Practical No.7

Aim:- Logistic Regression and Decision Tree

(A)

- Build a Logistic Regression Model to predict a binary outcome.
- Evaluate the model's performance using classification metrics.

Logistic Regression

Import the Libraries

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

[6]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
```

Splitting the Dataset

[8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=0)
```

[9]:
```
print(X_train)
print(Y_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
```

Feature Scaling

[10]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Logistic Regression Model

[11]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, Y_train)
```

[11]:
```
LogisticRegression
LogisticRegression(random_state=0)
```

Predicting a New Result

[12]:
```
# Predicting result for certain age and salary, person will purchase or not purchase
print(classifier.predict(sc.transform([[30, 87000]])))
```

```
[0]
```

Predicting the test results

[15]:
```
y_pred = classifier.predict(X_test)
print(np.concatenate(
    (y_pred.reshape(len(y_pred), 1), Y_test.reshape(len(Y_test), 1)), 1))
```

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

Making the Confusion Matrix

[16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy = accuracy_score(Y_test, y_pred)
print(accuracy)
```

```
[[65  3]
 [ 8 24]]
0.89
```

[B]

Aim :- Construct a Decision Tree model and interpret rules for classification.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn import tree
```

In [2]:
```
iris_data = load_iris()
iris = pd.DataFrame(iris_data.data)
```

In [3]:
```
print("Features Name : ", iris_data.feature_names)
```

```
Features Name :  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [4]:
```
#features
X = iris_data.data
print(X)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
```

In [5]:
```
#target or labels
Y = iris_data.target
print(Y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
X_train,X_test,y_train,y_test = train_test_split(X,Y,random_state = 50 , test_size = 0.3)
```

In [8]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(105, 4)
(45, 4)
(105,)
(45,)
```

In [10]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=100)
clf.fit(X_train,y_train)
```

Out[10]:
```
DecisionTreeClassifier(random_state=100)
```

In [11]:
```
#Prediction on testing data
Y_pred = clf.predict(X_test)
print(Y_pred)
```

```
[1 1 0 0 2 2 2 0 0 1 0 2 0 2 1 0 1 0 1 2 2 1 0 2 1 2 1 1 1 2 2 1 1 2 0 0 1
 1 1 0 0 1 2 0 2]
```

In [12]:
```
#Accuracy of the model
print("Accuracy:",accuracy_score(y_test,Y_pred))
```

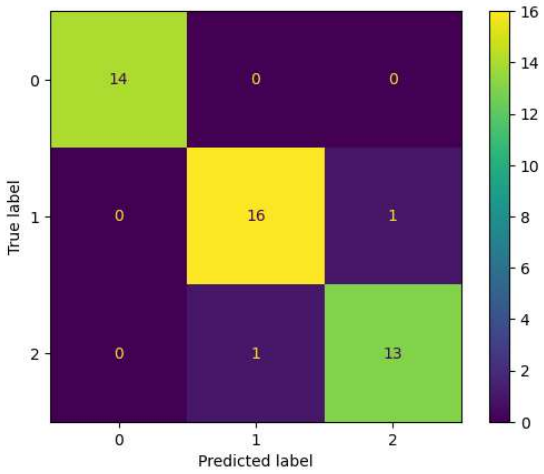
```
Accuracy: 0.9555555555555556
```

In [15]:
```
from sklearn.metrics import confusion_matrix
cm = np.array(confusion_matrix(y_test,Y_pred))
cm
```

Out[15]:
```
array([[14,  0,  0],
       [ 0, 16,  1],
       [ 0,  1, 13]], dtype=int64)
```

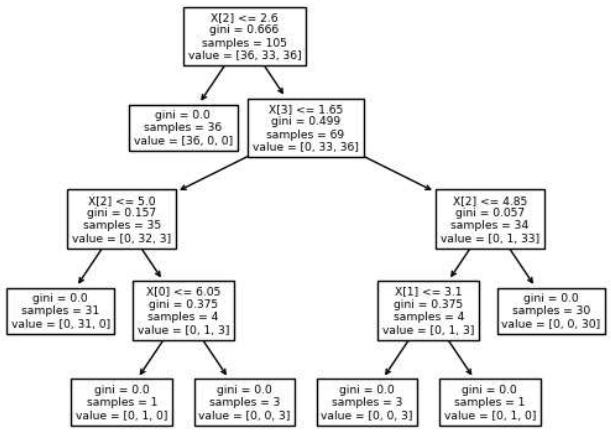
In [16]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf,X_test,y_test)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is de
precated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confu
sionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [17]:
```
tree.plot_tree(clf)
```

Out[17]:
```
[Text(0.4, 0.9, 'X[2] <= 2.6\ngini = 0.666\nsamples = 105\nvalue = [36, 33, 36]'),
 Text(0.3, 0.7, 'gini = 0.0\nsamples = 36\nvalue = [36, 0, 0]'),
 Text(0.5, 0.7, 'X[3] <= 1.65\ngini = 0.499\nsamples = 69\nvalue = [0, 33, 36]'),
 Text(0.2, 0.5, 'X[2] <= 5.0\ngini = 0.157\nsamples = 35\nvalue = [0, 32, 3]'),
 Text(0.1, 0.3, 'gini = 0.0\nsamples = 31\nvalue = [0, 31, 0]'),
 Text(0.3, 0.3, 'X[0] <= 6.05\ngini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.2, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.4, 0.1, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.8, 0.5, 'X[2] <= 4.85\ngini = 0.057\nsamples = 34\nvalue = [0, 1, 33]'),
 Text(0.7, 0.3, 'X[1] <= 3.1\ngini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.6, 0.1, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.8, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9, 0.3, 'gini = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```



Practical No.8

Aim:- K-Means Clustering

- Apply the K-Means Algorithm to group similar data points into clusters.
- Determine optimal number of clusters using elbow method or silhouette analysis.
- Visualize the clustering results and analyse the cluster characteristics.

In [7]:
```
#Importing  Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn.cluster as cluster
from sklearn.cluster import KMeans
import seaborn as sns
import sklearn.metrics as metrics
```

In [8]:
```
dataset = pd.read_csv('Iris.csv')
x = dataset.iloc[:,[0,1,2,3]].values
```

In [9]:
```
print(x)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
```

In [10]:
```
K = range(1,10)
# within-cluster-sum-of-square
wss = []
for k in K:
    kmeans=cluster.KMeans(n_clusters=k,init="k-means++")
    kmeans=kmeans.fit(x)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py
on Windows with MKL, when there are less chunks than available threac
P_NUM_THREADS=1.
  warnings.warn(
```