DHANALAKSHMI SRINIVASAN COLLEGE OF ENGINEERING & TECHNOLOGY

(Approved by AICTE & Affiliated to Anna University, Chennai)

East Cost Road, Mamallapuram, Chennai – 603 104

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

CCS355-NEURAL NETWORKS AND DEEP LEARNING LABORATORY

RECORD NOTE BOOK

Name	**..**
Reg.No	**..**
Year/Semester	**..**

DHANALAKSHMI SRINIVASAN COLLEGE OF ENGINEERING & TECHNOLOGY

(Approved by AICTE & Affiliated to Anna University,Chennai)

East Coast Road, Mamallapuram, Chennai - 603 104

Name	....
Reg. No	....
Department	....
Semester	....
Subject Name	....
Subject Code	....

Certified that this is the Bonafide record of the Practicals done for the above subject during the period .... in the academic year 20 - 20

Staff in Charge

Head of the Department

submitted for the University Practical Examination held on ....

Internal Examiner

External Examiner

TABLE OF CONTENTS				
Ex.No	**Date**	**Name of The Program**	**Page No**	**Staff Sign**
1		Simple Vector Addition in Tensorflow	1-5	
2		Implement a regression model in Keras	6-10	
3		Implement a perceptron in TensorFlow/Keras Environment	11-16	
4		Data Pre-processing in Machine Learning	17-21	
5		Simple Linear Regression using ScikitLearn	22-25	
6		Naïve Bayes Algorithm	26-30	
7		K-Nearest Neighbour Classifier(Ensembling Techniques)	31-36	
8		Decision Tree Classifier	37-42	
9		Random Forest Algorithm (Ensemble Learning)	42-49	
10		Support Vector Machine	50-54	
11		K-Means Clustering	55-58	
12		Implement a Feed-Forward Network in TensorFlow/Keras.	59-65	
13		Improve the Deep learning model by fine tuning hyper parameters.	66-72	
14		Implement an Image Classifier using CNN in TensorFlow/Keras	73-82	
15		Image generation using GAN	83-91	

Ex.No: 1	**Simple Vector Addition in Tensorflow**
Date:	

Aim:-

To write a python program to **demonstrate how to do Add two vectors in Tensorflow.**

Algorithm:

1. Import the Tensorflow packages in python
2. Assign a constant value to the tensor using tf.constant() method
3. Create two matrix in tensor using tf.constant() method
4. Print the value of the matrix
5. Create two tensor using tf.constant() method
6. Add the two matrix and store it in a tensor and print the value
7. Subtract the two matrix and store it in a tensor and print the value
8. Multiply the two matrix and store it in a tensor and print the value
9. Divide the two matrix and store it in a tensor and print the value
10. Calculate the dot product of two matrix and store it in a tensor and print the result of the tensor.
11. Stop the process

Program:

```
# importing packages
import tensorflow as tf

# creating a scalar
scalar = tf.constant(7)
scalar


scalar.ndim

# create a vector
vector = tf.constant([10, 10])

# checking the dimensions of vector
vector.ndim

# creating a matrix
matrix = tf.constant([[1, 2], [3, 4]])
print(matrix)
print('the number of dimensions of a matrix is :\
'+str(matrix.ndim))

# creating two tensors
matrix = tf.constant([[1, 2], [3, 4]])
matrix1 = tf.constant([[2, 4], [6, 8]])

# addition of two matrices
print(matrix+matrix1)

# creating two tensors
matrix = tf.constant([[1, 2], [3, 4]])
matrix1 = tf.constant([[2, 4], [6, 8]])

# subtraction of two matrices
print(matrix1 - matrix)

# creating two tensors
matrix = tf.constant([[1, 2], [3, 4]])
matrix1 = tf.constant([[2, 4], [6, 8]])

# multiplication of two matrices
print(matrix1 * matrix)
```

```
# creating two tensors
matrix = tf.constant([[1, 2],[3, 4]])
matrix1 = tf.constant([[2, 4],[6, 8]])

# division of two matrices
print(matrix1 / matrix)

# creating a matrix
matrix = tf.constant([[1, 2], [3, 4]])
# transpose of the matrix
print(tf.transpose(matrix))

# creating a matrix
matrix = tf.constant([[1, 2], [3, 4]])

# dot product of matrices
print('dot product of matrices is : ' +
      str(tf.tensordot(matrix, matrix, axes=1)))
```

Output:

```
#output for the scalar dimensions

Out[2]: <tf.Tensor: shape=(), dtype=int32, numpy=7>

the number of dimensions of a matrix is :2

#Output for Adding the two matrixes in tensor

tf.Tensor(

[[ 3  6]

 [ 9 12]], shape=(2, 2), dtype=int32)


#Output for Subtracting the two matrixes in tensor

tf.Tensor(

[[1 2]

 [3 4]], shape=(2, 2), dtype=int32)


#Output for Multiplying the two matrixes in tensor

tf.Tensor(

[[ 2  8]
```

```
 [18 32]], shape=(2, 2), dtype=int32)
```

#Output for Multiplying the two matrixes in tensor

```
tf.Tensor(
[[2. 2.]
 [2. 2.]], shape=(2, 2), dtype=float64)
```

#Output for Transpose the two matrixes in tensor

```
tf.Tensor(
[[1 3]
 [2 4]], shape=(2, 2), dtype=int32)
```

Result:-

A python program to demonstrate Simple Vector Addition has been implemented and executed successfully.

Ex.No: 2	**Implement a regression model in Keras.**
Date:	

Aim:-

To write a python program to **demonstrate how to do Simple Linear Regression in Keras.**

Algorithm:

1. Start the process
2. Import tensorflow, keras and scikitlearn library in python
3. Download the salary dataset and read the dataset using pandas library
4. Use the years of experience as input variable and the salary as the target variables
5. Split the dataset into training set and testing set into 80:20 ratio
6. Give input to the input layer using keras.Sequential() method
7. Compile the linear regression model having parameters optimer='adam', loss='mean_squared_error'.
8. Fit the linear regression model having batch size as 32, epochs=200.
9. Predict the target values using the regression model designed.
10. Plot the linear regression graph
11. Measure the performance of the model using Mean Squared Error
12. Stop the process

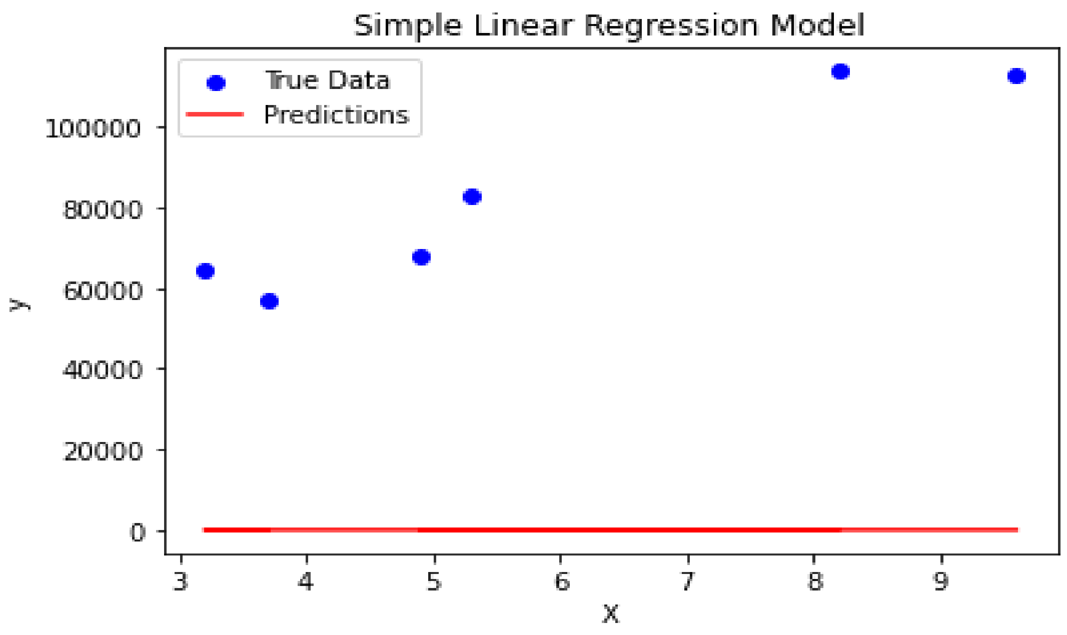
Program:

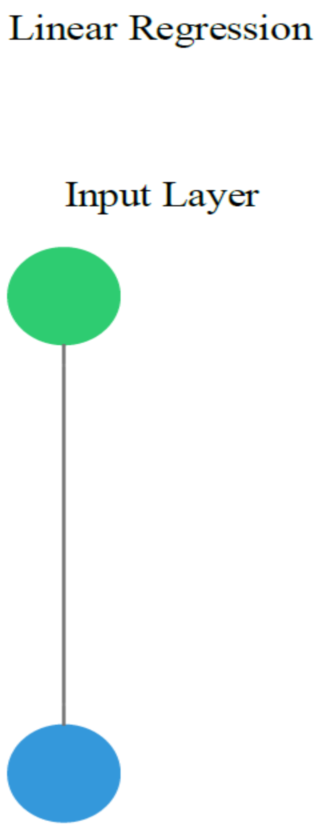
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = keras.Sequential([
    keras.layers.Input(shape=(1,)),
    keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=200, batch_size=32, verbose=1)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Visualize the data and the regression line
plt.scatter(X_test, y_test, color='blue', label='True Data')
plt.plot(X_test, y_pred, color='red', label='Predictions')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression Model')
plt.legend()
plt.show()

#Visualizing the neural network
from ann_visualizer.visualize import ann_viz
from graphviz import Source
ann_viz(model,title='Linear Regression')
graph_source=Source.from_file('network.gv')
```

Output

```
Epoch 1/200
1/1 [==============================] - 1s 1s/step - loss: 6270383616.0000
Epoch 2/200
1/1 [==============================] - 0s 3ms/step - loss: 6270382080.0000
Epoch 3/200
1/1 [==============================] - 0s 16ms/step - loss: 6270380544.0000
Epoch 4/200
1/1 [==============================] - 0s 0s/step - loss: 6270380544.0000
Epoch 5/200
1/1 [==============================] - 0s 0s/step - loss: 6270378496.0000
Epoch 6/200
1/1 [==============================] - 0s 0s/step - loss: 6270377984.0000
Epoch 7/200
1/1 [==============================] - 0s 0s/step - loss: 6270376448.0000
Epoch 8/200
1/1 [==============================] - 0s 16ms/step - loss: 6270375936.0000
Epoch 9/200
1/1 [==============================] - 0s 0s/step - loss: 6270375424.0000
Epoch 10/200
1/1 [==============================] - 0s 0s/step - loss: 6270373376.0000
………….
Epoch 200/200
1/1 [==============================] - 0s 16ms/step - loss: 6270372352.0000
```

Mean Squared Error: 7429406541.1625395


Simple Linear Regression Model
True Data
Predictions
100000
80000
60000
40000
20000
0
y
3
4
5
6
7
8
9
X


Linear Regression
Input Layer

Result:

Thus the program for Linear Regression using Keras had been implemented successfully and the output was verified.

Ex.No: 3	**Implement a perceptron in TensorFlow/Keras Environment**
Date:	

Aim:-

To write a python program to **demonstrate how to do Implement a Perceptron in TensorFlow/Keras Environment.**

Algorithm:

1. Start the process
2. Generate input values X1 and X2 using random function and use them as input values for the input layer of the Perceptron.
3. Assign the target values y to the perceptron
4. Give input to the input layer using keras.Sequential() method
5. Compile the linear regression model having parameters optimer=’adam’, loss=’mean_squared_error’.
6. Fit the linear regression model having batch size as 32, epochs=200.
7. Predict the target values using the regression model designed.
8. Visualize the Perceptron using Ann Visualizer
9. Stop the process

Program:

```
import tensorflow as tf
import numpy as np

# Generate some sample data
np.random.seed(42)
X = np.random.rand(100, 2)
y = np.where(X[:, 0] + X[:, 1] > 1, 1, 0)

# Define the perceptron model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(1, activation='sigmoid', input_shape=(2,))
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
#model.compile()

# Train the model
model.fit(X, y, epochs=100, verbose=1)

# Make predictions
predictions = model.predict(X)

# Print the predicted values
for i in range(len(predictions)):
    print(f"Input: {X[i]}, Predicted Output: {predictions[i]}, Actual Output: {y[i]}")


#Visualizing the neural network
from ann_visualizer.visualize import ann_viz
from graphviz import Source
ann_viz(model,title='Single Perceptron')
graph_source=Source.from_file('network.gv')
```

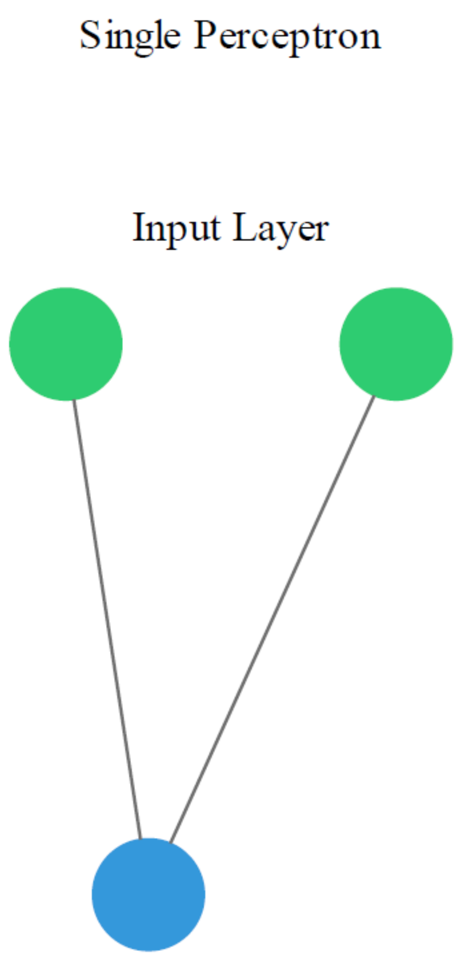
Output:

```
Epoch 1/100
4/4 [==============================] - 1s 16ms/step - loss: 0.6888 - accuracy:
0.4300
Epoch 2/100
4/4 [==============================] - 0s 0s/step - loss: 0.6876 - accuracy: 0.4300
Epoch 3/100
```

```
4/4 [==============================] - 0s 0s/step - loss: 0.6867 - accuracy: 0.4300
Epoch 4/100
4/4 [==============================] - 0s 0s/step - loss: 0.6856 - accuracy: 0.4300
Epoch 5/100
4/4 [==============================] - 0s 7ms/step - loss: 0.6846 - accuracy:
0.4300
Epoch 6/100
4/4 [==============================] - 0s 0s/step - loss: 0.6835 - accuracy: 0.4300
Epoch 7/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6826 - accuracy:
0.4300
Epoch 8/100
4/4 [==============================] - 0s 0s/step - loss: 0.6816 - accuracy: 0.4300
Epoch 9/100
4/4 [==============================] - 0s 0s/step - loss: 0.6808 - accuracy: 0.4300
Epoch 10/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6798 - accuracy:
0.4300
Epoch 11/100
4/4 [==============================] - 0s 0s/step - loss: 0.6788 - accuracy: 0.4300
Epoch 12/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6780 - accuracy:
0.4300
Epoch 13/100
4/4 [==============================] - 0s 0s/step - loss: 0.6771 - accuracy: 0.4300
Epoch 14/100
4/4 [==============================] - 0s 0s/step - loss: 0.6763 - accuracy: 0.4300
Epoch 15/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6755 - accuracy:
0.4300
Epoch 16/100
4/4 [==============================] - 0s 0s/step - loss: 0.6745 - accuracy: 0.4300
Epoch 17/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6738 - accuracy:
0.4300
Epoch 18/100
4/4 [==============================] - 0s 0s/step - loss: 0.6729 - accuracy: 0.4300
Epoch 19/100
4/4 [==============================] - 0s 0s/step - loss: 0.6722 - accuracy: 0.4300
Epoch 20/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6714 - accuracy:
0.4300
Epoch 21/100
4/4 [==============================] - 0s 0s/step - loss: 0.6707 - accuracy: 0.4300
```

```
Epoch 22/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6699 - accuracy: 
0.4300
Epoch 23/100
4/4 [==============================] - 0s 0s/step - loss: 0.6689 - accuracy: 0.4300
Epoch 24/100
4/4 [==============================] - 0s 0s/step - loss: 0.6681 - accuracy: 0.4300
Epoch 25/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6672 - accuracy: 
0.4300
Epoch 26/100
4/4 [==============================] - 0s 0s/step - loss: 0.6662 - accuracy: 0.4300
Epoch 27/100
4/4 [==============================] - 0s 0s/step - loss: 0.6654 - accuracy: 0.4400
Epoch 28/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6644 - accuracy: 
0.4400
Epoch 29/100
4/4 [==============================] - 0s 0s/step - loss: 0.6635 - accuracy: 0.4400
Epoch 30/100
4/4 [==============================] - 0s 5ms/step - loss: 0.6625 - accuracy: 
0.4400
Epoch 31/100
4/4 [==============================] - 0s 0s/step - loss: 0.6618 - accuracy: 0.4400
```

