0.75	0.76	0.74	133

Naïve Bayes Model:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem. The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

contribution to the outcome. The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Bayes Theorem:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$P(A|B) =P(A)P(B|A)/P(B)$

where A and B are events and P(B) is the probability of occurrence of event B.

Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence. P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B). P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen. The class-data relation from the Bayes Theorem can be obtained as follows:

P(Class|Data) =P(Class)P(Data|Class)/P(Data)

Where,

- P(Class|Data) = Posterior
- P(Class) = Prior
- P(Data|Class) = Likelihood
- P(Data) = Marginal Probability

In other words, it can be written as:

Posterior =Prior ∗ Likelihood/Marginal Probability

In application, we do not need to calculate the Marginal Probability for classification. We only need to calculate the numerator of the posterior for classification.

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

This is mostly used for document classification problem, i.e. whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoulli Naive Bayes:

This is similar to the multinomial Naive Bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

Confusion Matrix:

Predicted	0	1
Actual		
0	72	18
1	21	22

Classification report:

	precision	recall	f1-score	support
0	0.77	0.80	0.79	90
1	0.55	0.51	0.53	43
accuracy			0.71	133
macro avg	0.66	0.66	0.66	133
weighted avg	0.70	0.71	0.70	133

Similarly, all the models are trained in heart disease prediction system.

Comparison of the Models trained:

We trained 4 models using the 4 algorithms viz.

1. k-Nearest Neighbour

2. Gaussian Naive Bayes

3. SVM

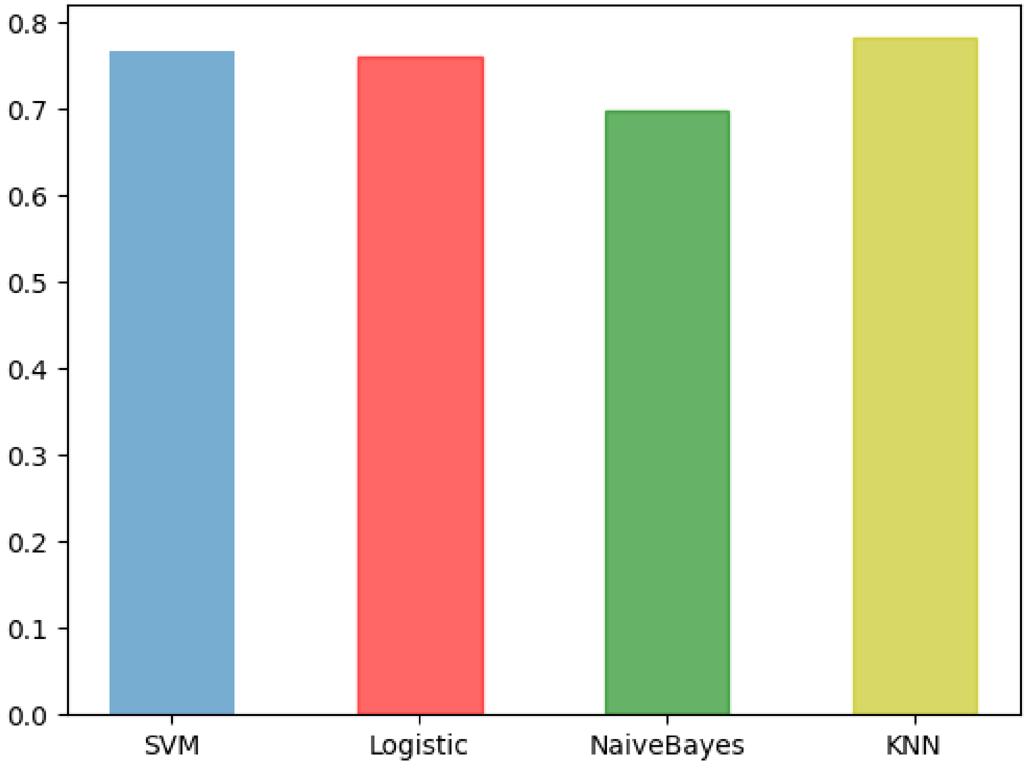
4. Logistic Regression

The 4 models had different accuracy.

For Diabetes Prediction:

The comparison of the accuracies of the models are

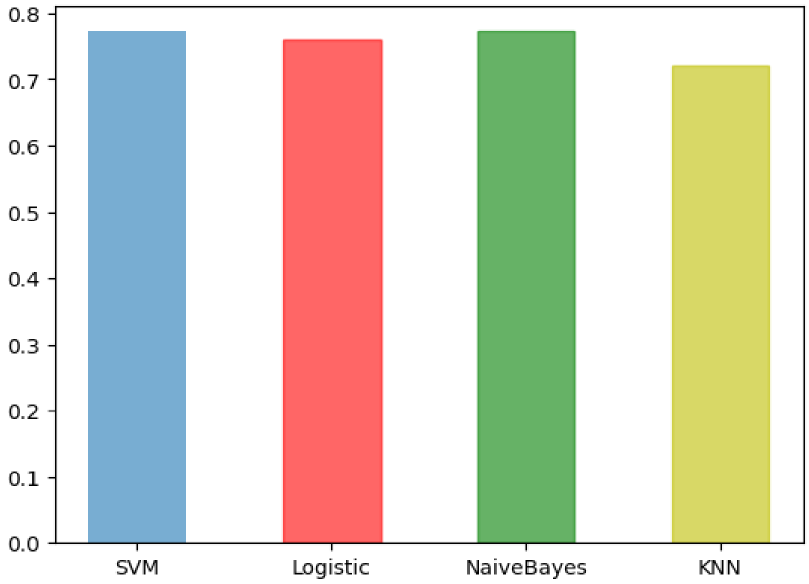
given below(**without outliers**):



Thus, from the above comparison we can see that the KNN model has the highest accuracy. So, our selected model is KNN Model.

Model	Accuracy %
k-Nearest Neighbour	78.195
Gaussian Naive Bayes	69.924
SVM	76.691
Logistic Regression	75.939

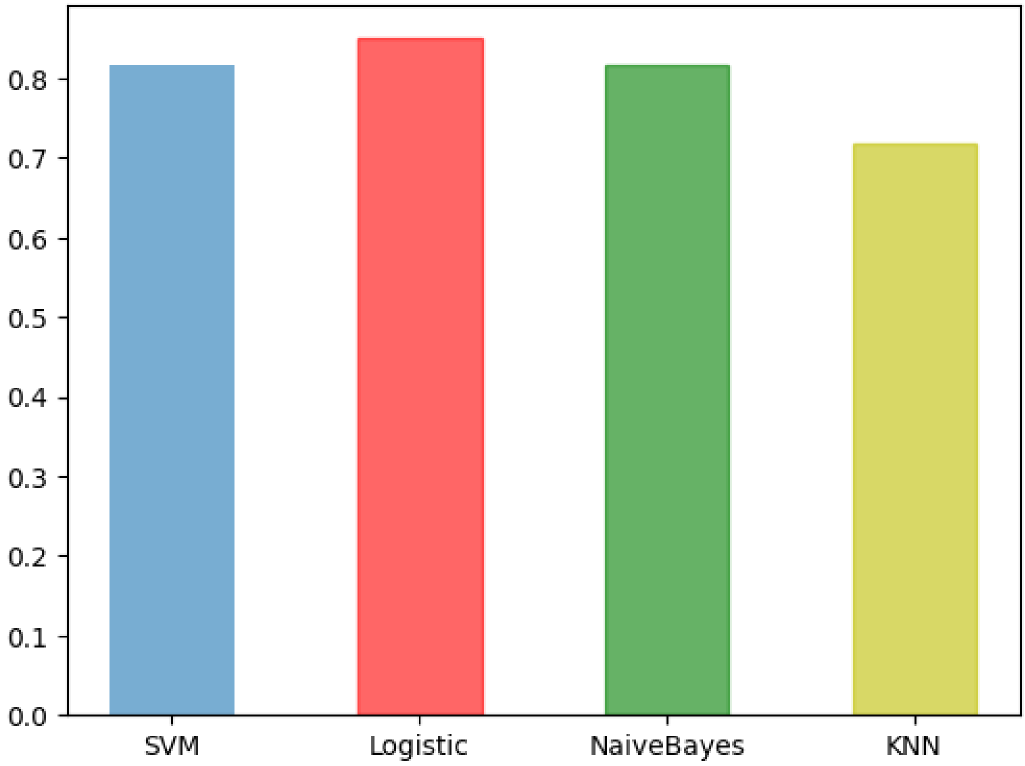
With outliers:



Model	Accuracy %
k-Nearest Neighbour	72.077
Gaussian Naive Bayes	77.272
SVM	77.272
Logistic Regression	75.974

For Heart Disease Prediction:

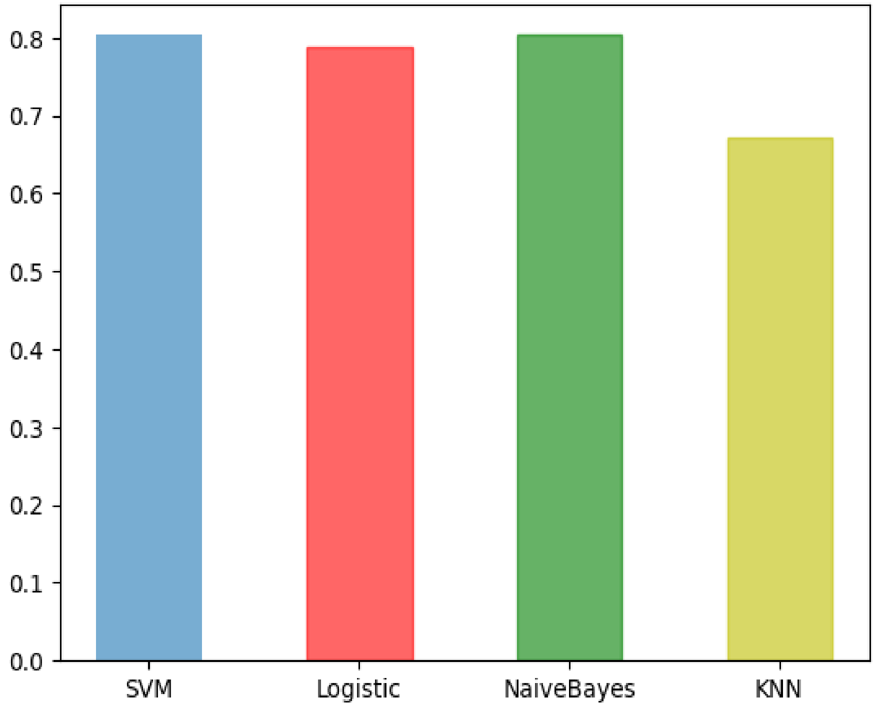
The comparison of the accuracies of the models are given below(**without outliers**):



Thus, from the above comparison we can see that the Logistic Regression model has the highest accuracy. So, our selected model is Logistic Regression Model.

Model	Accuracy %
k-Nearest Neighbour	71.667
Gaussian Naive Bayes	81.667
SVM	81.667
Logistic Regression	85

The comparison of the accuracies of the models (**with outliers**):



Model	Accuracy %
k-Nearest Neighbour	67.213
Gaussian Naive Bayes	80.327
SVM	80.327
Logistic Regression	78.688

CODES

MULTIPLE DISEASE PREDICTION SYSTEM:

DIABETES PREDICTION SYSTEM:

Importing the dependencies

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

Data Collection And Analysis PIMA Diabetes Dataset

```
#loading diabetes dataset to pandas dataframe
df=pd.read_csv('/content/diabetes.csv')
diabetes_df=df.copy()
```

```
diabetes_df
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72           35.0        0  33.6   
1              1       85             66           29.0        0  26.6   
2              8      183             64            NaN        0  23.3   
3              1       89             66           23.0       94  28.1   
4              0      137             40           35.0      168  43.1   
..           ...      ...            ...            ...      ...   ...   
763           10      101             76           48.0      180  32.9   
764            2      122             70           27.0        0  36.8   
765            5      121             72           23.0      112  26.2   
766            1      126             60            0.0        0  30.1   
767            1       93             70           31.0        0  30.4   

     DiabetesPedigreeFunction   Age  Outcome  
0                       0.627  50.0        1  
1                       0.351  31.0        0  
2                       0.672   NaN        1  
3                       0.167  21.0        0  
4                       2.288  33.0        1  
..                        ...   ...      ...  
763                     0.171  63.0        0  
764                     0.340  27.0        0  
765                     0.245  30.0        0  
766                     0.349  47.0        1  
767                     0.315  23.0        0  

[768 rows x 9 columns]
```

```
diabetes_df.isnull().sum()
```

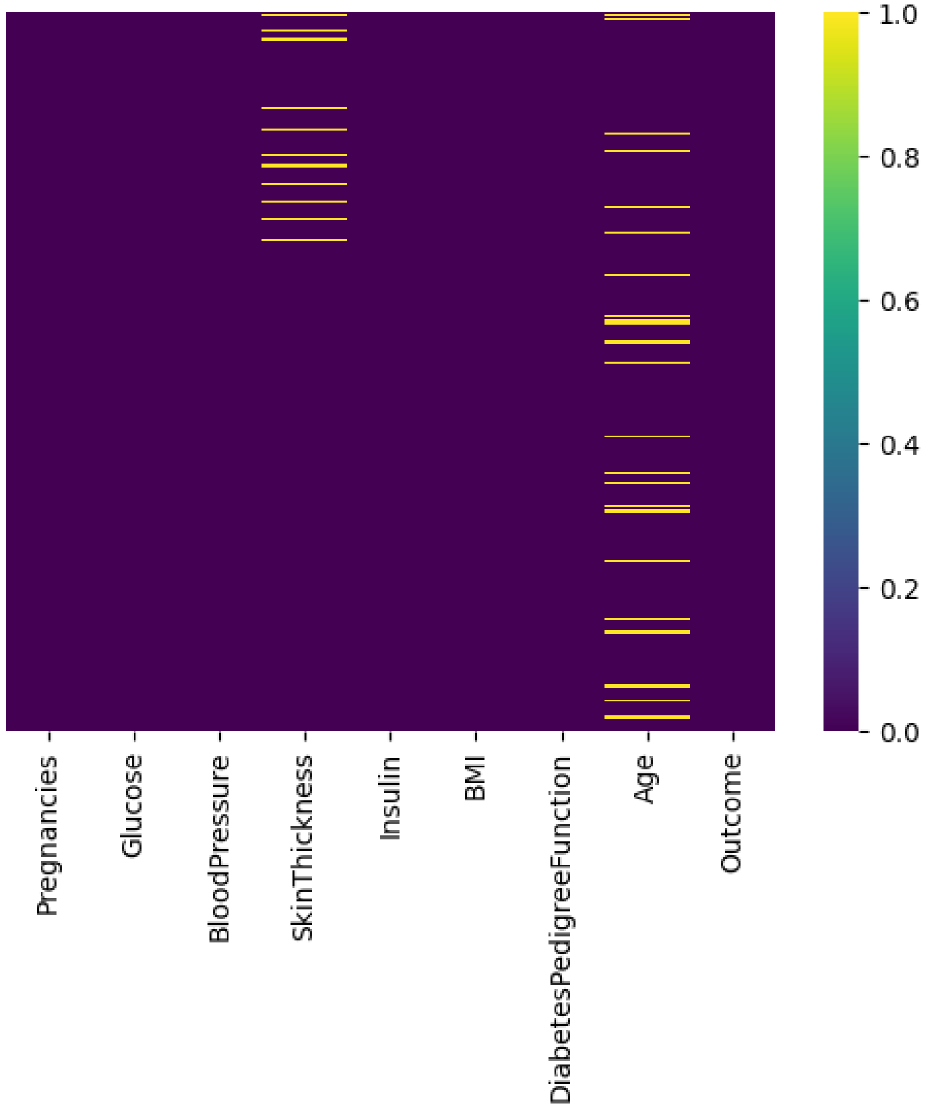
```
Pregnancies                  0
Glucose                      0
BloodPressure                0
SkinThickness               25
Insulin                      0
BMI                          0
DiabetesPedigreeFunction     0
Age                         71
Outcome                      0
dtype: int64
```

```
diabetes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             743 non-null    float64
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       697 non-null    float64
 8   Outcome                   768 non-null    int64  
dtypes: float64(4), int64(5)
memory usage: 54.1 KB
```

