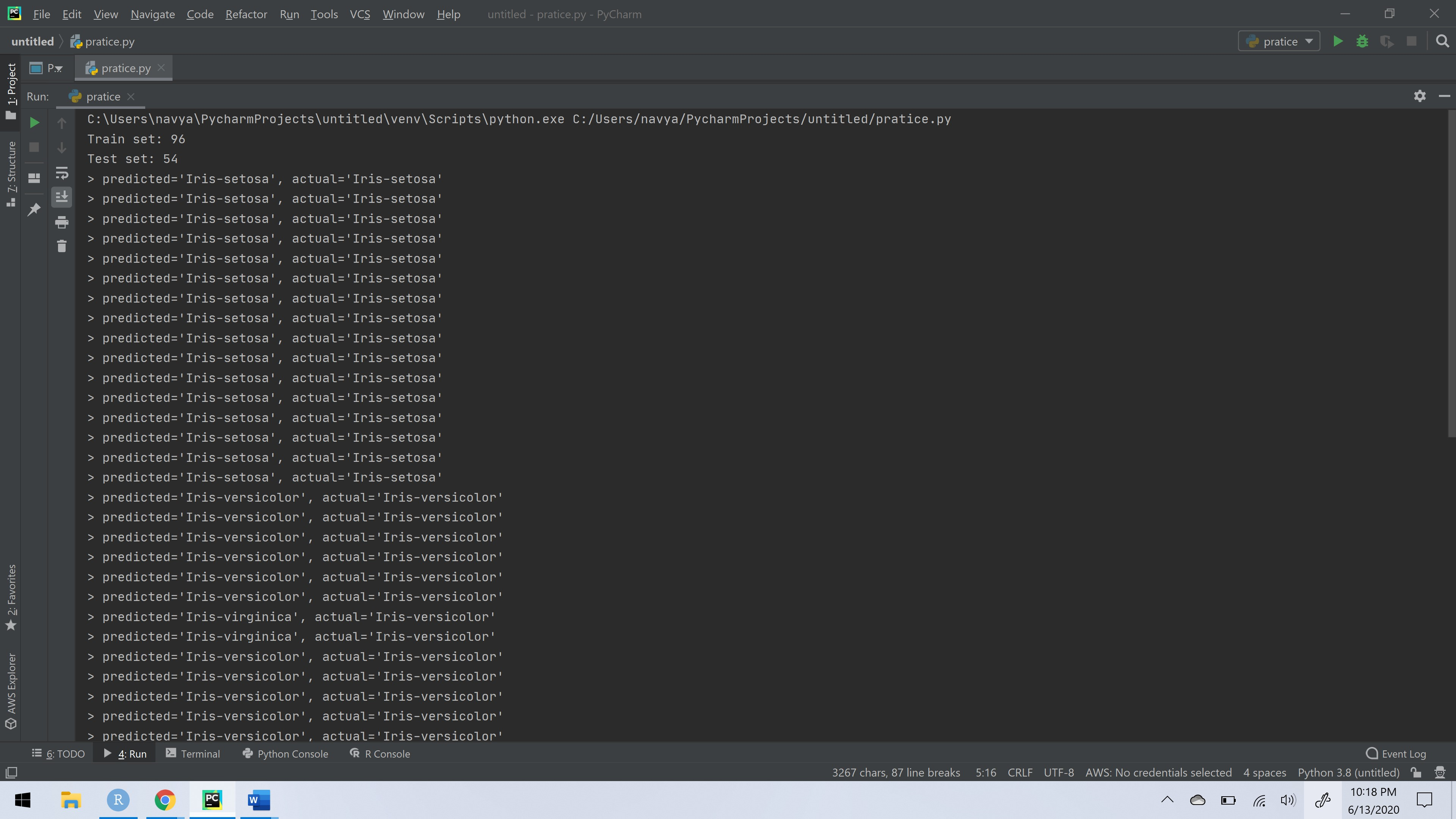
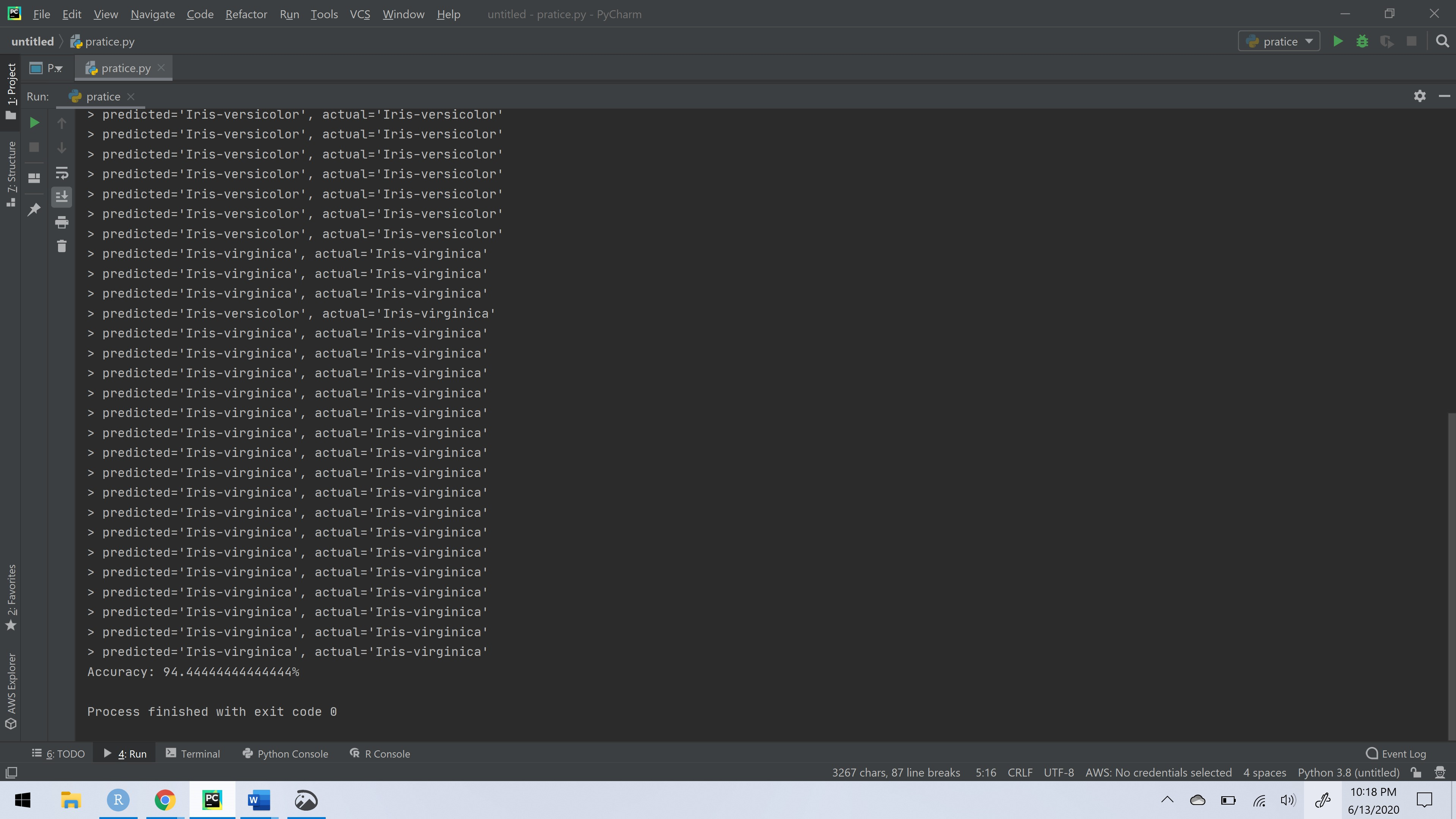
1. **One different runs, you’ll get different percentage values. Can you explain why?**

