



# **IG.3510 - Machine Learning**

## **PROJECT**

**Topic - Early stage diabetes risk prediction**

**Done by: Pradyumna KANAKAPURA HARISH BABU (61563)**

## Table of contents

1. Introduction.....	3
2. Data Description .....	3
3. Exploratory Data Analysis.....	3
4. Machine Learning.....	5
5. Deep learning.....	10
6. Results, conclusion and references.....	12
7. Appendix.....	13

## Introduction

Diabetes is a disorder where there is no proper secretion of insulin. This disorder will not regulate the sugar levels in the blood. It is known that this is primarily because of the imbalance in the hormones. More than 1.6 billion people leave this world because of this disorder. Back in my country India this is a very common disorder which does not have any permanent cure. The only way is to maintain the blood glucose level and monitor them with utmost determination. The motive of choosing the work on diabetes is because of couple of personal incidents. Half of my family whose age is greater than 50 years is affected by diabetes. This really motivates me to choose this subject for my Machine Learning project which gives me more understanding about the factors affecting the disorder. This study might give me more insights towards the understanding of determination of diabetes in the early stage. In this way we can prolog the disorder or even stand still against it along the life without getting affected to it.

## Data Description

For the purpose of the project we have referred the UCI Machine Learning repository to fetch the data for pur predictions. The dataset is taken based on the questionnaires answered by the patients from the city of Sylhet, Bangladesh and all the information is taken with the approval of the laboratory and doctor. The data did not have missing values. It seen that there are 16 attributes of data in order to predict the outcome of the model. The output will be either positive or negative, where positive indicates that the corresponding person with the the attributes is affected by diabetes and negative indicates that the person is free from the diabetes.

Except the age attribute, rest all other inputs are categorical in nature.

With this dataset descriptive statistics, supervised learning, unsupervised learning and deep learning models are applied or performed to detect the early stage diabetes in patients.

## Exploratory Data Analysis

Exploratory data analysis popularly known as EDA is a process of performing some initial investigations on the dataset to discover the structure and the content of the given dataset. It is often known as *Data Profiling*. It is an unavoidable step in the entire journey of data analysis right from the business understanding part to the deployment of the models created.

EDA is where we get the basic understanding of the data in hand which then helps us in the further process of Data Cleaning & Data Preparation.

## Descriptive Statistics

Descriptive statistics is a major part of EDA. Descriptive Statistics are measures that summarize important features of data, often with a single number. Producing descriptive statistics is a common first step to take after cleaning and preparing a data set for analysis. We've already seen several examples of deceptive statistics in earlier lessons, such as means and medians.

## Measures of Center

Measures of center are statistics that give us a sense of the "middle" of a numeric variable. In other words, centrality measures give you a sense of a typical value you'd expect to see. Common measures of center include the mean, median and mode.

Although the mean and median both give us some sense of the center of a distribution, they aren't always the same. The median always gives us a value that splits the data into two halves while the mean is a numeric average so extreme values can have a significant impact on the mean.

Descriptive statistics help you explore features of your data, like center, spread and shape by summarizing them with numerical measurements. Descriptive statistics help inform the direction of an analysis and let you communicate your insights to others quickly and succinctly. In addition, certain values, like the mean and variance, are used in all sorts of statistical tests and predictive models.

Steps to follow for performing EDA:

1. Importing python libraries - Numpy, Pandas, matplotlib, seaborn and some other necessary libraries are imported. This step is necessary to start with any data analysis.
2. Loading the dataset - using `pd.read_csv()` method of the pandas library imported in 1st step of EDA. Data is being read from a CSV(Comma separated values) file into a Pandas DataFrame naming it as a dataset.
3. Basic data pre-processing- Before exploring the data, we must know about rows and columns in our dataset. What type of data is stored in each column(variable), basically to check the summary of the column. This can be done with the help of predefined methods available in python.

To check the size of the dataset, **`dataset.shape`** function is used. Output shows the size of rows and columns of the dataset.

The **`dataset.info()`** method is used to identify the data type of the variables in the dataset. The result's index is the original DataFrame's columns. The type of data will be stored as an object if there are strings present in the variables. Also, it will be int or float if the data has numerical and decimal values respectively.

Pandas function **`describe()`** is used to view some basic statistical details like count, percentiles, mean, std and maximum value of a data frame or a series of numeric values.

When we import our dataset from a CSV file, many blank columns are imported as null values into the Data Frame which can later create problems while operating that data frame. Pandas **`isnull()`** method is used to check and manage NULL values in a data frame.

# Machine Learning

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values.

Machine learning methods enable computers to operate autonomously without explicit programming. ML applications are fed with new data, and they can independently learn, grow, develop, and adapt. Machine learning derives insightful information from large volumes of data by leveraging algorithms to identify patterns and learn in an iterative process. ML algorithms use computation methods to learn directly from data instead of relying on any predetermined equation that may serve as a model.

Machine learning algorithms are molded on a training dataset to create a model. As new input data is introduced to the trained ML algorithm, it uses the developed model to make a prediction.

## Types of Machine Learning

Machine learning algorithms can be trained in many ways, with each method having its pros and cons. Based on these methods and ways of learning, machine learning is broadly categorized into two main types:

### 1. Supervised machine learning

This type of ML involves supervision, where machines are trained on labeled datasets and enabled to predict outputs based on the provided training. The labeled dataset specifies that some input and output parameters are already mapped. Hence, the machine is trained with the input and corresponding output. A device is made to predict the outcome using the test dataset in subsequent phases.

The primary objective of the supervised learning technique is to map the input variable (a) with the output variable (b). Supervised machine learning is further classified into two broad categories:

- **Classification:** These refer to algorithms that address classification problems where the output variable is categorical; for example, yes or no, true or false, male or female, etc. Real-world applications of this category are evident in spam detection and email filtering.
- **Regression:** Regression algorithms handle regression problems where input and output variables have a linear relationship. These are known to predict continuous output variables. Examples include weather prediction, market trend analysis, etc.

In my project, i have used three supervised learning methods :

1. **Support Vector Machine** 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.

```
[ ] from sklearn.svm import SVC
sv=SVC(kernel='linear',random_state=0)
sv.fit(X_train,y_train)

SVC(kernel='linear', random_state=0)

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=sv, X=X_train ,y=y_train,cv=10)
print("accuracy is {:.2f} %".format(accuracies.mean()*100))
print("std is {:.2f} %".format(accuracies.std()*100))

accuracy is 83.18 %
```

In the above image, it is seen that kernel='linear' as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x\_train, y\_train). With this model, accuracy is 83.18% for the training set. Now, predicting accuracy for test set, accuracy is 90.3% and confusion matrix for the same is :

```
Confusion Matrix for SVM
[[34  4]
 [ 6 60]]
```

And classification report for the same is :