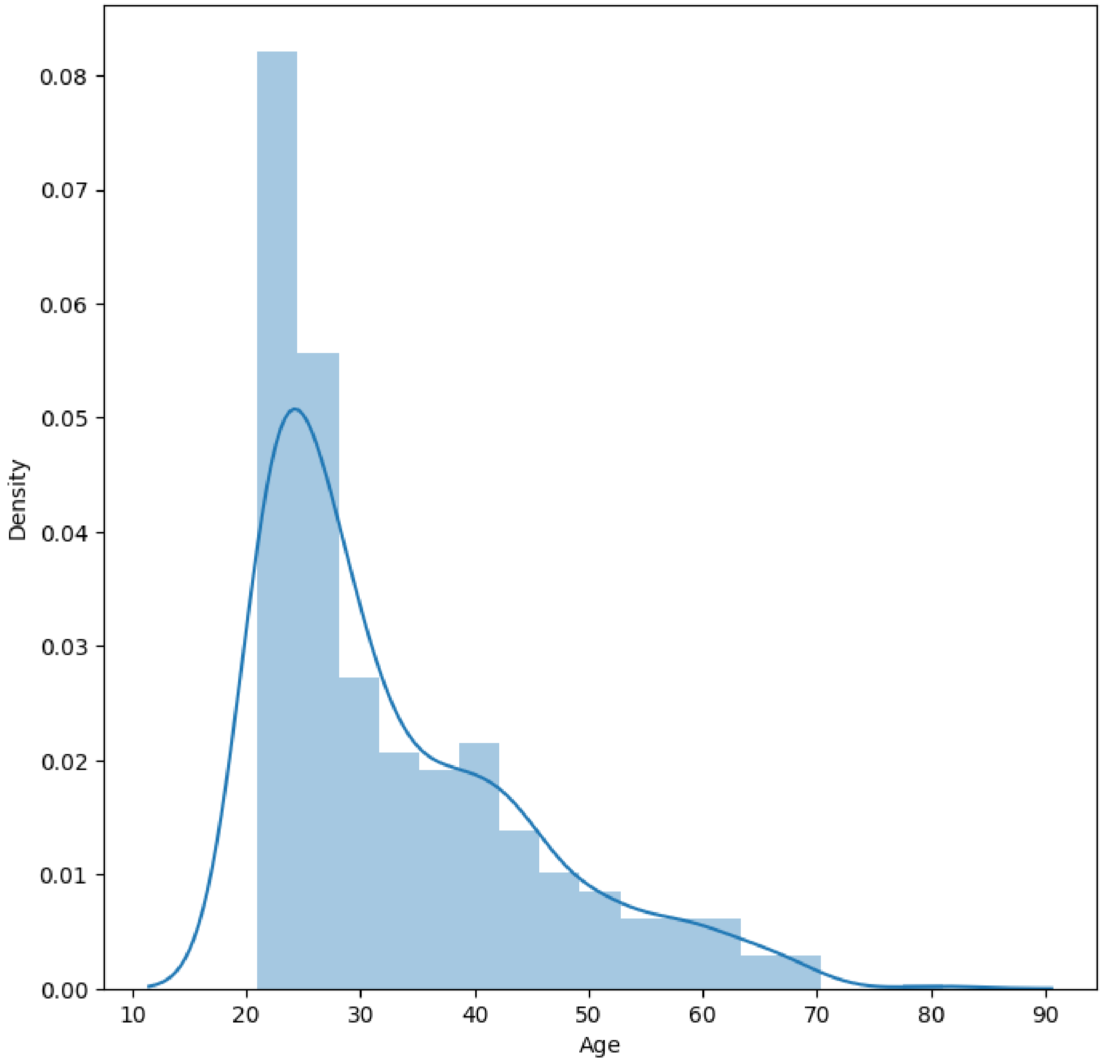
```
#heatmap
import seaborn as sns
sns.heatmap(diabetes_df.isnull(),yticklabels=False,cbar=True,cmap="viridis
")
```

```
<Axes: >
```



```
#distribution of data in age
fig,ax=plt.subplots(figsize=(8,8))
sns.distplot(diabetes_df.Age)


<ipython-input-512-a3786ae00d7b>:3: 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(diabetes_df.Age)
```

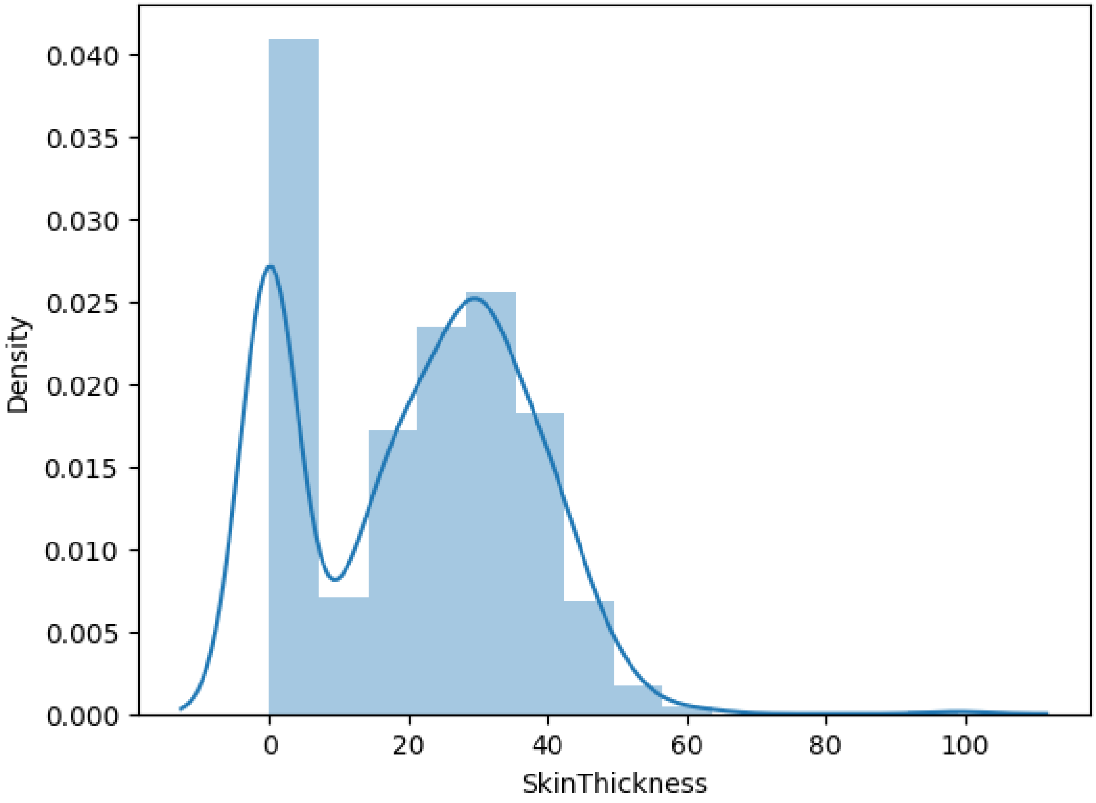
```
<Axes: xlabel='Age', ylabel='Density'>
```



```
sns.distplot(diabetes_df.SkinThickness)
```

```
<ipython-input-513-2fcc0f9ea930>: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(diabetes_df.SkinThickness)
```

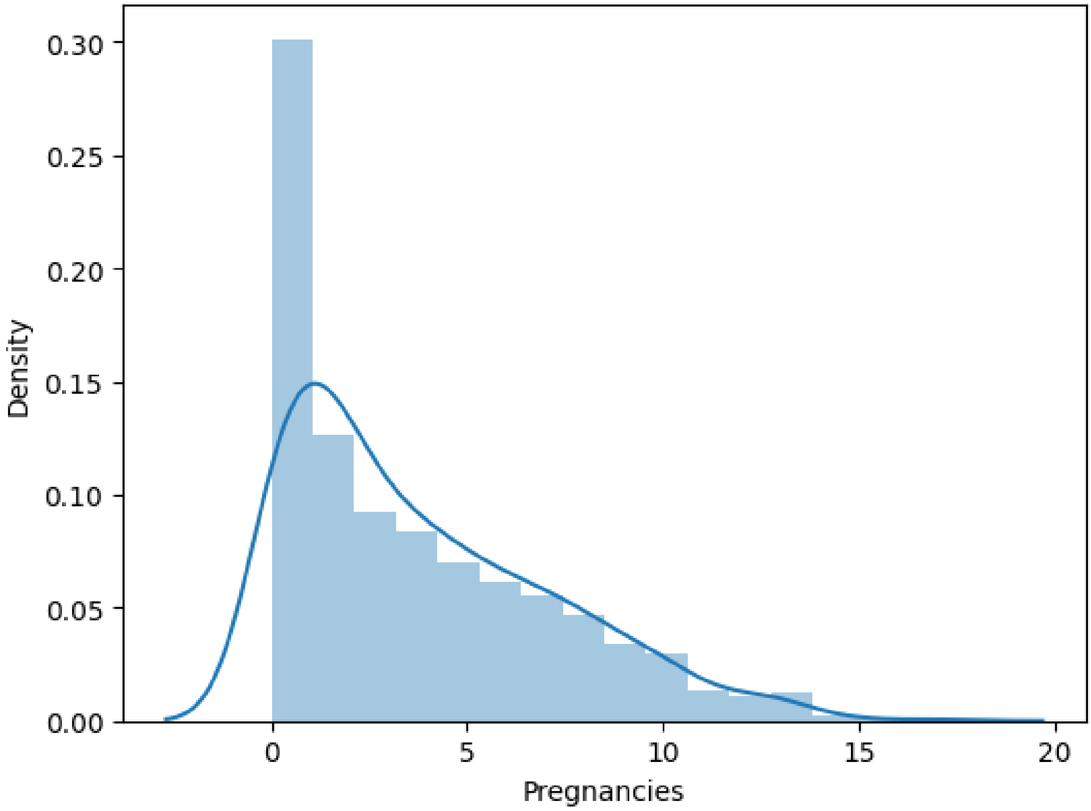
```
<Axes: xlabel='SkinThickness', ylabel='Density'>
```



```
sns.distplot(diabetes_df.Pregnancies)
```

```
<ipython-input-514-6d99d7eb254e>: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(diabetes_df.Pregnancies)
```

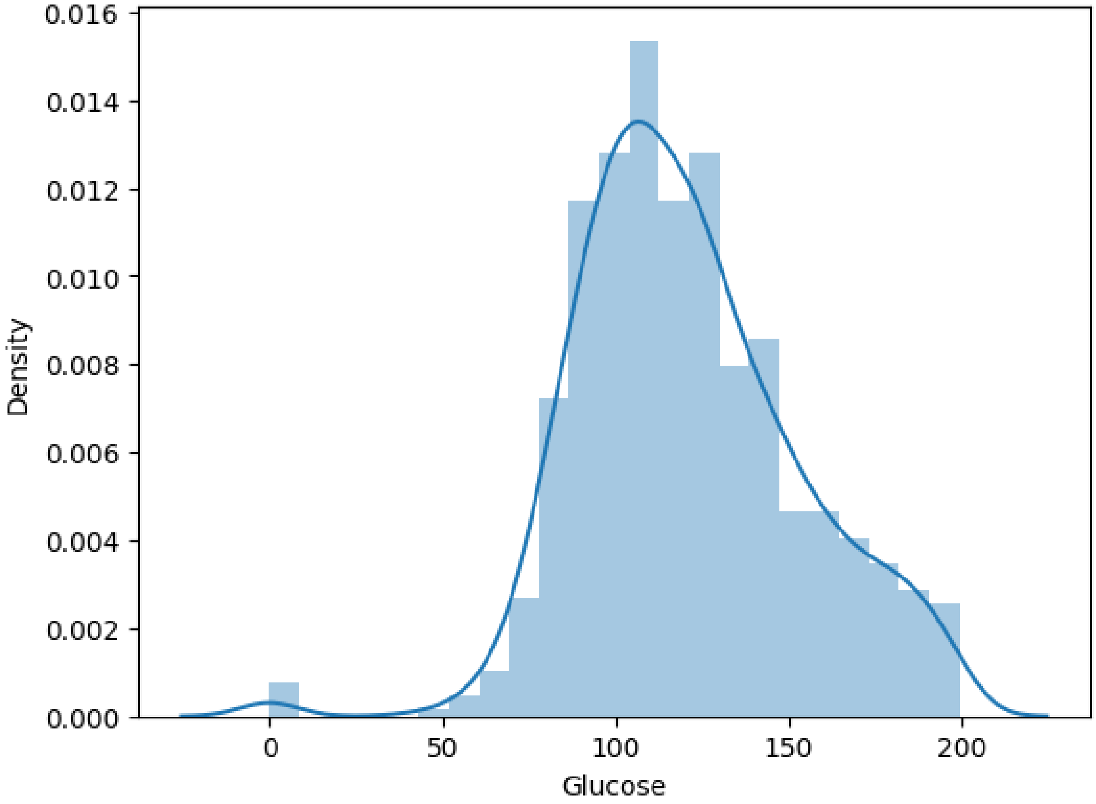
```
<Axes: xlabel='Pregnancies', ylabel='Density'>
```



```
sns.distplot(diabetes_df.Glucose)
```

```
<ipython-input-515-6aef5e1f755c>: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(diabetes_df.Glucose)
```

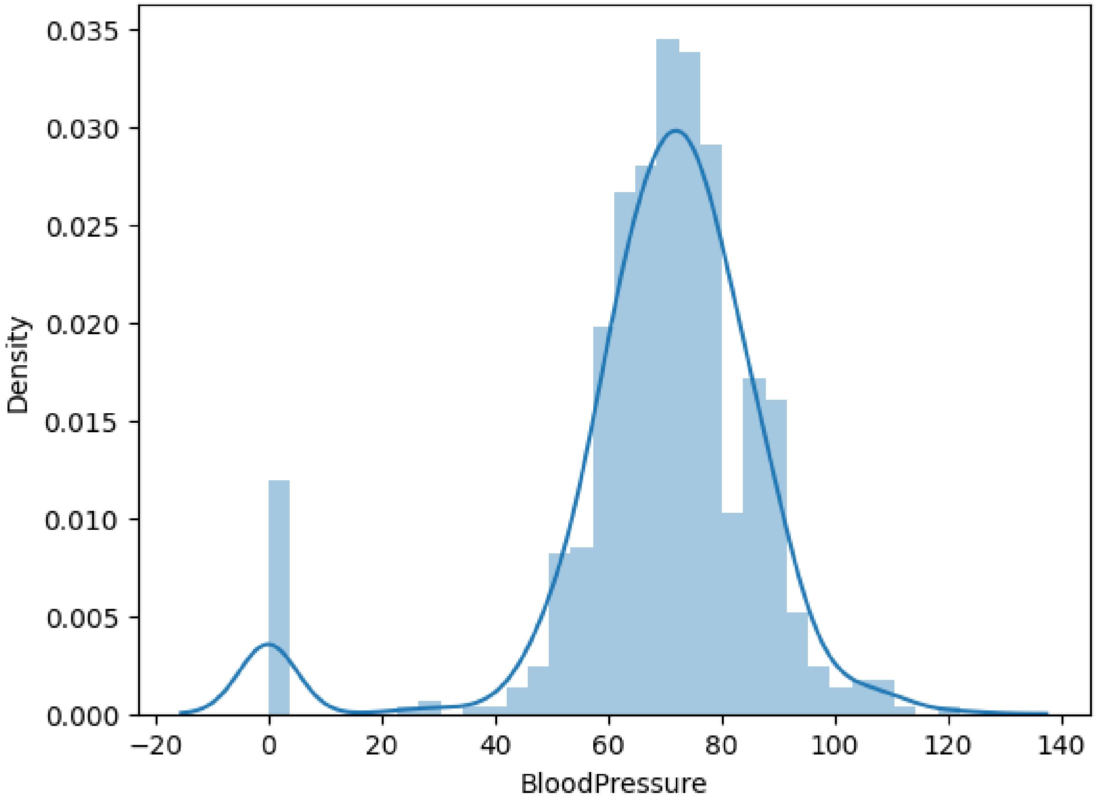
```
<Axes: xlabel='Glucose', ylabel='Density'>
```



```
sns.distplot(diabetes_df.BloodPressure)
```

```
<ipython-input-516-9385a01ab079>: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(diabetes_df.BloodPressure)
```

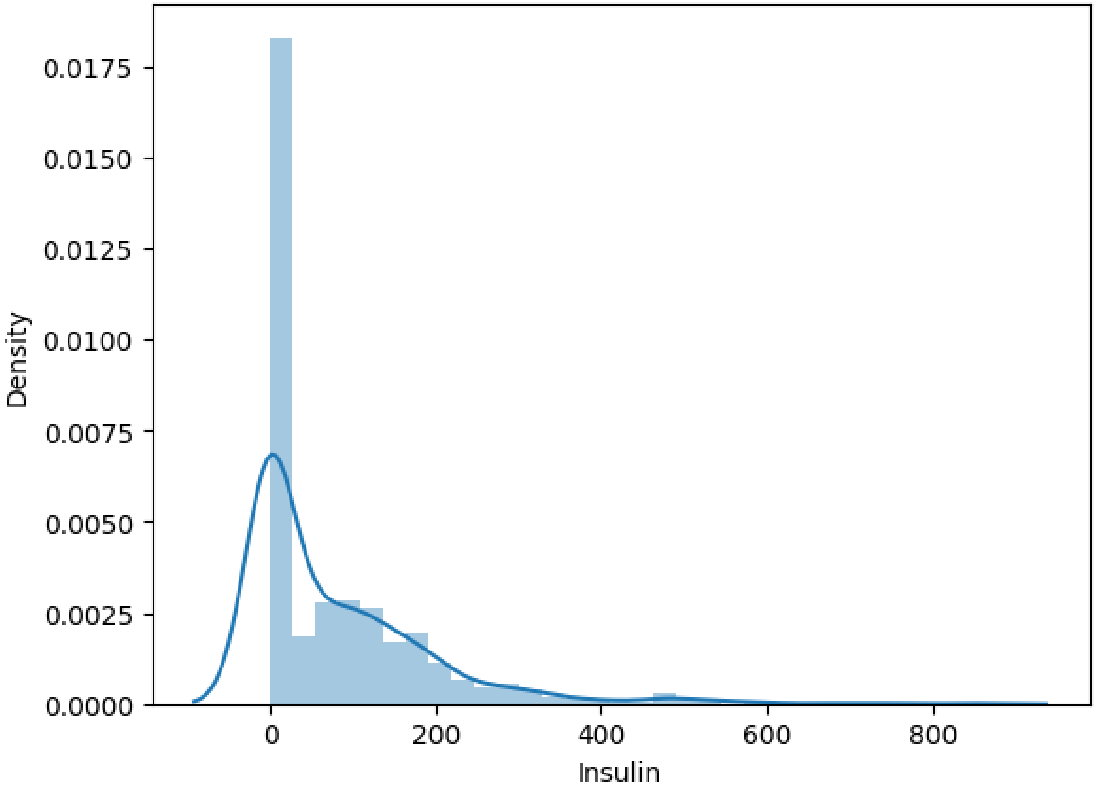
```
<Axes: xlabel='BloodPressure', ylabel='Density'>
```



```
sns.distplot(diabetes_df.Insulin)
```

```
<ipython-input-517-80c25523b1df>: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(diabetes_df.Insulin)
```

