CEE 505

Jean-Luc Jackson

10/28/2016

Assignment #3

CEE220Scores Program

Problem Statement

Develop a Python program that:

1. *Opens the provided file "CEE_220_Scores.txt"*
2. Reads and analyzes the header line to identify what columns contain Assignments, Labs, Midterms, and the Final (4 groups).
3. Read the second line to compute target scores for each group.
4. Loops through the remaining lines (list of students) and computes total points assigned for each of the four groups.
5. Compute a weighted score as sum[weight * (student score)/(target score)]
    a. Use weights HW, Labs, Midterms, Final = 0.25, 0.05, 0.40, 0.30
6. Compute the grade as (score – 0.20)/0.20
7. Round the grade to the nearest one decimal. Grades below 0.7 should be corrected to 0.0
8. Store name, group point sums, weighted score, and grade in a dictionary for all students
9. *Perform a statistical analysis of all assigned grades using pyplot's hist function.*
    a. Use bins in 0.1 intervals
    b. Use pyplot.savefig() to automatically save a PNG file for your report
10. Include sensible error handling
11. *After testing with "CEE_220_Scores.txt", run code on "CEE_220_AlternativeList.txt" and* present the statistics (figure) in report.

Code Description

Solution Details

This program opens and reads a provided text file containing student names and their assignment scores to create and plot a statistical histogram displaying the grade distribution for the course. The first line of the provided file contains assignment names, such as "Homework Assignment #3" or "Midterm 2", while the second line contains the total possible score of each assignment. This program uses these first two lines to create a dictionary containing assignment names as keys corresponding to possible points per assignment group as values. These assignment groups include Homeworks, Labs, Midterms, and the final. The program then loops through the remaining lines (all containing student information) and creates a dictionary with keys as the student's name and values as a dictionary containing that student's scores in each assignment group. Each student's weighted score and GPA grade are also calculated and stored. Finally, the GPA grades are plotted in a histogram showing the distribution of class grades.

Implementation Details

This program utilizes dictionaries to store data in multiple occasions. The following pairs are used as keys and values to create corresponding dictionaries, respectively:

- Assignment titles and possible points
- Assignment groups and group weights
- Assignment groups and target scores

- Assignment titles and a single student’s points
- Student names and student group scores

These dictionaries are used to find each student’s assignment group scores, weighted score, and GPA grade for the entire class. This program also utilizes string functions such as *strip()*, *lower()*, and *find()* to process the text file and organize data into relevant collections. Functions were created for tasks specific to this program and are stored in the file *ClassGradesFunctions.py*, accessed by the *import* function. Similarly, *numpy* and *matplotlib* are used to access plotting functions such as *hist()* and *savefig()*, creating a titled and labeled histogram which is saved in .PNG format with a name matching that of the provided text file.

ClassGradesFunctions

This program utilizes multiple functions for file reading and grade computing, all of which are in the file *ClassGradesFunctions.py*. These functions include *ReadFile()*, *SumGroups()*, and *GradeStudent()*.

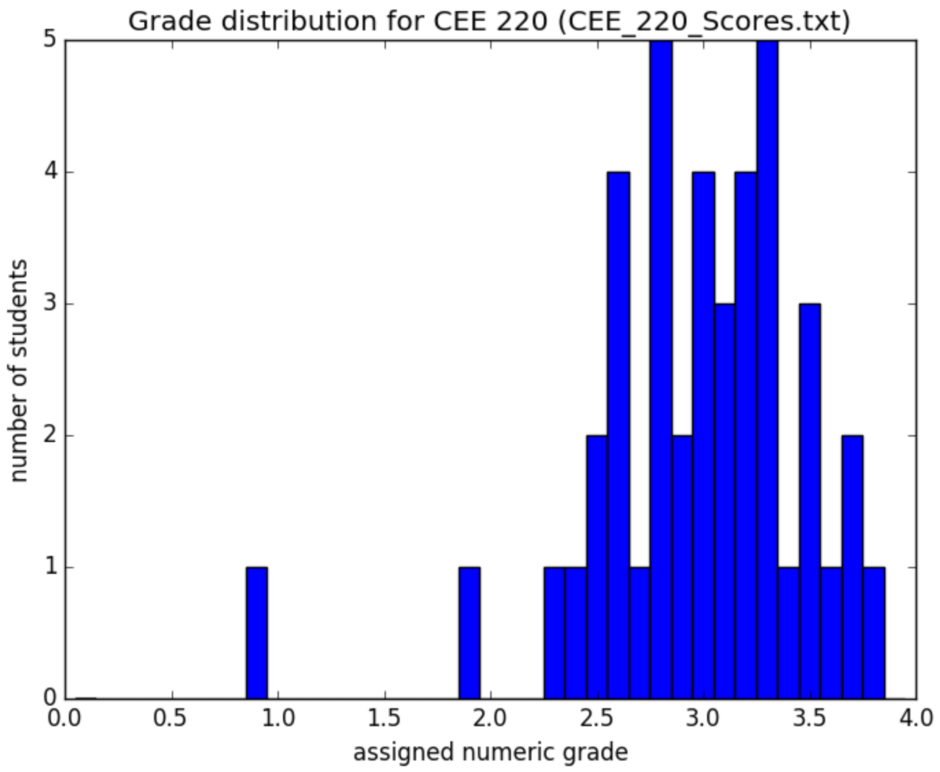
ReadFile is used to open a file for reading purposes. This function takes a file address as its only argument, utilizes “try-except” syntax to attempt to open the desired file anticipating an *IOError*, and returns the opened file.