```
Input: [0.73199394 0.59865848], Predicted Output: [0.63585544], Actual Output: 1
Input: [0.15601864 0.15599452], Predicted Output: [0.47456235], Actual Output: 0
Input: [0.05808361 0.86617615], Predicted Output: [0.5491509], Actual Output: 0
Input: [0.60111501 0.70807258], Predicted Output: [0.6265747], Actual Output: 1
Input: [0.02058449 0.96990985], Predicted Output: [0.55569535], Actual Output: 0
Input: [0.83244264 0.21233911], Predicted Output: [0.60581297], Actual Output: 1
Input: [0.18182497 0.18340451], Predicted Output: [0.482923], Actual Output: 0
Input: [0.30424224 0.52475643], Predicted Output: [0.5501375], Actual Output: 0
Input: [0.43194502 0.29122914], Predicted Output: [0.54333276], Actual Output: 0
Input: [0.61185289 0.13949386], Predicted Output: [0.5567235], Actual Output: 0
Input: [0.29214465 0.36636184], Predicted Output: [0.527305], Actual Output: 0
Input: [0.45606998 0.78517596], Predicted Output: [0.6106988], Actual Output: 1
Input: [0.19967378 0.51423444], Predicted Output: [0.52949077], Actual Output: 0
Input: [0.59241457 0.04645041], Predicted Output: [0.54109186], Actual Output: 0
```


Single Perceptron
Input Layer

Result

Thus the program to implement a perceptron in TensorFlow/Keras Environment had been implemented successfully and the output was verified.

Ex.No: 4	Data Pre-Processing in Machine Learning
Date:	

Aim:-

To write a python program to **demonstrate how to do Pre-Processing in machine learning.**

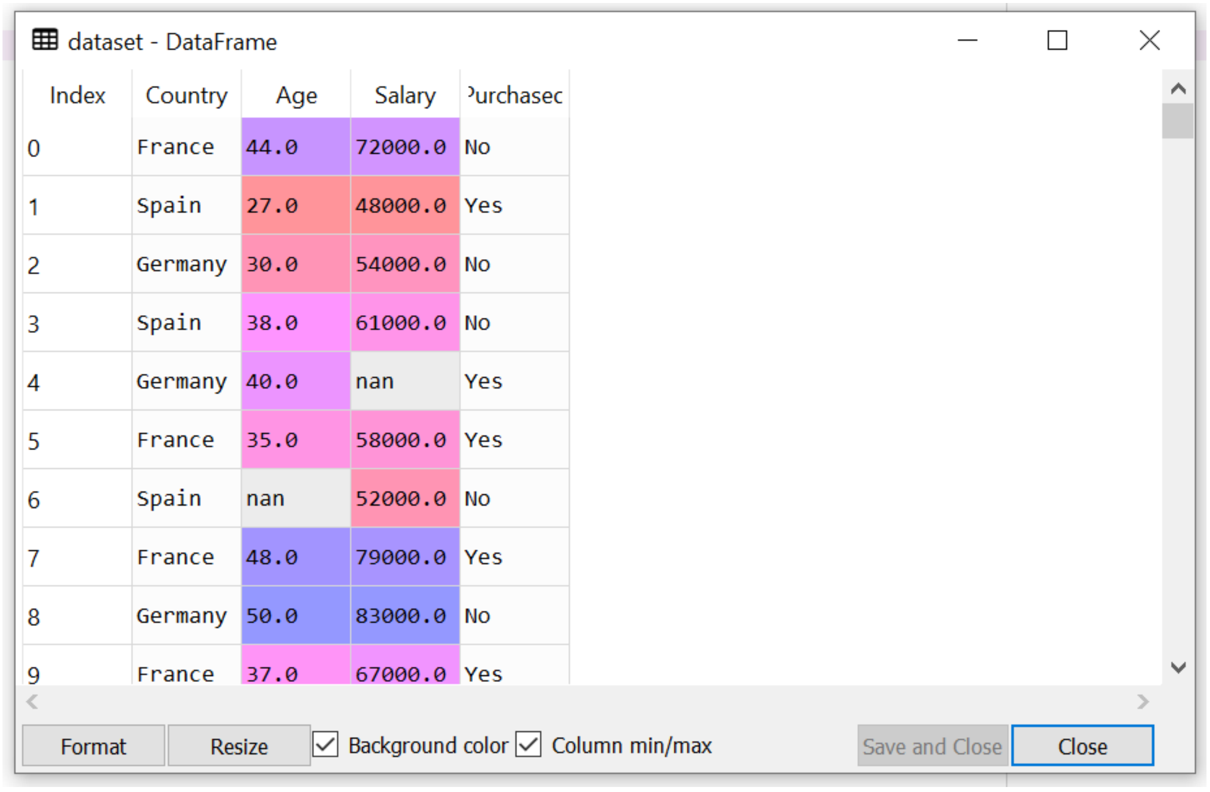
Procedure:-

1. Import the package pandas.
2. Import module label encoder from sklearn.preprocessing
3. Download the csv file 'insurance.csv' and stored in the drive E:
4. Display the data set which is not used for machine learning process.
5. To drop the duplicates from all the features use drop_duplicates and label encoder.
6. To transform the data in all the features use transform and fit method.
7. Display the data set after removing duplicates and transforms data which is used for machine learning process.

Program:-

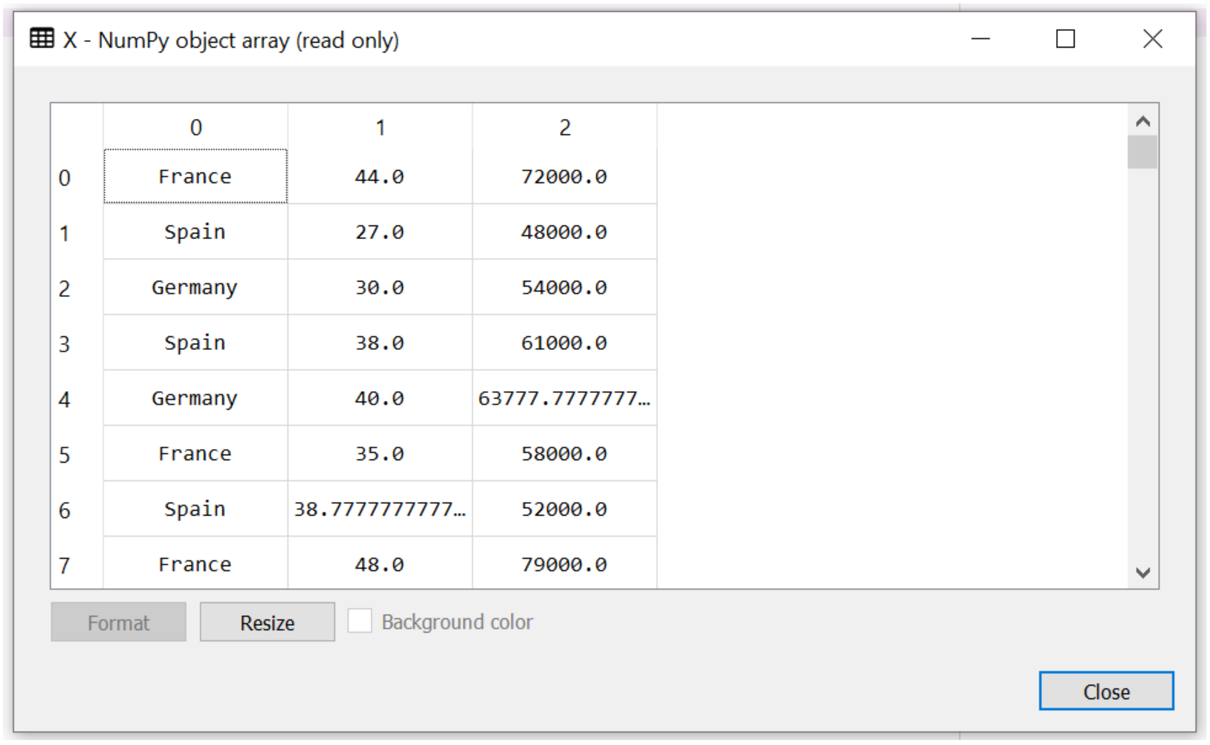
```
#Data preprocessing
#importing the libraries
import numpy as np
import matplotlib.pyplot
import pandas as pd

#importing the dataset
dataset=pd.read_csv('Data.csv')
X=dataset.iloc[:,:-1].values
Y=dataset.iloc[:,3].values

#taking care of the missing data
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan, strategy='mean')
#imputer=SimpleImputer()
imputer=imputer.fit(X[:,1:3])#the upperbound is excluded
X[:,1:3]=imputer.transform(X[:,1:3])
print(X)

# Encoding categorical data
# Encoding the Independent Variable Country
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(),
[0])],remainder='passthrough')
X=np.array(columnTransformer.fit_transform(X),dtype=np.str)
```

OUTUPUT



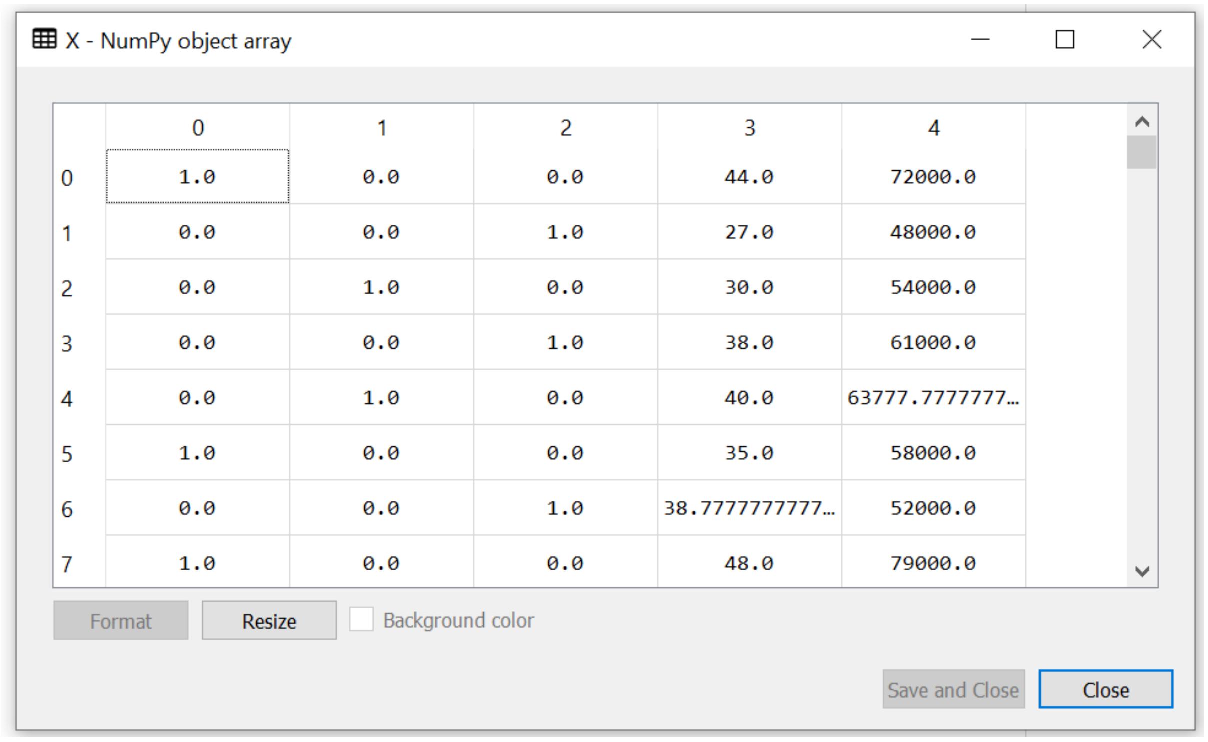
dataset - DataFrame

Index	Country	Age	Salary	Purchasec
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	nan	Yes
5	France	35.0	58000.0	Yes
6	Spain	nan	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes



X - NumPy object array (read only)

	0	1	2
0	France	44.0	72000.0
1	Spain	27.0	48000.0
2	Germany	30.0	54000.0
3	Spain	38.0	61000.0
4	Germany	40.0	63777.7777777…
5	France	35.0	58000.0
6	Spain	38.7777777777…	52000.0
7	France	48.0	79000.0



	0	1	2	3	4
0	1.0	0.0	0.0	44.0	72000.0
1	0.0	0.0	1.0	27.0	48000.0
2	0.0	1.0	0.0	30.0	54000.0
3	0.0	0.0	1.0	38.0	61000.0
4	0.0	1.0	0.0	40.0	63777.7777777…
5	1.0	0.0	0.0	35.0	58000.0
6	0.0	0.0	1.0	38.7777777777…	52000.0
7	1.0	0.0	0.0	48.0	79000.0

Result:-

A python program to demonstrate data pre-processing in machine learning has been implemented and executed successfully.

Ex.No: 5	Simple Linear Regression using ScikitLearn
Date:	

Aim:-

To write a python program to implement Simple Linear Regression technique on real time dataset in machine learning.

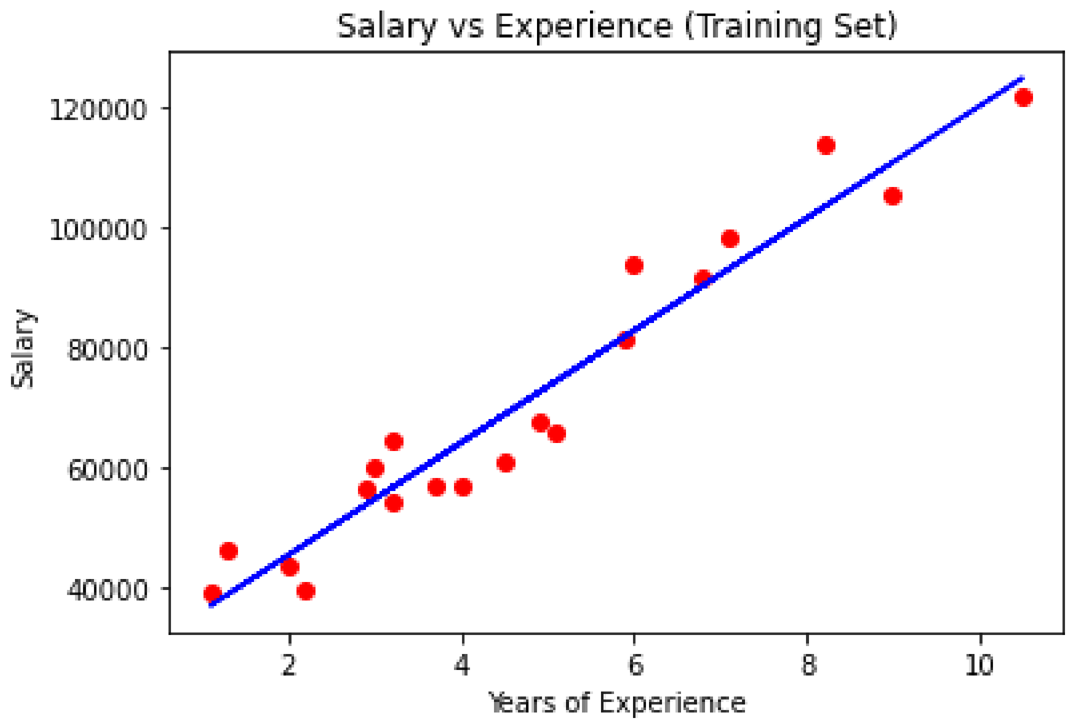
Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file ‘Salary_Data.csv' and stored in the corresponding folder
4. Assign the input variables in X and the target variables in y.
5. Load the data set into the data frame using read_csv.
6. from sklearn.model_selection import train_test_split and partition the dataset into training and testing dataset.
7. Import the Simple Linear Regression library from sklearn.linear_model and import LinearRegression
8. Fit the simple linear regression to the training set.
9. Visualize the regression line using matplotlib.

Program:-

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

#Fitting Simple Linear Regression to the Training Set
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,y_train)

#Predicting the Test set results
y_pred=regressor.predict(X_test)

#Visualising the Training Set Results
plt.scatter(X_train,y_train,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

#Visualising the Test Set Results
plt.scatter(X_test,y_test,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
```

Output:-


Salary vs Experience (Training Set)
Salary
120000
100000
80000
60000
40000
2
4
6
8
10
Years of Experience

Result:-

A python program for **simple linear regression** on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 6	**Naïve Bayes Algorithm**
Date:	

Aim:-

To write a python program to implement Naïve Bayes Algorithm on real time dataset in machine learning.

Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file 'diabetes.csv' and stored in the corresponding folder
4. Assign the input variables in X and the target variables in y.
5. Load the data set into the data frame using read_csv.
6. from sklearn.model_selection import train_test_split and partition the dataset into training and testing dataset.
7. Using sklearn.preprocessing library and import StandardScaler function to transform the dataset.
8. Fitting Naive Bayes algorithm to the Training set using sklearn.naive_bayes library and GaussianNB method.
9. Get the confusion matrix for the dataset using sklearn.metrics library using confusion_matrix method
10. Calculate the performance metrics for the naïve bayes model.

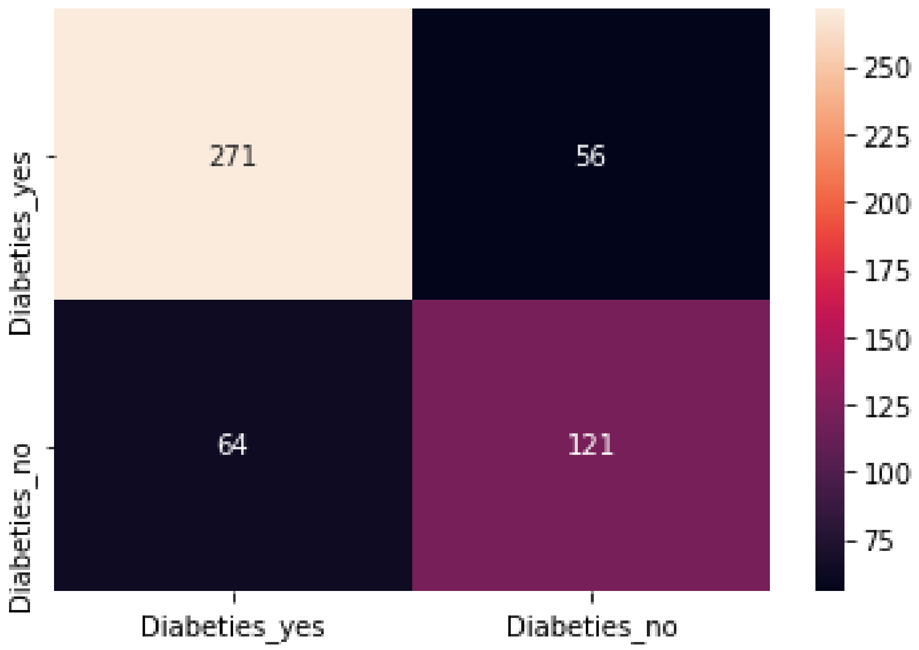
Program:-

```
# Naive Bayes Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns;

#Importing the dataset
dataset = pd.read_csv('diabetes.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 8]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 2)


# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Training set results
y_pred = classifier.predict(X_train)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred)

# Fitting Naive Bayes to the Testing set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_test, y_test)

# Predicting the Testing set results
y_pred1 = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test, y_pred1)

#create an empty data frame that we have to predict
```

