Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Tennisdata.csv

```
import csv
from collections import defaultdict

def load_data(filename):
    with open(filename, 'r') as f:
        return list(csv.DictReader(f))

def calc_probabilities(data):
    total = len(data)
    class_prob = defaultdict(int)
    cond_prob = defaultdict(lambda: defaultdict(int))

    # Calculate class probabilities and conditional probabilities
    for row in data:
        class_prob[row['PlayTennis']] += 1
        for col, val in row.items():
            if col != 'PlayTennis':
                cond_prob[(col, val)][row['PlayTennis']] += 1

    class_prob = {k: v / total for k, v in class_prob.items()}
    cond_prob = {k: {label: v / sum(d.values()) for label, v in d.items()} for k, d in cond_prob.items()}

    return class_prob, cond_prob

def predict(instance, class_prob, cond_prob):
    probs = {}
    for cls, p_cls in class_prob.items():
        prob = p_cls
        for col, val in instance.items():
            if (col, val) in cond_prob:
                prob *= cond_prob[(col, val)].get(cls, 0)
        probs[cls] = prob
    return max(probs, key=probs.get)

def accuracy(data, class_prob, cond_prob):
    correct = sum(predict({k: v for k, v in row.items() if k != 'PlayTennis'}, class_prob, cond_prob) ==
row['PlayTennis'] for row in data)
    return correct / len(data)

def naive_bayes(filename):
    data = load_data(filename)
    class_prob, cond_prob = calc_probabilities(data)
    print(f'Accuracy: {accuracy(data, class_prob, cond_prob) * 100:.2f}%')

naive_bayes('Tennisdata.csv')
```

2. You are provided with a dataset containing information about various plants with two features: Height (cm) and Width (cm). Each plant is labeled as either "Flower" or "Shrub." You need to use the K-Nearest Neighbors (K-NN) algorithm to classify a new, unlabeled plant based on its height and width.

```
import numpy as np

# Example dataset (with n rows and m columns)
X = np.array([[5, 2], [6, 3], [7, 2], [8, 3], [4, 1]])  # n=5, m=2
y = np.array(["flower", "flower", "shrub", "shrub", "flower"])

def knn_predict(X_train, y_train, test_point, k=3):
    # Vectorized computation of Euclidean distances between test_point and all training points
    distances = np.linalg.norm(X_train - test_point, axis=1)

    # Get the indices of the k smallest distances
    sorted_indices = distances.argsort()[:k]

    # Get the labels of the nearest neighbors
    nearest_labels = y_train[sorted_indices]

    # Predict the most common class among the k neighbors
    prediction = max(set(nearest_labels), key=list(nearest_labels).count)

    return prediction

# Take input for the test point
try:
    test_height = float(input("Enter the height of the plant: "))
    test_width = float(input("Enter the width of the plant: "))
    test_point = np.array([test_height, test_width])

    # Predict the class of the test point
    predicted_class = knn_predict(X, y, test_point)

    print(f"Predicted Class for the plant (Height: {test_height}, Width: {test_width}):
{predicted_class}")

except ValueError:
    print("Invalid input. Please enter numeric values for height and width.")
```

6. Construct decision tree also display the information gain for sunny, overcast and rain.

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

```
import math

# Dataset
# Format: [Outlook, Temperature, Humidity, Windy, PlayTennis]
data = [
    ["sunny", "hot", "high", False, "no"],
    ["sunny", "hot", "high", True, "no"],
    ["overcast", "hot", "high", False, "yes"],
    ["rain", "mild", "high", False, "yes"],
    ["rain", "cool", "normal", False, "yes"],
    ["rain", "cool", "normal", True, "no"],
    ["overcast", "cool", "normal", True, "yes"],
    ["sunny", "mild", "high", False, "no"],
    ["sunny", "cool", "normal", False, "yes"],
    ["rain", "mild", "normal", False, "yes"],
    ["sunny", "mild", "normal", True, "yes"],
    ["overcast", "mild", "high", True, "yes"],
    ["overcast", "hot", "normal", False, "yes"],
    ["rain", "mild", "high", True, "no"]
]

# Calculate entropy
def entropy(labels):
    total = len(labels)
    counts = {label: labels.count(label) for label in set(labels)}
    return -sum((count / total) * math.log2(count / total) for count in counts.values())

# Information gain calculation
def information_gain(data, attribute_index, target_index):
    total_entropy = entropy([row[target_index] for row in data])
    values = set(row[attribute_index] for row in data)
    weighted_entropy = 0
    for value in values:
```

```
        subset = [row for row in data if row[attribute_index] == value]
        subset_labels = [row[target_index] for row in subset]
        subset_entropy = entropy(subset_labels)
        weighted_entropy += (len(subset) / len(data)) * subset_entropy
    return total_entropy - weighted_entropy

# Display information gain for 'Outlook' (attribute index 0)
attributes = ["Outlook", "Temperature", "Humidity", "Windy"]
target_index = -1  # 'PlayTennis' is the target column

print("Information Gain for attributes:")
for i, attribute in enumerate(attributes):
    gain = information_gain(data, i, target_index)
    print(f"{attribute}: {gain:.4f}")
```

