MACHINE LEARNING
LAB DIGITAL ASSIGNMENT-3

Course Code: CSE4020

NAME: PADARTHY YAGNESH SAI
Reg.No:20BCE0625
LAB SLOT: L11+L12
MARCH 31, 2023

Name:Padarthy Yagnesh Sai

Reg.No:20BCE0625

Labslot:L11+L12

Question-1

1. Implement Random forest Trees algorithm and test the algorithm using any data set of your choice.The output should include Accuracy, Error rate, Precision and recall rate along with the confusion matrix.

In [42]:
```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,recall_score, classification_report
import warnings
warnings.filterwarnings('ignore')
```

In [43]:
```
# Load the dataset
df= pd.read_csv("SocialNetworkAds.csv")
```

In [44]:
```
# display the first 5 rows
df.head(n=5)
```

Out[44]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [45]:
```
# Shape of the dataset
shape = df.shape
print(f"The shape of the data set is {shape}")
```

```
The shape of the data set is (400, 5)
```

In [47]:
```
#describe() method for the dataset
df.describe()
```

Out[47]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

In [48]:
```
# info() method for the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [49]:
```
# check for missing values
df.isnull().sum()
```

Out[49]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [50]:
```
# Extracting Independent and dependent Variable
x= df.iloc[:, [2,3]].values
y= df.iloc[:, 4].values
```

In [51]:
```
# Splitting the dataset into training and test set.
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,random_state=0)
```

In [52]:
```
# Feature Scaling
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [53]:
```
# Model creation
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
```

In [54]:
```
# Fitting the model
classifier.fit(x_train, y_train)
```

Out[54]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [55]:
```
# Predict output
y_pred= classifier.predict(x_test)
```

In [56]:
```
# Sample input predict
sample_input = x_test[0, :]
sample_output = classifier.predict(sample_input.reshape(1, -1))
print("Sample input: ", sample_input)
print("Sample output: ", sample_output)
```

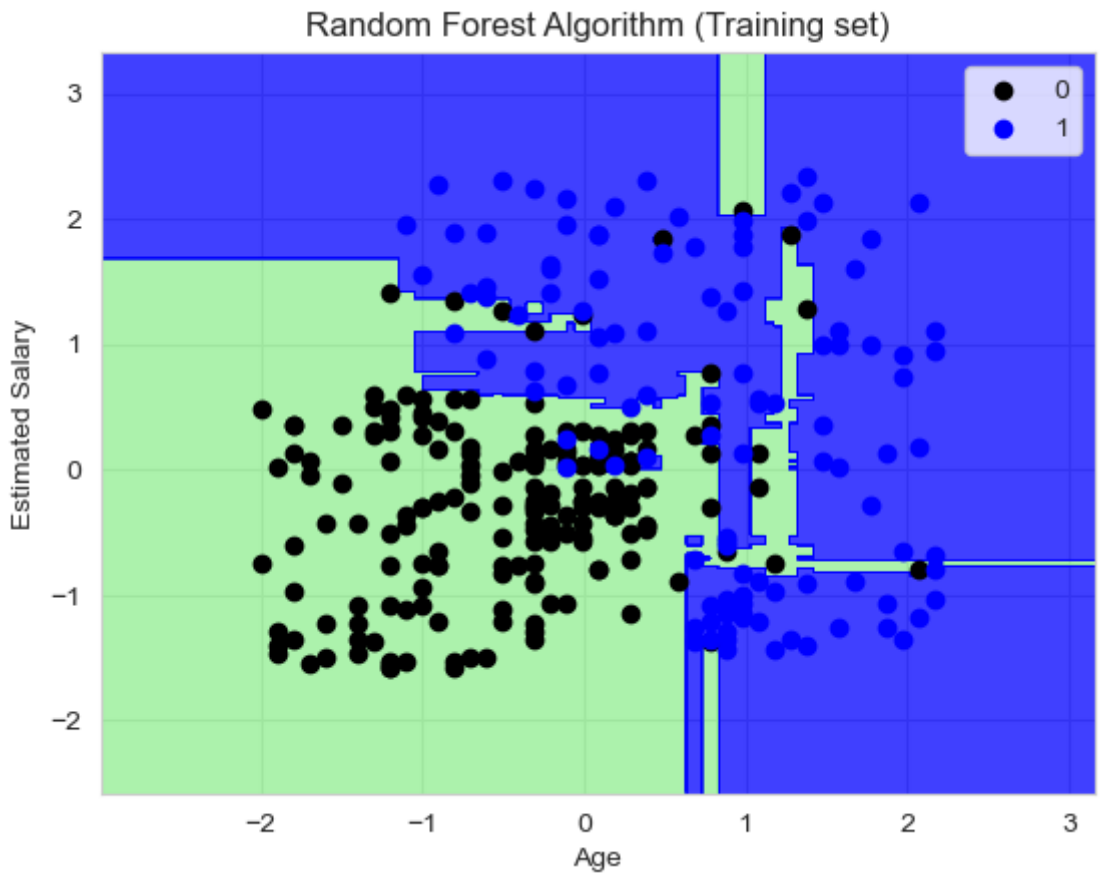
```
Sample input:  [-0.80480212  0.50496393]
Sample output:  [0]
```