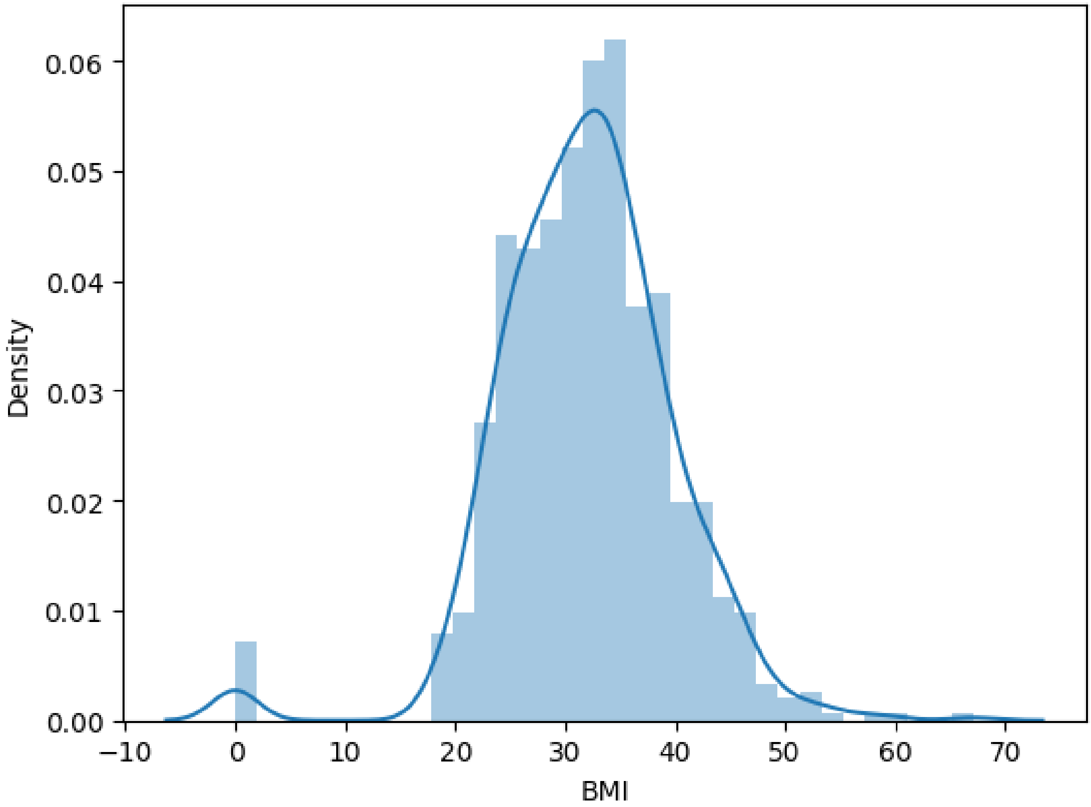
```
<Axes: xlabel='Insulin', ylabel='Density'>
```



```
sns.distplot(diabetes_df.BMI)
```

```
<ipython-input-518-d512f55bcf04>: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(diabetes_df.BMI)
```

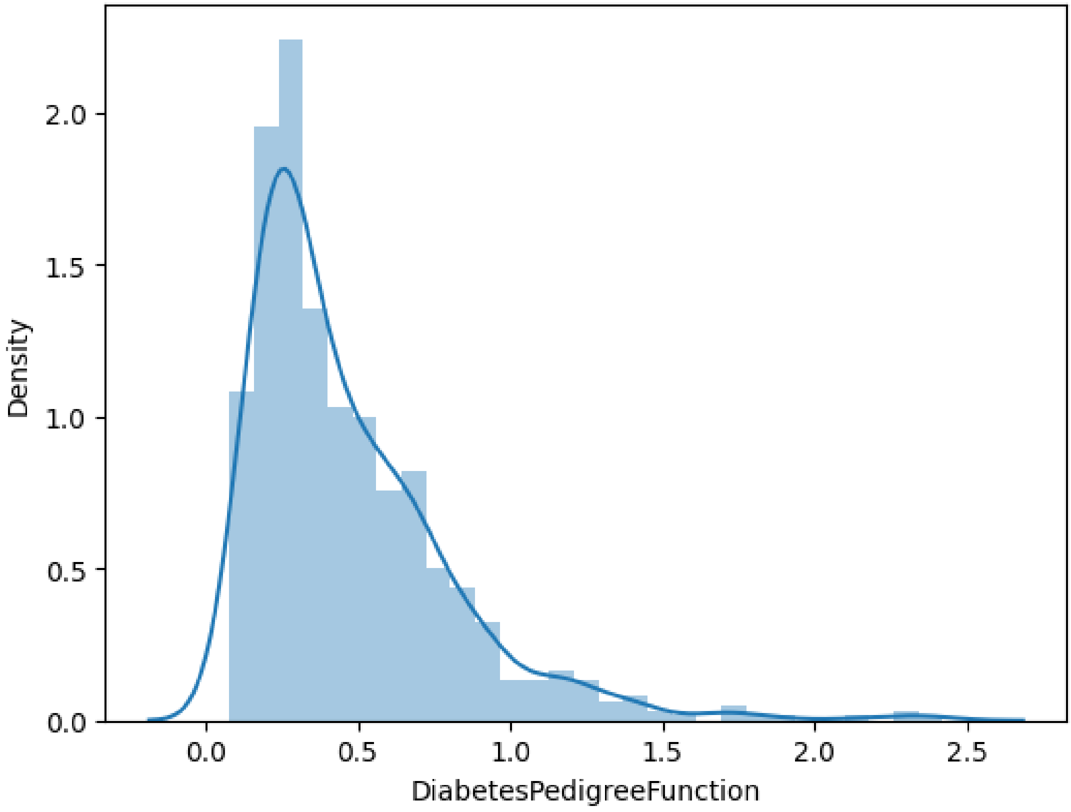
```
<Axes: xlabel='BMI', ylabel='Density'>
```



```
sns.distplot(diabetes_df.DiabetesPedigreeFunction)
```

```
<ipython-input-519-d282ce82f0d2>: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(diabetes_df.DiabetesPedigreeFunction)
```

```
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Density'>
```



```
#replace the missing values with median values
diabetes_df['Age'].fillna(diabetes_df['Age'].median(),inplace=True)
diabetes_df['SkinThickness'].fillna(diabetes_df['SkinThickness'].median(),
inplace=True)

def find_outliers(col):
  from scipy import stats
  z=np.abs(stats.zscore(col))
  idx_outliers=np.where(z>3,True,False)
  return pd.Series(idx_outliers,index=col.index)
idx=find_outliers(diabetes_df.BMI)
idx1=find_outliers(diabetes_df.Glucose)
idx2=find_outliers(diabetes_df.BloodPressure)
idx3=find_outliers(diabetes_df.Insulin)
idx4=find_outliers(diabetes_df.DiabetesPedigreeFunction)
idx
```

```
0      False
1      False
2      False
3      False
4      False
       ...
763    False
764    False
765    False
766    False
```

```
767    False
Length: 768, dtype: bool
```

```
display(diabetes_df.loc[idx==True].describe().round(3))
display(diabetes_df.loc[idx1==True].describe().round(3))
display(diabetes_df.loc[idx2==True].describe().round(3))
display(diabetes_df.loc[idx3==True].describe().round(3))
display(diabetes_df.loc[idx4==True].describe().round(3))
```

```
       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin     BMI  
\
count       14.000   14.000         14.000         14.000   14.000  14.000
mean         3.286  112.786         43.214         13.571   33.786  13.129
std          3.384   27.096         46.161         21.103   71.595  26.167
min          0.000   74.000          0.000          0.000    0.000   0.000
25%          0.000   96.000          0.000          0.000    0.000   0.000
50%          2.500  114.500         32.000          0.000    0.000   0.000
75%          5.750  124.500         81.000         23.000   10.500   0.000
max         10.000  180.000        110.000         63.000  240.000  67.100

       DiabetesPedigreeFunction     Age  Outcome
count                    14.000  14.000   14.000
mean                      0.599  29.143    0.286
std                       0.673  14.250    0.469
min                       0.102  21.000    0.000
25%                       0.238  22.000    0.000
50%                       0.304  24.500    0.000
75%                       0.623  26.000    0.750
max                       2.420  69.000    1.000

       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin     BMI  
\
count         5.00      5.0          5.000          5.000    5.000   5.000
mean          2.80      0.0         67.600         30.200    4.600  32.880
std           2.49      0.0         12.033          8.643   10.286   7.034
min           1.00      0.0         48.000         20.000    0.000  24.700
25%           1.00      0.0         68.000         23.000    0.000  27.700
50%           1.00      0.0         68.000         32.000    0.000  32.000
75%           5.00      0.0         74.000         35.000    0.000  39.000
max           6.00      0.0         80.000         41.000   23.000  41.000

       DiabetesPedigreeFunction     Age  Outcome
count                     5.000   5.000    5.000
mean                      0.380  27.800    0.400
std                       0.216   6.221    0.548
min                       0.140  22.000    0.000
25%                       0.299  22.000    0.000
50%                       0.346  29.000    0.000
75%                       0.389  29.000    1.000
max                       0.727  37.000    1.000

       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin     BMI  
\
count       35.000   35.000           35.0         35.000     35.0  35.000
```

```
mean         3.629  117.800            0.0          1.514      0.0  25.706
std          3.647   27.489            0.0          6.298      0.0  14.875
min          0.000   73.000            0.0          0.000      0.0   0.000
25%          0.000   97.500            0.0          0.000      0.0  21.650
50%          2.000  117.000            0.0          0.000      0.0  28.900
75%          6.000  133.500            0.0          0.000      0.0  34.550
max         13.000  183.000            0.0         30.000      0.0  52.300

       DiabetesPedigreeFunction     Age  Outcome
count                    35.000  35.000   35.000
mean                      0.388  29.257    0.457
std                       0.254   6.294    0.505
min                       0.102  21.000    0.000
25%                       0.198  25.000    0.000
50%                       0.282  29.000    0.000
75%                       0.573  31.000    1.000
max                       0.933  44.000    1.000
       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin     BMI  
\
count       18.000   18.000         18.000         18.000   18.000  18.000
mean         3.111  165.833         73.444         35.333  548.833  36.961
std          2.888   20.057         11.638          9.107  107.813   5.988
min          0.000  124.000         52.000         23.000  440.000  28.700
25%          1.000  155.000         63.500         26.250  480.000  31.375
50%          2.000  168.500         72.000         35.500  502.500  37.550
75%          4.750  180.000         83.500         43.500  570.500  40.300
max          8.000  197.000         90.000         49.000  846.000  52.300

       DiabetesPedigreeFunction     Age  Outcome
count                    18.000  18.000   18.000
mean                      0.661  34.722    0.667
std                       0.625  13.663    0.485
min                       0.128  21.000    0.000
25%                       0.244  23.500    0.000
50%                       0.540  29.500    1.000
75%                       0.687  45.250    1.000
max                       2.329  60.000    1.000
       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin     BMI  
\
count       11.000   11.000         11.000         11.000   11.000  11.000
mean         1.909  140.909         67.273         30.182  170.000  33.055
std          2.468   37.920         13.154         16.940  240.219  15.139
min          0.000   82.000         40.000          0.000    0.000   0.000
25%          0.000  118.000         61.000         21.000    0.000  25.950
50%          2.000  137.000         70.000         24.000   89.000  36.700
75%          2.500  176.500         77.000         39.000  221.500  41.250
max          8.000  197.000         82.000         63.000  744.000  59.400

       DiabetesPedigreeFunction     Age  Outcome
count                    11.000  11.000   11.000
mean                      1.914  28.273    0.364
std                       0.324   6.198    0.505
```

```
min                          1.476  21.000    0.000
25%                          1.699  25.000    0.000
50%                          1.781  25.000    0.000
75%                          2.212  30.000    1.000
max                          2.420  44.000    1.000
```

```
display(diabetes_df.loc[idx==False].describe().round(3))
diabetes_df=diabetes_df.loc[idx==False]
display(diabetes_df.loc[idx1==False].describe().round(3))
diabetes_df=diabetes_df.loc[idx1==False]
display(diabetes_df.loc[idx2==False].describe().round(3))
diabetes_df=diabetes_df.loc[idx2==False]
display(diabetes_df.loc[idx3==False].describe().round(3))
diabetes_df=diabetes_df.loc[idx3==False]
display(diabetes_df.loc[idx4==False].describe().round(3))
diabetes_df=diabetes_df.loc[idx4==False]
```

```
       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  
BMI  \
count      754.000  754.000        754.000        754.000  754.000  
754.000
mean         3.855  121.045         69.586         21.077   80.654  
32.343
std          3.371   32.052         18.224         15.548  115.756  
6.690
min          0.000    0.000          0.000          0.000    0.000  
18.200
25%          1.000   99.000         64.000          0.000    0.000  
27.500
50%          3.000  117.000         72.000         23.000   36.500  
32.250
75%          6.000  141.000         80.000         32.000  128.750  
36.600
max         17.000  199.000        122.000         99.000  846.000  
55.000

       DiabetesPedigreeFunction      Age  Outcome
count                   754.000  754.000  754.000
mean                      0.470   32.798    0.350
std                       0.322   11.241    0.477
min                       0.078   21.000    0.000
25%                       0.244   24.000    0.000
50%                       0.376   29.000    0.000
75%                       0.626   39.000    1.000
max                       2.329   81.000    1.000

       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  
BMI  \
count      749.000  749.000        749.000        749.000  749.000  
749.000
mean         3.862  121.853         69.599         21.016   81.162  
32.339
std          3.376   30.588         18.263         15.569  115.972  
6.692
```

```
min          0.000   44.000          0.000          0.000    0.000
18.200
25%          1.000   99.000         64.000          0.000    0.000
27.500
50%          3.000  117.000         72.000         23.000   38.000
32.300
75%          6.000  141.000         80.000         32.000  130.000
36.600
max         17.000  199.000        122.000         99.000  846.000
55.000

       DiabetesPedigreeFunction      Age  Outcome
count                   749.000  749.000  749.000
mean                      0.470   32.832    0.350
std                       0.323   11.262    0.477
min                       0.078   21.000    0.000
25%                       0.244   24.000    0.000
50%                       0.376   29.000    0.000
75%                       0.626   39.000    1.000
max                       2.329   81.000    1.000

       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
count      721.000  721.000        721.000        721.000  721.000
721.000
mean         3.878  121.791         72.302         21.759   84.313
32.347
std          3.363   30.737         12.281         15.336  117.075
6.642
min          0.000   44.000         24.000          0.000    0.000
18.200
25%          1.000   99.000         64.000          8.000    0.000
27.500
50%          3.000  117.000         72.000         23.000   48.000
32.300
75%          6.000  142.000         80.000         33.000  130.000
36.600
max         17.000  199.000        122.000         99.000  846.000
55.000

       DiabetesPedigreeFunction      Age  Outcome
count                   721.000  721.000  721.000
mean                      0.472   32.922    0.343
std                       0.325   11.404    0.475
min                       0.078   21.000    0.000
25%                       0.245   24.000    0.000
50%                       0.378   29.000    0.000
75%                       0.626   39.000    1.000
max                       2.329   81.000    1.000

       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
count      703.000  703.000        703.000        703.000  703.000
703.000
```

```
mean          3.898  120.663          72.273          21.411   72.420
32.229
std           3.374   30.136          12.304          15.309   90.011
6.619
min           0.000   44.000          24.000           0.000    0.000
18.200
25%           1.000   99.000          64.000           0.000    0.000
27.400
50%           3.000  116.000          72.000          23.000   43.000
32.000
75%           6.000  139.000          80.000          32.000  125.500
36.400
max          17.000  199.000         122.000          99.000  415.000
55.000

       DiabetesPedigreeFunction      Age  Outcome
count                   703.000  703.000  703.000
mean                      0.467   32.876    0.334
std                       0.313   11.348    0.472
min                       0.078   21.000    0.000
25%                       0.245   24.000    0.000
50%                       0.375   29.000    0.000
75%                       0.619   39.000    1.000
max                       2.288   81.000    1.000

       Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
count      696.000  696.000        696.000        696.000  696.000
696.000
mean         3.917  120.609         72.359         21.398   72.346
32.224
std          3.376   30.126         12.260         15.341   89.881
6.602
min          0.000   44.000         24.000          0.000    0.000
18.200
25%          1.000   99.000         64.000          0.000    0.000
27.400
50%          3.000  115.000         72.000         23.000   43.500
32.050
75%          6.000  139.000         80.000         32.000  125.250
36.325
max         17.000  199.000        122.000         99.000  415.000
55.000

       DiabetesPedigreeFunction      Age  Outcome
count                   696.000  696.000  696.000
mean                      0.454   32.907    0.335
std                       0.284   11.383    0.472
min                       0.078   21.000    0.000
25%                       0.245   24.000    0.000
50%                       0.370   29.000    0.000
75%                       0.605   39.000    1.000
max                       1.461   81.000    1.000
```

```
df_outliers=pd.DataFrame()
for col in df.describe().columns:
  df_outliers[col]=find_outliers(diabetes_df[col])
df_outliers.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  \
0        False    False          False          False    False  False   
1        False    False          False          False    False  False   
2        False    False          False          False    False  False   
3        False    False          False          False    False  False   
5        False    False          False          False    False  False   

   DiabetesPedigreeFunction    Age  Outcome  
0                     False  False    False  
1                     False  False    False  
2                     False  False    False  
3                     False  False    False  
5                     False  False    False  
```

```
test_outs=df_outliers.apply(lambda x:np.any(x),axis=1)
```

```
print(len(test_outs),df_outliers.shape)
test_outs
```

```
696 (696, 9)
```

```
0      False
1      False
2      False
3      False
5      False
       ...  
763    False
764    False
765    False
766    False
767    False
Length: 696, dtype: bool
```

```
np.sum(test_outs)
```

