

Homework 6 — College Admissions Algorithms

Due October 25th at 11:59pm

(50 points)

Objectives:

- Understand how to iterate through a list
- Practice implementing functions that return values
- Explore algorithmic decision-making and representations of people in data
- Understanding outliers, missing information in data, and human judgements in algorithmic decisions

Turn in:

- hw6_lastname_code.ipynb

You should turn this file in on Canvas under the “Homework 6 - College Admissions Algorithms” link. Complete the following checklist before turning in your code:

- o all cells in your file run without errors after restarting the kernel
- o you have no global variables (i.e., all your variables are inside functions)
- o your code is nicely commented
- o you have appropriately renamed the starter notebook
- o you have set the variable `filename = "superheroes_tiny.csv"`

Introduction:

As you know, the college admissions process involves a lot of types of data from prospective students to make decisions. With the number of applicants increasing, colleges may begin relying on algorithms to select which applications should receive more intensive human review. An algorithm could use quantitative data--such as GPA and SAT score--to provide initial recommendations. In fact, there is more data available than ever. [Many colleges even track data about prospective student engagement](#) - e.g., whether they open emails, visit the college website, engage on social media, etc. This creates a “demonstrated interest” value.

[Based on a recent survey of college admissions officers](#), we know some of the weights that humans tend to give to these different types of data. Your task will be to create a program that iterates through a list of data points and provides a recommendation for which prospective students are likely to be the best candidates for admission.

Prospective student data is organized in the **superheroes.csv** file such that the data for each student is on one line, with the values separated by tabs. An example of two students’ data might be:

```
Student,SAT,GPA,Interest,High School Quality,Sem1,Sem2,Sem3,Sem4
Abbess Horror ,1300,3.61,10,7,95,86,91,94
Adele Hawthorne ,1400,3.67,0,9,97,83,85,86
Adelicia von Krupp ,900,4,5,2,88,92,83,72
```



The data includes (in order):

Student: a unique identifier for each student (their string name)

SAT score: value between 0 and 1600

GPA: value between 0 and 5

Interest: value between 0 and 10 (from very low interest to very high interest)

High School Quality: value between 0 and 10 (from very low-quality to very high-quality high school curriculum)

Sem1: average grade for semester 1

Sem2: average grade for semester 2

Sem3: average grade for semester 3

Sem4: average grade for semester 4

Problem 1: Getting set up with our data (15 points)

First, we need to make sure that we can appropriately read in the data line by line, parsing each line into a list and converting each element to the appropriate type.

Task 1: read in your data set in `main()`, looping through its contents line by line. This data contains one row of headers at the beginning, so isolate the headers before reading through the rest of the file so you don't try to do calculations with them!

Make use of the `str.split(delimiter)` function to break individual lines into a list of elements. Print the contents of your list after using the split function. You'll delete this print statement later but make sure to double check this before moving on! Once you have each line in a list, save the student's name in a variable, then delete this element from your list.

Task 2: Once you have a list of strings for each line, you will write a function **`convert_row_type`** that takes one list of elements (representing the data for one student) as a parameter and converts it so that all numbers are converted to floats. Make sure not to lose any information when you do this conversion!

Example:

Input: `["1300", "3.61", "10", "7", "95", "86", "91", "94"]`

Return value: `[1300.0, 3.61, 10.0, 7.0, 95.0, 86.0, 91.0, 94.0]`

Task 3: in `main`, once you've called `convert_row_type` on the list representing one row, call the provided `check_row_type`. If this function returns `False`, print out an error message. Ensure that none of the rows in your data return `False` when passed to this function.

Task 4: separate your data. Use [list slicing](#) or while loops to separate your list (which should contain 8 numbers at this point) into two lists: one that contains the student's SAT, GPA, Interest, and High School Quality scores, and one that contains their 4 semester grades. You'll do Problems 2 - 5 with the first list of 4 numbers and Problem 6 with the list of grades.



Problem 2: Prospective Student Score (10 points)

Task 1: Write a function `calculate_score` that takes a list as a parameter and calculates an overall score for each student based on the following weights: 40% GPA, 30% SAT, 20% strength of curriculum, 10% demonstrated interest. The list parameter will contain all of the relevant information about the student. The return value is the student's calculated score.

To make this work, you will also need to normalize both GPA and SAT so that they are also on a 0 to 10 scale. To do this, multiply the GPA by 2, and divide the SAT score by 160.

Example:

Input: `[1300.0, 3.61, 10.0, 7.0]`

which represents a student with a 1300 SAT score, a 3.61 GPA, 10 out of 10 for interest and 7 out of 10 for high school quality

$((3.61 * 2) * 0.4) + ((1300 / 160) * 0.3) + (10 * 0.1) + (7 * 0.2) = 7.73$ out of 10

Output: 7.73

Round each output score to 2 decimal points.

