8.Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Load the Breast Cancer Dataset
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Labels (0: malignant, 1: benign)

# Convert to DataFrame for additional inspection
df = pd.DataFrame(data.data, columns=data.feature_names)
df['diagnosis'] = data.target

# Print dataset details
print("Head of dataset:")
print(df.head())

print("\nDataset Info:")
print(df.info())

print("\nDataset Shape:")
print(df.shape)
```

```
# Step 2: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Create and train the Decision Tree classifier using 'entropy'
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf.fit(X_train, y_train)

# Step 4: Make predictions on the test set
y_pred = clf.predict(X_test)

# Step 5: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=data.target_names))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
# Step 6: Classify a new sample
#new_sample = np.array([[17.99, 10.38, 122.8, 1001.0, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419, 0.07871,
                        # 1.095, 0.9053, 8.589, 153.4, 0.006399, 0.04904, 0.05373, 0.01587, 0.03003, 0.006193,
                         #25.38, 17.33, 184.6, 2019.0, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601, 0.1189]])
new_sample = np.array([[12.5, 19.2, 80.0, 500.0, 0.085, 0.1, 0.05, 0.02, 0.17, 0.06,0.4, 1.0, 2.5, 40.0, 0.006, 0.02, 0.03, 0
16.0, 25.0, 105.0, 900.0, 0.13, 0.25, 0.28, 0.12, 0.29, 0.08]])

prediction = clf.predict(new_sample)
print("\nNew Sample Prediction:")
print("Class:", data.target_names[prediction][0])

# Step 7: Visualize the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(clf,
          filled=True,
          feature_names=data.feature_names,
          class_names=data.target_names,
          fontsize=9)
plt.title("Decision Tree Visualization (Entropy)", fontsize=16)
plt.show()
```

9.Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training.
Compute the accuracy of the classifier, considering a few test data sets.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_olivetti_faces
data = fetch_olivetti_faces()

data.keys()

print("Data Shape:", data.data.shape)
print("Target Shape:", data.target.shape)
print("There are {} unique persons in the dataset".format(len(np.unique(data.target))))
print("Size of each image is {}x{}".format(data.images.shape[1],data.images.shape[1]))

def print_faces(images, target, top_n):
    # Ensure the number of images does not exceed available data
    top_n = min(top_n, len(images))

    # Set up figure size based on the number of images
    grid_size = int(np.ceil(np.sqrt(top_n)))
    fig, axes = plt.subplots(grid_size, grid_size, figsize=(15, 15))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.2, wspace=0.2)
```

```
    for i, ax in enumerate(axes.ravel()):
        if i < top_n:
            ax.imshow(images[i], cmap='bone')
            ax.axis('off')
            ax.text(2, 12, str(target[i]), fontsize=9, color='red')
            ax.text(2, 55, f"face: {i}", fontsize=9, color='blue')
        else:
            ax.axis('off')

    plt.show()

print_faces(data.images,data.target,400)

#let us extract unique charaters present in dataset
def display_unique_faces(pics):
    fig = plt.figure(figsize=(24, 10))  # Set figure size
    columns, rows = 10, 4  # Define grid dimensions
   # Loop through grid positions and plot each image
   for i in range(1, columns * rows + 1):
       img_index = 10 * i - 1  # Calculate the image index
       if img_index < pics.shape[0]:  # Check for valid image index
           img = pics[img_index, :, :]
           ax = fig.add_subplot(rows, columns, i)
           ax.imshow(img, cmap='gray')
           ax.set_title(f"Person {i}", fontsize=14)
           ax.axis('off')

   plt.suptitle("There are 40 distinct persons in the dataset", fontsize=24)
   plt.show()


display_unique_faces(data.images)

from sklearn.model_selection import train_test_split
X = data.data
Y = data.target
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state=46)

print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)
```

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score

# Train the model
nb = GaussianNB()
nb.fit(x_train, y_train)

# Predict the test set results
y_pred = nb.predict(x_test)

# Calculate accuracy
nb_accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)

# Display the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Display accuracy result
print(f"Naive Bayes Accuracy: {nb_accuracy}%")
```

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Initialize and fit Multinomial Naive Bayes
nb = MultinomialNB()
nb.fit(x_train, y_train)

# Predict the test set results
y_pred = nb.predict(x_test)

# Calculate accuracy
accuracy = round(accuracy_score(y_test, y_pred) * 100, 2)
print(f"Multinomial Naive Bayes Accuracy: {accuracy}%")
```

```
# Calculate the number of misclassified images
misclassified_idx = np.where(y_pred != y_test)[0]
num_misclassified = len(misclassified_idx)

# Print the number of misclassified images and accuracy
print(f"Number of misclassified images: {num_misclassified}")
print(f"Total images in test set: {len(y_test)}")
print(f"Accuracy: {round((1 - num_misclassified / len(y_test)) * 100, 2)}%")

# Visualize some of the misclassified images
n_misclassified_to_show = min(num_misclassified, 5)  # Show up to 5 misclassified images
plt.figure(figsize=(10, 5))
for i in range(n_misclassified_to_show):
    idx = misclassified_idx[i]
    plt.subplot(1, n_misclassified_to_show, i + 1)
    plt.imshow(x_test[idx].reshape(64, 64), cmap='gray')
    plt.title(f"True: {y_test[idx]}, Pred: {y_pred[idx]}")
    plt.axis('off')
plt.show()
```

10.Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer

# Load the Wisconsin Breast Cancer dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
actual_labels = data.target  # 0 = malignant, 1 = benign

print(df.head())
print(df.info())
print(df.shape)

# Standardize the dataset
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

```
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(df_scaled)
cluster_labels = kmeans.labels_

# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

# Create a DataFrame for visualization
df_visual = pd.DataFrame(df_pca, columns=['PC1', 'PC2'])
df_visual['Cluster'] = cluster_labels
df_visual['Actual'] = actual_labels

# Plot clusters and actual diagnoses
plt.figure(figsize=(12, 5))

# K-Means Clusters
plt.subplot(1, 2, 1)
sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=df_visual, palette='Set1')
plt.title('K-Means Clustering on Wisconsin Breast Cancer Dataset')

# Actual Diagnosis
plt.subplot(1, 2, 2)
sns.scatterplot(x='PC1', y='PC2', hue='Actual', data=df_visual, palette='Set2')
plt.title('Actual Diagnosis')

plt.tight_layout()
plt.show()
```