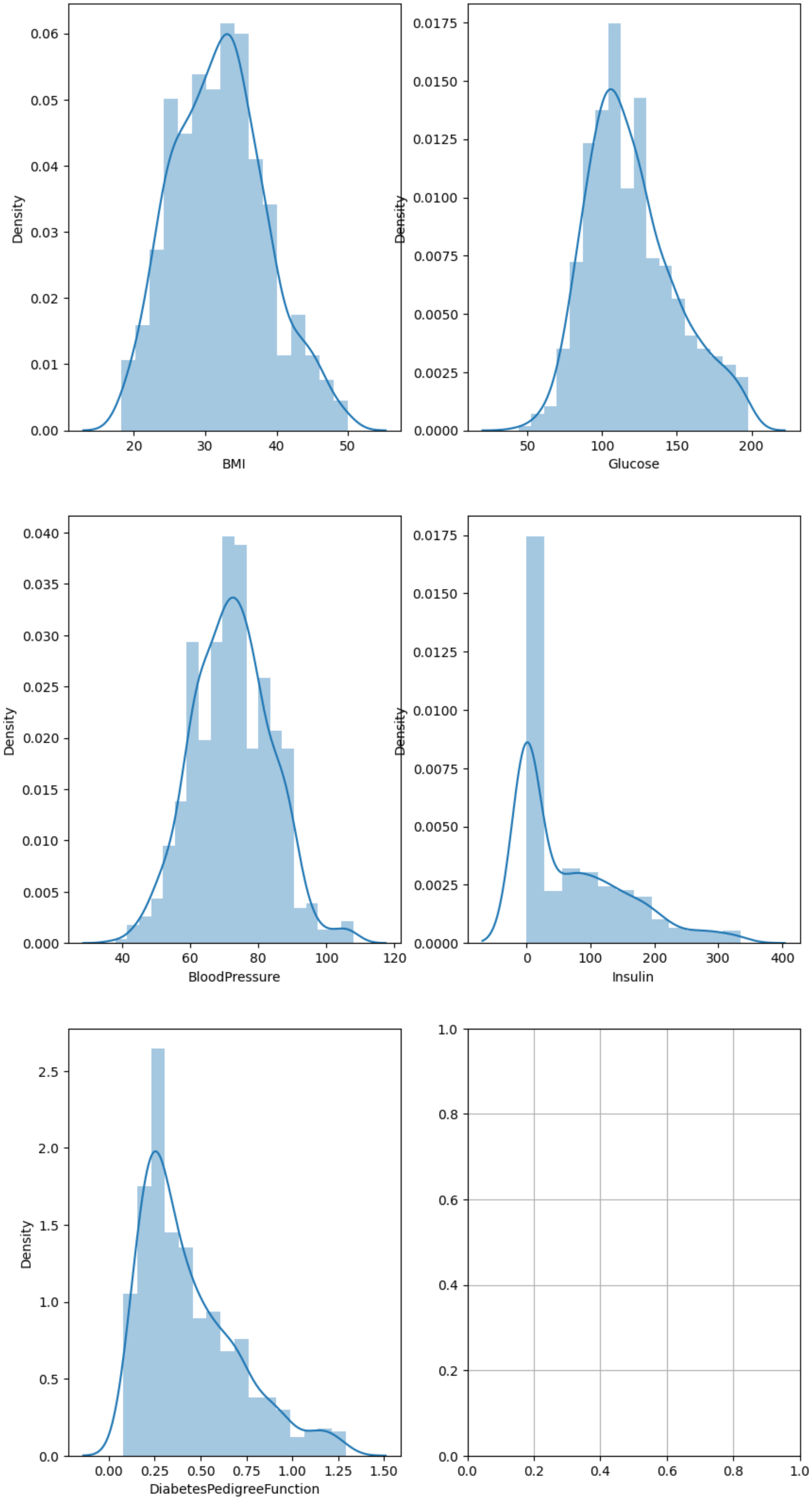
```
33
```

```
df_clean=diabetes_df.loc[test_outs==False]
df_clean.describe()
diabetes_df=df_clean
```

```
f,axes=plt.subplots(3,2,figsize=(10,20))
sns.distplot(diabetes_df.BMI,ax=axes[0][0])
plt.grid()
sns.distplot(diabetes_df.Glucose,ax=axes[0][1])
plt.grid()
sns.distplot(diabetes_df.BloodPressure,ax=axes[1][0])
plt.grid()
```

```
sns.distplot(diabetes_df.Insulin,ax=axes[1][1])
plt.grid()
sns.distplot(diabetes_df.DiabetesPedigreeFunction,ax=axes[2][0])
plt.grid()
```

```
<ipython-input-529-52febba1a34b>: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 `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(diabetes_df.BMI,ax=axes[0][0])
<ipython-input-529-52febba1a34b>:4: 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(diabetes_df.Glucose,ax=axes[0][1])
<ipython-input-529-52febba1a34b>:6: 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(diabetes_df.BloodPressure,ax=axes[1][0])
<ipython-input-529-52febba1a34b>:8: 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(diabetes_df.Insulin,ax=axes[1][1])
<ipython-input-529-52febba1a34b>:10: 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(diabetes_df.DiabetesPedigreeFunction,ax=axes[2][0])
```

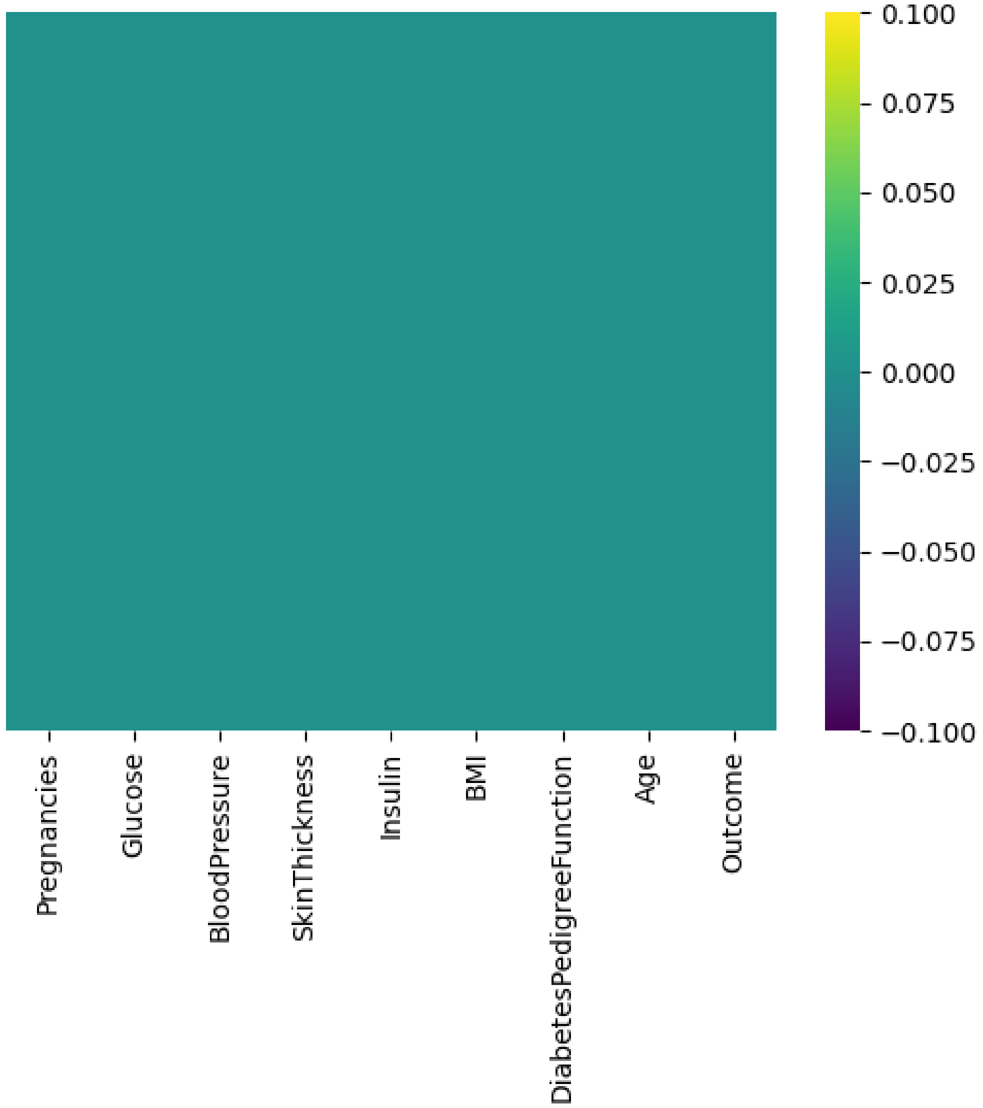

Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
20
30
40
50
BMI
Density
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
50
100
150
200
Glucose
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
40
60
80
100
120
BloodPressure
Density
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0
100
200
300
400
Insulin
Density
0.0
0.5
1.0
1.5
2.0
2.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
DiabetesPedigreeFunction
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

```
diabetes_df.isnull().sum()
```

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

```
sns.heatmap(diabetes_df.isnull(),yticklabels=False,cbar=True,cmap="viridis
")
```

```
<Axes: >
```



```
diabetes_df.shape
```

```
(663, 9)
```

```
diabetes_df.corr()
```

```
                          Pregnancies   Glucose  BloodPressure
SkinThickness  \
Pregnancies                  1.000000  0.154614       0.215797      -
0.092597
Glucose                      0.154614  1.000000       0.237370
0.012857
BloodPressure                0.215797  0.237370       1.000000
0.042274
SkinThickness               -0.092597  0.012857       0.042274
1.000000
Insulin                     -0.094486  0.255382      -0.034030
0.462058
BMI                          0.040603  0.196233       0.307080
0.378410
DiabetesPedigreeFunction     0.000952  0.066116       0.033880
0.150030
Age                          0.528851  0.241782       0.333496      -
0.142780
Outcome                      0.241065  0.490638       0.190532
0.045772

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.094486  0.040603                  0.000952
Glucose                   0.255382  0.196233                  0.066116
BloodPressure            -0.034030  0.307080                  0.033880
SkinThickness             0.462058  0.378410                  0.150030
Insulin                   1.000000  0.185041                  0.195903
BMI                       0.185041  1.000000                  0.131511
DiabetesPedigreeFunction  0.195903  0.131511                  1.000000
Age                      -0.065586  0.061163                  0.041320
Outcome                   0.104582  0.281973                  0.201347

                               Age   Outcome
Pregnancies               0.528851  0.241065
Glucose                   0.241782  0.490638
BloodPressure             0.333496  0.190532
SkinThickness            -0.142780  0.045772
Insulin                  -0.065586  0.104582
BMI                       0.061163  0.281973
DiabetesPedigreeFunction  0.041320  0.201347
Age                       1.000000  0.206625
Outcome                   0.206625  1.000000
```

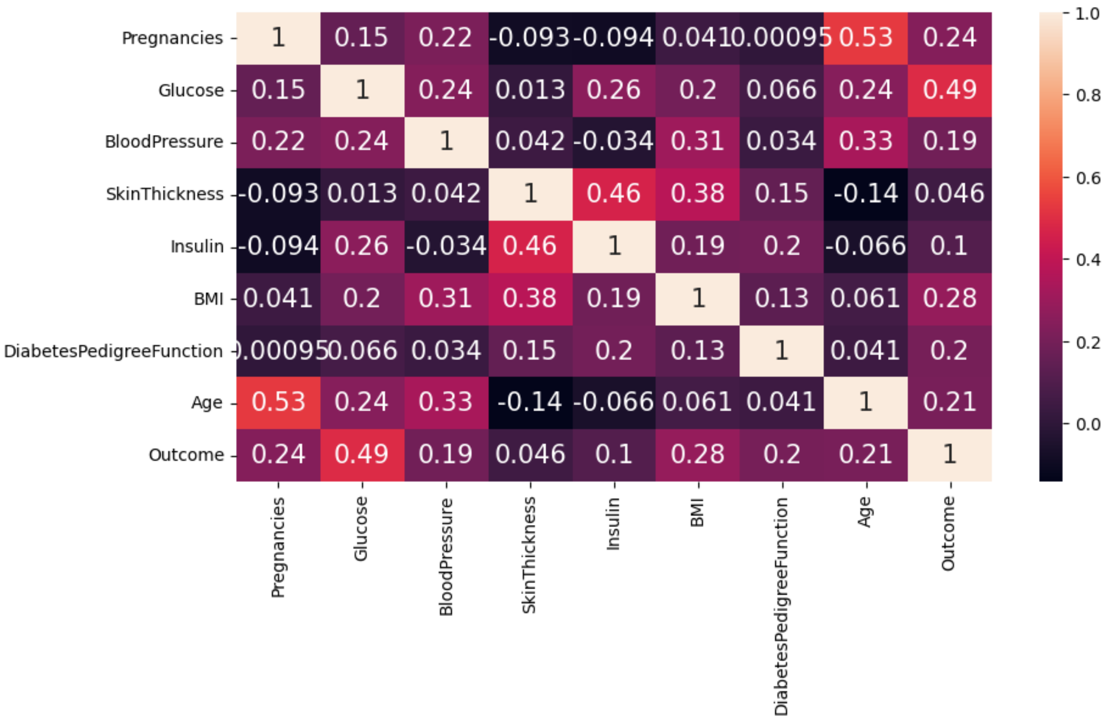
```
diabetes_df.corrwith(diabetes_df.Outcome)
```

```
Pregnancies                 0.241065
Glucose                     0.490638
BloodPressure               0.190532
SkinThickness               0.045772
Insulin                     0.104582
BMI                         0.281973
DiabetesPedigreeFunction    0.201347
```

```
Age                         0.206625
Outcome                     1.000000
dtype: float64
```

```
plt.figure(figsize = (10, 5))
sns.heatmap(diabetes_df.corr(), annot = True, annot_kws = {"size":15})
```

```
<Axes: >
```



```
#statistical measures of the data
diabetes_df.describe()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   663.000000  663.000000     663.000000     663.000000  663.000000
mean      3.852187  119.583710      72.286576      21.135747   68.761689
std       3.300992   29.656877      11.497901      14.992535   84.273332
min       0.000000   44.000000      38.000000       0.000000    0.000000
25%       1.000000   99.000000      64.000000       0.000000    0.000000
50%       3.000000  114.000000      72.000000      23.000000   40.000000
75%       6.000000  137.500000      80.000000      32.000000  122.000000
max      14.000000  198.000000     108.000000      60.000000  335.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  663.000000                663.000000  663.000000  663.000000
mean    32.071493                  0.441163   32.523379    0.324284
std      6.452334                  0.266151   10.947098    0.468460
min     18.200000                  0.078000   21.000000    0.000000
25%     27.350000                  0.244000   24.000000    0.000000
50%     32.000000                  0.365000   29.000000    0.000000
75%     36.100000                  0.592000   38.000000    1.000000
max     50.000000                  1.292000   67.000000    1.000000
```

```
diabetes_df.Outcome.value_counts() #diabetes_df['Outcome'].value_counts()
```

```
0    448
1    215
Name: Outcome, dtype: int64
```

0---->Non Diabetic 1----->Diabetic

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

```
         Pregnancies     Glucose  BloodPressure  SkinThickness    Insulin  
\
Outcome                                                                     
0           3.301339  109.511161      70.770089      20.660714  62.660714   
1           5.000000  140.572093      75.446512      22.125581  81.474419   

               BMI  DiabetesPedigreeFunction        Age  
Outcome                                                  
0        30.812054                  0.404067  30.957589  
1        34.695814                  0.518460  35.786047  
```

```
#Separating data and labels
X=diabetes_df.drop('Outcome',axis=1)
```

```
Y=diabetes_df['Outcome']
```

```
X.head(5)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72           35.0        0  33.6   
1            1       85             66           29.0        0  26.6   
2            8      183             64           23.0        0  23.3   
3            1       89             66           23.0       94  28.1   
5            5      116             74            0.0        0  25.6   

   DiabetesPedigreeFunction   Age  
0                     0.627  50.0  
1                     0.351  31.0  
2                     0.672  29.0  
3                     0.167  21.0  
5                     0.201  30.0  
```

```
Y
```

```
0      1
1      0
2      1
3      0
5      0
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 663, dtype: int64
```