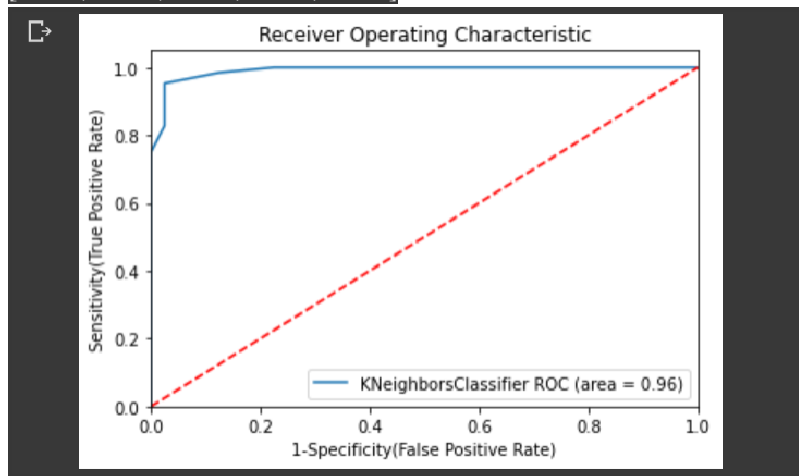
	precision	recall	f1-score	support
0	0.85	0.89	0.87	38
1	0.94	0.91	0.92	66
accuracy			0.90	104
macro avg	0.89	0.90	0.90	104
weighted avg	0.91	0.90	0.90	104

2. **K-Nearest Neighbor** is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

We run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

The result of using this algorithm, accuracy scores for this dataset are

[98.08, 98.08, 98.08, 97.12, 96.15]



3. **Decision Tree** is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

The result of using this technique, accuracy score for the training dataset is 91.84%.

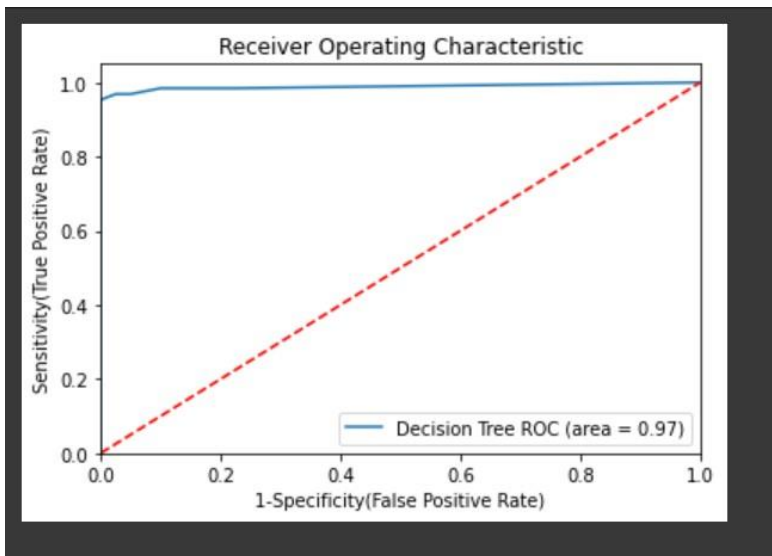
Now, let predict for test data and see accuracy score, confusion matrix and classification report,

```
0.9711538461538461
```

```
[[39  2]
 [ 1 62]]
```

	precision	recall	f1-score	support
0	0.97	0.95	0.96	41
1	0.97	0.98	0.98	63
accuracy			0.97	104
macro avg	0.97	0.97	0.97	104
weighted avg	0.97	0.97	0.97	104

ROC:



## 2. Unsupervised machine Learning

Unsupervised learning refers to a learning technique that's devoid of supervision. Here, the machine is trained using an unlabeled dataset and is enabled to predict the output without any supervision. An unsupervised learning algorithm aims to group the unsorted dataset based on the input's similarities, differences, and patterns.

Unsupervised machine learning is further classified into two types:

- **Clustering:** The clustering technique refers to grouping objects into clusters based on parameters such as similarities or differences between objects. For example, grouping customers by the products they purchase.
- **Association:** Association learning refers to identifying typical relations between the variables of a large dataset. It determines the dependency of various data items and maps associated variables. Typical applications include web usage mining and market data analysis.

For my project, **K-means clustering** is one of the simplest and popular unsupervised machine learning algorithms which is used for the dataset. A cluster refers to a collection of data points aggregated together because of certain similarities. The K-means algorithm identifies  $k$  number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids.

The predicted values for test data using this algorithm for the dataset is :

```
array([1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0], dtype=int32)
```



and accuracy score for the test data is 74.03%.

```
[50] import sklearn
print('Classification report:\n\n', sklearn.metrics.classification_report(y_test,kpred))

Classification report:

              precision    recall  f1-score   support

     0       0.62       0.82       0.71       40
     1       0.86       0.69       0.77       64

 accuracy       0.74
 macro avg       0.74
 weighted avg       0.77

[ ] from sklearn.metrics import confusion_matrix
print("Confusion Matrix :")

confusion_matrix(y_test, kpred),

Confusion Matrix :
(array([[33, 7],
       [20, 44]]),)
```

## Deep Learning

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

The term “deep” usually refers to the number of hidden layers in the neural network.

Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

In my project to perform deep learning, **MLP classifier** is being used which is a feedforward artificial neural network model that maps input data sets to a set of appropriate outputs. An MLP consists of multiple layers and each layer is fully connected to the following one. The nodes of the layers are neurons with nonlinear activation functions, except for the nodes of the input layer. Between the input and the output layer there may be one or more nonlinear hidden layers.

The following parameters at the model:

- `hidden_layer_sizes` : With this parameter we can specify the number of layers and the number of nodes we want to have in the Neural Network Classifier. Each element in the tuple represents the number of nodes at the *i*th position, where *i* is the index of the tuple. Thus, the length of the tuple indicates the total number of hidden layers in the neural network.
- `max_iter`: Indicates the number of epochs.

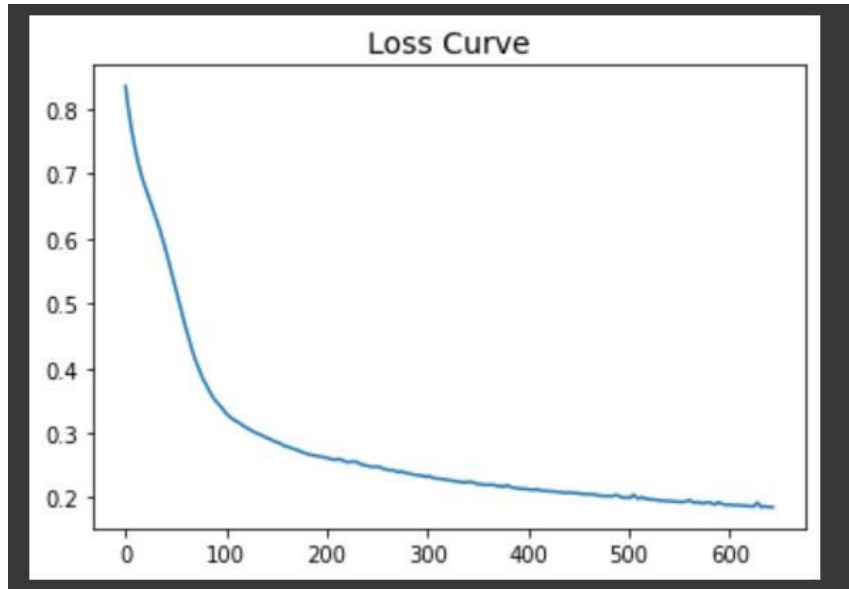
Accuracy score is 97% for this model, and confusion matrix, classification report ,

```
[ ] print(confusion_matrix(y_test,predictions))
```

```
[[39  1]
 [ 1 63]]
```

```
▶ print(classification_report(y_test,predictions))
```