In [57]:
```
# Performance metrics
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix: ")
print(cm)
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
# Error rate
error_rate = 1 - accuracy
print("Error rate: ", error_rate)
# Precision
precision = precision_score(y_test, y_pred, average='macro')
print("Precision: ", precision)
# Recall rate
recall = recall_score(y_test, y_pred, average='macro')
print("Recall rate: ", recall)
# Classification Report
print("Classification Report\n")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[64  4]
 [ 5 27]]
Accuracy:  0.91
Error rate:  0.08999999999999997
Precision:  0.8992519869097709
Recall rate:  0.8924632352941176
Classification Report

              precision    recall  f1-score   support

           0       0.93      0.94      0.93        68
           1       0.87      0.84      0.86        32

    accuracy                           0.91       100
   macro avg       0.90      0.89      0.90       100
weighted avg       0.91      0.91      0.91       100
```

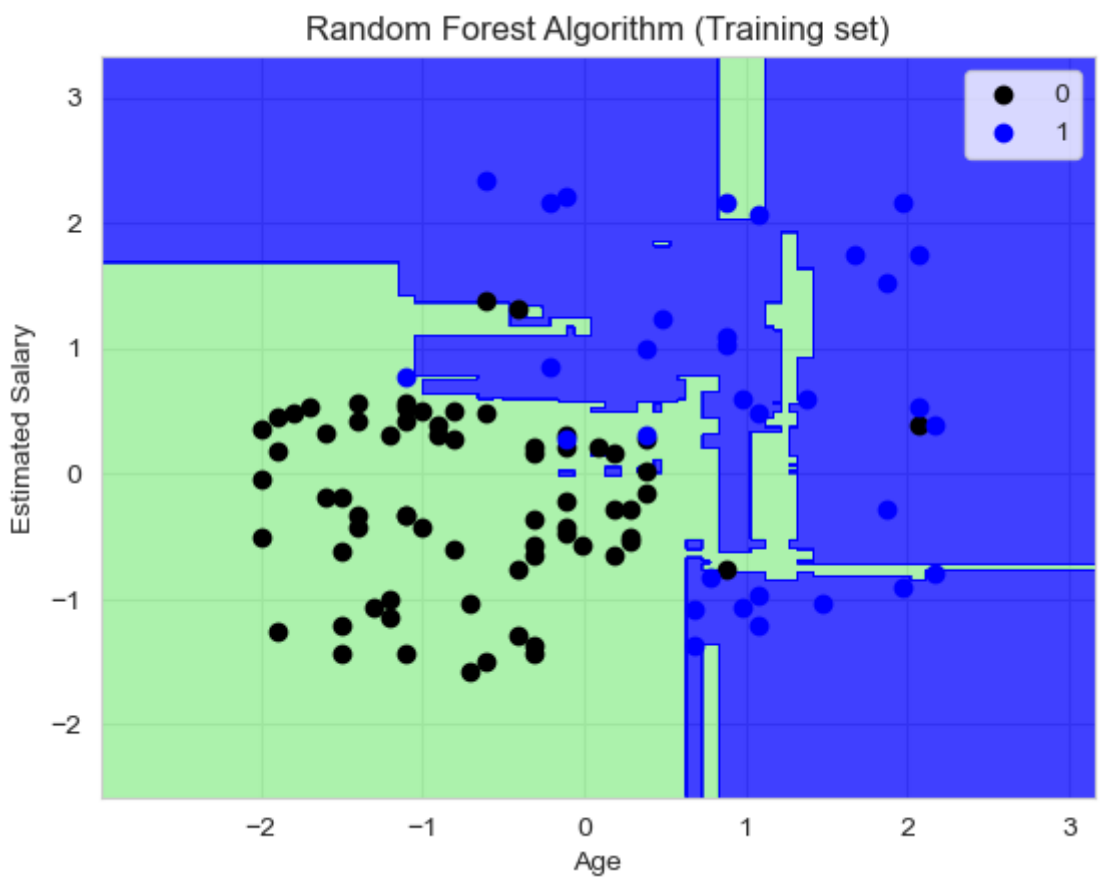
In [58]:
```
# Visualizing the training set result
from matplotlib.colors import ListedColormap
import seaborn as sns
# set the style to seaborn-whitegrid
sns.set_style("whitegrid")

x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(
    np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1, step=0.01),
    np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01)
)

plt.contourf(
    x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
    alpha=0.75, cmap=ListedColormap(('lightgreen', 'blue'))
)
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(
        x_set[y_set == j, 0], x_set[y_set == j, 1],
        c=ListedColormap(('black', 'blue'))(i), label=j
)
plt.title('Random Forest Algorithm (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pr
ecedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array w
ith a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pr
ecedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array w
ith a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [59]:
```
# Visualizing the test set result
from matplotlib.colors import ListedColormap
import seaborn as sns
# set the style to seaborn-whitegrid
sns.set_style("whitegrid")
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(
np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1, step=0.01),
np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01)
)
plt.contourf(
    x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
    alpha=0.75, cmap=ListedColormap(('lightgreen', 'blue'))
)
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(
        x_set[y_set == j, 0], x_set[y_set == j, 1],
        c=ListedColormap(('black', 'blue'))(i), label=j
    )
plt.title('Random Forest Algorithm (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pr
ecedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array w
ith a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pr
ecedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array w
ith a single row if you intend to specify the same RGB or RGBA value for all points.
```



Name:Padarthy Yagnesh Sai

Reg.No:20BCE0625

Labslot:L11+L12

Question-2

2. Implement AdaBoost algorithm and test the algorithm using any data set of your choice. The output should include Accuracy, Error rate, Precision and recall rate along with the confusion matrix.

In [5]:
```
#Importing Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,recall_score, classification_report
```

In [6]:
```
# Load the dataset
df= pd.read_csv("Iris.csv")
```

In [7]:
```
# display the first 5 rows
df.head(n=5)
```

Out[7]:

	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 [8]:
```
# Shape of the dataset
shape = df.shape
print(f"The shape of the data set is {shape}")
```