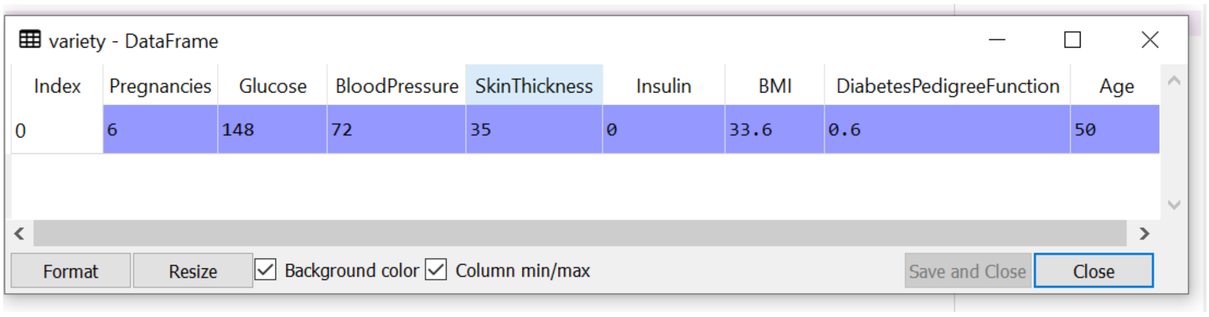
```
variety=pd.DataFrame()
variety['Pregnancies']=[6]
variety['Glucose']=[148]
variety['BloodPressure']=[72]
variety['SkinThickness']=[35]
variety['Insulin']=[0]
variety['BMI']=[33.6]
variety['DiabetesPedigreeFunction']=[0.627]
variety['Age']=[50]
print(variety)

y_pred1=classifier.predict(variety)
print("the Outcome of the Patient is:")
print(y_pred1)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)
```

Output:-

Performance metrics

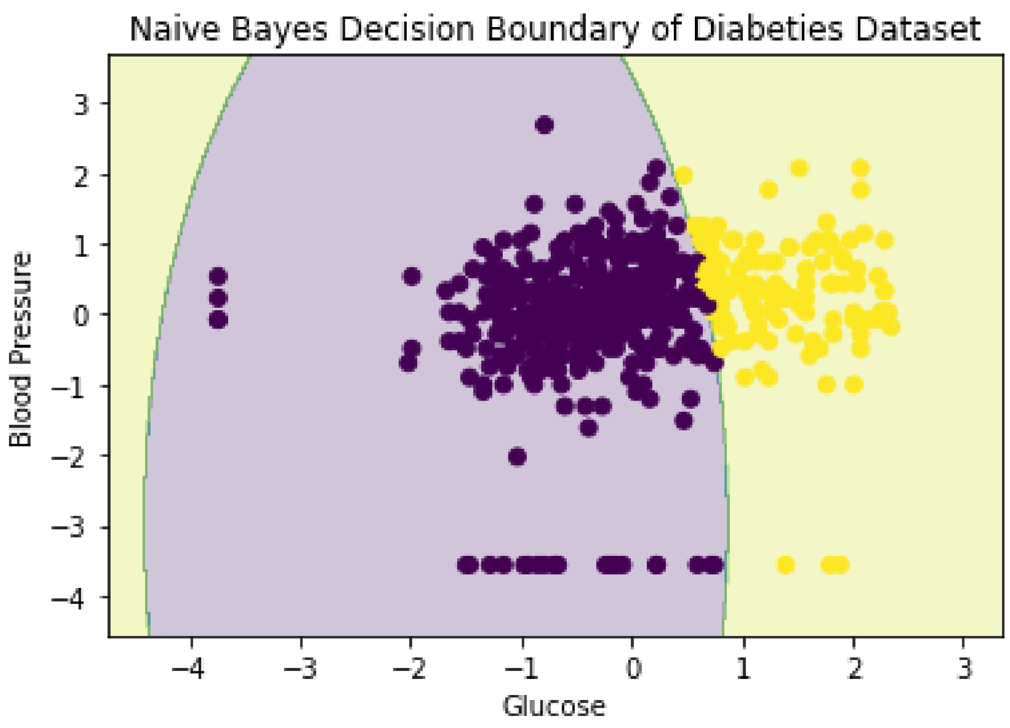


```
Recall [0.82874618 0.65405405]
Specificity [0.65405405 0.82874618]
Precision [0.80895522 0.68361582]
Negative Predictive Value [0.68361582 0.80895522]
False Positive Rate [0.34594595 0.17125382]
False Negative Rate [0.17125382 0.34594595]
False Discovery Rate [0.19104478 0.31638418]
Accuracry [0.765625 0.765625]
```



variety - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.6	50



Result:-

A python program for implementing Naïve Bayes Algorithm on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 7	**K-Nearest Neighbour Classifier(Ensembling Techniques)**
Date:	

Aim:-

To write a python program to implement Ensemble technique using K-Nearest Neighbour Algorithm on real time dataset in machine learning.

Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file ‘diabetes.csv' and stored in the corresponding folder
4. Assign the input variables in X and the target variables in y.
5. Load the data set into the data frame using read_csv.
6. from sklearn.model_selection import train_test_split and partition the dataset into training and testing dataset.
7. Using sklearn.preprocessing library and import StandardScaler function to transform the dataset.
8. Fitting K-Nearest algorithm to the Training set using sklearn.neighbors library and NeighborsClassifier method.
9. Get the confusion matrix for the dataset using sklearn.metrics library using confusion_matrix method
10. Calculate the performance metrics for the K-Nearest Neighbor model.

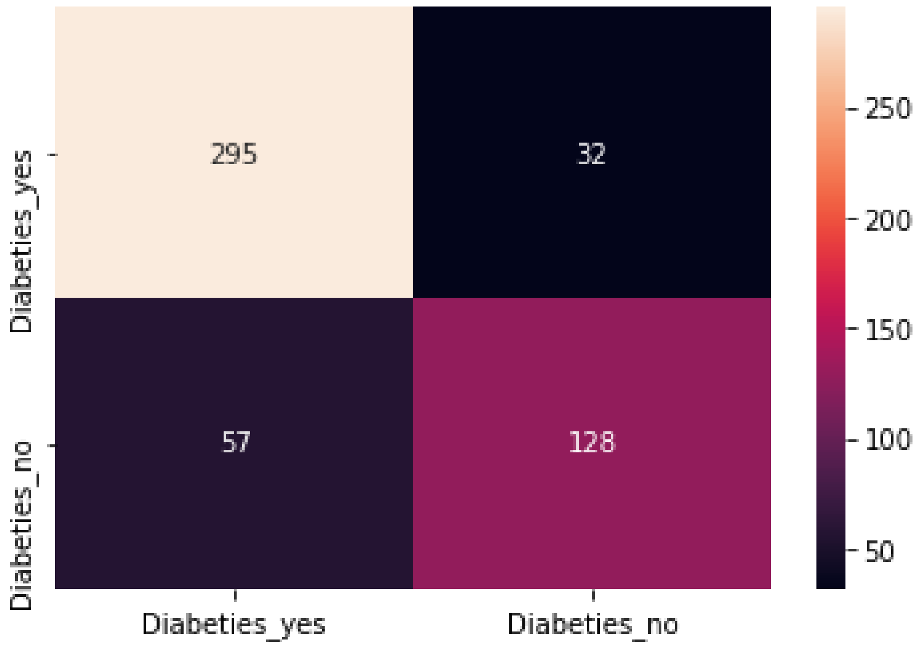
Program:-

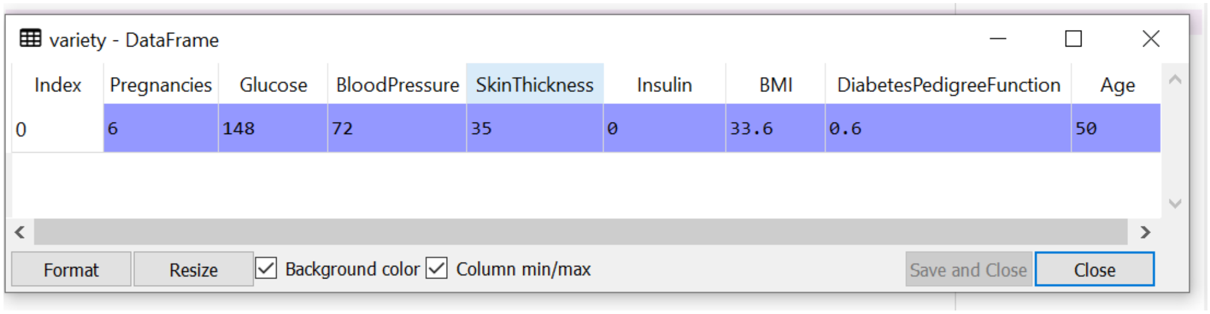
```
# Naive Bayes Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns;

#Importing the dataset
dataset = pd.read_csv('diabetes.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 8]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 2)


# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

# Predicting the Training set results
y_pred = classifier.predict(X_train)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)

#Calculating Performance Metrics for Training Set
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
```

```
TN = cm.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print("Recall",TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print("Specificity",TNR)
# Precision or positive predictive value
PPV = TP/(TP+FP)
print("Precision",PPV)
# Negative predictive value
NPV = TN/(TN+FN)
print("Negative Predictive Value",NPV)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print("False Positive Rate",FPR)
# False negative rate
FNR = FN/(TP+FN)
print("False Negative Rate",FNR)
# False discovery rate
FDR = FP/(TP+FP)
print("False Discovery Rate",FDR)
# Overall accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
print("Accuracry",ACC)
```

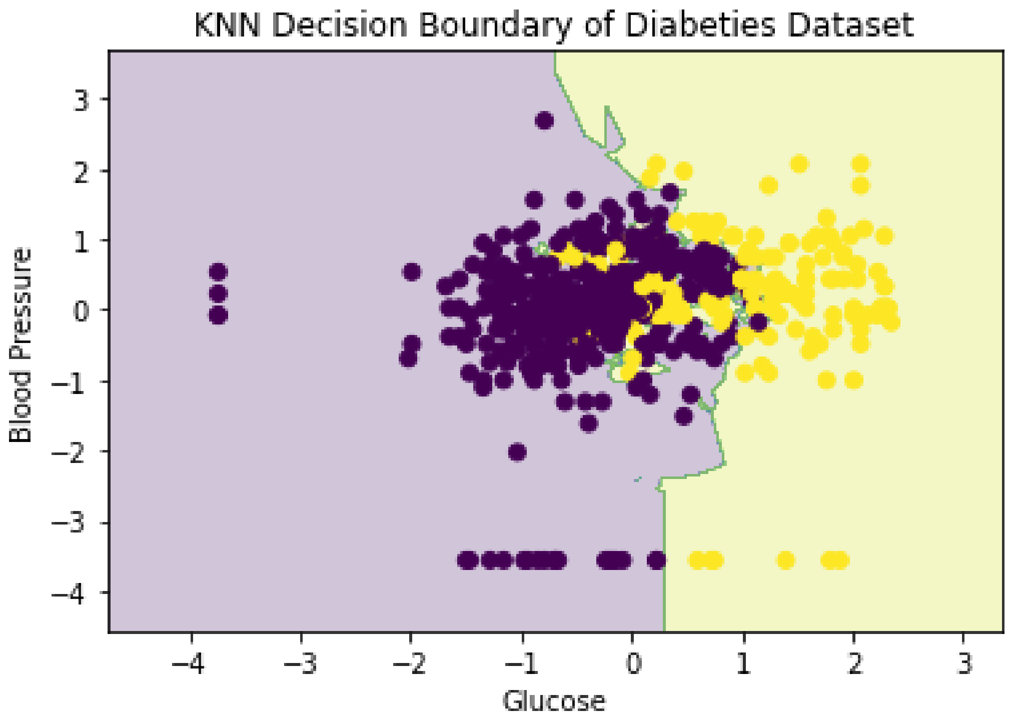
Output:-

```
 Recall [0.88990826 0.61621622]
Specificity [0.61621622 0.88990826]
Precision [0.8038674 0.76]
Negative Predictive Value [0.76      0.8038674]
False Positive Rate [0.38378378 0.11009174]
False Negative Rate [0.11009174 0.38378378]
False Discovery Rate [0.1961326 0.24]
Accuracry [0.79101562 0.79101562]
```




variety - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.6	50


KNN Decision Boundary of Diabeties Dataset
Blood Pressure
Glucose
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

Result:-

A python program for **implementing KNN** (Ensemble Learning) on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 8	**Decision Tree Classifier**
Date:	

Aim:-

To write a python program to implement Decision Tree Algorithm on real time dataset in machine learning.

Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file ‘diabetes.csv' and stored in the corresponding folder
4. Assign the input variables in X and the target variables in y.
5. Load the data set into the data frame using read_csv.
6. from sklearn.model_selection import train_test_split and partition the dataset into training and testing dataset.
7. Using sklearn.preprocessing library and import StandardScaler function to transform the dataset.
8. Fitting Decision Tree Induction algorithm to the Training set using sklearn.tree library and DecisionTreeClassifier method.
9. Get the confusion matrix for the dataset using sklearn.metrics library using confusion_matrix method
10. Calculate the performance metrics for the Decision Tree model.

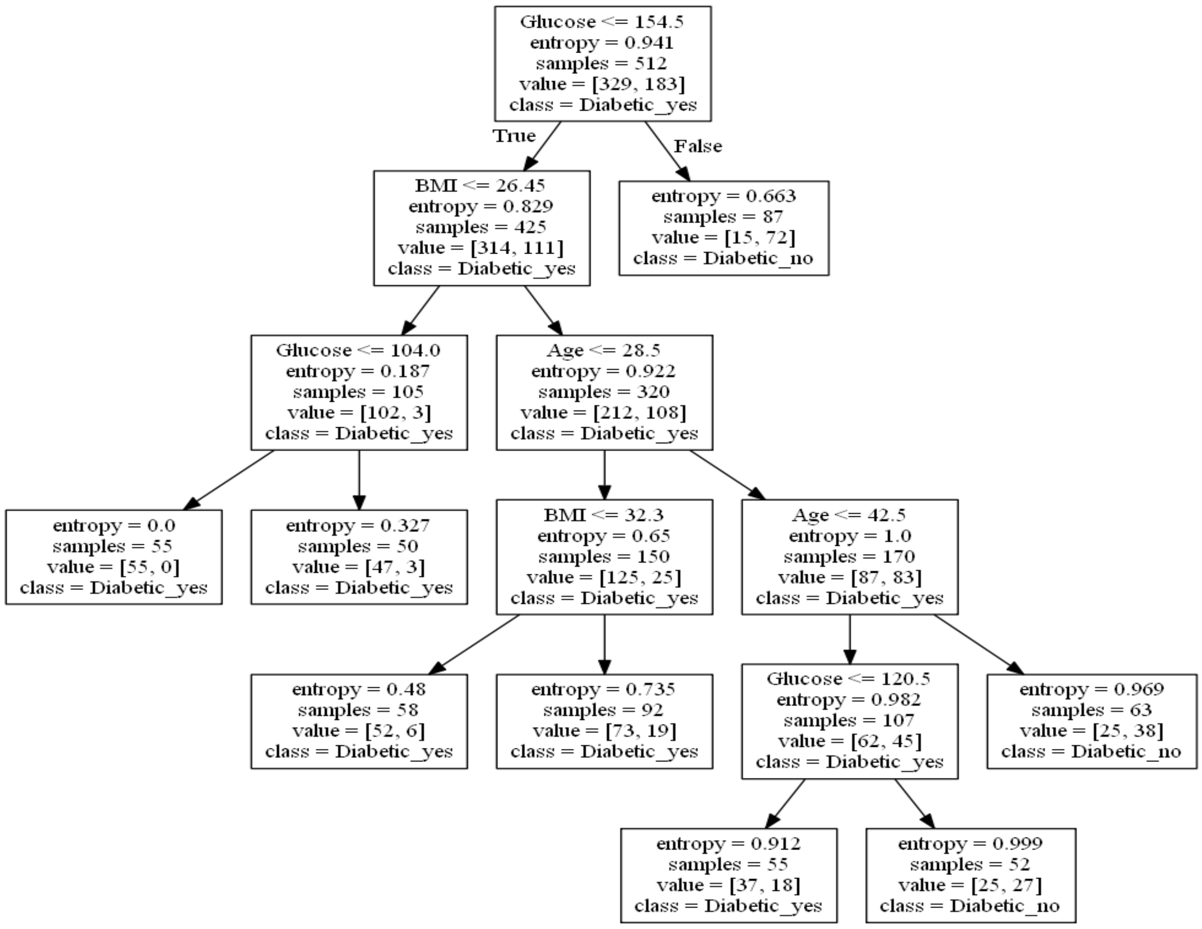
Program:-

```
# Decision Tree Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

#Importing the dataset
dataset = pd.read_csv('diabetes.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 8]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state =
0,max_depth=10,min_samples_leaf=50,max_features="log2",min_samples_split=50)
#classifier = DecisionTreeClassifier( random_state = 0)
classifier.fit(X_train, y_train)
# Predicting the Training set results
y_pred1 = classifier.predict(X_train)
# Making the Confusion Matrix for Training Set
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred1)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)
# Load libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from IPython.display import Image
from sklearn import tree
import pydotplus
# Create DOT data for Training Set
feature_cols=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','Diabete
sPedigreeFunction','Age']
dot_data = tree.export_graphviz(classifier, out_file=None,
                         feature_names=feature_cols,class_names=['Diabetic_yes','Diabetic_no'])

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png('diabeties.png')
```

```
# Show graph
Image(graph.create_png())

from sklearn import tree
```

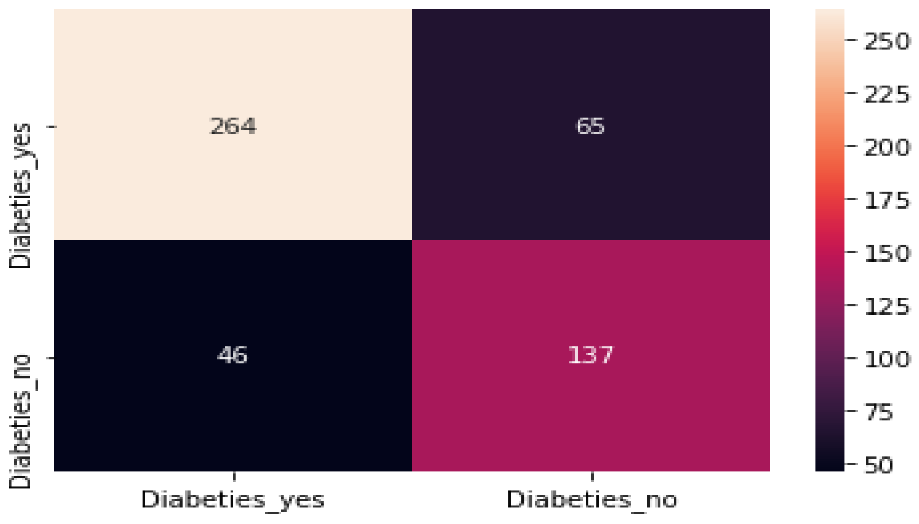
Output:-

Pictorial Representation of the Diabeties Decision Tree



Graphical Representation of the Diabeties Decision Tree

```
|--- Glucose <= 154.50
|   |--- BMI <= 26.45
|   |   |--- Glucose <= 104.00
|   |   |   |--- class: Diabetic_no
|   |   |--- Glucose >  104.00
|   |   |   |--- class: Diabetic_no
|   |--- BMI >  26.45
|   |   |--- Age <= 28.50
|   |   |   |--- BMI <= 32.30
|   |   |   |   |--- class: Diabetic_yes
|   |   |   |--- BMI >  32.30
|   |   |   |   |--- class: Diabetic_no
|   |   |--- Age >  28.50
|   |   |   |--- Age <= 42.50
|   |   |   |   |--- Glucose <= 120.50
|   |   |   |   |   |--- class: Diabetic_no
|   |   |   |   |--- Glucose >  120.50
|   |   |   |   |   |--- class: Diabetic_yes
|   |   |   |--- Age >  42.50
|   |   |   |   |--- class: Diabetic_yes
|--- Glucose >  154.50
|   |--- class: Diabetic_yes
```



```
Recall [0.90214067 0.69189189]
Specificity [0.69189189 0.90214067]
Precision [0.83806818 0.8       ]
Negative Predictive Value [0.8        0.83806818]
False Positive Rate [0.30810811 0.09785933]
False Negative Rate [0.09785933 0.30810811]
False Discovery Rate [0.16193182 0.2       ]
Accuracry [0.82617188 0.82617188]
```