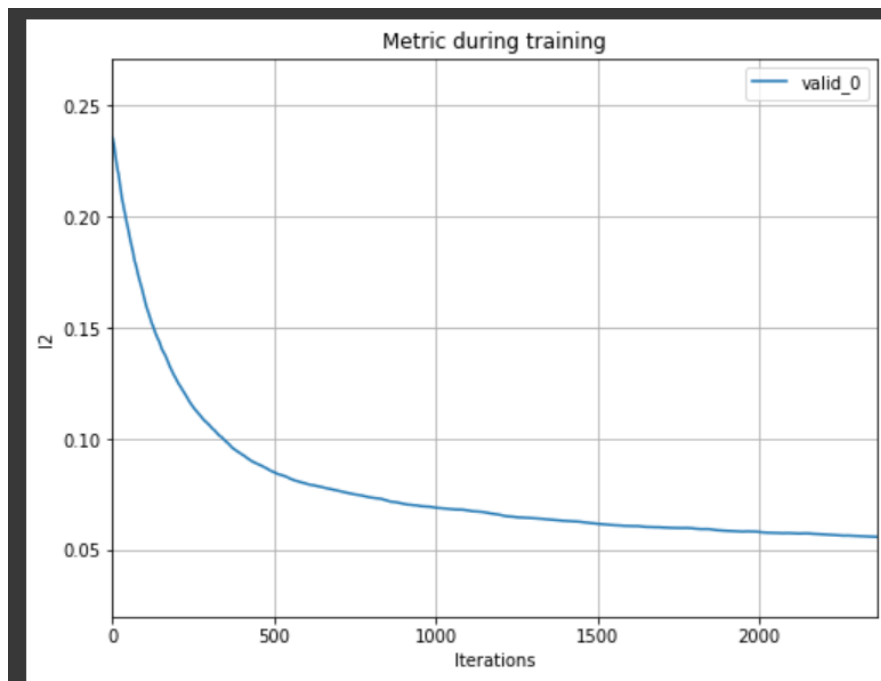
	precision	recall	f1-score	support
0	0.97	0.97	0.97	40
1	0.98	0.98	0.98	64
accuracy			0.98	104
macro avg	0.98	0.98	0.98	104
weighted avg	0.98	0.98	0.98	104



Above is the loss curve for the MLP Classifier model performed for the dataset.

Next, the deep learning model performed is **Light GBM Regressor**, is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Below plot shows the results of an evaluation metric during training,



Accuracy scores for training data is 76% and test data is 73%.

## Results

Following table shows the accuracy score, f-score and ROC values for the models performed on the dataset :

Model	Accuracy	F-Score	ROC
Support Vector Machines	94.4715%	0.9529	0.9866
K Nearest Neighbors	92.5436%	0.9337	0.9597
Decision Tree	94.2276%	0.9503	0.9445
K-means	74.03%	0.77	
Multi-Layer Perceptron	95.4413%	0.9507	0.9911
Light GBM	76		

Top three classifiers are Multi-Layer Perceptron, Decision Tree and Support Vector Machines as these classifiers performed well without overfitting in the test and train set.

## Conclusion

Machine learning and data mining techniques are valuable in disease diagnosis. The capability to predict diabetes early, assumes a vital role for the patient's appropriate treatment procedure.

Machine learning models can be used to predict various serious diseases like diabetes in humans at an early and curable stage. In this report we experimented with a diabetes dataset with different Machine Learning models.

The results drawn from training several machine learning models clearly indicate that Multi-Layer Perceptron Classifier proved to be the best model among the models used in the project for the concerned dataset with an accuracy score of 95.4413%, ROC score of 0.9911 and F-score of 0.9507.

## References

1. <http://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset>.
2. <https://iarjset.com/wp-content/uploads/2021/03/IARJSET.2021.8228.pdf>
3. <https://medium.com/analytics-vidhya/risk-prediction-of-diabetes-at-an-early-stage-using-machine-learning-approach-fe8eb256b6f8>

## APPENDIX

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix
%matplotlib inline
```

```
from google.colab import files
uploaded = files.upload()
```

diabetes\_data\_upload.csv

- **diabetes\_data\_upload.csv**(text/csv) - 34682 bytes, last modified: 5/19/2022 - 100% done  
Saving diabetes\_data\_upload.csv to diabetes\_data\_upload (1).csv

```
dataset=pd.read_csv("diabetes_data_upload.csv")
dataset
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring
<b>0</b>	40	1	0	1	0	1	0	0	0
<b>1</b>	58	1	0	0	0	1	0	0	1
<b>2</b>	41	1	1	0	0	1	1	0	0
<b>3</b>	45	1	0	0	1	1	1	1	0
<b>4</b>	60	1	1	1	1	1	1	0	1
...	...	...	...	...	...	...	...	...	...
<b>515</b>	39	0	1	1	1	0	1	0	0
<b>516</b>	48	0	1	1	1	1	1	0	0
<b>517</b>	58	0	1	1	1	1	1	0	1
<b>518</b>	32	0	0	0	0	1	0	0	1
<b>519</b>	42	1	0	0	0	0	0	0	0


520 rows × 17 columns



```
dataset.shape
```

(520, 17)

dataset.describe()

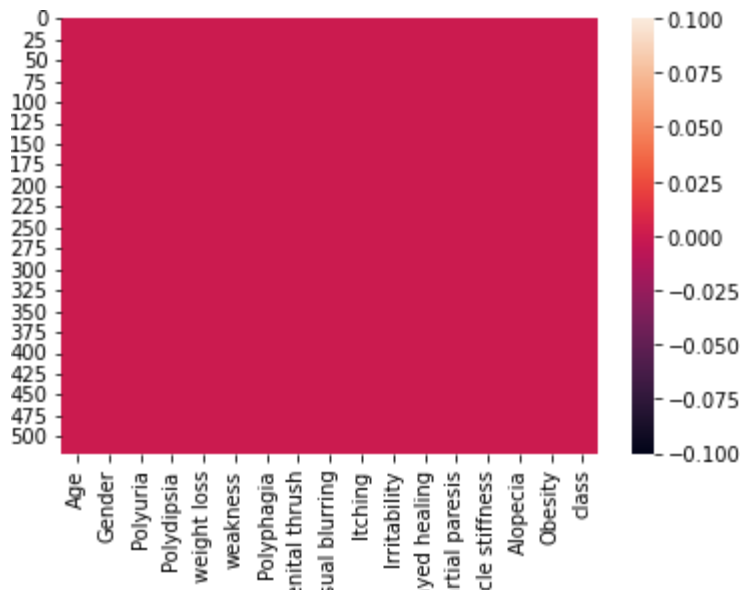
	Age 
<b>count</b>	520.000000
<b>mean</b>	48.028846
<b>std</b>	12.151466
<b>min</b>	16.000000
<b>25%</b>	39.000000
<b>50%</b>	47.500000
<b>75%</b>	57.000000
<b>max</b>	90.000000

dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Age                    520 non-null   int64
1   Gender                 520 non-null   object
2   Polyuria                520 non-null   object
3   Polydipsia             520 non-null   object
4   sudden weight loss     520 non-null   object
5   weakness                520 non-null   object
6   Polyphagia             520 non-null   object
7   Genital thrush         520 non-null   object
8   visual blurring        520 non-null   object
9   Itching                520 non-null   object
10  Irritability           520 non-null   object
11  delayed healing        520 non-null   object
12  partial paresis        520 non-null   object
13  muscle stiffness       520 non-null   object
14  Alopecia               520 non-null   object
15  Obesity                520 non-null   object
16  class                  520 non-null   object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

sns.heatmap(dataset.isnull())