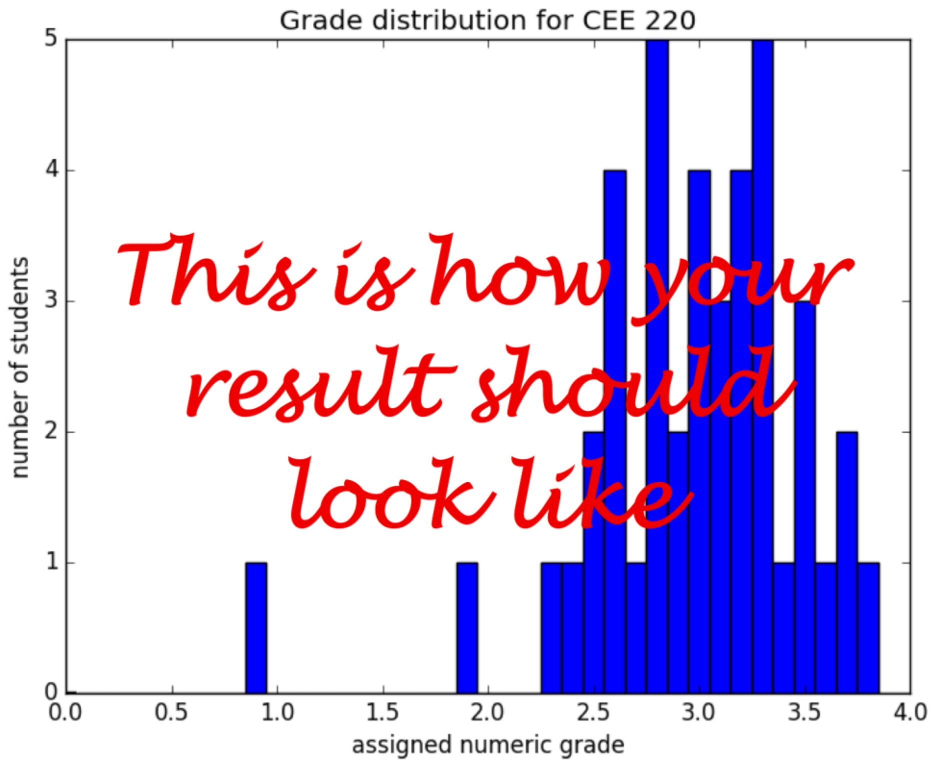
SumGroups() is used to sum up values pertaining to keys that contain specific substrings. In this case, it is used for each student in the file, adding up all points whose keys contain substrings “homework”, “lab”, “midterm”, and “final” and saving these summed points into a second dictionary with keys corresponding to these four groups. This function takes two dictionaries and a list as arguments: the first is the dictionary containing a particular student’s scores, which are summed over and organized into the second dictionary by group name. The second dictionary is intended to be passed to *SumGroups()* with keys matching those of the list and all zero values. The list argument contains strings which label each group of assignment. This function accounts for a missing assignment score, represented by ‘ ‘, and the possibility of a value being a string (such as the student’s name) by using “try-except” syntax to consider a ValueError. This function returns the second dictionary, which initially had all zero values, with total scores of each group for that student.