Result:-

A python program for data pre-processing technique (Standardization) on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 9	Random Forest Algorithm (Ensemble Learning)
Date:	

Aim:-

To write a python program to using **Ensemble Learning Technique and implement Random Forest Algorithm.**

Procedure:-

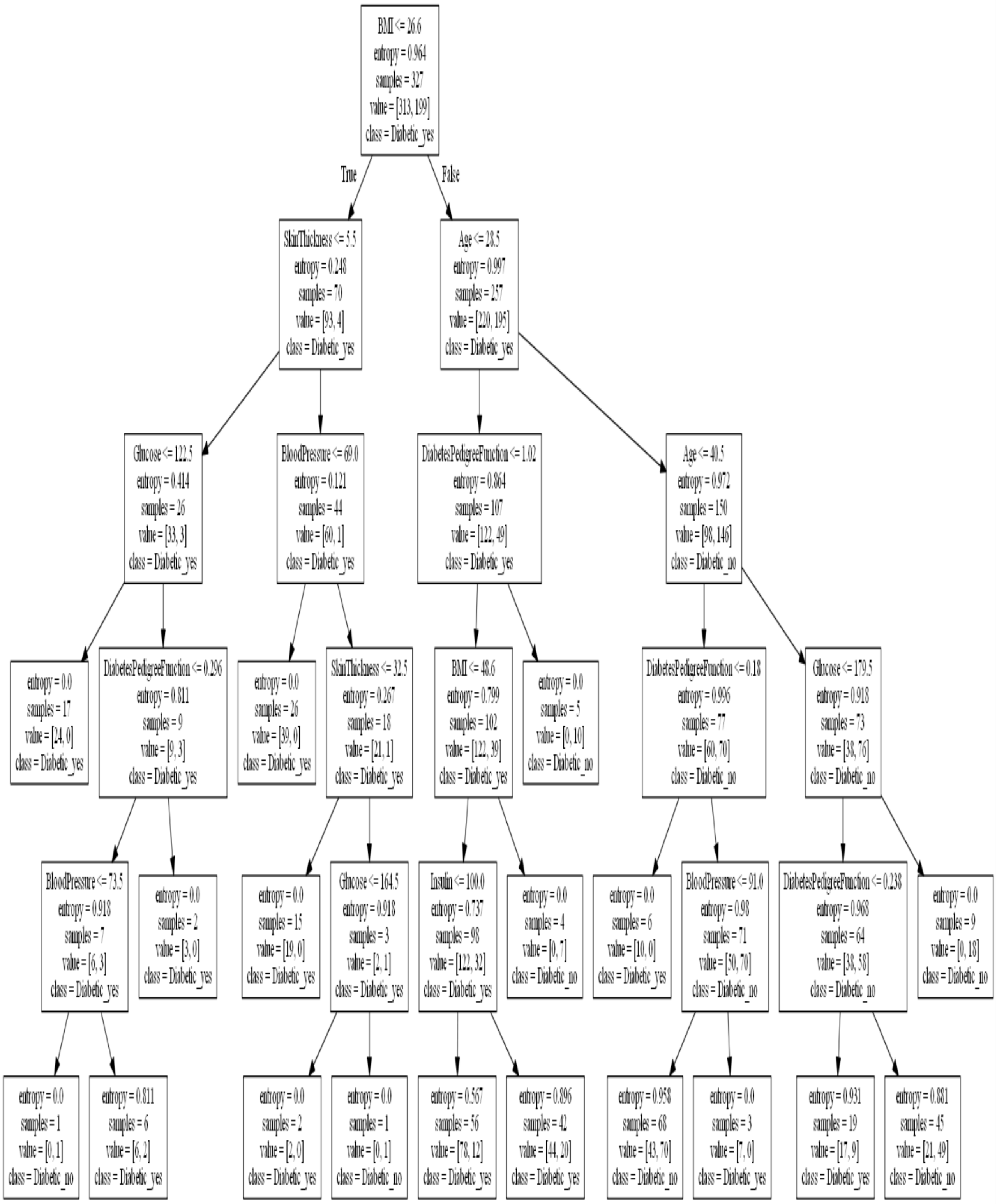
1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file ‘diabetes.csv' and stored in the corresponding folder
4. Assign the input variables in X and the target variables in y.
5. Load the data set into the data frame using read_csv.
6. from sklearn.model_selection import train_test_split and partition the dataset into training and testing dataset.
7. Using sklearn.preprocessing library and import StandardScaler function to transform the dataset.
8. Fitting Ensemble Learning algorithm to the Training set using sklearn.ensemble library and using the RandomForestClassifier
9. Get the confusion matrix for the dataset using sklearn.metrics library using confusion_matrix method
10. Calculate the performance metrics for the Random Forest model.

Program:-

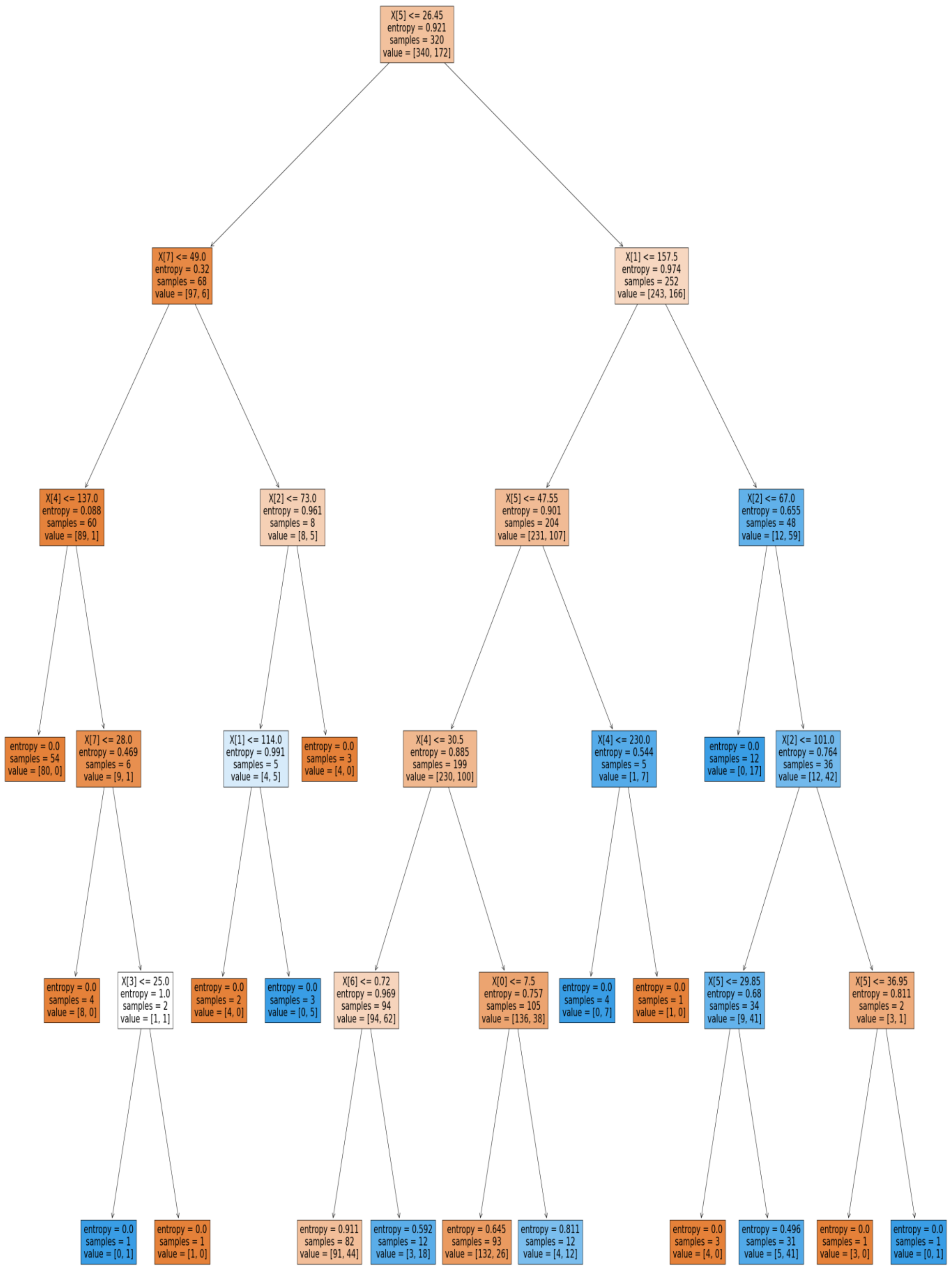
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Importing the dataset
dataset = pd.read_csv('diabetes.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 8]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 5, criterion = 'entropy', random_state = 0,
max_depth=5)
classifier.fit(X_train, y_train)

# Predicting the Training set results
y_pred = classifier.predict(X_train)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)

#Visulaizing the Random forest
from sklearn import tree
plt.figure(figsize=(50,50))
for i in range(len(classifier.estimators_)):
    print(tree.plot_tree(classifier.estimators_[i]))

#Calculating Performance Metrics for Training Set
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
```

```
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print("Recall",TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print("Specificity",TNR)
# Precision or positive predictive value
PPV = TP/(TP+FP)
print("Precision",PPV)
# Negative predictive value
NPV = TN/(TN+FN)
print("Negative Predictive Value",NPV)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print("False Positive Rate",FPR)
# False negative rate
FNR = FN/(TP+FN)
print("False Negative Rate",FNR)
# False discovery rate
FDR = FP/(TP+FP)
print("False Discovery Rate",FDR)
# Overall accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
print("Accuracry",ACC)
```

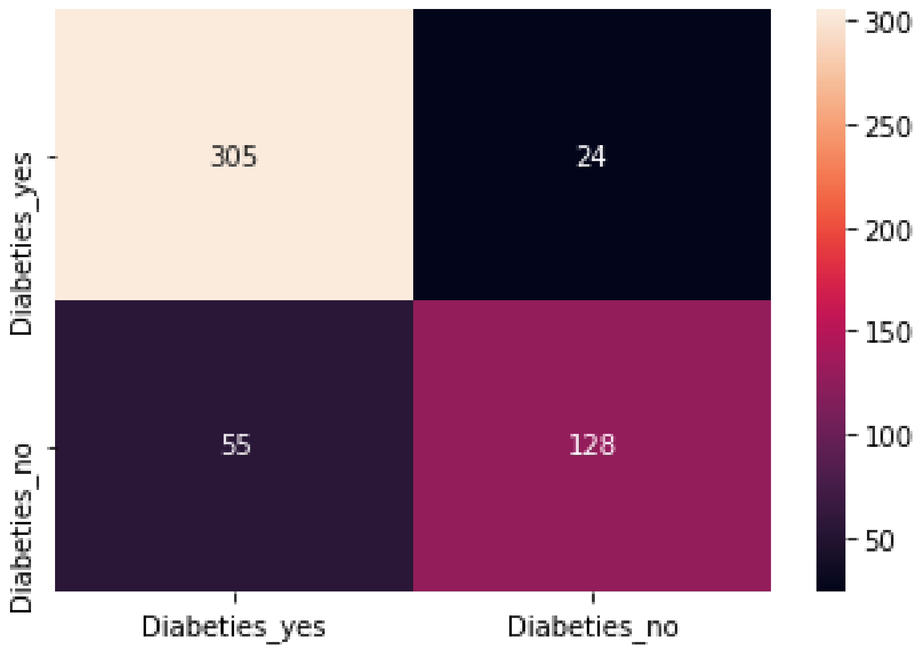
Output:-



BMI <= 26.6 entropy = 0.964 samples = 327 value = [313, 199] class = Diabetic_yes
True
False
SkinThickness <= 5.5 entropy = 0.248 samples = 70 value = [93, 4] class = Diabetic_yes
Age <= 28.5 entropy = 0.997 samples = 257 value = [220, 195] class = Diabetic_yes
Glucose <= 122.5 entropy = 0.414 samples = 26 value = [33, 3] class = Diabetic_yes
BloodPressure <= 69.0 entropy = 0.121 samples = 44 value = [60, 1] class = Diabetic_yes
DiabetesPedigreeFunction <= 1.02 entropy = 0.864 samples = 107 value = [122, 49] class = Diabetic_yes
Age <= 40.5 entropy = 0.972 samples = 150 value = [98, 146] class = Diabetic_no
entropy = 0.0 samples = 17 value = [24, 0] class = Diabetic_yes
DiabetesPedigreeFunction <= 0.296 entropy = 0.811 samples = 9 value = [9, 3] class = Diabetic_yes
entropy = 0.0 samples = 26 value = [39, 0] class = Diabetic_yes
SkinThickness <= 32.5 entropy = 0.267 samples = 18 value = [21, 1] class = Diabetic_yes
BMI <= 48.6 entropy = 0.799 samples = 102 value = [122, 39] class = Diabetic_yes
entropy = 0.0 samples = 5 value = [0, 10] class = Diabetic_no
DiabetesPedigreeFunction <= 0.18 entropy = 0.996 samples = 77 value = [60, 70] class = Diabetic_no
Glucose <= 179.5 entropy = 0.918 samples = 73 value = [38, 76] class = Diabetic_no
BloodPressure <= 73.5 entropy = 0.918 samples = 7 value = [6, 3] class = Diabetic_yes
entropy = 0.0 samples = 2 value = [3, 0] class = Diabetic_yes
entropy = 0.0 samples = 15 value = [19, 0] class = Diabetic_yes
Glucose <= 164.5 entropy = 0.918 samples = 3 value = [2, 1] class = Diabetic_yes
Insulin <= 100.0 entropy = 0.737 samples = 98 value = [122, 32] class = Diabetic_yes
entropy = 0.0 samples = 4 value = [0, 7] class = Diabetic_no
entropy = 0.0 samples = 6 value = [10, 0] class = Diabetic_yes
BloodPressure <= 91.0 entropy = 0.98 samples = 71 value = [50, 70] class = Diabetic_no
DiabetesPedigreeFunction <= 0.238 entropy = 0.968 samples = 64 value = [38, 58] class = Diabetic_no
entropy = 0.0 samples = 9 value = [0, 18] class = Diabetic_no
entropy = 0.0 samples = 1 value = [0, 1] class = Diabetic_no
entropy = 0.811 samples = 6 value = [6, 2] class = Diabetic_yes
entropy = 0.0 samples = 2 value = [2, 0] class = Diabetic_yes
entropy = 0.0 samples = 1 value = [0, 1] class = Diabetic_no
entropy = 0.567 samples = 56 value = [78, 12] class = Diabetic_yes
entropy = 0.896 samples = 42 value = [44, 20] class = Diabetic_yes
entropy = 0.958 samples = 68 value = [43, 70] class = Diabetic_no
entropy = 0.0 samples = 3 value = [7, 0] class = Diabetic_yes
entropy = 0.931 samples = 19 value = [17, 9] class = Diabetic_yes
entropy = 0.881 samples = 45 value = [21, 49] class = Diabetic_no


X[5] <= 26.45 entropy = 0.921 samples = 320 value = [340, 172]
X[7] <= 49.0 entropy = 0.32 samples = 68 value = [97, 6]
X[1] <= 157.5 entropy = 0.974 samples = 252 value = [243, 166]
X[4] <= 137.0 entropy = 0.088 samples = 60 value = [89, 1]
X[2] <= 73.0 entropy = 0.961 samples = 8 value = [8, 5]
X[5] <= 47.55 entropy = 0.901 samples = 204 value = [231, 107]
X[2] <= 67.0 entropy = 0.655 samples = 48 value = [12, 59]
entropy = 0.0 samples = 54 value = [80, 0]
X[7] <= 28.0 entropy = 0.469 samples = 6 value = [9, 1]
X[1] <= 114.0 entropy = 0.991 samples = 5 value = [4, 5]
entropy = 0.0 samples = 3 value = [4, 0]
X[4] <= 30.5 entropy = 0.885 samples = 199 value = [230, 100]
X[4] <= 230.0 entropy = 0.544 samples = 5 value = [1, 7]
entropy = 0.0 samples = 12 value = [0, 17]
X[2] <= 101.0 entropy = 0.764 samples = 36 value = [12, 42]
entropy = 0.0 samples = 4 value = [8, 0]
X[3] <= 25.0 entropy = 1.0 samples = 2 value = [1, 1]
entropy = 0.0 samples = 2 value = [4, 0]
entropy = 0.0 samples = 3 value = [0, 5]
X[6] <= 0.72 entropy = 0.969 samples = 94 value = [94, 62]
X[0] <= 7.5 entropy = 0.757 samples = 105 value = [136, 38]
entropy = 0.0 samples = 4 value = [0, 7]
entropy = 0.0 samples = 1 value = [1, 0]
X[5] <= 29.85 entropy = 0.68 samples = 34 value = [9, 41]
X[5] <= 36.95 entropy = 0.811 samples = 2 value = [3, 1]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.911 samples = 82 value = [91, 44]
entropy = 0.592 samples = 12 value = [3, 18]
entropy = 0.645 samples = 93 value = [132, 26]
entropy = 0.811 samples = 12 value = [4, 12]
entropy = 0.0 samples = 3 value = [4, 0]
entropy = 0.496 samples = 31 value = [5, 41]
entropy = 0.0 samples = 1 value = [3, 0]
entropy = 0.0 samples = 1 value = [0, 1]