&lt;matplotlib.axes.\_subplots.AxesSubplot at 0x7f2d16ef0210&gt;



```
dataset['class'].value_counts()
```

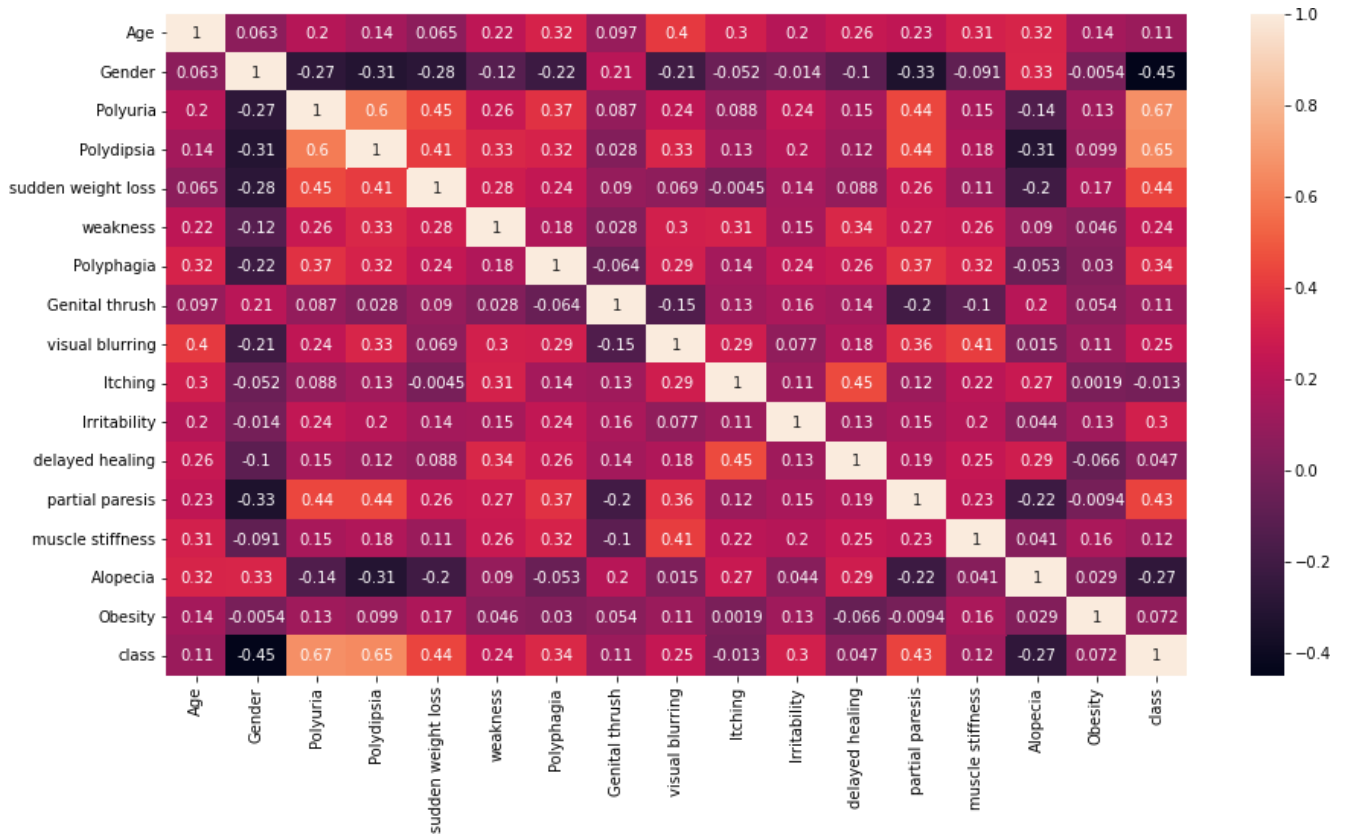
```
Positive    320
Negative    200
Name: class, dtype: int64
```

```
dataset['Gender'] = dataset['Gender'].map({'Male':1,'Female':0})
dataset['class'] = dataset['class'].map({'Positive':1,'Negative':0})
dataset['Polyuria'] = dataset['Polyuria'].map({'Yes':1,'No':0})
dataset['Polydipsia'] = dataset['Polydipsia'].map({'Yes':1,'No':0})
dataset['sudden weight loss'] = dataset['sudden weight loss'].map({'Yes':1,'No':0})
dataset['weakness'] = dataset['weakness'].map({'Yes':1,'No':0})
dataset['Polyphagia'] = dataset['Polyphagia'].map({'Yes':1,'No':0})
dataset['Genital thrush'] = dataset['Genital thrush'].map({'Yes':1,'No':0})
dataset['visual blurring'] = dataset['visual blurring'].map({'Yes':1,'No':0})
dataset['Itching'] = dataset['Itching'].map({'Yes':1,'No':0})
dataset['Irritability'] = dataset['Irritability'].map({'Yes':1,'No':0})
dataset['delayed healing'] = dataset['delayed healing'].map({'Yes':1,'No':0})
dataset['partial paresis'] = dataset['partial paresis'].map({'Yes':1,'No':0})
dataset['muscle stiffness'] = dataset['muscle stiffness'].map({'Yes':1,'No':0})
dataset['Alopecia'] = dataset['Alopecia'].map({'Yes':1,'No':0})
dataset['Obesity'] = dataset['Obesity'].map({'Yes':1,'No':0})
```

```
corrdata = dataset.corr()
```

```
ax,fig=plt.subplots(figsize=(15,8))
sns.heatmap(corrdata,annot=True)
```

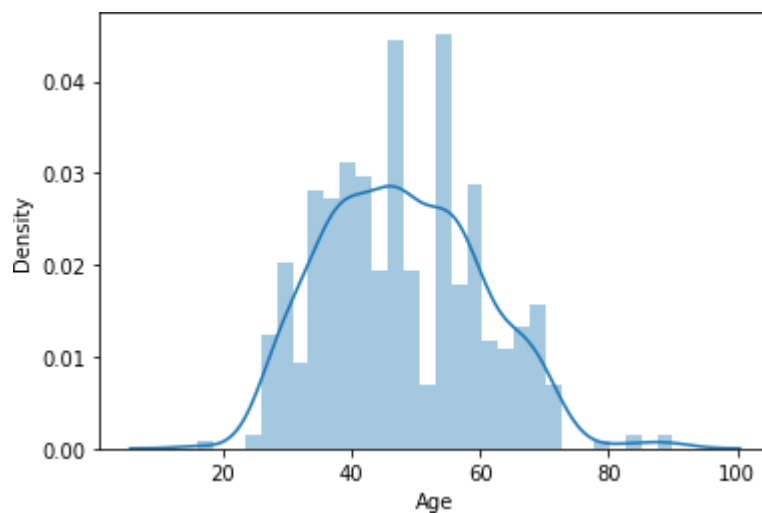
&lt;matplotlib.axes.\_subplots.AxesSubplot at 0x7f2d144a23d0&gt;



sns.distplot(dataset['Age'],bins=30).

