Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Προπτυχιακό Μάθημα: «Διαχείριση Σύνθετων Δεδομένων»

Assignment 1

Vasilis Nikas **AM: 3143**

Μέρος 1:

Αρχικά διαβάζοντας το csv αρχείο που θέλουμε να επεξεργαστούμε, ελέγχουμε την πρώτη γραμμή όπου υπάρχουν τα πεδία και βρίσκουμε το πεδίο που μας ενδιαφέρει. Στην περιπτωσή μας το πεδίο αυτό είναι το 'Income'. Έπειτα, διαβάζουμε όλες τις εγγραφές και αποθηκεύουμε τις τίμες από το πεδίο 'Income' σε έναν πίνακα (απορρίπτουμε τα κελιά που δεν έχουν τιμή).

Χρησιμοποιούμε τον αλγόριθμο quicksort για να ταξινομήσουμε τον πίνακα. Ο αλγόριθμος quicksort έχει μέση πολυπλοκότητα O(n*logn), όπου n είναι το μέγεθος του πίνακα. Η χειρότερη περίπτωση του αλγόριθμου είναι όταν πάμε να τον χρησιμοποιήσουμε σε έναν πίνακα με ήδη ταξινομημένα στοιχεία. Οπότε στην περιπτωσή μας είναι μια πάρα πολύ καλή λύση. Επιπλέον αποθηκεύουμε σε δύο μεταβλητές την μέγιστη και την ελάχιστη τιμή του πίνακα μας.

Ο κώδικας για την υλοποίηση του αλγόριθμου quicksort είναι ο εξής:

```
def partition(array, low, high):
    # choose the rightmost element as pivot
    pivot = array[high]
    # pointer for greater element
    i = low - 1

    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:
            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

            # Swapping element at i with element at j
            (array[i], array[j]) = (array[j], array[i])

    # Swap the pivot element with the greater element specified by i
    (array[i + 1], array[high]) = (array[high], array[i + 1])

    # Return the position from where partition is done
    return i + 1
```

```
def quickSort(array, low, high):
    if low < high:
        partitiontoFind = partition(array, low, high)

        # Recursive call on the left of partitiontoFind
        quickSort(array, low, partitiontoFind - 1)

        # Recursive call on the right of partitiontoFind
        quickSort(array, partitiontoFind + 1, high)
```

Στην συνέχεια υλοποιούμε τις συναρτήσεις για το equiwidth και equidepth ιστόγραμμα. Αρχικά για το equiwidth ιστόγραμμα βρίσκουμε το κατάλληλο εύρος που θα πρέπει να έχει το κάθε bin σύμφωνα με το bin size δηλαδή τον αριθμό των bins, που στην περιπτωσή μας είναι 100. Αποθηκεύουμε αυτά τα ranges σε έναν πίνακα και στην συνέχεια βρίσκουμε πόσες εγγραφές έχει κάθε εύρος από αυτά τα ranges. Αποθήκευουμε αυτές τις τιμές σε έναν άλλο πίνακα και τον επιστρέφουμε στο τέλος στην κλήση της συνάρτησης.

Ο κώδικας για το equiwidth:

```
def equiWidth(ranges, min, max, results):
    binsRange = (max - min)/binSize
    currentRange = min
    numtuples = []

    while currentRange <= max:
        ranges.append(currentRange)
        currentRange += binsRange
        currentRange = round(currentRange,2) ###

    counter = 0
    i = 1
    #for i in range(len(ranges) - 1):
    for j in results:
        if i < len(ranges) and j < ranges[i]:
            counter += 1
        else:
            if i <= len(ranges):
                numtuples.append(counter)
            i += 1
            counter = 1
            #j -= 1

    return numtuples
```

Στο equidepth ιστόγραμμα αρχικά υπολογίζουμε πόσες τιμές θα πρέπει να έχει το κάθε bin. Επειδή αυτός ο αριθμός θα πρέπει να είναι ίδιος για κάθε bin, στην περίπτωση που το τελευταίο bin θα έχει περισσότερες εγγραφές προτειμούμε να βγάλουμε εκτός αυτές τις εγγραφές και να κρατήσουμε ίσο τον αριθμό των bin σε όλα τα bin. Μετράμε τις εγγραφές που διαβάζουμε και μόλις φτάσουμε στον αριθμό που επιθυμούμε έχουμε βρει το range για την περίπτωση που βρισκόμαστε. Αποθηκεύουμε και εδώ τις τίμες των εγγραφών και τα ranges σε δύο πίνακες.