```
Performance Metrics for Random Forest Classifier
Recall [0.92705167 0.69945355]
Specificity [0.69945355 0.92705167]
Precision [0.84722222 0.84210526]
Negative Predictive Value [0.84210526 0.84722222]
False Positive Rate [0.30054645 0.07294833]
False Negative Rate [0.07294833 0.30054645]
False Discovery Rate [0.15277778 0.15789474]
Accuracy [0.84570312 0.84570312]
```



Result:-

A python program for Ensemble Learning using Random Forest on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 10	Support Vector Machine
Date:	

Aim:-

To write a python program to implement **Support Vector Machine using Machine Learning**.

Procedure:-

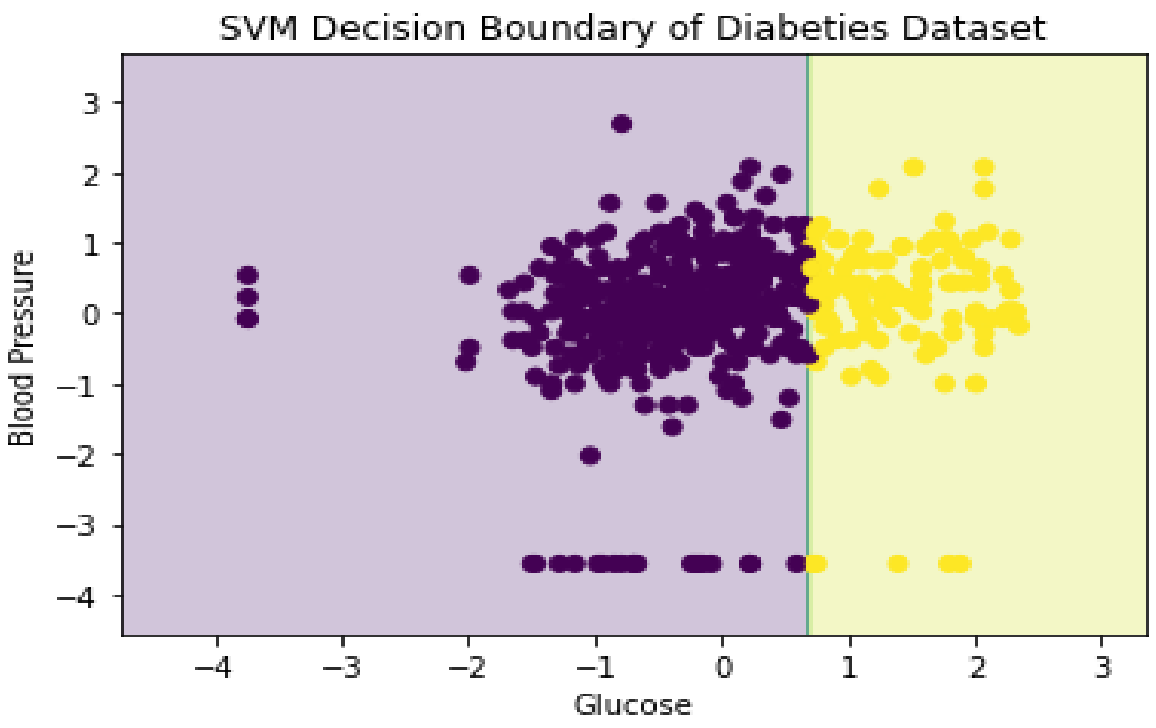
1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file ‘diabetes.csv' and stored in the corresponding folder
4. Assign the input variables in X and the target variables in y.
5. Load the data set into the data frame using read_csv.
6. from sklearn.model_selection import train_test_split and partition the dataset into training and testing dataset.
7. Using sklearn.preprocessing library and import StandardScaler function to transform the dataset.
8. Fitting Ensemble Learning algorithm to the Training set using sklearn.svm library and using the SVC method.
9. Get the confusion matrix for the dataset using sklearn.metrics library using confusion_matrix method
10. Calculate the performance metrics for the Support Vector model.

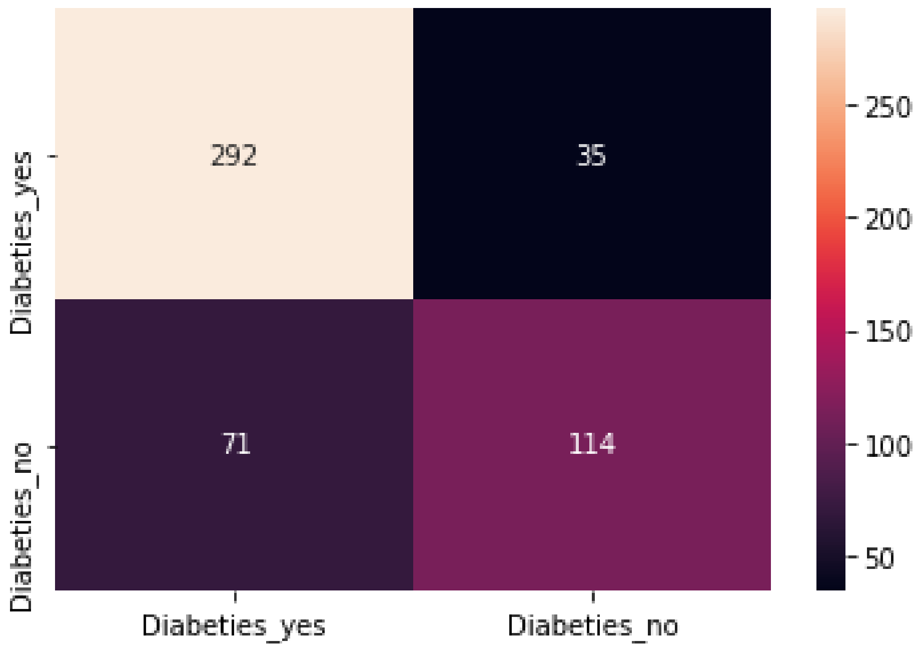
Program:-

```
# Naive Bayes Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns;

#Importing the dataset
dataset = pd.read_csv('diabetes.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 8]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 2)



# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 3)
classifier.fit(X_train, y_train)

# Predicting the Training set results
y_pred = classifier.predict(X_train)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred)

#Heat map of a confusion matrix
import seaborn as sns
```

```
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)

#create an empty data frame that we have to predict
variety=pd.DataFrame()
variety['Pregnancies']=[6]
variety['Glucose']=[148]
variety['BloodPressure']=[72]
variety['SkinThickness']=[35]
variety['Insulin']=[0]
variety['BMI']=[33.6]
variety['DiabetesPedigreeFunction']=[0.627]
variety['Age']=[50]
print(variety)

y_pred1=classifier.predict(variety)
print("the Outcome of the Patient is:")
print(y_pred1)
```

Output:-


SVM Decision Boundary of Diabeties Dataset
Blood Pressure
Glucose
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


292
35
71
114
Diabeties_yes
Diabeties_no
Diabeties_yes
Diabeties_no
250
200
150
100
50

```
Performance Metrics for Support Vector Machine

Recall [0.89296636 0.61621622]
Specificity [0.61621622 0.89296636]
Precision [0.80440771 0.76510067]
Negative Predictive Value [0.76510067 0.80440771]
False Positive Rate [0.38378378 0.10703364]
False Negative Rate [0.10703364 0.38378378]
False Discovery Rate [0.19559229 0.23489933]
Accuracry [0.79296875 0.79296875]
```

Result:-

A python program for Support Vector Machine on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 11	**K-Means Clustering**
Date:	

Aim:-

To write a python program to implement **K-Means Clustering in Machine Learning.**

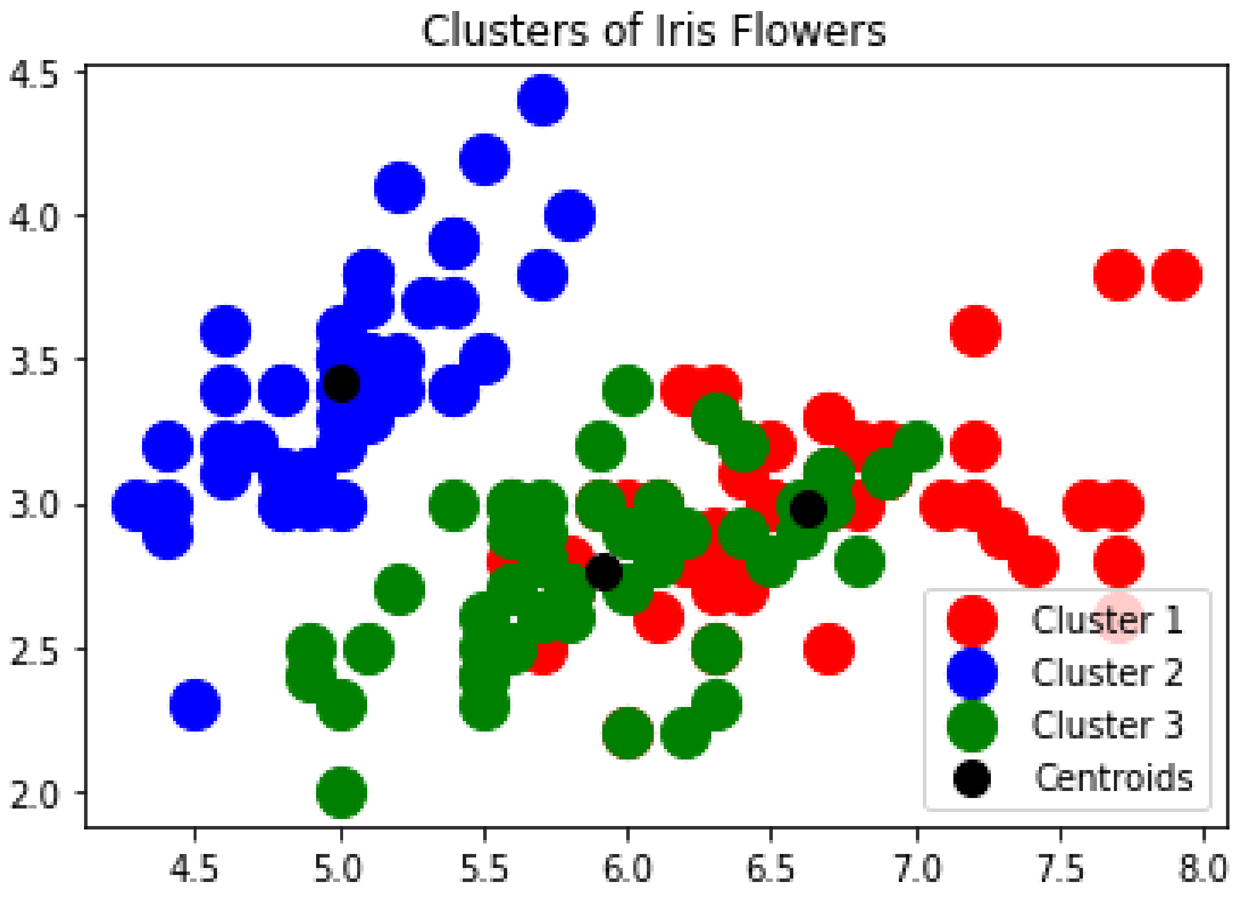
Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib from python library
3. Download the csv file 'Iris.csv' and stored in the corresponding folder
4. Assign all the input variables in X.
5. Load the data set into the data frame using read_csv.
6. Import the library sklearn.cluster and use the method KMeans to implement K-Means clustering for the Iris Dataset.
7. Visualize the clusters using Scatter plot method.

Program:-

```
# K-Means Clustering
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import re
colnames=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Species']
# Importing the dataset
dataset =
pd.read_csv('Iris1.csv',names=colnam
es,dtype=float)
X = dataset.iloc[:,:].values

dataset.dtypes

# Using the elbow method to find the
optimal number of clusters
from sklearn.cluster import KMeans
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 200, random_state = 42)
y_kmeans = kmeans.fit_predict(X)
ymeans=kmeans.fit(X)
print(kmeans.cluster_centers_)
#display cluster centers


# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 200, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 200, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 200, c = 'green', label = 'Cluster
3')
plt.scatter(kmeans.cluster_centers_[:,   0], kmeans.cluster_centers_[:,1],s = 100, c = 'black',
label = 'Centroids')   #plotting the centroids of the clusters
plt.legend()
plt.title('Clusters of Iris Flowers')
plt.legend()
plt.show()
```

OUTPUT:


Clusters of Iris Flowers
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Cluster 1
Cluster 2
Cluster 3
Centroids

Result:-

A python program for K-Means clustering on real time dataset in machine learning has been implemented and executed successfully.

Ex.No: 12	Implement a Feed-Forward Network in TensorFlow/Keras.
Date:	

Aim:-

To write a python program to implement Artificial Neural Network in machine learning.

Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib, tensorflow from python library
3. Download the csv file ‘paddy.csv' and stored in the corresponding folder
4. Assign all the input variables in X.
5. Load the data set into the data frame using read_csv.
6. Add the input layer to the Artificial Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
7. Add the hidden layer to the Artificial Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
8. Add the output layer to the Artificial Neural Network using optimizer as adam, loss function as binary_crossentropy, and metrics as accuracy.
9. Visualize Neural Network using graphviz library function
10. Analyse the performance of the ANN using the performance metrics.

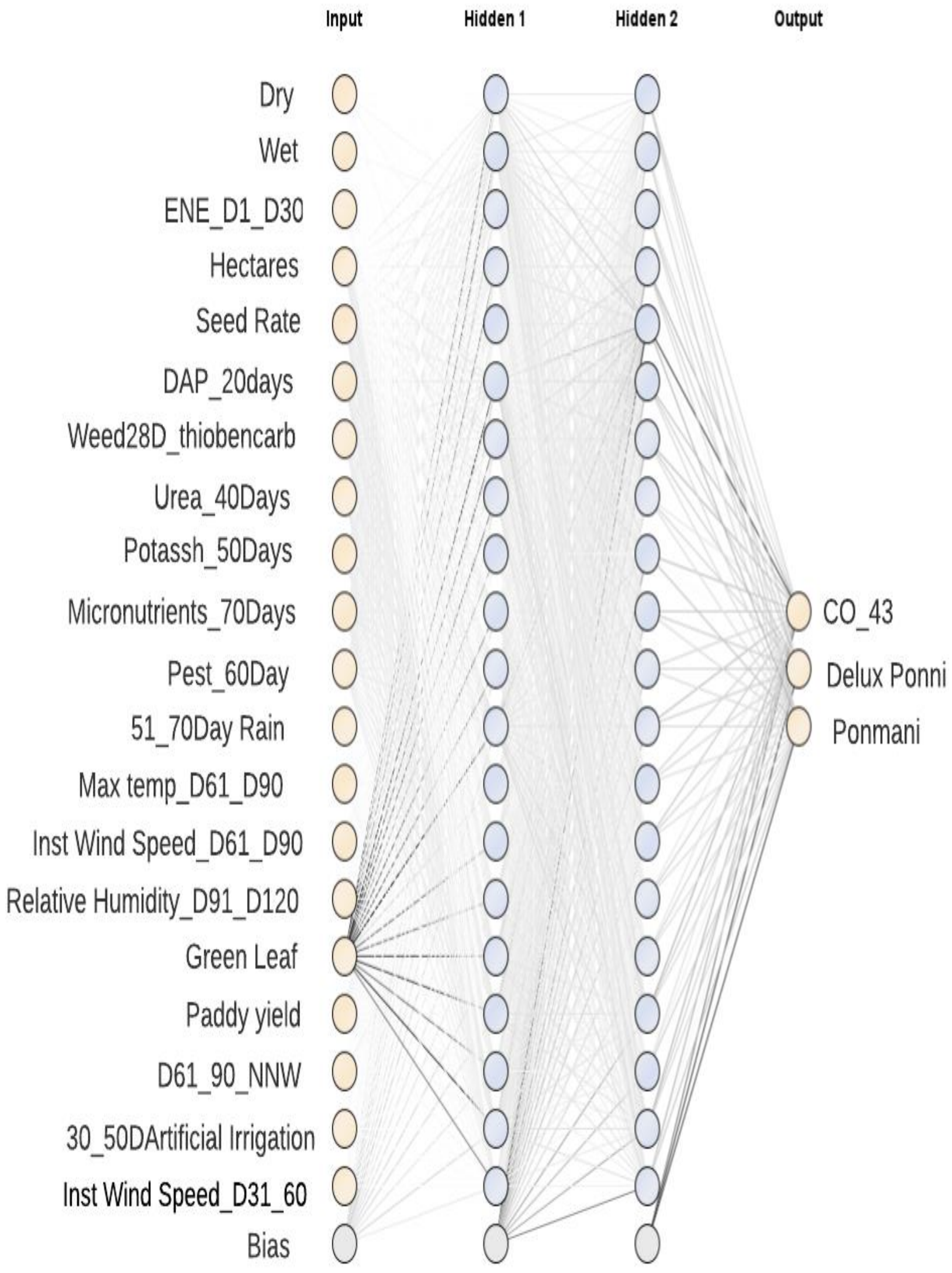
Program:-

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

#Importing the dataset
dataset = pd.read_csv('paddy.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 21]

# Splitting the dataset into the Training set and Test set
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)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units=6, activation='relu', kernel_initializer='uniform', input_dim = 8))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

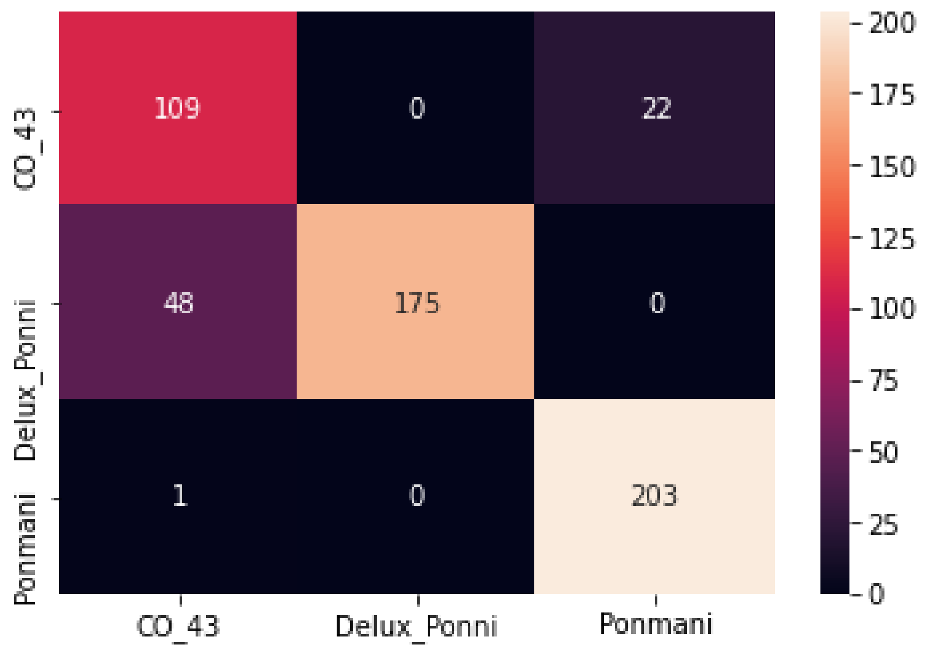
```
# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)