Data Standardization

```
scaler=StandardScaler()

scaler.fit(X)

StandardScaler()

standardized_data=scaler.transform(X)

standardized_data

array([[ 0.6511481 ,  0.95889205, -0.02494303, ...,  0.23707093,
         0.69876731,  1.59766692],
       [-0.86469175, -1.16700828, -0.54717137, ..., -0.84862689,
        -0.33902209, -0.13926328],
       [ 1.25748404,  2.13994779, -0.72124749, ..., -1.36045586,
         0.86797211, -0.32209803],
       ...,
       [ 0.34798013,  0.04779191, -0.02494303, ..., -0.91066676,
        -0.73759339, -0.23068066],
       [-0.86469175,  0.21651416, -1.06939972, ..., -0.30577798,
        -0.34654231,  1.32341478],
       [-0.86469175, -0.89705268, -0.19901915, ..., -0.25924808,
        -0.47438593, -0.87060231]])

X=standardized_data
Y=diabetes_df['Outcome']

print(X)
print(Y)

[[ 0.6511481   0.95889205 -0.02494303 ...  0.23707093  0.69876731
   1.59766692]
 [-0.86469175 -1.16700828 -0.54717137 ... -0.84862689 -0.33902209
  -0.13926328]
 [ 1.25748404  2.13994779 -0.72124749 ... -1.36045586  0.86797211
  -0.32209803]
 ...
 [ 0.34798013  0.04779191 -0.02494303 ... -0.91066676 -0.73759339
  -0.23068066]
 [-0.86469175  0.21651416 -1.06939972 ... -0.30577798 -0.34654231
   1.32341478]
 [-0.86469175 -0.89705268 -0.19901915 ... -0.25924808 -0.47438593
  -0.87060231]]
0      1
1      0
2      1
3      0
5      0
      ..
763    0
764    0
765    0
766    1
```

```
767    0
Name: Outcome, Length: 663, dtype: int64
```

Train Test Split

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,stratify=
Y,random_state=2)
```

```
print(X.shape,X_train.shape,X_test.shape)
```

```
(663, 8) (530, 8) (133, 8)
```

Support Vector Machine

```
#from sklearn.linear_model import LogisticRegression
classifier=svm.SVC(kernel="linear")
#classifier=LogisticRegression()
```

```
#training svmc
classifier.fit(X_train,Y_train)
```

```
SVC(kernel='linear')
```

Model Evaluation

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

Printing accuracy score

```
print(training_data_accuracy)
```

```
0.7981132075471699
```

```
#accuracy score on the test data
X_test_prediction=classifier.predict(X_test)
svm_data_accuracy=accuracy_score(X_test_prediction,Y_test)
print(svm_data_accuracy)
print(classification_report(Y_test,X_test_prediction))
print(confusion_matrix(Y_test,X_test_prediction))

#                 predicted
# actual     class-0      class-1
# class-0       TN            FP
# class-1       FN            TP
```

```
0.7669172932330827
              precision    recall  f1-score   support

           0       0.78      0.91      0.84        90
           1       0.71      0.47      0.56        43

    accuracy                           0.77       133
   macro avg       0.75      0.69      0.70       133
weighted avg       0.76      0.77      0.75       133
```

```
[[82  8]
 [23 20]]
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression()
```

```
clf.fit(X_train,Y_train)
```

```
LogisticRegression()
```

```
log_train_prediction=clf.predict(X_train)
print(accuracy_score(Y_train,log_train_prediction))
```

```
0.7981132075471699
```

```
log_test_prediction=clf.predict(X_test)
log_data_accuracy=accuracy_score(Y_test,log_test_prediction)
print(log_data_accuracy)
```

```
0.7593984962406015
```

```
print(classification_report(Y_test,log_test_prediction))
```

```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83        90
           1       0.68      0.49      0.57        43

    accuracy                           0.76       133
   macro avg       0.73      0.69      0.70       133
weighted avg       0.75      0.76      0.75       133
```

```
print(confusion_matrix(Y_test,log_test_prediction))
```

```
[[80 10]
 [22 21]]
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rclf=RandomForestClassifier()
```

```
rclf.fit(X_train,Y_train)
```

```
RandomForestClassifier()
```

```
train_predictions=rclf.predict(X_train)
print(accuracy_score(Y_train,train_predictions))
```

```
1.0
```

```
test_predictions=rclf.predict(X_test)
random_data_accuracy=accuracy_score(Y_test,test_predictions)
print(random_data_accuracy)
```

```
0.7669172932330827
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
```

```
nb.fit(X_train,Y_train)
```

```
GaussianNB()
```

```
x_train_pred=nb.predict(X_train)
print(accuracy_score(Y_train,x_train_pred))
```

```
0.7849056603773585
```

```
x_test_pred=nb.predict(X_test)
bayes_data_accuracy=accuracy_score(Y_test,x_test_pred)
print(bayes_data_accuracy)
```

```
0.6992481203007519
```

```
print(classification_report(Y_test,x_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.80      0.78        90
           1       0.54      0.49      0.51        43

    accuracy                           0.70       133
   macro avg       0.65      0.64      0.65       133
weighted avg       0.69      0.70      0.70       133
```

```
print(confusion_matrix(Y_test,x_test_pred))
```

```
[[72 18]
 [22 21]]
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
kn=KNeighborsClassifier(n_neighbors=5,metric="euclidean",n_jobs=-1)
```

```
kn.fit(X_train,Y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_jobs=-1)
```

```
xtrain_predict=kn.predict(X_train)
print(accuracy_score(Y_train,xtrain_predict))
```

```
0.8226415094339623
```

```
xtest_predict=kn.predict(X_test)
knn_data_accuracy=accuracy_score(Y_test,xtest_predict)
print(knn_data_accuracy)
```

```
0.7819548872180451
```

```
print(classification_report(Y_test,xtest_predict))
```

```
              precision    recall  f1-score   support

           0       0.78      0.94      0.85        90
           1       0.79      0.44      0.57        43

    accuracy                           0.78       133
   macro avg       0.79      0.69      0.71       133
weighted avg       0.78      0.78      0.76       133
```

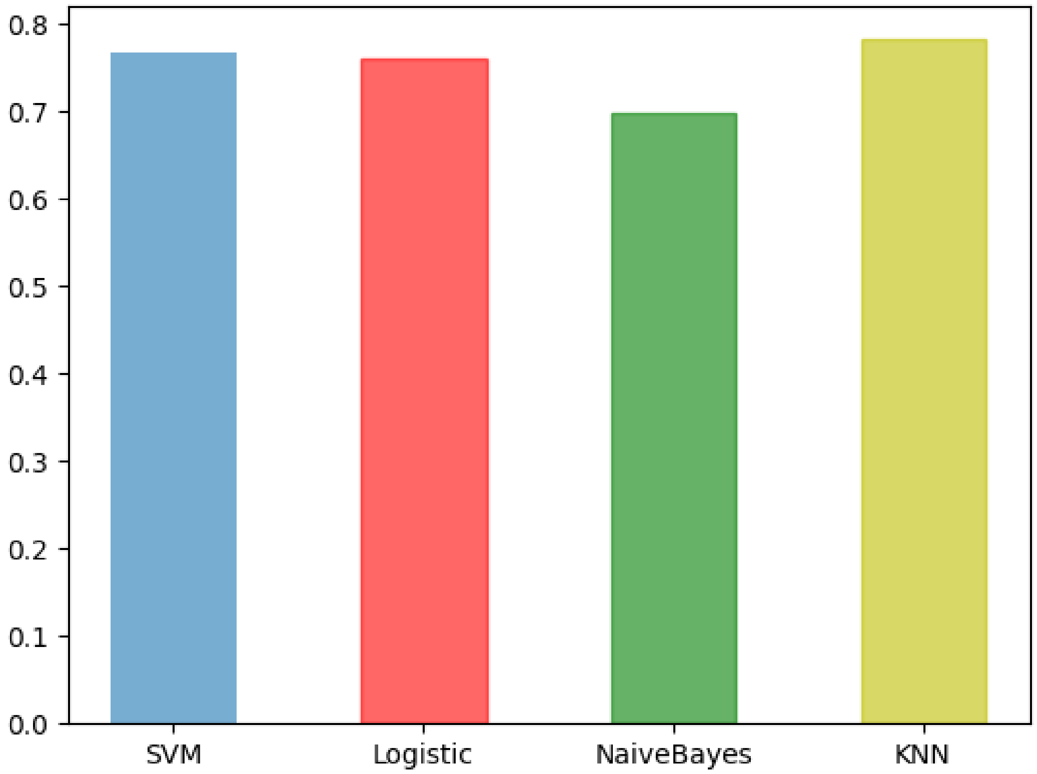
```
print(confusion_matrix(Y_test,xtest_predict))
```

```
[[85  5]
 [24 19]]
```

Comparing accuracy scores of 5 models

```
models=["SVM","Logistic","NaiveBayes","KNN"]
accuracies=[svm_data_accuracy,log_data_accuracy,bayes_data_accuracy,knn_da
ta_accuracy]
barlist=plt.bar(models,accuracies,width=0.5,alpha=0.6)

barlist[1].set_color('r')
barlist[2].set_color('g')
barlist[3].set_color('y')
print(svm_data_accuracy,log_data_accuracy,bayes_data_accuracy,knn_data_acc
uracy)
```

```
0.7669172932330827 0.7593984962406015 0.6992481203007519
0.7819548872180451
```



Making a Predictive System

```
input_data=(4,110,92,0,37,6,0.191,30)
#changing input to numpy array
input_data_numpy_arr=np.asarray(input_data)

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

#standardize input data
std_data=scaler.transform(input_data_reshaped)
print(std_data)

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

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

```
[[ 0.04481216 -0.32339703  1.71581811 -1.41081572 -0.37717352 -4.04368046
  -0.94063914 -0.23068066]]
[0]
The person is not diabetic
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
X does not have valid feature names, but StandardScaler was fitted with
feature names
  warnings.warn(
import pickle

filename='diabetes_model.sav'
pickle.dump(classifier,open(filename,'wb'))
```

Heart Disease Prediction Model:

Importing Dependencies

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

Data Collection and Processing

```
#loading csv data to pandas dataframe
df=pd.read_csv('/content/heart.csv')
heart_df=df.copy()
```

```
heart_df.head(5)
```

```
    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0    1   3       145   233    1        0    150.0      0      2.3   
1  37.0    1   2       130   250    0        1    187.0      0      3.5   
2  41.0    0   1       130   204    0        0    172.0      0      1.4   
3  56.0    1   1       120   236    0        1    178.0      0      0.8   
4  57.0    0   0       120   354    0        1    163.0      1      0.6   

   slope  ca  thal  target  
0      0   0     1       1  
1      0   0     2       1  
2      2   0     2       1  
3      2   0     2       1  
4      2   0     2       1  
```

```
heart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       276 non-null    float64
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   280 non-null    float64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(3), int64(11)
memory usage: 33.3 KB
```

```
#printing last 5 rows of dataset
heart_df.tail(5)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
298  57.0    0   0       140   241    0        1      NaN      1      0.2
299  45.0    1   3       110   264    0        1    132.0      0      1.2
300  68.0    1   0       144   193    1        1    141.0      0      3.4
301  57.0    1   0       130   131    0        1    115.0      1      1.2
302  57.0    0   1       130   236    0        0    174.0      0      0.0

     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

```
#number of rows and columns in dataset
heart_df.shape
```

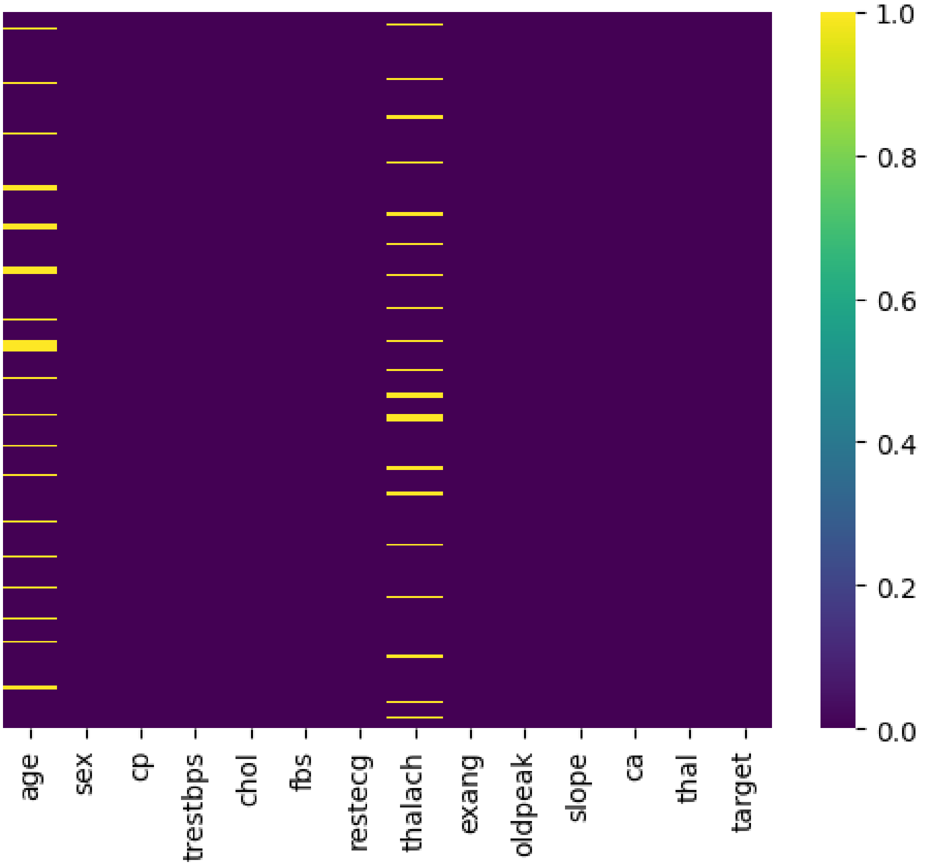
```
(303, 14)
```

```
heart_df.isnull().sum()
```

```
age         27
sex          0
cp           0
trestbps     0
chol         0
fbs          0
restecg      0
thalach     23
exang        0
oldpeak      0
slope        0
ca           0
thal         0
target       0
dtype: int64
```

```
#heatmap
import seaborn as sns
sns.heatmap(heart_df.isnull(),yticklabels=False,cbar=True,cmap="viridis")
```

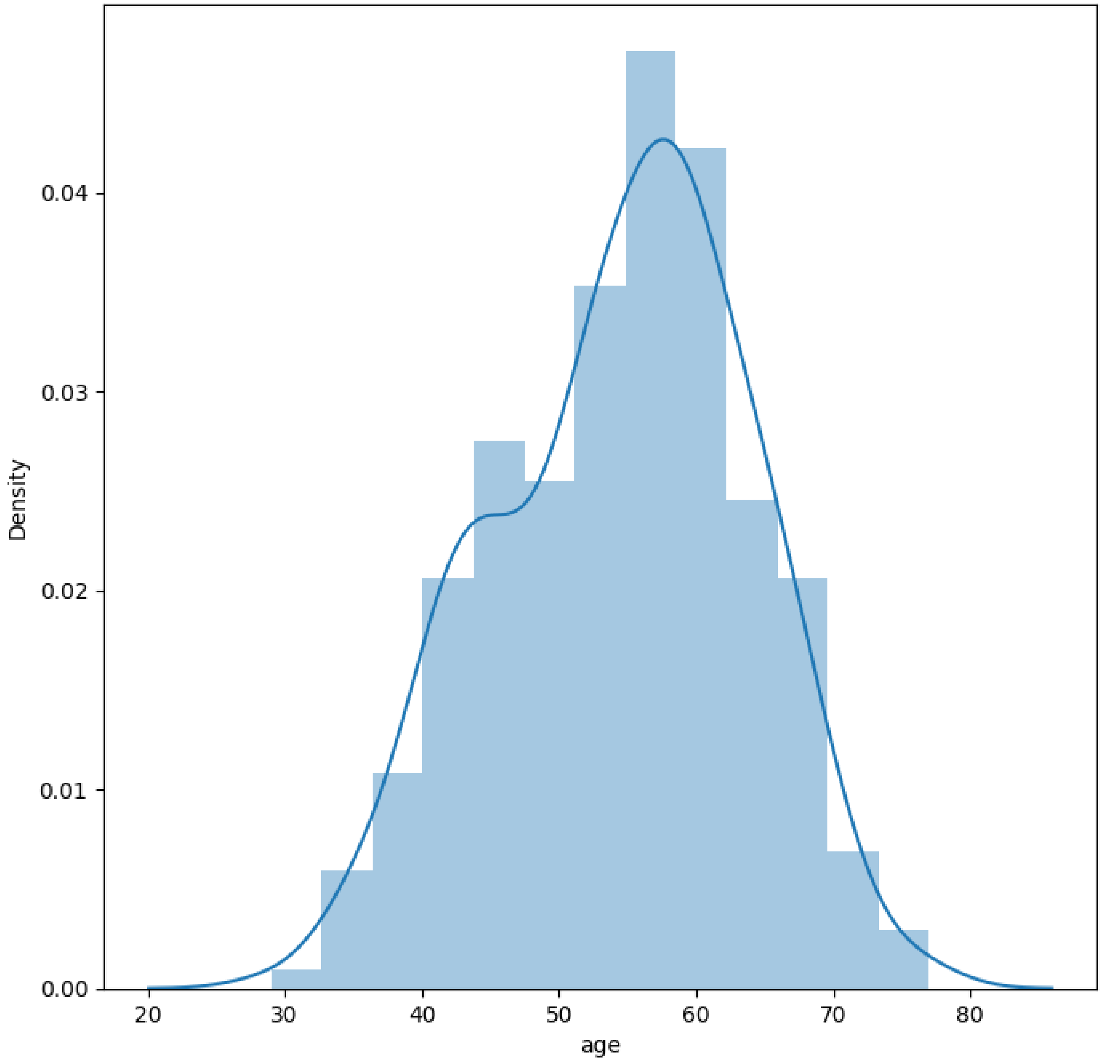
```
<Axes: >
```