Yes, it is because of the data division between test and train dataset. That is the reason we are getting the different accuracy for different runs.

# kNN implementation in Python 3  
import csv  
import random  
import math  
import operator  
import urllib.request  
# Old Way Commented-Out  
### FUNCTION GET LINES: Get the files, either directly online or by saving it locally:  
def getLines(filename):  
 lines = []  
 if (filename.startswith(('http', 'ftp', 'sftp')) ):  
 # Skip downloading it and open directly online:  
 response = urllib.request.urlopen(filename)  
 lines = csv.reader(response.read().decode('utf-8').splitlines())  
 else:  
 # TutorialsPoint IDE requires 'r', not 'rb'  
 with open(filename, 'r') as csvfile:  
 # csvreader is an object that is essentially a list of lists  
 csvreader = csv.reader(csvfile)  
 for line in csvreader:  
 lines.append(line)  
 return lines  
def loadDatasetFinal(filename, split, trainingSet=[] , testSet=[]):  
 lines = getLines(filename)  
 #for row in lines:  
 # print (', '.join(row))  
 dataset = list(lines)  
 for x in range(len(dataset)-1):  
 for y in range(4): # legend = ('Sepal length', 'Sepal width', 'Petal length', 'Petal width')  
 dataset[x][y] = float(dataset[x][y])  
 if random.random() < split:  
 trainingSet.append(dataset[x])  
 else:  
 testSet.append(dataset[x])  
def euclideanDistance(instance1, instance2, length):  
 distance = 0  
 for x in range(length):  
 distance += pow((instance1[x] - instance2[x]), 2)  
 return math.sqrt(distance)  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance)-1  
 for x in range(len(trainingSet)):  
 dist = euclideanDistance(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct/float(len(testSet))) \* 100.0  
def main():  
 # set our parameters  
 k = 3  
 split = 0.67  
 filename = 'iris.data.csv'  
 filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'  
 # prepare data  
 trainingSet=[]  
 testSet=[]  
 loadDatasetFinal(filename, split, trainingSet, testSet)  
 print('Train set: ' + repr(len(trainingSet)))  
 print('Test set: ' + repr(len(testSet)))  
 # generate predictions  
 predictions=[]  
 for x in range(len(testSet)):  
 neighbors = getNeighbors(trainingSet, testSet[x], k)  
 result = getResponse(neighbors)  
 predictions.append(result)  
 print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))  
 accuracy = getAccuracy(testSet, predictions)  
 print('Accuracy: ' + repr(accuracy) + '%')  
main()

Output:





1. **Can you add a wrapper to this function so that it computers multiple runs(a user-specified value) and computes an average accuracy over the multiple runs rather than a single accuracy?**

# kNN implementation in Python 3  
import csv  
import random  
import math  
import operator  
import urllib.request  
# Old Way Commented-Out  
### FUNCTION GET LINES: Get the files, either directly online or by saving it locally:  
def getLines(filename):  
 lines = []  
 if (filename.startswith(('http', 'ftp', 'sftp')) ):  
 # Skip downloading it and open directly online:  
 response = urllib.request.urlopen(filename)  
 lines = csv.reader(response.read().decode('utf-8').splitlines())  
 else:  
 # TutorialsPoint IDE requires 'r', not 'rb'  
 with open(filename, 'r') as csvfile:  
 # csvreader is an object that is essentially a list of lists  
 csvreader = csv.reader(csvfile)  
 for line in csvreader:  
 lines.append(line)  
 return lines  
def loadDatasetFinal(filename, split, trainingSet=[] , testSet=[]):  
 lines = getLines(filename)  
 #for row in lines:  
 # print (', '.join(row))  
 dataset = list(lines)  
 for x in range(len(dataset)-1):  
 for y in range(4):  
 # legend = ('Sepal length', 'Sepal width', 'Petal length', 'Petal width')  
 dataset[x][y] = float(dataset[x][y])  
 if random.random() < split:  
 trainingSet.append(dataset[x])  
 else:  
 testSet.append(dataset[x])  
def euclideanDistance(instance1, instance2, length):  
 distance = 0  
 for x in range(length):  
 distance += pow((instance1[x] - instance2[x]), 2)  
 return math.sqrt(distance)  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance)-1  
 for x in range(len(trainingSet)):  
 dist = euclideanDistance(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct/float(len(testSet))) \* 100.0  
def main():  
 # set our parameters  
 listOfAccuracy = []  
 noIterations = int(input("Enter no of Interations: "))  
 k = 3  
 split = 0.67  
 filename = 'iris.data.csv'  
 filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'  
 for y in range(noIterations):  
 # prepare data  
 trainingSet=[]  
 testSet=[]  
 loadDatasetFinal(filename, split, trainingSet, testSet)  
 print('Train set: ' + repr(len(trainingSet)))  
 print('Test set: ' + repr(len(testSet)))  
 # generate predictions  
 predictions=[]  
 for x in range(len(testSet)):  
 neighbors = getNeighbors(trainingSet, testSet[x], k)  
 result = getResponse(neighbors)  
 predictions.append(result)  
 accuracy = getAccuracy(testSet, predictions)  
 listOfAccuracy.append(accuracy)  
 print("The accuracies for " + repr(noIterations) + ' Iterations are' + repr(listOfAccuracy))  
 avg = sum(listOfAccuracy) / noIterations  
 print('the Average Accuracy is: ' + repr(avg))  
main()

