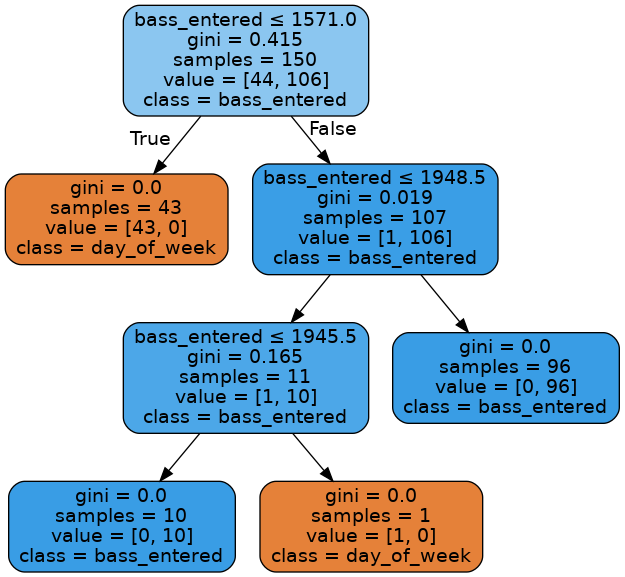
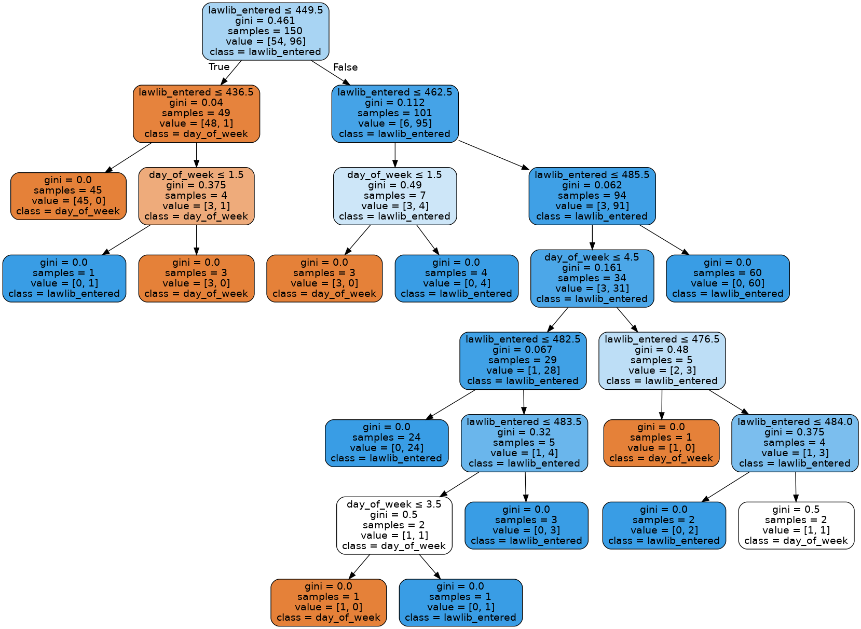
Note: In some cases I process the data in a separate python file. In other cases I have the processing in the same python file, but it is commented out because it only needs to be run once. I am including all the csv files you need to run each program on its own. If you want to verify that they process the data correctly, you can uncomment the sections I have commented out.

**process\_data.py**: creates a file of all the ids in **door\_data.csv**, called **ids.csv**. Also makes **library\_data.csv**

**library\_data.csv:** has six columns: day, day\_of\_week, number of people who entered bass before 6pm, number of people who returned books to bass, number of people who entered the lab library before 6pm, and number of people who returned books to the law library. There is one row per day.

**decision\_tree1.py**: trains decision tree on bass library data from **library\_data.csv**, tests condition bass with 2400 visitors before 6pm on Saturday and outputs “Bass on a Saturday with 2400 visitors by 6 p.m. is predicted to be a high-return day.” Also makes file **decision\_tree\_1.png**, which is the left image below. If you want to run this, I recommend running it on the zoo. Graphviz is a pain to install, at least it was for me.

**decision\_tree2.py:** trains decision tree on law library data from **library\_data.csv**, tests condition law library on Thursday with 450 visitors before 6pm and outputs “The Law Library on Thursday with 450 visitors by 6 p.m. is predicted to be a high-return day.” Also makes **decision\_tree\_2.png,** which is the right image below with Graphviz.

For 2400 visitors before 6pm on a Saturday for Bass library, we get a high return day. Looking at the tree on the left, bass entered is greater than 1571 so we go right. It is greater than 1948.5 so we go right. We are now at a leaf with 96 high return days and 0 low return days, so it predicts a high return day.

For 450 on Thursday before 6pm in the law library, we follow this path:   
right, left, right, ending at a place with four samples where all four have high return days.