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `d
warnings.warn(msg, FutureWarning)
```

&lt;matplotlib.axes.\_subplots.AxesSubplot at 0x7f2d13c7c0d0&gt;





```
X1 = dataset.iloc[:,0:-1]  
y1 = dataset.iloc[:, -1]
```


X1.columns

```
Index(['Age', 'Gender', 'Polyuria', 'Polydipsia', 'sudden weight loss',
      'weakness', 'Polyphagia', 'Genital thrush', 'visual blurring',
      'Itching', 'Irritability', 'delayed healing', 'partial paresis',
      'muscle stiffness', 'Alopecia', 'Obesity'],
      dtype='object')
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
best_feature = SelectKBest(score_func=chi2,k=10)
fit = best_feature.fit(X1,y1)
```

```
dataset_scores = pd.DataFrame(fit.scores_)
dataset_cols = pd.DataFrame(X1.columns)
```

```
featurescores=pd.concat([dataset_cols,dataset_scores],axis=1)
featurescores.columns=['column','scores']
featurescores
```

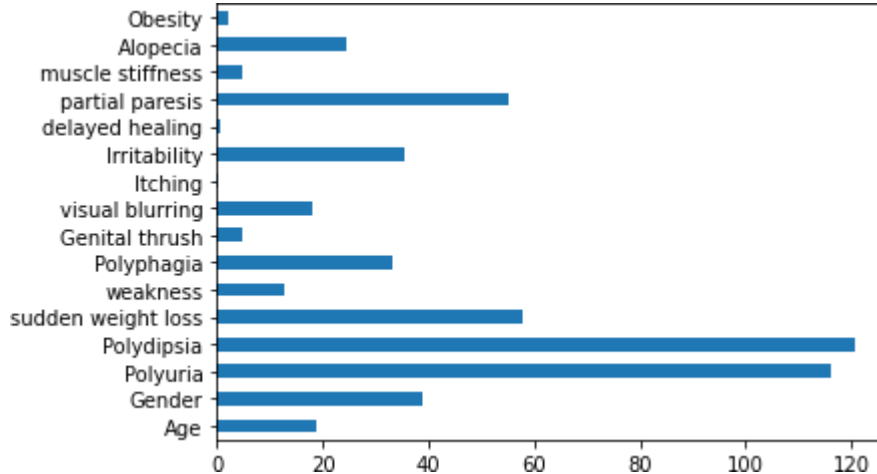
	column	scores	
0	Age	18.845767	
1	Gender	38.747637	
2	Polyuria	116.184593	
3	Polydipsia	120.785515	
4	sudden weight loss	57.749309	
5	weakness	12.724262	
6	Polyphagia	33.198418	
7	Genital thrush	4.914009	
8	visual blurring	18.124571	
9	Itching	0.047826	
10	Irritability	35.334127	
11	delayed healing	0.620188	
12	partial paresis	55.314286	
13	muscle stiffness	4.875000	
14	Alopecia	24.402793	
15	Obesity	2.250284	

```
print(featurescores.nlargest(10,'scores'))
```

	column	scores
3	Polydipsia	120.785515
2	Polyuria	116.184593
4	sudden weight loss	57.749309
12	partial paresis	55.314286
1	Gender	38.747637
10	Irritability	35.334127
6	Polyphagia	33.198418
14	Alopecia	24.402793
0	Age	18.845767
8	visual blurring	18.124571

```
featureview=pd.Series(fit.scores_,index=X1.columns)
featureview.plot(kind='barh')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f2d0fd9e5d0>



```
from sklearn.feature_selection import VarianceThreshold
feature_high_variance = VarianceThreshold(threshold=(0.5*(1-0.5)))
falls=feature_high_variance.fit(X1)
```

```
dataset_scores1 = pd.DataFrame(falls.variances_)
dat1 = pd.DataFrame(X1.columns)
```

```
high_variance=pd.concat([dataset_scores1,dat1],axis=1)
high_variance.columns=['variance','cols']
high_variance[high_variance['variance']>0.2]
```

	variance	cols
0	147.374168	Age
1	0.232899	Gender
2	0.249985	Polyuria
3	0.247304	Polydipsia
4	0.243162	sudden weight loss
5	0.242511	weakness
6	0.248044	Polyphagia
8	0.247304	visual blurring
9	0.249819	Itching
11	0.248369	delayed healing



```
X = dataset[['Polydipsia','sudden weight loss','partial paresis','Irritability','Polyphagia',
y = dataset['class']
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

## ▼ Supervised Learning

```
from sklearn.svm import SVC
sv=SVC(kernel='linear',random_state=0)
sv.fit(X_train,y_train)
```

```
SVC(kernel='linear', random_state=0)
```

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=sv, X=X_train ,y=y_train,cv=10)
print("accuracy is {:.2f} %".format(accuracies.mean()*100))
print("std is {:.2f} %".format(accuracies.std()*100))
```

```
accuracy is 83.18 %
std is 4.94 %
```

```
pre1=sv.predict(X_test)
```

```
svm_linear=accuracy_score(pre1,y_test)
print("Accuracy Score for SVM")
print(accuracy_score(pre1,y_test))
print("\nConfusion Matrix for SVM")
print(confusion_matrix(pre1,y_test))
```

```
Accuracy Score for SVM
0.9038461538461539
```

```
Confusion Matrix for SVM
[[34  4]
 [ 6 60]]
```

```
from sklearn.metrics import classification_report
print(classification_report(pre1,y_test))
```

	precision	recall	f1-score	support
0	0.85	0.89	0.87	38
1	0.94	0.91	0.92	66
accuracy			0.90	104
macro avg	0.89	0.90	0.90	104
weighted avg	0.91	0.90	0.90	104

```
from sklearn.svm import SVC
svrf=SVC(kernel='rbf',random_state=0)
svrf.fit(X_train,y_train)
```

```
SVC(random_state=0)
```

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=svrf, X=X_train ,y=y_train,cv=10)
print("accuracy is {:.2f} %".format(accuracies.mean()*100))
print("std is {:.2f} %".format(accuracies.std()*100))
```

```
accuracy is 88.47 %
std is 3.69 %
```

```
pre2=svrf.predict(X_test)
```

```
svm_rbf=accuracy_score(pre2,y_test)
print("Accuracy Score for SVM")
print(accuracy_score(pre2,y_test))
print("\nConfusion Matrix for SVM")
print(confusion_matrix(pre2,y_test))
```

Accuracy Score for SVM  
0.9807692307692307

Confusion Matrix for SVM  
[[39 1]  
[ 1 63]]

```
from sklearn.metrics import classification_report
print(classification_report(pre2,y_test))
```

	precision	recall	f1-score	support
0	0.97	0.97	0.97	40
1	0.98	0.98	0.98	64
accuracy			0.98	104
macro avg	0.98	0.98	0.98	104
weighted avg	0.98	0.98	0.98	104

```
from sklearn.neighbors import KNeighborsClassifier
score=[]
```

```
for i in range(1,10):
```

```
    knn=KNeighborsClassifier(n_neighbors=i,metric='minkowski',p=2)
    knn.fit(X_train,y_train)
    pre3=knn.predict(X_test)
    ans=accuracy_score(pre3,y_test)
    score.append(round(100*ans,2))
print(sorted(score,reverse=True)[:5])
knn=sorted(score,reverse=True)[:1]
```

```
[98.08, 98.08, 98.08, 97.12, 96.15]
```

```
from sklearn.tree import DecisionTreeClassifier
dc=DecisionTreeClassifier(criterion='gini')
dc.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
from sklearn.model_selection import cross_val_score
accuracies.=cross_val_score(estimator=dc,X=X_train,y=y_train,cv=10)
print("accuracy is {:.2f}%".format(accuracies.mean()*100))
print("std is {:.2f}%".format(accuracies.std()*100))
```

