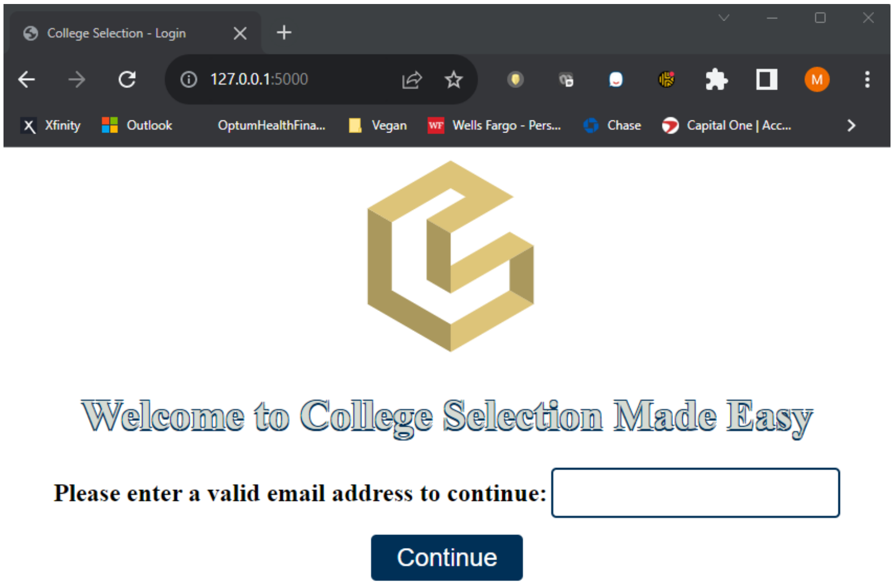
Contents

Frontend

Login

After following the link in python, users should be directed to the login page shown below in their default browser. You will need to input an email address to continue to the landing page. The validation is treating input text containing “@” as a valid email. Therefore, entering text not containing “@” will result in redirects to the login page until the validation is passed.



Welcome to College Selection Made Easy

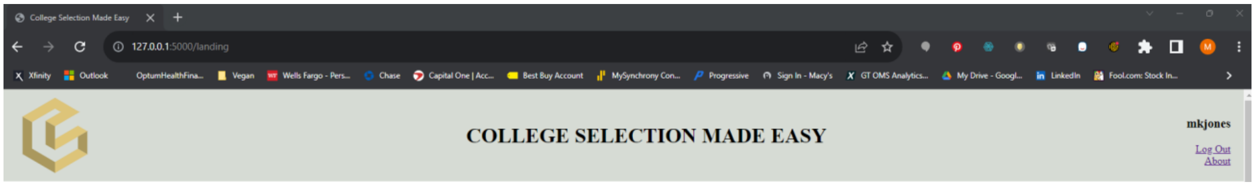
Please enter a valid email address to continue:

Landing

The landing page is organized into 4 distinct regions: header, profile, map, and table.

Header

The header features our app logo, app name, the user name (extracted from the email address), a “Log Out” link, and an “About” link. “Log Out” will direct the user back to the login page. The user’s profile results will be saved the next time they log back in, but the profile selections will not be saved. Some of the selections like test scores could be considered sensitive information, which we do not want to make easily available given the app is not password protected. “About” will direct the user to the project’s GitHub where they can learn more about the application or report any issues.