3 . Apply PCA to reduce the dimensionality to 1 component, and visualise the result in a 2D scatter plot.

Sample	Feature_1	Feature_2
1	5.702	4.386
2	9.884	1.020
3	2.089	1.613
4	6.531	2.533
5	4.663	2.444
6	1.590	1.104
7	6.563	1.382
8	1.966	3.687
9	8.210	0.971
10	8.379	0.961

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

# Sample data
data = np.array([
    [5.702, 4.386],
    [9.884, 1.020],
    [2.089, 1.613],
    [6.531, 2.533],
    [4.663, 2.444],
    [1.590, 1.104],
    [6.563, 1.382],
    [1.966, 3.687],
    [8.210, 0.971],
    [8.379, 0.961],
])

# PCA function
def pca_manual(data, n_components=1):
    # Step 1: Center the data
    mean_vec = np.mean(data, axis=0)
    centered_data = data - mean_vec

    # Step 2: Calculate the covariance matrix
```

```
    cov_matrix = np.cov(centered_data.T)

    # Step 3: Calculate eigenvalues and eigenvectors
    eig_values, eig_vectors = np.linalg.eig(cov_matrix)

    # Step 4: Sort eigenvectors by eigenvalues in descending order
    sorted_indices = np.argsort(eig_values)[::-1]
    eig_vectors = eig_vectors[:, sorted_indices]
    eig_values = eig_values[sorted_indices]

    # Step 5: Project the data onto the top n_components eigenvectors
    reduced_data = centered_data @ eig_vectors[:, :n_components]
    return reduced_data, eig_vectors[:, :n_components]

# Reduce to 1 dimension
reduced_data, top_components = pca_manual(data, n_components=1)

# Plot the reduced data
plt.scatter(data[:, 0], data[:, 1], color='blue', label='Original Data')
plt.scatter(reduced_data, np.zeros(len(reduced_data)), color='red', label='PCA Reduced Data')
plt.axhline(0, color='black', linewidth=0.5)
plt.title('PCA: Original Data vs Reduced Data')
plt.legend()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

4. Classify the retinal diseases using CNN. USE dataset from this :
https://www.kaggle.com/code/muhammadfaizan65/retinal-disease-classification

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set up ImageDataGenerator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Set up directories for training and testing
```

```
train_dir = '/path_to_train_data'
test_dir = '/path_to_test_data'

# Prepare data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
test_loss, test_acc = model.evaluate(test_generator, verbose=2)
print(f"Test accuracy: {test_acc:.2f}")
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.legend(loc='upper right')
plt.show()

model.save('retinal_disease_classifier.h5')

model = tf.keras.models.load_model('retinal_disease_classifier.h5')
```

5. Apply gradient on a simple linear regression (single variable). It takes a set of 15 data points and iteratively updates the parameters to minimise the mean squared error

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

# Sample data
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
y = np.array([3, 4, 2, 5, 6, 7, 8, 6, 10, 9, 11, 14, 13, 16, 15])
```

```
# Initialize parameters
m = 0  # Slope
b = 0  # Intercept
learning_rate = 0.01
epochs = 1000

# Gradient descent
for epoch in range(epochs):
    y_pred = m * X + b
    error = y - y_pred
    m_gradient = -(2 / len(X)) * np.sum(X * error)
    b_gradient = -(2 / len(X)) * np.sum(error)
    m -= learning_rate * m_gradient
    b -= learning_rate * b_gradient

# Final parameters
print(f"Final slope (m): {m}, intercept (b): {b}")

# Plot the results
plt.scatter(X, y, color="blue", label="Original Data")
plt.plot(X, m * X + b, color="red", label="Linear Regression Fit")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Gradient Descent for Linear Regression")
plt.legend()
plt.show()
```