#Visualizing the neural network
from ann_visualizer.visualize import ann_viz
from graphviz import Source
ann_viz(classifier,title='Neural Network')
graph_source=Source.from_file('network.gv')

print(graph_source.source)
classifier.get_weights()

#Predicting the input record
#create an empty data frame that we have to predict
variety=pd.DataFrame()
variety['Pregnancies']=[6]
variety['Glucose']=[148]
variety['BloodPressure']=[72]
variety['SkinThickness']=[35]
variety['Insulin']=[0]
variety['BMI']=[33.6]
variety['DiabetesPedigreeFunction']=[0.627]
variety['Age']=[50]
print(variety)

y_pred1=classifier.predict(variety)
print("the Outcome of the Patient is:")
print(y_pred1)

#Calculating Performance Metrics for Training Set
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
```

```
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print("Recall",TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print("Specificity",TNR)
# Precision or positive predictive value
PPV = TP/(TP+FP)
print("Precision",PPV)
# Negative predictive value
NPV = TN/(TN+FN)
print("Negative Predictive Value",NPV)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print("False Positive Rate",FPR)
# False negative rate
FNR = FN/(TP+FN)
print("False Negative Rate",FNR)
# False discovery rate
FDR = FP/(TP+FP)
print("False Discovery Rate",FDR)
# Overall accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
print("Accuracry",ACC)
```

Output:-



Input
Hidden 1
Hidden 2
Output
Dry
Wet
ENE_D1_D30
Hectares
Seed Rate
DAP_20days
Weed28D_thiobencarb
Urea_40Days
Potassh_50Days
Micronutrients_70Days
Pest_60Day
51_70Day Rain
Max temp_D61_D90
Inst Wind Speed_D61_D90
Relative Humidity_D91_D120
Green Leaf
Paddy yield
D61_90_NNW
30_50DArtificial Irrigation
Inst Wind Speed_D31_60
Bias
CO_43
Delux Ponni
Ponmani



Performance Metrics for the ANN

Recall [0.87850467 0.61702128]
Specificity [0.61702128 0.87850467]
Precision [0.83928571 0.69047619]
Negative Predictive Value [0.69047619 0.83928571]
False Positive Rate [0.38297872 0.12149533]
False Negative Rate [0.12149533 0.38297872]
False Discovery Rate [0.16071429 0.30952381]
Accuracry [0.7987013 0.7987013]

65

Result:-

A python program to implement Artificial Neural Network in machine learning has been implemented and executed successfully.

Ex.No: 13	**Improve the Deep learning model by fine tuning hyper parameters.**
Date:	

Aim:-

To write a python program to implement Artificial Neural Network in machine learning.

Procedure:-

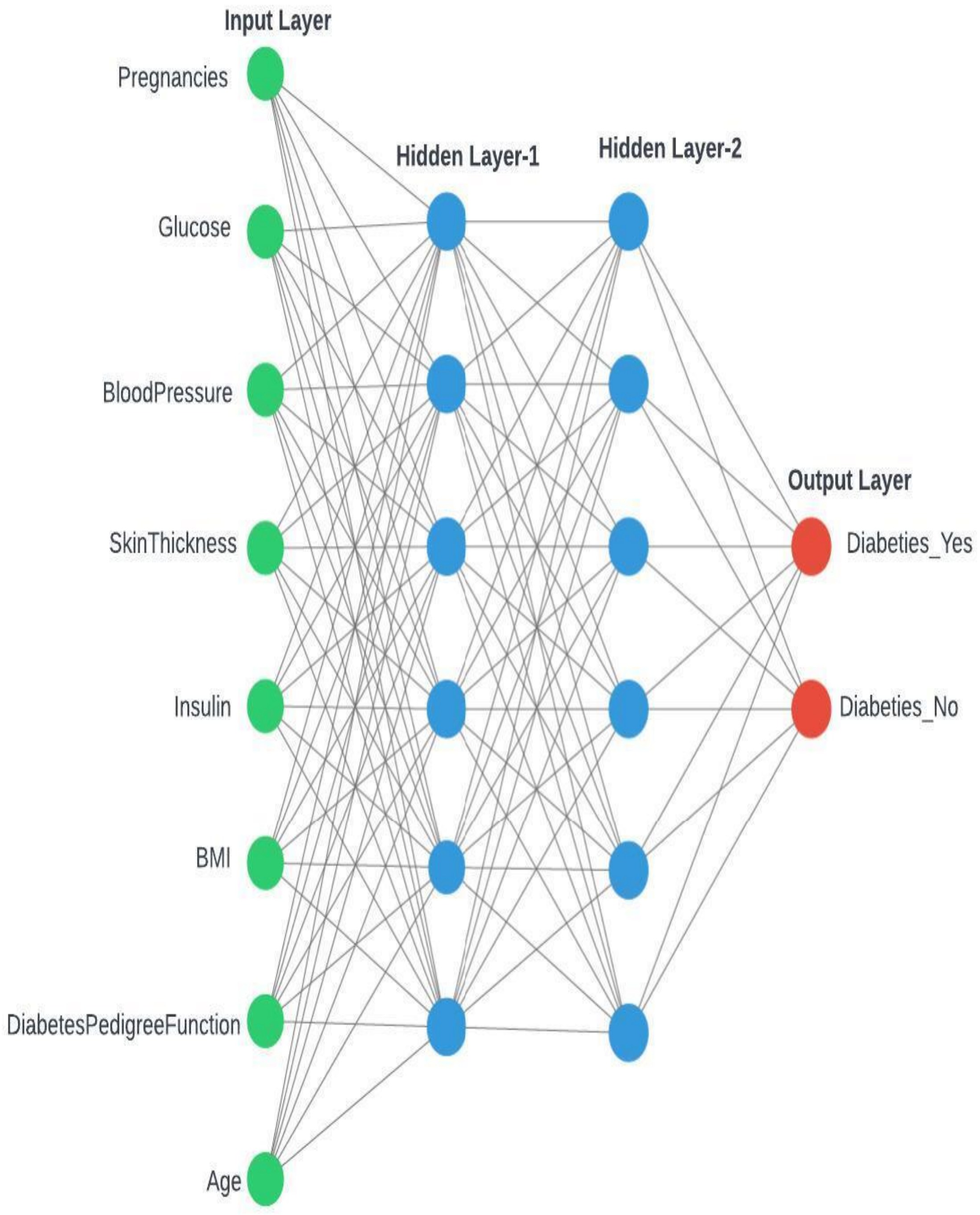
1. Import read_csv from the package pandas.
2. Import numpy, matplotlib, tensorflow from python library
3. Download the csv file ‘Sugarcane.csv' and stored in the corresponding folder
4. Assign all the input variables in X.
5. Load the data set into the data frame using read_csv.
6. Add the input layer to the Artificial Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
7. Add the hidden layer to the Artificial Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
8. Add the output layer to the Artificial Neural Network using optimizer as adam, loss function as binary_crossentropy, and metrics as accuracy.
9. Visualize Neural Network using graphviz library function
10. Analyse the performance of the ANN using the performance metrics.

Program

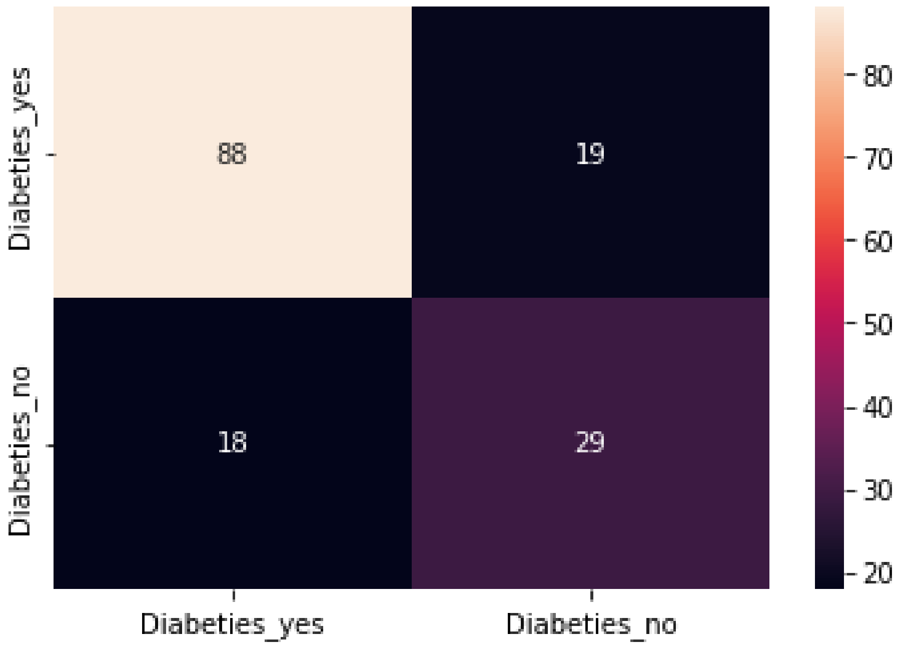
```
# Part 1 - Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

#Importing the dataset
dataset = pd.read_csv('diabetes.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 8]

# Splitting the dataset into the Training set and Test set
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 keras.utils import to_categorical
y_train = to_categorical(y_train, 2)
y_test = to_categorical(y_test, 2)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD, Adam, Adadelta, RMSprop

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units=6, activation='relu', kernel_initializer='uniform', input_dim = 8))

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 2, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
```

```
classifier.compile(Adam(learning_rate = 0.01), "categorical_crossentropy", metrics =
["accuracy"])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Predicting the Test set results
y_pred_probabilities = classifier.predict(X_test)
y_pred_class = y_pred_probabilities.argmax(axis=-1)
from sklearn.metrics import confusion_matrix
y_pred = classifier.predict(X_test)
y_test_class = np.argmax(y_test, axis=1)
cm=confusion_matrix(y_test_class, y_pred_class)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['Diabeties_yes','Diabeties_no'],yticklabels=['Diabetie
s_yes','Diabeties_no'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)

#Visualizing the neural network
from ann_visualizer.visualize import ann_viz
from graphviz import Source
ann_viz(classifier,title='Neural Network Diabeties')
graph_source=Source.from_file('network.gv')

print(graph_source.source)
classifier.get_weights()

#Predicting the input record
#create an empty data frame that we have to predict
variety=pd.DataFrame()
variety['Pregnancies']=[6]
variety['Glucose']=[148]
variety['BloodPressure']=[72]
variety['SkinThickness']=[35]
variety['Insulin']=[0]
variety['BMI']=[33.6]
variety['DiabetesPedigreeFunction']=[0.627]
variety['Age']=[50]
print(variety)

y_pred1=classifier.predict(variety)
if y_pred1[0][0]>0.5:
    print("the Outcome of the Patient is:Diabeties")
else:
    print("The patient has no Diabeties")
print(y_pred1)

#Calculating Performance Metrics for Training Set
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print("Recall",TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print("Specificity",TNR)
# Precision or positive predictive value
PPV = TP/(TP+FP)
print("Precision",PPV)
# Negative predictive value
NPV = TN/(TN+FN)
print("Negative Predictive Value",NPV)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print("False Positive Rate",FPR)
# False negative rate
FNR = FN/(TP+FN)
print("False Negative Rate",FNR)
# False discovery rate
FDR = FP/(TP+FP)
print("False Discovery Rate",FDR)
# Overall accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
print("Accuracry",ACC)
```

Output:


Input Layer
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Hidden Layer-1
Hidden Layer-2
Output Layer
Diabeties_Yes
Diabeties_No

Performance Metrics for the ANN



```
Recall [0.82242991 0.61702128]
Specificity [0.61702128 0.82242991]
Precision [0.83018868 0.60416667]
Negative Predictive Value [0.60416667 0.83018868]
False Positive Rate [0.38297872 0.17757009]
False Negative Rate [0.17757009 0.38297872]
False Discovery Rate [0.16981132 0.39583333]
Accuracy [0.75974026 0.75974026]
1/1 [==============================] - 0s 31ms/step
The patient has no Diabeties
```

Result:-

A python program to implement **Deep learning model by fine tuning hyper parameters** in machine learning has been implemented and executed successfully.

Ex.No: 14	**Implement an Image Classifier using CNN in TensorFlow/Keras**
Date:	

Aim:-

To write a python program to implement an Image Classifier using CNN in Tensorflow and Keras.

Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib, tensorflow from python library
3. Download the ‘mallatai_300' dataset and stored in the corresponding folder
4. Using ImageDataGenerator classify the images into training set and testing set.
5. Add the input layer to the Convolutional Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
6. Add the Pooling layer to the Convolution layer to the Convolutional Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
7. Add the Second Convolutional layer to the Convolutional Neural Network using optimizer as adam, loss function as categorical_crossentropy, and metrics as accuracy.
8. Visualize Neural Network using graphviz library function
9. Analyse the performance of the CNN using the performance metrics.

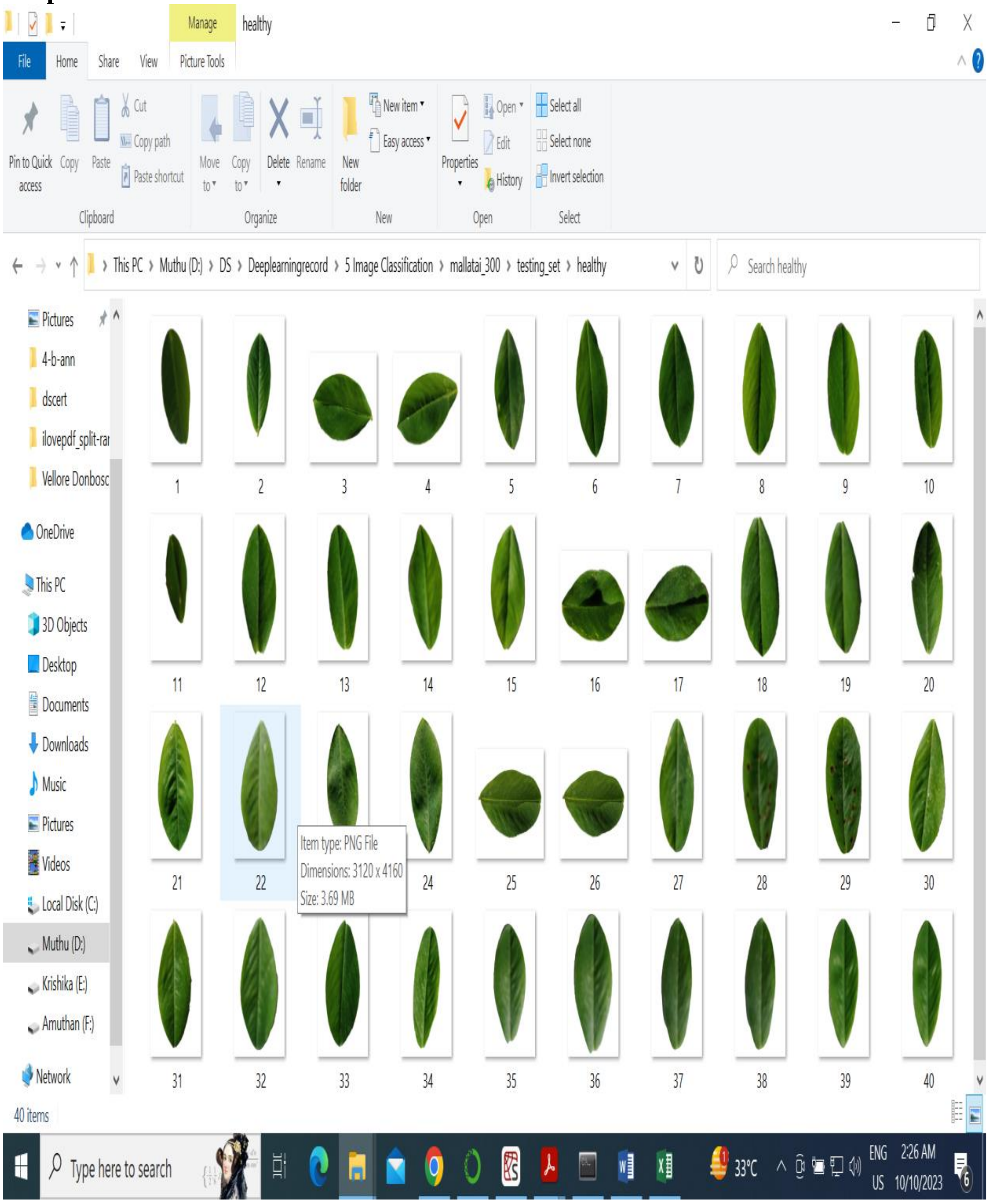
Program

```
# Convolutional Neural Network

# Importing the libraries
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
tf.__version__
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
# Part 1 - Data Preprocessing

# Preprocessing the Training set
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

# Preprocessing the Test set
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

#Visual Representation of ImageDataGenerator

# Part 2 - Building the CNN

# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Step 1 - Convolution
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64,
64, 3]))