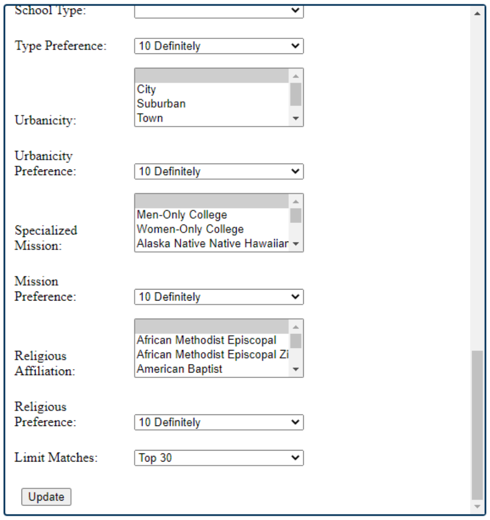


COLLEGE SELECTION MADE EASY

mkjones

Profile

Below the header to the left of the map is the profile section where the user can make selections on up to 16 distinct features (see table 2) along with specifying each feature's preference or importance to them (see table 1). "10 Definitely" is treated as a hard constraint while the rest are soft constraints weighted according to the preference levels specified by the user. By default, all preferences are set to 10. "States" and "Degree" are always hard constraints. Each feature also has a default value. The feature is ignored if the default value is unchanged, degree being an exception. There is a scrollbar to the right of the container to confine the input to a smaller space. Scrolling to the very bottom of the container, the user has the option to return the top N results, with N ranging from 5 to 30 in increments of 5. The retrieval is capped at 30 to encourage users to adjust results by altering preferences. Once a user is ready to view results, they must click the "Update" button to populate the map and table. These both will be empty when a user first signs in.



School Type:

Type Preference: 10 Definitely

Urbanicity: City
Suburban
Town

Urbanicity Preference: 10 Definitely

Specialized Mission: Men-Only College
Women-Only College
Alaska Native Native Hawaiiar

Mission Preference: 10 Definitely

Religious Affiliation: African Methodist Episcopal
African Methodist Episcopal Zi
American Baptist

Religious Preference: 10 Definitely

Limit Matches: Top 30

Figure 1 Profile

Table 1 Preferences

1 Equally
2 Between 1 & 3
3 Moderately
4 Between 3 & 5
5 Strongly
6 Between 5 & 6
7 Very Strongly
8 Between 7 & 9
9 Extremely
10 Definitely

