PANIMALAR ENGINEERING COLLEGE
CHENNAI CITY CAMPUS

(A CHRISTIAN MINORITY INSTITUTION)

JAISAKTHI EDUCATIONAL TRUST

APPROVED BY ALL INDIA COUNCIL FOR TECHNICAL EDUCATION (AICTE), NEW DELHI

AFFLIATED TO ANNA UNIVERSITY CHENNAI

23, RAILWAY COLONY II STREET, NELSON MANIKAM ROAD, CHENNAI – 600 029

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3491 - ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING (PRACTICAL) (LAB MANUAL)

II YEAR/ IV SEMESTER

CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING (PRACTICAL)

COURSE OBJECTIVES:

- Study about uninformed and Heuristic search techniques.
- Learn techniques for reasoning under uncertainty
- Introduce Machine Learning and supervised learning algorithms
- Study about ensembling and unsupervised learning algorithms
- Learn the basics of deep learning using neural networks

PRACTICAL EXERCISES:

1. Implementation of Uninformed search algorithms (BFS, DFS)
2. Implementation of Informed search algorithms (A*, memory-bounded A*)
3. Implement naïve Bayes models
4. Implement Bayesian Networks
5. Build Regression models
6. Build decision trees and random forests
7. Build SVM models
8. Implement ensembling techniques
9. Implement clustering algorithms
10. Implement EM for Bayesian networks
11. Build simple NN models
12. Build deep learning NN models

TOTAL: 30 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Use appropriate search algorithms for problem solving

CO2: Apply reasoning under uncertainty

CO3: Build supervised learning models

CO4: Build ensembling and unsupervised models

CO5: Build deep learning neural network models

TABLE OF CONTENTS

Implementation of Uninformed search algorithms (BFS, DFS)

Ex.No: 1 (a) **BREADTH FIRST SEARCH**

AIM:

To implement the Breadth First search graph traversal strategy to find the goal state in a state space tree.

ALGORITHM:

1. Initialize a queue data structure to keep track of the nodes to visit.
2. Initialize an array visited[] to keep track of the visited nodes. 3. Add the starting node to the queue.
4. Mark the starting node as visited and add it to the visited set. 5. While the queue is not empty:
6. Dequeue the node from the front of the queue.
7. Visit the node.
8. For each of the node's neighbors:
9. If the neighbor has not been visited yet, mark it as visited and add it to the visited set.
10. Enqueue the neighbor onto the back of the queue. 11. If all nodes have been visited, then the BFS is complete.

In [1]:
```
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
   # Make a list visited[] to check if a node is already visited or not
    def addEdge(self,u,v):
        self.graph[u].append(v)
        self.visited=[]

    # Function to print a BFS of graph
    def BFS(self, s):

        # Create a queue for BFS
        queue = []

        # Add the source node in
        # visited and enqueue it
        queue.append(s)
        self.visited.append(s)

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
            print (s, end = " ")

            # Get all adjacent vertices of the
            # dequeued vertex s. If a adjacent
            # has not been visited, then add it
            # in visited and enqueue it
            for i in self.graph[s]:
```

```
                if i not in self.visited:
                    queue.append(i)
                    self.visited.append(s)

# Driver code

# Create a graph given
g = Graph()
n=int(input("Enter number of Edges : "))
for i in range (0,n):
    u,v = map(int, input("Enter the egde (u-->v) : ").split())
    g.addEdge(u, v)

src=int(input("Enter the Source vertex : "))
g.BFS(src)
```

```
Enter number of Edges : 2
Enter the egde (u-->v) : 1 2
Enter the egde (u-->v) : 1 3
Enter the Source vertex : 1
1 2 3
```

RESULT:
Thus, breadth first search algorithm is successfully implemented and executed.

DEPTH FIRST SEARCH

Ex.No: 1 (b)

AIM:

To implement the Depth First search graph traversal strategy to find the goal state in a state space tree.

ALGORITHM:

1. Initialize a stack data structure to keep track of the nodes to visit.
2. Initialize a set data structure to keep track of the visited nodes.
3. Add the starting node to the stack.
4. Mark the starting node as visited and add it to the visited set.
5. While the stack is not empty:
6. Peek at the node at the top of the stack.
7. If the node has any unvisited neighbors:
8. Choose an unvisited neighbor of the node.
9. Mark the neighbor as visited and add it to the visited set.
10. Push the neighbor onto the stack.
11. If the node has no unvisited neighbors, pop it from the stack.
12. If all nodes have been visited, then the DFS is complete.

In [ ]:
```python
from collections import defaultdict


# This class represents a directed graph using
# adjacency list representation
class Graph:

    # Constructor
    def __init__(self):

        # Default dictionary to store graph
        self.graph = defaultdict(list)


    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)


    # A function used by DFS
    def DFSUtil(self, v, visited):

        # Mark the current node as visited
        # and print it
        visited.add(v)
        print(v, end=' ')

        # Recur for all the vertices
        # adjacent to this vertex
        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)


    # The function to do DFS traversal. It uses
    # recursive DFSUtil()
    def DFS(self, v):

        # Create a set to store visited vertices
        visited = set()
```

```
        # Call the recursive helper function
        # to print DFS traversal
        self.DFSUtil(v, visited)


# Driver's code
#if __name__ == "__main__":
    # Create a graph given
g = Graph()
n=int(input("Enter number of Edges : "))
for i in range (0,n):
    u,v = map(int, input("Enter the egde (u-->v) : ").split())
    g.addEdge(u, v)

src=int(input("Enter the Source vertex : "))
g.DFS(src)
```

RESULT:
Thus, depth first search algorithm is successfully implemented and executed.

Implementation of Informed Search Algorithms

A* SEARCH

Ex. No: 2

AIM:
To implement A* search, an informed search algorithm to find the shortest path to the goal state.

ALGORITHM:

1. Initialise the open set to contain the start node, and the closed set to be empty.
2. Create a dictionary g to store the cost of the shortest path found so far from the start node to each node in the graph, and set g[start_node] to 0.
3. Create a dictionary parents to store the parent of each node in the shortest path found so far from the start node to that node, and set parents[start_node] to start_node.
4. While the open set is not empty, do the following:
   a. Find the node n in the open set with the lowest f-score, where f(n) = g(n) + h(n). If there are multiple nodes with the same lowest f-score, choose any of them.
   b. If n is the goal node or there are no neighbours of n, terminate the algorithm.
   c. For each neighbour m of n, do the following:
      i. If m is not in the open set or the closed set, add m to the open set, set parents[m] = n, and set g[m] = g[n] + cost(n, m).
      ii. Otherwise, if g[m] > g[n] + cost(n, m), update g[m] to g[n] + cost(n, m), set parents[m] = n, and move m from the closed set to the open set.
   d. If n is not the goal node and there are still neighbours to explore, remove n from the open set and add it to the closed set.
5. If the goal node was found, construct the path from start_node to stop_node by following the parent pointers from stop_node to start_node. Return the path.
6. If the goal node was not found, return None.

In [1]:
```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n): n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        #update g(m)
                        g[m] = g[n] + weight
                        #change parent of m to n
                        parents[m] = n
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None
#function to return neighbour and its distance from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
def heuristic(n):
    H_dist = { 'A': 366, 'B': 374, 'C': 329, 'D': 244, 'E': 253, 'F': 178, 'G': 1
    return H_dist[n]

#We use dictionary to represent the graph.Each node is key and the path to adjace
```

```
Graph_nodes = {
'A': [('B', 75), ('C', 118),('E',140)],
'B': [('A', 75)],
'C': [('A', 118), ('D', 111)],
'D': [('C', 111)],
'E': [('A', 140), ('G', 80), ('F', 99)],
'F': [('E', 99), ('I', 211)],
'G': [('E', 80), ('H', 97)],
'H': [('G', 97), ('I', 101)],
'I': [('H', 101), ('F', 211)],
}
aStarAlgo('A', 'I')
```

```
Path found: ['A', 'E', 'G', 'H', 'I']
```

```
Out[1]: ['A', 'E', 'G', 'H', 'I']
```

RESULT:
Thus, program to implement A* search is successfully executed.

Ex. No: 3 **Implement Naive Bayes Models**

AIM:
To write a program to implement Naive Bayes model.

ALGORITHM:

1. Define the research question and specify the variables of Interest
2. Choose a statistical model that describes the relationship between the variables. This may involve specifying a likelihood function that describes the probability of the observed data given the parameters of the model.
3. Specify a prior distribution for the parameters of the model. The prior distribution reflects the researcher's beliefs about the parameters before observing any data.
4. Use Bayes' theorem to calculate the posterior distribution for the parameters of the model, given the observed data. The posterior distribution reflects the researcher's updated beliefs about the parameters after observing the data.
5. Use the posterior distribution to make inferences about the parameters of the model, such as computing point estimates, credible intervals, or hypothesis tests.
6. Evaluate the model's fit to the data and assess whether it provides a good representation of the underlying data-generating process. This may involve checking for model assumptions, assessing goodness of fit, and comparing alternative models.

Naive Bayes Model
Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem. It is called "naive" because it assumes that the features of a data point are independent of each other, even though this may not be true in reality. In a Naive Bayes model, we first train the model on a labeled dataset, where each data point has a set of features and a corresponding label or class. The model learns the probability distribution of each feature for each class, which allows it to predict the most likely class for a new data point based on its features.

```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
# Load the iris dataset
iris = load_iris()
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.3, random_state=0)
# Create a Gaussian Naive Bayes classifier
gnb = GaussianNB()
# Train the classifier on the training data
gnb.fit(X_train, y_train)
# Predict the classes of the test data
y_pred = gnb.predict(X_test)
# Evaluate the performance of the classifier
accuracy = (y_pred == y_test).sum() / len(y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 1.0
```

RESULT:
Thus, program to implement Naive Bayes model is successfully executed.

Ex. No: 4 **BAYESIAN BELIEF NETWORK**

AIM:

To write a program to implement Bayesian Belief Network.

ALGORITHM:

1. Import the required libraries.
2. Load the Cleveland Heart Disease dataset as a DataFrame using pandas —> heartDisease
3. Perform the necessary preprocessing on the dataset such as replacing null values '?' with NAN.
4. Build a Bayesian Network Model for the heart dataset by calling the BayesianNetwork() function, passing its features and class label as input parameters.
5. Fit the model using Maximum Likelihood estimators.
6. Drive the inference from the Bayesian Network model using Variable Elimination —> HeartDiseasetest_infer
7. Now compute the probability of Heart Disease given a particular feature (Eg:restecg,cp) using HeartDiseasetest_infer.