GradeStudent() is used to grade a particular student relative to target scores, using group weights to calculated a weighted score and a GPA grade. This function takes three dictionaries as its arguments: the first dictionary has keys matching the assignment groups and values that are that student’s group points; the second dictionary has the same keys as the first, but has values which are the target scores for each group; the third dictionary also has keys matching the four groups and values of the weights to be applied to each assignment group. This function uses a simple weighting equation to calculate a weighted score for student, followed by a GPA grade equation. The grade is rounded to a single decimal point, and if the grade is 0.7 or lower it is rounded to 0.0. A dictionary containing the “Score” and “Grade” are returned with each relevant value.

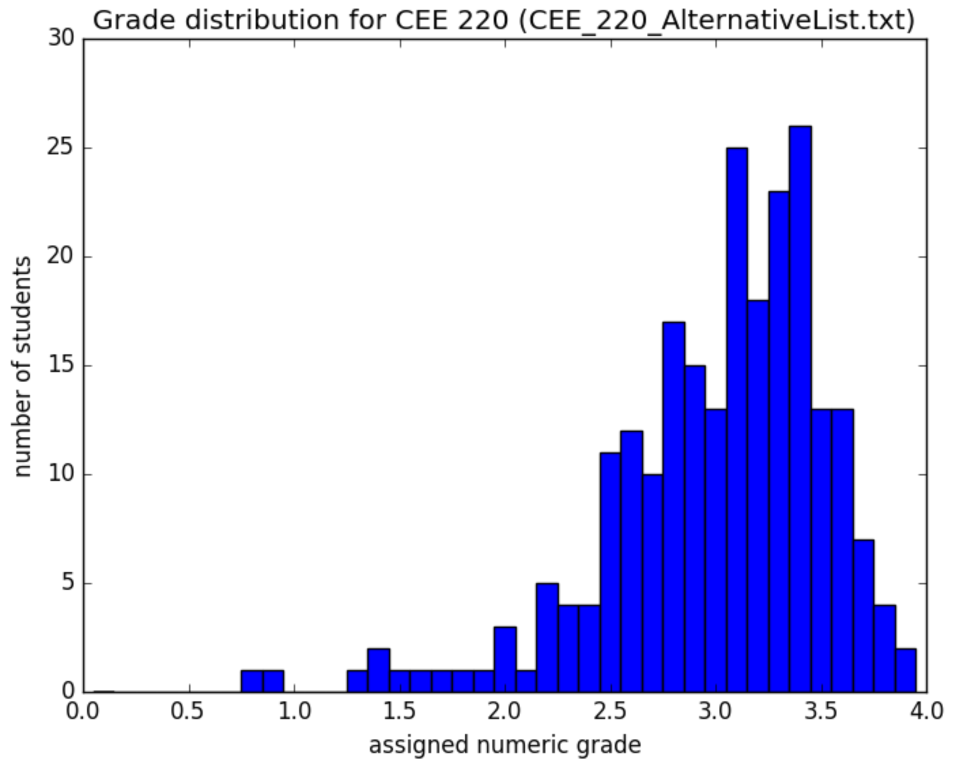
Statistical Analysis Results

The following histogram shows the plotted results for the initial *CEE_220_Scores.txt* file. The second histogram is that provided by the professor for program verification purposes.





The following histogram shows the plotted results for the secondary *CEE_220_AlternativeList.txt* file, provided by the professor for further program verification.