```
accuracy is 91.35 %
std is 3.91 %
```

```
pre5=dc.predict(X_test)
```

```
Decisiontress_classifier=accuracy_score(pre5,y_test)
print(accuracy_score(pre5,y_test))
print(confusion_matrix(pre5,y_test))
```

```
0.9711538461538461
[[39  2]
 [ 1 62]]
```

```
from sklearn.metrics import classification_report
print(classification_report(pre5,y_test))
```

```

┌→
precision    recall  f1-score   support

0           0.97      0.95      0.96         41
1           0.97      0.98      0.98         63

accuracy          0.97         104
macro avg          0.97      0.97      0.97         104
weighted avg       0.97      0.97      0.97         104
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
```

```
plt.figure()
```

```
models = [
```

```
    {
        'label': 'KNeighborsClassifier',
        'model': KNeighborsClassifier(n_neighbors=i,metric='minkowski',p=2),
    },
```

```
]
```

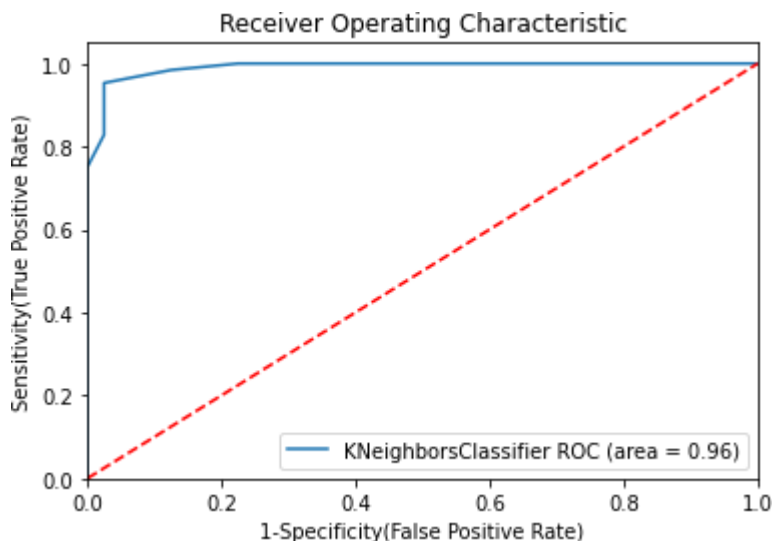
```
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
```

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,-1])
```

```
auc = metrics.roc_auc_score(y_test,model.predict(X_test))
```

```
plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))
```

```
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



```
from sklearn import metrics
import matplotlib.pyplot as plt
```

```
plt.figure()
```

```
models = [
```

```
{
    'label': 'Decision Tree',
    'model': DecisionTreeClassifier(criterion='gini'),
},
```

```
]
```

```
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
```

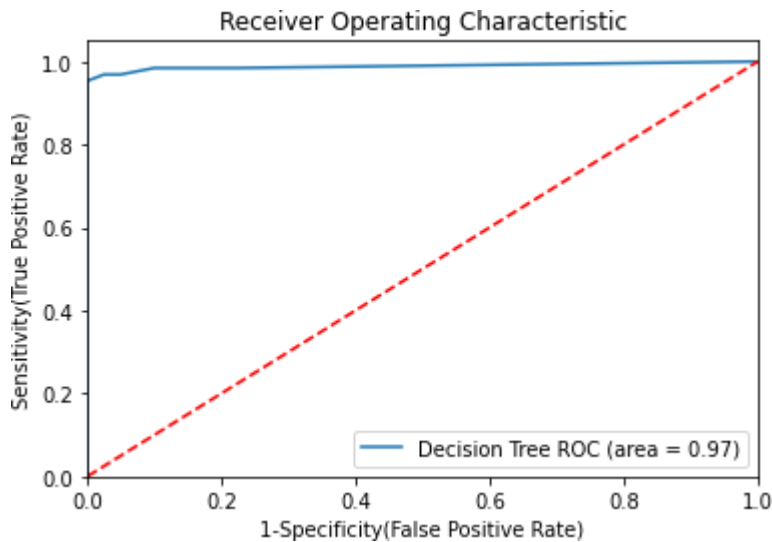
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,-1])
```

```
auc = metrics.roc_auc_score(y_test,model.predict(X_test))
```

```
plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))
```



```
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



## ▼ Unsupervised Learning

```
from sklearn.cluster import KMeans
KMeans_Clustering = KMeans(n_clusters=2, random_state=0)
KMeans_Clustering.fit(X_train)
```

```
KMeans(n_clusters=2, random_state=0)
```

```
kpred = KMeans_Clustering.predict(X_test)
kpred
```

```
array([1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0], dtype=int32)
```

```
print(accuracy_score(kpred,y_test))
```

```
0.7403846153846154
```

```
import sklearn
print('Classification report:\n\n', sklearn.metrics.classification_report(y_test,kpred))
```

Classification report:

	precision	recall	f1-score	support
0	0.62	0.82	0.71	40
1	0.86	0.69	0.77	64
accuracy			0.74	104
macro avg	0.74	0.76	0.74	104
weighted avg	0.77	0.74	0.74	104

```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix :")
```

```
confusion_matrix(y_test, kpred),
```

```
Confusion Matrix :
(array([[33,  7],
       [20, 44]]),)
```

## ▼ Deep Learning

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(11,11,11), max_iter=1000)
mlp.fit(X_train, y_train.values.ravel())
```

```
MLPClassifier(hidden_layer_sizes=(11, 11, 11), max_iter=1000)
```

```
predictions = mlp.predict(X_test)
print(predictions)
```

```
[1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 1 1 1 1
 0 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 0 1 1 1 0 1 1 0 0 1 1
 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1]
```

```
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, predictions)))
```

```
Accuracy: 0.97
```

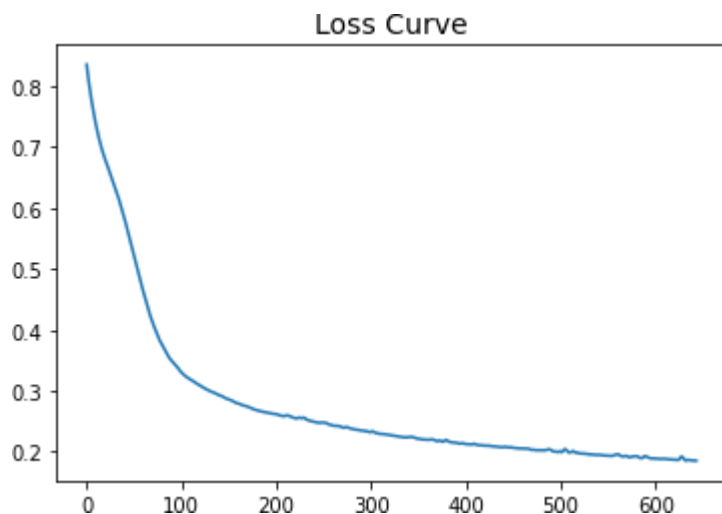
```
print(confusion_matrix(y_test,predictions))
```

```
[[39  1]
 [ 1 63]]
```

```
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.97	0.97	0.97	40
1	0.98	0.98	0.98	64
accuracy			0.98	104
macro avg	0.98	0.98	0.98	104
weighted avg	0.98	0.98	0.98	104