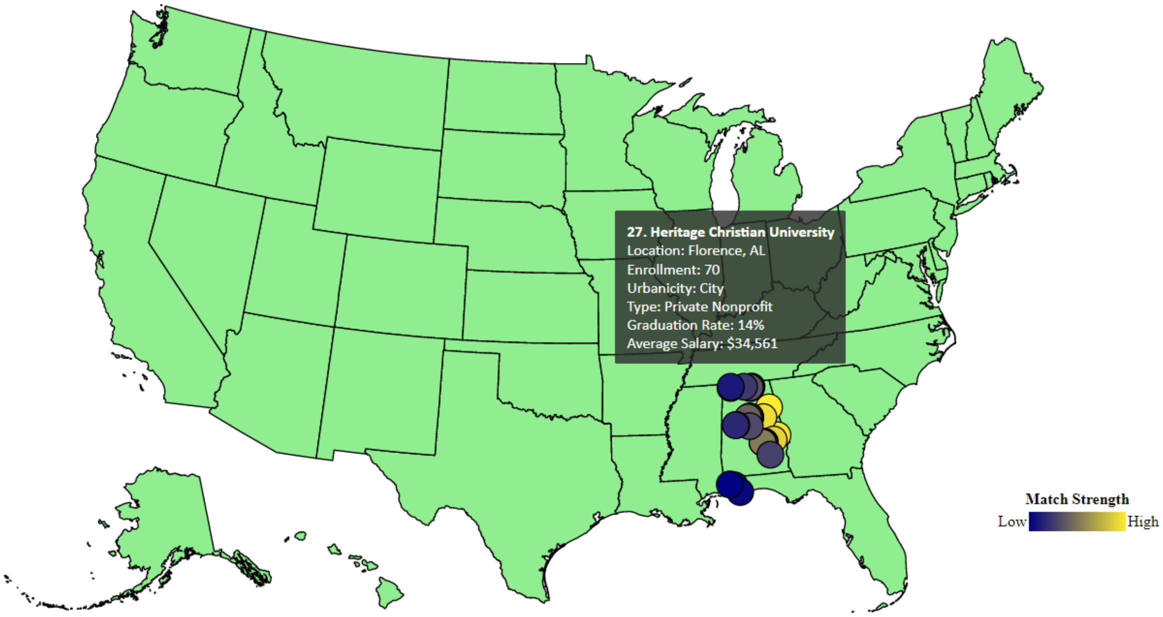
Table 2 Features

Feature(s)	**Type**	**Constraints**
States	Multi Selection box	Hard filter only. Limited to U.S. states and territories
Home Zip Code, Max Distance	Input box, Numeric value	Zip must be 5 digits. Geodesic distance in miles (think radius) measured from home zip to institution zip. Used with the max distance to determine if school is in range. Max Distance must be in increments of 5 miles.
Degree	Dropdown	1 selection from 8 different degree/certificate types
Field of Study	Dropdown	1 selection from over 100 different majors
Household Income, Max Avg Annual Cost	Dropdown, Numeric value	There are 5 income brackets to choose from that affect the average cost reported after including financial aid. Therefore, users in higher income bracket will see higher average cost than those in lower income brackets. Max Cost must be in increments of $1,000.
Median Expected Salary	Numeric value	Must be greater than or equal to 0. Salary must be in increments of $5k.
SAT Math	Slider	Scores range from 200 to 800

SAT CR	Slider	Scores range from 200 to 800
ACT Score	Slider	Scores range from 10 to 36
Acceptance Rate	Slider	0 to 100%
Graduation Rate	Slider	0 to 100%
Student Body Size	Multi Selection box	Null, Small, Medium, Large. Determined based on quantiles.
School Type	Dropdown	Public, Private Nonprofit, Private For-Profit
Urbanicity	Multi Selection box	City, Suburban, Town, Rural
Specialized Mission	Multi Selection box	
Religious Affiliation	Multi Selection box	

Map

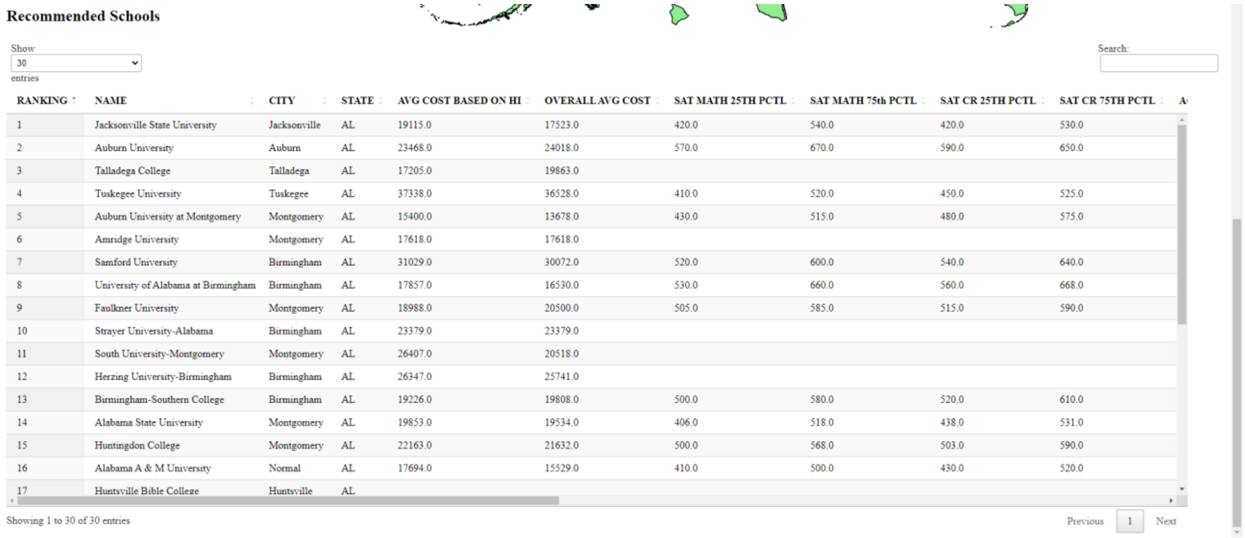
The map provides a visual display of where recommended schools are in relation to each other along with some neat characteristics. Data points are colored on a gradient from navy blue to yellow based on low to high rank, respectively. A visual cue that all hard constraints are used occurs when all circles are yellow due to each school being ranked 1. When hovering over a data point, a tool tip appears with varying information depending on what is available in the database.



