```
import csv
import random
import math
import operator

def safe_div(x,y):
 if y == 0:
  return 0
 return x/y

# 1.Data Handling
# 1.1 Loading the Data from csv file of ConceptLearning dataset.
def loadCsv(filename):
  lines = csv.reader(open(filename))
  dataset = list(lines)
  for i in range(len(dataset)):
    dataset[i] = [float(x) for x in dataset[i]]
  return dataset


#1.2 Splitting the Data set into Training Set
def splitDataset(dataset, splitRatio):
  trainSize = int(len(dataset) * splitRatio)
  trainSet = []
  copy = list(dataset)
  i=0
  while len(trainSet) < trainSize:
  #index = random.randrange(len(copy))
    trainSet.append(copy.pop(i))
  return [trainSet, copy]


#2.Summarize Data
#The naive bayes model is comprised of a
#summary of the data in the training dataset.
#This summary is then used when making predictions.
#involves the mean and the standard deviation for each attribute, by class value

#2.1: Separate Data By Class
#Function to categorize the dataset in terms of classes
#The function assumes that the last attribute (-1) is the class value.
#The function returns a map of class values to lists of data instances.

def separateByClass(dataset):
        separated = {}
        for i in range(len(dataset)):
                vector = dataset[i]
                if (vector[-1] not in separated):
                        separated[vector[-1]] = []
                separated[vector[-1]].append(vector)
        return separated

#The mean is the central middle or central tendency of the data,
# and we will use it as the middle of our gaussian distribution
# when calculating probabilities

#2.2 : Calculate Mean
```

```
def mean(numbers):
  return safe_div(sum(numbers),float(len(numbers)))

#The standard deviation describes the variation of spread of the data,
#and we will use it to characterize the expected spread of each attribute
#in our Gaussian distribution when calculating probabilities.

#2.3 : Calculate Standard Deviation
def stdev(numbers):
  avg = mean(numbers)
  variance = safe_div(sum([pow(x-avg,2) for x in numbers]),float(len(numbers)-1))
  return math.sqrt(variance)

#2.4 : Summarize Dataset
#Summarize Data Set for a list of instances (for a class value)
#The zip function groups the values for each attribute across our data instances
#into their own lists so that we can compute the mean and standard deviation values
#for the attribute.

def summarize(dataset):
  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
  del summaries[-1]
  return summaries

#2.5 : Summarize Attributes By Class
#We can pull it all together by first separating our training dataset into
#instances grouped by class.Then calculate the summaries for each attribute.

def summarizeByClass(dataset):
  separated = separateByClass(dataset)
  summaries = {}
  for classValue, instances in separated.items():
    summaries[classValue] = summarize(instances)
  print("Summarize Attributes By Class")
  print(summaries)
  print(" ")
  return summaries

#3.Make Prediction
#3.1 Calculate Probaility Density Function
def calculateProbability(x, mean, stdev):
  exponent = math.exp(-safe_div(math.pow(x-mean,2),(2*math.pow(stdev,2))))
  final = safe_div(1 , (math.sqrt(2*math.pi) * stdev)) * exponent
  return final

#3.2 Calculate Class Probabilities
def calculateClassProbabilities(summaries, inputVector):
  probabilities = {}
  for classValue, classSummaries in summaries.items():
   probabilities[classValue] = 1
  for i in range(len(classSummaries)):
    mean, stdev = classSummaries[i]
    x = inputVector[i]
    probabilities[classValue] *= calculateProbability(x, mean, stdev)
  return probabilities

#3.3 Prediction : look for the largest probability and return the associated class
```

```
def predict(summaries, inputVector):
  probabilities = calculateClassProbabilities(summaries, inputVector)
  bestLabel, bestProb = None, -1
  for classValue, probability in probabilities.items():
    if bestLabel is None or probability > bestProb:
      bestProb = probability
      bestLabel = classValue
  return bestLabel

#4.Make Predictions
# Function which return predictions for list of predictions
# For each instance
def getPredictions(summaries, testSet):
  predictions = []
  for i in range(len(testSet)):
    result = predict(summaries, testSet[i])
    predictions.append(result)
  return predictions

#5. Computing Accuracy
def getAccuracy(testSet, predictions):
  correct = 0
  for i in range(len(testSet)):
    if testSet[i][-1] == predictions[i]:
      correct += 1
  accuracy = safe_div(correct,float(len(testSet))) * 100.0
  return accuracy

def main():
  filename = 'diabetes2.csv'
  splitRatio = 0.9
  dataset = loadCsv(filename)
  trainingSet, testSet = splitDataset(dataset, splitRatio)
  print('Split {0} rows into'.format(len(dataset)))
  print('Number of Training data: ' + (repr(len(trainingSet))))
  print('Number of Test Data: ' + (repr(len(testSet))))
  print("\nThe values assumed for the concept learning attributes are\n")
  print("OUTLOOK=> Sunny=1 Overcast=2 Rain=3\nTEMPERATURE=> Hot=1 Mild=2 Cool=3\nHUMIDITY=> High=1
  print("TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5")
  print("\nThe Training set are:")
  for x in trainingSet:
    print(x)
    print("\nThe Test data set are:")
  for x in testSet:
    print(x)
  print("\n")

# prepare model
  summaries = summarizeByClass(trainingSet)

# test model
  predictions = getPredictions(summaries, testSet)
  actual = []
  for i in range(len(testSet)):
   vector = testSet[i]
  actual.append(vector[-1])
```

```
# Since there are five attribute values, each attribute constitutes to 20% accuracy. So if all attr
#match with predictions then 100% accuracy
  print('Actual values: {0}%'.format(actual))
  print('Predictions: {0}%'.format(predictions))
  accuracy = getAccuracy(testSet, predictions)
  print('Accuracy: {0}%'.format(accuracy))

main()
```