Ο κώδικας για το equidepth:

```
def equiDepth(ranges, min, results):
    valuesPerBin = len(results)/binSize
    #begin from -1 because the i in for loop starts from the first
element not from zero
    valuesCounter = -1
    ranges.append(min)
    numtuples = []
    for i in results:
        valuesCounter += 1
        if valuesCounter == int(valuesPerBin):
            ranges.append(i)
            numtuples.append(valuesCounter)
            valuesCounter = 0
        if (len(numtuples) > binSize - 1):
            break

    return numtuples
```

Τέλος, εκτυπώνουμε τα ιστογράμματα στην οθόνη.

Μέρος 2:

Αρχικά ζητάμε από τον χρήστη να δώσει το διάστημα τιμών που επιθυμεί για να εκτιμήσουμε πόσες πλειάδες έχει το αποτέλεσμα μιας ερώτησης επιλογής στο πεδίο Income και αποθηκεύουμε αυτές τις δύο τιμές σε δύο μεταβλητές.

Χρησιμοποιούμε binary search για να βρούμε σε ποιο range πρέπει να κοιτάξουμε. Ο αλγόριθμος binary search έχει πολυπλοκότητα O(logn) και είναι ιδανικός για την περιπτωσή μας. Δεν χρησιμοποιούμε hashing διότι η τιμή που ψάχνουμε μπορεί να μην υπάρχει στον πίνακα που κάνουμε την αναζήτηση, ενώ με binary search παίρνουμε την αμέσως μικρότερη τιμή απο αύτο που ψάχνουμε.

Ο κώδικας για τον αλγόριθμο binary search:

```
def findLowestLimit(array, lengthOfTheArray, target):
    # Check if the target is outside our array
    if (target < array[0]):
        return array[0]
    if (target > array[lengthOfTheArray - 1]):
        return 0

    # Using binary search
    currentSmallestIndex = 0
    currentLength = lengthOfTheArray
    middle = 0
    while (currentSmallestIndex < currentLength):
        middle = (currentSmallestIndex + currentLength) // 2

        # Check if the current middle value if the value we are
searching for
        if (array[middle] == target):
            return array[middle]
```

```
        # If target is smaller we search to the left half of the
array
        if (target < array[middle]):
            # Check if the target is bigger than the value one
position before the current middle and return it if so
            if (middle > 0 and target > array[middle - 1]):
                return array[middle - 1]
            # or else repeat for left half
            currentLength = middle

        # If target is bigger we search to the right half of the
array
        else:
            # Check if the target is smaller than the value one
position after the current middle and return it if so
            if (middle < lengthOfTheArray - 1 and target <
array[middle + 1]):
                return array[middle]
            # or else repeat for right half
            currentSmallestIndex = middle
    return array[middle]
```

Εκτός από το να βρούμε το αποτέλεσμα που ψάχουμε πρέπει να λάβουμε υπόψην και τις περιπτώσεις που οι τιμές δεν υπάρχουν στα δεδομένα μας ή το πως πρέπει να υπολογιστεί σε ακραίες περιπτώσεις το εκτιμώμενο αποτέλεσμα.

Μελέτη 2-ου μέρους:

Μετά από αρκετές πειραματικές προσπάθειες, εκτελόντας το πρόγραμμα για διάφορα διαστήματα τιμών, παρατηρούμε ότι αν τα bins στο equiwidth ιστόγραμμα έχουν μεγάλη απόκλιση στον αριθμό εγγραφών που πιερέχουν τότε βγάζει χειρότερα αποτελέσματα από το equidepth ιστόγραμμα. Ενώ αν οι τίμες στα bins του δεν έχουν μεγάλη απόκλιση μεταξύ τους τότε βγάζει πάντα καλύτερα αποτελέσματα από το equidepth ιστόγραμμα. Επομένως, αν το διάστημα τιμών που δώσει ο χρήστης θα έχει τιμές που θα ανήκουν σε bins, έτσι ώστε δύο bins ή και παραπάνω θα έχουν μεγάλη διαφορά στον αριθμό των εγγραφών που περιέχουν (για το equiwidth ιστόγραμμα) τότε το αποτέλεσμα που θα επιστρέψει το equiwidth ιστόγραμμα θα είναι χειρότερο από το equidepth. Το equidepth ιστόγραμμα γνωρίζουμε ότι έχει πάντα τον ίδιο αριθμό εγγραφών σε κάθε bin του. Άρα το equiwidth ιστόγραμμα θα είναι πάντα καλύτερο αν οι τιμές στα bins του είναι με κάποιο τρόπο ομοιόμορφα μοιρασμένες.