27. Heritage Christian University
Location: Florence, AL
Enrollment: 70
Urbanicity: City
Type: Private Nonprofit
Graduation Rate: 14%
Average Salary: $34,561

Table

The table appears at the very bottom of the webpage and is populated with 22 columns relevant to the profile features. The left 2 columns, Ranking and Name, are fixed in place when the user scrolls horizontally to aid in keeping track of what school and rank is being referenced. The viewer has the option to display all results on one page or view the results across multiple pages up to 6. By default, all the results are available on one page to make it easier to copy and paste into Excel. This can be useful if the user would like to fill in gaps in the data such as test scores or would like to include other features they may care about such as athletics.



Recommended Schools

Show 30 entries

Search:

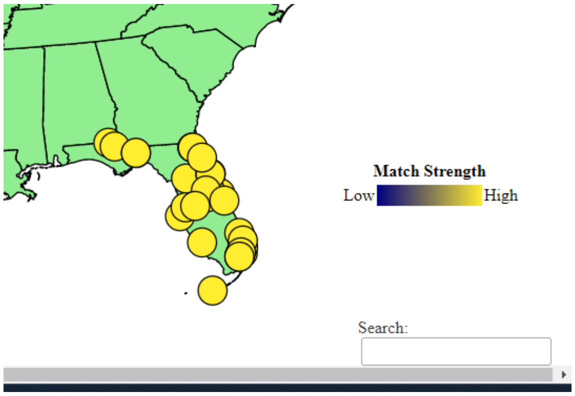
RANKING	NAME	CITY	STATE	AVG COST BASED ON HI	OVERALL AVG COST	SAT MATH 25TH PCTL	SAT MATH 75th PCTL	SAT CR 25TH PCTL	SAT CR 75TH PCTL	A
1	Jacksonville State University	Jacksonville	AL	19115.0	17523.0	420.0	540.0	420.0	530.0	
2	Auburn University	Auburn	AL	23468.0	24018.0	570.0	670.0	590.0	650.0	
3	Talladega College	Talladega	AL	17205.0	19863.0					
4	Tuskegee University	Tuskegee	AL	37338.0	36528.0	410.0	520.0	450.0	525.0	
5	Auburn University at Montgomery	Montgomery	AL	15400.0	13678.0	430.0	515.0	480.0	575.0	
6	Amridge University	Montgomery	AL	17618.0	17618.0					
7	Samford University	Birmingham	AL	31029.0	30072.0	520.0	600.0	540.0	640.0	
8	University of Alabama at Birmingham	Birmingham	AL	17857.0	16530.0	530.0	660.0	560.0	668.0	
9	Faulkner University	Montgomery	AL	18988.0	20500.0	505.0	585.0	515.0	590.0	
10	Strayer University-Alabama	Birmingham	AL	23379.0	23379.0					
11	South University-Montgomery	Montgomery	AL	26407.0	20518.0					
12	Herzing University-Birmingham	Birmingham	AL	26347.0	25741.0					
13	Birmingham-Southern College	Birmingham	AL	19226.0	19808.0	500.0	580.0	520.0	610.0	
14	Alabama State University	Montgomery	AL	19853.0	19534.0	406.0	518.0	438.0	531.0	
15	Huntingdon College	Montgomery	AL	22163.0	21632.0	500.0	568.0	503.0	590.0	
16	Alabama A & M University	Normal	AL	17694.0	15529.0	410.0	500.0	430.0	520.0	
17	Huntsville Bible College	Huntsville	AL							

Showing 1 to 30 of 30 entries

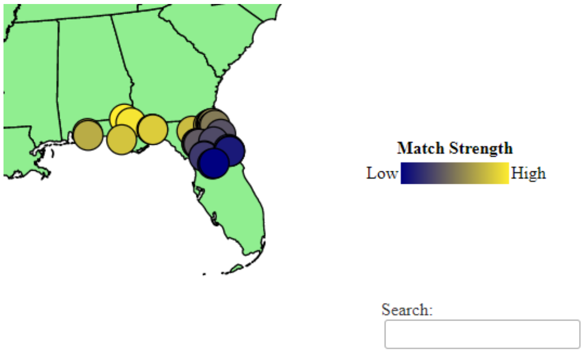
Example

To get a clear understanding of the advantages of our tool, let's use a simple example to demonstrate.