**OUTPUT:**

Enter no of Interations: 7

Train set: 112

Test set: 38

Train set: 95

Test set: 55

Train set: 99

Test set: 51

Train set: 106

Test set: 44

Train set: 102

Test set: 48

Train set: 100

Test set: 50

Train set: 103

Test set: 47

The accuracies for 7 Iterations are[89.47368421052632, 98.18181818181819, 96.07843137254902, 93.18181818181817, 97.91666666666666, 96.0, 95.74468085106383]

the Average Accuracy is: 95.22529992349175

1. Implementation of alternative Distance Measure:

# kNN implementation in Python 3  
import csv  
import random  
import math  
import operator  
import urllib.request  
  
# Old Way Commented-Out  
### FUNCTION GET LINES: Get the files, either directly online or by saving it locally:  
def vecDotprod(instance1,instance2,length):  
 distance = 0  
 for x in range(length):  
 distance += ((instance1[x] \* instance2[x]))  
 return distance  
def manhattandistance(instance1,instance2,length):  
 distance = 0  
 for x in range(length):  
 distance += abs((instance1[x] -instance2[x]))  
 return distance  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance)-1  
 for x in range(len(trainingSet)):  
 dist = manhattandistance(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct/float(len(testSet))) \* 100.0  
def getLines(filename):  
 lines = []  
 if (filename.startswith(('http', 'ftp', 'sftp')) ):  
 # Skip downloading it and open directly online:  
 response = urllib.request.urlopen(filename)  
 lines = csv.reader(response.read().decode('utf-8').splitlines())  
 else:  
 # TutorialsPoint IDE requires 'r', not 'rb'  
 with open(filename, 'r') as csvfile:  
 # csvreader is an object that is essentially a list of lists  
 csvreader = csv.reader(csvfile)  
 for line in csvreader:  
 lines.append(line)  
 return lines  
def loadDatasetFinal(filename, split, trainingSet=[] , testSet=[]):  
 lines = getLines(filename)  
 dataset = list(lines)  
 for x in range(len(dataset)-1):  
 for y in range(4):  
 # legend = ('Sepal length', 'Sepal width', 'Petal length', 'Petal width')  
 dataset[x][y] = float(dataset[x][y])  
 if random.random() < split:  
 trainingSet.append(dataset[x])  
 else:  
 testSet.append(dataset[x])  
  
def main():  
 # set our parameters  
 listOfAccuracy = []  
 noIterations = int(input("Enter no of Interations: "))  
 print('Calculating distance - manhattandistance')  
 k = 3  
 split = 0.67  
 filename = 'iris.data.csv'  
 filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'  
 # prepare data  
 for i in range(noIterations):  
 trainingSet=[]  
 testSet=[]  
 loadDatasetFinal(filename, split, trainingSet, testSet)  
 print('Train set: ' + repr(len(trainingSet)))  
 print('Test set: ' + repr(len(testSet)))  
 # generate predictions  
 predictions = []  
 for x in range(len(testSet)):  
 neighbors = getNeighbors(trainingSet, testSet[x], k)  
 result = getResponse(neighbors)  
 predictions.append(result)  
 accuracy = getAccuracy(testSet, predictions)  
 print('weighted Accuracy: ' + repr(accuracy) + '%')  
 listOfAccuracy.append(accuracy)  
 print("The accuracies for " + repr(noIterations) + ' Iterations are ' + repr(listOfAccuracy))  
 avg = sum(listOfAccuracy) / len(listOfAccuracy)  
 print('The Average Accuracy is: ' + repr(avg))  
main()

Output:

Enter no of Interations: 7

Calculating distance - manhattandistance

Train set: 112

Test set: 38

weighted Accuracy: 97.36842105263158%

Train set: 102

Test set: 48

weighted Accuracy: 97.91666666666666%

Train set: 109

Test set: 41

weighted Accuracy: 87.8048780487805%

Train set: 101

Test set: 49

weighted Accuracy: 95.91836734693877%

Train set: 101

Test set: 49

weighted Accuracy: 97.95918367346938%

Train set: 102

Test set: 48

weighted Accuracy: 95.83333333333334%

Train set: 87

Test set: 63

weighted Accuracy: 95.23809523809523%

The accuracies for 7 Iterations are [97.36842105263158, 97.91666666666666, 87.8048780487805, 95.91836734693877, 97.95918367346938, 95.83333333333334, 95.23809523809523]

1. The Average Accuracy is: 95.4341350514165Distance-weight contribution in KNN

# kNN implementation in Python 3  
import csv  
import random  
import math  
import operator  
import urllib.request  
  
# Old Way Commented-Out  
### FUNCTION GET LINES: Get the files, either directly online or by saving it locally:  
def vecDotprod(instance1,instance2,length):  
 distance = 0  
 for x in range(length):  
 distance += ((instance1[x] \* instance2[x]))  
 return distance  
def manhattandistance(instance1,instance2,length):  
 distance = 0  
 for x in range(length):  
 distance += abs((instance1[x] -instance2[x]))  
 return distance  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance)-1  
 for x in range(len(trainingSet)):  
 dist = vecDotprod(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct/float(len(testSet))) \* 100.0  
def getLines(filename):  
 lines = []  
 if (filename.startswith(('http', 'ftp', 'sftp')) ):  
 # Skip downloading it and open directly online:  
 response = urllib.request.urlopen(filename)  
 lines = csv.reader(response.read().decode('utf-8').splitlines())  
 else:  
 # TutorialsPoint IDE requires 'r', not 'rb'  
 with open(filename, 'r') as csvfile:  
 # csvreader is an object that is essentially a list of lists  
 csvreader = csv.reader(csvfile)  
 for line in csvreader:  
 lines.append(line)  
 return lines  
def loadDatasetFinal(filename, split, trainingSet=[] , testSet=[]):  
 lines = getLines(filename)  
 dataset = list(lines)  
 for x in range(len(dataset)-1):  
 for y in range(4):  
 # legend = ('Sepal length', 'Sepal width', 'Petal length', 'Petal width')  
 dataset[x][y] = float(dataset[x][y])  
 if random.random() < split:  
 trainingSet.append(dataset[x])  
 else:  
 testSet.append(dataset[x])  
  
def main():  
 # set our parameters  
 listOfAccuracy = []  
 noIterations = int(input("Enter no of Interation: "))  
 print('Calculating distance - manhattandistance')  
 k = 3  
 split = 0.67  
 filename = 'iris.data.csv'  
 filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'  
 # prepare data  
 for i in range(noIterations):  
 trainingSet=[]  
 testSet=[]  
 loadDatasetFinal(filename, split, trainingSet, testSet)  
 print('Train set: ' + repr(len(trainingSet)))  
 print('Test set: ' + repr(len(testSet)))  
 # generate predictions  
 predictions = []  
 for x in range(len(testSet)):  
 neighbors = getNeighbors(trainingSet, testSet[x], k)  
 result = getResponse(neighbors)  
 predictions.append(result)  
 accuracy = getAccuracy(testSet, predictions)  
 print('weighted Accuracy: ' + repr(accuracy) + '%')  
 listOfAccuracy.append(accuracy)  
 print("The accuracies for " + repr(noIterations) + ' Iterations are ' + repr(listOfAccuracy))  
 avg = sum(listOfAccuracy) / len(listOfAccuracy)  
 print('The Average Accuracy is: ' + repr(avg))  
main()

**OutPut:**

Enter no of Interation: 7

Calculating distance - manhattandistance

Train set: 92

Test set: 58

weighted Accuracy: 36.206896551724135%

Train set: 96

Test set: 54

weighted Accuracy: 31.48148148148148%

Train set: 95

Test set: 55

weighted Accuracy: 41.81818181818181%

Train set: 96

Test set: 54

weighted Accuracy: 35.18518518518518%

Train set: 94

Test set: 56

weighted Accuracy: 30.357142857142854%

Train set: 101

Test set: 49

weighted Accuracy: 30.612244897959183%

Train set: 92

Test set: 58

weighted Accuracy: 32.758620689655174%

The accuracies for 7 Iterations are [36.206896551724135, 31.48148148148148, 41.81818181818181, 35.18518518518518, 30.357142857142854, 30.612244897959183, 32.758620689655174]

