MIT Teacher Student Matching Program

Jessica Lin

Disclaimer: I did use AI to help me write out the code but in terms of the ideas and functions, they were all from my own brain and creativity,

Objective:

The goal of this program is to match MIT undergraduates who are taking education courses with teachers around the Boston area to fulfill classroom observation requirements. This program will only be used internally for MIT students.

Inputs:

This program was designed to take inputs from two different CSV files:

Student CSV File

1) student csv file (titled ‘student_list.csv). The information format to fill in include the following:

Column name	**Description and Format**	**Example**
student_name	name of the student	Calvin M. Jessica Lin
major(s)	MIT uses numbers for majors, so this will be kept. Rather than dashes, use periods for majors with specific tracks. Separate out all majors with a comma and a space afterwards	6.3 *(instead of 6-3)*, 14 21H
class_year	current student standing; write either: freshman, sophomore, junior, senior, other *consider switching to years (e.g. 2024, 2025, 2026, 2027)*	senior junior
group_preferred	age range/group that the student prefers observing. Only write from	middle elementary high

	the following options: middle, elementary, high *consider adding pre-k to the options if that is something necessary*	
subjects_preferred	Students list subjects they are interested in observing. These will be abbreviated using capitalization and shorthand. Use commas and spaces to separate out subjects *This standardized formatting can be developed at a later time*	CS, MATH, FL (*foreign language)*, ECON
s_availability	IMPORTANT SECTION Students will list out their availability in this section. Each entry will be in this format with NO SPACES expect between day and time range: day starttime-endtime separated out by commas and spaces. day = day of the week (Monday, Tuesday, …) *consider abbreviating to Mon, Tues, …* Time = to make it easier, possibly, I recommend using a 24-hour clock (military time) so that students don't get confused between writing for example, 7:00am vs 7:00pm if they only wrote 7:00. *again military time is just a suggestion but if am/pm preferred that's also okay*	Monday 13:00-15:00, Tuesday 7:00-9:00

	Keep in mind students should be listing a minimum of a two hour time block to account for the commute	
assigned_teachers	LEAVE THIS FIELD BLANK. This will be for the program to fill in. it will contain a list of teachers the student has been assigned to after the matching process is complete.	*after program is done, this field will look like this:* Ms. Smith, Sarah Wharton
current_hours	LEAVE THIS FIELD BLANK. This will be for the program to fill in. This entry will contain the current number of hours the student has for that week. This will be used to see if the complete the 2-3 hour requirement per week of classroom observation.	1

Teacher CSV File

2) teacher csv file (titled ‘teacher_list.csv). The information format to fill in include the following:

Column name	**Description and Format**	**Example**
teacher_name	name of the teacher	Sarah Wharton Ms. P
school	Name of the school the teacher teaches at	PHA Medford HS
group_taught	age range/group that the teachers teach. Only write from the following options: middle, elementary, high *consider adding pre-k to the options if that is something necessary*	middle elementary high

subjects_taught	Teachers list subjects they teach. These will be abbreviated using capitalization and shorthand. Use commas and spaces to separate out subjects *This standardized formatting can be developed at a later time*	CS, MATH, FL (*foreign language)*, ECON
t_availability	IMPORTANT SECTION Teachers will list out their availability in this section. Each entry will be in this format with NO SPACES expect between day and time range: day starttime-endtime separated out by commas and spaces. day = day of the week (Monday, Tuesday, …) *consider abbreviating to Mon, Tues, …* Time: to make it easier, possibly, I recommend using a 24-hour clock (military time) so that students don't get confused between writing for example, 7:00am vs 7:00pm if they only wrote 7:00. *again military time is just a suggestion but if am/pm preferred that's also okay*	Monday 13:00-15:00, Tuesday 7:00-9:00
max_students	the maximum number of students teachers are willing to have come in to do classroom observation	2

num_current_students	LEAVE THIS FIELD BLANK. The program will fill this in. Current number of students he teacher is assigned. This should be no more than the maximum number of students they put down as a preference	1

Output:

The output should be a data frame that looks like the following:

```
Formatted Schedule:
```

Day	**Time**	**Subject**	**Teacher**	**Group**	**Students**
Monday	13:30 - 14:30	CS	Sarah Wharton	middle	Calvin M., Srithi
Tuesday	07:30 - 8:30	MATH	Mr. Smith	middle	Calvin M.

Libraries Used:

- **CSV**: used for reading the information out of the csv and converting it into list of dictionaries where each dictionary is a student or teacher.
- **Datetime**: used to compute time overlap and hours to make sure that the availability works out between the teacher and students.
- **IO**: used to test the parsing function (using string inputs). Not needed for the actual functioning of the program since we can load from csv.

Code Block Walkthrough

1) Wrote a helper function called `list_subjects` that turns the string list of subjects (from student and teacher CSV files) into an actual list of subjects (as strings).
2) Wrote two functions `load_students()` and `load_teachers()` that don't take any parameters to load data of the student and teacher csv files.
3) Wrote some test code to produce sample output and make sure that the csv was being processed correctly and nothing was missing from the output dictionary lists
4) Wrote out different parsing functions:

- `parse_time(time)`: converts the string time (e.g. `‘13:00’` or `‘7:00’` into a datetime object for the datetime library to work with.
- `parse_block(timeblock)`: converts the day-time blocks (e.g. `‘Monday 13:00-15:00’`) into the separate components (string, datetime objects) inside a tuple (e.g. `(‘Monday’, datetime(13:00), datetime(15:00)`).
- `parse_availability(availability)`: converts the many availability entries from the students into a list of those blocks.
- `parse_csv(source, kind=”student”)`: goes through the entire csv file to take out all the student/teacher availability blocks.

5) Tested them with the sample data written converted to io.StringIO to put into CSV format for simulation.
6) Started writing the matching algorithm functions. First thing to do was convert the list of subjects into sets by writing a `set_subjects(raw)` function.
7) Matched the student and teacher subjects using `subject_match(student,teacher)`. Returns `True` if there is at least one subject that overlaps
8) Matched the student and teacher age groups using group_match(student, teacher). Returns `True` if the age groups match.
9) Matching teacher and student availability. This one required the use of four different functions:
    - `adjust_time(t,str)`: Takes a 'HH:MM' string or datetime/time and shifts by minutes. Returns a datetime object (with dummy date)
    - `blocks_overlap(block1, block2, buffer_minutes=30, min_overlap_minutes=30)`: checks if two time blocks overlap at all and takes into account the 30 minute travel time for students.
    - `overlap_hours(block1, block2)`: calculates how many hours the blocks actually overlap.
    - `availability_match(student, teacher)`: goes through the ALL time blocks for students and teachers to return `True` if the block of the teachers and students overlap.
10) Wrote `is_match(student, teacher)` function to return True if the subject, group, and availability preferences all lined up for the student and teacher.
11) Wrote the `find_all_matches(students, teachers)` function to go through the entire CSV to find all matches between students and teachers. Returns a list of matches.
12) Wrote `assign_students(students, teachers)` function to produce the actual groupings of students and teachers with the day, time, subject and group information. It returns a list of dictionaries that have that information mentioned.
13) Used sample data to test to see if the `assign_students(students, teachers)` would work.

14) Wrote `format_schedule(assignments)` to format the dictionary information into a data frame. This can be saved and rewritten into a csv file or doing a UI implementation. This comes after the sample data testing which I used because that's where I defined the results variable that holds the dictionary list information.