Timed Lab 2 – Stock Data Sorter

This is a Timed Lab; this Timed Lab is worth 25 Exam and Timed Lab points.

For this Timed Lab, you *may* use	However, you *may not*
• Course notes • Homeworks • Recitation assignments • Other course material • Any material you may find on the Internet that don't involve communicating "live" with other people.	• Communicate with other people/students in real-time via any means. This means no Facebook, email, Piazza, IM, IRC, cell phones, Google Talk, smoke signals, etc. • Share code with other students. • Look at other students work.

The TAs will be available to answer clarifying questions about the problem, but they are not permitted to give assistance with debugging code, program logic, etc. You will have an entire recitation period to work on this assignment; this time begins *exactly* when your recitation begins and ends *exactly* when your recitation ends: **No extra time will be given if you arrive late**, except in highly extenuating circumstances that must be approved by Dr. Summet or the Head TA.

T-Square will not permit any late submissions; ensure that you submit your code to T-Square several times to prevent earning a zero due to you being unable to submit. Your TAs will give a verbal warning 10 and 5 minutes before the end of the recitation period; you should submit at these times.

In your collaboration statement, if you use code from somewhere that is *not* a class resource (i.e. not listed on the course calendar), please list where this code came from. Ensure that you fill out the header at the top of the file.

Note that you must check out with your TA before you leave the recitation room. If you do not check out with your TA or you modify your submission after you leave the recitation room, **you will receive a grade of zero on the timed lab.** No submissions will be accepted after T-Square closes the assignment (i.e. it will not let you submit).

Problem Description:

Your client wants your analysis report on a stock he is thinking of investing in. Being new to investing he doesn't understand all the financial jargon, he just wants a short line to tell him how the stock traded in the year he requests. He provides you with a file containing the daily open, close, and volume data for the years 2001-2011. You will sort through this file to get the average daily change in price and total volume traded for each year using three functions described below.

ReadDataFile:

The purpose of the ReadDataFile function is to read in the csv file and convert it to a list of lists. The major list will consist of minor lists whose elements are each line of the csv file as separated by commas. Feel free to use the csv reader or simple file reader for this function as you see fit. This function is a helper function that will be called in the final function. There is one parameter that is the file name of the csv file your client provided you. The function should return the data in a form like this:

[[‘12/30/2011', ‘403.51’, ‘403.27’, ‘6416500’],[‘12/29/2011’, ‘403.4’, ‘403.39’, ‘7713500’], …..]

Calculations:

This function named Calculations should take in one parameter, the data from ReadDataFile. This function will take that data and arrange it into a dictionary of yearly cumulated data. **Notice that the year is the final 4 digits in the date column.** The dictionary will have keys corresponding to the years 2001-2011 and values for each year being a tuple of (average daily price change over the year, total volume changed over the year). You may need to first sort into a dictionary containing all the daily changes and volumes before calculating into the yearly averages and sums. The calculations for the first two example data points are found below:

Total volume changed over the year 2011 = sum(6416500,7713500,......)

Average daily price change over the year 2011 = average(403.27-403.51,403.39-403.4,......)

You will then return this dictionary with these values in a tuple.

Dictionary['2011'] => (-1.624166666666666, 4430740100)

DataQuery:

This function will call the other two functions before doing anything else. Call ReadDataFile using the .csv file name as the input and store the output as a variable. Next pass that output in as the input for Calculations and store its output into another variable. At this point you will ask the user (client) for the year they are requesting the summary of data for using an input request. At this point we will use the dictionary outputted by Calculations to format a line for our client. You should print the message "For <INPUTTED_YEAR>, the average daily change was $<AVG_DAILY_CHG> and the volume moved was <TOTAL_VOLUME>." without the quotes or <>, and with INPUTTED_YEAR, AVG_DAILY_CHG, and TOTAL_VOLUME replaced with their respective values for the year requested, with the average daily change displayed to two decimal places. This function should not return a value.

Grading:

ReadDataFile (9 points)

- +2 – Correct function header
- +2 – Correct return type
- +2 – Opens file and reads in data
- +1 – Closes file handle
- +2 – Correctly sorts rows of data into lists

Calculations (10 points)

- +2 – Correct function header
- +2 – Correct return type
- +2 – Correctly uses functions for each calculation
- +1 – Handles all trading days each year
- +2 – Saves average and sum correctly
- +1 – Returns the correct dictionary

FeeGenerator (9 points)

- +1 – Correctly calls ReadDataFile
- +1 – Correctly calls Calculations
- +1 – Asks for user input
- +1 – Prints result
- +1 – No return value
- +1 – Prints correct year
- +1 – Prints correct volume
- +1 – Prints correct average daily change
- +1 – Prints to two decimal places