CEE220Scores Program (Python Code)

```
'''
Jean-Luc Jackson
CEE 505 HW #3
10/28/16
'''

### Import functions
from ClassGradesFunctions import *
import numpy as np
import matplotlib.pyplot as plt


### File Initialization
# Open orignal scores file for reading
filename = 'CEE_220_Scores.txt'
#filename = 'CEE_220_AlternativeList.txt'
fIn = ReadFile(filename)
print "\nFile ", filename, " opened.\n"


### Organize Headers
# Create list of Headers and Points Possible
assignments = fIn.readline().split('\t')   #read next line & split it up by tabs (\t)
headers = [item.strip() for item in assignments]          #strip to remove \n strings
print "Headers: ", headers, "\n"

points = fIn.readline().split('\t')         #read next line & split it up by tabs (\t)
pointsPoss = [item.strip() for item in points]            #strip to remove \n strings
print "Possible Points: ", pointsPoss, "\n"

# Create Dictionary that maps possible points to each assignment (keys = Headers,
values = PossiblePoints)
AssignmentGrader = dict(zip(headers,pointsPoss))
print "AssignmentGrader: ", AssignmentGrader, "\n"


### Compute Target Score for each Assignment Group (HW, Labs, etc)
groups = ["Assignment","Lab","Midterm","Final","Score","Grade"]
weights = {"Assignment":0.25,"Lab":0.05,"Midterm":0.40,"Final":0.30}
targetScores = dict.fromkeys(groups[:(len(groups) - 2)],0)

# Calculate Target Scores using SumGroups() function
targetGrader = AssignmentGrader.copy()
for thing in AssignmentGrader:     #remove any bonus assignments from target
    if thing.lower().find('bonus') > -1:
        targetGrader.pop(thing)

targetScores = SumGroups(targetGrader, targetScores, groups)
print "Target Scores: ", targetScores, "\n"


### Create Student Dictionaries organized by Groups
gradeVec = [] #used for plotting later
students = {} #dictionary to collect all students and their points
```

```
rows = 0
for line in fIn:
    rows += 1
    # Clean up each line for reading
    splitline = line.split('\t')
    #studentName = splitline[0]
    studentLine = dict(zip(headers,splitline))
    studentName = studentLine["Student"]

    # Use line's data to sum up into groups
    studentScores = dict.fromkeys(groups,0)
    studentScores = SumGroups(studentLine, studentScores, groups) # sum up points for 
each group
    students[studentName] = studentScores   # add this dictionary to global 
dictionary

    # Calculate Score and Grade for each student
    scoreGrade = GradeStudent(studentScores,targetScores,weights)
    students[studentName]["Score"] = scoreGrade["Score"]
    students[studentName]["Grade"] = scoreGrade["Grade"]
    gradeVec.append(scoreGrade["Grade"])    # add current student's grade to grade 
vector for plotting

print "There are {} students.\n".format(rows)


### Plotting Statistical Analysis - Histogram
bins = [x/100.0 for x in range(5,405,10)]
plt.hist(gradeVec, bins)
plt.ylabel('number of students')
plt.xlabel('assigned numeric grade')
plt.title('Grade distribution for CEE 220 ({})'.format(filename))
plotfilename = filename[:(len(filename) - 4)] + '.png'
plt.savefig(plotfilename)
print "Figure saved as {}.\n".format(plotfilename)
plt.show()


# Close file to prevent corruption
fIn.close()
if fIn.closed:
    print "File ",filename, " is closed.\n"
else:
    print "File ",filename, " not properly closed.\n"
```

ClassGradesFunctions (Python Code)

```
'''
Jean-Luc Jackson
CEE 505 HW #3
10/28/16

Functions used by CEE220Scores.py
'''


def ReadFile(address):
    '''
    Opens file for reading at specified address.
    '''
    try:
        f = open(address,'r')
        return f
    except IOError:
        print "Could not open file for reading at: ({})".format(address)


def WriteFile(address):

    try:
        f = open(address,'w')
        return f
    except IOError:
        print "Could not open file for writing at: ({})".format(address)


def SumGroups(dIter,dReturn,groups):
    '''
    Iterates over keys in dIter. If the key matches a category in groups, the value
for that key
    is added to the corresponding current value in dReturn (usually dReturn starts
with all 0 values).
    '''
    for key in dIter:
        if dIter[key] == '':
            adder = 0
        else:
            try:
                adder = float(dIter[key])
            except ValueError:
                adder = 0

        if key.lower().find(groups[0].lower()) > -1:
            dReturn[groups[0]] = dReturn[groups[0]] + adder

        if key.lower().find(groups[1].lower()) > -1:
            dReturn[groups[1]] = dReturn[groups[1]] + adder

        if key.lower().find(groups[2].lower()) > -1:
            dReturn[groups[2]] = dReturn[groups[2]] + adder
        if key.lower().find(groups[3].lower()) > -1:
            dReturn[groups[3]] = dReturn[groups[3]] + adder

    return dReturn

def GradeStudent(studentScores,targetScores,weights):
    '''
    Calculates the Score & Grade for a given student's studentScores, compared to
targetScores.
    Uses passed weights dictionary to weigh score calculation.
    '''
    score = 0
    for key in targetScores:
        score += weights[key] * (studentScores[key] / targetScores[key])

    grade = round( (score - 0.20)/0.20 , 1 )

    if grade <= 0.7:
        grade = 0

    return {"Score":min(score,1.0),"Grade":grade}
```