In [4]:
```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model= BayesianNetwork([('age','heartdisease'),('gender','heartdisease'),
                        ('exang','heartdisease'),('cp','heartdisease'),
                        ('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],
                                evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],
                                evidence={'cp':2})
print(q2)
```

```
Sample instances from the dataset are given below
   age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0   63       1   1       145   233    1        2      150      0      2.3
1   67       1   4       160   286    0        2      108      1      1.5
2   67       1   4       120   229    0        2      129      1      2.6
3   37       1   3       130   250    0        0      187      0      3.5
4   41       0   2       130   204    0        2      172      0      1.4

   slope ca thal  heartdisease
0      3  0    6             0
1      2  3    3             2
2      2  2    7             1
3      3  0    3             0
4      1  0    3             0

 Attributes and datatypes
age                int64
gender             int64
cp                 int64
trestbps           int64
chol               int64
fbs                int64
restecg            int64
thalach            int64
exang              int64
oldpeak          float64
slope              int64
ca                object
thal              object
heartdisease       int64
dtype: object

Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
```

```
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

RESULT:
Hence, program to implement Bayesian Belief Network is successfully executed.

Ex. No: 5 LINEAR REGRESSION

AIM:
To implement single and multiple Linear Regression models in Python.

ALGORITHM:

1. Define the research question and specify the variables of interest. Collect data on the dependent and independent variables.
2. Examine the data for outliers, missing values, and other issues that may affect the analysis.
3. Plot the data to explore the relationship between the dependent and independent variables.
4. Specify a linear regression model that describes the relationship between the dependent variable and one or more independent variables.
5. Estimate the coefficients of the linear regression model using a method such as least squares regression.
6. Assess the fit of the linear regression model by examining the residuals.
7. Use the estimated coefficients to make predictions about the dependent variable for new values of the independent variable(s).
8. Evaluate the statistical significance of the coefficients and test whether they are different from zero.
9. Interpret the results of the linear regression model in the context of the research question and the data.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # data visualisation and plotting
import matplotlib.pyplot as plt # data plotting
import warnings

# Seaborn default configuration
sns.set_style("darkgrid")

# set the custom size for my graphs
sns.set(rc={'figure.figsize':(8.7,6.27)})

# filter all warnings
warnings.filterwarnings('ignore')

# set max column to 999 for displaying in pandas
pd.options.display.max_columns=999
data = pd.read_csv('Iris.csv')
data.head()
```

Out[1]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [3]:
```
data.describe()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

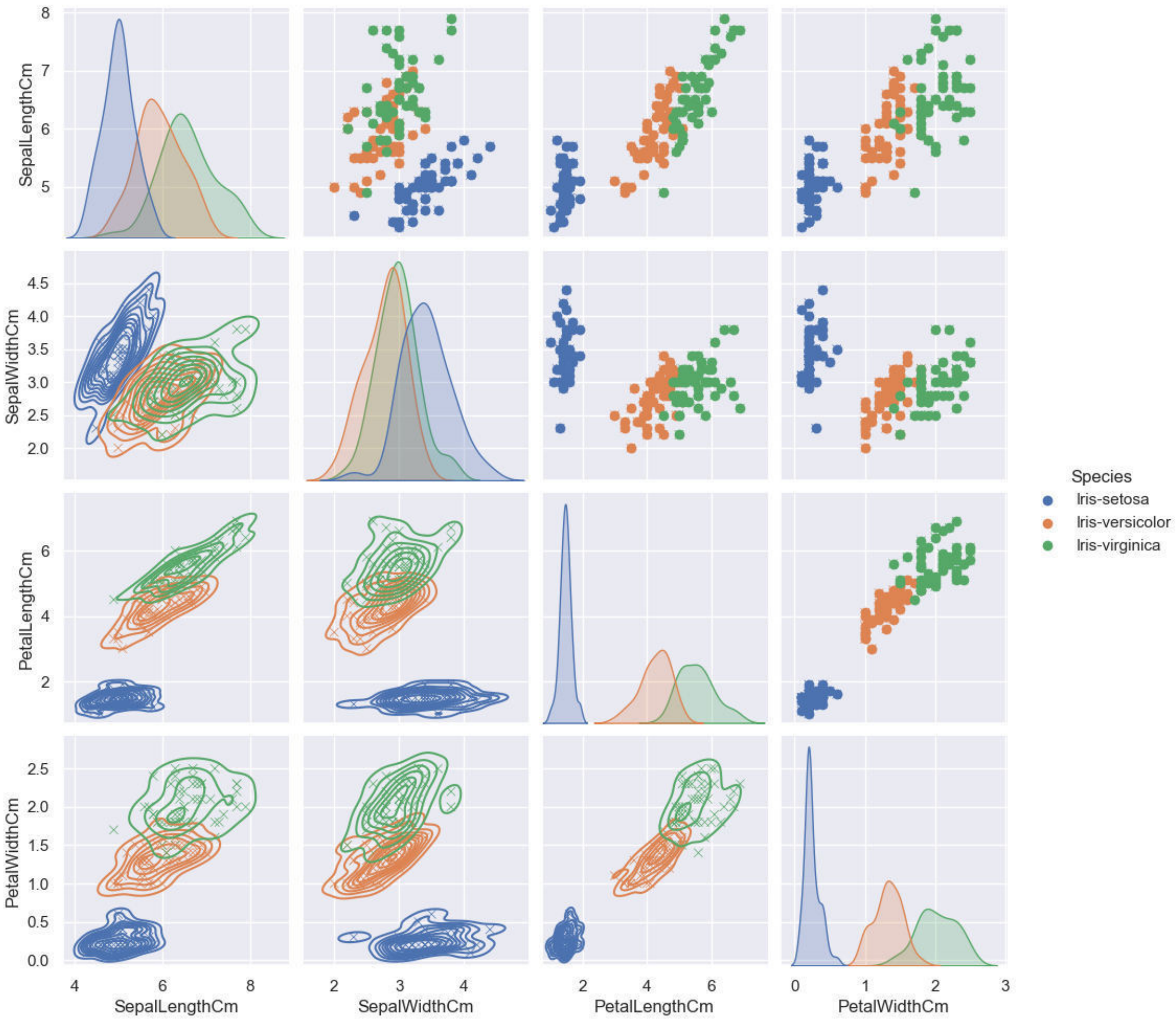
In [4]:
```
data['Species'].value_counts()
```

Out[4]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

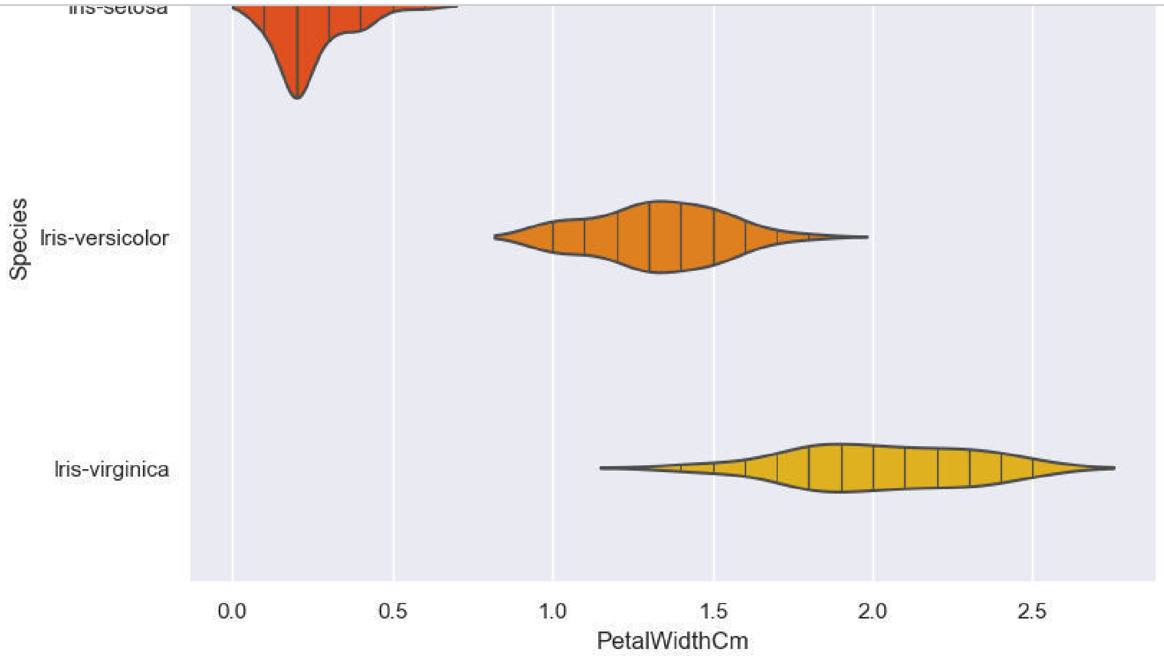
In [5]:
```
rows, col = data.shape
print("Rows : %s, column : %s" % (rows, col))
```

```
Rows : 150, column : 6
```

```
In [6]: snsdata = data.drop(['Id'], axis=1)
        g = sns.pairplot(snsdata, hue='Species', markers='x')
        g = g.map_upper(plt.scatter)
        g = g.map_lower(sns.kdeplot)
```



In [7]:
```
sns.violinplot(x='SepalLengthCm', y='Species', data=data,
               inner='stick', palette='autumn')
plt.show()
sns.violinplot(x='SepalWidthCm', y='Species', data=data,
               inner='stick', palette='autumn')
plt.show()
sns.violinplot(x='PetalLengthCm', y='Species', data=data,
               inner='stick', palette='autumn')
plt.show()
sns.violinplot(x='PetalWidthCm', y='Species', data=data,
               inner='stick', palette='autumn')
plt.show()
```



In [ ]:

In [1]:
```
# Multiple Regression
# Multiple regression is like linear regression, but with more than one indepen
# meaning that we try to predict a value based on two or more variables.
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
# Car, Model, Volume, Weight, CO2
df = pd.read_csv('cars.csv')
print(df.head())
```

```
          Car       Model  Volume  Weight  CO2
0      Toyoty        Aygo    1000     790   99
1  Mitsubishi  Space Star    1200    1160   95
2       Skoda      Citigo    1000     929   95
3        Fiat         500     900     865   90
4        Mini      Cooper    1500    1140  105
```

In [2]:
```
df.describe()
```

Out[2]:

	Volume	Weight	CO2
count	36.000000	36.000000	36.000000
mean	1611.111111	1292.277778	102.027778
std	388.975047	242.123889	7.454571
min	900.000000	790.000000	90.000000
25%	1475.000000	1117.250000	97.750000
50%	1600.000000	1329.000000	99.000000
75%	2000.000000	1418.250000	105.000000
max	2500.000000	1746.000000	120.000000

In [3]:
```
#print(df) # 0, Toyota, Aygo, 1000, 790, 99
x = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
# Tip: It is common to name the list of independent values with a upper case X,
# and the list of dependent values with a lower case y.
regr.fit(x, y)
# predict the CO2 emission of a car where the weight is 2300kg, and the volume
predictedCO2 = regr.predict([[2300, 1300]])
print("predictedCO2 [weight=2300kg, volume=1300ccm]:")
print(predictedCO2) # [107.2087328]

# Coefficient
# The coefficient is a factor that describes the relationship with an unknown v
print("Coefficient [weight, volume]")
print(regr.coef_) # [0.00755095 0.00780526]
```

```
predictedCO2 [weight=2300kg, volume=1300ccm]:
[107.2087328]
Coefficient [weight, volume]
[0.00755095 0.00780526]
```

In [4]:
```
# What if we increase the weight with 1000kg?
predictedCO2 = regr.predict([[3300, 1300]])
print("predictedCO2 [weight=3300kg, volume=1300ccm]")
print(predictedCO2)
```