# Step 2 - Pooling
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

# Step 4 - Full Connection
```

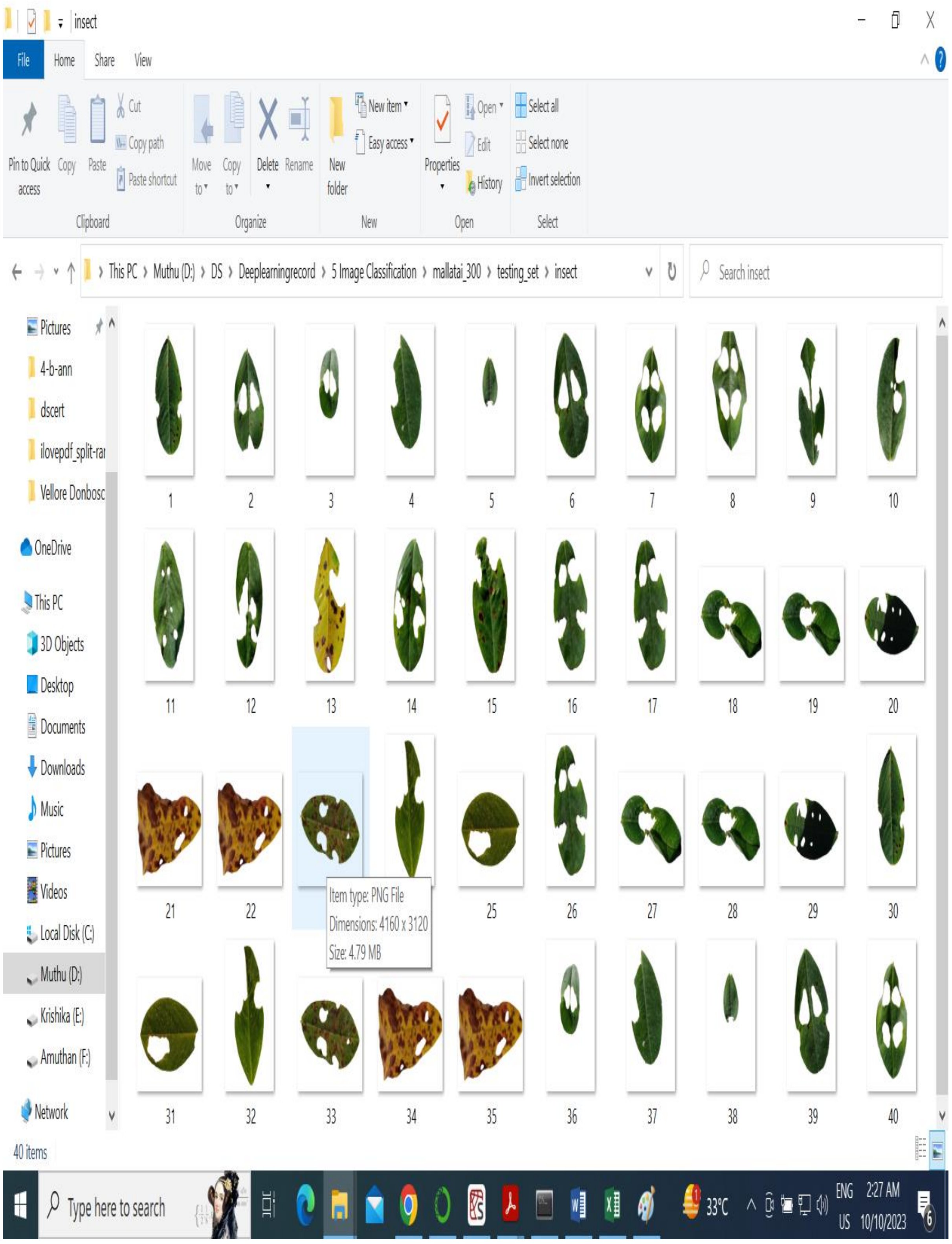
```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# Step 5 - Output Layer
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Part 3 - Training the CNN

# Compiling the CNN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Training the CNN on the Training set and evaluating it on the Test set
cnn.fit(x = training_set, validation_data = test_set, epochs = 25)

# Part 4 - Making a single prediction

import numpy as np
import keras.utils as image
test_image = image.load_img('dataset/single/cord2.jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
print(result)
a=training_set.class_indices
print(a)
if result[0] == 0:
    prediction = 'cats'
elif result[0]==1:
    prediction = 'dogs'
else:
    prediction='Not defined'
print(prediction)

#Coding for confusion matrix for training set
import numpy as np
from sklearn import metrics
from sklearn.metrics import confusion_matrix
print('Confusion Matrix')
#Coding for confusion matrix for training set
print('Confusion Matrix')
preds=np.round(cnn.predict(training_set),0)
cm=confusion_matrix(training_set.classes,preds)
print(cm)
labels=['Cats','Dogs']
classification_metrics=metrics.classification_report(training_set.classes,preds,target_names=l
abels)
print(classification_metrics)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,annot=True,xticklabels=labels,yticklabels=labels,fmt='d')
```

```
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print("Recall",TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print("Specificity",TNR)
# Precision or positive predictive value
PPV = TP/(TP+FP)
print("Precision",PPV)
# Negative predictive value
NPV = TN/(TN+FN)
print("Negative Predictive Value",NPV)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print("False Positive Rate",FPR)
# False negative rate
FNR = FN/(TP+FN)
print("False Negative Rate",FNR)
# False discovery rate
FDR = FP/(TP+FP)
print("False Discovery Rate",FDR)
# Overall accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
print("Accuracry",ACC)
```

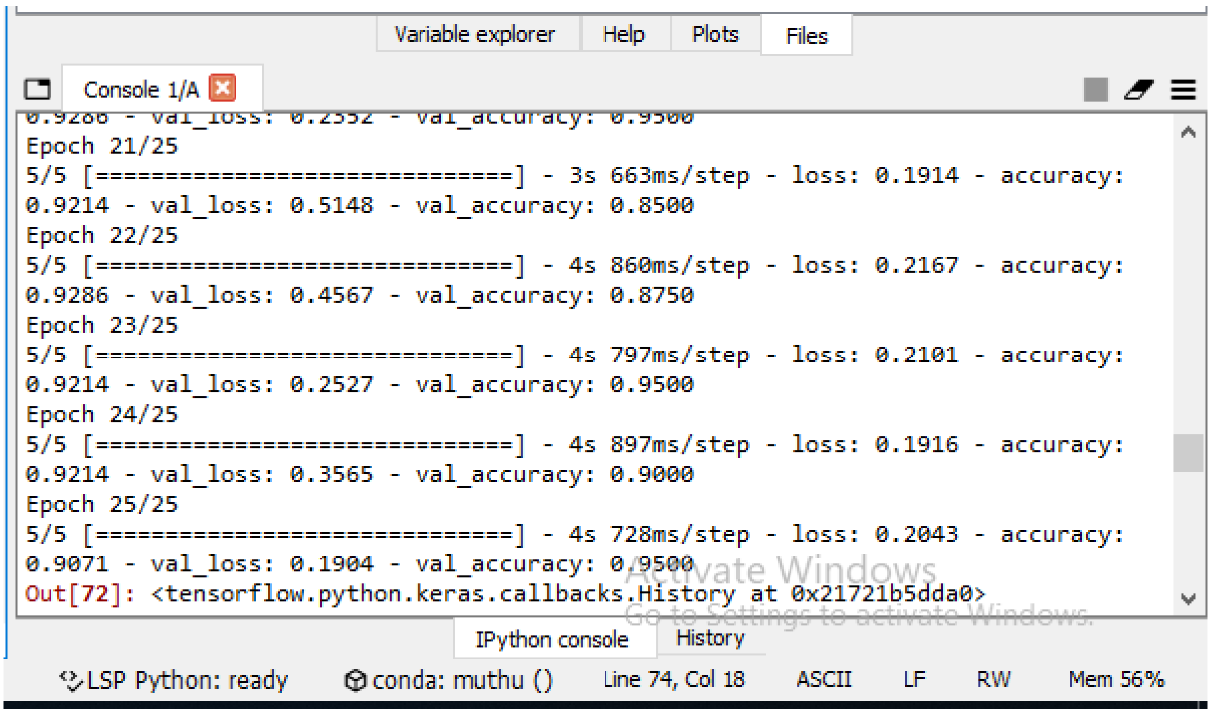
Output



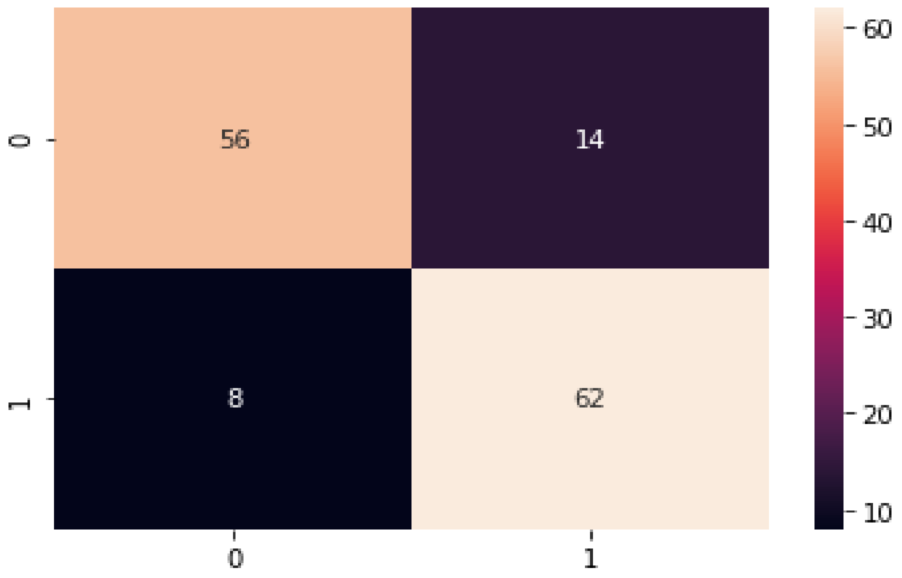
Manage
healthy
File
Home
Share
View
Picture Tools
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC › Muthu (D:) › DS › Deeplearningrecord › 5 Image Classification › mallatai_300 › testing_set › healthy
Search healthy
Pictures
4-b-ann
dscert
ilovepdf_split-rar
Vellore Donbosc
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Muthu (D:)
Krishika (E:)
Amuthan (F:)
Network
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
Item type: PNG File
Dimensions: 3120 x 4160
Size: 3.69 MB
40 items
Type here to search
33°C
ENG US
2:26 AM
10/10/2023



insect
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC › Muthu (D:) › DS › Deeplearningrecord › 5 Image Classification › mallatai_300 › testing_set › insect
Search insect
Pictures
4-b-ann
dscert
ilovepdf_split-rai
Vellore Donbosc
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Muthu (D:)
Krishika (E:)
Amuthan (F:)
Network
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
Item type: PNG File
Dimensions: 4160 x 3120
Size: 4.79 MB
40 items
Type here to search
33°C
ENG
US
2:27 AM
10/10/2023

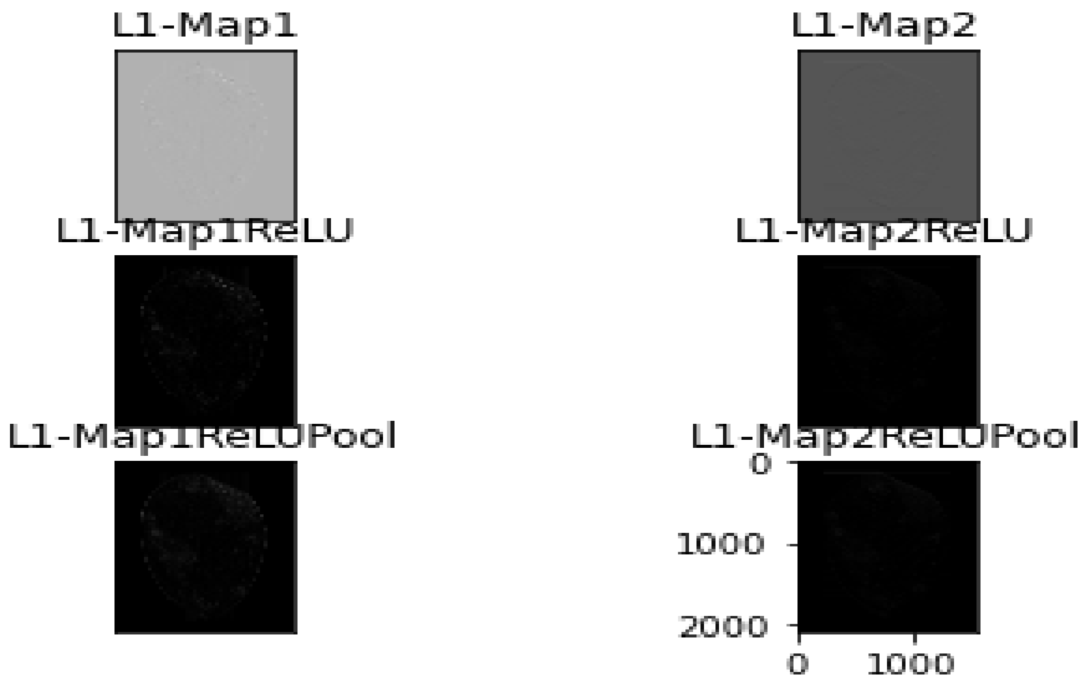


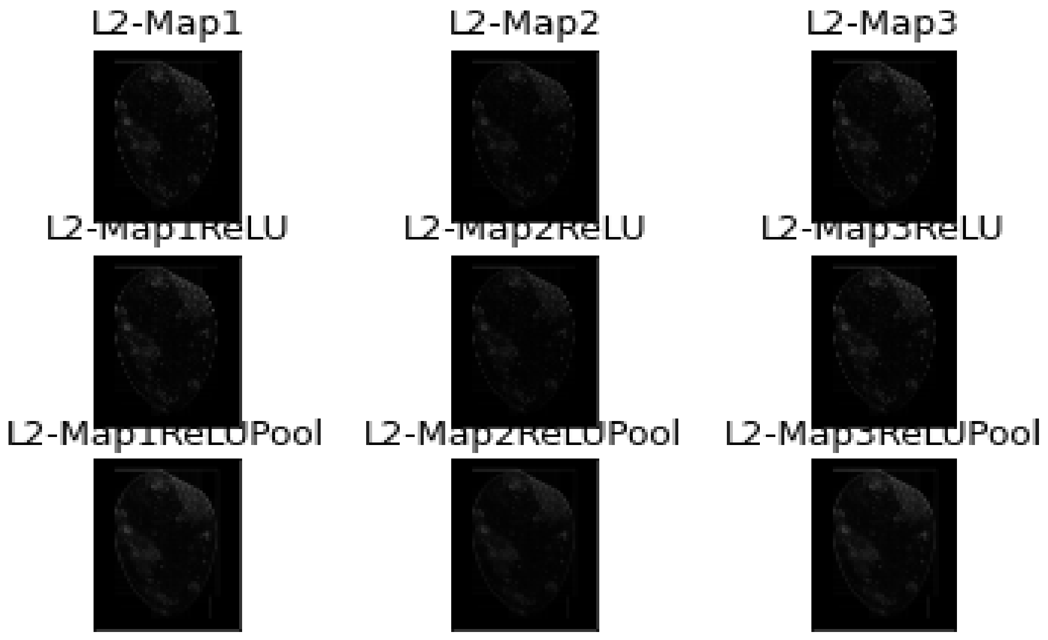
```
0.9286 - val_loss: 0.2352 - val_accuracy: 0.9500
Epoch 21/25
5/5 [==============================] - 3s 663ms/step - loss: 0.1914 - accuracy:
0.9214 - val_loss: 0.5148 - val_accuracy: 0.8500
Epoch 22/25
5/5 [==============================] - 4s 860ms/step - loss: 0.2167 - accuracy:
0.9286 - val_loss: 0.4567 - val_accuracy: 0.8750
Epoch 23/25
5/5 [==============================] - 4s 797ms/step - loss: 0.2101 - accuracy:
0.9214 - val_loss: 0.2527 - val_accuracy: 0.9500
Epoch 24/25
5/5 [==============================] - 4s 897ms/step - loss: 0.1916 - accuracy:
0.9214 - val_loss: 0.3565 - val_accuracy: 0.9000
Epoch 25/25
5/5 [==============================] - 4s 728ms/step - loss: 0.2043 - accuracy:
0.9071 - val_loss: 0.1904 - val_accuracy: 0.9500
Out[72]: <tensorflow.python.keras.callbacks.History at 0x21721b5dda0>
```

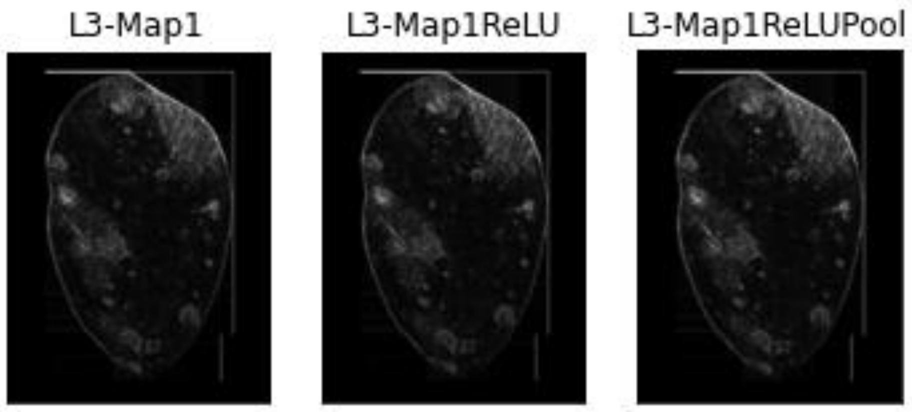