Both of these trees make fairly reasonable sense for what we expect. They have cutoffs for how many people enter the library and say lower than this we don’t generally don’t expect a high return day and higher than this we generally do. There are a few complications based on the day of the week in both trees, but these are interestingly not really major factors in these decision trees. It appears that behavior is generally pretty similar on most days. One might think that Saturday and Sunday are much more common days to return books, but this doesn’t appear to be a major factor (only appears in a couple places).

**find\_regular\_appointment.py:** first makes a file called **regular\_appointment\_data.csv,** which includes all the lines from **door\_data.csv** that are at the health center. It then makes a file called **common\_data.csv**, which just lists the ids of people with 15 or more appointments over 150 days, which I counted as necessary for an appointment to be considered regular. It then goes through every id in common data and checks that its appointments (all yale health data is stored in the “data” variable) are all on the same day and within 30 minutes of the first appointment. It outputs a list of around 90 people with regular appointments. This is one of them: '7064193937378363'

This problem rests a little bit on your definition of regular. I played around with different definitions (every week vs every two weeks vs every month) and different amount of flexibility (within 60 minutes of same time, 90 percent of appointments on same day, low standard deviation in the swipe time, etc…) but they didn’t appear to make much difference. I settled on >=15 appointments all on same day within 30 min of the original swipe time.

**predict\_sleep2.py:** I started off doing this badly, so I redid it (there’s no 1). First, I reverse the **door\_data.csv** file and get the last swipe into a residential college for each student on each day (including the information from midnight to 2 am. I do this by scaling everything back two hours and only resetting when I pass the 2am mark. There are very few swipes into residential colleges in this time period. In my experience, I return to my residential college to sleep after 2am on <1 percent of days, so this should be sufficiently accurate). This is the **pre\_sleep\_data.csv** file. Then, I reverse this into **sleep\_data.csv.** I only do this once (it is commented out in the file).

Then, I load ids from **ids.csv** into a dictionary and give them each a list of 7 lists (for each day of the week) and go through sleep data, putting the last college swiped into each day into the respective list for each student. I then take the mode of each sublist (if there are any errors with this, then the program cannot determine the most common college a student sleeps in for a certain day of the week and the student is ignored).

I then iterate through the dictionary. For each non-homogenous schedule without errors, I compare it to every other schedule and print whenever there is an exact match. This prints every pair of two that match (if there’s a triple A, B, and C, that all match, it will print AB, BC, and AC). Here is one pair:

#Student 3398615277913271 and Student 7960088203188404:  
#Sunday: Pauli Murray  
#Monday: Pauli Murray  
#Tuesday: Pauli Murray  
#Wednesday: Pauli Murray  
#Thursday: Pauli Murray  
#Friday: Davenport  
#Saturday: Pauli Murray

**analyze\_meal\_plan.py** first loads the ids from **meal\_plan.csv** with on\_meal\_plan values of one into a dictionary indexed by their student\_id. It then opens **door\_data.csv** and adds the value of is\_dining\_hall to the value indexed by the student\_id in the dictionary for lines in which the student id is in the dictionary. This gives a raw total number of meals eaten by each student on the meal plan. Then, for each student, divide this number by the number of weeks in 150 days, which is 150/7. This gives me the number of meals per week on average. For each student in the dictionary, I then print out something like “Student 1197594711925368: 6.77 meal swipes per week on average.” If their average is below 7. This is true for about 70 students.

For the write\_email function, I first check in **meal\_plan.csv** that they are on meal plan. I then go through **door\_data.csv** adding up the total number of dining hall swipes they have used. If they’ve used less than 150 (which would be one per day, or half of their allowed meal swipes), then I print out an email as specified in the expected output. This could also be done by looking up the value in the dictionary (I have commented out this, but left it in the document) but I was assuming that writing the email is an independent procedure from finding all students who are not on meal plan (you want to be able to write an email to a single student without finding every student who is not using half their swipes on average).

**nearest\_neighbors.py** first loads the ids from **ids.csv** into a dictionary. Each index in the dictionary is initialized with a list of 300 -1s (just markers). It goes through **door\_data.csv** and puts the time that each student went to dinner and where they went to dinner in successive indices (0 and even indices are the dining hall, odd indices are the time). If a student did not go to dinner on a particular day (did not swipe into a dining hall between 5 and 8) then both numbers remain -1.

I define a compare function to pass into the nearest neighbors function as a metric. It adds one to the score if two people had dinner in the same place and swiped in within 30 minutes of each other for each day. If they both skipped dinner one day, then it adds .25 to the score (they may be going out to dinner or an event together. However, this is not as certain so I add less to the score). Note that the nearest neighbors algorithm looks for the lowest scores, so I return (150-score) from this function.

I move all the dining hall schedules from the dictionary into a list so they can be input into the nearest neighbors function from sklearn, I use compare as the metric, I fit the samples, and then I run kneighbors with n=6 (because the closest/first return value will be the students’ own schedule). I keep track of the indices of the five nearest neighbors, find them in the dictionary, and print them out.

Potential friends for student 2969414704160674:  
2571501919439390  
0889619518791047  
4337759318218479  
3060879441123832  
5098296110212146