```
predictedCO2 [weight=3300kg, volume=1300ccm]
[114.75968007]
```

RESULT:
Hence, we have successfully implemented single and multiple Linear Regression models.

Ex. No: 6 Built Decision Trees and Random Forests

AIM:

To write a Python program to build Decision Trees & Random forests

ALGORITHM FOR DECISION TREE:

1. Select the best attribute: The ID3 algorithm selects the attribute that provides the most information gain or has the lowest entropy.
2. Create a decision node: The selected attribute becomes the decision node, and a branch is created for each possible value of the attribute.
3. Split the dataset: The dataset is split into subsets for each value of the selected attribute.
4. Recursively repeat the process: The algorithm recursively repeats the process for each subset until all the data is classified into specific classes.
5. Prune the tree: Finally, the algorithm prunes the tree by removing unnecessary branches or nodes to prevent overfitting.

ALGORITHM FOR RANDOM FOREST

1. Select the number of trees to build: The first step is to determine the number of decision trees to build in the random forest.
2. Randomly select samples: For each tree, randomly select samples from the dataset with replacement. This process is called bootstrapping or bagging.
3. Select random features: Randomly select a subset of features from the dataset. This process is called feature sampling or feature bagging.
4. Build decision tree: Build a decision tree using the bootstrapped dataset and the randomly selected features.
5. Repeat the process: Repeat steps 2-4 until the desired number of trees have been built.
6. Prediction: To make a prediction for a new input, predict the class using all of the trees in the random forest and taking a majority vote.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

In [2]:
```
df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")
df.head()
```

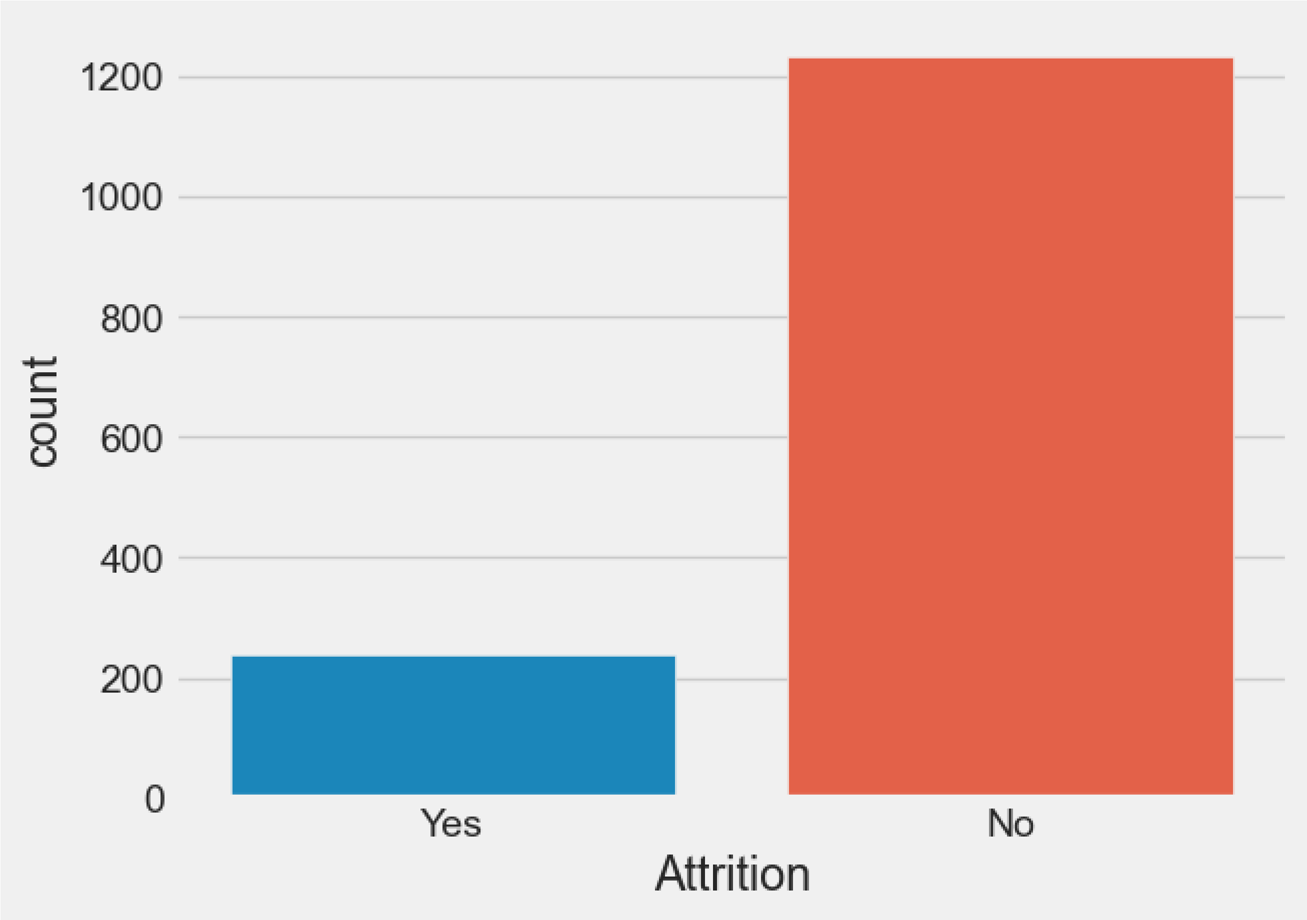
Out[2]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	I
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	

5 rows × 35 columns

In [3]:
```
sns.countplot(x='Attrition', data=df)
```

Out[3]: <Axes: xlabel='Attrition', ylabel='count'>



In [4]:
```
df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'],
        axis="columns", inplace=True)

categorical_col = []
for column in df.columns:
    if df[column].dtype == object and len(df[column].unique()) <= 50:
        categorical_col.append(column)

df['Attrition'] = df.Attrition.astype("category").cat.codes
```

In [5]:
```
categorical_col.remove('Attrition')
```

In [6]:
```
from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()
for column in categorical_col:
    df[column] = label.fit_transform(df[column])
```

In [7]:
```
from sklearn.model_selection import train_test_split

X = df.drop('Attrition', axis=1)
y = df.Attrition

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
```

```
In [9]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n=================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n=================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

In [10]:
```
from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    853.0  176.0       1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[853   0]
 [  0 176]]

Test Result:
================================================
Accuracy Score: 77.78%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.887363   0.259740  0.777778    0.573551      0.800549
recall       0.850000   0.327869  0.777778    0.588934      0.777778
f1-score     0.868280   0.289855  0.777778    0.579067      0.788271
support    380.000000  61.000000  0.777778  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[323  57]
 [ 41  20]]
```

In [11]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

params = {
    "criterion":("gini", "entropy"),
    "splitter":("best", "random"),
    "max_depth":(list(range(1, 20))),
    "min_samples_split":[2, 3, 4],
    "min_samples_leaf":list(range(1, 20)),
}


tree_clf = DecisionTreeClassifier(random_state=42)
tree_cv = GridSearchCV(
    tree_clf,
    params,
    scoring="f1",
    n_jobs=-1,
    verbose=1,
    cv=5
)
tree_cv.fit(X_train, y_train)
best_params = tree_cv.best_params_
print(f"Best paramters: {best_params})")

tree_clf = DecisionTreeClassifier(**best_params)
tree_clf.fit(X_train, y_train)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Fitting 5 folds for each of 4332 candidates, totalling 21660 fits
Best paramters: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 19, 'min_samples_split': 2, 'splitter':
'best'})
Train Result:
================================================
Accuracy Score: 86.78%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.887568    0.692308  0.867833     0.789938      0.854170
recall       0.962485    0.409091  0.867833     0.685788      0.867833
f1-score     0.923510    0.514286  0.867833     0.718898      0.853516
support    853.000000  176.000000  0.867833  1029.000000   1029.000000
_______________________________________________
Confusion Matrix:
 [[821  32]
 [104  72]]

Test Result:
================================================
Accuracy Score: 87.30%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.891304   0.592593  0.873016    0.741948      0.849986
recall       0.971053   0.262295  0.873016    0.616674      0.873016
f1-score     0.929471   0.363636  0.873016    0.646554      0.851204
support    380.000000  61.000000  0.873016  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[369  11]
 [ 45  16]]
```

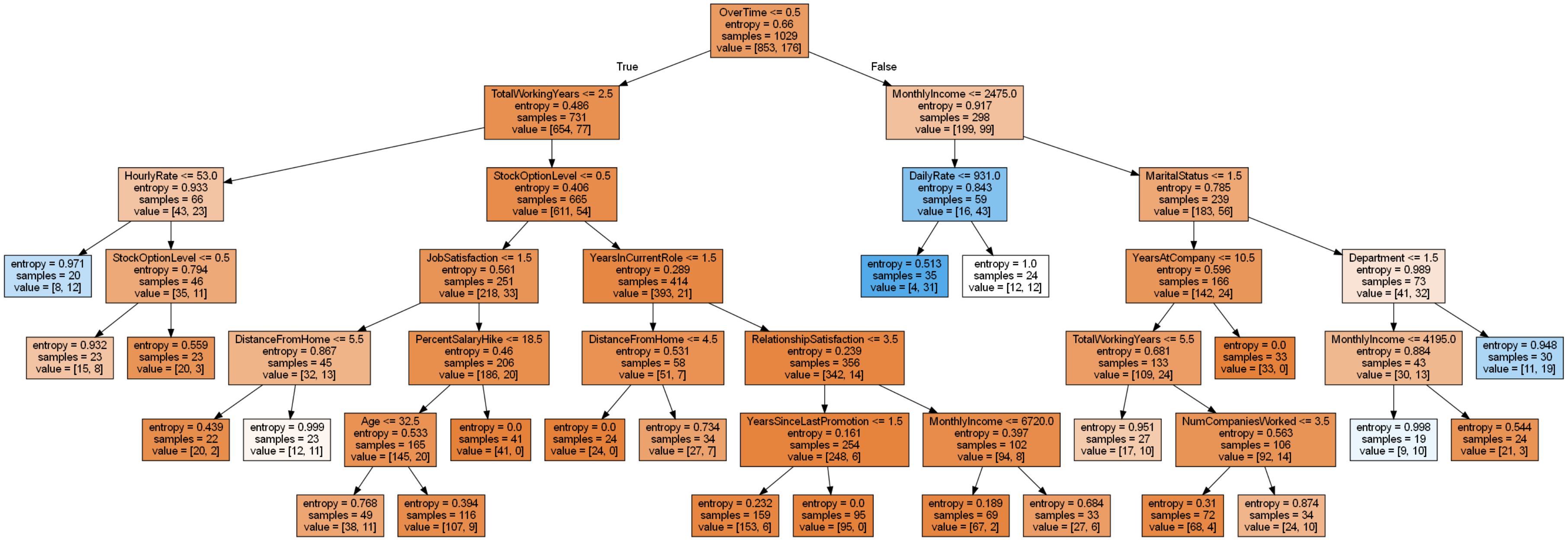
In [14]:
```
pip install pydot
```

```
Collecting pydotNote: you may need to restart the kernel to use updated packages.

  Downloading pydot-2.0.0-py3-none-any.whl (22 kB)
Requirement already satisfied: pyparsing>=3 in c:\users\alwin\anaconda3\lib\site-packages (from pydot) (3.0.9)
Installing collected packages: pydot
Successfully installed pydot-2.0.0
```

In [15]:
```python
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydot

features = list(df.columns)
features.remove("Attrition")
```

In [16]:
```python
dot_data = StringIO()
export_graphviz(tree_clf, out_file=dot_data, feature_names=features, filled=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

Out[16]:



In [17]:
```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    853.0  176.0       1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[853   0]
 [  0 176]]

Test Result:
================================================
Accuracy Score: 85.71%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.869464   0.416667  0.857143    0.643065      0.806832
recall       0.981579   0.081967  0.857143    0.531773      0.857143
f1-score     0.922126   0.136986  0.857143    0.529556      0.813524
support    380.000000  61.000000  0.857143  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[373   7]
 [ 56   5]]
```

RESULT:
Thus, program to build decision trees and random forests is successfully implemented.

Ex. No: 7 SUPPORT VECTOR MACHINE

AIM:
To write a Python program to build Support Vector Machine (SVM) models.

ALGORITHM:
1. Input the training data: The first step is to input the training dataset, which includes the input features and their corresponding labels.
2. Choose the kernel function: Choose a kernel function, which maps
the input features to a higher-dimensional space, where the data is more separable. Some common kernel functions include linear, polynomial, and radial basis function (RBF) kernels.
3. Define the optimization problem: Define the optimization problem, which involves finding the hyperplane that maximizes the margin between the two classes while minimizing the classification error. The hyperplane is defined as the decision boundary that separates the two classes.
4. Solve the optimization problem: Use an optimization algorithm, such as quadratic programming (QP), to solve the optimization problem and find the optimal hyperplane.
5. Predict new inputs: To make a prediction for a new input, use the trained SVM model to classify the input based on which side of the hyperplane it falls.