1. Let's say I want to go to school in Florida. I start by selecting FL, enter my home zip code (the only mandatory field) of "30324", and click "Update". The map populates with the results below. The top 30 schools in the database are returned with no ranking.

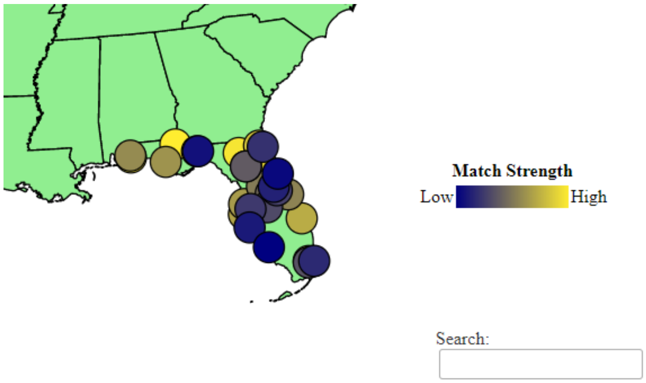


2. Let's say I want to give higher rank to schools that are closer. For the "Max Distance", input 5 miles and set the Distance Preference to "1 Equally", then click "Update". Visually, the schools closer to Georgia have a higher rank now.

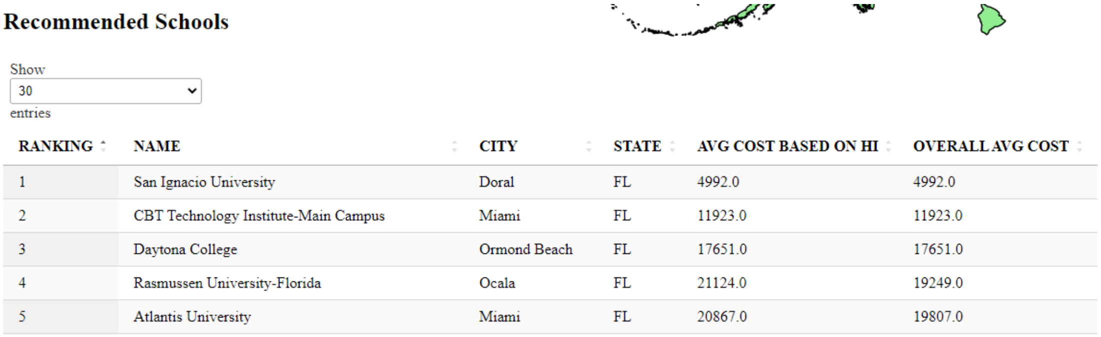


3. Apart from proximity, I am most concerned about cost. I don't want to spend more than $10k per year. Leaving Household Income unchanged, input 10,000 as the "Max Avg Annual Cost" and

set the preference to “9 Extremely” before clicking “Update”. Now, schools in the southern and central parts of Florida are populated due to cost being a higher preference than distance.



4. Reviewing the table, notice most of the schools are public. In general, the majority student population of public schools are in-state, which creates bias in cost for those paying out-of-state tuition. Therefore, set the school type to “Private for-profit” (Florida has no “Private nonprofit” schools) and preference to “7 Very Strongly”. Now, the tables ranks private schools higher than public with the top 2 universities being in South Florida.