```
plt.plot(mlp.loss_curve_)
plt.title("Loss Curve", fontsize=14)
plt.show()
```



```
hyper_params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': ['l1', 'l2'],
    'learning_rate': 0.005,
    'feature_fraction': 0.7,
    'bagging_fraction': 0.6,
    'bagging_freq': 10,
    'verbose': 0,
    "max_depth": 12,
    # "num_leaves": 128,
    # "max_bin": 512,
    "num_iterations": 10000
}
```

```
from lightgbm import *
```

```
gbm = LGBMRegressor(**hyper_params)
```

```
gbm_model = gbm.fit(X_train, y_train,
                    eval_set=[(X_test, y_test)],
                    eval_metric='RMSE',
                    early_stopping_rounds=100)
```

[2310]	valid_0's l1: 0.17313	valid_0's rmse: 0.237158	valid_0's l2: 0.0562
[2311]	valid_0's l1: 0.173168	valid_0's rmse: 0.237143	valid_0's l2: 0.0562
[2312]	valid_0's l1: 0.173232	valid_0's rmse: 0.237156	valid_0's l2: 0.0562
[2313]	valid_0's l1: 0.173222	valid_0's rmse: 0.237122	valid_0's l2: 0.0562
[2314]	valid_0's l1: 0.173287	valid_0's rmse: 0.237137	valid_0's l2: 0.0562
[2315]	valid_0's l1: 0.173287	valid_0's rmse: 0.237132	valid_0's l2: 0.0562
[2316]	valid_0's l1: 0.173325	valid_0's rmse: 0.237118	valid_0's l2: 0.0562
[2317]	valid_0's l1: 0.173329	valid_0's rmse: 0.237117	valid_0's l2: 0.0562
[2318]	valid_0's l1: 0.173394	valid_0's rmse: 0.237132	valid_0's l2: 0.0562
[2319]	valid_0's l1: 0.173431	valid_0's rmse: 0.237119	valid_0's l2: 0.0562
[2320]	valid_0's l1: 0.173468	valid_0's rmse: 0.237106	valid_0's l2: 0.0562
[2321]	valid_0's l1: 0.173511	valid_0's rmse: 0.237092	valid_0's l2: 0.0562
[2322]	valid_0's l1: 0.173539	valid_0's rmse: 0.237065	valid_0's l2: 0.0562
[2323]	valid_0's l1: 0.173589	valid_0's rmse: 0.237058	valid_0's l2: 0.0561
[2324]	valid_0's l1: 0.173615	valid_0's rmse: 0.237035	valid_0's l2: 0.0561
[2325]	valid_0's l1: 0.173627	valid_0's rmse: 0.237028	valid_0's l2: 0.0561
[2326]	valid_0's l1: 0.173664	valid_0's rmse: 0.237037	valid_0's l2: 0.0561
[2327]	valid_0's l1: 0.173689	valid_0's rmse: 0.237014	valid_0's l2: 0.0561
[2328]	valid_0's l1: 0.173715	valid_0's rmse: 0.237007	valid_0's l2: 0.0561
[2329]	valid_0's l1: 0.173744	valid_0's rmse: 0.237021	valid_0's l2: 0.0561
[2330]	valid_0's l1: 0.173786	valid_0's rmse: 0.237008	valid_0's l2: 0.0561
[2331]	valid_0's l1: 0.173805	valid_0's rmse: 0.237005	valid_0's l2: 0.0561
[2332]	valid_0's l1: 0.173806	valid_0's rmse: 0.236998	valid_0's l2: 0.0561
[2333]	valid_0's l1: 0.173807	valid_0's rmse: 0.236984	valid_0's l2: 0.0561
[2334]	valid_0's l1: 0.173747	valid_0's rmse: 0.236919	valid_0's l2: 0.0561
[2335]	valid_0's l1: 0.173756	valid_0's rmse: 0.236919	valid_0's l2: 0.0561
[2336]	valid_0's l1: 0.173779	valid_0's rmse: 0.236918	valid_0's l2: 0.0561
[2337]	valid_0's l1: 0.173781	valid_0's rmse: 0.236916	valid_0's l2: 0.0561
[2338]	valid_0's l1: 0.173784	valid_0's rmse: 0.236901	valid_0's l2: 0.0561
[2339]	valid_0's l1: 0.173779	valid_0's rmse: 0.236876	valid_0's l2: 0.0561
[2340]	valid_0's l1: 0.173743	valid_0's rmse: 0.236834	valid_0's l2: 0.0560
[2341]	valid_0's l1: 0.17371	valid_0's rmse: 0.236819	valid_0's l2: 0.0560
[2342]	valid_0's l1: 0.173688	valid_0's rmse: 0.236803	valid_0's l2: 0.0560
[2343]	valid_0's l1: 0.173676	valid_0's rmse: 0.236798	valid_0's l2: 0.0560
[2344]	valid_0's l1: 0.173656	valid_0's rmse: 0.236787	valid_0's l2: 0.0560
[2345]	valid_0's l1: 0.173632	valid_0's rmse: 0.236779	valid_0's l2: 0.0560
[2346]	valid_0's l1: 0.173627	valid_0's rmse: 0.236752	valid_0's l2: 0.0560
[2347]	valid_0's l1: 0.173617	valid_0's rmse: 0.236737	valid_0's l2: 0.0560
[2348]	valid_0's l1: 0.173609	valid_0's rmse: 0.236715	valid_0's l2: 0.0560
[2349]	valid_0's l1: 0.173589	valid_0's rmse: 0.2367	valid_0's l2: 0.0560269
[2350]	valid_0's l1: 0.17358	valid_0's rmse: 0.236678	valid_0's l2: 0.0560
[2351]	valid_0's l1: 0.173551	valid_0's rmse: 0.236674	valid_0's l2: 0.0560
[2352]	valid_0's l1: 0.173524	valid_0's rmse: 0.23665	valid_0's l2: 0.0560033
[2353]	valid_0's l1: 0.173488	valid_0's rmse: 0.23662	valid_0's l2: 0.0559891
[2354]	valid_0's l1: 0.173451	valid_0's rmse: 0.236594	valid_0's l2: 0.0559

```

[2355] valid_0's l1: 0.17342    valid_0's rmse: 0.236574    valid_0's l2: 0.0559
[2356] valid_0's l1: 0.173388   valid_0's rmse: 0.236556    valid_0's l2: 0.0559
[2357] valid_0's l1: 0.173352   valid_0's rmse: 0.236544    valid_0's l2: 0.0559
[2358] valid_0's l1: 0.173315   valid_0's rmse: 0.23654    valid_0's l2: 0.0559512
[2359] valid_0's l1: 0.173305   valid_0's rmse: 0.236521    valid_0's l2: 0.0559
[2360] valid_0's l1: 0.173261   valid_0's rmse: 0.236492    valid_0's l2: 0.0559
[2361] valid_0's l1: 0.173234   valid_0's rmse: 0.236487    valid_0's l2: 0.0559
[2362] valid_0's l1: 0.173215   valid_0's rmse: 0.236468    valid_0's l2: 0.0559
[2363] valid_0's l1: 0.173203   valid_0's rmse: 0.236458    valid_0's l2: 0.0559
Early stopping, best iteration is:
[2263] valid_0's l1: 0.172941   valid_0's rmse: 0.237576    valid_0's l2: 0.0564

```