SUPPERT VECTOR MACHINE (SVM)

SVM was developed in the 1960s and refined in the 1990s. It becomes very popular in the machine learning field because SVM is very powerful compared to other algorithms.

SVM (Support Vector Machine) is a supervised machine learning algorithm. That’s why training data is available to train the model. SVM uses a classification algorithm to classify a two-group problem. SVM focus on decision boundary and support vectors, which we will discuss in the next section.

In [1]:
```python
# Importing Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:
```python
df = pd.read_csv('UniversalBank.csv')
df.head()
```

Out[3]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

In [4]:
```python
# Checking for null values
df.isnull().sum()
```

Out[4]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

In [5]:
```
# Dropping ID and ZIP Code columns from the dataset
df1 = df.drop(["ID","ZIP Code"], axis = 1)
df1.head()
```

Out[5]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	25	1	49	4	1.6	1	0	0	1	0	0	0
1	45	19	34	3	1.5	1	0	0	1	0	0	0
2	39	15	11	1	1.0	1	0	0	0	0	0	0
3	35	9	100	1	2.7	2	0	0	0	0	0	0
4	35	8	45	4	1.0	2	0	0	0	0	0	1

In [9]:
```
zero_class = df1[df1.CreditCard==0]
zero_class.shape
```

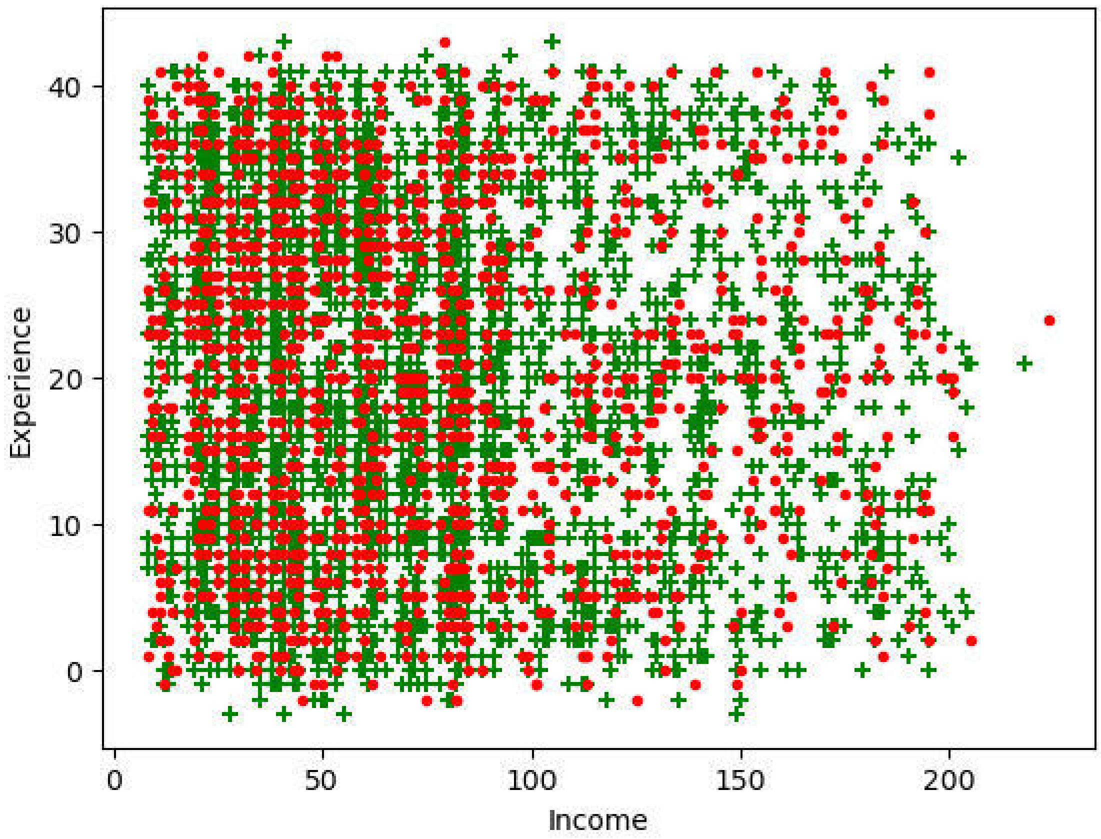
Out[9]:
```
(3530, 12)
```

In [10]:
```
one_class = df1[df1.CreditCard==1]
one_class.shape
```

Out[10]:
```
(1470, 12)
```

In [11]:
```
plt.xlabel('Income')
plt.ylabel('Experience')
plt.scatter(zero_class['Income'],zero_class['Experience'], color = 'green', marker='+')
plt.scatter(one_class['Income'], one_class['Experience'], color = 'red', marker='.')
```

Out[11]: <matplotlib.collections.PathCollection at 0x1e219418490>



In [13]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled = scaler.fit(df1.drop('CreditCard',axis=1)).transform(df1.drop('CreditCard',axis=1))
df_scaled = pd.DataFrame(scaled, columns=df1.columns[:-1])
df_scaled.head()
```

Out[13]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online
0	-1.774417	-1.666078	-0.538229	1.397414	-0.193385	-1.049078	-0.555524	-0.325875	2.928915	-0.25354	-1.216618
1	-0.029524	-0.096330	-0.864109	0.525991	-0.250611	-1.049078	-0.555524	-0.325875	2.928915	-0.25354	-1.216618
2	-0.552992	-0.445163	-1.363793	-1.216855	-0.536736	-1.049078	-0.555524	-0.325875	-0.341423	-0.25354	-1.216618
3	-0.901970	-0.968413	0.569765	-1.216855	0.436091	0.141703	-0.555524	-0.325875	-0.341423	-0.25354	-1.216618
4	-0.901970	-1.055621	-0.625130	1.397414	-0.536736	0.141703	-0.555524	-0.325875	-0.341423	-0.25354	-1.216618

In [15]:
```
# Splitting the columns in to dependent variable (x) and independent variable (y).
x = df_scaled
y = df1['CreditCard']
```

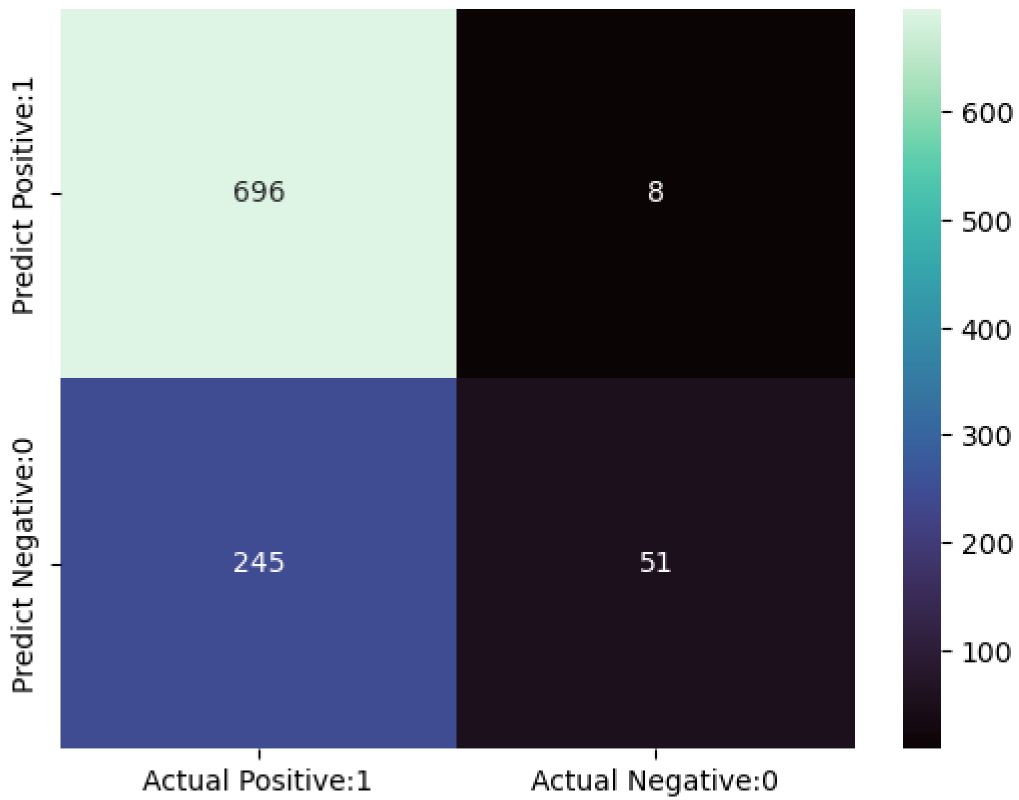
In [16]:
```
# Split data in to train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

In [18]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
linear_classifier=SVC(kernel='linear').fit(x_train,y_train)
y_pred = linear_classifier.predict(x_test)
print('Model accuracy with linear kernel : {0:0.3f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy with linear kernel : 0.747
```

In [20]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                  index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='mako')
```

Out[20]: <AxesSubplot:>



RESULT:
Thus, program to build SVM models is successfully implemented.

Ex. No: 8 BAGGING,RANDOM FOREST AND ADABOOST

AIM:

To write a Python program to implement Ensembling Techniques, namely - bagging, Random forest and Adaboost.

ALGORITHMS:

1. Choose an ensemble technique: Select one or more ensemble techniques, such as Bagging, Boosting, Stacking, Random Forest, or Gradient Boosting Machines.
2. Train multiple models: Train multiple instances of the same model, or multiple different models, using the training dataset.
3. Combine model predictions: Combine the predictions of each model using a specific technique, such as averaging for Bagging and Random Forest, or weighted averaging for Boosting and Gradient Boosting Machines.
4. Evaluate the ensemble model: Evaluate the performance of the ensemble model using a validation dataset, and adjust the hyperparameters of the models as needed.
5. Make predictions: Once the ensemble model has been trained and validated, use it to make predictions for new input data by combining the predictions of each individual model.

ENSEMBLE TECHNIQUES

Ensemble techniques combine multiple machine learning models to improve accuracy and reduce overfitting. Some popular ensemble techniques are as follows:

- Choose an ensemble technique: Select one or more ensemble techniques, such as Bagging, Boosting, Stacking, Random Forest, or Gradient Boosting Machines.
- Train multiple models: Train multiple instances of the same model, or multiple different models, using the training dataset.
- Combine model predictions: Combine the predictions of each model using a specific technique, such as averaging for Bagging and Random Forest, or weighted averaging for Boosting and Gradient Boosting Machines.
- Evaluate the ensemble model: Evaluate the performance of the ensemble model using a validation dataset, and adjust the hyperparameters of the models as needed.
- Make predictions: Once the ensemble model has been trained and validated, use it to make predictions for new input data by combining the predictions of each individual model.
- Ensemble techniques can improve the accuracy and robustness of machine learning models by leveraging the strengths of multiple models, and by reducing the variance and bias of the model. The key to successfully applying ensemble techniques is to choose the right combination of models and techniques, and to optimize the hyperparameters of each individual model.

In [15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

In [16]:
```
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [17]:
```
df.info()
```

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

In [18]:
```
pd.set_option('display.float_format', '{:.2f}'.format)
df.describe()
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	120.89	69.11	20.54	79.80	31.99	0.47	33.24	0.35
std	3.37	31.97	19.36	15.95	115.24	7.88	0.33	11.76	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	0.08	21.00	0.00
25%	1.00	99.00	62.00	0.00	0.00	27.30	0.24	24.00	0.00
50%	3.00	117.00	72.00	23.00	30.50	32.00	0.37	29.00	0.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00	1.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00	1.00

In [19]:
```
categorical_val = []
continous_val = []
for column in df.columns:
#     print('==============================')
#     print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