Recommended Schools

RANKING	NAME	CITY	STATE	AVG COST BASED ON HI	OVERALL AVG COST
1	San Ignacio University	Doral	FL	4992.0	4992.0
2	CBT Technology Institute-Main Campus	Miami	FL	11923.0	11923.0
3	Daytona College	Ormond Beach	FL	17651.0	17651.0
4	Rasmussen University-Florida	Ocala	FL	21124.0	19249.0
5	Atlantis University	Miami	FL	20867.0	19807.0

Backend

The College Selection Made Easy web application is built using Flask (see **app.py**) to facilitate key features such as routing, templating, and request handling.

1. The route for the **index()** function renders the **index.html** template which generates the login webpage and requests the **login()** function
2. The login() function checks the email input from the user and renders the **landing.html** template with default values for parameters if the entry is valid, otherwise it redirects to the index.html for the user to reattempt login with a valid email.
3. The landing.html routes to the **update()** function when the update button is clicked. This function updates the default parameter values with those specified by the user and uses this data with functions in the **DatabaseHandler** and **rankHandler** classes to generate the **ranked_results.csv** before re-rendering the landing.html
4. The map and table in the landing.html load the ranked_results.csv yielding results based on profile selections.

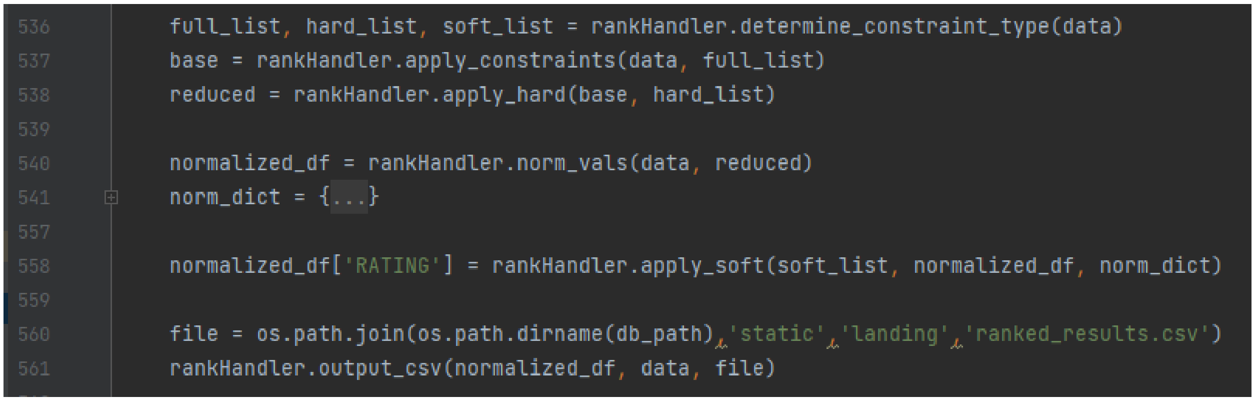
DatabaseHandler

The DatabaseHandler uses the get_table() function to return either a list or DataFrame based on the name parameter. Lists are used to populate dropdowns and user selections in the landing.html. For example, DatabaseHandler.get_table('sizes') will return the list: ['', 'Small', 'Medium', 'Large']. DataFrames are generated from queries to the sqlite cs.db. This database contains the following tables:

- **inst** – main table that contains various features of an institution.
- **fields** – provides the 3-year median earnings, majors, and degrees offered by an institution.
- **relig** – associates RELAFFIL (numeric value) in inst with name of religious affiliation.
- **geo** – used to lookup the latitude and longitude of a zip code.
- **user** – this table is populated by the login() function and is intended to be used with future updates involving the handling of concurrent users.

rankHandler

The rankHandler class uses the various functions that follow to generate the ranked_results.csv. Below is a snippet from app.py that shows how they are used together.



```
    full_list, hard_list, soft_list = rankHandler.determine_constraint_type(data)
    base = rankHandler.apply_constraints(data, full_list)
    reduced = rankHandler.apply_hard(base, hard_list)

    normalized_df = rankHandler.norm_vals(data, reduced)
    norm_dict = {...}

    normalized_df['RATING'] = rankHandler.apply_soft(soft_list, normalized_df, norm_dict)

    file = os.path.join(os.path.dirname(db_path),'static','landing','ranked_results.csv')
    rankHandler.output_csv(normalized_df, data, file)
```

determine_constraint_type(data_dict)

The input for this function is a dictionary with the constraint as a key and a nested dictionary as the value. The nested dictionary has the following key: value pairs:

- pref: the preference level set by the user from the default parameter values.
- val: likewise the value specified by the user also from the default parameter values.
- multi: an indictor if the constraint allows multiple values are not, Y or N.