```
#distribution of data in age
fig,ax=plt.subplots(figsize=(8,8))
sns.distplot(heart_df.age)
```

```
<ipython-input-73-423799895cf4>:3: 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(heart_df.age)
```

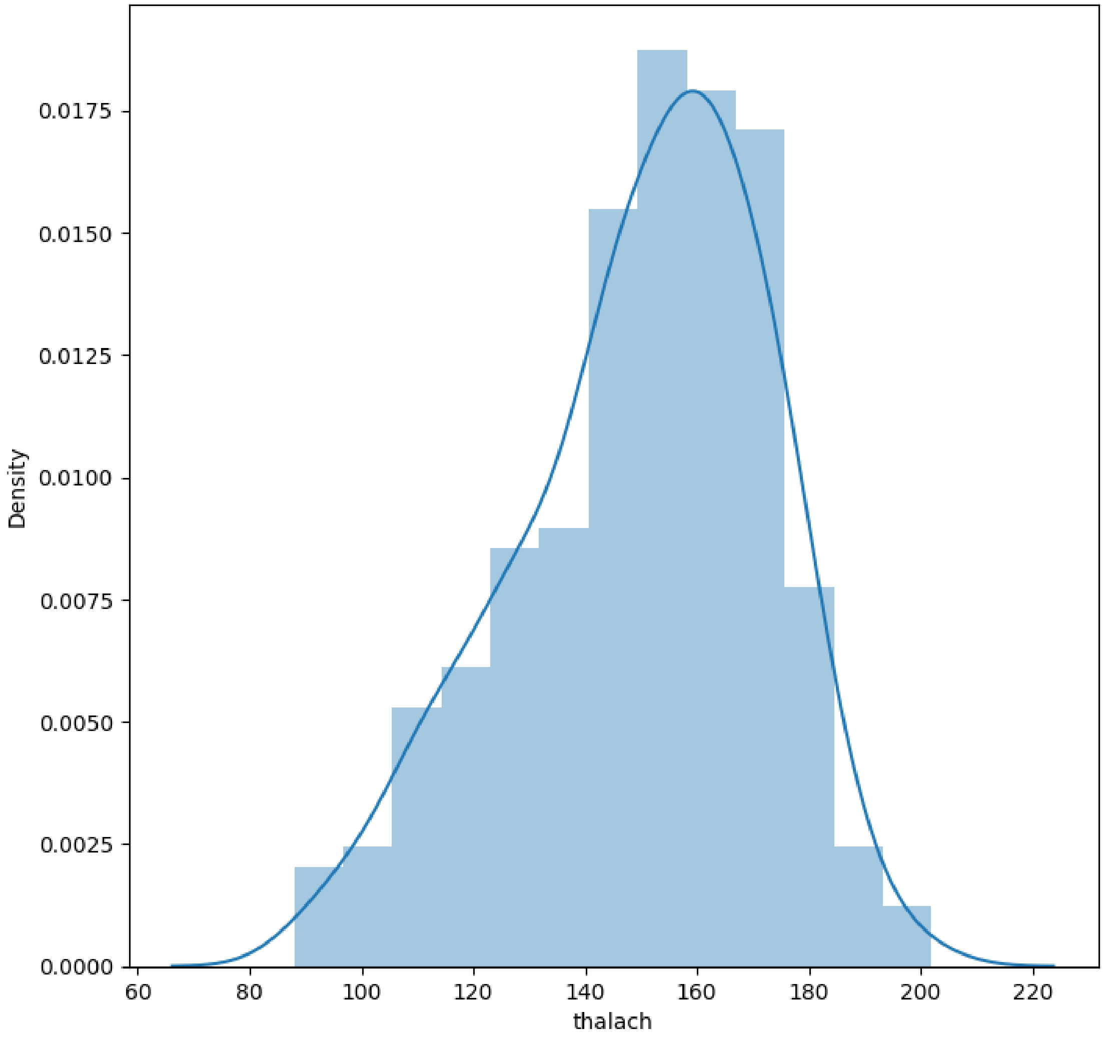
```
<Axes: xlabel='age', ylabel='Density'>
```



```
#distribution of data in thalach
fig,ax=plt.subplots(figsize=(8,8))
sns.distplot(heart_df.thalach)
```

```
<ipython-input-74-f84436d2239f>:3: 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(heart_df.thalach)
```

```
<Axes: xlabel='thalach', ylabel='Density'>
```



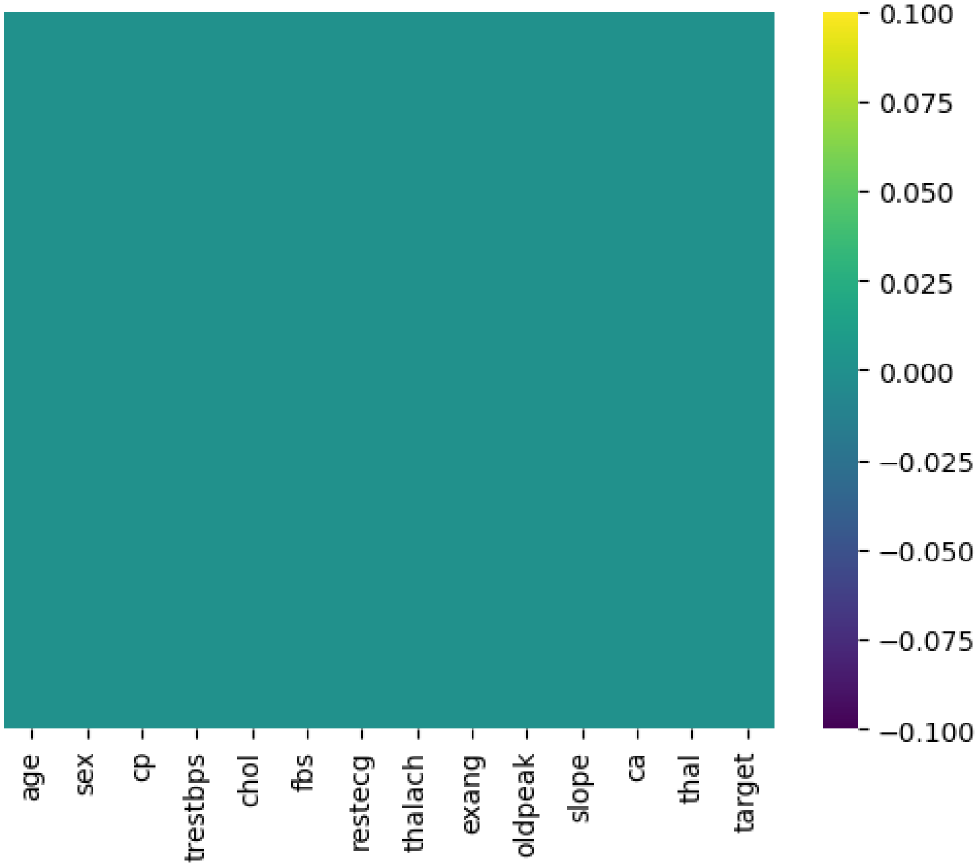
```
#replace the missing values with median values
heart_df['age'].fillna(heart_df['age'].median(),inplace=True)
heart_df['thalach'].fillna(heart_df['thalach'].median(),inplace=True)
```

```
heart_df.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
sns.heatmap(heart_df.isnull(),yticklabels=False,cbar=True,cmap="viridis")
```

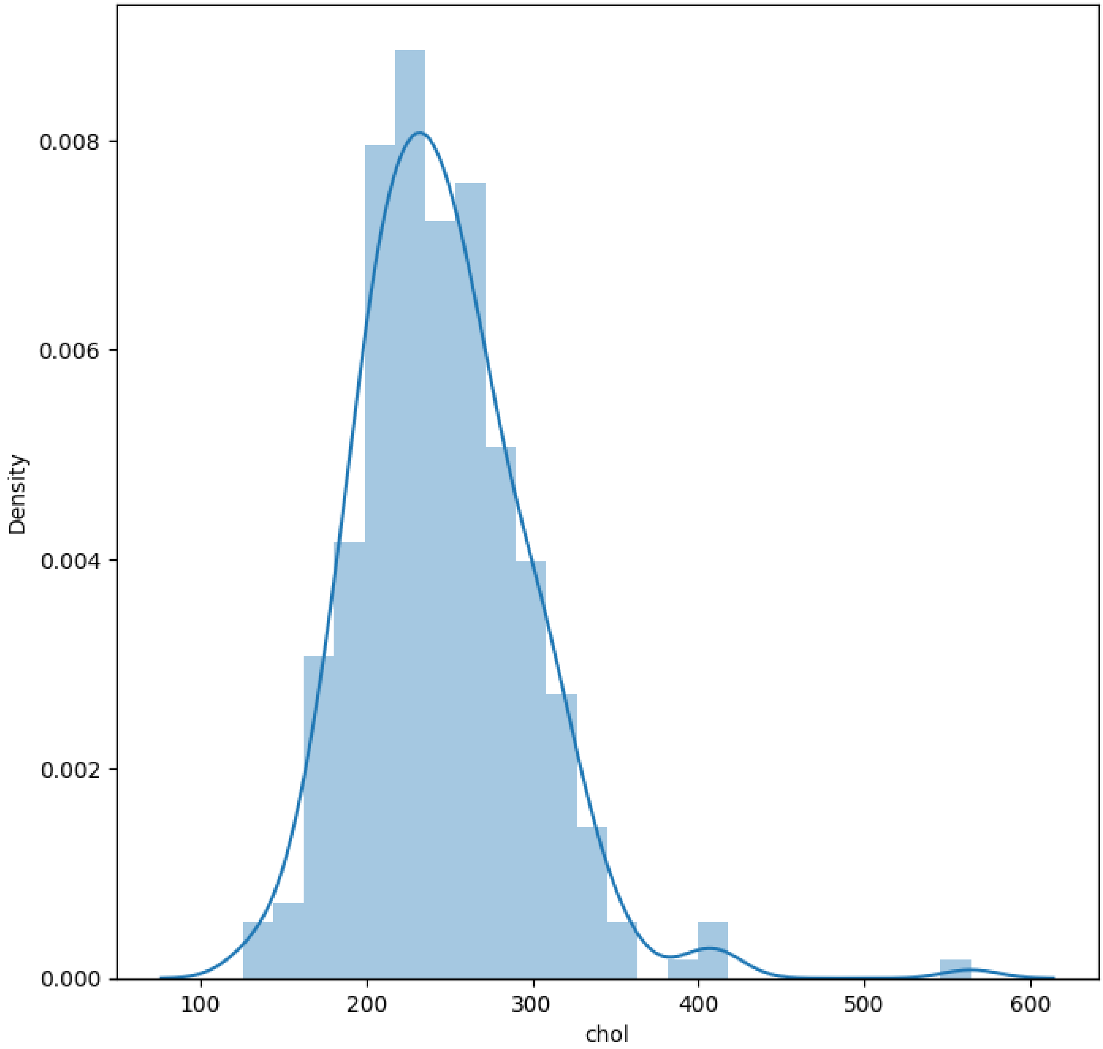
```
<Axes: >
```



```
f,axes=plt.subplots(figsize=(8,8))
sns.distplot(heart_df.chol)
```

```
<ipython-input-78-051f31b8fda1>: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 `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(heart_df.chol)
```

```
<Axes: xlabel='chol', ylabel='Density'>
```



```
def find_outliers(col):
  from scipy import stats
  z=np.abs(stats.zscore(col))
  idx_outliers=np.where(z>3,True,False)
  return pd.Series(idx_outliers,index=col.index)
idx=find_outliers(heart_df.chol)

idx

0      False
1      False
2      False
3      False
4      False
       ...
298    False
299    False
300    False
301    False
302    False
Length: 303, dtype: bool
```

```
display(heart_df.loc[idx==True].describe().round(3))
```

```
          age  sex     cp  trestbps     chol   fbs  restecg  thalach
exang  \
count    4.000  4.0  4.000     4.000    4.000  4.00      4.0    4.000
4.00
mean    62.750  0.0  1.000   134.750  449.250  0.25      0.0  152.500
0.25
std      4.787  0.0  1.155    14.728   76.622  0.50      0.0    1.732
0.50
min     56.000  0.0  0.000   115.000  407.000  0.00      0.0  150.000
0.00
25%     61.250  0.0  0.000   129.250  408.500  0.00      0.0  152.250
0.00
50%     64.000  0.0  1.000   137.000  413.000  0.00      0.0  153.000
0.00
75%     65.500  0.0  2.000   142.500  453.750  0.25      0.0  153.250
0.25
max     67.000  0.0  2.000   150.000  564.000  1.00      0.0  154.000
1.00

       oldpeak  slope     ca  thal  target
count    4.000   4.00  4.000  4.00   4.000
mean     2.075   1.25  1.500  2.75   0.500
std      1.365   0.50  1.291  0.50   0.577
min      0.800   1.00  0.000  2.00   0.000
25%      1.400   1.00  0.750  2.75   0.000
50%      1.750   1.00  1.500  3.00   0.500
75%      2.425   1.25  2.250  3.00   1.000
max      4.000   2.00  3.000  3.00   1.000
```

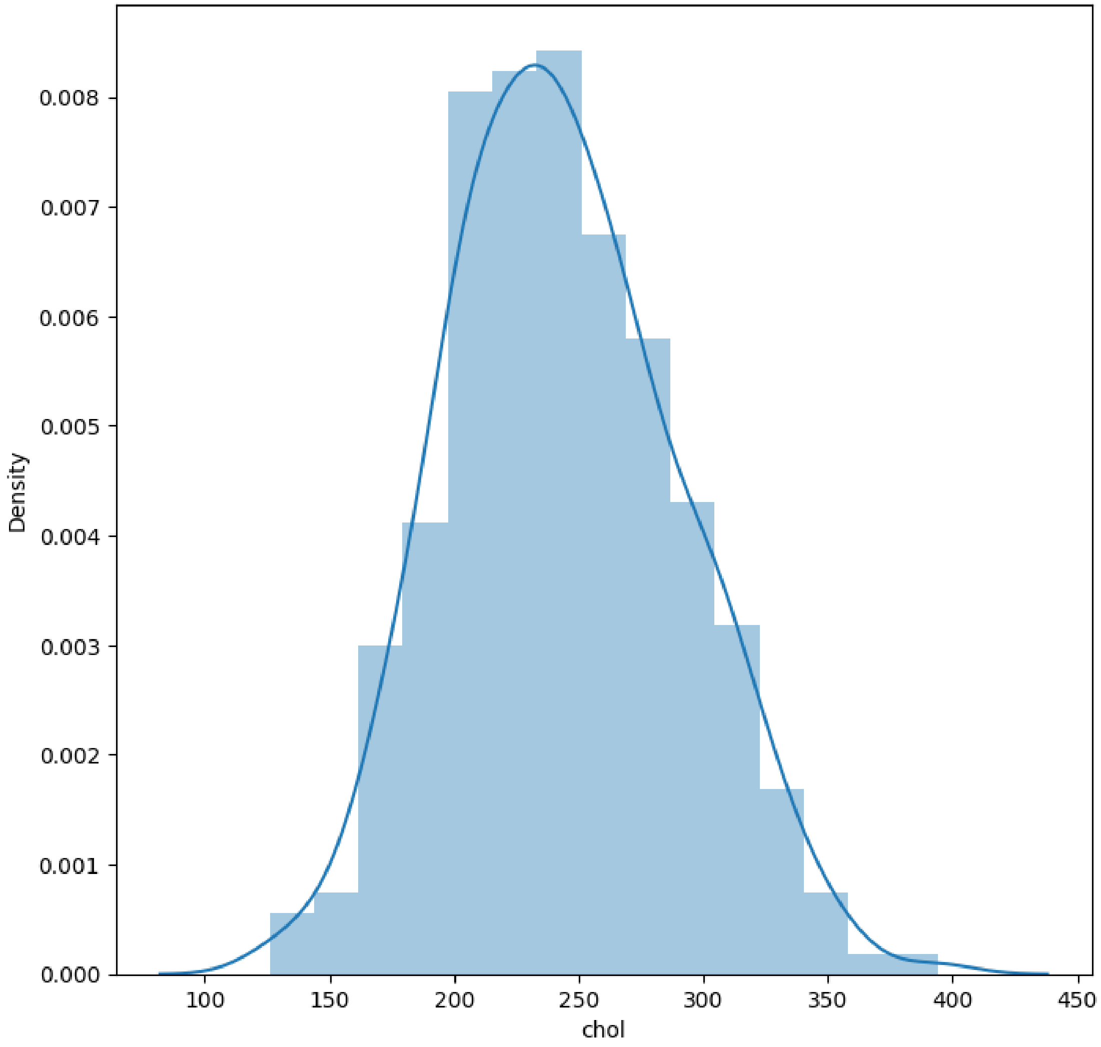
```
display(heart_df.loc[idx==False].describe().round(3))
heart_df=heart_df.loc[idx==False]
```