In [20]:
```
df.columns
```

Out[20]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [21]:
```python
# How many missing zeros are mising in each feature
feature_columns = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'
]

for column in feature_columns:
    print("============================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
============================================
Pregnancies ==> Missing zeros : 111
============================================
Glucose ==> Missing zeros : 5
============================================
BloodPressure ==> Missing zeros : 35
============================================
SkinThickness ==> Missing zeros : 227
============================================
Insulin ==> Missing zeros : 374
============================================
BMI ==> Missing zeros : 11
============================================
DiabetesPedigreeFunction ==> Missing zeros : 0
============================================
Age ==> Missing zeros : 0
```

In [22]:
```python
from sklearn.model_selection import train_test_split


X = df[feature_columns]
y = df.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [26]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report


def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

In [28]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()
bagging_clf = BaggingClassifier(base_estimator=tree, n_estimators=1500, random_state=42)
bagging_clf.fit(X_train, y_train)

evaluate(bagging_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[349   0]
 [  0 188]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   1.00   1.00      1.00       1.00          1.00
recall      1.00   1.00      1.00       1.00          1.00
f1-score    1.00   1.00      1.00       1.00          1.00
support   349.00 188.00      1.00     537.00        537.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[117  34]
 [ 25  55]]
ACCURACY SCORE:
0.7446
CLASSIFICATION REPORT:
               0     1  accuracy  macro avg  weighted avg
precision   0.82  0.62      0.74       0.72          0.75
recall      0.77  0.69      0.74       0.73          0.74
f1-score    0.80  0.65      0.74       0.72          0.75
support   151.00 80.00      0.74     231.00        231.00
```

In [31]:
```
scores = {
    'Bagging Classifier': {
        'Train': accuracy_score(y_train, bagging_clf.predict(X_train)),
        'Test': accuracy_score(y_test, bagging_clf.predict(X_test)),
    },
}
```

Random Forest

In [32]:
```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42, n_estimators=1000)
rf_clf.fit(X_train, y_train)
evaluate(rf_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[349   0]
 [  0 188]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   1.00   1.00      1.00       1.00          1.00
recall      1.00   1.00      1.00       1.00          1.00
f1-score    1.00   1.00      1.00       1.00          1.00
support   349.00 188.00      1.00     537.00        537.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[122  29]
 [ 28  52]]
ACCURACY SCORE:
0.7532
CLASSIFICATION REPORT:
               0     1  accuracy  macro avg  weighted avg
precision   0.81  0.64      0.75       0.73          0.75
recall      0.81  0.65      0.75       0.73          0.75
f1-score    0.81  0.65      0.75       0.73          0.75
support   151.00 80.00      0.75     231.00        231.00
```

In [33]:
```
scores['Random Forest'] = {
        'Train': accuracy_score(y_train, rf_clf.predict(X_train)),
        'Test': accuracy_score(y_test, rf_clf.predict(X_test)),
    }
```

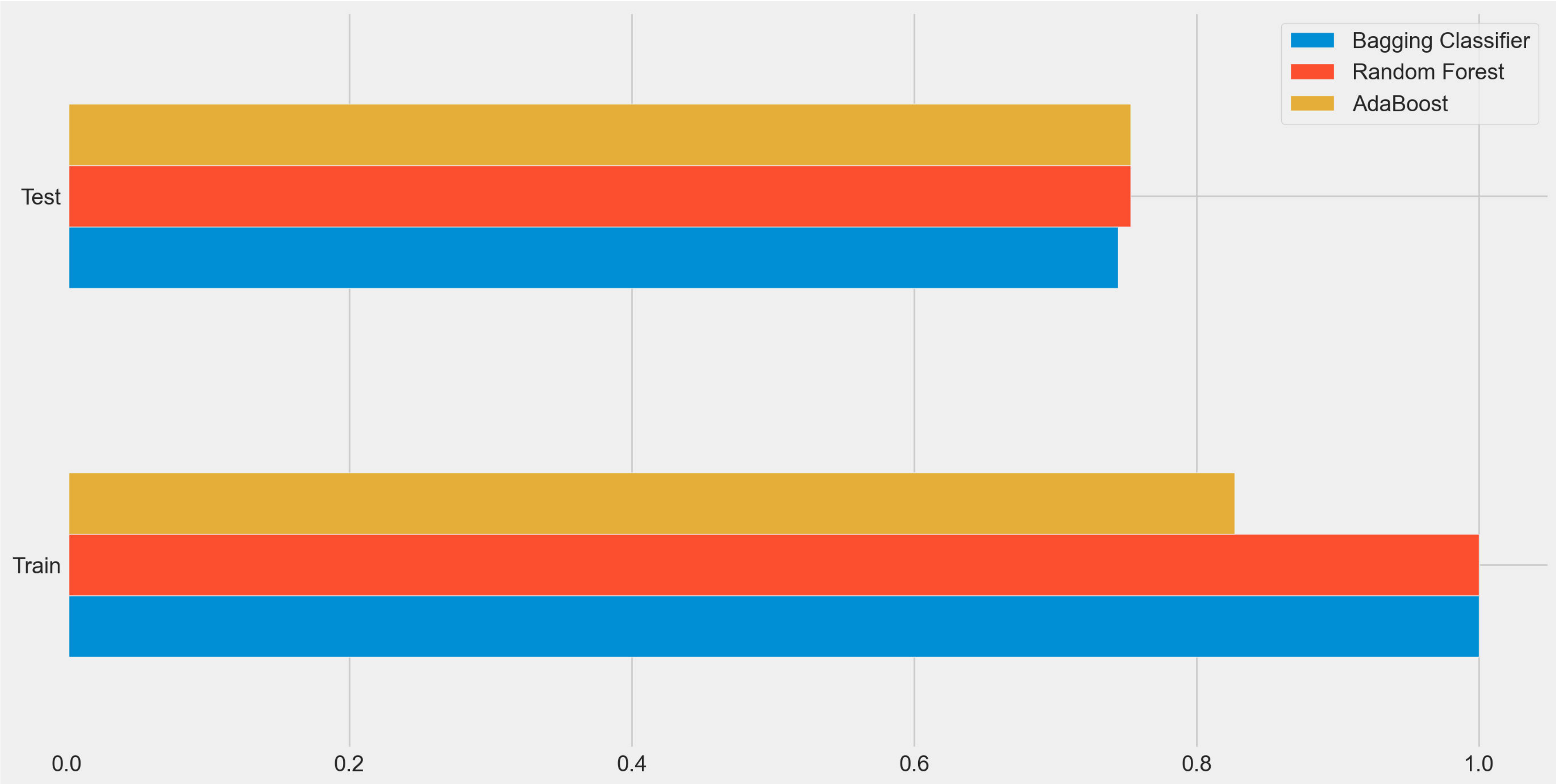
In [34]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_boost_clf = AdaBoostClassifier(n_estimators=30)
ada_boost_clf.fit(X_train, y_train)
evaluate(ada_boost_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[311  38]
 [ 55 133]]
ACCURACY SCORE:
0.8268
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision   0.85   0.78      0.83       0.81          0.82
recall      0.89   0.71      0.83       0.80          0.83
f1-score    0.87   0.74      0.83       0.81          0.82
support   349.00 188.00      0.83     537.00        537.00
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[122  29]
 [ 28  52]]
ACCURACY SCORE:
0.7532
CLASSIFICATION REPORT:
               0     1  accuracy  macro avg  weighted avg
precision   0.81  0.64      0.75       0.73          0.75
recall      0.81  0.65      0.75       0.73          0.75
f1-score    0.81  0.65      0.75       0.73          0.75
support   151.00 80.00      0.75     231.00        231.00
```

In [35]:
```
scores['AdaBoost'] = {
        'Train': accuracy_score(y_train, ada_boost_clf.predict(X_train)),
        'Test': accuracy_score(y_test, ada_boost_clf.predict(X_test)),
    }
```

In [36]:
```
scores_df = pd.DataFrame(scores)

scores_df.plot(kind='barh', figsize=(15, 8))
```

Out[36]: <AxesSubplot:>



RESULT:
Thus, program to implement ensembling techniques is successfully executed

Ex. No: 9 **K-MEANS CLUSTERING**

AIM:

To write a Python program to implement K-Means clustering algorithm.

ALGORITHM:

1. Choose the number of clusters K: Decide on the number of clusters K that the algorithm should create.
2. Initialize cluster centroids: Randomly select K data points from the dataset as initial cluster centroids.
3. Assign each data point to the nearest centroid: For each data point in the dataset, calculate the distance to each of the K centroids and assign the data point to the nearest centroid.
4. Recalculate cluster centroids: Once all data points have been assigned to a cluster, recalculate the centroid of each cluster based on the mean of the data points in that cluster.
5. Repeat steps 3 and 4 until convergence: Repeat steps 3 and 4 until the cluster assignments no longer change or a maximum number of iterations is reached.
6. Output the final clusters: Once the algorithm converges, output the final cluster assignments.

In [1]:
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import io
```

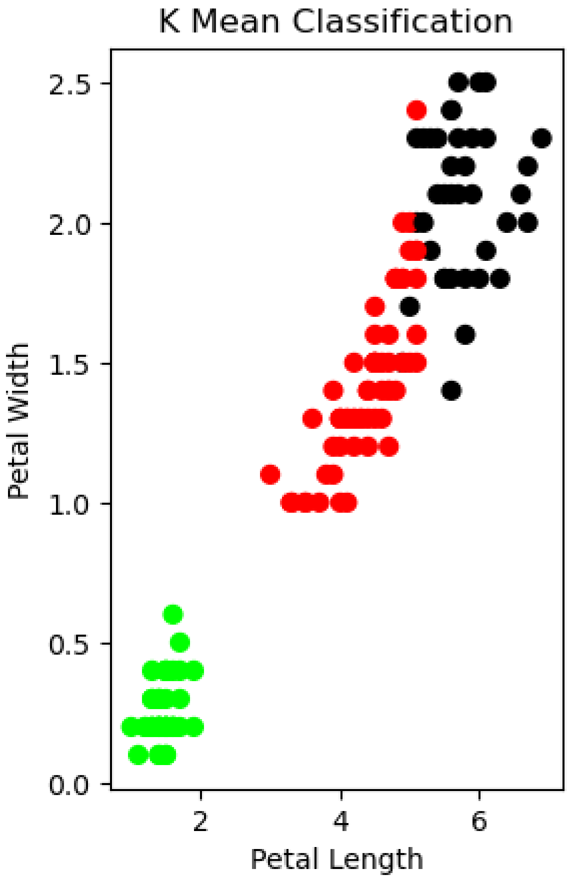
In [2]:
```
iris = datasets.load_iris()
```

In [3]:
```
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
model = KMeans(n_clusters=3)
model.fit(X)
```