Task 2: In your `main()` function, modify your loop that reads in and converts your data to call the `calculate_score` function for each line (row) of data (after you've converted it). Then, write the student's name and their calculated score to a new file called `student_scores.csv` such that each row contains a student's name and their score, separated by a comma.

Example:

```
Abbess Horror ,1300,3.61,10,7,95,86,91,94
Adele Hawthorne ,1400,3.67,0,9,97,83,85,86
Adelicia von Krupp ,900,4,5,2,88,92,83,72
```

lines written to file:

```
Abbess Horror ,7.73
Adele Hawthorne ,7.36
Adelicia von Krupp ,5.79
```

Task 3: Write all the student names for all students who have a score of 6 or higher to a file called `chosen_students.txt`. You should do this in your `main()` function, where you have access to the returned calculated score for each student and their student name.

Example:

```
Abbess Horror ,1300,3.61,10,7,95,86,91,94
Adele Hawthorne ,1400,3.67,0,9,97,83,85,86
Adelicia von Krupp ,900,4,5,2,88,92,83,72
```

lines written to file:

```
Abbess Horror
Adele Hawthorne
```



Problem 3: Looking for Outliers (10 points)

Consider ways that this algorithm might systematically miss certain kinds of edge cases. For example, what if a student has a 0 for demonstrated interest because they don't use social media or have access to a home computer? What if a student has a very high GPA but their SAT score is low enough to bring their score down; could this mean that they had a single bad test taking day?

Task 1: Write a function **is_outlier** that can check for certain kinds of outliers. It should check for: **(1)** demonstrated interest scores of 0 and **(2)** a normalized GPA that is more than 2 points higher than the normalized SAT score. If either of these conditions is true, it should return True (because this student is an outlier); otherwise, the function returns False.

Task 2: Call **is_outlier** for each student from your **main()** function and write the students' names to a file called **outliers.txt**, one name per line if they are an outlier.

Task 3: Combine the work that you've done now to create an improved list of students to admit to your school. Write the student names, one per line to the file **chosen_improved.txt** if they either have a score of 6 or greater or if they are an outlier and their score is 5 or greater. Make sure to take advantage of the work that you've already done by calling your functions from previous problems to help you out!

Problem 4: GPA Checker (10 points)

A single GPA score is not a full picture of a student's academic performance, as it may have improved over time or included outlier courses or semesters. A more context-sensitive algorithm could consider a student's entire transcript and check for, for example, a single class score that is more than two letter grades (20 points) lower than all other scores. For this task, you will use the second half of the data for each student in the provided file.

Task 1: Write a function **grade_outlier** that takes in a list of grades (of *any* length at least 2) and returns **True** if the lowest number is more than 20 points lower than all other numbers; otherwise, **False**. This function must not modify the list passed in!

Example:

Input: [99, 94, 87, 89, 56, 78, 89]

Hint: Sort the list from lowest to highest, and check for the difference between the two lowest grades.

$78 - 56 = 22$; $22 > 20$

Output: True



Next, consider the data that we have: a list of grades for each student, one grade per semester for four semesters.

Make sure that your `grade_outlier` function works by calling it for every row, passing in the grades list you isolated in problem 1, task 4. Print out an informative message about which students have a single grade outlier. You'll delete this later but it's a great way of testing your function!

Finally, consider the importance of an algorithm being able to flag students who might have a lower overall GPA but have shown improvement over time.

Task 2: Create a function **`grade_improvement`** that returns **`True`** if the score of each semester is higher than each previous semester and **`False`** otherwise. Hint: investigate how the `==` operator works between two lists and think about using the `sorted()` function.

Task 3: Using the grade information that you've just learned, create your own conditions based on the information from the previous problems and `grade_outlier` and `grade_improvement` to chose all students if they either have a score of 6 or greater or if they have a score of 5 or more and at least one of the following is true: `is_outlier` returns `True`, `grade_outlier` returns `True`, or `grade_improvement` returns `true`. Write the students who fit this description to **`extra_improved_chosen.txt`**, one name per line.

Note!

We will be testing your program using a file with the same format but different data! Make sure that you haven't "hard coded" anything specific to your data. We do not guarantee that all scenarios are tested by the code that we have provided you.

Comments & style (5 points)

Your variable names must be informative.

You must have one `main()` function that is structured like a "table of contents" and use parameters and returns appropriately. All code except your call to `main` must be contained within a function.

Your code must be commented. You must include a file comment, inline comments, and function comments. An example function comment is shown below:

```
def fahrenheit_to_celsius(fahrenheit):  
    """  
    This function converts a temperature in fahrenheit to celsius  
    and prints the equivalence.  
    Parameters:  
        fahrenheit (int or float): degrees fahrenheit  
    Return:  
        celsius (float): equivalent degrees celsius  
    """
```



```
"""  
celsius = (fahrenheit - 32) * (5/9)  
return celsius
```