```
           age      sex       cp  trestbps     chol      fbs  restecg  \
count  299.000  299.000  299.000   299.000  299.000  299.000  299.000
mean    54.602    0.692    0.967   131.582  243.548    0.147    0.535
std      8.743    0.462    1.032    17.590   45.858    0.355    0.526
min     29.000    0.000    0.000    94.000  126.000    0.000    0.000
25%     49.000    0.000    0.000   120.000  211.000    0.000    0.000
50%     56.000    1.000    1.000   130.000  240.000    0.000    1.000
75%     60.000    1.000    2.000   140.000  273.500    0.000    1.000
max     77.000    1.000    3.000   200.000  394.000    1.000    2.000

       thalach    exang  oldpeak    slope       ca     thal   target
count  299.000  299.000  299.000  299.000  299.000  299.000  299.000
mean   150.214    0.328    1.026    1.401    0.719    2.308    0.545
std     21.695    0.470    1.154    0.618    1.017    0.612    0.499
min     88.000    0.000    0.000    0.000    0.000    0.000    0.000
25%    138.500    0.000    0.000    1.000    0.000    2.000    0.000
50%    153.000    0.000    0.700    1.000    0.000    2.000    1.000
75%    165.000    1.000    1.600    2.000    1.000    3.000    1.000
max    202.000    1.000    6.200    2.000    4.000    3.000    1.000
```

```
f,axes=plt.subplots(figsize=(8,8))
sns.distplot(heart_df.chol)
```

```
<ipython-input-82-051f31b8fda1>: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 `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(heart_df.chol)
```

```
<Axes: xlabel='chol', ylabel='Density'>
```



```
heart_df.corrwith(heart_df.target)
```

```
age        -0.224007
sex        -0.288805
cp          0.426488
trestbps   -0.140696
chol       -0.104369
fbs        -0.037664
restecg     0.137884
thalach     0.405654
exang      -0.435335
oldpeak    -0.427715
slope       0.343767
ca         -0.384939
thal       -0.342352
target      1.000000
dtype: float64
```

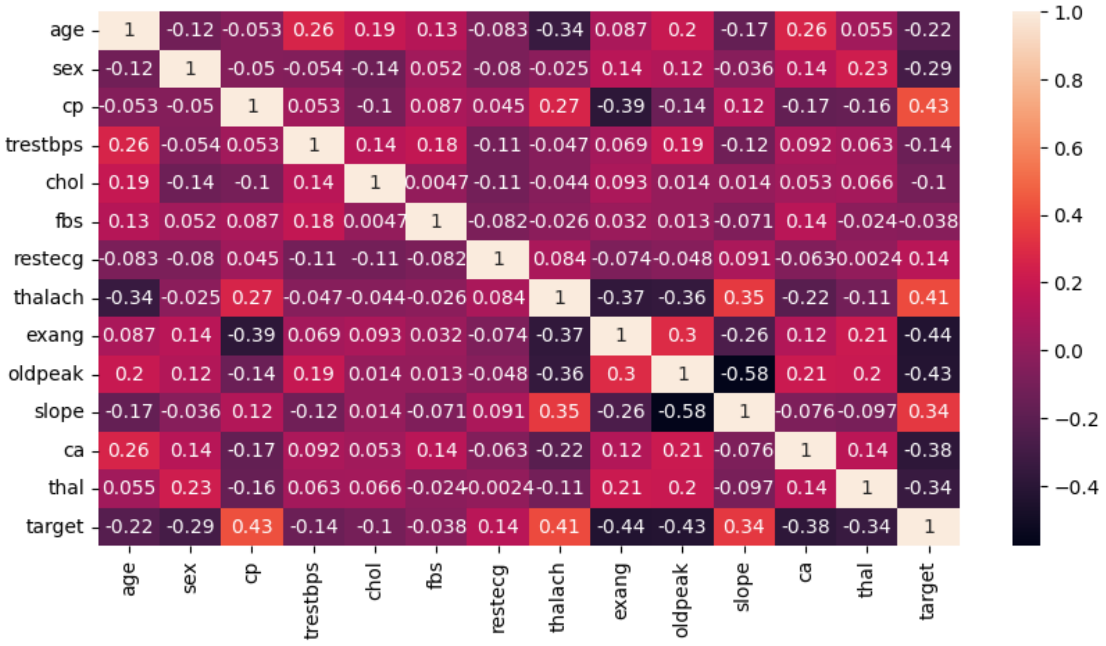
```
heart_df.corr()
```

```
               age       sex        cp  trestbps      chol       fbs  \
age       1.000000 -0.120055 -0.053149  0.264521  0.191458  0.128176   
sex      -0.120055  1.000000 -0.049753 -0.054249 -0.139533  0.051924   
cp       -0.053149 -0.049753  1.000000  0.053367 -0.101672  0.086751   
trestbps  0.264521 -0.054249  0.053367  1.000000  0.142659  0.176550   
chol      0.191458 -0.139533 -0.101672  0.142659  1.000000  0.004715   
fbs       0.128176  0.051924  0.086751  0.176550  0.004715  1.000000   
restecg  -0.082719 -0.079645  0.045442 -0.112884 -0.112560 -0.081747   
thalach  -0.338080 -0.024527  0.265942 -0.047277 -0.043502 -0.025900   
exang     0.086527  0.141314 -0.392098  0.068965  0.092797  0.031749   
oldpeak   0.195627  0.116748 -0.142571  0.188864  0.013709  0.012834   
slope    -0.169671 -0.036135  0.115759 -0.123414  0.013755 -0.071281   
ca        0.264302  0.136664 -0.171915  0.091869  0.052803  0.142793   
thal      0.054926  0.228916 -0.158858  0.062779  0.066402 -0.023764   
target   -0.224007 -0.288805  0.426488 -0.140696 -0.104369 -0.037664   

           restecg   thalach     exang   oldpeak     slope        ca  \
age      -0.082719 -0.338080  0.086527  0.195627 -0.169671  0.264302   
sex      -0.079645 -0.024527  0.141314  0.116748 -0.036135  0.136664   
```

```
cp         0.045442  0.265942 -0.392098 -0.142571  0.115759 -0.171915
trestbps  -0.112884 -0.047277  0.068965  0.188864 -0.123414  0.091869
chol      -0.112560 -0.043502  0.092797  0.013709  0.013755  0.052803
fbs       -0.081747 -0.025900  0.031749  0.012834 -0.071281  0.142793
restecg    1.000000  0.084065 -0.073863 -0.047656  0.090725 -0.063052
thalach    0.084065  1.000000 -0.369432 -0.361457  0.352446 -0.216063
exang     -0.073863 -0.369432  1.000000  0.296586 -0.257863  0.115981
oldpeak   -0.047656 -0.361457  0.296586  1.000000 -0.577924  0.207042
slope      0.090725  0.352446 -0.257863 -0.577924  1.000000 -0.076253
ca        -0.063052 -0.216063  0.115981  0.207042 -0.076253  1.000000
thal      -0.002406 -0.109574  0.208045  0.199552 -0.096869  0.144648
target     0.137884  0.405654 -0.435335 -0.427715  0.343767 -0.384939

              thal    target
age        0.054926 -0.224007
sex        0.228916 -0.288805
cp        -0.158858  0.426488
trestbps   0.062779 -0.140696
chol       0.066402 -0.104369
fbs       -0.023764 -0.037664
restecg   -0.002406  0.137884
thalach   -0.109574  0.405654
exang      0.208045 -0.435335
oldpeak    0.199552 -0.427715
slope     -0.096869  0.343767
ca         0.144648 -0.384939
thal       1.000000 -0.342352
target    -0.342352  1.000000
```

```python
plt.figure(figsize = (10, 5))
sns.heatmap(heart_df.corr(), annot = True, annot_kws = {"size":10})
```

```
<Axes: >
```



```
#statistical measure about the data
heart_df.describe()

              age         sex          cp    trestbps        chol  
fbs  \
count  299.000000  299.000000  299.000000  299.000000  299.000000  
299.000000
mean    54.602007    0.692308    0.966555  131.581940  243.548495  
0.147157
std      8.743481    0.462312    1.032469   17.589726   45.857602  
0.354856
min     29.000000    0.000000    0.000000   94.000000  126.000000  
0.000000
25%     49.000000    0.000000    0.000000  120.000000  211.000000  
0.000000
50%     56.000000    1.000000    1.000000  130.000000  240.000000  
0.000000
75%     60.000000    1.000000    2.000000  140.000000  273.500000  
0.000000
max     77.000000    1.000000    3.000000  200.000000  394.000000  
1.000000

          restecg     thalach       exang     oldpeak       slope  
ca  \
count  299.000000  299.000000  299.000000  299.000000  299.000000  
299.000000
mean     0.535117  150.214047    0.327759    1.025753    1.401338  
0.719064
std      0.525782   21.694671    0.470183    1.154495    0.618071  
1.017302
min      0.000000   88.000000    0.000000    0.000000    0.000000  
0.000000
25%      0.000000  138.500000    0.000000    0.000000    1.000000  
0.000000
50%      1.000000  153.000000    0.000000    0.700000    1.000000  
0.000000
75%      1.000000  165.000000    1.000000    1.600000    2.000000  
1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000  
4.000000

             thal      target  
count  299.000000  299.000000  
mean     2.307692    0.545151  
std      0.612214    0.498792  
min      0.000000    0.000000  
25%      2.000000    0.000000  
50%      2.000000    1.000000  
75%      3.000000    1.000000  
max      3.000000    1.000000  

#checking the distribution of target variable
heart_df.target.value_counts()
```

```
1    163
0    136
Name: target, dtype: int64
```

1--->Defective Heart 0--->Healthy Heart

Splitting the features and target

```
X=heart_df.drop('target',axis=1)
Y=heart_df['target']
```

```
print(X)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
\
0    63.0    1   3       145   233    1        0    150.0      0      2.3
1    37.0    1   2       130   250    0        1    187.0      0      3.5
2    41.0    0   1       130   204    0        0    172.0      0      1.4
3    56.0    1   1       120   236    0        1    178.0      0      0.8
4    57.0    0   0       120   354    0        1    163.0      1      0.6
..    ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298  57.0    0   0       140   241    0        1    153.0      1      0.2
299  45.0    1   3       110   264    0        1    132.0      0      1.2
300  68.0    1   0       144   193    1        1    141.0      0      3.4
301  57.0    1   0       130   131    0        1    115.0      1      1.2
302  57.0    0   1       130   236    0        0    174.0      0      0.0

     slope  ca  thal
0        0   0     1
1        0   0     2
2        2   0     2
3        2   0     2
4        2   0     2
..     ...  ..   ...
298      1   0     3
299      1   0     3
300      1   2     3
301      1   1     3
302      1   1     2

[299 rows x 13 columns]
```

```
print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
```

```
302    0
Name: target, Length: 299, dtype: int64
```

Splitting the data into training and test data

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_st
ate=2,stratify=Y)
```

```
print(X.shape,X_train.shape,X_test.shape)
```

```
(299, 13) (239, 13) (60, 13)
```

Model Training

```
#Logistic Regression
clf=LogisticRegression()
```

```
#training model with training data
clf.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

Model evaluation

```
#accuracy on training data
X_train_prediction=clf.predict(X_train)
```

```
training_data_accuracy=accuracy_score(Y_train,X_train_prediction)
print("Accuracy on Training data ",training_data_accuracy)
```

```
Accuracy on Training data  0.8410041841004184
```

```
#accuracy on test data
X_test_prediction=clf.predict(X_test)
log_data_accuracy=accuracy_score(Y_test,X_test_prediction)
print("Accuracy on Test data ",log_data_accuracy)
print(classification_report(Y_test,X_test_prediction))
print(confusion_matrix(Y_test,X_test_prediction))
```

```
Accuracy on Test data  0.85
              precision    recall  f1-score   support

           0       0.82      0.85      0.84        27
           1       0.88      0.85      0.86        33

    accuracy                           0.85        60
   macro avg       0.85      0.85      0.85        60
weighted avg       0.85      0.85      0.85        60
```

```
[[23  4]
 [ 5 28]]
```

SVM

```
from sklearn import svm
classifier=svm.SVC(kernel="linear")
classifier.fit(X_train,Y_train)
```

```
SVC(kernel='linear')
```

```
#accuracy score on the training data
svc_x_train_prediction=classifier.predict(X_train)
svc_training_data_accuracy=accuracy_score(svc_x_train_prediction,Y_train)
```

```
print(svc_training_data_accuracy)
```

```
0.8451882845188284
```

```
svc_x_test_prediction=classifier.predict(X_test)
svm_data_accuracy=accuracy_score(svc_x_test_prediction,Y_test)
print(svm_data_accuracy)
```

```
0.8166666666666667
```

```
print(classification_report(Y_test,svc_x_test_prediction))
print(confusion_matrix(Y_test,svc_x_test_prediction))
```

```
              precision    recall  f1-score   support

           0       0.86      0.70      0.78        27
           1       0.79      0.91      0.85        33

    accuracy                           0.82        60
   macro avg       0.83      0.81      0.81        60
weighted avg       0.82      0.82      0.81        60
```

```
[[19  8]
 [ 3 30]]
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rclf=RandomForestClassifier()
```

```
rclf.fit(X_train,Y_train)
```

```
RandomForestClassifier()
```

```
train_predictions=rclf.predict(X_train)
print(accuracy_score(Y_train,train_predictions))
```

```
1.0
```

```
test_predictions=rclf.predict(X_test)
random_data_accuracy=accuracy_score(Y_test,test_predictions)
print(random_data_accuracy)
```

```
0.85
```

```
print(classification_report(Y_test,test_predictions))
print(confusion_matrix(Y_test,test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.83      0.91      0.87        33

    accuracy                           0.85        60
   macro avg       0.85      0.84      0.85        60
weighted avg       0.85      0.85      0.85        60

[[21  6]
 [ 3 30]]
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
```

```
nb.fit(X_train,Y_train)
```

```
GaussianNB()
```

```
x_train_pred=nb.predict(X_train)
print(accuracy_score(Y_train,x_train_pred))
```

```
0.8326359832635983
```

```
x_test_pred=nb.predict(X_test)
bayes_data_accuracy=accuracy_score(Y_test,x_test_pred)
print(bayes_data_accuracy)
```

```
0.8166666666666667
```

```
print(classification_report(Y_test,x_test_pred))
print(confusion_matrix(Y_test,x_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.79        27
           1       0.82      0.85      0.84        33

    accuracy                           0.82        60
   macro avg       0.82      0.81      0.81        60
weighted avg       0.82      0.82      0.82        60
```

```
[[21  6]
 [ 5 28]]
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
kn=KNeighborsClassifier(n_neighbors=5,metric="euclidean",n_jobs=-1)
```

```
kn.fit(X_train,Y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_jobs=-1)
```

```
xtrain_predict=kn.predict(X_train)
print(accuracy_score(Y_train,xtrain_predict))
```

```
0.7531380753138075
```

```
xtest_predict=kn.predict(X_test)
knn_data_accuracy=accuracy_score(Y_test,xtest_predict)
print(knn_data_accuracy)
```

```
0.7166666666666667
```

```
print(classification_report(Y_test,xtest_predict))
```

```
              precision    recall  f1-score   support

           0       0.73      0.59      0.65        27
           1       0.71      0.82      0.76        33

    accuracy                           0.72        60
   macro avg       0.72      0.71      0.71        60
weighted avg       0.72      0.72      0.71        60

```

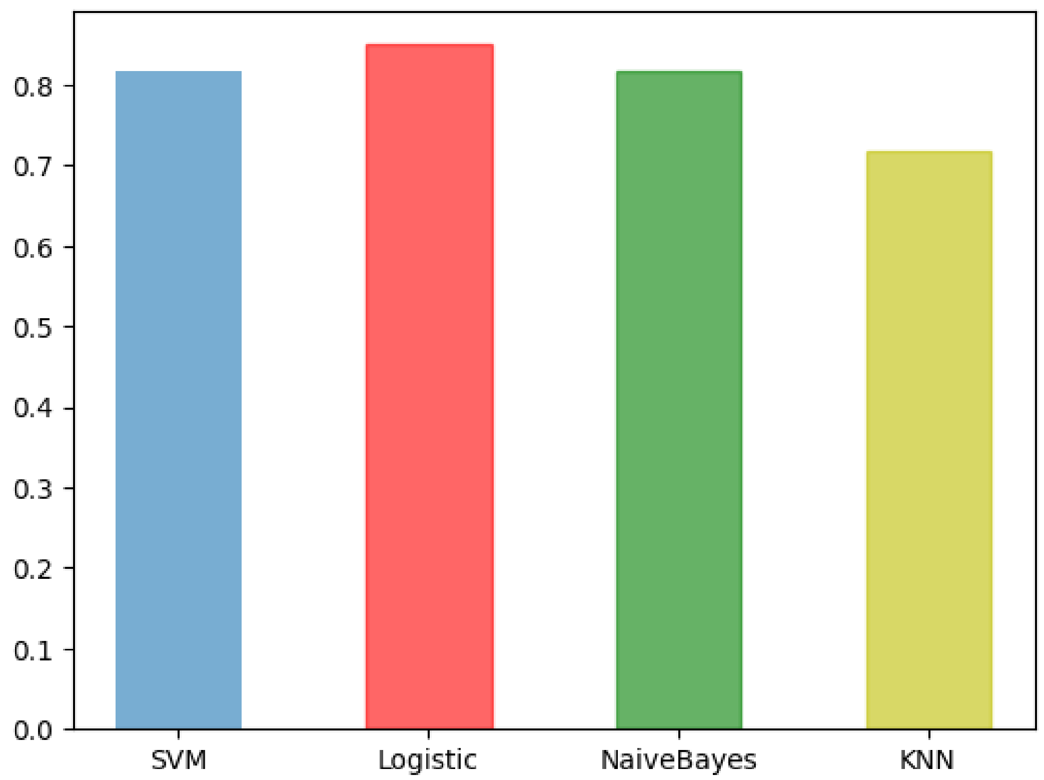
```
print(confusion_matrix(Y_test,xtest_predict))
```