Out[3]:
```
KMeans(n_clusters=3)
```

In [8]:
```
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```



RESULT:
Thus, program to implement K-Means clustering is successfully implemented.

Ex. No: 10 **K-NN ALGORITHM**

AIM:

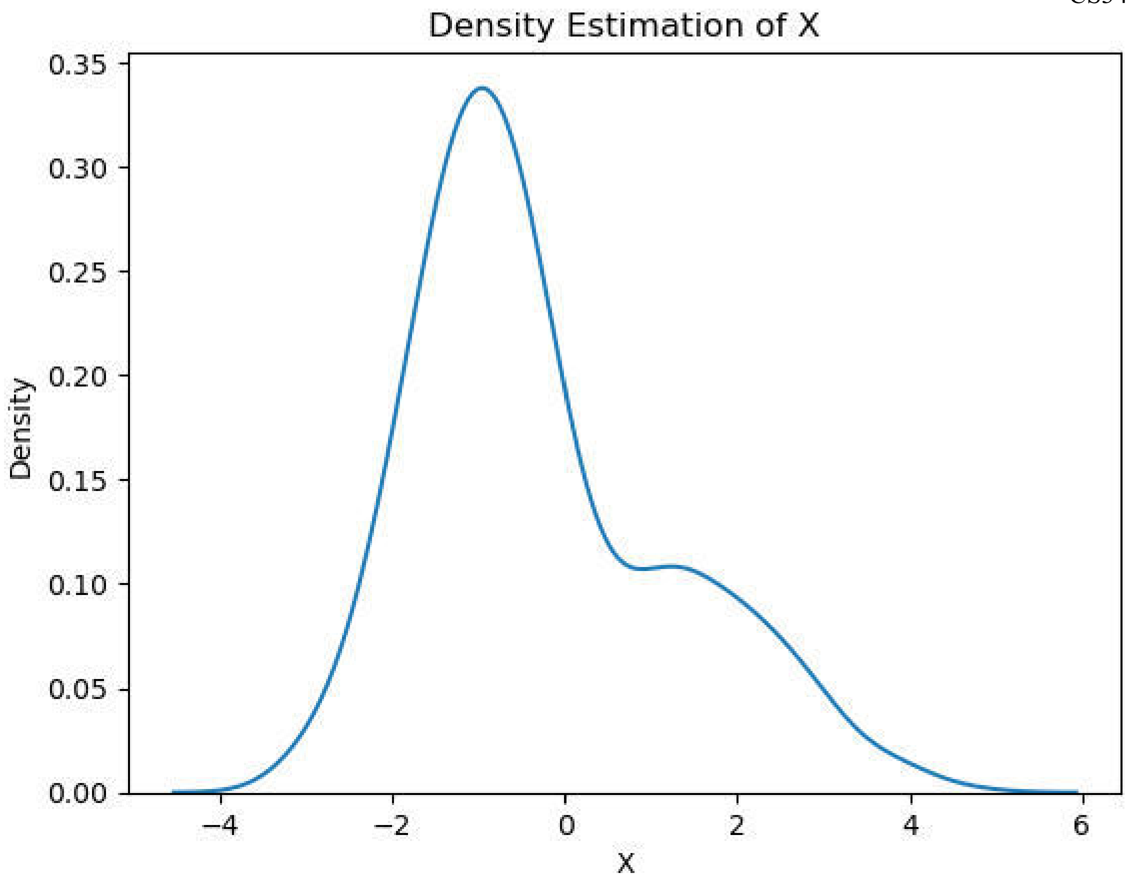
To write a Python program to implement the K-NN algorithm.

ALGORITHM:

1. Choose the number of neighbors K: Decide on the number of nearest neighbors K that the algorithm should consider.
2. Calculate distances: Calculate the distance between the query point and each point in the training set using a distance metric such as Euclidean distance.
3. Find the K-nearest neighbors: Identify the K points in the training set that are closest to the query point based on the distance metric.
4. Assign the class: Assign the class of the query point to be the most common class among its K-nearest neighbors. This is known as majority voting.
5. Output the predicted class: Once the algorithm assigns a class to the query point, output the predicted class.

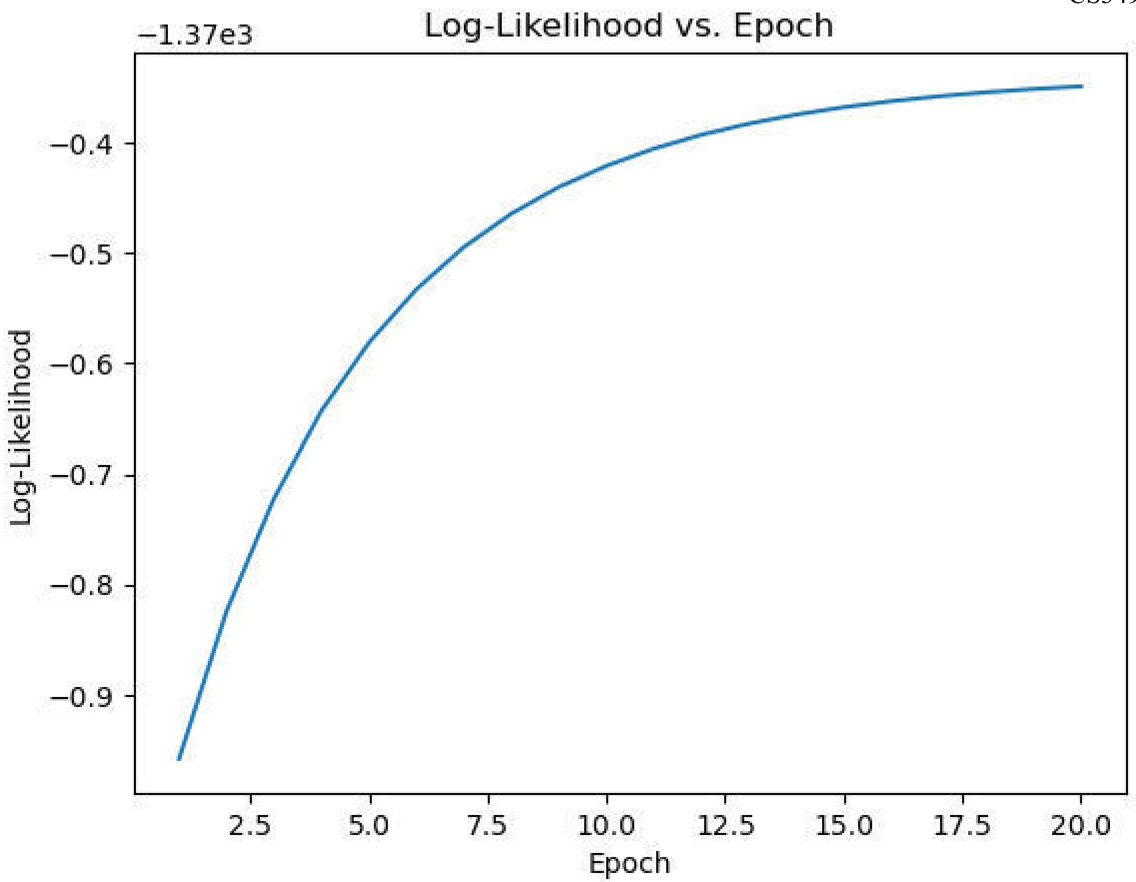
In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.stats import norm
import scipy.stats as stats
import seaborn as sns
```

In [20]:
```
# Generate a dataset with two Gaussian components
mu1, sigma1 = 2, 1
mu2, sigma2 = -1, 0.8
X1 = np.random.normal(mu1, sigma1, size=200)
X2 = np.random.normal(mu2, sigma2, size=600)
X = np.concatenate([X1, X2])

# Plot the density estimation using seaborn
sns.kdeplot(X)
plt.xlabel('X')
plt.ylabel('Density')
plt.title('Density Estimation of X')
plt.show()
```

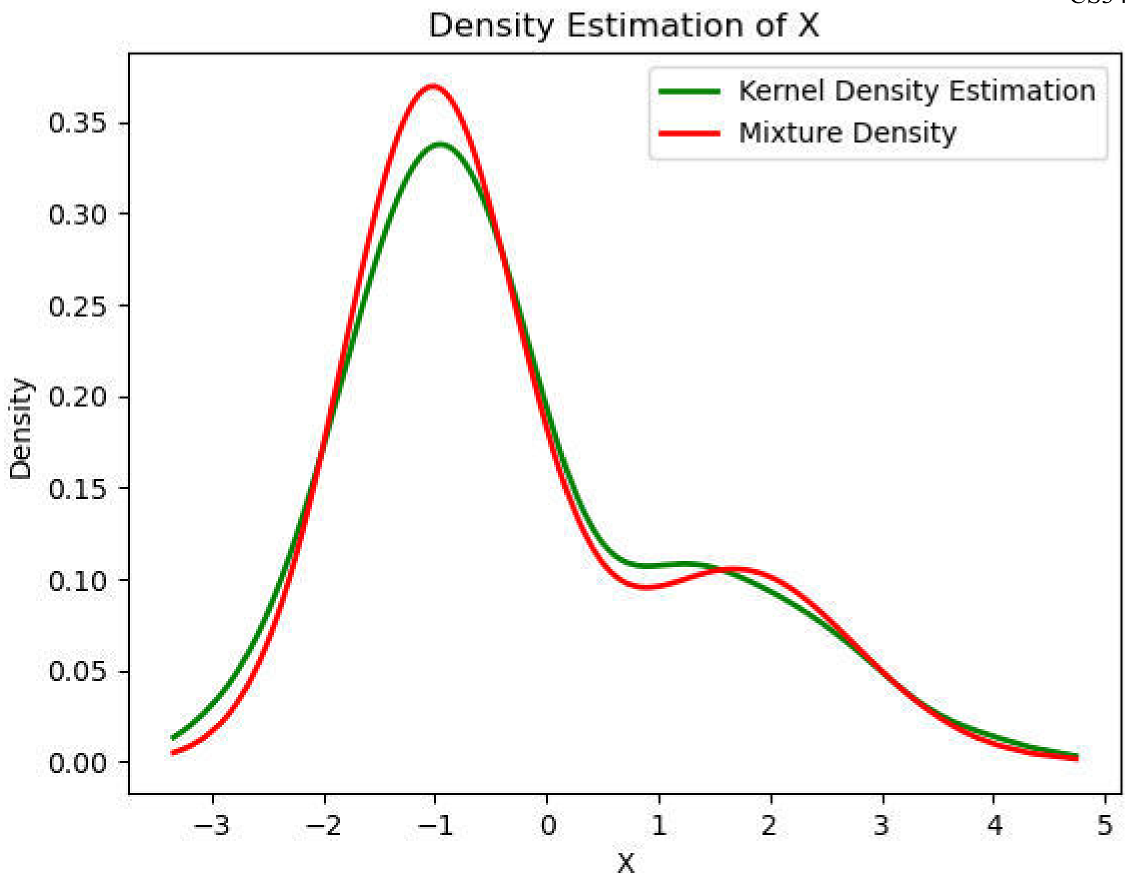


In [21]:
```
# Initialize parameters
mu1_hat, sigma1_hat = np.mean(X1), np.std(X1)
mu2_hat, sigma2_hat = np.mean(X2), np.std(X2)
pi1_hat, pi2_hat = len(X1) / len(X), len(X2) / len(X)
```

In [22]:
```
# Perform EM algorithm for 20 epochs
num_epochs = 20
log_likelihoods = []

for epoch in range(num_epochs):
    # E-step: Compute responsibilities
    gamma1 = pi1_hat * norm.pdf(X, mu1_hat, sigma1_hat)
    gamma2 = pi2_hat * norm.pdf(X, mu2_hat, sigma2_hat)
    total = gamma1 + gamma2
    gamma1 /= total
    gamma2 /= total

    # M-step: Update parameters
    mu1_hat = np.sum(gamma1 * X) / np.sum(gamma1)
    mu2_hat = np.sum(gamma2 * X) / np.sum(gamma2)
    sigma1_hat = np.sqrt(np.sum(gamma1 * (X - mu1_hat)**2) / np.sum(gamma1))
    sigma2_hat = np.sqrt(np.sum(gamma2 * (X - mu2_hat)**2) / np.sum(gamma2))
    pi1_hat = np.mean(gamma1)
    pi2_hat = np.mean(gamma2)

    # Compute log-likelihood
    log_likelihood = np.sum(np.log(pi1_hat * norm.pdf(X, mu1_hat, sigma1_hat)
                                   + pi2_hat * norm.pdf(X, mu2_hat, sigma2_hat)))
    log_likelihoods.append(log_likelihood)

# Plot log-likelihood values over epochs
plt.plot(range(1, num_epochs+1), log_likelihoods)
plt.xlabel('Epoch')
plt.ylabel('Log-Likelihood')
plt.title('Log-Likelihood vs. Epoch')
plt.show()
```


Log-Likelihood vs. Epoch
−1.37e3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.9
Log-Likelihood
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch

In [24]:
```
# Plot the final estimated density
import matplotlib.pyplot as plt
X_sorted = np.sort(X)
density_estimation = pi1_hat*norm.pdf(X_sorted,mu1_hat, sigma1_hat) + pi2_hat * norm.pdf(X_sorted,mu2_hat, sigma2_hat)


plt.plot(X_sorted, stats.gaussian_kde(X_sorted)(X_sorted), color='green', linewidth=2)
plt.plot(X_sorted, density_estimation, color='red', linewidth=2)
plt.xlabel('X')
plt.ylabel('Density')
plt.title('Density Estimation of X')
plt.legend(['Kernel Density Estimation','Mixture Density'])
plt.show()
```



RESULT:
Thus, program to implement the K-NN algorithm is successfully executed.