```
Recall [0.82242991 0.61702128]
Specificity [0.61702128 0.82242991]
Precision [0.83018868 0.60416667]
Negative Predictive Value [0.60416667 0.83018868]
False Positive Rate [0.38297872 0.17757009]
False Negative Rate [0.17757009 0.38297872]
False Discovery Rate [0.16981132 0.39583333]
Accuracy [0.75974026 0.75974026]
```

1/1 [==============================] - 0s 31ms/step
The patient has no Diabeties


L1-Map1
L1-Map2
L1-Map1ReLU
L1-Map2ReLU
L1-Map1ReLUPool
L1-Map2ReLUPool
0
1000
2000
0
1000


L2-Map1
L2-Map2
L2-Map3
L2-Map1ReLU
L2-Map2ReLU
L2-Map3ReLU
L2-Map1ReLUPool
L2-Map2ReLUPool
L2-Map3ReLUPool


L3-Map1
L3-Map1ReLU
L3-Map1ReLUPool

Result:-

A python program to implement **Image Classification using Deep Learning** has been implemented and executed successfully.

Ex.No: 15	**Image generation using GAN**
Date:	

Aim:-

To write a python program to implement Image Generation using CNN in Tensorflow and Keras.

Procedure:-

1. Import read_csv from the package pandas.
2. Import numpy, matplotlib, tensorflow from python library
3. Set random seed for reproducibility
4. Load and preprocess the MNIST dataset
5. Split the dataset into training set and testing set
6. Define the generator model using buildgenerator() method.
7. Define the discriminator model using build_discriminator() method.
8. Compile the discriminator using loss function as binary_crossentropy, optimizer as Adam and learning_rate 0.02.
9. Add the Pooling layer to the Convolution layer to the Convolutional Neural Network using Sequential() and use activation function as relu, kernel_initializer as uniform.
10. Add the Second Convolutional layer to the Convolutional Neural Network using optimizer as adam, loss function as categorical_crossentropy, and metrics as accuracy.
11. Train the GAN to generate image with batch_size=64, epochs=10000 and sample_interval=1000.
12. Visualize generated image using matplotlib and graphviz library function

Program

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Reshape
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

# Set random seed for reproducibility
tf.random.set_seed(42)
np.random.seed(42)

# Load and preprocess the MNIST dataset
(train_images, train_labels), (_, _) = mnist.load_data()
train_images = (train_images - 127.5) / 127.5  # Normalize images to the range [-1, 1]
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)  # Reshape for the
discriminator

# Define the generator model
def build_generator():
    model = Sequential()
    model.add(Dense(256, input_dim=100, activation='relu'))
    model.add(Dense(784, activation='tanh'))
    model.add(Reshape((28, 28, 1)))  # Output shape matches MNIST image shape
    return model

generator = build_generator()

# Define the discriminator model
def build_discriminator():
    model = Sequential()
    model.add(Flatten(input_shape=(28, 28, 1)))  # Input shape matches MNIST image shape
    model.add(Dense(256, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model

discriminator = build_discriminator()

# Compile the discriminator
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002),
metrics=['accuracy'])
discriminator.trainable = False  # Freeze the discriminator during the generator training

# Combine the generator and discriminator into a GAN model
gan = Sequential([generator, discriminator])
gan.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0002))
```

```
# Training parameters
batch_size = 64
epochs = 10000
sample_interval = 1000

# Training the GAN
for epoch in range(epochs):
    # Generate random noise as input for the generator
    noise = np.random.normal(0, 1, (batch_size, 100))

    # Generate fake images using the generator
    generated_images = generator.predict(noise)

    # Select a random batch of real images from the dataset
    real_images = train_images[np.random.randint(0, train_images.shape[0], batch_size)]

    # Create labels for real and fake images
    real_labels = np.ones((batch_size, 1))
    fake_labels = np.zeros((batch_size, 1))

    # Train the discriminator on real and fake images
    d_loss_real = discriminator.train_on_batch(real_images, real_labels)
    d_loss_fake = discriminator.train_on_batch(generated_images, fake_labels)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # Generate noise for the generator
    noise = np.random.normal(0, 1, (batch_size, 100))

    # Create labels for the generator
    valid_labels = np.ones((batch_size, 1))

    # Train the generator via the GAN model
    g_loss = gan.train_on_batch(noise, valid_labels)

    # Print progress and save generated images at sample intervals
    if epoch % sample_interval == 0:
        print(f"Epoch {epoch}/{epochs}, D Loss: {d_loss[0]}, G Loss: {g_loss}")
        # Save generated images
        generated_images = (generated_images + 1) * 0.5  # Rescale images to [0, 1]
        fig, axs = plt.subplots(5, 5)
        count = 0
        for i in range(5):
            for j in range(5):
                axs[i, j].imshow(generated_images[count].reshape(28, 28), cmap='gray')
                axs[i, j].axis('off')
                count += 1
        plt.savefig(f"output/gan_image_epoch_{epoch}.png")
        plt.show()
```

Output

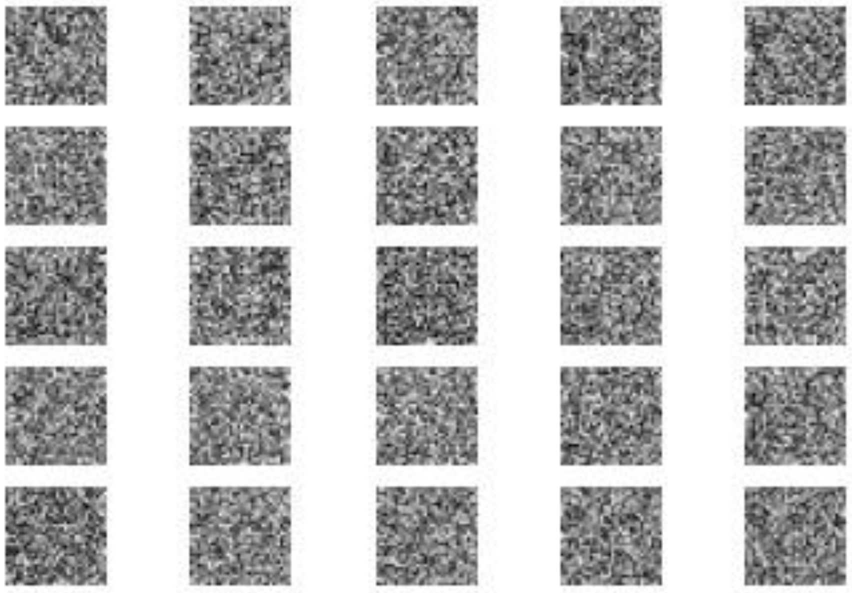


Figure 1: Image Generated for the first iteration (interval 1000)

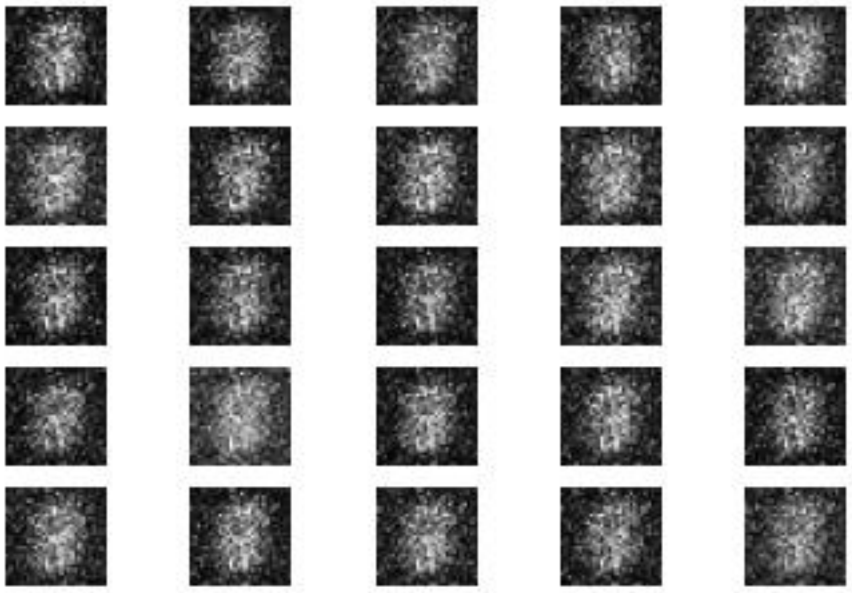


Figure 2: Image Generated for the second iteration (interval 2000)

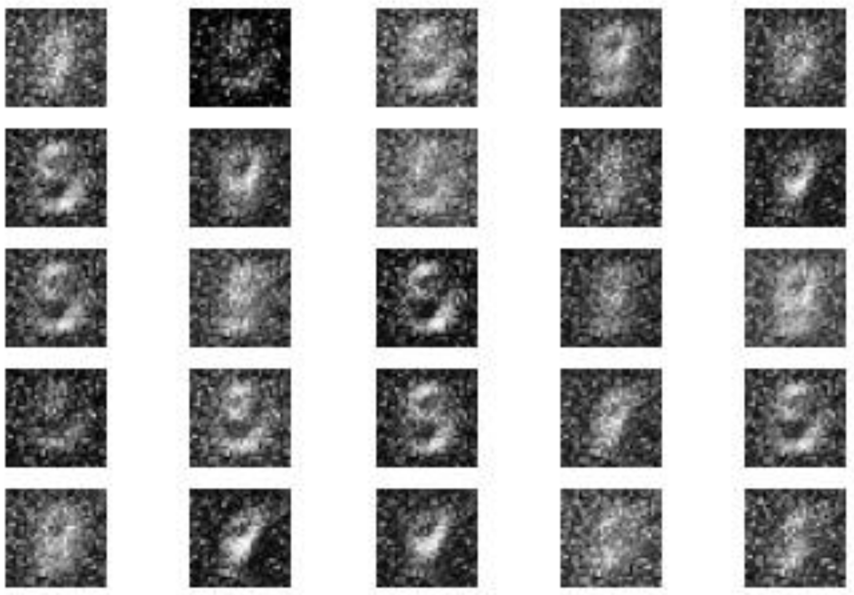


Figure 3: Image Generated for the third iteration (interval 3000)

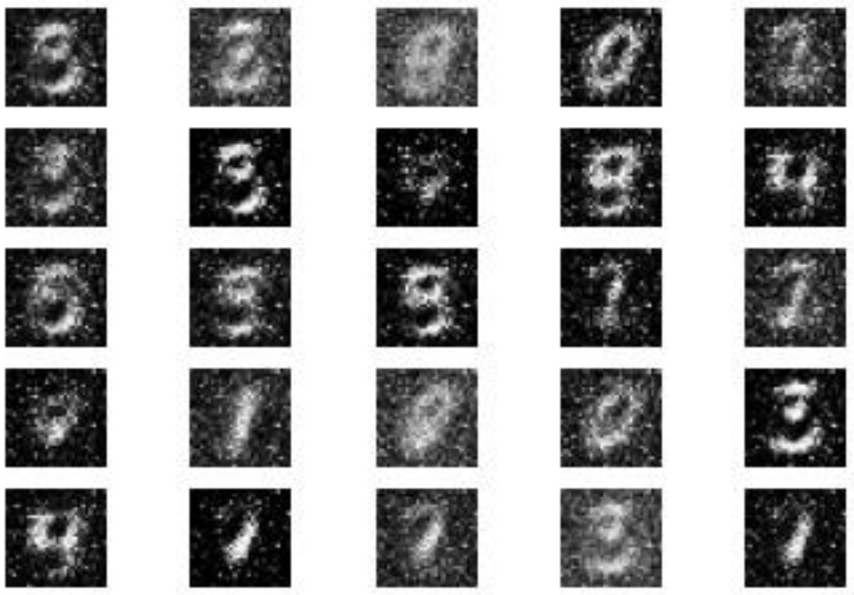


Figure 4: Image Generated for the fourth iteration (interval 4000)

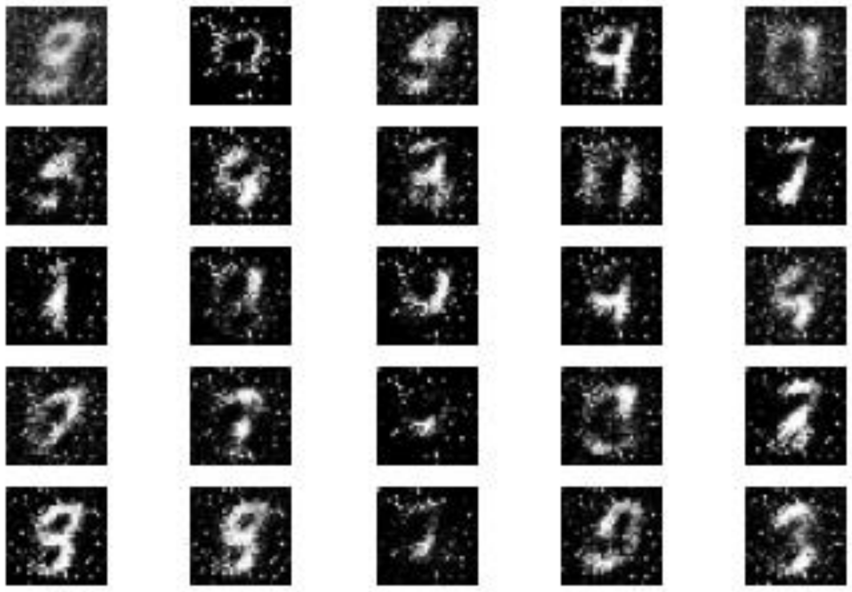


Figure 5: Image Generated for the fifth iteration (interval 5000)

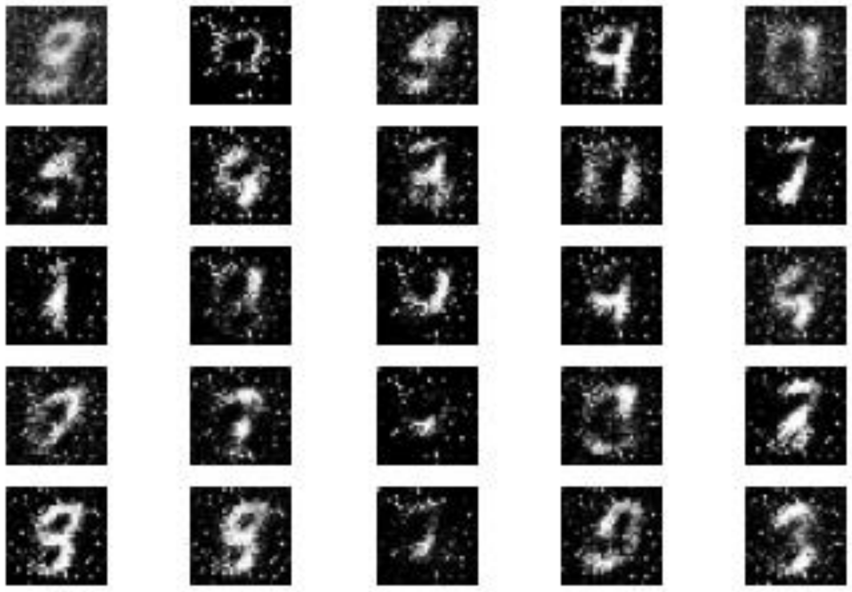


Figure 6: Image Generated for the sixth iteration (interval 6000)

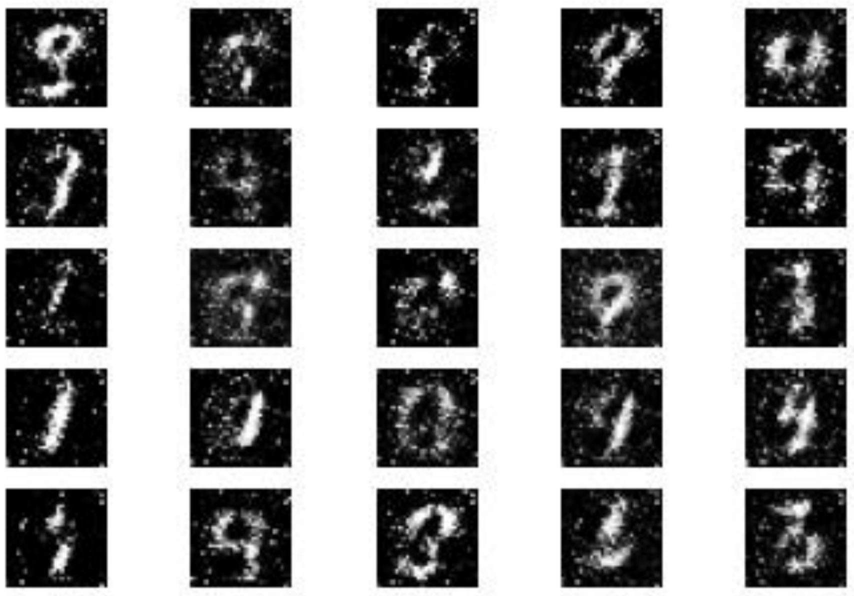


Figure 4: Image Generated for the seventh iteration (interval 7000)

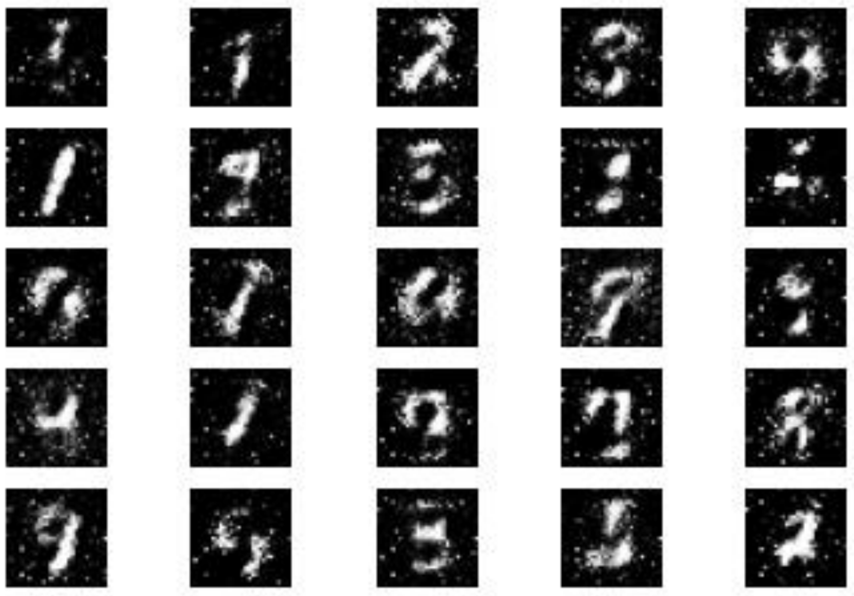


Figure 4: Image Generated for the eight iteration (interval 8000)

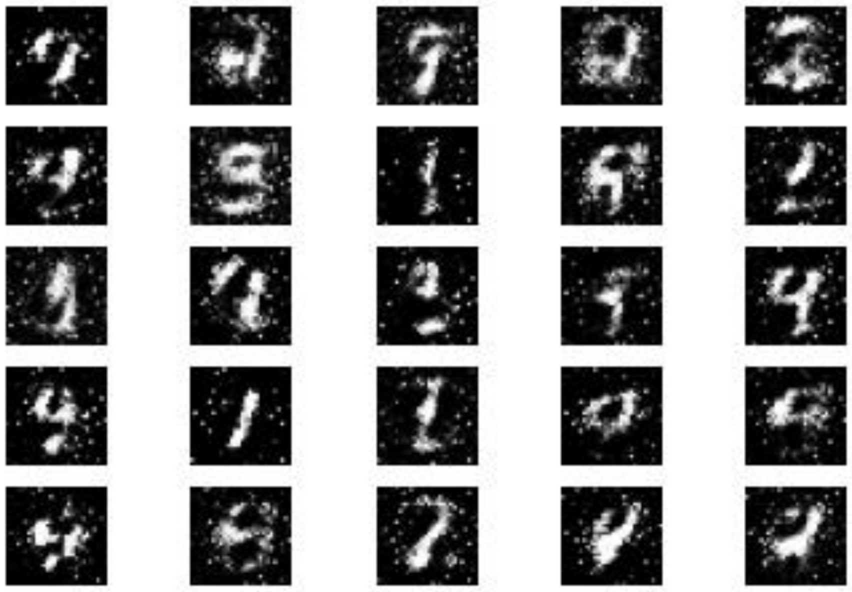


Figure 4: Image Generated for the nineth iteration (interval 4000)

Result:-

A python program to implement **Image Classification using Deep Learning** has been implemented and executed successfully.