The purpose of this function is to determine which list a constraint belongs to given its preference and value. Constraints with preferences of "10 Definitely" are in the hard list since they are a hard requirement whereas all other preference levels are soft constraints. Degree and states start off in the hard list since these features do not allow preference levels. All other constraints begin in the soft list. If no value is specified by the user, or contains the default value, the constraint is ignored and does not appear in either the soft or hard list.

Returns lists:

- full_list: static list containing all constraints.
- hard_list: dynamic list based on user inputs containing constraints with a hard requirements.

- soft_list: dynamic list based on user containing constraints with soft requirements.

calc_dist(input_zip)

A helper function used in apply_constraints() that takes the user’s home zip as a parameter. The lat and lon of the zip is first retrieved from the geo table through the DatabaseHandler. Each institution and their coordinates are also retrieved. Using geopy, the geodesic distance in miles is calculated from the home zip to the zip code of every institution. Finally, a DataFrame with the geodesic distance in miles to every institution is returned.

apply_constraints(data_dict, full_list)

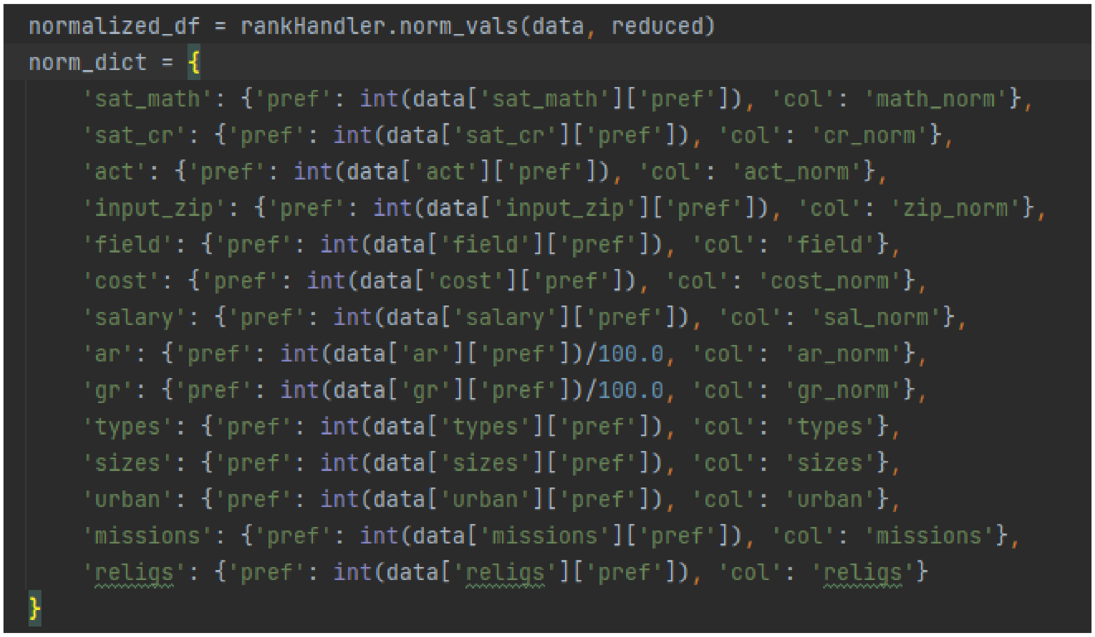
The most logic intensive function, its purpose is to create a standardized table to be used with the apply_hard() function. Essentially, it iterates through the full list of constraints and checks for every institution in the inst table whether it matches the constraint value specified from the user in the data dictionary. At this step we disregard whether the constraint is hard or soft. The result of each constraint (1 if a match, 0 if no match) is appended as a column to the DataFrame that is returned.