```
[[16 11]
 [ 6 27]]
```

Comparing accuracies of 5 models

```
models=["SVM","Logistic","NaiveBayes","KNN"]
accuracies=[svm_data_accuracy,log_data_accuracy,bayes_data_accuracy,knn_da
ta_accuracy]
barlist=plt.bar(models,accuracies,width=0.5,alpha=0.6)

barlist[1].set_color('r')
barlist[2].set_color('g')
barlist[3].set_color('y')
print(svm_data_accuracy,log_data_accuracy,bayes_data_accuracy,knn_data_acc
uracy)
```

```
0.8166666666666667 0.85 0.8166666666666667 0.7166666666666667
```



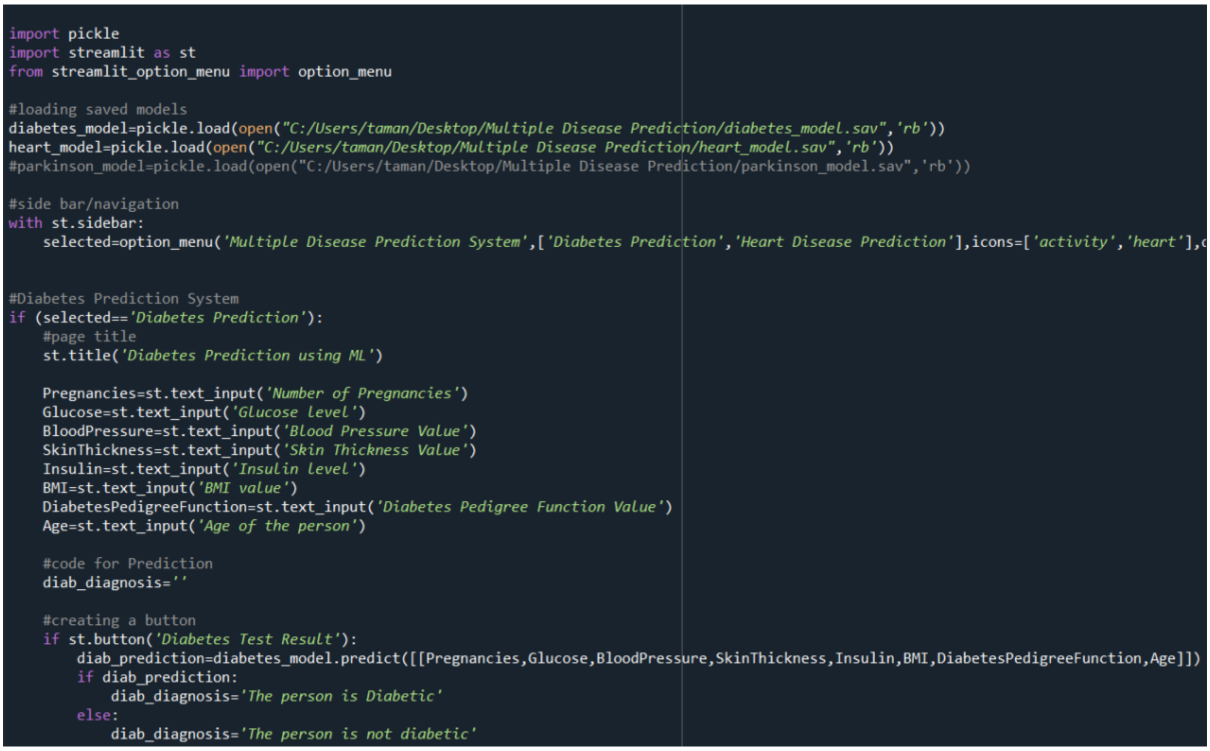
Building a predictive system

```
input_data=(62,0,0,140,268,0,0,160,0,3.6,0,2,2)
#change input data to numpy array
input_data_as_numpy_array=np.asarray(input_data)
#reshape the numpy array as we are predicting for only one instance
input_data_reshaped=input_data_as_numpy_array.reshape(1,-1)
prediction=clf.predict(input_data_reshaped)
print(prediction)
if prediction:
  print("Person has heart disease")
else:
  print("Person doesn't have heart disease")
```

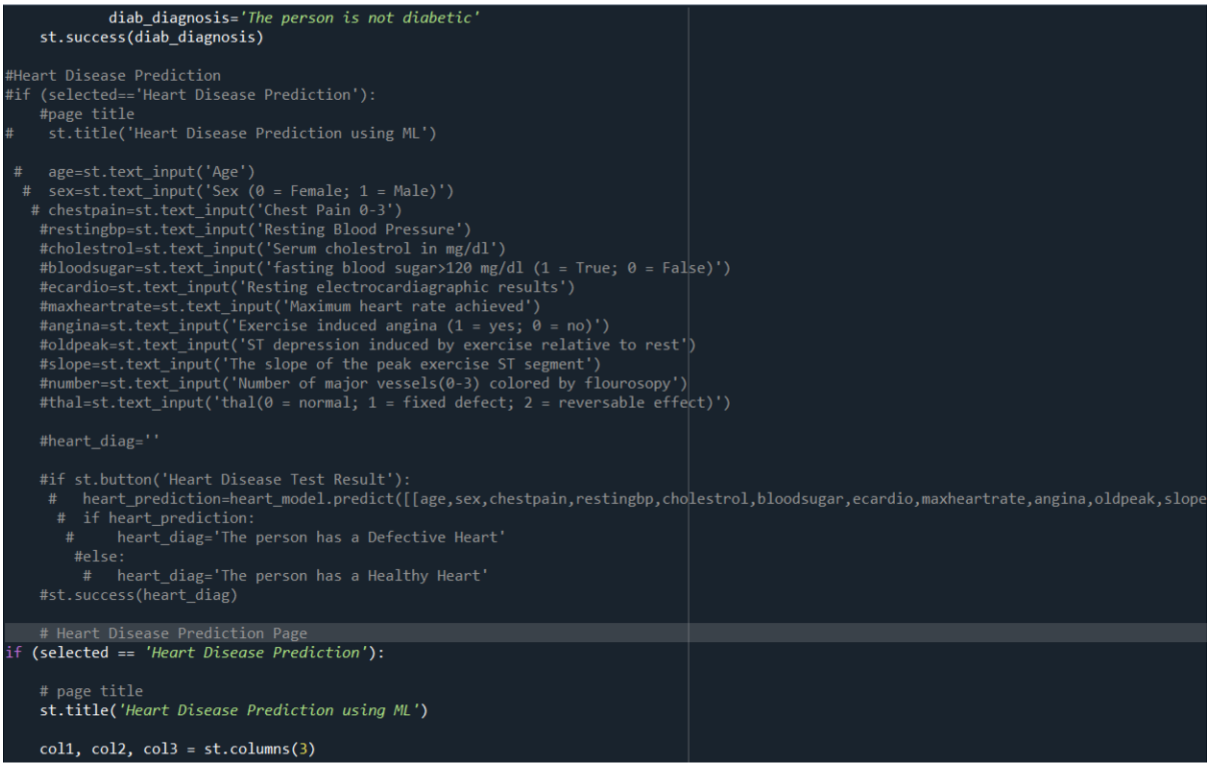
```
[0]
Person doesn't have heart disease
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
X does not have valid feature names, but LogisticRegression was fitted
with feature names
  warnings.warn(
import pickle
filename='heart_model.sav'
pickle.dump(clf,open(filename,'wb'))
```

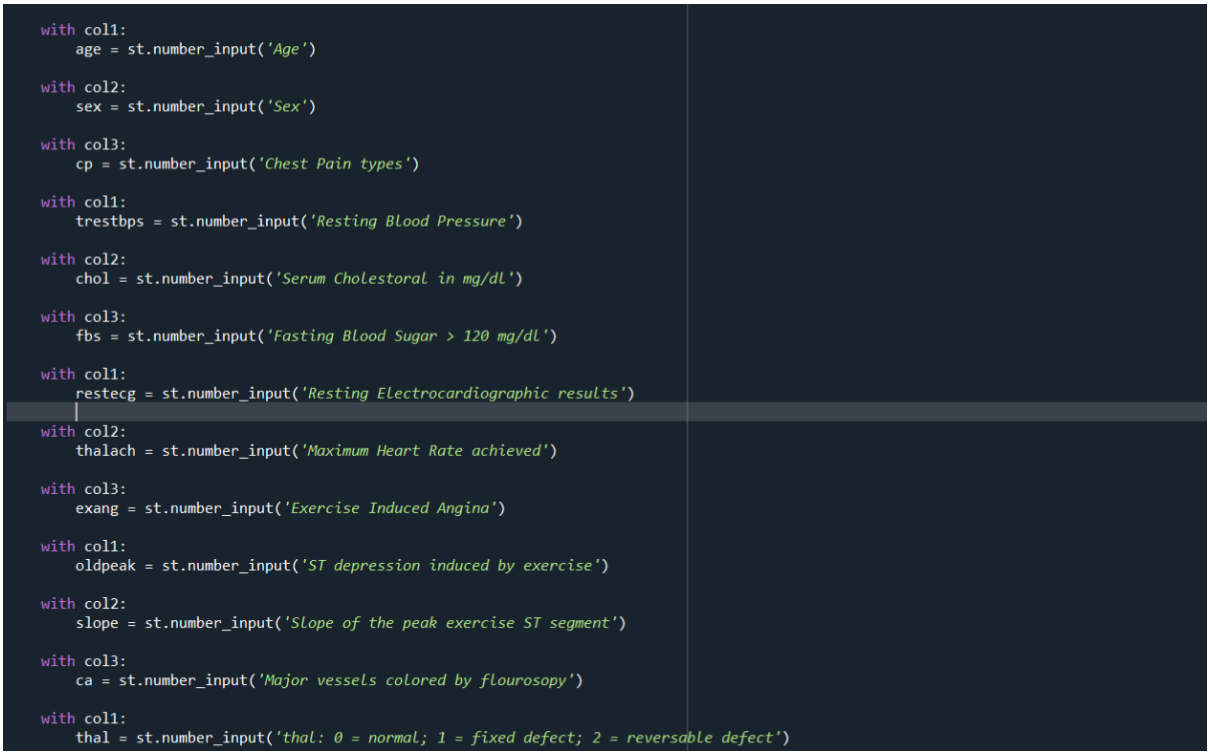
Making Multiple Disease Prediction System using stream-lit:



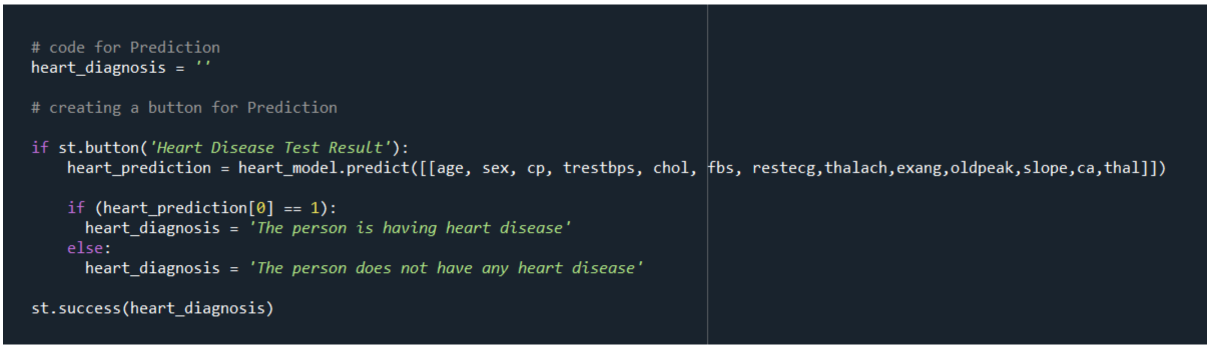
```
import pickle
import streamlit as st
from streamlit_option_menu import option_menu

#loading saved models
diabetes_model=pickle.load(open("C:/Users/taman/Desktop/Multiple Disease Prediction/diabetes_model.sav",'rb'))
heart_model=pickle.load(open("C:/Users/taman/Desktop/Multiple Disease Prediction/heart_model.sav",'rb'))
#parkinson_model=pickle.load(open("C:/Users/taman/Desktop/Multiple Disease Prediction/parkinson_model.sav",'rb'))

#side bar/navigation
with st.sidebar:
    selected=option_menu('Multiple Disease Prediction System',['Diabetes Prediction','Heart Disease Prediction'],icons=['activity','heart'],c


#Diabetes Prediction System
if (selected=='Diabetes Prediction'):
    #page title
    st.title('Diabetes Prediction using ML')

    Pregnancies=st.text_input('Number of Pregnancies')
    Glucose=st.text_input('Glucose level')
    BloodPressure=st.text_input('Blood Pressure Value')
    SkinThickness=st.text_input('Skin Thickness Value')
    Insulin=st.text_input('Insulin level')
    BMI=st.text_input('BMI value')
    DiabetesPedigreeFunction=st.text_input('Diabetes Pedigree Function Value')
    Age=st.text_input('Age of the person')

    #code for Prediction
    diab_diagnosis=''

    #creating a button
    if st.button('Diabetes Test Result'):
        diab_prediction=diabetes_model.predict([[Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age]])
        if diab_prediction:
            diab_diagnosis='The person is Diabetic'
        else:
            diab_diagnosis='The person is not diabetic'
```



```
            diab_diagnosis='The person is not diabetic'
    st.success(diab_diagnosis)

#Heart Disease Prediction
#if (selected=='Heart Disease Prediction'):
    #page title
#    st.title('Heart Disease Prediction using ML')

 #   age=st.text_input('Age')
  #  sex=st.text_input('Sex (0 = Female; 1 = Male)')
   # chestpain=st.text_input('Chest Pain 0-3')
    #restingbp=st.text_input('Resting Blood Pressure')
    #cholestrol=st.text_input('Serum cholestrol in mg/dl')
    #bloodsugar=st.text_input('fasting blood sugar>120 mg/dl (1 = True; 0 = False)')
    #ecardio=st.text_input('Resting electrocardiagraphic results')
    #maxheartrate=st.text_input('Maximum heart rate achieved')
    #angina=st.text_input('Exercise induced angina (1 = yes; 0 = no)')
    #oldpeak=st.text_input('ST depression induced by exercise relative to rest')
    #slope=st.text_input('The slope of the peak exercise ST segment')
    #number=st.text_input('Number of major vessels(0-3) colored by flourosopy')
    #thal=st.text_input('thal(0 = normal; 1 = fixed defect; 2 = reversable effect)')

    #heart_diag=''

    #if st.button('Heart Disease Test Result'):
     #   heart_prediction=heart_model.predict([[age,sex,chestpain,restingbp,cholestrol,bloodsugar,ecardio,maxheartrate,angina,oldpeak,slope
      #  if heart_prediction:
       #     heart_diag='The person has a Defective Heart'
        #else:
         #   heart_diag='The person has a Healthy Heart'
    #st.success(heart_diag)

    # Heart Disease Prediction Page
if (selected == 'Heart Disease Prediction'):

    # page title
    st.title('Heart Disease Prediction using ML')

    col1, col2, col3 = st.columns(3)
```



```
with col1:
    age = st.number_input('Age')

with col2:
    sex = st.number_input('Sex')

with col3:
    cp = st.number_input('Chest Pain types')

with col1:
    trestbps = st.number_input('Resting Blood Pressure')

with col2:
    chol = st.number_input('Serum Cholestoral in mg/dl')

with col3:
    fbs = st.number_input('Fasting Blood Sugar > 120 mg/dl')

with col1:
    restecg = st.number_input('Resting Electrocardiographic results')

with col2:
    thalach = st.number_input('Maximum Heart Rate achieved')

with col3:
    exang = st.number_input('Exercise Induced Angina')

with col1:
    oldpeak = st.number_input('ST depression induced by exercise')

with col2:
    slope = st.number_input('Slope of the peak exercise ST segment')

with col3:
    ca = st.number_input('Major vessels colored by flourosopy')

with col1:
    thal = st.number_input('thal: 0 = normal; 1 = fixed defect; 2 = reversable defect')
```



```
# code for Prediction
heart_diagnosis = ''

# creating a button for Prediction

if st.button('Heart Disease Test Result'):
    heart_prediction = heart_model.predict([[age, sex, cp, trestbps, chol, fbs, restecg,thalach,exang,oldpeak,slope,ca,thal]])

    if (heart_prediction[0] == 1):
      heart_diagnosis = 'The person is having heart disease'
    else:
      heart_diagnosis = 'The person does not have any heart disease'

st.success(heart_diagnosis)
```

FUTURE SCOPE OF IMPROVEMENT

ML models heavily rely on high-quality and diverse data for accurate predictions. Future improvements can focus on collecting larger and more comprehensive datasets that include a wider range of patient demographics, genetic information, lifestyle factors, and environmental data. Incorporating real-time data from wearable devices and electronic health records can also provide valuable insights.

Deep learning algorithms, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have shown great potential in healthcare applications.

ML can be leveraged to develop personalized disease prediction models that consider an individual's unique characteristics, such as genetic makeup, medical history, lifestyle, and environmental factors. By tailoring predictions and treatment recommendations to specific patients, healthcare providers can offer more targeted and effective interventions.

CERTIFICATE

This is to certify that Miss Tamanna Sharma of Lovely Professional University, registration number: 12110221, has successfully completed a project on *Multiple Disease prediction* using *Machine Learning with Python* under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School