Ex. No: 11 **BPN ALGORITHM**

AIM:

To implement BPN algorithm to build a simple Neural Network model in Python.

ALGORITHM:

1. Define the architecture: Decide on the number of layers and nodes in each layer of the neural network. The input layer receives input data, the output layer produces the output, and the hidden layers perform intermediate computations.
2. Initialize the weights: Randomly initialize the weights between the nodes of each layer.
3. Feedforward: Pass the input data through the network from the input layer to the output layer. In each layer, compute the weighted sum of the inputs and apply a non-linear activation function such as sigmoid.
4. Calculate error: Calculate the difference between the predicted output and the actual output using a loss function such as mean squared error or cross-entropy.
5. Backpropagation: Propagate the error backwards through the network, adjusting the weights to minimize the error using an optimization algorithm such as stochastic gradient descent. Update the weights in each layer by computing the gradient of the loss function with respect to the weights.
6. Repeat steps 3 to 5: Repeat the feedforward and backpropagation steps for a given number of epochs or until the error converges.
7. Output the predicted result: Once the neural network has been trained, use it to make predictions on new input data by passing it through the network and obtaining the output.

In [1]:
```
import numpy as np
from sklearn.model_selection import train_test_split

db = np.loadtxt("duke-breast-cancer.txt")
print("Database raw shape (%s,%s)" % np.shape(db))
```

```
Database raw shape (86,7130)
```

In [2]:
```
np.random.shuffle(db)
y = db[:, 0]
x = np.delete(db, [0], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
print(np.shape(x_train),np.shape(x_test))
```

```
(77, 7129) (9, 7129)
```

In [3]:
```
hidden_layer = np.zeros(72)
weights = np.random.random((len(x[0]), 72))
output_layer = np.zeros(2)
hidden_weights = np.random.random((72, 2))
```

In [4]:
```
#Sum function
def sum_function(weights, index_locked_col, x):
    result = 0
    for i in range(0, len(x)):
        result += x[i] * weights[i][index_locked_col]
    return result
```

In [5]:
```
#Activation function
def activate_layer(layer, weights, x):
    for i in range(0, len(layer)):
        layer[i] = 1.7159 * np.tanh(2.0 * sum_function(weights, i, x) / 3.0)
```

In [6]:
```
#SoftMax function
def soft_max(layer):
    soft_max_output_layer = np.zeros(len(layer))
    for i in range(0, len(layer)):
        denominator = 0
        for j in range(0, len(layer)):
            denominator += np.exp(layer[j] - np.max(layer))
        soft_max_output_layer[i] = np.exp(layer[i] - np.max(layer)) / denominator
    return soft_max_output_layer
```

In [7]:
```
#Recalculate weights function
def recalculate_weights(learning_rate, weights, gradient, activation):
    for i in range(0, len(weights)):
        for j in range(0, len(weights[i])):
            weights[i][j] = (learning_rate * gradient[j] * activation[i]) + weights[i][j]
```

In [8]:
```
#Back-propagation function
def back_propagation(hidden_layer, output_layer, one_hot_encoding, learning_rate, x):
    output_derivative = np.zeros(2)
    output_gradient = np.zeros(2)
    for i in range(0, len(output_layer)):
        output_derivative[i] = (1.0 - output_layer[i]) * output_layer[i]
    for i in range(0, len(output_layer)):
        output_gradient[i] = output_derivative[i] * (one_hot_encoding[i] - output_layer[i])
    hidden_derivative = np.zeros(72)
    hidden_gradient = np.zeros(72)
    for i in range(0, len(hidden_layer)):
        hidden_derivative[i] = (1.0 - hidden_layer[i]) * (1.0 + hidden_layer[i])
    for i in range(0, len(hidden_layer)):
        sum_ = 0
        for j in range(0, len(output_gradient)):
            sum_ += output_gradient[j] * hidden_weights[i][j]
        hidden_gradient[i] = sum_ * hidden_derivative[i]
    recalculate_weights(learning_rate, hidden_weights, output_gradient, hidden_layer)
    recalculate_weights(learning_rate, weights, hidden_gradient, x)
```

In [9]:
```
one_hot_encoding = np.zeros((2,2))
for i in range(0, len(one_hot_encoding)):
    one_hot_encoding[i][i] = 1
training_correct_answers = 0
for i in range(0, len(x_train)):
    activate_layer(hidden_layer, weights, x_train[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    training_correct_answers += 1 if y_train[i] == np.argmax(output_layer) else 0
    back_propagation(hidden_layer, output_layer, one_hot_encoding[int(y_train[i])], -1, x_train[i])
print("MLP Correct answers while learning: %s / %s (Accuracy = %s) on %s database." % (training_correct_answers,
                                                                                       len(x_train),
                                                                                       training_correct_answers/len(x_train) ,
                                                                                       "Duke breast cancer"))
```

```
MLP Correct answers while learning: 55 / 77 (Accuracy = 0.7142857142857143) on Duke breast cancer database.
```

In [10]:
```
testing_correct_answers = 0
for i in range(0, len(x_test)):
    activate_layer(hidden_layer, weights, x_test[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    testing_correct_answers += 1 if y_test[i] == np.argmax(output_layer) else 0
print("MLP Correct answers while testing: %s / %s (Accuracy = %s) on %s database" % (testing_correct_answers,
                                                                                     len(x_test),
                                                                                     testing_correct_answers/len(x_test),
                                                                                     "Duke breast cancer"))
```

```
MLP Correct answers while testing: 9 / 9 (Accuracy = 1.0) on Duke breast cancer database
```

RESULT:
Thus, a simple NN model built using ANN algorithm is successfully implemented.

Ex. No: 12 **DEEP NEURAL NETWORK**

AIM:

To implement Deep Neural Network models in Python.

ALGORITHM:

1. Load the MNIST dataset: Load the dataset using Keras. The dataset consists of 60,000 training images and 10,000 testing images, each of size 28x28 pixels.
2. Preprocess the data: Reshape the images into a single vector of size 784 (28x28), scale the pixel values to be between 0 and 1, and one-hot encode the labels.
3. Define the model: Define a DNN model using Keras with several hidden layers and an output layer. The input layer should have 784 nodes (one for each pixel), and the output layer should have 10 nodes (one for each digit).
4. Use an appropriate activation function such as ReLU or sigmoid for the hidden layers and softmax for the output layer.
5. Compile the model: Compile the model with an appropriate loss function such as categorical cross-entropy and an optimization algorithm such as Adam or SGD. Also, include metrics such as accuracy to monitor the performance during training.
6. Train the model: Train the model on the training set using the fit() method of the Keras model.
7. Evaluate the model: Evaluate the model on the test set using the evaluate() method of the Keras model. This will give the overall accuracy of the model on the unseen test images.
8. Make predictions: Use the predict() method of the Keras model to make predictions on new images. The output will be a probability distribution over the 10 classes, and the predicted class can be obtained by taking the argmax of the output.

In [8]:

```
#
from sklearn.metrics import confusion_matrix
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop,Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
# import warnings
import warnings
# filter warnings
warnings.filterwarnings('ignore')

model = Sequential()
#
model.add(Conv2D(filters = 8, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
#
model.add(Conv2D(filters = 16, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
# fully connected
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
```

In [9]:
```
# Define the optimizer
optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999)
```

In [10]:
```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
```

In [11]:
```
epochs = 10  # for better result increase the epochs
batch_size = 250
```

In [12]:
```
# read train
train = pd.read_csv("12_train.csv")
print(train.shape)
train.head()
```

```
(42000, 785)
```

Out[12]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel7
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 785 columns

In [13]:
```
# read test
test= pd.read_csv("12_test.csv")
print(test.shape)
test.head()
```

(28000, 784)

Out[13]:

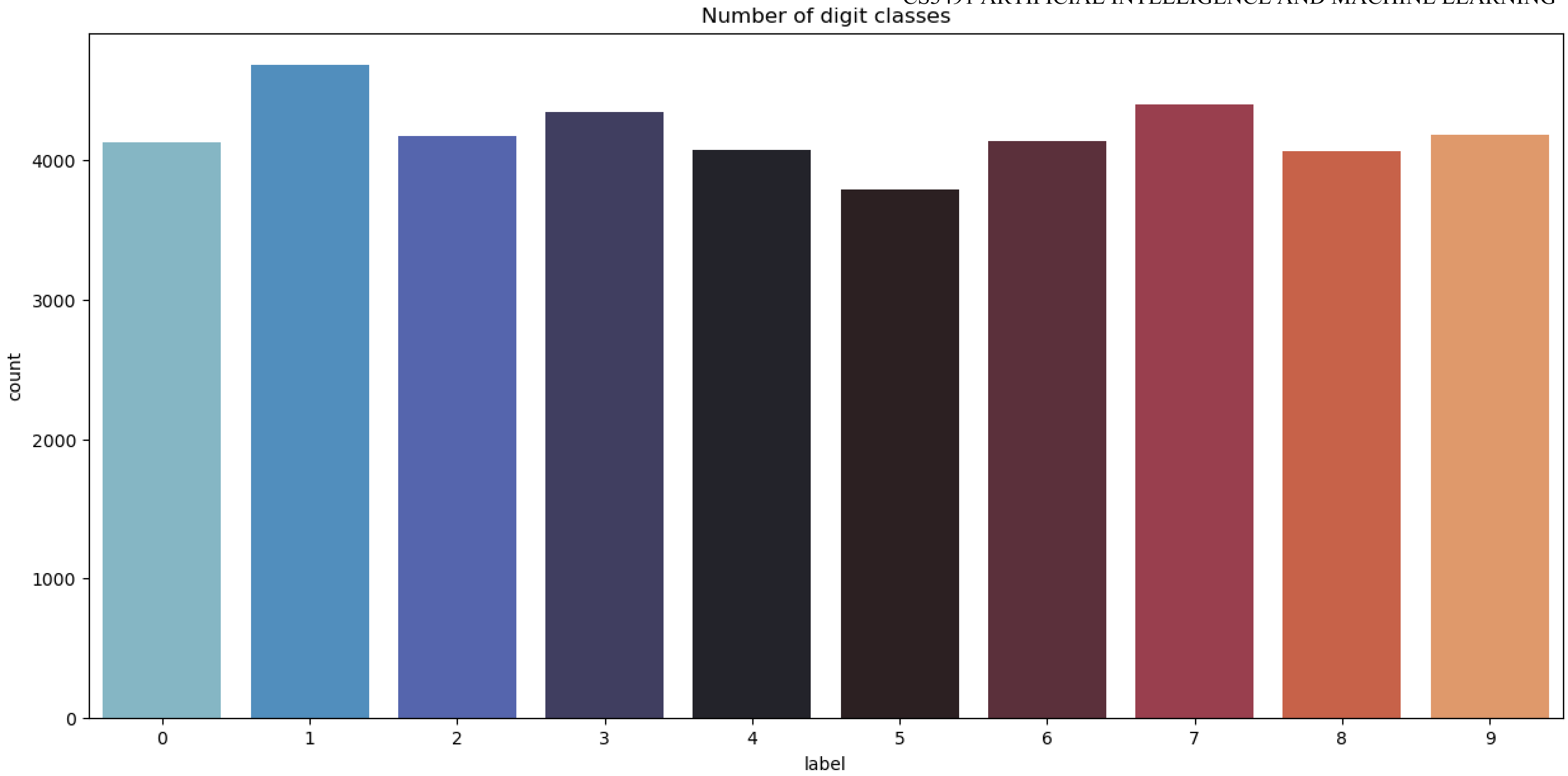
	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixe
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 784 columns