apply_hard(base_table, hard_list)

This function takes the DataFrame returned from apply_constraints() along with the hard list and filters the DataFrame to only include records that satisfied the hard constraints. It also does some transformations on the salary and cost columns.

norm_vals(data_dict, df)

This function is responsible for creating normalized numeric values (between -1 and 1) for the records in the reduced DataFrame from apply_hard(). Categorical constraints like field use their respective match columns since these are already binary. The norm_dict below specifies which column to use for a given constraint. Those ending in “_norm” are generated by norm_vals() and the remaining were previously generated by apply_constraints().



```
normalized_df = rankHandler.norm_vals(data, reduced)
norm_dict = {
    'sat_math': {'pref': int(data['sat_math']['pref']), 'col': 'math_norm'},
    'sat_cr': {'pref': int(data['sat_cr']['pref']), 'col': 'cr_norm'},
    'act': {'pref': int(data['act']['pref']), 'col': 'act_norm'},
    'input_zip': {'pref': int(data['input_zip']['pref']), 'col': 'zip_norm'},
    'field': {'pref': int(data['field']['pref']), 'col': 'field'},
    'cost': {'pref': int(data['cost']['pref']), 'col': 'cost_norm'},
    'salary': {'pref': int(data['salary']['pref']), 'col': 'sal_norm'},
    'ar': {'pref': int(data['ar']['pref'])/100.0, 'col': 'ar_norm'},
    'gr': {'pref': int(data['gr']['pref'])/100.0, 'col': 'gr_norm'},
    'types': {'pref': int(data['types']['pref']), 'col': 'types'},
    'sizes': {'pref': int(data['sizes']['pref']), 'col': 'sizes'},
    'urban': {'pref': int(data['urban']['pref']), 'col': 'urban'},
    'missions': {'pref': int(data['missions']['pref']), 'col': 'missions'},
    'religs': {'pref': int(data['religs']['pref']), 'col': 'religs'}
}
```

The following table summarizes our method for normalizing certain constraints:

Column	**Method**	**Range**
math_norm	If the institution 25th Percentile SAT Math score > input SAT Math: (sat_math - satmt25)/ satmt25 Else: (sat_math - satmt25)/ sat_math	(-1,1)
cr_norm	If the institution 25th Percentile SAT CR score > input SAT CR: (sat_cr - satcr25)/ satcr25 Else: (sat_cr - satcr25)/ sat_cr	(-1,1)
act_norm	If the institution 25th Percentile ACT score > input ACT: (act - act25)/ act25 Else: (act - act25)/ act	(-1,1)
zip_norm	(max distance [in reduced] - distance to institution)/ max distance	(0,1]
cost_norm	(max cost [in reduced] - institution cost)/ max cost	(0,1]
sal_norm	institution salary / max salary [in reduced]	(0,1]
ar_norm	institution acceptance rate / max acceptance rate [in reduced]	(0, 1]
gr_norm	institution graduation rate / max graduation rate [in reduced]	(0, 1]

apply_soft(soft_list, norm_df, norm_dict)

This function takes the DataFrame with the normalized values created in norm_vals() and uses the soft list and norm dictionary to determine the respective constraints and columns to use to calculate a rating. The calculation involves first determining the weight of each constraint. For example, if I have 2 soft constraints “sizes” at 6 and “cost” at 9, the weight of “sizes” is 6/(6+9) or 40% and weight of cost is 60%. The sum product of these weights and the normalized values create a rating for each institution, whose values are returned and mapped to a “Rating” column.

output_csv(df, data_dict, filename)

The final function queries the normalized DataFrame limiting the records to the top N matches specified by the user (default is N=30). The results are ordered in descending order by the Rating column created in apply_soft(). Therefore, if no Rating exists due to no soft constraints, then the ranking will be the same for all records. The query results are written to the “ranked_results.csv”.