```
pred_tar = gbm_model.predict(X_test[:100000])
```

```
pred_tar
```

```

array([ 9.01898158e-01,  8.67658853e-01,  1.03822187e+00,  7.35419891e-02,
        5.05479774e-01,  8.46197022e-01,  9.45429424e-01,  7.57195030e-01,
        3.41959739e-01,  1.08241770e+00,  9.41112066e-01, -3.60251492e-02,
        7.70245597e-01,  6.25232812e-01,  8.12002207e-01,  9.83131168e-01,
        3.60858318e-01,  1.09232010e+00,  7.23540854e-01,  9.83131168e-01,
        8.84374124e-01,  7.15908815e-01,  9.95469619e-01,  1.40736274e-01,
        1.04963382e-01,  3.41959739e-01,  8.87028801e-01,  7.64060331e-01,
        8.51277379e-01,  8.78123154e-01,  1.07201572e+00,  2.74855492e-01,
        3.28859949e-01,  6.25232812e-01,  8.65816979e-01,  8.12461387e-01,
        1.14390427e+00,  3.27527956e-01,  1.03822187e+00,  9.17252925e-01,
        9.33135311e-01,  5.12414829e-02,  1.05102897e+00,  3.76022525e-01,
        1.04963382e-01,  9.83131168e-01,  9.57293321e-01,  2.82061931e-01,
        9.57293321e-01,  3.56060194e-01,  7.33091615e-01,  3.41959739e-01,
        9.79608157e-01,  7.35419891e-02,  9.57293321e-01,  4.75657783e-01,
        1.00069429e+00,  8.87286325e-01,  9.82048568e-01,  9.83131168e-01,
        2.83479012e-01,  1.08302698e-01,  1.08241770e+00,  6.01830807e-01,
        9.45076206e-01,  4.25186486e-01,  8.99756839e-01,  3.76022525e-01,
        1.00668187e+00,  9.57293321e-01, -3.67934338e-02,  3.13632573e-01,
        9.19177362e-01,  1.04818521e+00,  7.35419891e-02,  9.01898158e-01,
       -7.07243665e-02,  7.05993427e-01,  1.02784072e+00,  7.35419891e-02,
        8.60200383e-01,  1.09409055e+00, -3.60251492e-02,  7.35419891e-02,
        8.51277379e-01,  1.79596089e-01,  9.44245482e-01,  8.84374124e-01,
        8.40252140e-02,  8.27773617e-03,  8.44977406e-02,  8.40252140e-02,
        9.67450537e-04,  3.27527956e-01,  5.05479774e-01,  9.17696166e-01,
        8.78136545e-01,  6.40923775e-01,  4.45994095e-01,  7.35419891e-02,
        3.13632573e-01,  8.55568152e-01,  1.21274647e+00,  7.90056548e-01])

```

```

print("\nTest  R2 Score : %.2f"%gbm.score(X_train, y_train))
print("Train R2 Score : %.2f"%gbm.score(X_test, y_test))

```

```

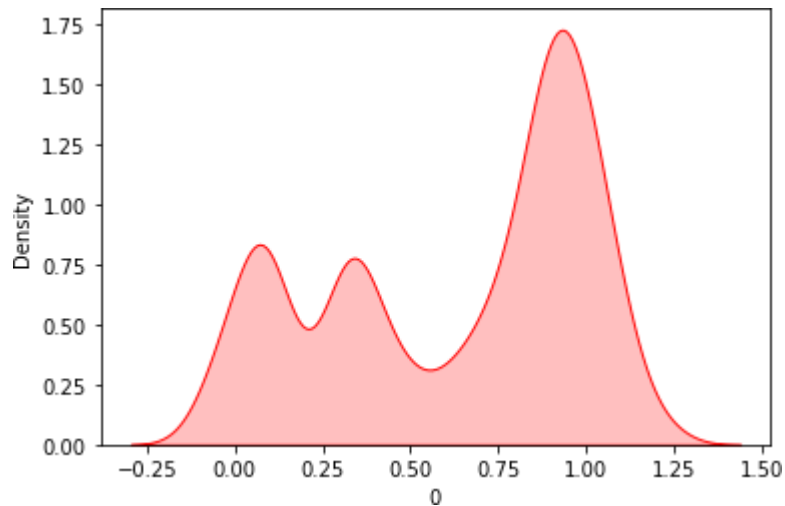
Test R2 Score : 0.73
Train R2 Score : 0.76

```

```
import seaborn as sns
```

```
sns.kdeplot(pred_tar[0], shade=True, bw=0.2, color="red")
plt.figure(figsize=(20,10))
plt.show()
```

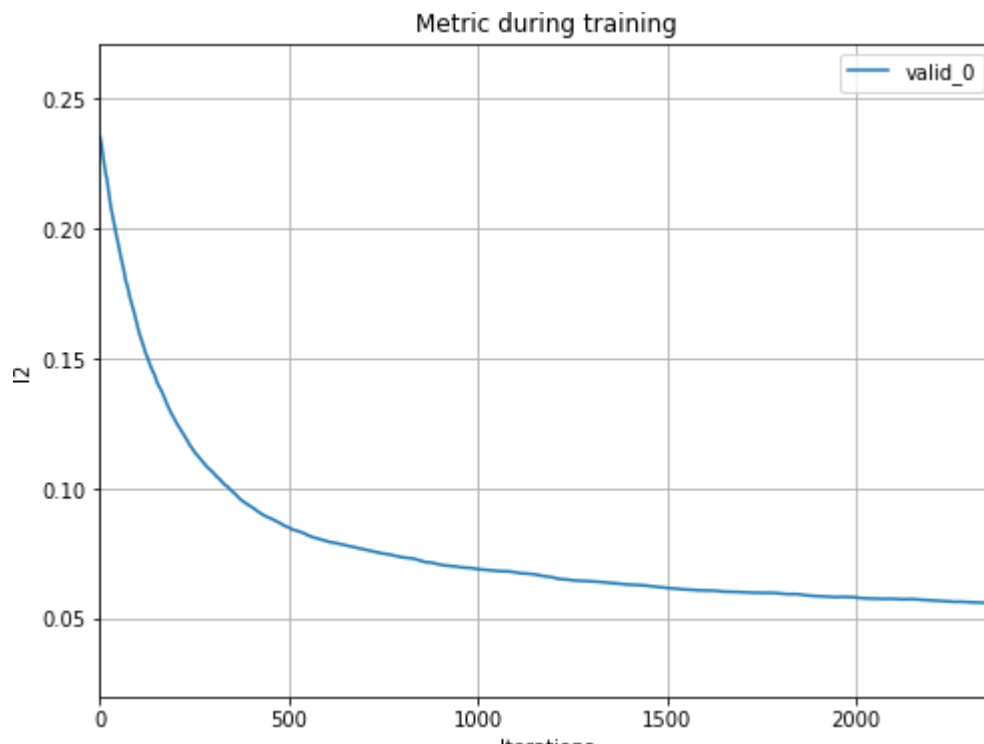
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:1699: FutureWarning: Th  
warnings.warn(msg, FutureWarning)



<Figure size 1440x720 with 0 Axes>

```
import lightgbm as lgb
lgb.plot_metric(gbm, figsize=(8,6));
```

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:2: UserWarning: more than



```
lgb.plot_tree(gbm, tree_index = 1, figsize=(20,15));
```



✓ 0s completed at 3:23 PM