In [14]:
```
# put labels into y_train variable
Y_train = train["label"]
# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)
```

In [15]:
```
# visualize number of digits classes
plt.figure(figsize=(15,7))
g = sns.countplot(Y_train, palette="icefire")
plt.title("Number of digit classes")
Y_train.value_counts()
```

Out[15]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```



In [16]:
```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
print("x_train shape: ",X_train.shape)
print("test shape: ",test.shape)
```

```
x_train shape:  (42000, 784)
test shape:  (28000, 784)
```

In [17]:
```
# Reshape
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
print("x_train shape: ",X_train.shape)
print("test shape: ",test.shape)
```

```
x_train shape:  (42000, 28, 28, 1)
test shape:  (28000, 28, 28, 1)
```

In [18]:
```
# data augmentation
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # dimesion reduction
        rotation_range=5,  # randomly rotate images in the range 5 degrees
        zoom_range = 0.1, # Randomly zoom image 10%
        width_shift_range=0.1,  # randomly shift images horizontally 10%
        height_shift_range=0.1,  # randomly shift images vertically 10%
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

datagen.fit(X_train)
```

In [20]:
```
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
Y_train = to_categorical(Y_train, num_classes = 10)
```

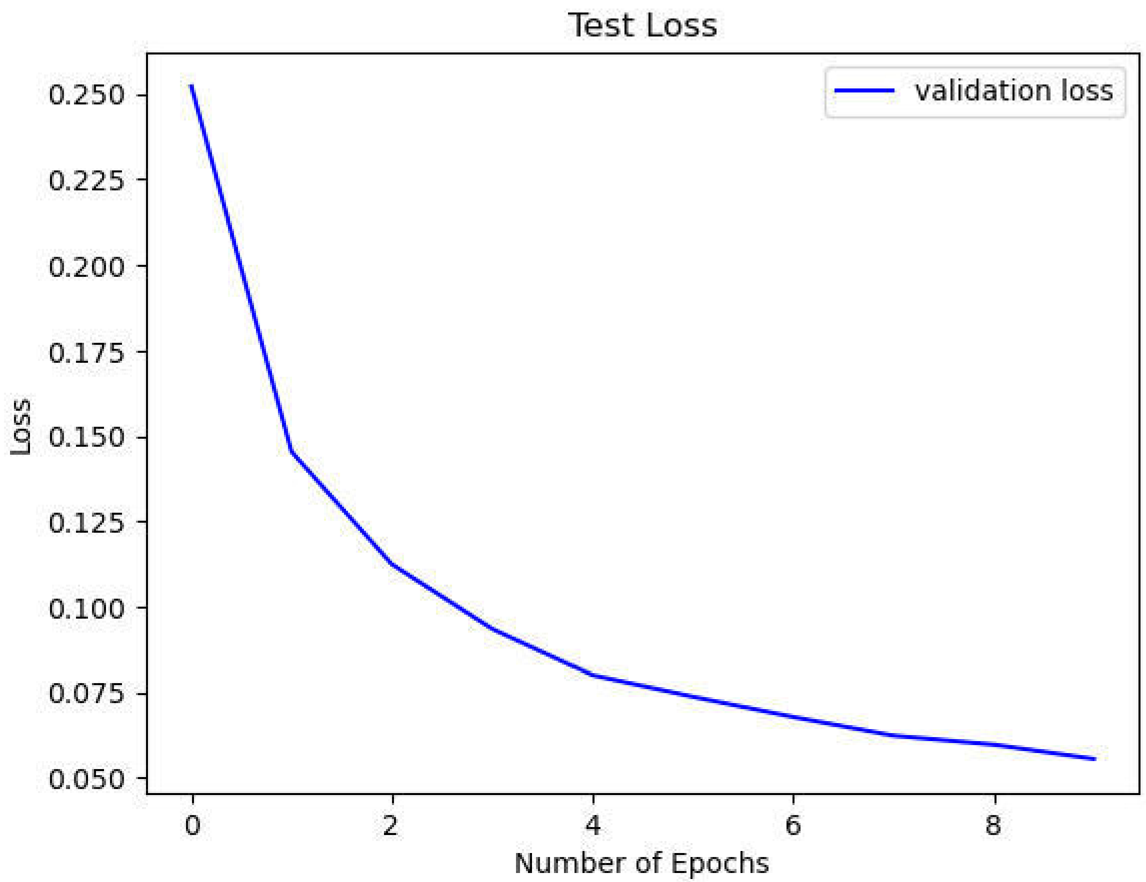
In [21]:
```
# Split the train and the validation set for the fitting
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=2)
print("x_train shape",X_train.shape)
print("x_test shape",X_val.shape)
print("y_train shape",Y_train.shape)
print("y_test shape",Y_val.shape)
```

```
x_train shape (37800, 28, 28, 1)
x_test shape (4200, 28, 28, 1)
y_train shape (37800, 10)
y_test shape (4200, 10)
```

In [22]:
```
# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val),
                              steps_per_epoch=X_train.shape[0] // batch_size)
```

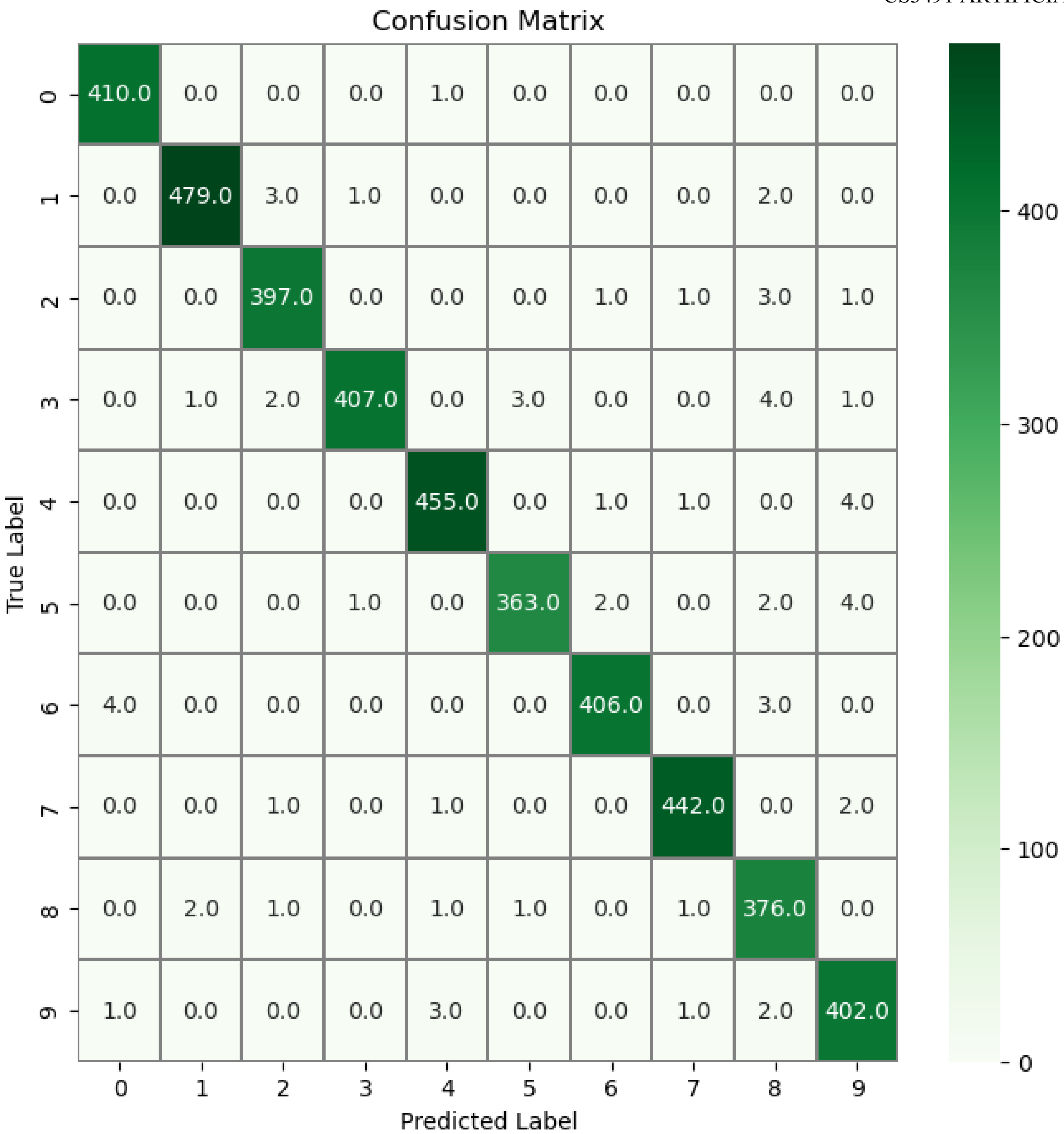
```
Epoch 1/10
151/151 [==============================] - 23s 145ms/step - loss: 1.1468 - accuracy: 0.6143 - val_loss: 0.2522 - val_accur
acy: 0.9333
Epoch 2/10
151/151 [==============================] - 23s 149ms/step - loss: 0.4687 - accuracy: 0.8518 - val_loss: 0.1453 - val_accur
acy: 0.9567
Epoch 3/10
151/151 [==============================] - 22s 145ms/step - loss: 0.3370 - accuracy: 0.8945 - val_loss: 0.1124 - val_accur
acy: 0.9669
Epoch 4/10
151/151 [==============================] - 20s 134ms/step - loss: 0.2894 - accuracy: 0.9091 - val_loss: 0.0935 - val_accur
acy: 0.9729
Epoch 5/10
151/151 [==============================] - 20s 132ms/step - loss: 0.2498 - accuracy: 0.9221 - val_loss: 0.0800 - val_accur
acy: 0.9760
Epoch 6/10
151/151 [==============================] - 22s 143ms/step - loss: 0.2326 - accuracy: 0.9291 - val_loss: 0.0738 - val_accur
acy: 0.9781
Epoch 7/10
151/151 [==============================] - 20s 133ms/step - loss: 0.2097 - accuracy: 0.9339 - val_loss: 0.0679 - val_accur
acy: 0.9798
Epoch 8/10
151/151 [==============================] - 20s 134ms/step - loss: 0.1980 - accuracy: 0.9383 - val_loss: 0.0624 - val_accur
acy: 0.9798
Epoch 9/10
151/151 [==============================] - 20s 133ms/step - loss: 0.1838 - accuracy: 0.9443 - val_loss: 0.0598 - val_accur
acy: 0.9821
Epoch 10/10
151/151 [==============================] - 31s 208ms/step - loss: 0.1764 - accuracy: 0.9457 - val_loss: 0.0556 - val_accur
acy: 0.9850
```

In [23]:
```
# Plot the loss and accuracy curves for training and validation
plt.plot(history.history['val_loss'], color='b', label="validation loss")
plt.title("Test Loss")
plt.xlabel("Number of Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [24]:
```
# confusion matrix
import seaborn as sns
# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="Greens",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
132/132 [==============================] - 1s 4ms/step
```



RESULT:

Hence, program to build Deep NN models is successfully implemented