```
The shape of the data set is (150, 6)
```

In [9]:
```
#describe() method for the dataset
df.describe()
```

Out[9]:

	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 [10]:
```
# info() method for the dataset
df.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 [11]:
```
# check for missing values
df.isnull().sum()
```

Out[11]:
```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

In [12]:
```
# Extracting Independent and dependent Variable
data = df.drop('Id',axis=1)
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
print("Shape of X is %s and shape of y is %s"%(X.shape,y.shape))
```

```
Shape of X is (150, 5) and shape of y is (150,)
```

In [13]:
```
# Splitting the dataset into training and test set.
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size=0.25,random_state=28)
```

In [14]:
```
# Feature Scaling
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)
```

In [15]:
```
# Model creation
# Creating adaboost classifier model
adb = AdaBoostClassifier()
```

In [16]:
```
# Fitting the model
adb.fit(X_train,Y_train)
```

Out[16]:
```
AdaBoostClassifier()
```

In [17]:
```
# Predict output
y_pred= adb.predict(X_test)
```

In [18]:
```
# Performance metrics
# Confusion Matrix
cm = confusion_matrix(Y_test, y_pred)
print("Confusion Matrix: ")
print(cm)
# Accuracy
accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy: ", accuracy)
# Error rate
error_rate = 1 - accuracy
print("Error rate: ", error_rate)
# Precision
precision = precision_score(Y_test, y_pred, average='macro')
print("Precision: ", precision)
# Recall rate
recall = recall_score(Y_test, y_pred, average='macro')
print("Recall rate: ", recall)
# Classification Report
print("Classification Report\n")
print(classification_report(Y_test, y_pred))
```

```
Confusion Matrix:
[[11  0  0]
 [ 0 15  0]
 [ 0  0 12]]
Accuracy:  1.0
Error rate:  0.0
Precision:  1.0
Recall rate:  1.0
Classification Report

                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        15
 Iris-virginica       1.00      1.00      1.00        12

       accuracy                           1.00        38
      macro avg       1.00      1.00      1.00        38
   weighted avg       1.00      1.00      1.00        38
```