The Average Accuracy is: 34.05996478304712

1. Implementation of KNN in sklearn:

# kNN implementation in Python 3  
import csv  
import random  
import math  
import operator  
import urllib.request  
#from sklearn.neighbors import KNeighborsClassifier  
#import sklearn  
from sklearn.neighbors import KNeighborsClassifier  
  
# Old Way Commented-Out  
### FUNCTION GET LINES: Get the files, either directly online or by saving it locally:  
def getLines(filename):  
 lines = []  
 if (filename.startswith(('http', 'ftp', 'sftp')) ):  
 # Skip downloading it and open directly online:  
 response = urllib.request.urlopen(filename)  
 lines = csv.reader(response.read().decode('utf-8').splitlines())  
 else:  
 # TutorialsPoint IDE requires 'r', not 'rb'  
 with open(filename, 'r') as csvfile:  
 # csvreader is an object that is essentially a list of lists  
 csvreader = csv.reader(csvfile)  
 for line in csvreader:  
 lines.append(line)  
 return lines  
def loadDatasetFinal(filename, split, trainingSet=[] , testSet=[]):  
 lines = getLines(filename)  
 dataset = list(lines)  
 for x in range(len(dataset)-1):  
 for y in range(4):  
 dataset[x][y] = float(dataset[x][y])  
 if random.random() < split:  
 trainingSet.append(dataset[x])  
 else:  
 testSet.append(dataset[x])  
def euclideanDistance(instance1, instance2, length):  
 distance = 0  
 for x in range(length):  
 distance += pow((instance1[x] - instance2[x]), 2)  
 return math.sqrt(distance)  
def vecDotprod(instance1,instance2,length):  
 distance = 0  
 for x in range(length):  
 distance += ((instance1[x] \* instance2[x]))  
 return distance  
def manhattandistance(instance1,instance2,length):  
 distance = 0  
 for x in range(length):  
 distance += abs((instance1[x] -instance2[x]))  
 return distance  
def getNeighbors(trainingSet, testInstance, k):  
 distances = []  
 length = len(testInstance)-1  
 for x in range(len(trainingSet)):  
 dist = euclideanDistance(testInstance, trainingSet[x], length)  
 distances.append((trainingSet[x], dist))  
 distances.sort(key=operator.itemgetter(1))  
 neighbors = []  
 for x in range(k):  
 neighbors.append(distances[x][0])  
 return neighbors  
def getResponse(neighbors):  
 classVotes = {}  
 for x in range(len(neighbors)):  
 response = neighbors[x][-1]  
 if response in classVotes:  
 classVotes[response] += 1  
 else:  
 classVotes[response] = 1  
 sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)  
 return sortedVotes[0][0]  
def getAccuracy(testSet, predictions):  
 correct = 0  
 for x in range(len(testSet)):  
 if testSet[x][-1] == predictions[x]:  
 correct += 1  
 return (correct/float(len(testSet))) \* 100.0  
def main():  
 # set our parameters  
 split = 0.67  
 filename = "iris.data.csv"  
 filename = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'  
 trainingSet = []  
 testSet = []  
 loadDatasetFinal(filename, split, trainingSet, testSet)  
 print('Train set: ' + repr(len(trainingSet)))  
 print('Test set: ' + repr(len(testSet)))  
 learnset\_data = list(i[0:3] for i in trainingSet)  
 learnset\_labels = list(i[3] for i in trainingSet)  
 testset\_data = list(i[0:3] for i in testSet)  
 knn = KNeighborsClassifier()  
 knn.fit(learnset\_data, learnset\_labels)  
 KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric = 'minkowski', metric\_params=None, n\_jobs=1, n\_neighbors=3, p=2, weights='uniform')  
 predictions = knn.predict(testset\_data)  
 accuracy = getAccuracy(testset\_labels, predictions)  
 print('Sklearn Accuracy: ' + repr(accuracy) + '%')  
main()

**OUTPUT:**

Train set: 111

Test set: 39

SKlearn Accuracy: 97.43589743589743%